Imperial College of Science, Technology and Medicine
Department of Computing

On the Feasibility of Using Fully-Convolutional Variational Autoencoders to Advance Deep Symbolic Reinforcement Learning

D. G. Sherburn

Submitted in part fulfilment of the requirements for the degree of
Master of Engineering in Joint Mathematics and Computing
of Imperial College, June 2017

Abstract

Reinforcement learning has achieved considerable success in recent years. One significant breakthrough was Google DeepMind's deep Q-network (DQN), which mastered a wide range of Atari 2600 games to a super-human level using only the pixels and score. However, reinforcement learning agents lack essential properties necessary for artificial general intelligence (AGI); namely they are slow to learn, unable to transfer knowledge between similar tasks and are unable to reason abstractly. In this thesis we seek to advance a method likely to overcome these drawbacks, known as deep symbolic reinforcement learning (DSRL). By applying recent advancements in the unsupervised learning of generative factors to fully-convolutional variational autoencoders, we develop a first iteration solution for a scalable DSRL system.

Acknowledgements

I would like to acknowledge:

- Pedro Mediano for his invaluable advice and supervision of this thesis.
- Murray Shanahan for his supervision of the first half of this thesis.
- Nat Dilokthanakul and Christos Kaplanis for their fruitful discussions.
- Marta Garnelo for discovering such an exciting research topic.

Dedication

This thesis is dedicated to mum and dad, for their immense sacrifices and never-ending support.

"Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law."

Douglas R. Hofstadter

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

A long term goal of artificial intelligence (AI) is the development of artificial general intelligence (AGI). Since the field's inception in the 1950s, it has swung between hype and breakthroughs, followed by disappointment and reduced funding, known as AI winters [17]. During the first period of hype from the 50s to the early 70s, Marvin Minsky made the following prediction: [9]

> "In from three to eight years we will have a machine with the general intelligence of an average human being." - Marvin Minsky, 1970

This prediction was clearly not realised, and the first AI winter would shortly follow.

Symbolic AI was developed during this winter, which encodes knowledge as human-readable rules and facts, making it easy to comprehend chains of actions and abstract relationships [23]. For instance, given the unary relations `red` and `strawberry`, and the binary relation `bigger`, we can say that `A` is the smallest red strawberry by writing

```
red(A)  strawberry(A)  ∀B bigger(B, A)
```

But given the unary relations `yellow` and `banana` we could also write that `A` is the third biggest yellow strawberry, or a red banana, and so on. We can see that the rules and facts in symbolic logic can be endlessly recombined and extended. This allows for the manipulation of high-level abstract concepts, which is key to AGI [13].

However, symbolic AI has a major philosophical problem: the facts and rules are only meaningful to the human writing them; their meaning is not intrinsic to the system itself. This is known as the *symbol grounding problem*.

Today we find ourselves in yet another period of hype and exciting breakthroughs not afflicted by the symbol grounding problem. Reinforcement learning (RL) has become a prominent area of research, with many considering it fundemental for AGI [15], as have deep neural networks. Recently, deep reinforcement learning (DRL) systems have achieved impressive feats, including mastering a wide range of Atari 2600 games to a superhuman level using only raw pixels and score as input, and the board game Go [21, 28].

Figure 1.1: May 1997: Gary Kasparov makes his first move against IBM's Deep Blue. Deep Blue would later emerge the victor in the best of six games; the first time a reigning world chess champion is defeated by a computer. [26].

Figure 1.2: March 2016: Lee Sedol, one of the greatest modern Go players, plays his first move of game three against AlphaGo. AlphaGo won four of five games. This feat was considered by many to be a decade away. [22].

Though DRL systems are not afflicted by the same problems as symbolic AI, they have a number of drawbacks of their own. Namely, they are: [13]

1. **Slow to learn**. Neural networks require large data sets and are therefore slow to learn.

2. **Unable to transfer past experience**. They often fail to perform well on tasks very similar to those they have mastered.

3. **Unable to reason abstractly**. They fail to exploit statistical regularities in the data.

4. **Hard to reason about**. It's often difficult to extract a comprehensible chain of reasons for why a deep neural network operated in the way it did.

Deep symbolic reinforcement learning (DSRL) is a marrying of DRL and symbolic AI; a recent advance which overcomes the symbol grounding problem and the drawbacks associated with DRL [13]. That is, DSRL systems overcome the symbol grounding problem, and are:

1. **Fast to learn**. Large data sets are not necessary.

2. **Able to transfer past experience**. Symbolic AI lends itself to multiple processes associated with high-level reasoning, including transfer learning.

3. **Able to reason abstractly**. The agent is able to exploit statistical regularities in the training data by using high-level processes like planning or causal reasoning.

4. **Easy to reason about**. Since the front end uses symbolic AI, its knowledge is encoded as human-readble facts and rules, making the extraction of comprehensible chains of logic much easier.

An overview of DSRL is shown in Figure 1.3. The neural back end takes a high-dimensional input and outputs a symbolic representation. This symbolic representation is then fed to the symbolic front end, whose role is action selection. The agent then acts on the environment and obtains a reward and the sensory input of the next time step. As the neural back end learns how to represent the raw input data in a compositionally structured representation in an unsupervised manner, and the symbolic front end learns to select the action with maximum expected reward over time, the system as a whole learns end-to-end.

The unsupervised extraction of features from a wide range of scenes is still a challenge in AI research. Since DSRL systems require this extraction in forming the symbolic representation,

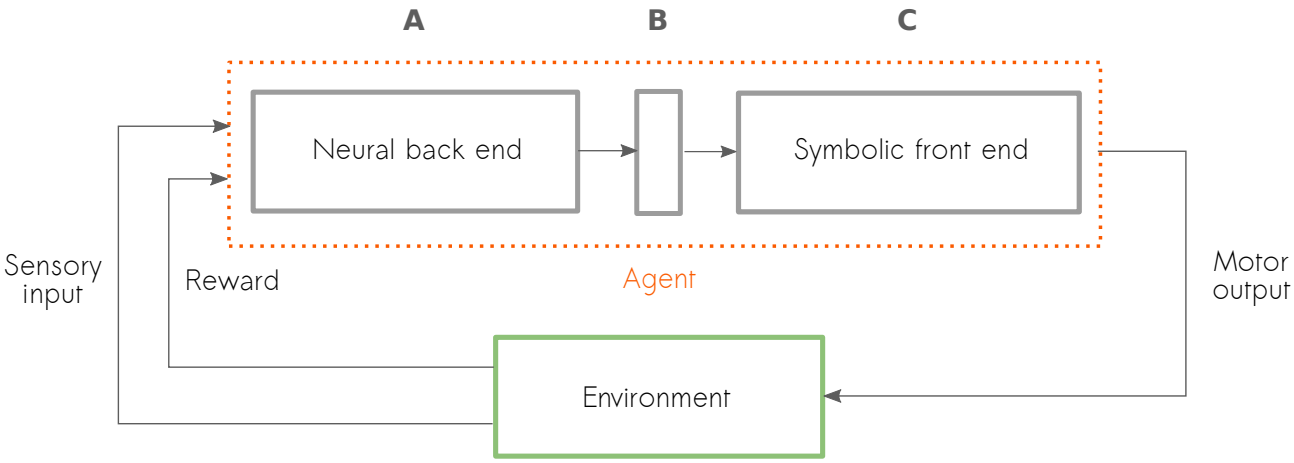


Figure 1.3: Overview of deep symbolic reinforcement learning system architecture. **A**: The neural back end maps high-dimensional raw input data to a compositionally structured symbolic representation. **B**: The compositionally structured symbolic representation. **C**: Reinforcement learning of mapping from symbolic representation to action with maximum expected reward over time. *Adapted from: Garnelo et al.* [13].

the current system only works for toy data sets. In order for the system to scale to complex scenes, a scalable method of extracting an object's type and position must be found.

1.2 Contributions

With a recent development, β-VAE, it is possible to learn independent generative factors in complex scenes using variational autoencoders [14]. In this thesis, we apply the technique proposed in β-VAE to the novel fully-convolutional variational autoencoder architecture, and in turn assess the feasibility of this architecture for advancing DSRL. We approach this in the following way:

- **Propose the novel architecture of the fully-convolutional variational autoencoder.** This is (as far as we know) the first consideration of a fully-convolutional variational autoencoder.

- **Propose a number of novel methods to solve the low-level extraction of objects and their location in fully-convolutional variational autoencoders.** This is done by inventing novel architectures and loss functions, which are inspired by the developments

made in β-VAE. These approaches are shown to be good starting points in solving this open problem.

- **Collect experimental evidence for each proposed method.** Each proposed method is assessed by a range of experiments. These include reconstructing its input, qualitatively examining the latent representation of high-level objects in the scene as well as generating new samples by sampling from the prior and using Markov chain Monte Carlo (MCMC).

- **Assess the feasibility of using fully-convolutional variational autoencoders to advance DSRL.** Upon assessment of the experiments, we may propose a suitable method that (at least) partially solves the problem of latent representation of high-level objects in the scene.

Chapter 2

Background

We will cover how deep symbolic reinforcement learning extracts symbolic representations from raw input data, which will motivate a discussion of loss functions and the introduction of autoencoders. Seeing the limitations of the current approach in extracting symbolic representations, we can appreciate the recent development of β-VAE, a variant of the variational autoencoder used to learn disentangled representations. Finally, we can conclude by mentioning less technical matters, such as libraries and hardware used.

2.1 Loss functions

The idea of image reconstruction plays a vital role throughout this project. Although it's possible to qualitatively compare the original to its reconstruction, it's important to be able to quantify the difference, which lends itself to automation. The loss function will quantify how similar two images are.

To compare loss functions, we'll use the MNIST data set. MNIST is a collection of $70,000$ black-and-white images of handwritten digits, with $60,000$ in the training set containing and $10,000$ in the test set. These images will be represented as vectors without loss of generality.

2.1.1 Euclidean Distance

The Euclidean distance between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined by

$$\sqrt{\sum_i (x_i - y_i)^2}$$

where x_i and y_i are the i^{th} components of $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively.

Euclidean distance is an intuitive measure of the distance between two points in space. Unfortunately, this doesn't also translate to visual similarity, as illustrated by Doersch et al. [11]. Figure 2.1 is a digit drawn from the MNIST dataset, and Figures 2.3 and 2.2 are attempted reconstructions. Of the reconstructions, Figure 2.2 looks most like the original, but Figure 2.3 is closer in space.

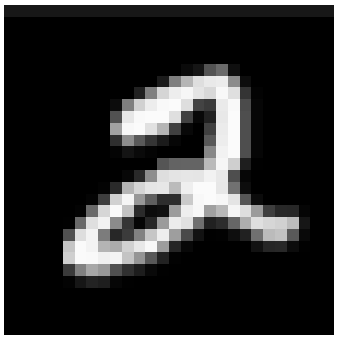


Figure 2.1

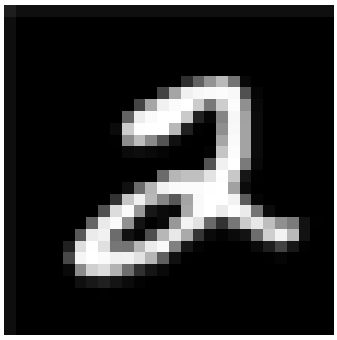


Figure 2.2

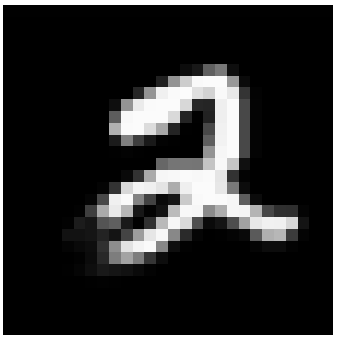


Figure 2.3

This leads to an alternative measure, binary cross-entropy, which gives a much better quantification of how visually similar two images are.

2.1.2 Binary Cross-Entropy

Consider a single black-and-white pixel with probability $p(0) = c$ of being 0 and $p(1) = 1 - c$ of being 1. Here $p(x)$ is a probability distribution over the possible pixel values $x \in \{0, 1\}$. Suppose a given model tries to learn the distribution described by $p(x)$, and says that the pixel has probability $q(0) = \hat{c}$ of being 0 and $q(1) = 1 - \hat{c}$ of being 1. The model is perfect if it learns

the true distribution, that is, if $q(x) = p(x)$ for $x \in \{0, 1\}$. We'd like to quantify how similar the distributions p and q are.

This is done by computing the binary cross-entropy between p and q, which is defined by

$$H(p, q) = -c \log \hat{c} - (1 - c) \log(1 - \hat{c})$$

To see how we may use this as a similarity measure among images, consider a 1×1 image. Normalising this image yields a pixel value in the interval $[0, 1]$, which may now be interpreted as a probability, corresponding to c above. In the normalised reconstructed image, the pixel value corresponds to $\hat{c}$. We simply compute the binary cross-entropy to measure the similarity of these two distributions, and in turn, the similarity of the images themselves! (Note: we could have also assigned the probabilities to $1 - y$ and $1 - \hat{y}$ by symmetry of binary cross-entropy).

For images larger than 1×1, we may take the component-wise binary cross-entropy, then, for example, average the components. How the component-wise binary cross-entropies are suitably combined to give a single floating point number will vary from problem to problem.

2.2 Stella

Video games are becoming increasingly popular in the generation of data sets for machine learning purposes. One reason for this is that they do not inherit many operational drawbacks from real world data, such as noise, the stability of the camera or the observation of rare circumstances (the video game may just be queried for rare cases) [34, 24].

2.3 Arcade Learning Environment

The Arcade Learning Environment (ALE) is a framework built on top of the Atari 2600 emulator Stella [1]. This library was a suitable choice because:

- Emulation details are abstracted away from the researcher
- The same agent can be used for all games, due to its modular design
- Frames can be saved during game play, which is used to collect training data
- A Python wrapper for the entire framework is provided, so our agents developed in Keras interact seamlessly

2.4 Keras

Keras is a high-level neural network library written in Python [6]. It was a suitable choice because it supports:

- Convolutional, deconvolutional and pooling layers
- Lambda functions, which is necessary for sampling in variational autoencoders
- The creation of custom loss functions, which is necessary when developing new methods
- Nividia GPUs

Chapter 3

Related Work

3.1 Autoencoders

An autoencoder is a neural network that learns a compression algorithm for its input data in an unsupervised manner [20]. This is achieved by placing constraints on a hidden layer, called the latent space, and setting the target values to the input values, effectively learning the identity function. Since the network is trying to reconstruct the original input from the constrained latent space, over time the latent space corresponds to a meaningful compression of the network's input.

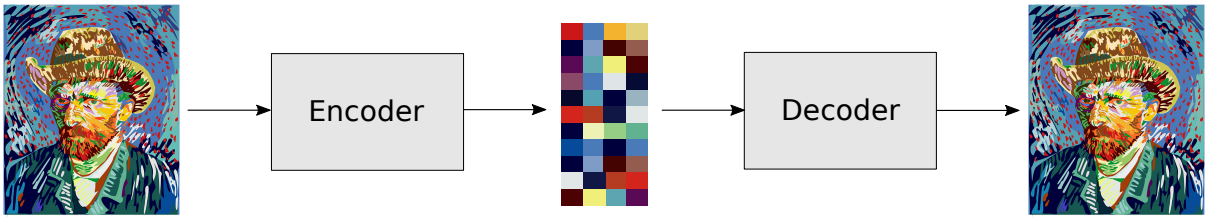


Figure 3.1: A black-box description of an autoencoder. The autoencoder learns the identity function, and in turn, the encoder and decoder learn suitable encoding and decoding algorithms respectively.

As before, we will use the MNIST data set to compare architectures. Unless specified in the example, the Adam optimiser is used with a learning rate of $1e-4$, the batch size is 1 and the loss function is binary cross-entropy. Intermediate layers use the ReLU activation function, while the final layer uses sigmoid.

3.1.1 Fully-Connected Autoencoders

In dense feed-forward neural networks we may place a constraint on the latent space by reducing the number of neurons, as shown in Figure 3.2. Images must be flattened into vectors to be fed as input. Consequently, any spatial information is destroyed in dense feed-forward neural networks.

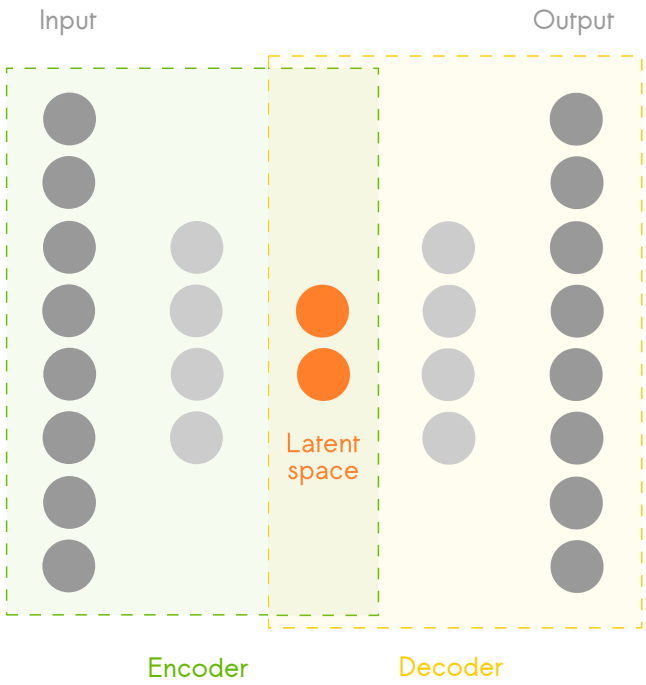


Figure 3.2: An example architecture of a fully-connected autoencoder. The latent space is constrained by having fewer neurons than the input and output layers.

An example architecture is given in Table 3.1, which was trained on MNIST. Despite the latent space being $\sim 4\%$ of the size of the input space, the network is capable of producing realistic reconstructions. For verification, a collection of samples from the dataset and their corresponding reconstructions are shown in Figure 3.3.

Layer Type	**Output Shape**
InputLayer	(1, 28, 28)
Flatten	(784,)
Dense	(32,)
Dense	(784,)
Reshape	(1, 28, 28)

Table 3.1: A simple fully-connected autoencoder with one hidden layer. After 15 epochs, the validation score was recorded to be 71.94.

Figure 3.3: A collection of images from the MNIST data set and their respective reconstructions using the fully-connected autoencoder specified in Table 3.1. The original MNIST images are in odd columns, and their reconstructions to their immediate right.

3.1.2 Fully-Convolutional Autoencoders

In fully-convolutional feed-forward neural networks, we may place a constraint on the latent space by reducing the number and/or size of the filters, as shown in Figure 3.4. To compare the fully-convolutional autoencoder to the fully-connected, we'll train the architecture in Table 3.2 on MNIST. As before, we'll compare the reconstructions to the originals, which can be found in Figure 3.5.

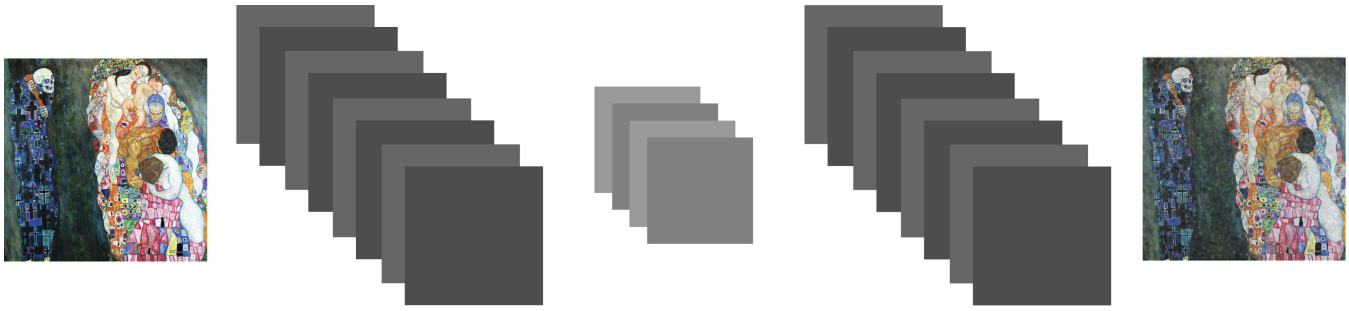


Figure 3.4: An example architecture of a fully-convolutional autoencoder. The latent space is constrained by reducing the number and/or size of the filters.

Layer Type	**Output Shape**
InputLayer	(1, 28, 28)
Conv2D	(32, 28, 28)
MaxPooling2D	(32, 14, 14)
Conv2D	(4, 14, 14)
MaxPooling2D	(4, 7, 7)
UpSampling2D	(4, 14, 14)
Conv2DTranspose	(32, 14, 14)
UpSampling2D	(32, 28, 28)
Conv2DTranspose	(1, 28, 28)

Table 3.2: A simple fully-convolutional autoencoder with 2D convolutions and max pooling, plus the corresponding deconvolutional layers. After 15 epochs, the validation score was recorded to be 64.89.

Convolutional layers have been shown to be effective in tasks with images as input [18, 33, 30]. This is because spatial information is preserved in convolutional layers, and the number of trainable parameters is far less in a convolutional layer than it is in a fully connected layer. Convolutional layers will be used from here on as we'll be using images as input.

Figure 3.5: A collection of images from the MNIST data set and their respective reconstructions using the fully-convolutional autoencoder specified in Table 3.2. The original MNIST images are in odd columns, and their reconstructions to their immediate right.

3.2 Variational Autoencoders

[16]

The variational autoencoder is central to this project, and we'll therefore dedicate a considerable amount of time exploring it. First we'll precisely define the problems the variational autoencoder solves, after which we may develop its loss function and detail its implementation. This section will conclude by developing an intution of the theory covered by way of examples.

3.2.1 A Probabilistic Perspective

We'll begin by making necessary definitions and describe the input data as samples from a generative process.. This sets the context to detail the problems the variational autoencoder solves.

Let $X = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$ be the data set of N independent and identically distributed samples of the variable $\boldsymbol{x}$. (X may be a data set of images, for instance). Let us assume that these samples are generated by a random process with parameters $\boldsymbol{\theta}^*$ involving an unobserved latent variable $\boldsymbol{z}$ in the following way:

Algorithm 1 Generate data set X

1: **for** $i = 1 \rightarrow N$ **do**
2: $\quad \boldsymbol{z}^{(i)} \sim p_{\boldsymbol{\theta}^*}(\boldsymbol{z})$ // Sample from true prior
3: $\quad \boldsymbol{x}^{(i)} \sim p_{\boldsymbol{\theta}^*}(\boldsymbol{x}|\boldsymbol{z}^{(i)})$ // Sample from true conditional
4: $\quad$ Append $\boldsymbol{x}^{(i)}$ to X
5: **end for**

We only observe the data set X in this process. The parameters $\boldsymbol{\theta}^*$ and latent variables $Z = \{\boldsymbol{z}^{(i)}\}_{i=1}^{N}$ are unknown to us. Let us assume that the prior $p_{\boldsymbol{\theta}^*}(\boldsymbol{z})$ and conditional $p_{\boldsymbol{\theta}^*}(\boldsymbol{x}|\boldsymbol{z})$ are parameterised by the distributions $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ respectively. In this context, the variational autoencoder provides [16]:

1. ML or MAP estimation for the parameters $\boldsymbol{\theta}$

2. An approximation of the latent variable $\boldsymbol{z}^{(i)}$ given $\boldsymbol{x}^{(i)}$ and set of parameters $\boldsymbol{\theta}$

3. Approximate marginal inference of the variable $\boldsymbol{x}$

3.2.2 Overcoming the Intractable Posterior

The variational autoencoder solves the problems above by approximate inference of the latent variable $\boldsymbol{z}$. Exact inference is not possible, and to see this we may use Bayes' theorem to find an expression for the posterior:

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})} \tag{3.1}$$

The marginal likelihood

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})d\boldsymbol{z} \tag{3.2}$$

involves an exponential-time integral over every combination of the latent variable $\boldsymbol{z}$, and is therefore computationally intractable [16]. Instead, we define an approximation $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ to the intractable posterior. Since $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ gives a distribution over the possible latent variables $\boldsymbol{z}$ that generated the given data point $\boldsymbol{x}$, it is known as the probabilistic decoder. By the same reasoning, $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ is known as the probabilistic decoder.

3.2.3 Finding a Suitable Loss Function: the ELBO

The variational autoencoers's ability to learn the generative parameters $\boldsymbol{\theta}^*$ relies on how closley $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ approximates $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. In the interest of training a model, this difference will be quantified. For this we use the KL divergence

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right] \tag{3.3}$$

which measures how much information is lost when we represent $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ with $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ (measured in nats) [4]. Using the KL divergence, our problem now amounts to the optimisation problem [19]:

$$\boldsymbol{\theta}^*, \boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\theta}^*, \boldsymbol{\phi}^*} D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) \tag{3.4}$$

To see how we can start to minimise the KL divergence, we'll start by rewriting it in a different form:

$$\begin{aligned}
D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) &= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right] && (3.5)\\
&= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log_e q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\big] - \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log_e p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\big] && (3.6)\\
&= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log_e q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\big] - \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\right] && (3.7)\\
&= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}\right] + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log_e p_{\boldsymbol{\theta}}(\boldsymbol{x})] && (3.8)\\
&= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}\right] + \log_e p_{\boldsymbol{\theta}}(\boldsymbol{x}) && (3.9)
\end{aligned}$$

Here we see that the KL divergence depends on the intractable marginal likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x})$! There's no way we can minimise it if we can't write down $p_{\boldsymbol{\theta}}(\boldsymbol{x})$. However, we can get around this: we'll minimise the KL divergence, but not directly. Instead, we try to find a quantity which we can maximise, and show that in turn this minimises the KL divergence. The trick is not obvious, but is simply done by finding a lower bound on the log marginal likelihood.

Using Jensen's inequality

$$f(\mathbf{E}\big[X\big]) \geq \mathbf{E}\big[f(X)\big] \tag{3.10}$$

we can write down a lower bound on the log marginal likelihood:

$$\log_e p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \log_e \int p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} \tag{3.11}$$

$$= \log_e \int p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \tag{3.12}$$

$$= \log_e \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \right] \tag{3.13}$$

$$\geq \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log_e \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \right] \tag{3.14}$$

$$:= \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) \tag{3.15}$$

Expression (3.14) is called the *ELBO* (short for expected lower bound) [2, 16].

How does the *ELBO* help us with minimising the KL divergence? First recall the alternative form of the KL divergence:

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})} \right] + \log_e p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{3.9}$$

Writing this in terms of the *ELBO* we have:

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) = -\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) + \log_e p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{3.16}$$

Since the KL divergence is the negative of the *ELBO* up to an additive constant (with respect to q), minimising the KL divergence is equivalent to maximising the *ELBO* [3]. Now we may make a revision to our original optimisaiton problem (3.4). Our problem is written as [19]:

$$\boldsymbol{\theta}^*, \boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\theta}^*, \boldsymbol{\phi}^*} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) \tag{3.17}$$

3.2.4 Writing the ELBO in Closed-Form

We've found that we can maximise the *ELBO* to minimise the KL divergence between the approximation $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and the intractable posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. Now we will seek to write the *ELBO* in closed-form, after which we will be able to implement the variational autoencoder.

To write the $ELBO$ in closed-form, we'll start with a useful manipulation:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] \tag{3.18}$$

$$= -\mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}\right] \tag{3.19}$$

$$= -\mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}\right] \tag{3.20}$$

$$= -\mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z})}\right] + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log_e p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] \tag{3.21}$$

$$= -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] \tag{3.22}$$

Thus for a single data point $\boldsymbol{x}^{(i)}$, the $ELBO$ becomes:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\big] \tag{3.23}$$

With some assumptions and a bit of work, we're finally in a position to write the $ELBO$ in closed-form. The first term is the KL divergence between the probabilistic encoder and the prior. To make our lives simpler, we can choose the prior to be the isotropic multivariate Gaussian:

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{3.24}$$

We will also assume that the posterior is a k-dimensional Gaussian with diagonal covariance. It follows that the approximate posterior should take the same form. That is,

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) = \mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I}) \tag{3.25}$$

where $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\sigma}^{(i)}$ are the mean and standard deviation (respectively) for data point $\boldsymbol{x}^{(i)}$. With these two assumptions, and the KL divergence for k-dimensional Gaussians,

$$D_{KL}(\mathcal{N}_0||\mathcal{N}_1) = \frac{1}{2}\left[\operatorname{tr}(\boldsymbol{\Sigma}_{\mathbf{1}}^{-1}\boldsymbol{\Sigma}_{\mathbf{0}}) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}_{\mathbf{1}}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) - k + \ln\left(\frac{\det \boldsymbol{\Sigma}_{\mathbf{1}}}{\det \boldsymbol{\Sigma}_{\mathbf{0}}}\right)\right] \tag{3.26}$$

we may write the first term in closed-form:

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) = \frac{1}{2}\Big[\sum_{j=1}^{k}\sigma_j^{2(i)} + \sum_{j=1}^{k}\mu_j^{2(i)} - k - \ln\prod_{j=1}^{k}\sigma_j^{2(i)}\Big] \tag{3.27}$$

Now we turn our attention to the second term of the $ELBO$:

$$\mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\big] \tag{3.28}$$

Luckily, maximising terms like this is encountered regularly in statistics, and is known as maximum likelihood estimation. This can be taken to be the reconstruction loss (defined it earlier) [19].

Therefore, using an unspecified reconstruction loss, the $ELBO$ is:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \frac{1}{2}\Big[\sum_{j=1}^{k}\sigma_j^{2(i)} + \sum_{j=1}^{k}\mu_j^{2(i)} - k - \ln\prod_{j=1}^{k}\sigma_j^{2(i)}\Big] + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\big] \tag{3.29}$$

3.2.5 Implementing the Variational Autoencoder

The variational autoencoder uses a neural network for the probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ [16]. First a naïve implementation of the variational autoencoder will be proposed. As we shall see, this implementation needs one adjustment, known as the "reparameterisatoin trick". This will lead to the final implementation of the variational autoencoder, and conclude our formal study.

The probabilistic encoder

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) = \mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I}) \tag{3.25}$$

requires the mean $\boldsymbol{\mu}^{(i)}$ and standard deviation $\boldsymbol{\sigma}^{(i)}$ for a given data point $\boldsymbol{x}^{(i)}$. These two vectors will be represented by distinct layers in the latent space of a neural network, as shown in Figure (3.6) [19].

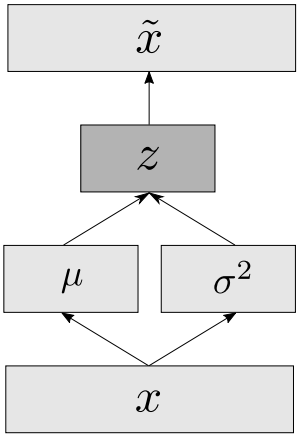


Figure 3.6: A naïve implementation of the variational autoencoder. The input $\boldsymbol{x}$ is mapped to intermediate layers taking the values of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$. The latent variable $\boldsymbol{z}$ is then sampled from the probabilistic encoder $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. Finally $\boldsymbol{z}$ is mapped back to the input dimension to give reconstruction $\tilde{\boldsymbol{x}}$. *Adapted from* [19].

Unfortuntely we cannot perform backpropagation, as it makes no sense to differentiate the the random variable $\boldsymbol{z}$ wrt $\boldsymbol{\phi}$ ($\boldsymbol{z}$ is drawn from a distribution, and not a function of $\boldsymbol{\phi}$). To solve this, we introduce an auxillary variable $\boldsymbol{\epsilon}$ and vector-valued function $g_\phi(\boldsymbol{x}, \boldsymbol{\epsilon})$ parameterised by $\boldsymbol{\phi}$ [16]. The auxillary variable $\boldsymbol{\epsilon}$ is governed by a parameterless distribution, which we will take to be the multivariate isotropic Gaussian.

The sampling step can now be written as

$$\boldsymbol{z} = g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \tag{3.30}$$

where $\odot$ represents the element-wise product. This is a reparameterisation of the random variable $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ with a differentiable function $g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon})$, and is suitable known as the "reparameterisation trick" [16]. The reparameterisation trick allows backpropagation to be used, and thus completes our method of solving optimisation problem (3.17).

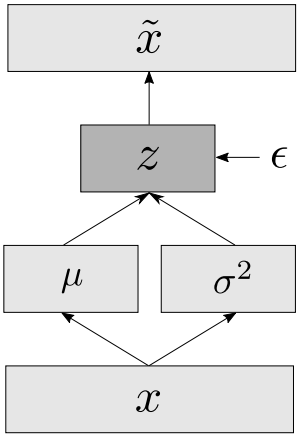


Figure 3.7: A viable implementation of the variational autoencoder. Sampling from the probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is simulated by evaluating $\boldsymbol{z} = g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$. *Adapted from* [19].

3.2.6 Intuition Behind the Variational Autoencoder

In this section, we'll develop the intuition behind the variational autoencoder. We've been considering the data set $X = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$. Now we'll choose X to be the MNIST data set (the vector $\boldsymbol{x^{(i)}}$ now corresponds to a flattened MNIST image of length $28 \times 28 = 784$). For visualisation purposes, we'll also take the latent space dimension to be $k = 2$, that is, $\boldsymbol{z} = (z_1, z_2)$.

The probabilistic encoder

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) = \mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I}) \tag{3.25}$$

gives a normal distribution over the values of the latent variable $\boldsymbol{z}^{(i)}$ given data point $\boldsymbol{x}^{(i)}$. A sample is drawn from this normal distribution and passed through to the decoder. The decoder then reconstructs the original image from the latent representation. Since we've chosen our latent space to be two-dimensional, it's possible to visualise this process, which is done in Figure (3.8). It's then possible to compare the reconstruction to the original (using an appropriate reconstruction loss function), and therefore to train the autoencoder as a whole.

Recall that we chose the prior to be the standard multivariate Gaussian

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{3.24}$$

We also derived the loss function to be

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\big] \tag{3.23}$$

which we want to maximise. Since

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) \geq 0 \tag{3.31}$$

maximising the $ELBO$ is equivalent to reducing the KL divergence between the probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ and the prior $p_{\boldsymbol{\theta}}(\boldsymbol{z})$. Therefore after training, we expect that the probabilistic encoder should map samples to the standard multivariate Gaussian. (In practice, $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}) > 0 \quad \forall i$, so we therefore only expect it to be *approximately* mapped to the standard Gaussian). Decoding samples from the prior should correspond to meaningful reconstructions. A visualisation of sampling from the prior is given in Figure (3.9).

We also know that

$$\mathcal{N}(\mu_1, \sigma_1^2) + \mathcal{N}(\mu_2, \sigma_2^2) = \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2) \tag{3.32}$$

and therefore, by taking the sum of the distributions of all data points $\boldsymbol{x}^{(i)}$, we have

$$\sum_{i=1}^{N} q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) = \mathcal{N}(\sum_{i=1}^{N} \boldsymbol{\mu}^{(i)}, \sum_{i=1}^{N} \boldsymbol{\sigma}^{2(i)}) \tag{3.33}$$

Surprisingly, this happens to approximate the prior $p(\boldsymbol{z})$! That is,

$$\sum_{i=1}^{N} q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) \approx p(\boldsymbol{z}) \tag{3.34}$$

A nice visualisation of this result is given in Figure (3.10).

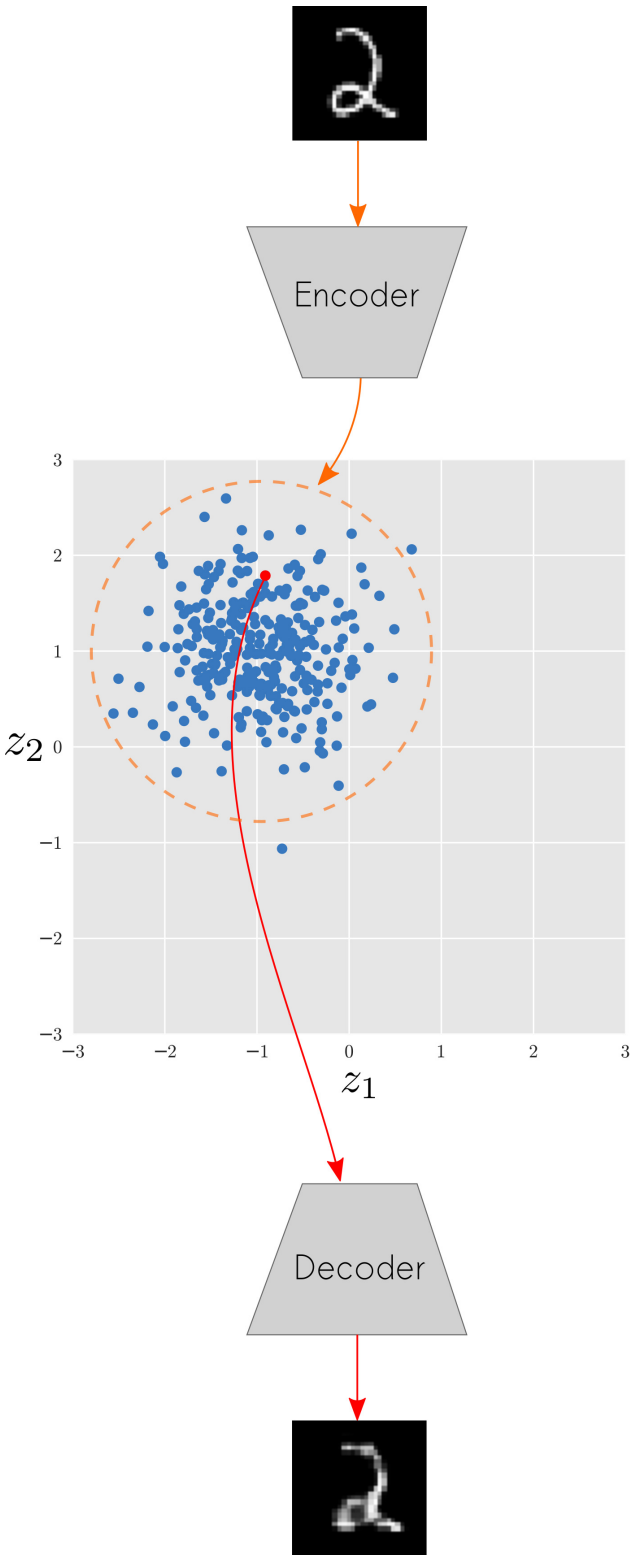


Figure 3.8: The encoder takes a data point and returns a normal distribution (orange); some samples of which are shown (blue). A sample is drawn from the normal distribution (red) and decoded. *Adapted from:* [12].

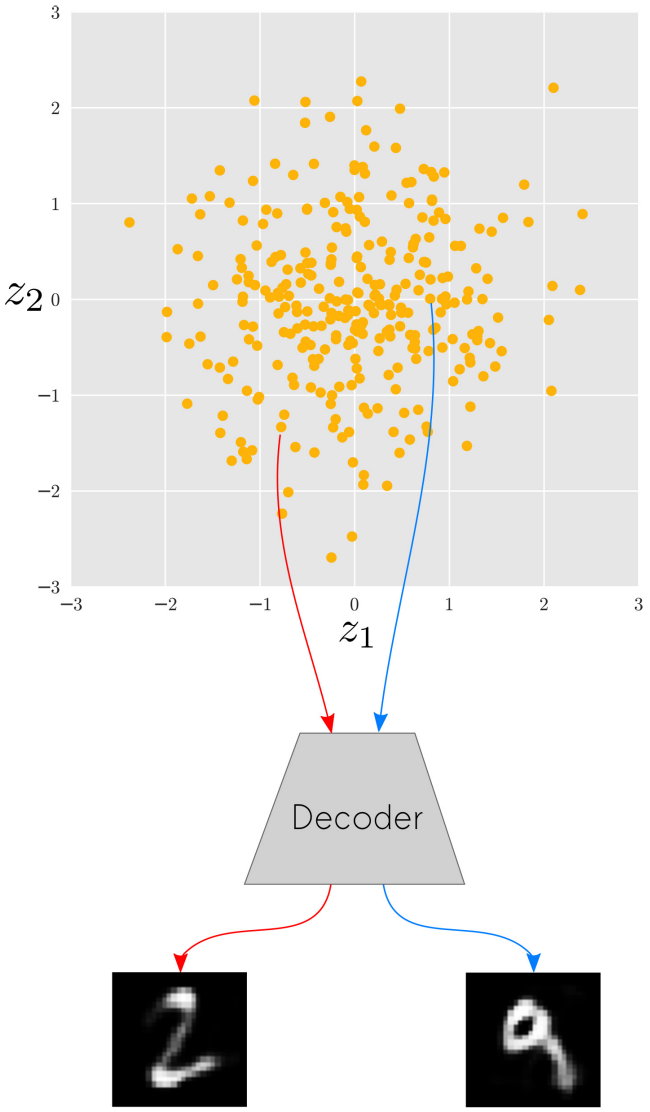


Figure 3.9: The prior distribution should approximate the standard multivariate Gaussian $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Samples of the prior are shown (yellow); two of which are decoded (red and blue). *Adapted from:* [12].

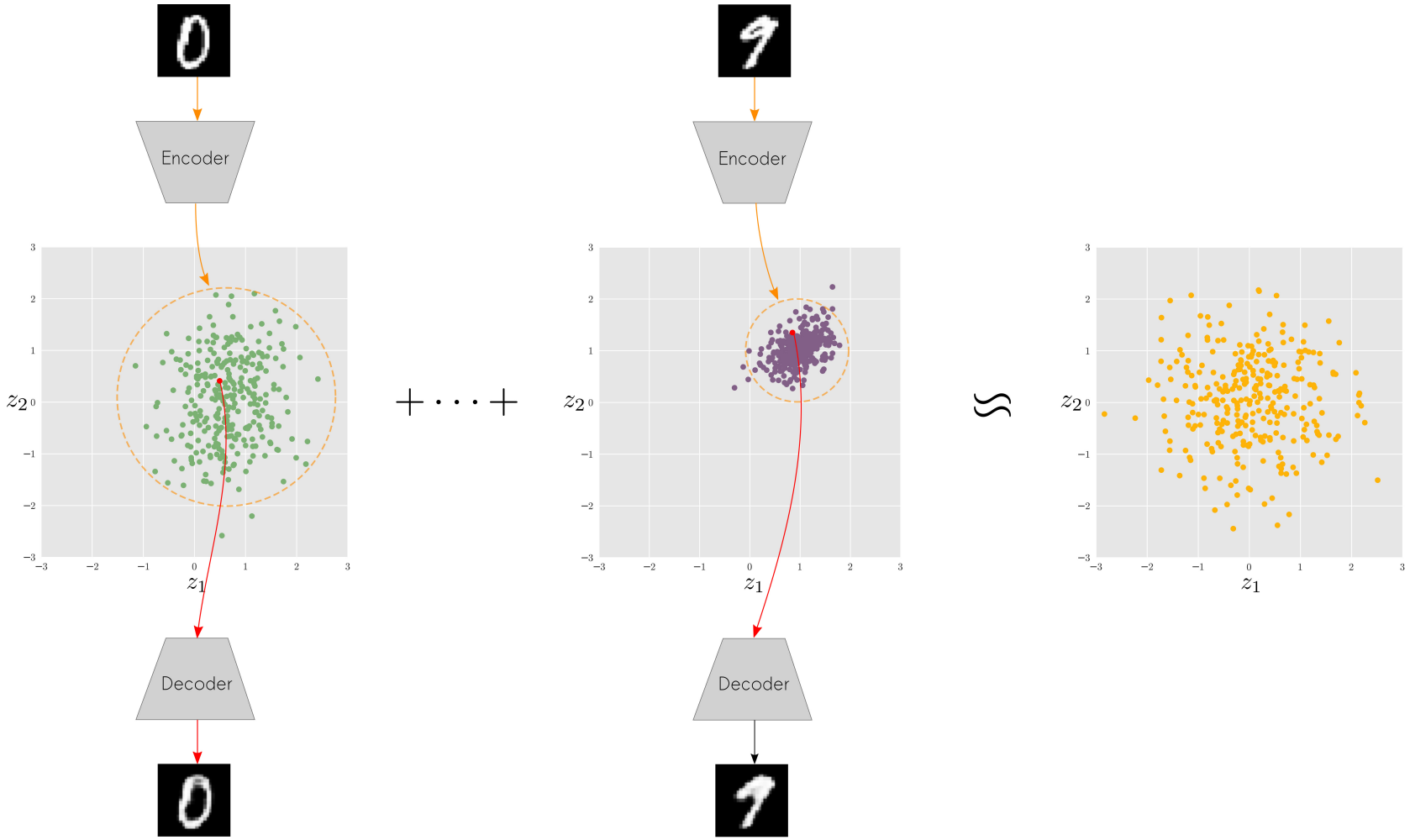


Figure 3.10: The sum over the latent space distributions of all data points $\boldsymbol{x}^{(i)}$ approximates the multivariate isotropic Gaussian. *Adapted from:* [12].

3.3 Unsupervised Learning of Generative Factors

Learning disentangled generative factors of a scene in an unsupervised manner is an open challenge in AI research. Although many attempts have been made, they have not scaled well [32, 27, 10, 31, 7]. However, two recent advancements have made significant headway: β-VAE and InfoGAN [5, 32].

3.3.1 InfoGAN

InfoGAN is an information theoretic extension of the Generative Adversarial Network, which has convincingly learnt generative factors of multiple data sets in an unsupervised manner, including MNIST, 3D faces and 3D chairs [5].

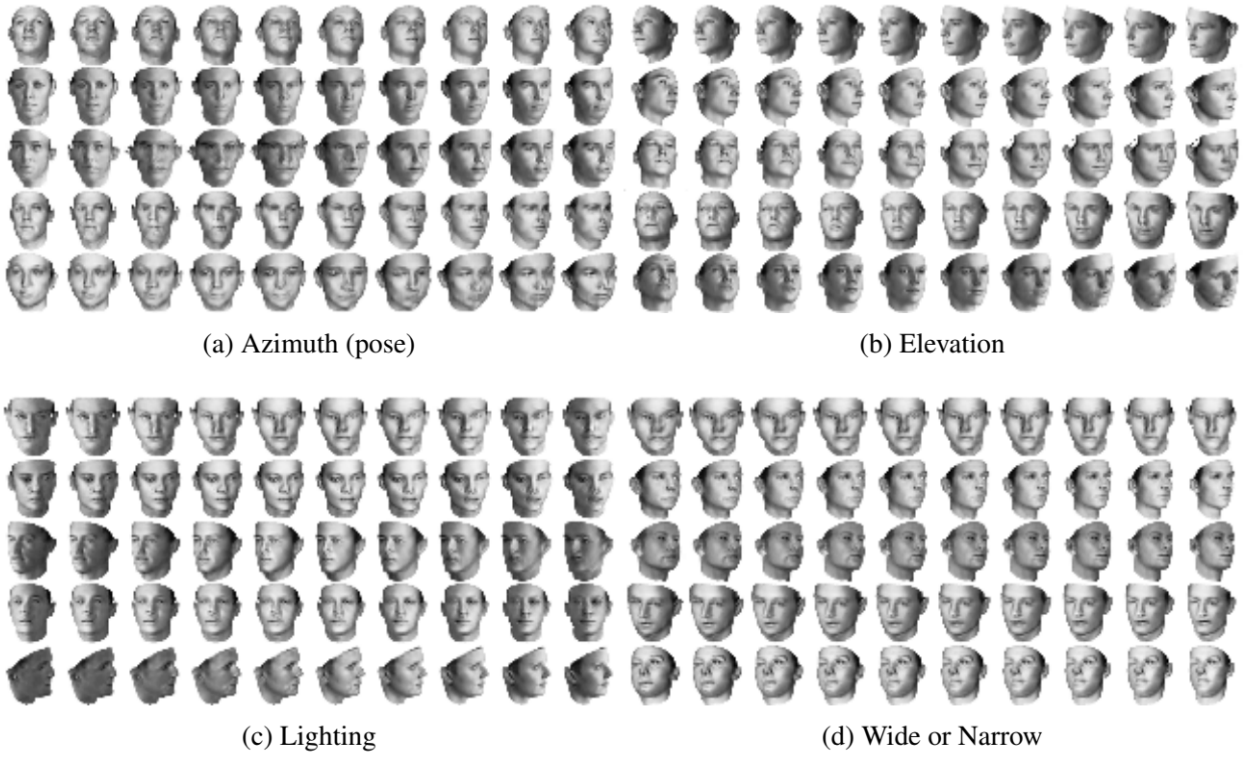


Figure 3.11: InfoGAN convincingly learns the underlying generative factors in the 3D face data set. Rows correspond to a data point and columns the value of the latent variable (varied from -1 to 1). Each section (a), (b), (c) and (d) consider a different latent variable. *Source:* [5].

However, InfoGAN is sensitive to the choice of the prior distribution, and therefore requires a priori knowledge of the data set, as well as the number and type of generative factors [32]. Ide-

ally, the number of generative factors would be inferred and not made explicit by the designer. On this basis, we can conclude that future work is needed.

3.3.2 β-VAE

β-VAE is the first method to overcome what InfoGAN could not: to learn the (unspecified amount of) generative factors of a data set in an unsupervised manner. First a formal derivation of β-VAE will be proposed, then a number of experimental results.

Derivation

Recall that for the variational autoencoder, we assumed the data point $\boldsymbol{x}$ was generated by a process involving a latent variable $\boldsymbol{z}$ and conditional $p(\boldsymbol{x}|\boldsymbol{z})$. For β-VAE we make a slightly different assumption: data point $\boldsymbol{x} \in \mathbb{R}^N$ was generated by conditionally independent factors $\boldsymbol{v} \in \mathbb{R}^K$ s.t. $p(\boldsymbol{v}|\boldsymbol{x}) = \prod_k p(v_k|\boldsymbol{x})$, conditionally dependent factors $\boldsymbol{w} \in \mathbb{R}^H$ and the conditional $p(\boldsymbol{x}|\boldsymbol{v}, \boldsymbol{w})$. In short, this is written as

$$p(\boldsymbol{x}|\boldsymbol{v}, \boldsymbol{w}) = Sim(\boldsymbol{v}, \boldsymbol{w}) \tag{3.35}$$

where Sim is short for a true world simulator. β-VAE seeks to represent these generative factors in the latent variable $\boldsymbol{z} \in \mathbb{R}^M$, such that

$$p(\boldsymbol{x}|\boldsymbol{z}) \approx p(\boldsymbol{x}|\boldsymbol{v}, \boldsymbol{w}) = Sim(\boldsymbol{v}, \boldsymbol{w}) \tag{3.36}$$

Note that, unlike InfoGAN, β-VAE does not specify the number independent generative factors K; instead it is inferred. However, in order for $\boldsymbol{z} \in \mathbb{R}^M$ to learn the conditionally independent factors $\boldsymbol{v} \in \mathbb{R}^K$, we must assert that $M \geq K$.

As before, the posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ is intractable, which motivates the introduction of the probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. A reasonable objective is to find the most likely parameters of the model over all latent variables that produced the observed data. This is summarised more

precisely as the optimisation problem

$$\boldsymbol{\theta}^*, \boldsymbol{\phi}^* = \max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] \tag{3.37}$$

A constraint is added to control the capacity of information in the latent space and encourage the factors $\boldsymbol{v}$ to be learnt in a disentangled manner. The probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is pressured to be close to an isotropic Gaussian, implicitly applying an independence pressure. By using the KL divergence, we form the optimisation problem

$$\begin{aligned} \boldsymbol{\theta}^*, \boldsymbol{\phi}^* &= \max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] \\ &\text{s.t. } D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})) < \epsilon \end{aligned} \tag{3.38}$$

The Lagrangian of optimisation problem (3.38) is

$$\mathcal{F}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] - \beta(D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})) - \epsilon) \tag{3.39}$$

Since $\beta, \epsilon > 0$,

$$\mathcal{F}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) \geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] - \beta D_{KL} q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})) \tag{3.40}$$

By applying pressure for $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ to be close to the prior $p(\boldsymbol{z})$, the latent space learns the conditionally independent factors $\boldsymbol{v}$ in a different subset of $\boldsymbol{z}$ than the conditionally dependent factors $\boldsymbol{w}$. β controls the degree of this pressure. Also note that $\beta = 1$ yields the $ELBO$ formulated earlier (3.22), and $\beta = 0$ yields the maximum likelihood case. It's hypothesised that disentangled representations are learnt for $\beta > 1$.

As the prior needs to be isotropic, it's often chosen to be the standard multivariate Gaussian:

$$p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{3.41}$$

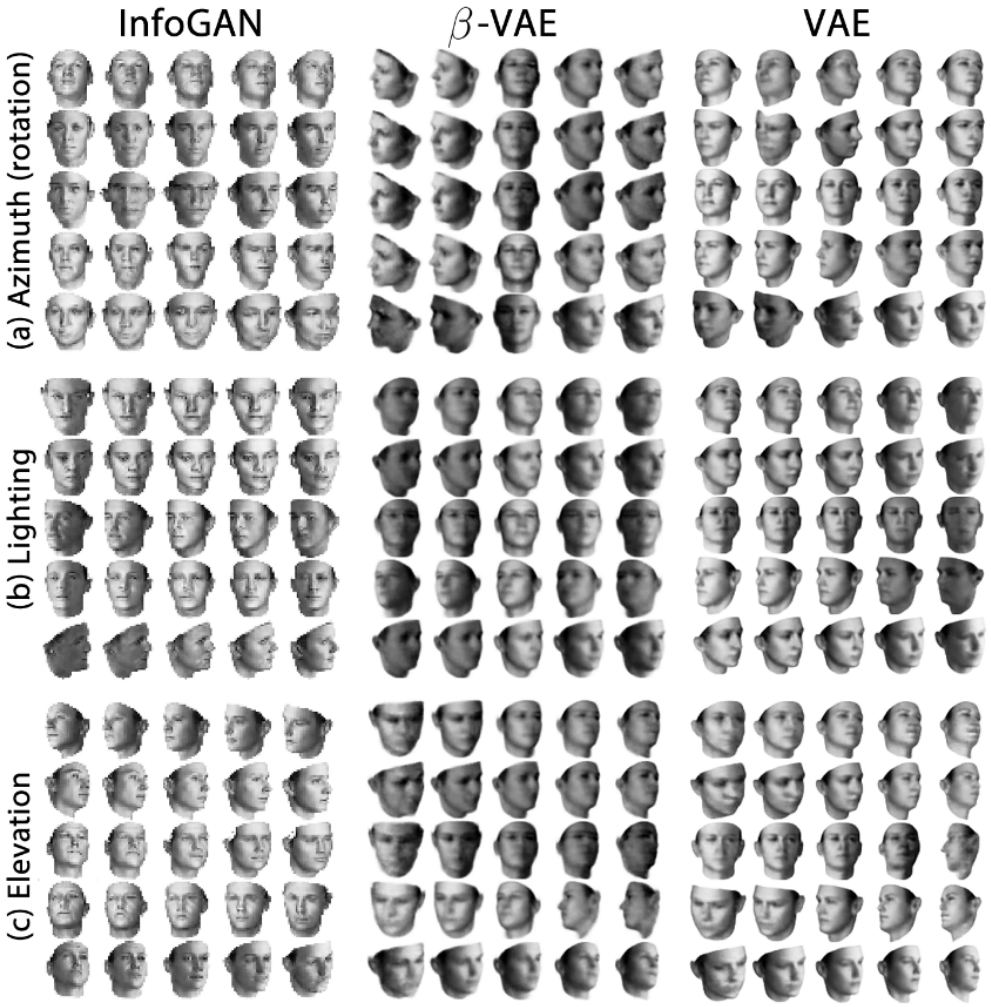


Figure 3.12: A comparison of InfoGAN, β-VAE ($\beta = 20$) and VAE on the 3D face data set. Different latent variables are varied for sections (a), (b) and (c). All models learnt lighting and elevation, but only InfoGAN and β-VAE managed to continuously vary the azimuth. *Source:* [32].

3.4 Improving Sampling from Generative Autoencoders with Markov Chains

A generative autoencoder may be defined as an autoencoder that pressures its latent distribution $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ to match a given prior $p(\boldsymbol{z})$ [8]. As discussed earlier, the KL divergence term in the $ELBO$ applies this pressure:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] \tag{3.23}$$

In Section (3.2) it was assumed that the data set X was generated by

$$\boldsymbol{z}^{(i)} \sim p(\boldsymbol{z}) \qquad \boldsymbol{x}^{(i)} \sim p_{\boldsymbol{\theta}^*}(\boldsymbol{x}|\boldsymbol{z}^{(i)}) \tag{3.42}$$

where $p(\boldsymbol{z})$ is a parameterless prior distribution and $p_{\boldsymbol{\theta}^*}(\boldsymbol{x}|\boldsymbol{z}^{(i)})$ is the true conditional. As the true set of parameters $\boldsymbol{\theta}^*$ is unknown to us, we may attempt to generate samples with learnt parameters $\boldsymbol{\theta}$:

$$\boldsymbol{z}^{(i)} \sim p(\boldsymbol{z}) \qquad \boldsymbol{x}^{(i)} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}^{(i)}) \tag{3.43}$$

Now suppose

$$\int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} = \hat{p}(\boldsymbol{z}) \tag{3.44}$$

where $p(\boldsymbol{x})$ is the distribution that generated data set X. The generative procedure (3.43) makes the erroneous assumption that $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ perfectly matches the prior $p(\boldsymbol{z})$, that is, $p(\boldsymbol{z}) = \hat{p}(\boldsymbol{z})$ [8]. Since only pressure was applied to match the two, this is clearly not true in general. That is, in general,

$$p(\boldsymbol{z}) \neq \hat{p}(\boldsymbol{z}) \tag{3.45}$$

Therefore we have

$$\int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\hat{p}(\boldsymbol{z})d\boldsymbol{z} = p(\boldsymbol{x}) \neq \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z} \tag{3.46}$$

which suggests that the generative procedure (3.43) does not produce samples from $p(\boldsymbol{x})$ exactly. To sample from $p(\boldsymbol{x})$, a new generative procedure is proposed [8]:

$$\boldsymbol{z}^{(i)} \sim \hat{p}(\boldsymbol{z}) \qquad \boldsymbol{x}^{(i)} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}^{(i)}) \tag{3.47}$$

As $\hat{p}(\boldsymbol{z})$ is unknown, it is not obvious how to sample from generative autoencoders using

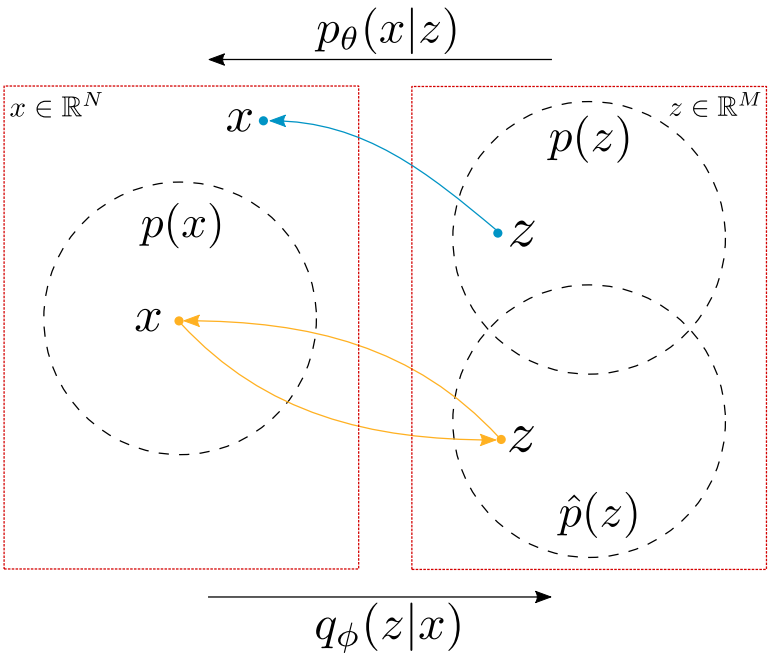


Figure 3.13: The probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ maps a given data point $\boldsymbol{x}$ to the unknown distribution $\hat{p}(\boldsymbol{z})$. The probabilistic decoder $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ is trained to map samples from $\hat{p}(\boldsymbol{z})$ back to $p(\boldsymbol{x})$, since its inputs are drawn from the probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. A sample from the prior $p(\boldsymbol{z})$ will not be mapped back by $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ to $p(\boldsymbol{x})$ exactly if $p(\boldsymbol{z}) \neq \hat{p}(\boldsymbol{z})$.
Adapted from: [8].

procedure (3.47). However, it is possible to formulate a Markov chain Monte Carlo (MCMC) chain that starts with an arbitrary latent variable $\boldsymbol{z}_{t=0}$ and converges with $\boldsymbol{z}_{t\to\infty} \sim \hat{p}(\boldsymbol{z})$ [8]. This is done by successively encoding and decoding the same sample as follows:

$$\boldsymbol{x}_{t=k+1} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_{t=k}) \qquad \boldsymbol{z}_{t=k+1} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_{t=k+1}) \tag{3.48}$$

where $\boldsymbol{z}_{t=0}$ is arbitrarily chosen.

(a) VAE (initial) (b) VAE (1 step) (c) VAE (5 steps) (d) VAE (10 steps)

Figure 3.14: Samples from a variational autoencoder trained on the CelebA data set after $t = 0, 1, 5$ and 10 steps of the generative procedure (3.48). The MCMC chain was initialised with a sample from the prior $\boldsymbol{z}_{t=0} \sim p(\boldsymbol{z})$, which often improves the quality of the samples. *Source:* [8].

Chapter 4

Implementation

4.1 Dimensionality Reduction

Machine learning with high-dimensional data, such as images, is often computationally intensive. Atari 2600 frames are RGB images (3 channels) of size 210×160, expressed in shorthand as $(3, 210, 160)$. (This data is much higher in dimensionality than MNIST, which has dimensions $(1, 28, 28)$). Considering the training sets considered in this project are of hundreds of thousands of images, data points of dimension $(3, 210, 160)$ are too computationally intensive with the best hardware available ($2 \times$ NVIDIA Tesla K80 GPU Accelerators). It is therefore necessary to reduce the dimensionality of our data set. Google DeepMind's *Human-level Control Through Deep Reinforcement Learning* [21] made use of Stella, as we have, and have convincingly struck a reasonable balance between resolution and dimensionality reduction. This section will be a short but necessary mentioning of the pre-processing pipeline used to generate the data sets in later chapters.

4.1.1 Pre-processing Pipeline

Ensuring Object Persistence

Atari 2600 games can only store a limited number of sprites per frame due to the limitations in hardware during its development [21]. This is an issue as some objects that appear in one frame fail to appear in the next. To solve this lack of object persistence, even and odd frames are combined by taking the maximum over each channel (RGB). By taking the maximum, we ensure that any object present in one frame is also present in the other.

Extracting Luminance and Cropping

The luminance Y is then extracted from the RGB image [29]:

$$Y = 0.2126 \times R + 0.7152 \times G + 0.0722 \times B \tag{4.1}$$

The resultant greyscale image of shape $(1, 210, 160)$ is cropped to $(1, 84, 84)$.

File Formats

Image formats were discovered to be more important than originally thought. Empirically, PNG or GIF formats were reasonable formats, but using JPEG often resulted in distortions near the perimeter of objects. An example of this effect is shown in Figure (4.2).

4.2 Qualitative Assessment Using GUIs

To qualitatively assess the significance of a latent neuron in the final reconstruction, we change its value over a given range and inspect the reconstructions. In order to speed up this process, Tkinter was used to develop a GUI with a sliding bar corresponding to the latent neuron's

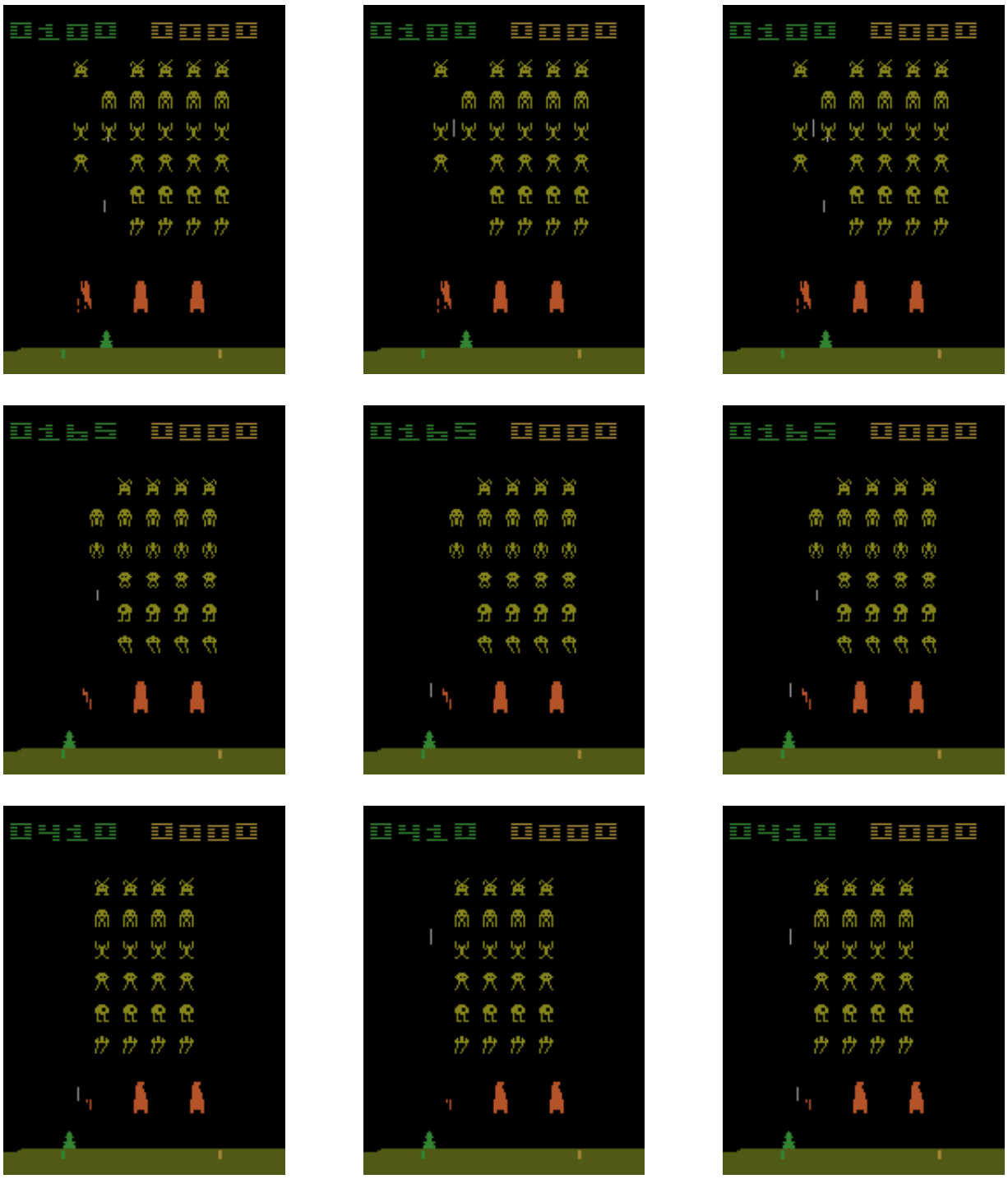


Figure 4.1: A collection of frames captured from Space Invaders emulated on Stella. **Left column:** an even frame. **Middle column:** the (odd) frame following. **Right column:** Combining the even and odd frames by taking the maximal value over each channel (RGB). Clearly the bullets visible in one frame fail to persist in the next. As mentioned, this is due to the limited number of sprites Atari 2600 can load in a single frame.

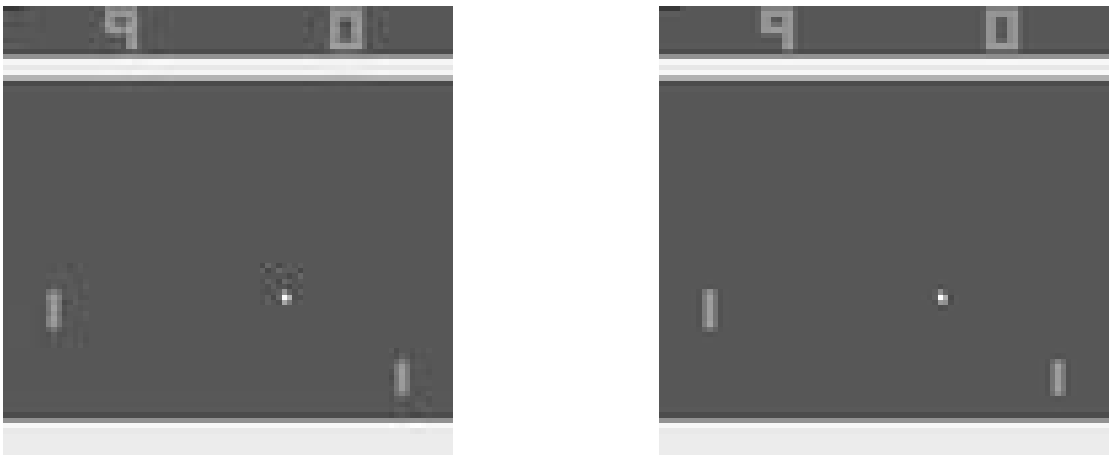


Figure 4.2: Pre-processed frames captured from Pong emulated on Stella. These frames were originally 84×84, but are printed here as 168×168 to emphasise distortions. **Left:** The JPEG format distorts the ball, paddle and score sprites. **Right:** The PNG format displays the frame without such distortions.

value. This real-time reconstruction allows for much quicker feedback, and hence we were able to explore the reconstruction process much quicker.

4.3 Training and Validation Data Generators

Since our data set consists of $\sim 100,000$ high-dimensional data points, it is not possible to load these directly into memory. Instead, a custom data generator was made that pulls data samples from a given directory. In this way, only `batch_size`-many data points are loaded at once. Once a data point is pulled, an internal counter is incremented so that no data point is repeated. The data generator then throws an exception when it's queried to return more unique data points than it's given.

4.4 Ensuring Numerical Stability in the Latent Space

In the implementation of the variational autoencoder loss function, we use log variance instead of variance to spread values evenly, which helps with backpropagation. This is because it's easier to learn with an approximately equal step size between values rather than those that are

either clumped or sparse. Since log is monotonic, the value ordering is invariant, which allows back propagation to implicitly optimise the variance σ^2.

4.5 Activation Functions in the Latent Space

Conventionally, activation functions are used in the hidden layers, such as sigmoid or, more commonly, ReLU. However, we must be careful not to mindlessly follow this trend. It makes no sense to use a ReLU activation function on the mean layer of the variational autoencoder. The mean $\mu \in \mathbb{R}$ is unconstrained, and hence a no activation function is appropriate. Likewise, since we use log variance, we also have the property that $\ln \sigma \in \mathbb{R}$. By the same reasoning, we do not apply an activation function to this layer to leave $\ln \sigma$ unbounded.

4.6 Keras Callbacks

Throughout the results section we use a number of Keras' features. These are:

- **Learning rate annealing.** Reducing the learning rate with the number of epochs can be quite beneficial in consistently achieving an optimum.
- **Check-pointing model weights and architecture.** Saving the model after each epoch (if it betters the previous best) is extremely useful. This allows for the continuation of training at a later date and for real time testing.
- **Reduce LR on plateau.** Often it's hard to pick the right factor to reduce the learning rate each epoch when implementing LR annealing. An easier method is reducing the LR on plateau, where the LR is reduced by a given factor when the monitored quantity does not improve.
- **Early stopping.** When training models, sometimes the model starts to make very marginal improvements. Early stopping terminates the training at this point, often significantly reducing training time.

Chapter 5

Methods

> "It is not unscientific to make a guess, although many people who are not in science think it is."
>
> *R. P. Feynman*

In order for symbolic extraction to take place in a DSRL system, the latent space must preserve the types of objects and the spatial information of the original scene. In other words, from the latent space alone, it must be possible to deduce an object's type and location. We explore the feasibility of using fully-convolutional variational autoencoders for this purpose. This is exciting new territory; the fully-convolutional variational autoencoder has (to the best of our knowledge) never been explored. We therefore have many degrees of freedom with how we proceed. In this section, we will make educated guesses of how a working architecture would look and encode desirable properties in loss functions, which will be inspired by the successful techniques developed in β-VAE.

5.1 Single Latent Filter

As mentioned, the type and position of an object must be preserved in the latent space. This may be achieved by using a single latent filter, with the object type corresponding to the value

of the activation at the relevant position on the filter. Recall that it does not make sense to use an activation function in the latent space, and therefore the object type is represented by an unbounded real number.

5.1.1 Architecture

The input image is passed through convolutional layers to build increasingly meaningful hidden representations. The convolutional mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$, both of shape $(1, m, n)$, are sampled using the reparameterisation trick

$$\boldsymbol{z} = g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{3.30}$$

so we may use backpropagation. The corresponding deconvolutional layers are applied to reconstruct the original image.

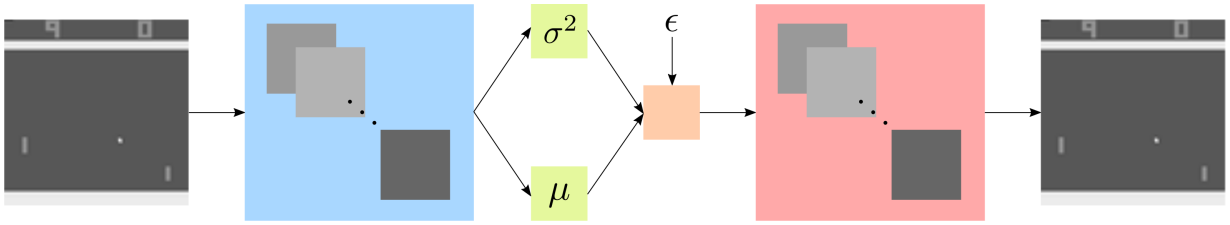


Figure 5.1: The fully-convolutional single latent filter architecture. **Blue:** An arbitrary amount of convolutional layers. **Green:** The latent mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$, which are both single filters of shape $(1, m, n)$. **Orange:** A single latent filter of shape $(1, m, n)$ sampled component-wise from $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$. **Red:** The corresponding deconvolutional layers.

5.1.2 Neuron-Level Redundancy Reduction

As with all autoencoders, a reconstruction loss term must appear in the loss function as to learn the identity function, which ensures a meaningful lower-dimensional representation is learnt in the latent space. Hence we also include the reconstruction loss term

$$\mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] \tag{5.1}$$

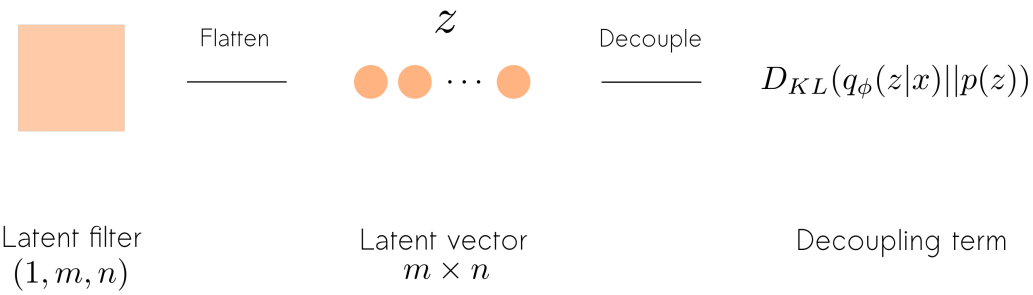
in the final loss function for this architecture.

We also saw in β-VAE that the multiplicative factor on the KL loss term

$$-\beta D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) \tag{5.2}$$

varies the pressure of redundancy reduction on the latent space. As we seek to have non-zero activations in the latent space if and only if an object is present, it's in our interest to reduce the number of unnecessarily activated neurons. Therefore the application of a redundancy pressure term seems suitable, and is therefore included in the loss function. However, as we no longer have a one-dimensional latent space, we must decide on how this term will appropriately reduce redundancy in a two-dimensional latent space.

To reduce the redundancy in a two-dimensional latent space, we choose to flatten the $(1, m, n)$-dimensional single latent filter to an $m \times n$ vector $\boldsymbol{z}$, which is then matched to the prior $p(\boldsymbol{z})$, an isotropic Gaussian. We choose the prior to be $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ for convenience.



By reducing the two-dimensional latent space to one dimension, we arrive at the same loss function as in the β-VAE case:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = -\beta D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] \tag{5.3}$$

5.2 Multiple Latent Filters

Although it's conceivable that a single latent filter could represent types and locations of objects in the scene, it seems unlikely to do so. It's well known that filters may learn a set of weights to recognise information-rich features in the scene, like edges or corners. Perhaps this property could be leveraged by using multiple filters in the latent space, where any given latent filter is not responsible for representing multiple types of objects, but one specific object.

By the same reasoning for the single latent filter architecture, we keep the reconstruction loss

$$\mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] \tag{5.4}$$

to learn a meaningful low-dimensional representation in the latent space. However, we propose three techniques of redundancy reduction in the latent space to achieve our desiderata:

- Neuron-level redundancy reduction
- Naïve filter-level redundancy reduction
- Weighted filter-level redundancy reduction

5.2.1 Architecture

The architecture is the same as the fully-convolutional single lantent filter architecture, but with strictly more than one filter in the latent space. The convolutional mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$, both of shape (k, m, n), are sampled using the reparameterisation trick

$$\boldsymbol{z} = g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{3.30}$$

so we may use backpropagation. The corresponding deconvolutional layers are applied to reconstruct the original image.

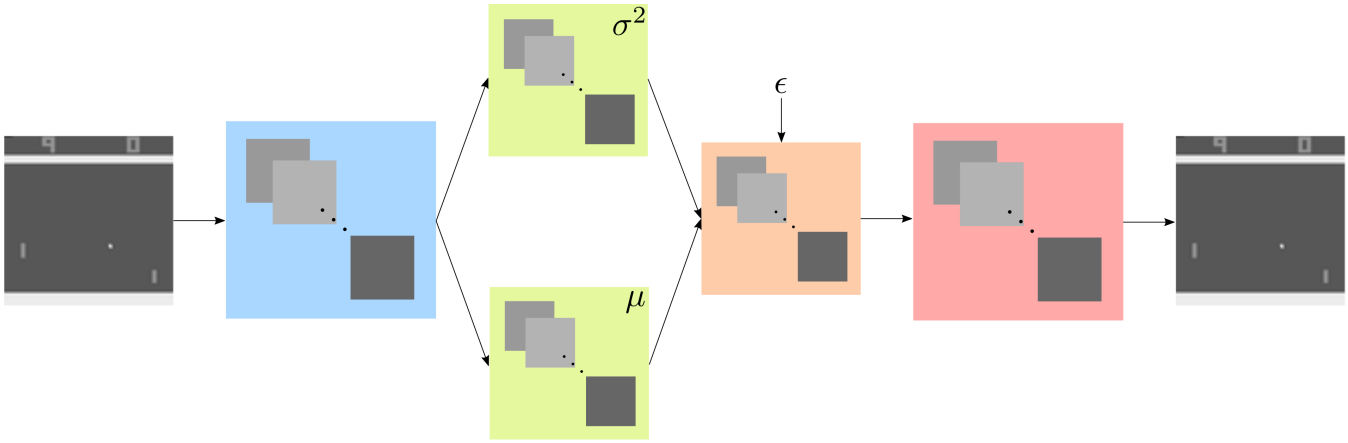
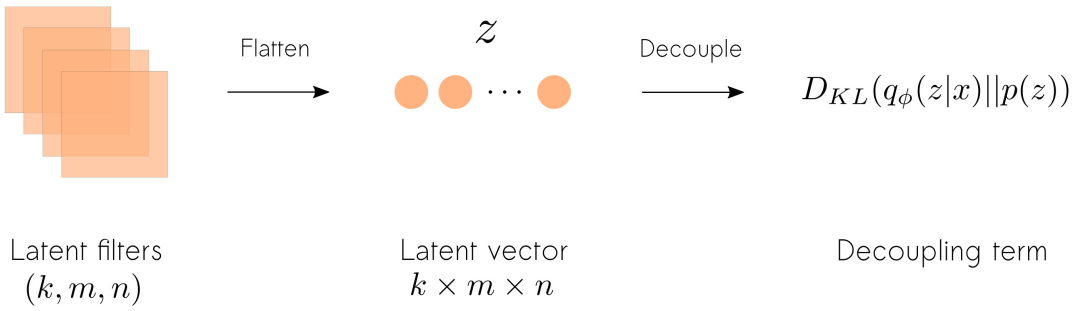


Figure 5.2: The fully-convolutional multiple latent filter architecture. **Blue:** Unchanged. **Green:** The latent mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$, which are both single filters of shape (k, m, n). **Orange:** A single latent filter of shape (k, m, n) sampled component-wise from $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$. **Red:** Unchanged.

5.2.2 Neuron-Level Redundancy Reduction

To reduce the redundancy of activated neurons across the whole (k, m, n)-dimensional latent space, so that neurons are only activated when necessary, we apply neuron-level redundancy reduction. The latent filter of shape (k, m, n) is flattened to a $k \times m \times n$ vector $\boldsymbol{z}$, which is then pressured to match the prior $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.



The loss function is therefore written as

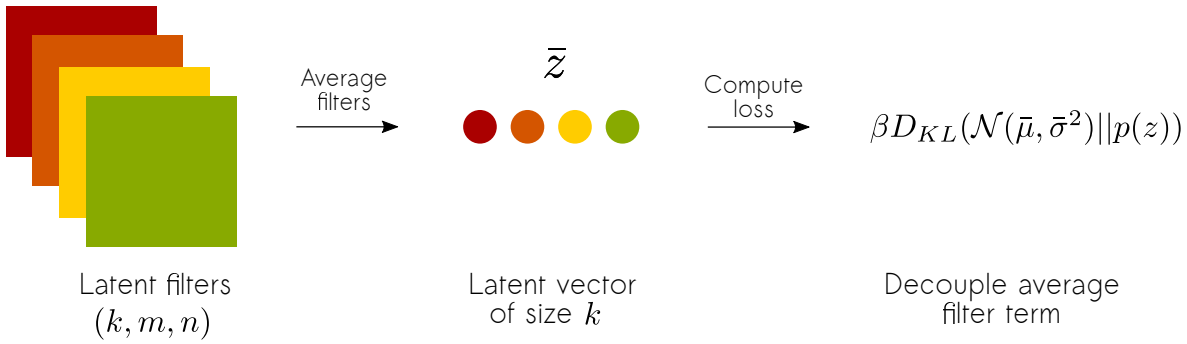
$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = -\beta D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] \tag{5.5}$$

where $\boldsymbol{z} \in \mathbb{R}^{k \times m \times n}$ is the flattened (k, m, n)-dimensional latent space.

5.2.3 Naïve Filter-Level Redundancy Reduction

β-VAE sought to factorise the latent space into independent and dependent generative factors of the scene, in effect "disentangling" the latent space. By similar reasoning, we'd like the activation of filters to clearly correspond to independent object types, with no two filters active for the same object. Hence we seek a factorisation over the activity of latent filters with no redundancy.

To achieve this, the k filters of the (k, m, n)-dimensional latent space are averaged across their activations and stored in a vector $\bar{\boldsymbol{z}}$. We then pressure the vector $\bar{\boldsymbol{z}}$ to match the prior $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.



The loss function is therefore written as

$$\mathcal{L}_{NF}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = -\beta D_{KL}(\mathcal{N}(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\sigma}}^2)||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p(\boldsymbol{x}|\boldsymbol{z})\big] \tag{5.6}$$

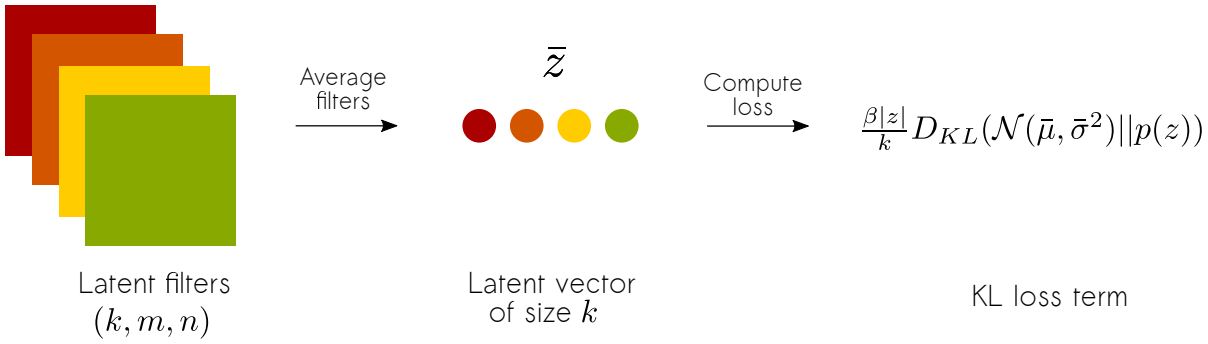
where $\bar{\boldsymbol{\mu}} \in \mathbb{R}^k$ and $\bar{\boldsymbol{\sigma}}^2 \in \mathbb{R}^k$ are the averages of $\boldsymbol{\mu} \in \mathbb{R}^{k \times m \times n}$ and $\boldsymbol{\sigma}^2 \in \mathbb{R}^{k \times m \times n}$.

5.2.4 Weighted Filter-Level Redundancy Reduction

It's a subtle but necessary point that we should be careful to evaluate loss terms in a comparable way. For example, taking the sum of one and comparing it to the average of another (which is always less than the sum) serves to bias the loss function. This leads to the possibility that the average filter activation KL loss term in the naive filter-level redundancy technique

is significantly under-represented when taking the sum reconstruction loss. (For simplicity, assume that we will take the sum reconstruction loss from here on.)

When evaluating the average filter activations, we get a single floating point number for $m \times n$ pixels, where m and n are the width and height of the latent filter respectively. Hence, when calculating the KL loss term, we may multiply by $m \times n$ so that the KL loss and reconstruction loss are of comparable size.



The loss function is therefore written as

$$\mathcal{L}_{NF}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = -\frac{\beta|\boldsymbol{z}|}{k} D_{KL}(\mathcal{N}(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\sigma}}^2)||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p(\boldsymbol{x}|\boldsymbol{z})\big] \tag{5.7}$$

where $\bar{\boldsymbol{\mu}} \in \mathbb{R}^k$ and $\bar{\boldsymbol{\sigma}}^2 \in \mathbb{R}^k$ are the averages of $\boldsymbol{\mu} \in \mathbb{R}^{k \times m \times n}$ and $\boldsymbol{\sigma}^2 \in \mathbb{R}^{k \times m \times n}$.

5.3 Winner Takes All

As mentioned, we'd like the activation of filters to clearly correspond to independent object types, with no two filters active for the same object. Assuming the latent space takes this structure, we would expect no more than one filter to be activated in any given position. (Objects must have distinct types: a sprite cannot be a gunship and a number at the same time.) This leads to the Winner Takes All method, where we apply a position-wise pressure to the latent space so that no more than one filter has a high activation in any given position.

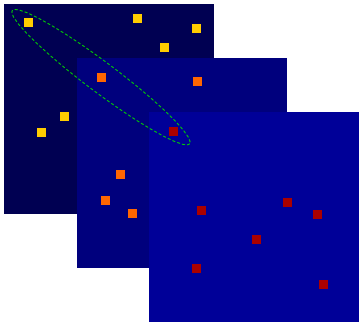


Figure 5.3: Multiple types of objects recognised in same position.

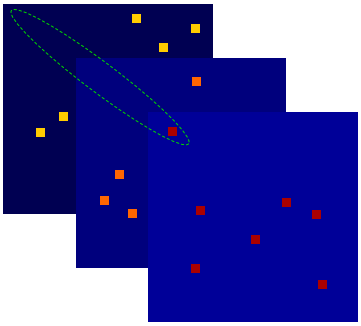


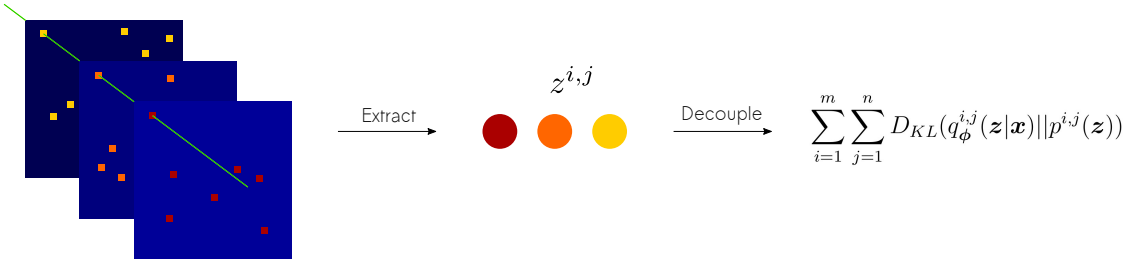
Figure 5.4: At most one object recognised in a given position.

5.3.1 Position-Wise Redundancy Reduction

Consider a convolutional latent space $\boldsymbol{z}$ of shape (k, m, n). Let $\boldsymbol{z}^{i,j} \in \mathbb{R}^k$ be the vector storing the activations over the k filters at position (i, j). Naturally, we also let $q_{\boldsymbol{\phi}}^{i,j}(\boldsymbol{z}|\boldsymbol{x})$ represent the probabilistic encoder for $\boldsymbol{z}^{i,j} \in \mathbb{R}^k$. By matching the probabilistic encoder to an isotropic Gaussian we may reduce the redundancy, as argued previously. We choose to include the following term in our loss function:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} D_{KL}(q_{\boldsymbol{\phi}}^{i,j}(\boldsymbol{z}|\boldsymbol{x})||p^{i,j}(\boldsymbol{z})) \quad \text{where} \quad p^{i,j}(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{5.8}$$

As shown with β-VAE, the multiplicative β controls the redundancy pressure. We will include this also and see if we observe a similar effect later.



Therefore the final loss function is given as:

$$\mathcal{L}_{WTA}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] - \beta \sum_{i=1}^{m} \sum_{j=1}^{n} D_{KL}(q_{\boldsymbol{\phi}}^{i,j}(\boldsymbol{z}|\boldsymbol{x})||p^{i,j}(\boldsymbol{z})) \tag{5.9}$$

5.4 Separating Colour Spaces

As the task of unsupervised object recognition in fully-convolutional variational autoencoders is an open problem, it's reasonable to start by solving easy recognition tasks. Since sprites in Atari games are of different colours, we can use these colour channels to make the separation of objects easier. An example for a frame of Space Invaders is shown in Figure (5.9). Since this technique is independent of architecture, it applies to all mentioned in the chapter.

Figure 5.5: Original

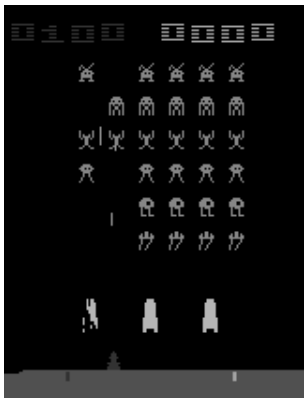


Figure 5.6: Red

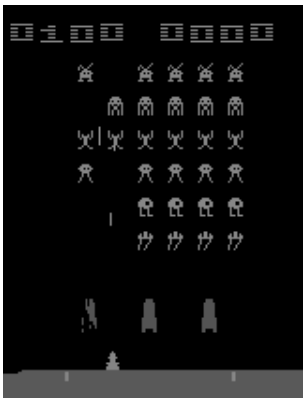


Figure 5.7: Green

Figure 5.8: Blue

Figure 5.9: A comparison of a 210 × 160 RGB frame from Space Invaders and its red, green and blue channels. The bullet is clearly separated from other sprites in the blue channel. The red and green channels separate collections of sprites. The red channel excludes the gunship and players score, while the green partially excludes the barriers.

One obvious limitation of this method is that the number of sprites separated is limited by the number of colour channels.

Chapter 6

Results

In this chapter we will perform a number of experiments to assess the proposed methods from last chapter. We wish to keep the architectures fixed and vary the method, which is done to accurately assess its feasibility. Of course, some methods require a change in architecture, in which case we make the smallest possible change. First we will mention the architectures used, after which we will explore each proposed method in detail.

Unless specified otherwise, the Adam optimiser was used with a learning rate of $1e-3$. We also use a data set of $100,000$ pre-processed frames of Space Invaders (generated with the Arcade Learning Environment), with a 90-10 training-test split. With such a large data set of high-dimensional data, we train the models with early stopping (until the gains are no longer qualitatively different) to ensure a breadth of parameters are explored.

6.1 Architectures

In this section we will use architectures similar to one proven to capture the generative factors in the Atari game *Space Invaders* [14]. We do not claim that these architectures are optimal, but simply that they are capable of learning the generative factors in the scene.

The architectures used are specified in Tables (6.1), (6.2) and (6.3).

Layer	Output shape	Connected to
InputLayer	(1, 84, 84)	
Conv2D_1	(32, 42, 42)	InputLayer
Conv2D_2	(64, 21, 21)	Conv2D_1
Conv2D_3	(1, 8, 8)	Conv2D_2
Conv2D_4	(1, 8, 8)	Conv2D_2
Sampling	(1, 8, 8)	Conv2D_3 & Conv2D_4
Deconv2D_1	(64, 20, 20)	Sampling
Deconv2D_2	(32, 40, 40)	Deconv2D_1
Deconv2D_3	(1, 84, 84)	Deconv2D_2

Table 6.1: Fully-convolutional single-filter variational autoencoder.

Layer	Output shape	Connected to
InputLayer	(1, 84, 84)	
Conv2D_1	(32, 42, 42)	InputLayer
Conv2D_2	(64, 21, 21)	Conv2D_1
Conv2D_3	(8, 8, 8)	Conv2D_2
Conv2D_4	(8, 8, 8)	Conv2D_2
Sampling	(8, 8, 8)	Conv2D_3 & Conv2D_4
Deconv2D_1	(64, 20, 20)	Sampling
Deconv2D_2	(32, 40, 40)	Deconv2D_1
Deconv2D_3	(1, 84, 84)	Deconv2D_2

Table 6.2: Fully-convolutional multiple latent filter variational autoencoder.

Layer	Output shape	Connected to
InputLayer	(3, 84, 84)	
Conv2D_1	(32, 42, 42)	InputLayer
Conv2D_2	(64, 21, 21)	Conv2D_1
Conv2D_3	(8, 8, 8)	Conv2D_2
Conv2D_4	(8, 8, 8)	Conv2D_2
Sampling	(8, 8, 8)	Conv2D_3 & Conv2D_4
Deconv2D_1	(64, 20, 20)	Sampling
Deconv2D_2	(32, 40, 40)	Deconv2D_1
Deconv2D_3	(3, 84, 84)	Deconv2D_2

Table 6.3: Fully-convolutional multiple latent filter variational autoencoder for RGB images.

6.2 Single Latent Filter

6.2.1 Results

The single latent architecture, found in Table (6.1), was trained for 10 epochs, and the validation, validation KL and validation reconstruction loss recorded. The results are shown in Figure (6.1). The single latent image method is clearly capable of achieving reasonable reconstruction and KL loss on unseen data.

We see that the reconstructions of several inputs, shown in Figure (6.14), are qualitatively perfect. For each value of β, the reconstruction is indistinguishable from its input. This comes as quite a surprise, as we initially suspected this method is too restrained in the latent space.

The latent representations of scenes, shown in Figure (6.27), do not meaningfully correspond to their original scene. Even for high β, the scenes do not seem to become anything more interpretable than noise. We further took the average of the latent filters over the test set of $10,000$ data points, and achieved similar results, which are shown in Figure (6.40).

Likewise, the decoded samples from the prior $p(\boldsymbol{z})$ and unknown distribution $\hat{p}(\boldsymbol{z})$, shown in Figures (6.31) and (6.36) respectively, do not have any meaningful relation to the original scene. This is true for all values of β tested. This tells us that even for reasonable values of the KL loss term, the latent image method struggles to generate realistic samples.

6.2.2 Summary

- The latent image method is capable of qualitatively perfect reconstruction of its input.
- Despite the perfect reconstruction, the method seems to struggle with learning interpretable latent representations of high-level concepts in the scene for all values of β.
- The latent image method also seemed to struggle to produce meaningful generative samples from the prior $p(\boldsymbol{z})$ and unknown distribution $\hat{p}(\boldsymbol{z})$ despite reasonable values of validation KL loss.

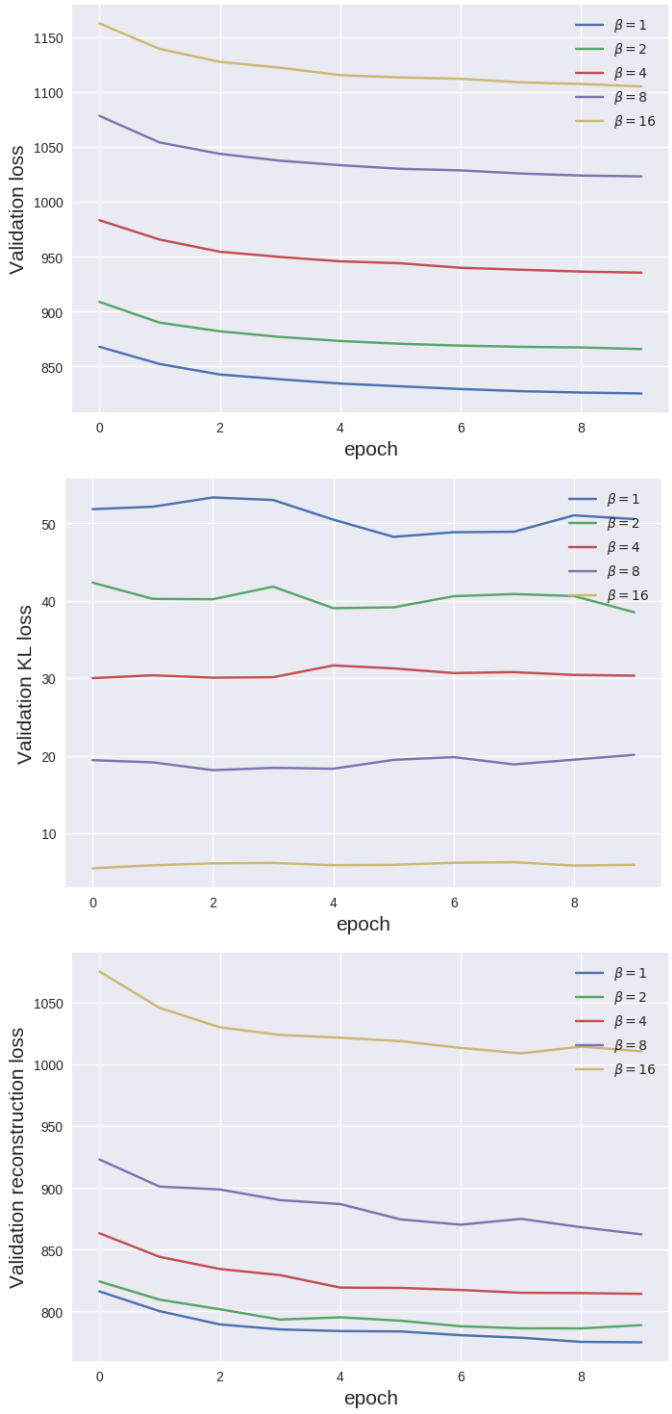


Figure 6.1: **Single latent filter architecture**. The validation, validation KL and validation reconstruction loss for the latent image architecture and different values of β.

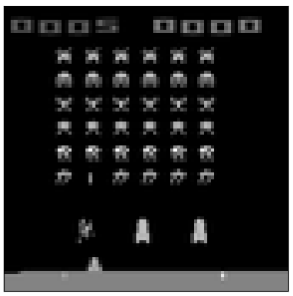


Figure 6.2: Original

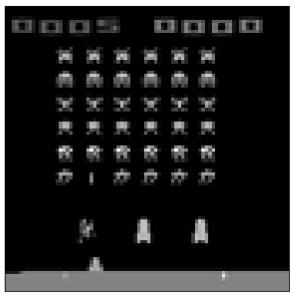


Figure 6.3: $\beta = 1$

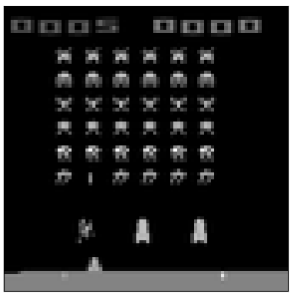


Figure 6.4: $\beta = 4$

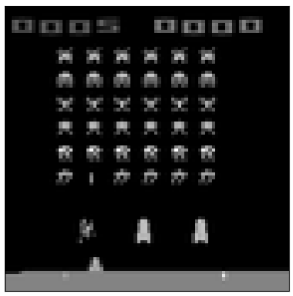


Figure 6.5: $\beta = 16$

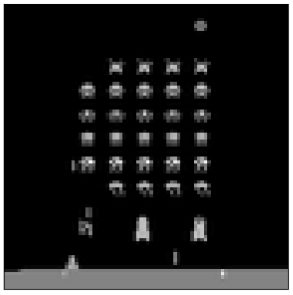


Figure 6.6: Original

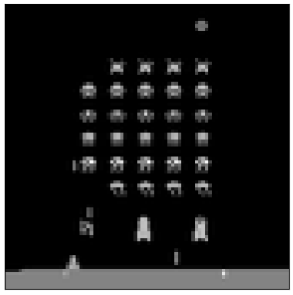


Figure 6.7: $\beta = 1$

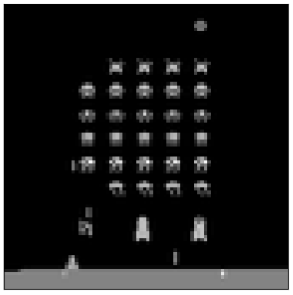


Figure 6.8: $\beta = 4$

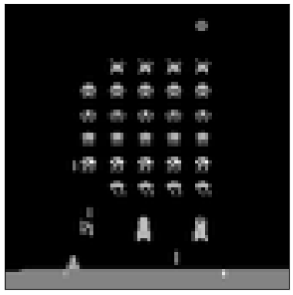


Figure 6.9: $\beta = 16$

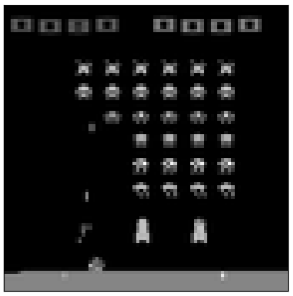


Figure 6.10: Original

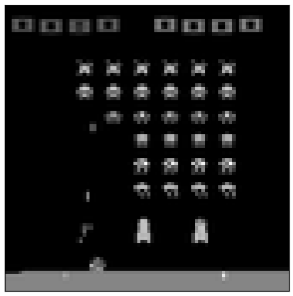


Figure 6.11: $\beta = 1$

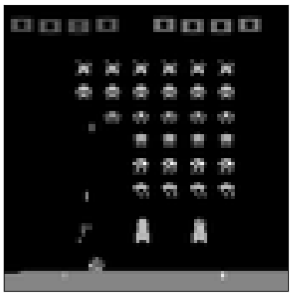


Figure 6.12: $\beta = 4$

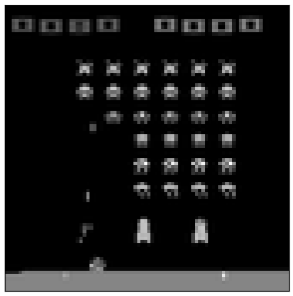


Figure 6.13: $\beta = 16$

Figure 6.14: **Single latent filter architecture**. A selection of Space Invader frames and their corresponding reconstructions for different values of β.

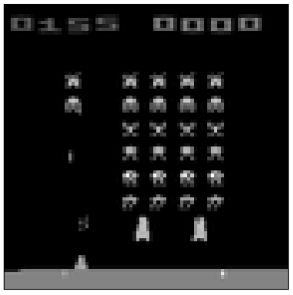


Figure 6.15: Original

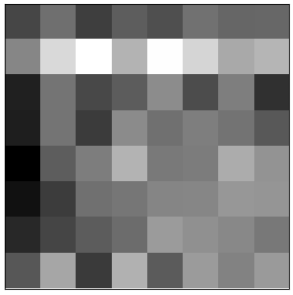


Figure 6.16: $\beta = 1$

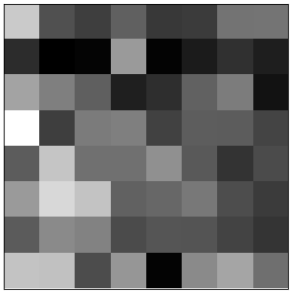


Figure 6.17: $\beta = 4$

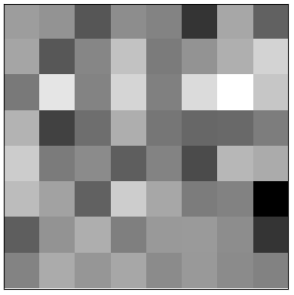


Figure 6.18: $\beta = 16$

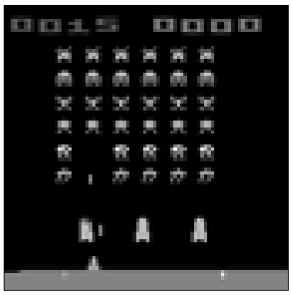


Figure 6.19: Original

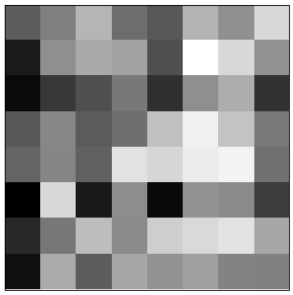


Figure 6.20: $\beta = 1$

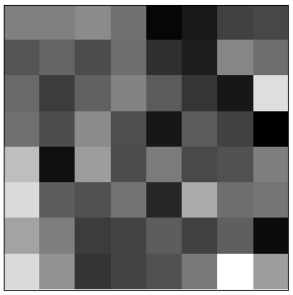


Figure 6.21: $\beta = 4$

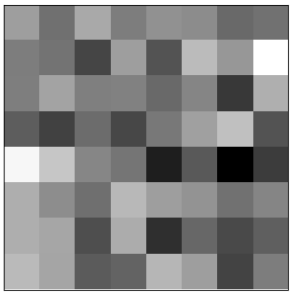


Figure 6.22: $\beta = 16$

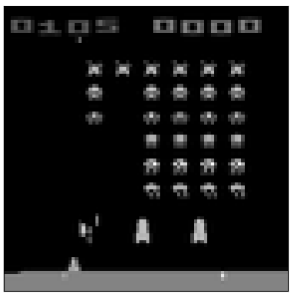


Figure 6.23: Original

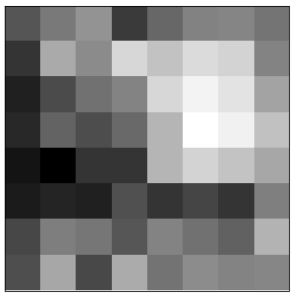


Figure 6.24: $\beta = 1$

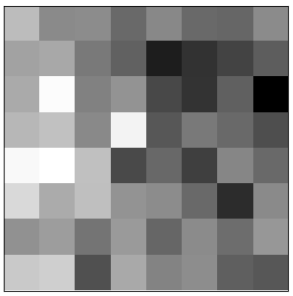


Figure 6.25: $\beta = 4$

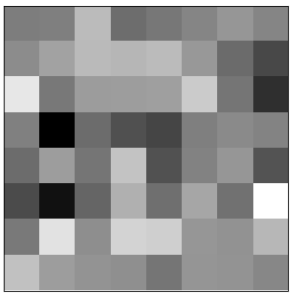


Figure 6.26: $\beta = 16$

Figure 6.27: **Single latent filter architecture**. Frames from Space Invaders and their corresponding latent images for different values of β.

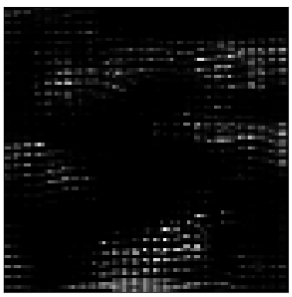


Figure 6.28: $\beta = 1$

Figure 6.29: $\beta = 4$

Figure 6.30: $\beta = 16$

Figure 6.31: **Single latent filter architecture**. The best of 10 samples from the prior $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ for different values of β.

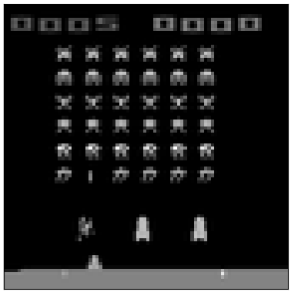


Figure 6.32: Original

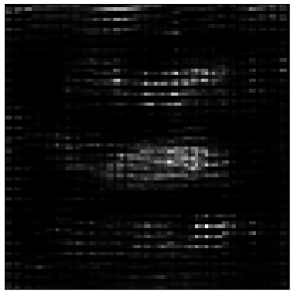


Figure 6.33: $\beta = 1$

Figure 6.34: $\beta = 4$

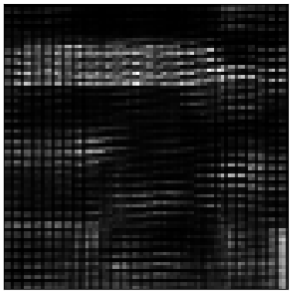


Figure 6.35: $\beta = 16$

Figure 6.36: **Single latent filter architecture**. Samples from the unknown distribution $\hat{p}(\boldsymbol{z})$ after one step of MCMC for different values of β. Samples for subsequent steps did not improve, hence we only include the first for each value of β.

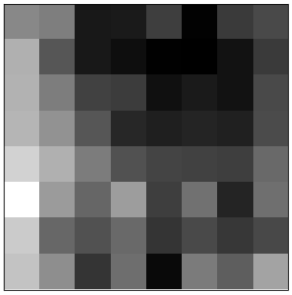


Figure 6.37: $\beta = 1$

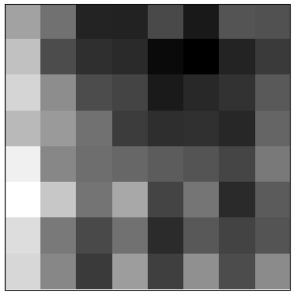


Figure 6.38: $\beta = 4$

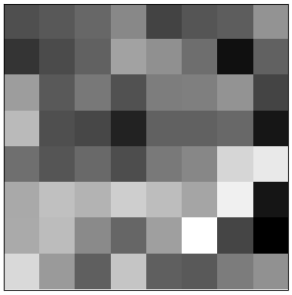


Figure 6.39: $\beta = 16$

Figure 6.40: **Single latent filter architecture**. An average of the activations in the latent image over $10,000$ images from the test set for different values of β.

6.3 Neuron-Level Redundancy Reduction

6.3.1 Results

The multiple latent filter architecture, found in Table (6.2), was trained for 10 epochs with the neuron-level redundancy reduction loss function, and the validation, validation KL and validation reconstruction loss recorded. The results are shown in Figure (6.41). The multiple latent filter architecture with neuron-level redundancy reduction is clearly capable of achieving reasonable reconstruction and KL loss on unseen data, and achieves similar KL loss values as the single latent filter architecture, but considerably better reconstruction loss values.

Although the method reconstructs its input near perfectly, as shown in Figure (6.54), the latent representations do not represent objects in the scene in an interpretable way. This is true also for the averaging of the latent filters over the $10,000$ test data points for all values of β. These results are shown in Figures (6.59) and (6.80) respectively.

Surprisingly, the decoded samples from the prior resemble the original scenes, shown in Figure (6.63). From these results, it's clear that the realism of these generated scenes improves with increasing β. As expected, the generative samples from the unknown distribution $\hat{p}(\boldsymbol{z})$ are more realistic. It's also clear that increasing β encourages the samples from $\hat{p}(\boldsymbol{z})$ to stray further from the input, but in a statistically consistent way.

6.3.2 Summary

- The multiple latent filter architecture with neuron-level redundancy reduction is capable of near perfectly reconstructs of its input.
- It also fails to learn interpretable latent representations of high-level concepts in the scene for all values of β.
- However, it's capable of generating semi-realistic and statistically consistent samples. Varying β has noticeable effects on the realism of the generative samples.

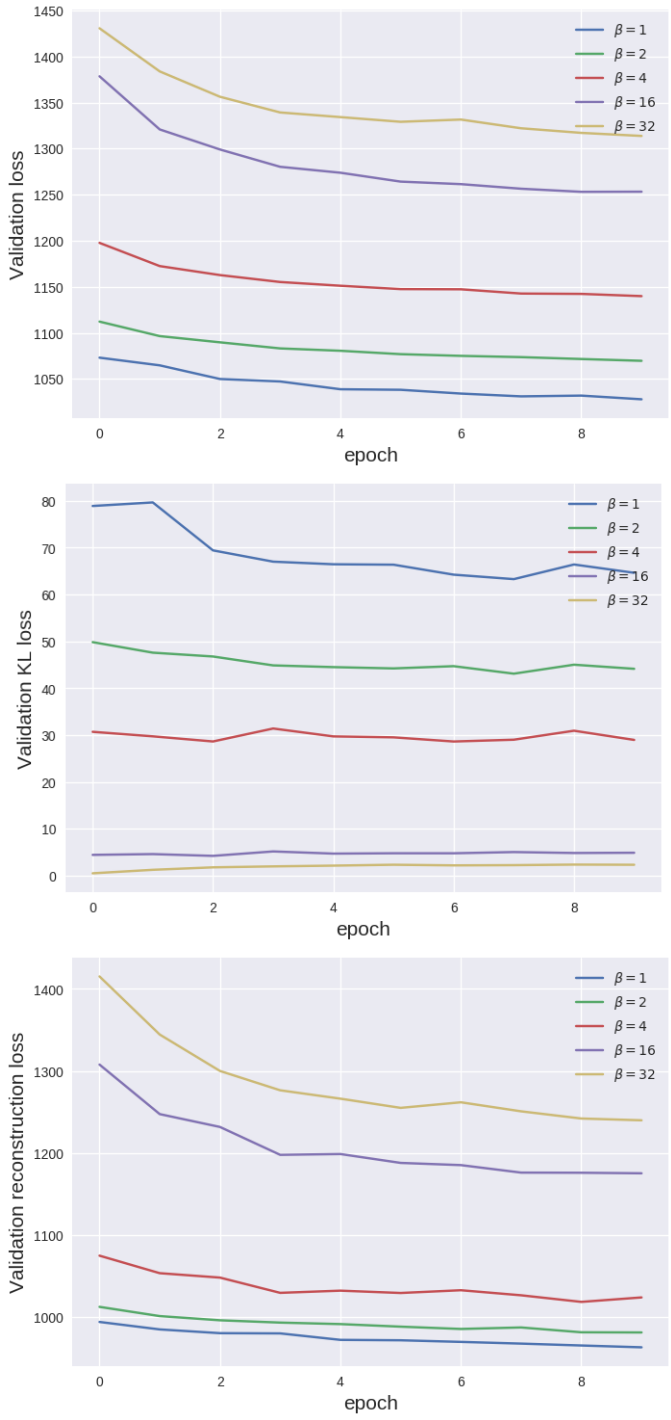


Figure 6.41: **Multiple latent filter archiecture with neuron-level redundancy reduction**. The validation, validation KL and validation reconstruction loss for the latent image architecture and different values of β.

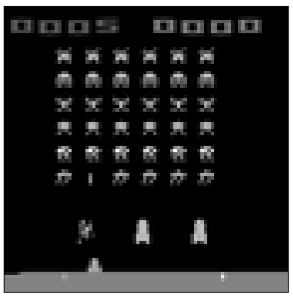


Figure 6.42: Original

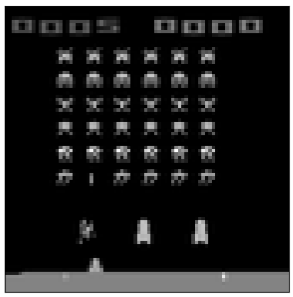


Figure 6.43: $\beta = 1$

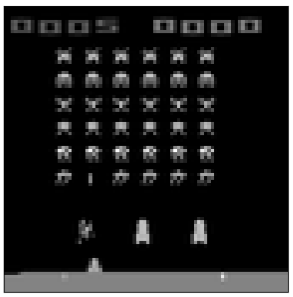


Figure 6.44: $\beta = 4$

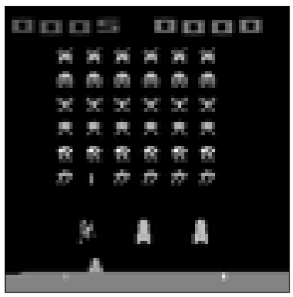


Figure 6.45: $\beta = 32$

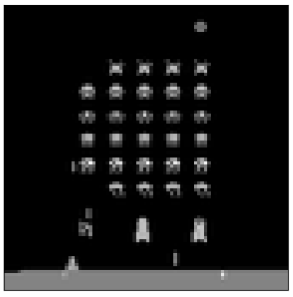


Figure 6.46: Original

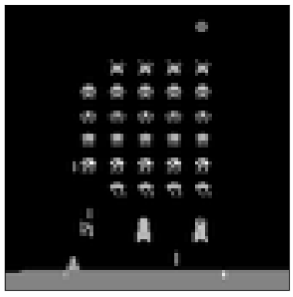


Figure 6.47: $\beta = 1$

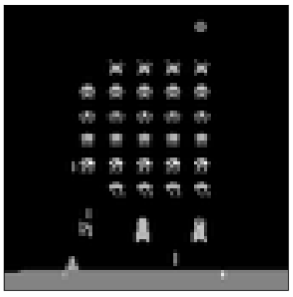


Figure 6.48: $\beta = 4$

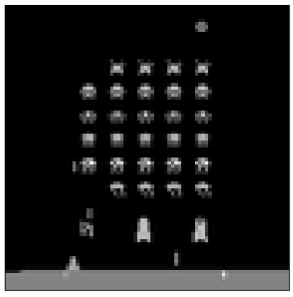


Figure 6.49: $\beta = 32$

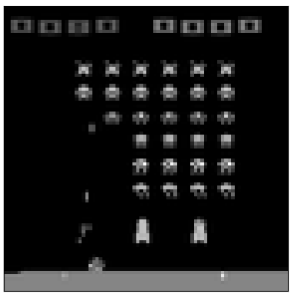


Figure 6.50: Original

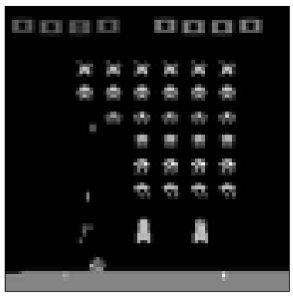


Figure 6.51: $\beta = 1$

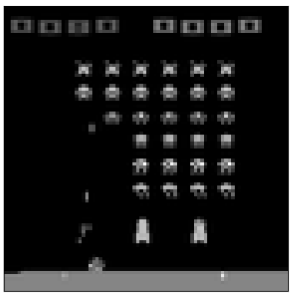


Figure 6.52: $\beta = 4$

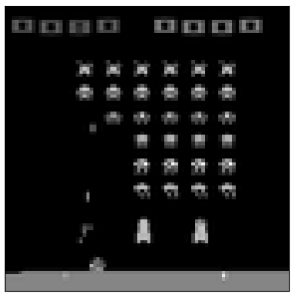


Figure 6.53: $\beta = 32$

Figure 6.54: **Multiple latent filter architecture with neuron-level redundancy reduction**. A selection of Space Invader frames and their corresponding reconstructions for different values of β.

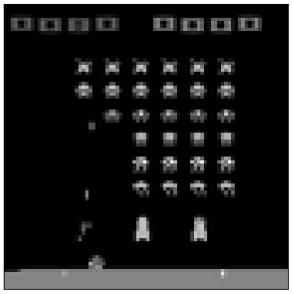


Figure 6.55: Original

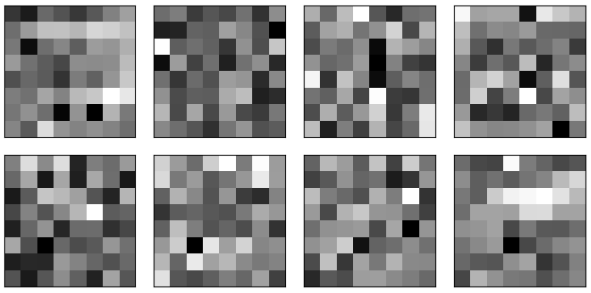


Figure 6.56: $\beta = 1$

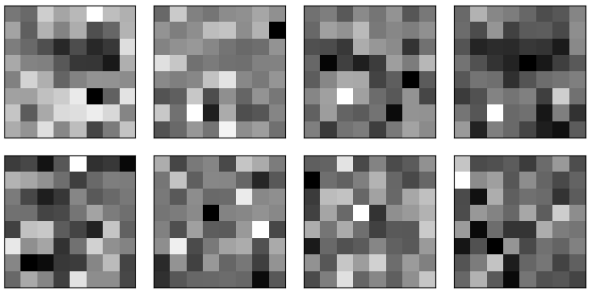


Figure 6.57: $\beta = 4$

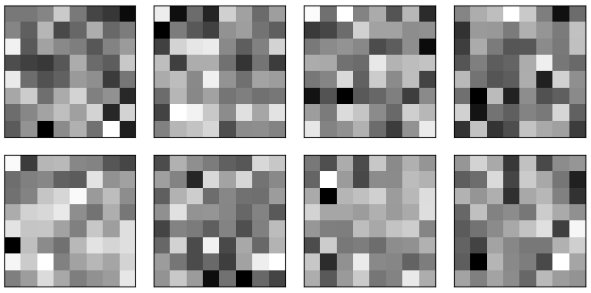


Figure 6.58: $\beta = 32$

Figure 6.59: **Multiple latent filter architecture with neuron-level redundancy reduction**. A Space Invaders frame and the activations over its corresponding latent filters for different values of β.

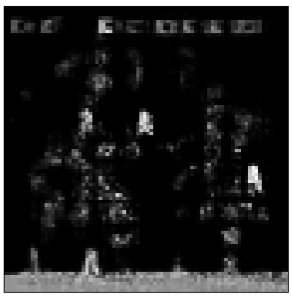


Figure 6.60: $\beta = 1$

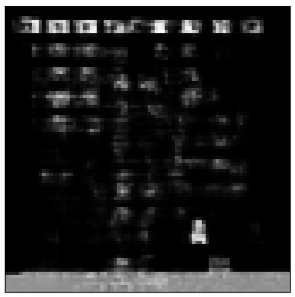


Figure 6.61: $\beta = 4$

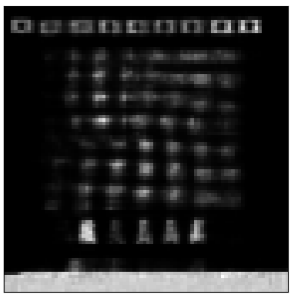


Figure 6.62: $\beta = 32$

Figure 6.63: **Multiple latent filter architecture with neuron-level redundancy reduction**. The best of 10 samples from the prior $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ for different values of β.

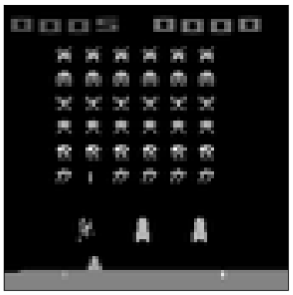


Figure 6.64: $\beta = 1$ (original)

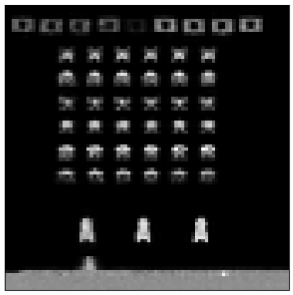


Figure 6.65: $\beta = 1$ (19 steps)

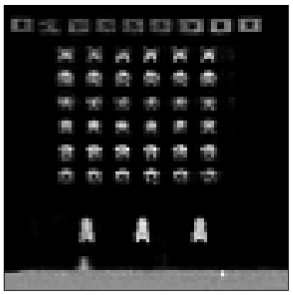


Figure 6.66: $\beta = 1$ (26 steps)

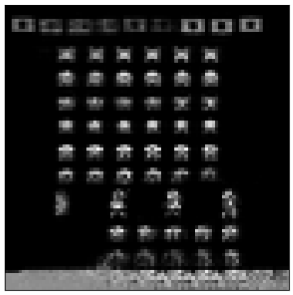


Figure 6.67: $\beta = 1$ (60 steps)

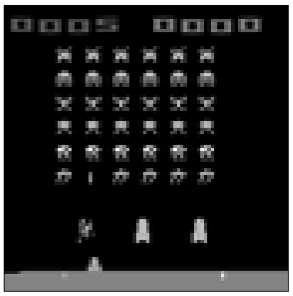


Figure 6.68: $\beta = 4$ (original)

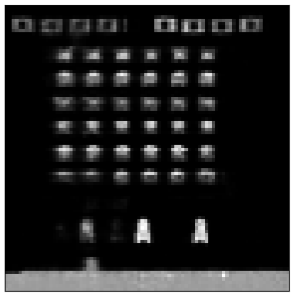


Figure 6.69: $\beta = 4$ (1 step)

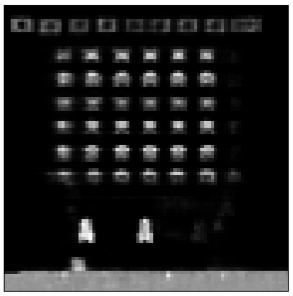


Figure 6.70: $\beta = 4$ (7 steps)

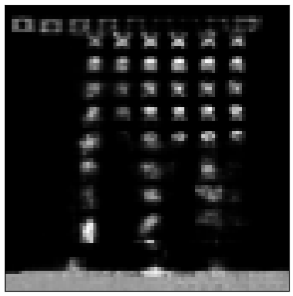


Figure 6.71: $\beta = 4$ (98 steps)

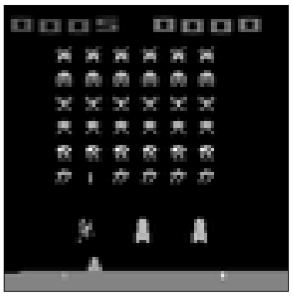


Figure 6.72: $\beta = 32$ (original)

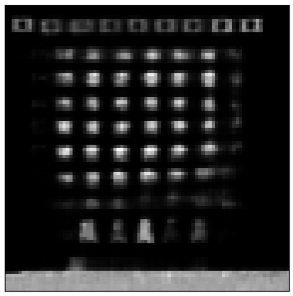


Figure 6.73: $\beta = 32$ (1 step)

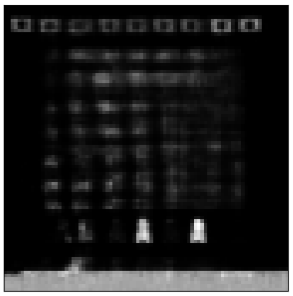


Figure 6.74: $\beta = 32$ (15 steps)

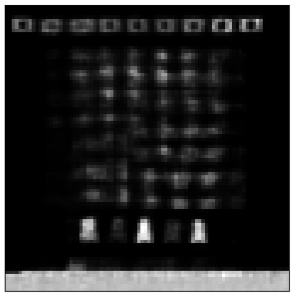


Figure 6.75: $\beta = 32$ (48 steps)

Figure 6.76: **Multiple latent filter architecture with neuron-level redundancy reduction**. A selection of Space Invader frames and the following samples from the unknown prior $\hat{p}(\boldsymbol{z})$ using MCMC for different values of β.

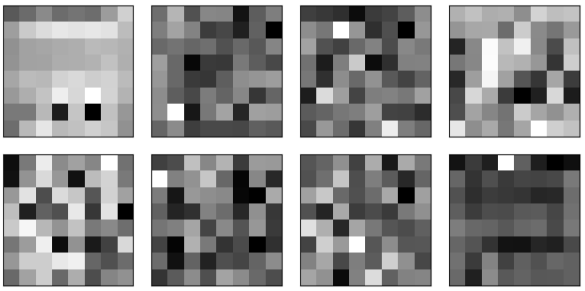


Figure 6.77: $\beta = 1$

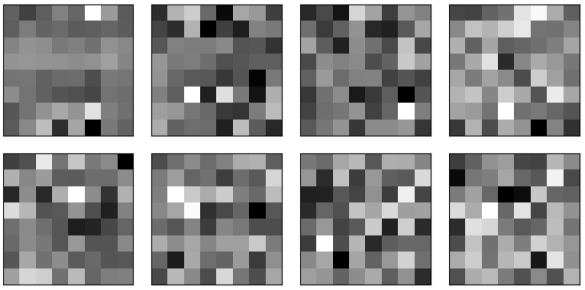


Figure 6.78: $\beta = 4$

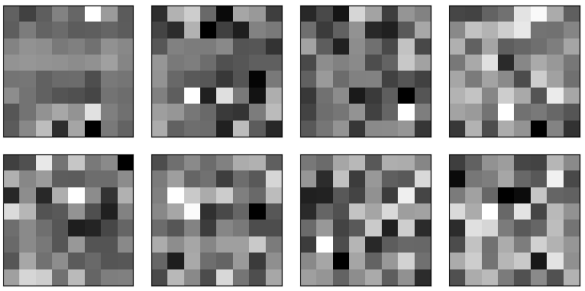


Figure 6.79: $\beta = 32$

Figure 6.80: **Multiple latent filter architecture with neuron-level redundancy reduction**. An average of the activations in the latent image over $10,000$ images from the test set for different values of β.

6.4 Naïve Filter-Level Redundancy Reduction

6.4.1 Results

The multiple latent filter architecture, found in Table (6.2), was trained for 10 epochs with the naïve filter-level redundancy reduction loss function, and the validation, validation KL and validation reconstruction loss recorded, which are shown in Figure (6.81). As with previous methods, the image reconstructions are near perfect even for high β, as shown in Figure (6.94).

Interestingly, in Figure (6.99), we see the beginnings of high-level objects in the latent filters! It's possible to make out the representation of the score bar for each value of β. We also see some very promising results on the average activations of the latent filters in Figure (6.120). The score bar and ground are recognised in at least one filter, with sometimes noticeable activations on the barriers. The value of β does not have a noticeable influence on the latent filters.

Further, we see that the samples from the prior $p(\boldsymbol{z})$ are far from resembling the original scene, but samples from the unknown distribution $\hat{p}(\boldsymbol{z})$ are very realistic for high β! We expect poor samples from the prior as we pressure $\bar{\boldsymbol{\mu}}$ and $\bar{\boldsymbol{\sigma}}$ to be close to $p(\boldsymbol{z})$, not $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ as before.

6.4.2 Summary

- As with the previous methods, the naïve filter-level redundancy reduction method near perfectly reconstructs its inputs irrespective of the value of β.
- The method achieves the first signs of latent filter representations of high-level objects in the scene. This is especially obvious in the average activations of latent filters, where the score, floor and barriers are clearly recognised.
- However, the generative power of this method is quite strong. Samples from the unknown distribution $\hat{p}(\boldsymbol{z})$ resemble the original scene after a threshold value of β.
- We therefore have a method that learns persistent high-level features and may generate realistic novel samples.

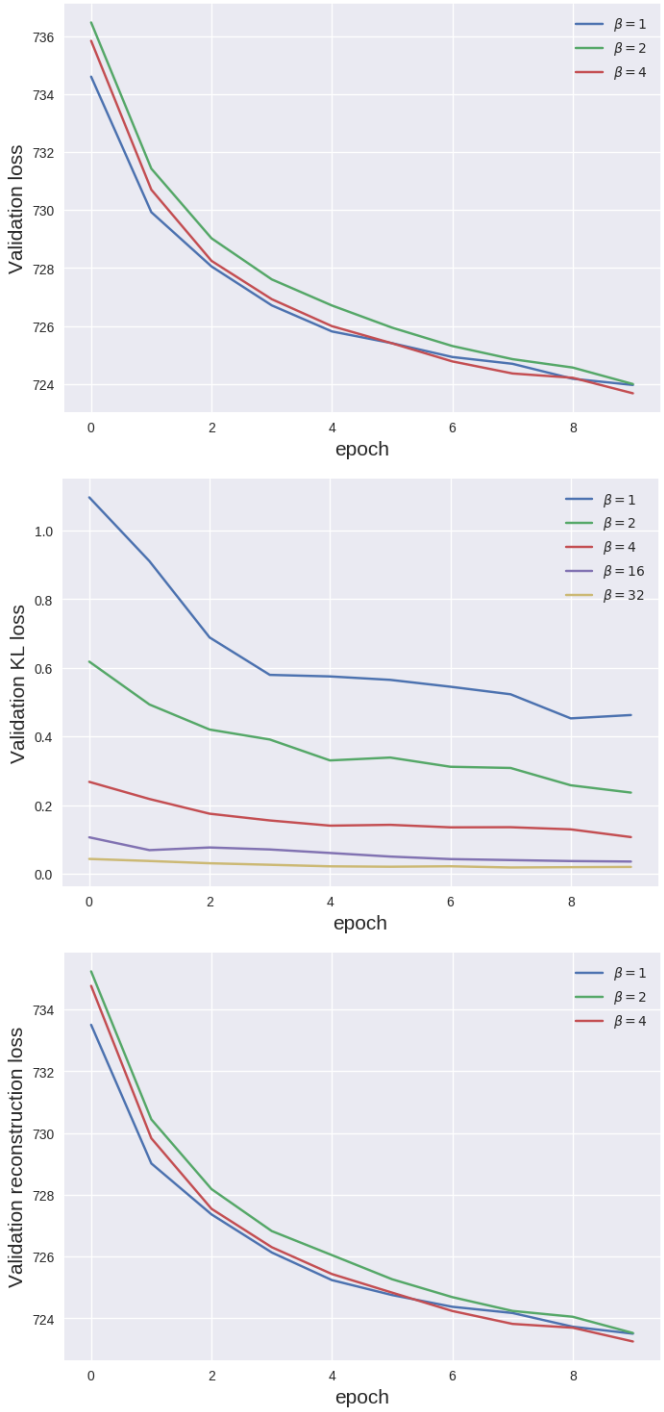


Figure 6.81: **Multiple latent filter architecture with naïve filter-level redundancy reduction**. The validation, validation KL and validation reconstruction loss for the latent image architecture and different values of β. The validation reconstruction loss for $\beta = 16, 32$ were ~ 950, and are therefore excluded in two plots for readability.

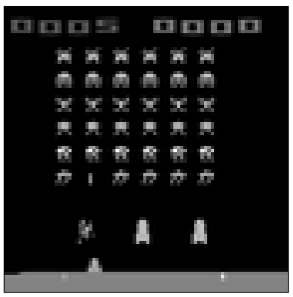


Figure 6.82: Original

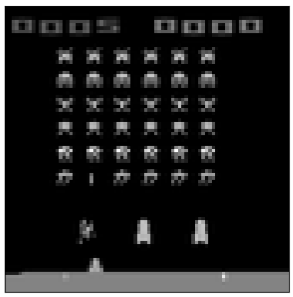


Figure 6.83: $\beta = 1$

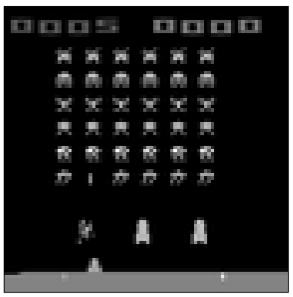


Figure 6.84: $\beta = 2$

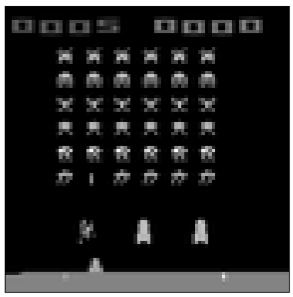


Figure 6.85: $\beta = 32$

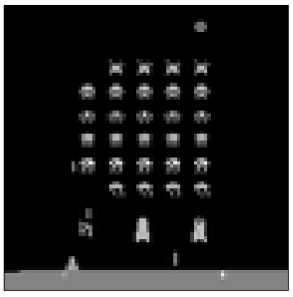


Figure 6.86: Original

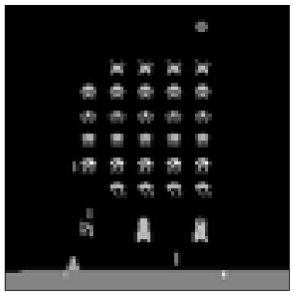


Figure 6.87: $\beta = 1$

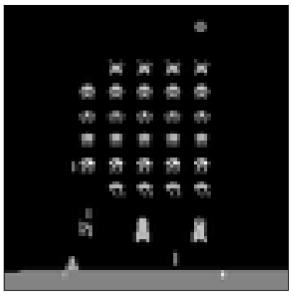


Figure 6.88: $\beta = 2$

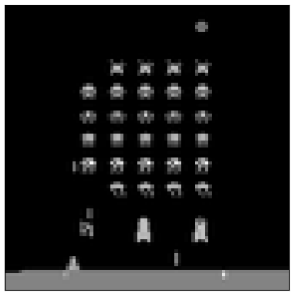


Figure 6.89: $\beta = 4$

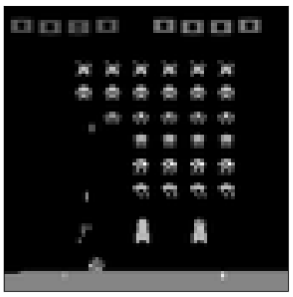


Figure 6.90: Original

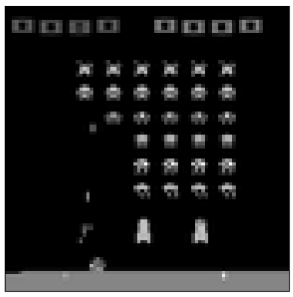


Figure 6.91: $\beta = 1$

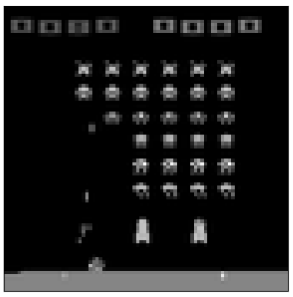


Figure 6.92: $\beta = 2$

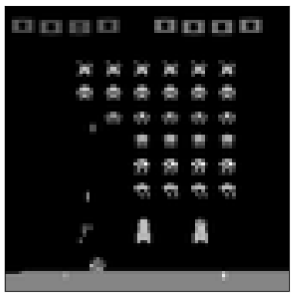


Figure 6.93: $\beta = 4$

Figure 6.94: **Multiple latent filter architecture with naïve filter-level redundancy reduction**. A selection of Space Invader frames and their corresponding reconstructions for different values of β.

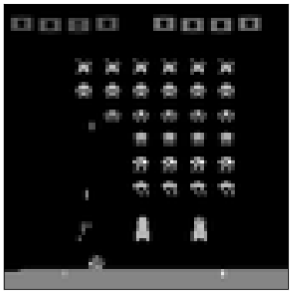


Figure 6.95: Original

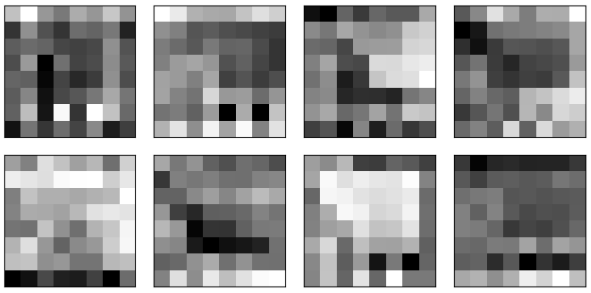


Figure 6.96: $\beta = 1$

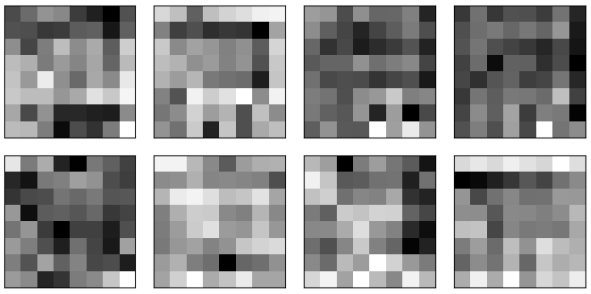


Figure 6.97: $\beta = 4$

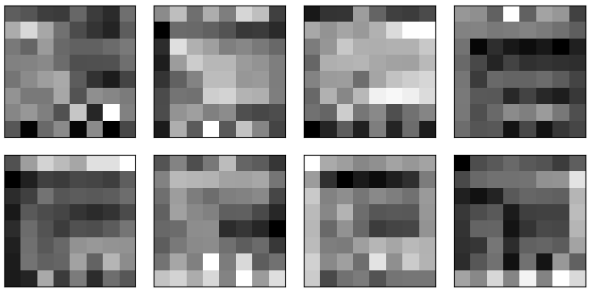


Figure 6.98: $\beta = 32$

Figure 6.99: **Multiple latent filter architecture with naïve filter-level redundancy reduction**. A Space Invaders frame and the activations over its corresponding latent filters for different values of β.

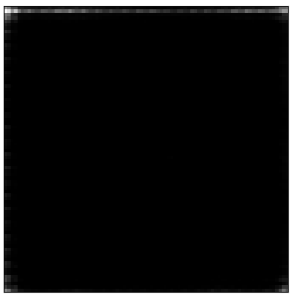


Figure 6.100: $\beta = 1$

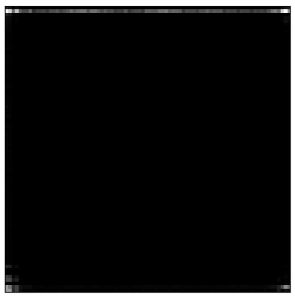


Figure 6.101: $\beta = 4$

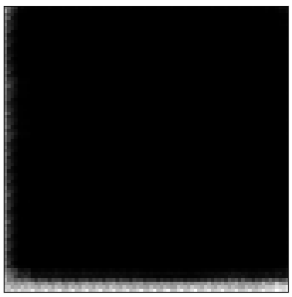


Figure 6.102: $\beta = 32$

Figure 6.103: **Multiple latent filter architecture with naïve filter-level redundancy reduction**. The best of 10 samples from the prior $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ for different values of β.

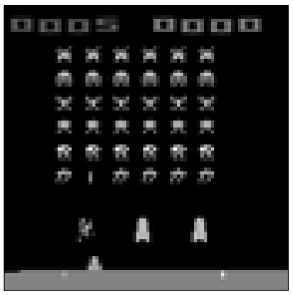


Figure 6.104: $\beta = 1$ (original)

Figure 6.105: $\beta = 1$ (1 step)

Figure 6.106: $\beta = 1$ (5 steps)

Figure 6.107: $\beta = 1$ (10 steps)

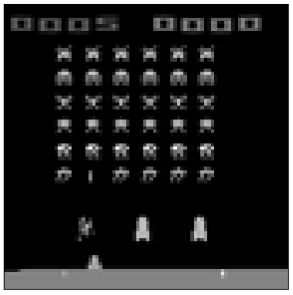


Figure 6.108: $\beta = 4$ (original)

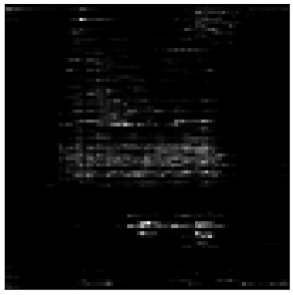


Figure 6.109: $\beta = 4$ (1 step)

Figure 6.110: $\beta = 4$ (5 steps)

Figure 6.111: $\beta = 4$ (10 steps)

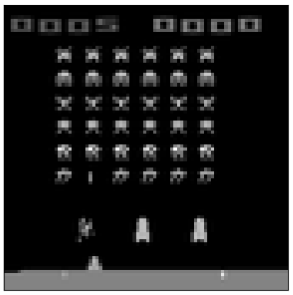


Figure 6.112: $\beta = 32$ (original)

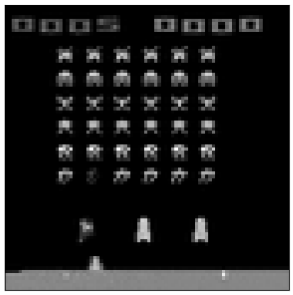


Figure 6.113: $\beta = 32$ (1 step)

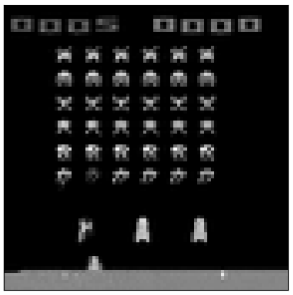


Figure 6.114: $\beta = 32$ (5 steps)

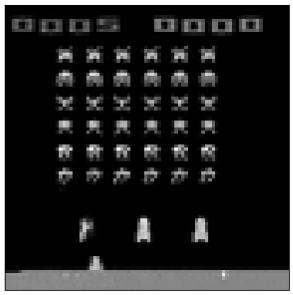


Figure 6.115: $\beta = 32$ (10 steps)

Figure 6.116: **Multiple latent filter architecture with naïve filter-level redundancy reduction**. A selection of Space Invader frames and the following samples from the unknown prior $\hat{p}(\boldsymbol{z})$ using MCMC for different values of β.

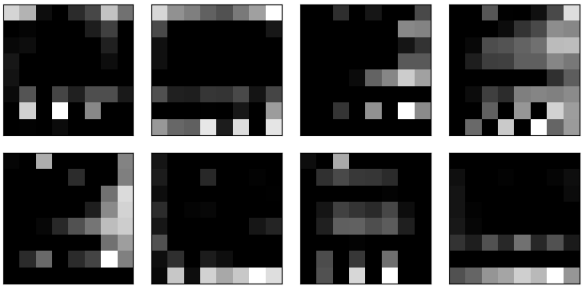


Figure 6.117: $\beta = 1$

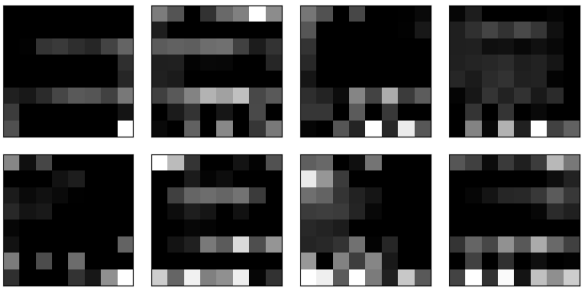


Figure 6.118: $\beta = 4$

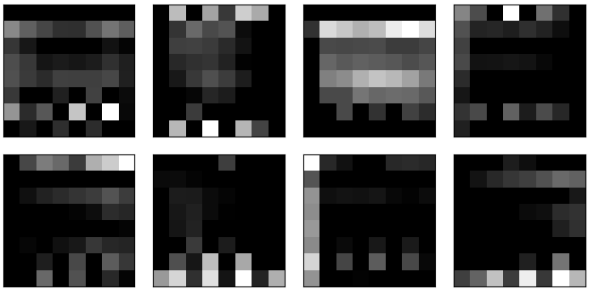


Figure 6.119: $\beta = 32$

Figure 6.120: **Multiple latent filter architecture with naïve filter-level redundancy reduction**. An average of the activations in the latent image over $10,000$ images from the test set for different values of β.

6.5 Weighted Filter-Level Redundancy Reduction

6.5.1 Results

The multiple latent filter architecture, found in Table (6.2), was trained for 10 epochs with the naïve filter-level redundancy reduction loss function, and the validation, validation KL and validation reconstruction loss recorded, which are shown in Figure (6.121). Despite the large weight on the KL loss term, the image reconstructions are near perfect for all β, as shown in Figure (6.134).

High-level objects are clearly represented in the latent filters, particularly for $\beta = 4$. The barriers and score are again clearly identified in the latent filters. This does not seem to be the case with $\beta = 32$, which suggests there is an optimal value of β. Further, the average activations of the latent filters in Figure (6.156) show that, on average, $\beta = 4$ and $\beta = 32$ both detect persistent high-level objects in the scene.

When comparing the input to the corresponding latent filters, we find an interesting result. We can easily spot the barriers and score for $\beta = 4$, but not so for $\beta = 32$, as shown in Figure (6.139). Further, the samples from the unknown distribution $\hat{p}(\boldsymbol{z})$ are much more interesting for $\beta = 4$ than $\beta = 32$. For $\beta = 32$, the samples do not differ very much from its input, whereas samples for $\beta = 4$ differ widely (bullets disappear and scores change), but in a realistic consistent way. These results are shown in Figure (6.187).

6.5.2 Summary

- The method is capable of near perfectly reconstructing the original image, despite having a large multiplicative factor on the KL loss term.
- Latent representations of persistent high-level objects in the scene are clear for some β.
- Further, values of β also have a noticeable effect on the generated samples from the unknown distribution $\hat{p}(\boldsymbol{z})$.

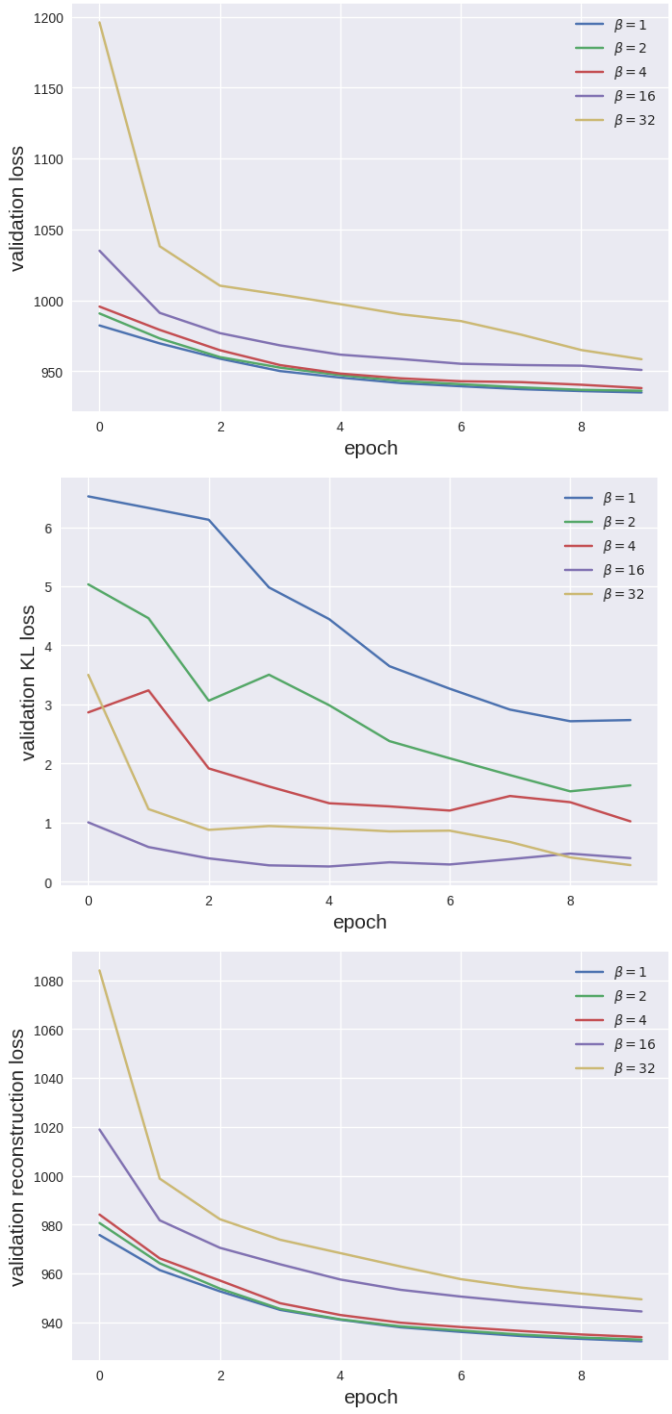


Figure 6.121: **Multiple latent filter architecture with weighted filter-level redundancy reduction**. The validation, validation KL and validation reconstruction loss for the latent image architecture and different values of β.

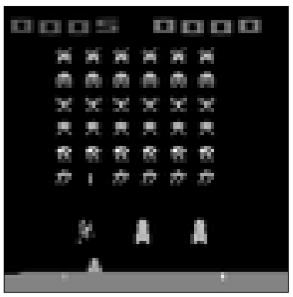


Figure 6.122: Original

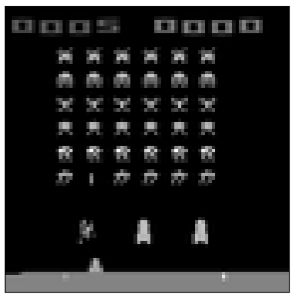


Figure 6.123: $\beta = 1$

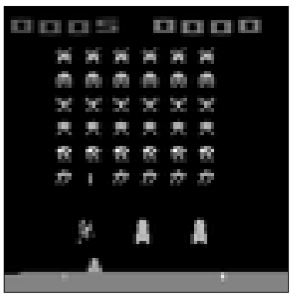


Figure 6.124: $\beta = 4$

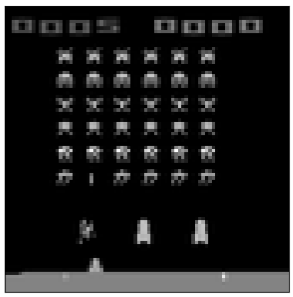


Figure 6.125: $\beta = 32$

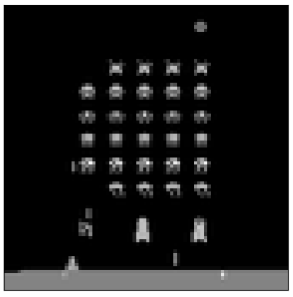


Figure 6.126: Original

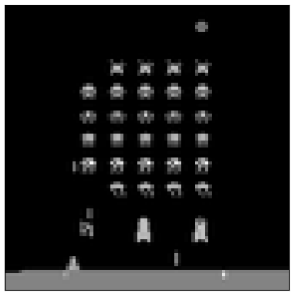


Figure 6.127: $\beta = 1$

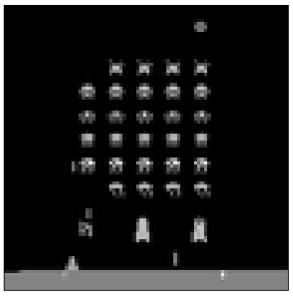


Figure 6.128: $\beta = 4$

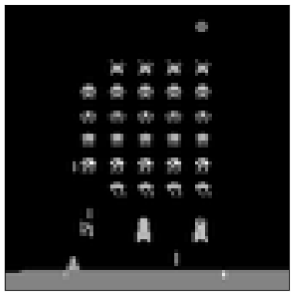


Figure 6.129: $\beta = 32$

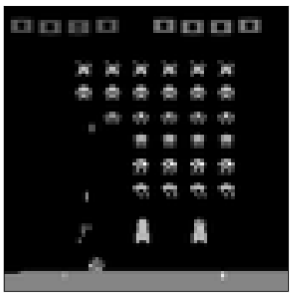


Figure 6.130: Original

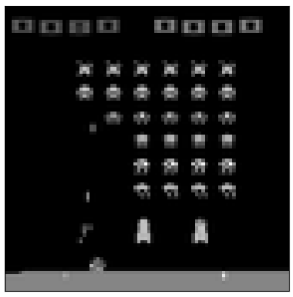


Figure 6.131: $\beta = 1$

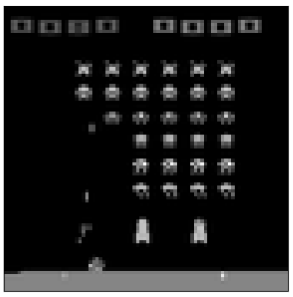


Figure 6.132: $\beta = 4$

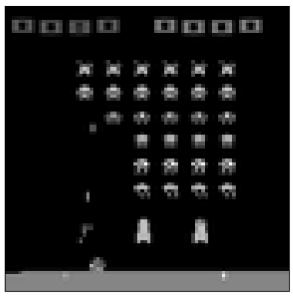


Figure 6.133: $\beta = 32$

Figure 6.134: **Multiple latent filter architecture with weighted filter-level redundancy reduction**. A selection of Space Invader frames and their corresponding reconstructions for different values of β.

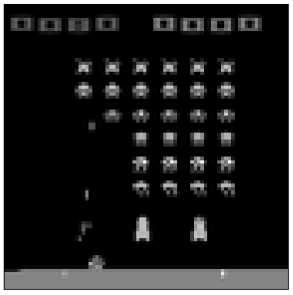


Figure 6.135: Original

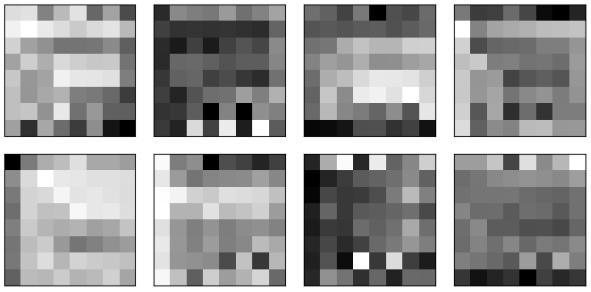


Figure 6.136: $\beta = 1$

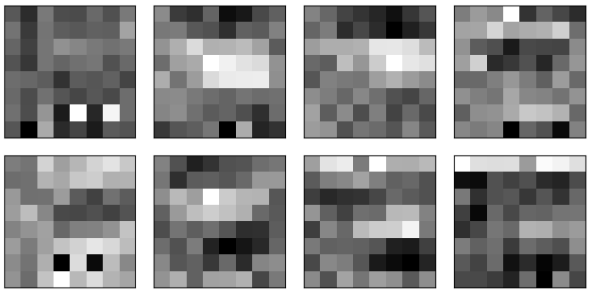


Figure 6.137: $\beta = 4$

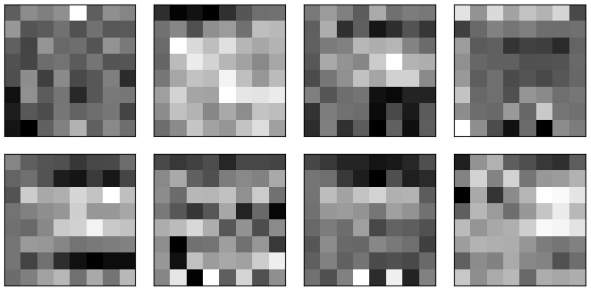


Figure 6.138: $\beta = 32$

Figure 6.139: **Multiple latent filter architecture with weighted filter-level redundancy reduction**. A Space Invaders frame and the activations over its corresponding latent filters for different values of β.

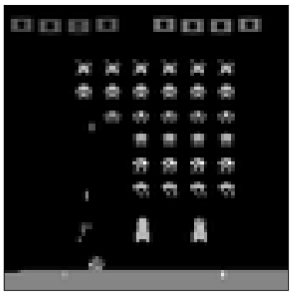


Figure 6.140: $\beta = 1$ (original)

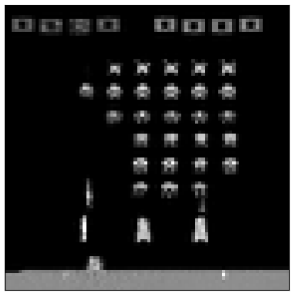


Figure 6.141: $\beta = 1$ (18 steps)

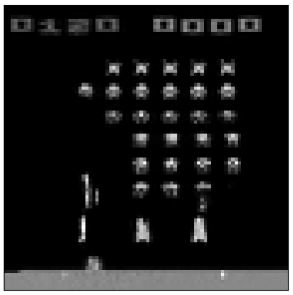


Figure 6.142: $\beta = 1$ (26 steps)

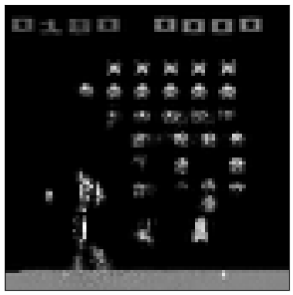


Figure 6.143: $\beta = 1$ (39 steps)

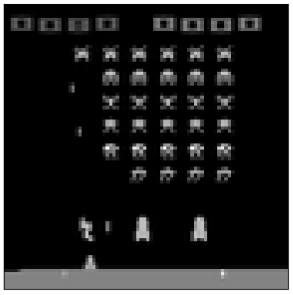


Figure 6.144: $\beta = 4$ (original)

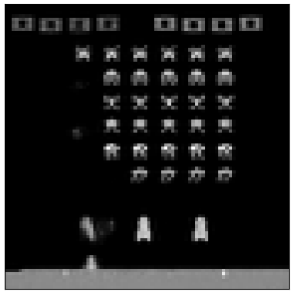


Figure 6.145: $\beta = 4$ (1 step)

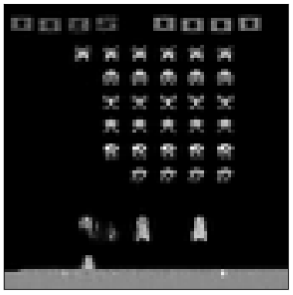


Figure 6.146: $\beta = 4$ (5 steps)

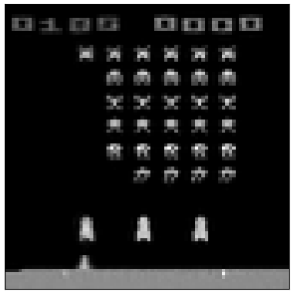


Figure 6.147: $\beta = 4$ (40 steps)

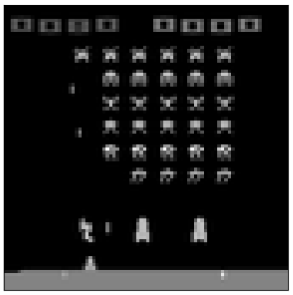


Figure 6.148: $\beta = 32$ (original)

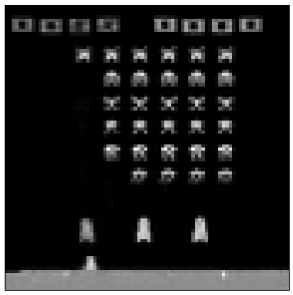


Figure 6.149: $\beta = 32$ (1 step)

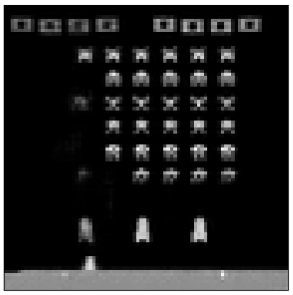


Figure 6.150: $\beta = 32$ (47 steps)

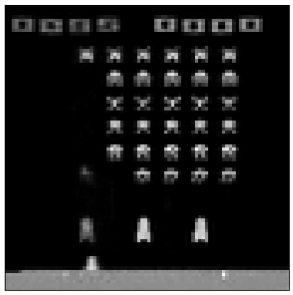


Figure 6.151: $\beta = 32$ (62 steps)

Figure 6.152: **Multiple latent filter architecture with weighted filter-level redundancy reduction**. An average of the activations in the latent image over $10,000$ images from the test set for different values of β.

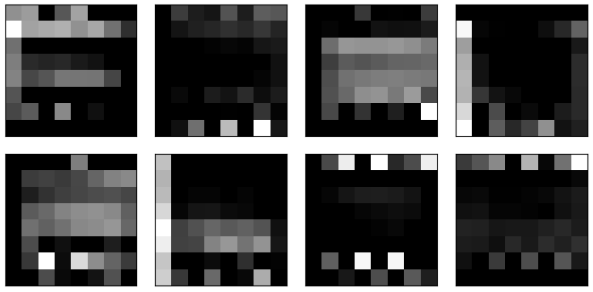


Figure 6.153: $\beta = 1$

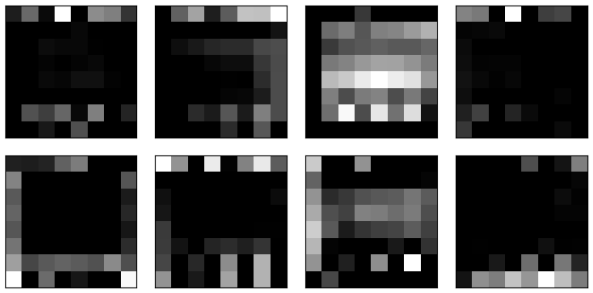


Figure 6.154: $\beta = 2$

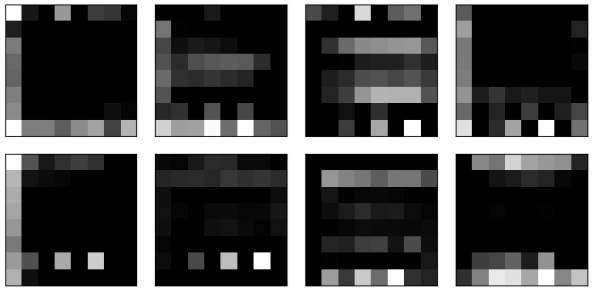


Figure 6.155: $\beta = 4$

Figure 6.156: **Multiple latent filter architecture with weighted filter-level redundancy reduction**. An average of the activations in the latent image over $10,000$ images from the test set for different values of β.

6.6 Separating Colour Spaces

6.6.1 Results

The multiple latent filter RGB architecture, found in Table (6.3), was trained for 10 epochs, and the validation, validation KL and validation reconstruction loss recorded.

6.6.2 Summary

- Consider two architectures: shallow and deep
- Consider Pong, Space Invaders (in progress) and Breakout
- Plot reconstruction loss and KL divergence
- Show reconstructions and convolutional layers of each
- Show mean activations of filters
- Alter latent space variable and show reconstruction
- Add noise to filters

6.7 The No Free Lunch Relationship Between Reconstruction Loss and KL Divergence

- Vary β and plot BCE and KL
- Show example images for different β
- Consider Space Invaders, Pong and Breakout

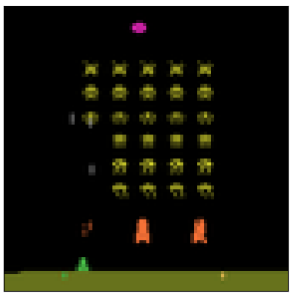


Figure 6.157: Original

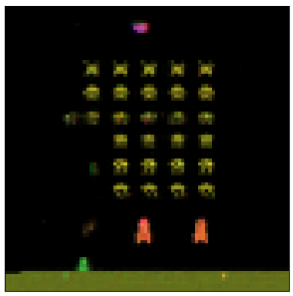


Figure 6.158: $\beta = 1$

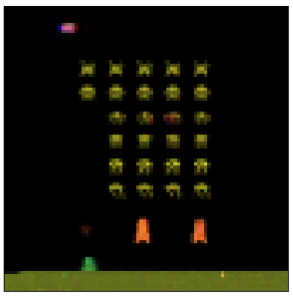


Figure 6.159: $\beta = 2$

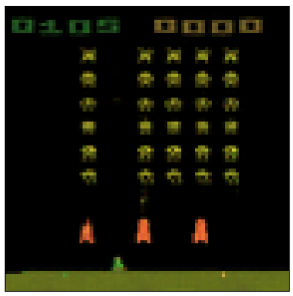


Figure 6.160: $\beta = 4$

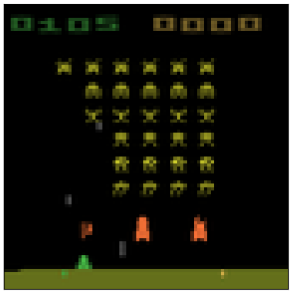


Figure 6.161: Original

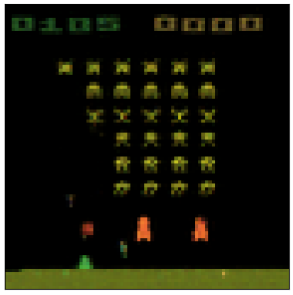


Figure 6.162: $\beta = 1$

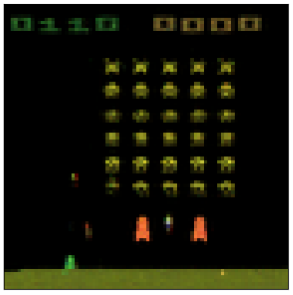


Figure 6.163: $\beta = 2$

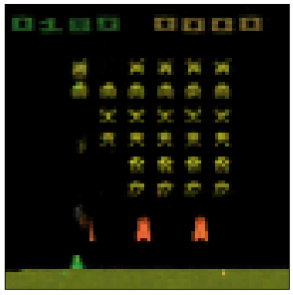


Figure 6.164: $\beta = 4$

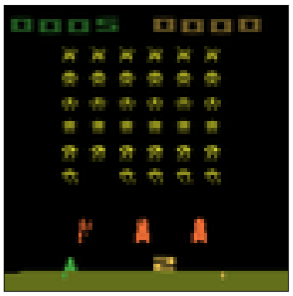


Figure 6.165: Original

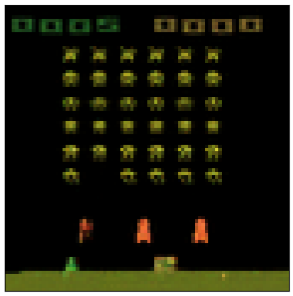


Figure 6.166: $\beta = 1$

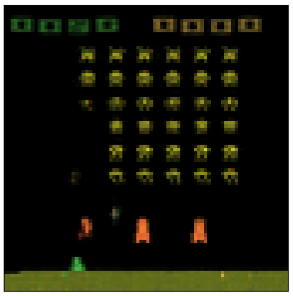


Figure 6.167: $\beta = 2$

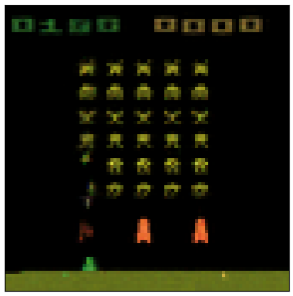


Figure 6.168: $\beta = 4$

Figure 6.169: A selection of Space Invader frames and their corresponding reconstructions for different values of β.

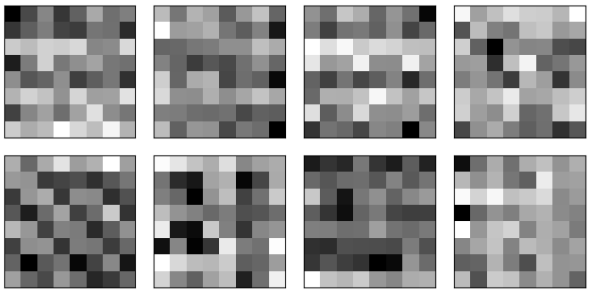


Figure 6.171: $\beta = 1$

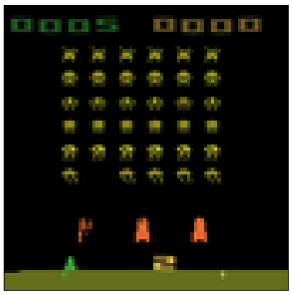


Figure 6.170: Original

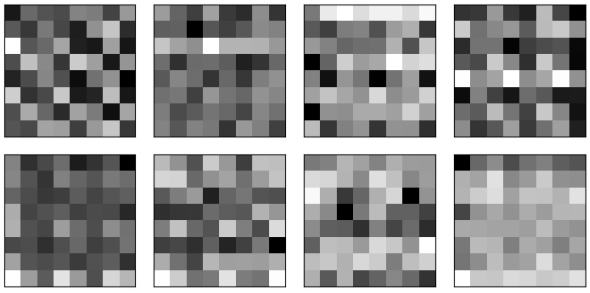


Figure 6.172: $\beta = 2$

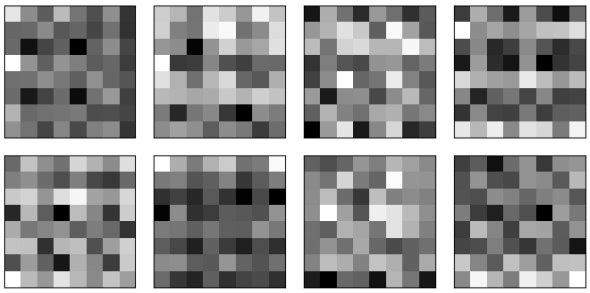


Figure 6.173: $\beta = 4$

Figure 6.174: A Space Invaders frame and the activations over its corresponding latent filters for different values of β.

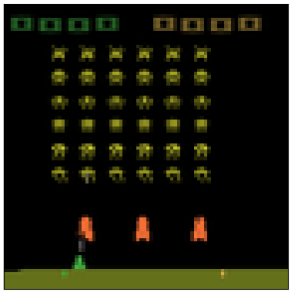


Figure 6.175: $\beta = 1$ (original)

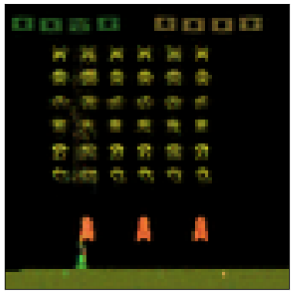


Figure 6.176: $\beta = 1$ (7 steps)

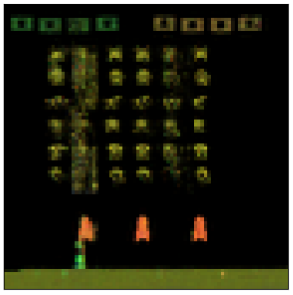


Figure 6.177: $\beta = 1$ (17 steps)

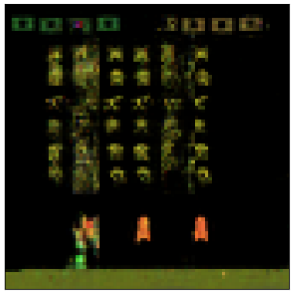


Figure 6.178: $\beta = 1$ (18 steps)

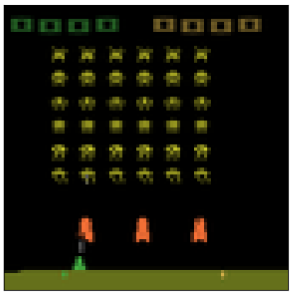


Figure 6.179: $\beta = 2$ (original)

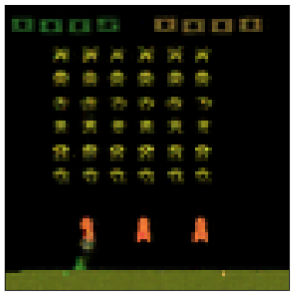


Figure 6.180: $\beta = 2$ (10 steps)

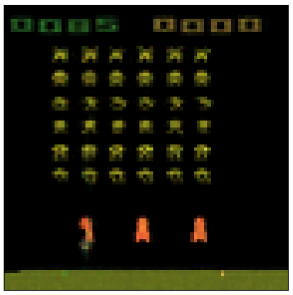


Figure 6.181: $\beta = 2$ (17 steps)

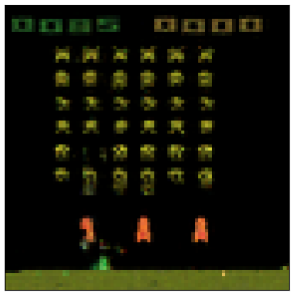


Figure 6.182: $\beta = 2$ (31 steps)

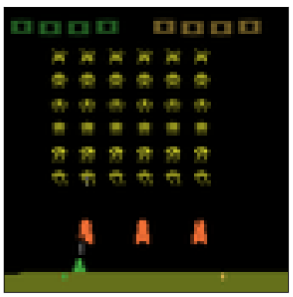


Figure 6.183: $\beta = 4$ (original)

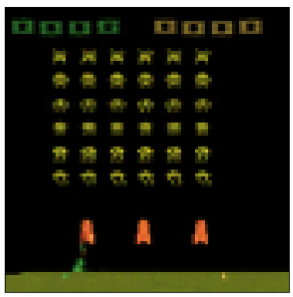


Figure 6.184: $\beta = 4$ (1 step)

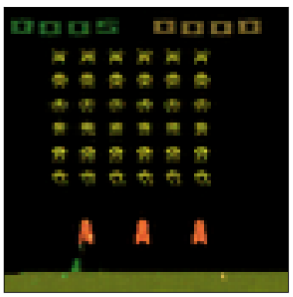


Figure 6.185: $\beta = 4$ (3 steps)

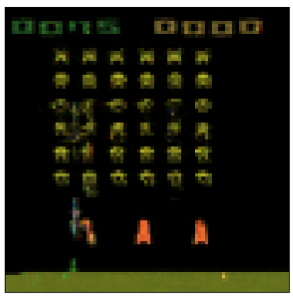


Figure 6.186: $\beta = 4$ (20 steps)

Figure 6.187: An average of the activations in the latent image over $10,000$ images from the test set for different values of β.

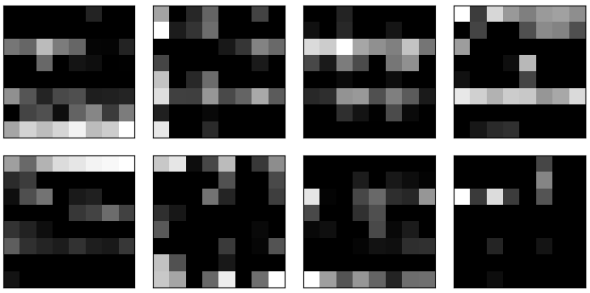


Figure 6.188: $\beta = 1$

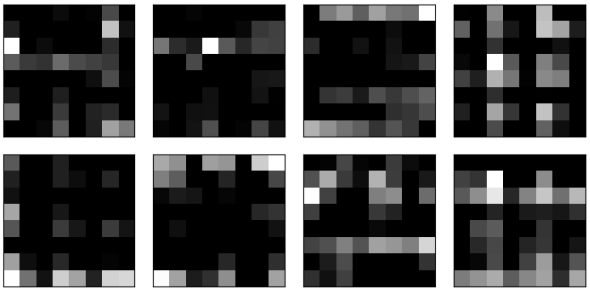


Figure 6.189: $\beta = 2$

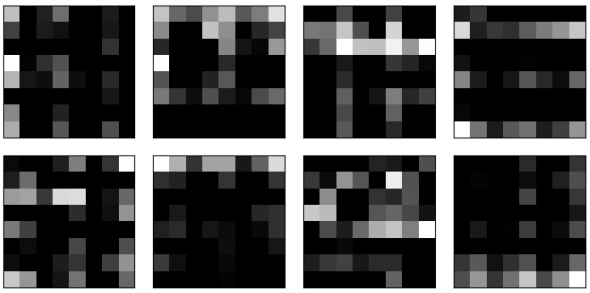


Figure 6.190: $\beta = 4$

Figure 6.191: An average of the activations in the latent image over $10,000$ images from the test set for different values of β.

Chapter 7

Conclusion

7.1 Summary of Thesis Achievements

This thesis has made small novel steps in solving the open problem of the scalable unsupervised extraction of an object's type and location using fully-convolutional variational autoencoders. We have:

- **Proposed a novel architecture**: the fully-convolutional variational autoencoder. This is an exciting first step in the development of the preservation of spatial information in learning representations of objects in a scene.
- **Proposed a number of novel methods** to solve the low-level extraction of objects and their location in fully-convolutional variational autoencoders. Namely, this was the proposal of the single latent filter architecture, the multiple latent filter architecture, neuron-level redundancy reduction, naïve filter-level redundancy reduction and weighted filter-level redundancy reduction.
- **Collected experimental evidence for each proposed method.** Each method's capabilities were tested, including how well they reconstructed their input and if the latent space learnt an interpretable encoding of the original scene.

- **Assessed the feasibility of using fully-convolutional variational autoencoders to advance DSRL.** After examining the experimental data collected, we can confidently say that we have made a small novel step towards solving the problem of scalable unsupervised extraction of an object's type and location. This is achieved with a multiple latent filter architecture using weighted filter-level redundancy reduction.

7.2 Future Work

There are two very promising avenues to explore from here.

The first is a step away from the variational autoencoder and back to a deterministic fully-convolutional autoencoder with a regularization technique called *OrthoReg*, introduced in *Regularizing CNNs with Locally Constrained Decorrelations* [25]. This regularization technique reduces redundancy among the latent filters, which is precisely what we were trying to achieve with the filter-level redundancy techniques developed.

The second is the exploration of the Winner Takes All method. Although this method was originally thought to be equivalent to neuron-level redundancy reduction, it was realised too late that this was not so. Given more time, it would be interesting to see if this method works as intended.

Bibliography

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 2015-Janua, pages 4148–4152, 2015.

[2] D. M. Blei. Variational Inference, 2011.

[3] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational Inference: A Review for Statisticians. jan 2016.

[4] K. Burnham and D. Anderson. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach (2nd ed)*, volume 172. 2002.

[5] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. *arXiv:1606.03657 [cs.LG]*, pages 1–14, 2016.

[6] F. Chollet. Building Autoencoders in Keras, 2016.

[7] T. S. Cohen and M. Welling. Transformation Properties of Learned Visual Representations. dec 2014.

[8] A. Creswell, K. Arulkumaran, and A. A. Bharath. Improving Sampling from Generative Autoencoders with Markov Chains. oct 2016.

[9] B. Darrach. Meet Shakey, the First Electronic Person. *Life*, pages 58–68, 1970.

[10] G. Desjardins, A. Courville, and Y. Bengio. Disentangling Factors of Variation via Generative Entangling. oct 2012.

[11] C. Doersch. Tutorial on Variational Autoencoders. *arXiv*, pages 1–23, 2016.

[12] I. Dykeman. Conditional Variational Autoencoders, 2016.

[13] M. Garnelo, K. Arulkumaran, and M. Shanahan. Towards Deep Symbolic Reinforcement Learning. *arXiv Preprint*, pages 1–13, 2016.

[14] I. Higgins, L. Matthey, X. Glorot, A. Pal, B. Uria, C. Blundell, S. Mohamed, and A. Lerchner. Early Visual Concept Learning with Unsupervised Deep Learning. *arXiv*, 2016.

[15] M. Hutter. *Universal Artificial Intelligence*, volume 1. 2005.

[16] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *Iclr*, (Ml):1–14, 2014.

[17] W. Knight. AI Winter Isn't Coming. *MIT Technology Review*, 2016.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012.

[19] F.-F. S. U. Li, J. S. U. Johnson, and S. S. U. Yeung. Convolutional Neural Networks for Visual Recognition, 2016.

[20] C. Y. Liou, J. C. Huang, and W. C. Yang. Modeling word perception using the Elman network. In *Neurocomputing*, volume 71, pages 3150–3157, 2008.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[22] D. Ormerod. Lee Sedol plays the first move of game three against AlphaGo, 2016.

[23] E. U. o. T. Reingold. PSY371F Higher Cognitive Processes, 2001.

[24] S. R. Richter, V. Vineet, S. Roth, and V. Koltun. Playing for data: Ground truth from computer games. In *Lecture Notes in Computer Science (including subseries Lecture Notes*

in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 9906 LNCS, pages 102–118, 2016.

[25] P. Rodríguez, J. Gonzàlez, G. Cucurull, J. M. Gonfaus, and X. Roca. Regularizing CNNs with Locally Constrained Decorrelations. nov 2016.

[26] R. Rosen. IBM's Deep Blue vs Gary Kasparov, 2012.

[27] J. Schmidhuber. Learning Factorial Codes by Predictability Minimization. *Neural Computation*, 4(6):863–879, 1992.

[28] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[29] M. Stokes, M. Anderson, S. Chandrasekar, and R. Motta. A Standard Default Color Space for the Internet - sRGB, 1996.

[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 07-12-June, pages 1–9, 2015.

[31] Y. Tang and G. Hinton. Tensor Analyzers. *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 28, 2013.

[32] B. G. Thiagarajan, A. Member, and G. Z. Voyiadjis. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. *ICLR'17 submission*, (July):1–13, 2016.

[33] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8689 LNCS, pages 818–833, 2014.

[34] K. Zsolnai-Fehér. Computer Games Empower Deep Learning Research, 2016.