Looking for Livable Cities with Deep Learning

Supervisor: Dr. Sarah Clinch

Tian Breznik

A third year report submitted for the degree of
BSc Computer Science and Mathematics

School of Computer Science
University of Manchester
April, 2021

Abstract

This report explores the interdisciplinary space of research on Urban Analytics with a focus on using deep learning algorithms for image data compression and subsequent analysis, specifically considering disentangled variations of the variational autoencoder (VAE) architecture such as BetaVAE and JointVAE.

Our aim was to replicate these algorithms and train the models on a custom generated dataset of urban form maps, which visually encode characteristics of the urban environment relevant to livability, such as urban greenery, density of the built environment and the walkable paths.

Our models successfully reconstructed original urban form images, preserving general spatial information, but blurring out the details. We show a number of interesting techniques for visually exploring and interpreting the latent space learned by the models. Most of all, our results highlight the structural complexity of our generated dataset.

Acknowledgements

I would like to thank my supervisor Dr. Sarah Clinch for her consistent support, discussion and encouragement throughout this project. I would also like to thank Dr. Andre Freitas and his Phd student Giangiacomo Mercatali for their knowledge, experience and excitement about collaboration.

Contents

Chapter 1

Introduction

This project presents an application of deep learning methods to an urban planning problem of increasing contemporary importance, which is *livable cities through the lens of urban forms*[1]. Specifically, our focus is on:

- Applying novel deep learning methods to the problem of planning livable cities.
- Discussing our results, in relation to our application of deep learning and to the relevance of our results to real urban planning issues.
- Presenting the results in an interactive webpage, encouraging the exploration of the analyzed dataset.

Though some related works will be referenced in our discussion, urban planning theory on livable cities will not be discussed at length.

1.1 Key Terminology: Livable Cities and Urban Form

According to the United Nations, 68% of the world's population will live in urban areas by the year 2050 (United Nations Department of Economic and Social Affairs, 2018). Projections show that urbanization, the gradual shift in residence of the human population from rural to urban areas, combined with the overall growth of the world's population, could add another 2.5 billion people to urban areas by 2050 (UN DESA | United Nations Department of Economic and Social Affairs, 2018). With such a proportion of human development on both the societal and individual level taking place in urban areas, we need to carefully consider how these environments might shape our future and the future of the planet. With the existential threat of climate change looming over us, we will need to start incorporating significant changes into our lives to move towards a more sustainable future. Some of these changes will have to happen in cities and towns on a massive scale, whilst considering the inherent complexity of those structures.

The term *complexity* refers to the higher-order phenomena arising from a system's many connected, interacting subcomponents and describes both dynamics (i.e. processes) and structure (i.e. patterns and configurations) (Batty, 2005). Human societies and hence the environments we build are examples of dynamic complex ecosystems which are constantly evolving and reshaping in response to various pressures. The resulting physical patterns construe the urban form and can be studied in terms of network character, fractal structure, diversity (of various sorts), and entropy. At higher levels of abstraction, we can analyse the resilience, robustness, and adaptive capacity of urban complex systems and how they respond to perturbation given their spatial patterns, structure, connectedness, and efficiency (Geoff Boeing, 2018).

[1] See Section 1.1 for an explanation of these terms.

Urban form are the physical characteristics that make up built areas, including the shape, size, density, and configuration of settlements. It can be considered at different scales: from regional, to urban, neighbourhood, 'block' and street (Urban form and infrastructure: a morphological review, n.d.). Importantly, experiences and observations of our existing urban environments (i.e. urban form) already demonstrate a strong need for careful urban planning and/or design. For example:

- Hard urban surfaces and reduced shading can lead to heat island effects, with associated health risks such as heat exhaustion and heat stroke, especially impacting the elderly (Smith and Levermore, 2008; Carmona et al., 2010). Urban green space has a cooling effect that could mitigate some of these issues (Bowler et al., 2010).
- Survey-based approaches have identified that living in a green environment has positive associations with self-reported health, including the number of symptoms experienced in the last two weeks (de Vries et al., 2003).
- A higher perceived confinement of space and sense of intimacy (as measured by visual enclosure) is related to increased pedestrian activity (Yin and Wang, 2016).
- In fact, planning for compact and dense urban spaces has been broadly acknowledged as an effective strategy towards decreasing a city's overall carbon footprint while providing accessible amenities to urban dwellers (Senbel et al. 2014)
- Specific streetscape features such as the proportion of windows on the street, the proportion of active street frontage and the amount of street furniture also significantly impact pedestrian activity (Ewing et al., 2016). Such measures also impact the formation of urban cultures, like the car-centric United States or the urban cycling culture in the Netherlands.

In this project, we analyze the urban form maps of international cities and towns (see Figure 1 for an example). We create information-dense vector-based representations of these maps that capture the complex structures of the image in significantly fewer dimensions than the image itself. The reduced dimensionality of these representations makes them ideal for a comparative analysis to find similarities and differences between the form of urban environments around the world. Throughout this report, we refer to these representations as **urban vectors**.

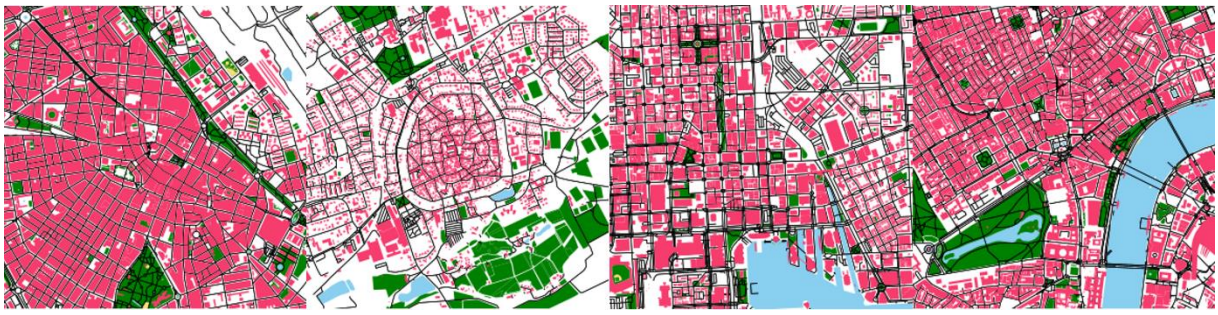


Figure 1. Examples of urban form maps generated. Cities shown left to right: Albacete, Bad Windsheim, Portland, London.

Our research considers not only the street network, but also the building footprints, waterways, urban greenery and the walkable street network (Figure 1). Including these expands the feature space to urban planning metrics such the presence of green space, building density/compactness, the circuity[2] of the walkable street network and many others, all of which contribute to the interpretability of our results in terms of livability factors. **Livability** is a broad and abstract term, which encompasses many different aspects of urban living. To put it simply, a livable city is one that meets the conditions in which social, economic, physical, and environmental requirements are fulfilled to provide the long-term comfort and wellbeing of the community (Bérenger and Verdier-Chouchane, 2007). Though urban form serves as an abstract object of discussion, it is nevertheless both a very real consequence and a driver of urban life

[2] Circuity, the ratio of network distances to straight-line distances, is an important measure of urban street network structure and transportation efficiency. (Geoff Boeing, 2017)

(Holanda 2013), and urban livability is in many ways dependent on urban form (Martino, Girling and Lu, 2021). The points listed above are only some examples on how many aspects of the built environment can impact urban living, illustrating that the design and planning of cities is not a negligible problem when combatting larger issues such as climate change and general human health.

1.2 Motivation

On a more personal note, this project was initially inspired by moving from Maribor, Slovenia, to Manchester, UK, which was expectedly accompanied with a conglomeration of emotions. Growing up in Maribor, a city filled with green spaces and surrounded by nature, my brain provided some resistance, fueled in part by homesickness, when taking on the task of making the gritty and industrial Manchester my home for three years. I began thinking about how much of my moods, good or bad, can be attributed to my immediate environment and the spaces I occupy. This went on for about two years and after answering some of my questions through traditional urban planning literature, I decided to take a computational approach to answering these questions. This is how the idea for this project came to me.

1.3 Aims and Objectives

Our aim with this research was to use deep learning algorithms on a dataset of 175,937 urban form maps and their associated urban vectors to observe which features emerge from our models once applied to the dataset. To put it more romantically, how will the model *understand* the complexities of the built environment.

To achieve this aim, this project set out to reach the following objectives:

- To review the current state of research on the topic of livability and urban forms
- To generate a dataset of cities and towns around the world
- To generate a dataset of their associated urban centre map images annotating different components of the urban environment.
- To build and train the model on the data
- To extract urban vectors from the model
- To conduct cluster analysis on the urban vectors
- To display the results in a web-based interactive visualisation

We hope to reveal interesting hidden connections between seemingly unrelated cities and towns. We also hope to explore how our findings relate to existing livability metrics and real city and town data.

Chapter 2

Background

In this section we first provide an overview of the main deep learning architectures studied and used in this research. We will start with a short introductive section on deep learning. Next, we will discuss the state of the current methods in and approaches to research in this very interdisciplinary field. Through this we will explain how and where our paper fits in this space, how we contribute to it and where our methods fall short in comparison to other research.

2.1 Deep Learning

Machine learning is a subset of the larger artificial intelligence family of techniques that enable computers to mimic human behavior. These techniques focus on teaching an algorithm to learn to perform certain tasks without being explicitly programmed. Traditionally these algorithms define a set of features in their data. These features are hand crafted and therefore time consuming, brittle and not scalable. Deep learning in turn is a subset of machine learning. Unlike in machine learning, these algorithms automatically extract useful patterns directly from raw data, using these patterns as the features to learn to perform a certain task. This process is akin to how the human brain functions. At the heart of it are neural networks, algorithms inspired by the biology of the interconnections between the neurons in our brains, hence the name. Though of course, neural networks have discrete layers, connections between neurons and directions of data propagation, whereas in a real brain a neuron can transmit information to any other neuron within its proximity.

Deep learning is a wide and growing field. According to Li Deng and Dong Yu in their book *Deep Learning Methods and Applications* (2014) the techniques and architectures can be generally categorized into the following three groups based on whether they are intended for synthesis or classification tasks:

- “Deep networks for unsupervised or generative learning”
- “Deep networks for supervised learning”
- “Hybrid deep networks”

In our research we work with unlabeled data, so we focus mostly on architectures which fall into the first category above, more specifically, variational autoencoders and some augmentations on the base algorithm capable of identifying disentangled features in the data.

2.1.1 Perceptron

The fundamental building block of every neural network is just a single neuron or node, also called a perceptron. It either received input from other nodes in the network or from an external source on which an operation is performed and a single numerical output is calculated. Each of the input numbers is multiplied by their corresponding weight in a dot product outputting the scalar. The magnitude of the weight depends on their relative importance. The perceptron takes this single value and passes it through a non-linear activation function, to produce $\hat{y}$, the final output of the node. This is the forward propagation process of the perceptron. A schematic representation is shown in (Figure 2). And in linear algebra terms it can be written as:

$$\hat{y} = g\ (w_0 + X^T W)$$

Where: $X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$, $W = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$ and w_0 is the bias term accompanying the input values.

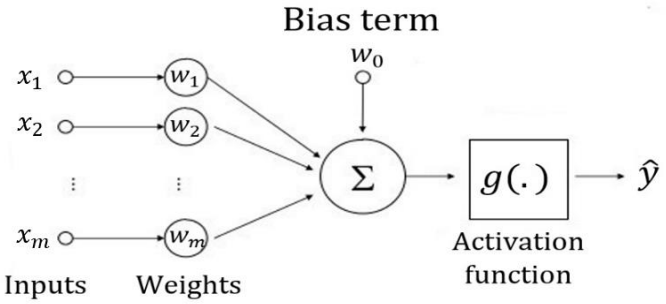
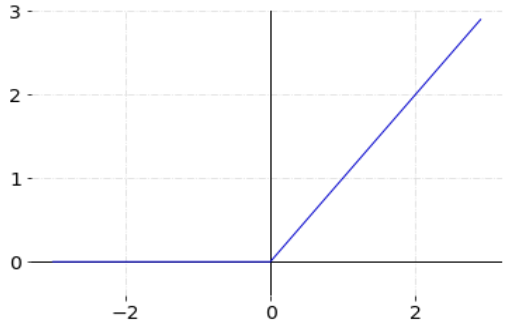


Figure 2: Perceptron

The role of the function g is to introduce nonlinearity into the output of the perceptron enabling it to deal with nonlinear data, which is what most real-world data is like. There are many activation functions, some of the most popular being the *Sigmoid Function*, the *Hyperbolic Tangent* and the *Rectified Linear Unit (ReLU)*. The architectures studied in our research used only the ReLU function (Figure 3), so it will be the only one we describe.



$$g(z) = \max(0, z)$$

With

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & otherwise \end{cases}$$

Figure 3: ReLU function

The main advantage of the ReLU function is that for negative input values the result becomes zero, which means that the node is not activated. This significantly reduces the number of active nodes and makes it a much more efficient function, as compared to sigmoid and the hyperbolic tangent functions.

2.1.2 Multilayer Perceptron (MLP)

A single perceptron by itself is just an evaluation of a nonlinear function of the sum of a bias term and a dot product of input and weight vectors. The true ability to learn complex features in the training data

comes from interconnecting the inputs and outputs of many perceptrons in complex multi-layered nets. One of the most basic such architectures is the multilayer perceptron (MLP). They are formed of at least three fully connected layers, in which each perceptron is connected to every other perceptron in the next layer. These layers are the input layer, the hidden layer, and the output layer.

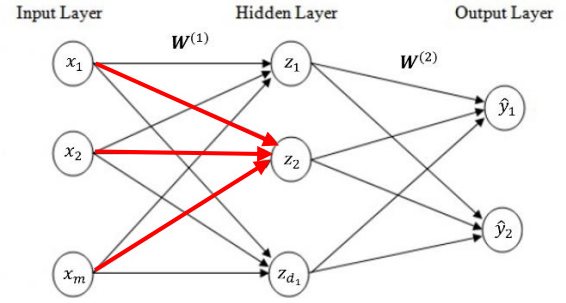


Figure 4: three-layer MLP

Perceptron i in the hidden layer (without activation function) has value $z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j W_{j,i}^{(1)}$, the activation function is applied in the calculation of the output $\hat{y}_i = g(w_{0,i}^{(2)} + \sum_{j=1}^{m} g(z_j) W_{j,i}^{(2)})$.

The red arrows highlight the exact same process that takes place in the perceptron pre-activation (activation function), i.e. $z_2 = W_{0,2}^{(1)} + \sum_{j=1}^{m} x_j W_{j,2}^{(1)}$.

Each layer i in the MLP has an associated learnable weight matrix $W^{(i)}$ and the column j of this matrix is the vector of weights for the perceptron j in layer $i + 1$. Deep neural networks are created by stacking layers to create more hierarchical architectures. During training, these matrices are updated in response to the accuracy of the outputted results through a process called backpropagation described in the appendix.

One of the disadvantages of MLPs is that it does not account for spatial information. It does not support matrix imputs, but rather inputs in a matrix form need to be flattened. This significantly increases the number of required input nodes and since the layers are fully connected, introduces redundancies in such a high number of parameters making the algorithm less efficient and more prone to overfitting. It also means that the semantic value regarding the spatial patterns is lost. These disadvantages are solved by Convolutional Neural Networks.

2.1.3 Convolutional Neural Networks (CNNs)

Similarly to MLPs, Convolutional Neural Networks are too made up of neurons/perceptrons, which perform a dot product on its inputs and weights, add a bias and feed the value to the nonlinear activation function. CNNs however, make the assumption that the inputs are images, which changes the underlying architecture. This enables the algorithm to learn the relevance of different parts of the image, i.e. to better capture the spatial dependencies in the image. The architecture usually consists of three types of layers. The first two being the convolution and the pooling layer, which extract the visual features of the image. The third layer is a fully connected layer, which performs the mapping of the extracted features into the final output.

2.1.3.1 Convolution

The convolution layer is typically made up of a combination of the convolution operation and the activation function operations on the image. Convolution is a linear operation where a patch of weights, called a kernel or filter applied across an image, performing an element-wise multiplication between the weights in the kernel and the pixel values of the image within the area of the kernel and summing the resulting values. This operation can be understood as connecting the kernel to a node in the subsequent layer or feature map, which holds the summand. The connections between the input image and the feature map are defined by sliding the kernel across the image (Figure 5). This operation now maintains the spatial information in the visual features. The kernel weights are shared across all the image positions, which reduces the number of redundant learnable parameters compared to fully connected networks. The typical kernel size is 3 x 3, though 5 x 5 or 7 x 7 in some applications. Images with more

than one channel, such as colour images with the red, green, blue and alpha (optional) channels are convolved with windows of multiple kernels (Figure 5). (Yamashita et al., 2018)

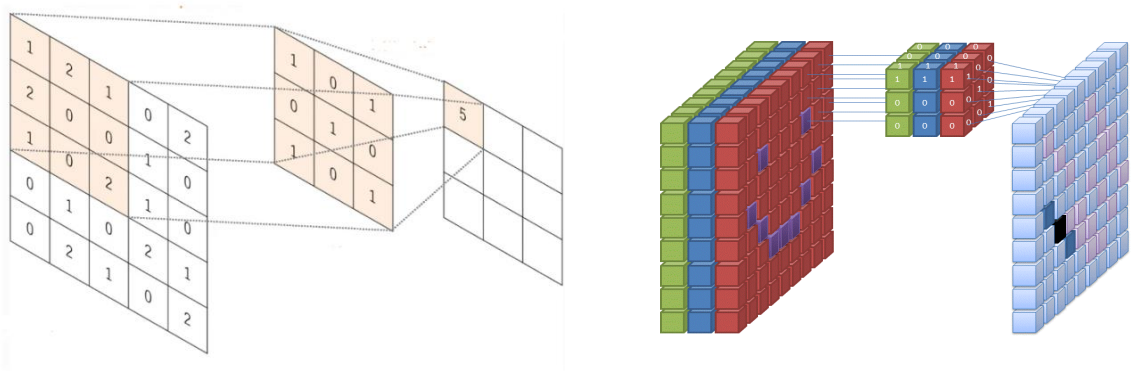


Figure 5: Convolution operation with a 3x3 kernel (left), Convolution of an RGB image (right).

Convolution by itself is a linear operation, so to introduce nonlinearity into the network, a nonlinear activation function is performed element-wise on the convolution outputs. For a 3 x 3 kernel, the value at the node in the subsequent hidden layer can hence be computed as:

$$g(\sum_{i=1}^{3}\sum_{j=1}^{3} w_{i,j}x_{i+p,j+q} + b)$$

Where b is the bias term. To extract different features from the images a number various different kernels of weights are used, which means that the output of a single convolutional layer is a volume of images rather than a single image of output values.

2.1.3.2 Pooling

The features in the feature maps are sensitive to location in the input. To lower this sensitivity, a solution is to down sample the feature maps, in order to make the features locally invariant to small shifts and distortions. This operation is again performed with a small patch of values applied across the feature map. However, unlike the learnable kernel weights in the convolution layer, these values are fixed. One of the most popular pooling variations is max pooling, which takes the maximum value in the patch applied across the feature map and discards all other values in the patch (Figure 6).

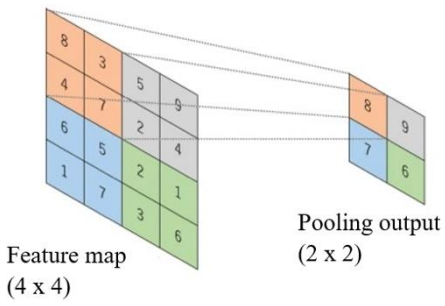


Figure 6: Pooling with a 2x2 patch

2.1.3.3 Dense layer

The final output feature maps of the convolution and pooling layers are flattened into a vector of features and passed to the MLP part of the architecture, which consists of one or more fully connected layers. The features in the input vector are connected to the output or to a node in the hidden layer by a learnable weight. Every fully connected layer is again followed by a nonlinear activation function. This layer gives the final outputs of the CNN.

By stacking convolution and pooling layers with many different kernels the network learns a complex hierarchy of features in the input images. From obvious low-level features to high-level details in the images, these deep architectures have become incredibly powerful for computer vision tasks.

2.1.4 Autoencoders

An autoencoder is an unsupervised generative algorithm that is trained to learn a model that represents the probability distribution of the model's input data. This way, the model learns the density estimation of the data, but because the distribution is continuous, the model also understands and can generate new synthetic samples of data, which fall somewhere within the probability distribution modelled on the training data. The main goal of the algorithm is therefore to learn a probability distribution most similar to that of the true distribution of the data. This enables us to automatically uncover the underlying structure and features in a dataset, which is a very powerful result as it is difficult to know how those features are distributed within large datasets. For example, in our dataset we have 175,937 images of unique urban forms. We do not understand anything significant about these forms, as they are incredibly varied across many different properties. An autoencoder can learn the landscape of the features in our dataset and by doing so uncover the regions in the training distribution corresponding to different features in the data. The street networks in (Figure 7.) all show high gridedness and high density, whereas the images in (Figure 8.) also show the street network, but those differ much more with respect to a diverse set of properties. All these features are somehow represented in the distribution of the data. This is what an autoencoder wants to learn to estimate.

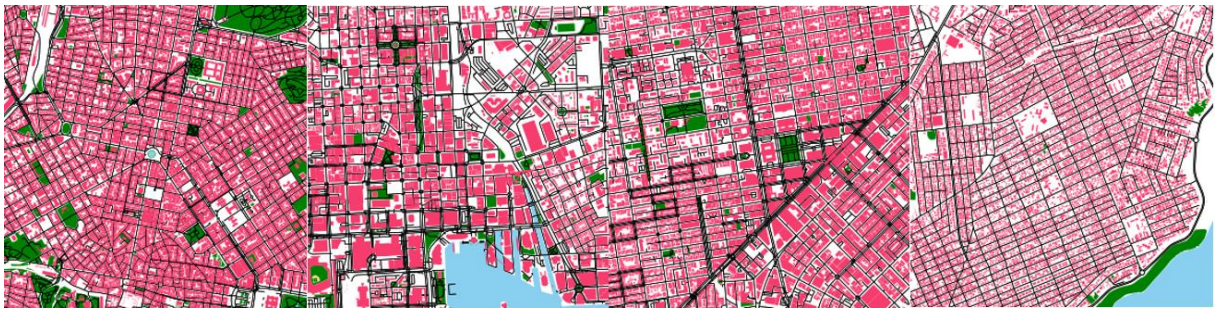


Figure 7: High density and gridedness urban form.

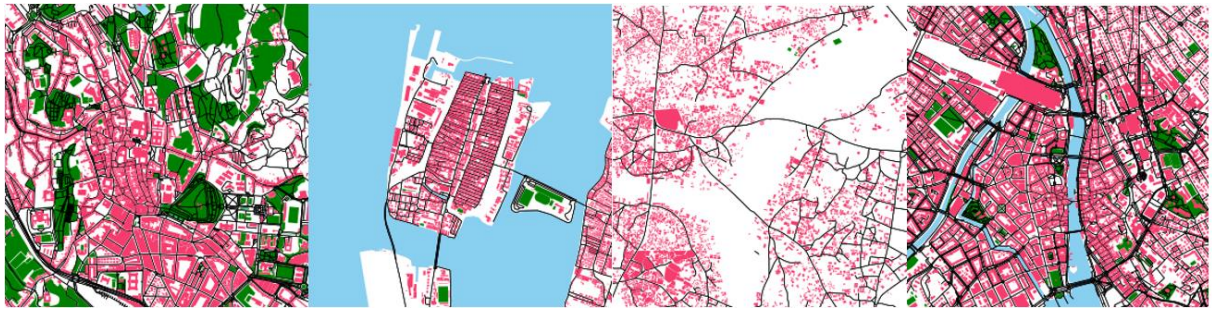


Figure 8: Urban forms with a diverse set of features.

An autoencoder consists of two parts. An encoder and a decoder (Figure 9). The encoder is fed an input, which it then passes through many successive deep neural network layers to generate a low-dimensional latent space at the output. It learns a mapping to encodes a piece of data x into a compact vector of latent variables z– a latent space vector. Autoencoders are an unsupervised algorithm, so there are no labels for z to train on. Therefore, the decoder is needed. The decoder reconstructs an image observation $\hat{x}$ of original input x from the latent space z. The decoder, too, consists of a series of neural network layers, but instead of compressing the data, these layers are used in reverse to learn to reconstruct it (Figure 9). The whole autoencoder network is then trained by minimizing the mean squared error between the original image x and reconstruction $\hat{x}$. This enables the model to learn the latent variables exclusively by observing the data, which is immensely powerful. However, the latent space is a deterministic encoding of the input data, which is good for reconstruction and denoising, other than that the applications are limited. The latent space is discrete, it is made up of points distinct encodings and the rest of the space does not mean anything to the model. To improve the model's understanding of the dataset, we need to incorporate continuity into the model, which is what Variational Autoencoders do

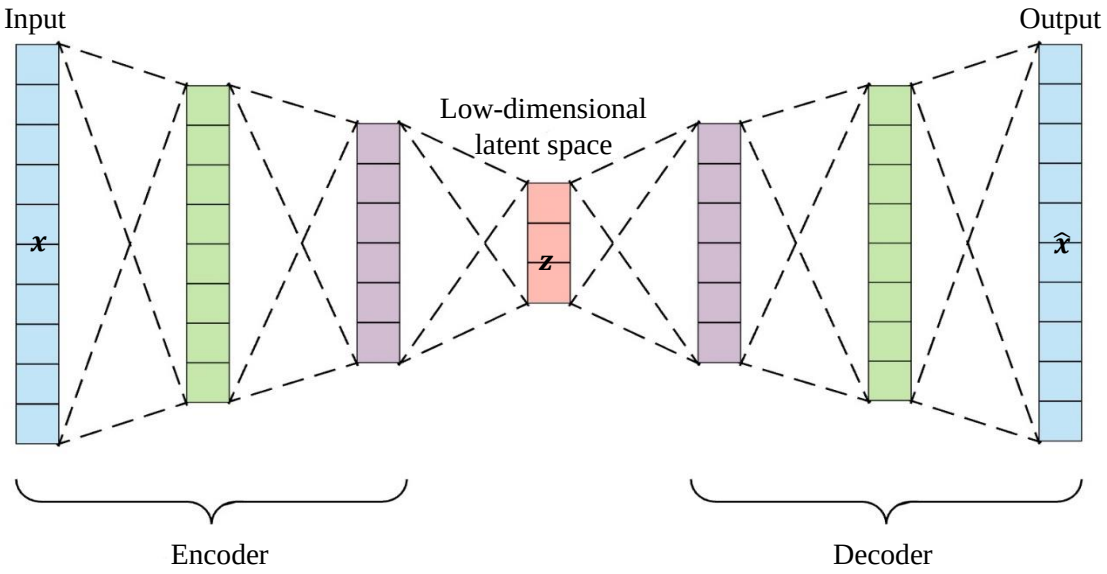


Figure 9: Autoencoder architecture

2.1.5 Variational Autoencoders (VAEs)

Variational Autoencoders replace the deterministic latent space z with a stochastic sampling layer. Instead of directly learning the latent variables, the VAE learns a mean μ and a standard deviation σ, which parameterize a probability for each latent variable. Instead of learning a vector of latent variables z the VAE learns two vectors – a vector of means μ and a vector of variances σ^2, z is then samples from the distribution defined by these parameters, creating a probabilistic representation of the latent space.

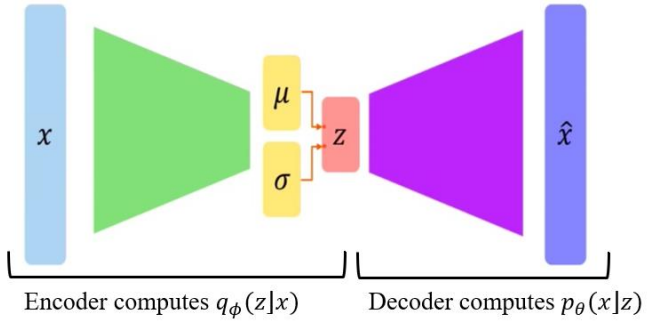


Figure 10: Variational Autoencoder

The encoder and decoder are now probabilistic models. The encoder will learn a probability distribution of z given the input data x and based on that learned latent representation z, the decoder will compute a new probability distribution for x given z. These distributions are learned by a separate set of weights (and biases) ϕ and θ. To train this model the following loss function is defined:

$$l_i(\phi, \theta, x_i) = \mathbb{E}_{z \sim q_\phi(z|x_i)}[\log (p_\theta(x_i|z) - \mathbb{KL}((q_\phi(z|x_i) \| p(z))$$

Here the first term is the reconstruction loss for the i-th data point. The overall loss is computed by summing over all the data points. It measures how effectively the decoder has managed to reconstruct an input image x given its latent representation z. The second term is the regularization term, which is introduced to enforce the learned latent variables z to follow the prior distribution, specified as a normal distribution with mean zero and variance one (Kempinska and Murcio, 2019). The regularized $\mathbb{KL}$ is the Kullback-Leibler divergence between the encoder's learned distribution $q_\phi(z|x_i)$ and the prior $p(z)$, more concretely, it measures how much information is lost if the prior is used to represent the input. The regularizer ensures that the latent representations z of each data point are sufficiently diverse and distributed approximately according to a normal distribution, from which we can easily sample, ensuring a better organisation of the latent space. This prevents the model from encoding data distributions that are far apart in the latent space and encourages overlapping of distributions (Figure 11), meaning that the decoder can sample from these overlapping areas, ensuring continuity and completeness of the reconstructions (Rocca, 2019). Continuity in the latent space means that latent vectors which are close to each other in the latent space, based on some distance metric will remain similar after decoding, while completeness ensures that the decoded samples will be meaningful with respect to the original data

distribution. Regularisation also keeps the network from overfitting on certain parts of the latent space, by encouraging the latent variables to follow a distribution similar to the prior $p(z)$.

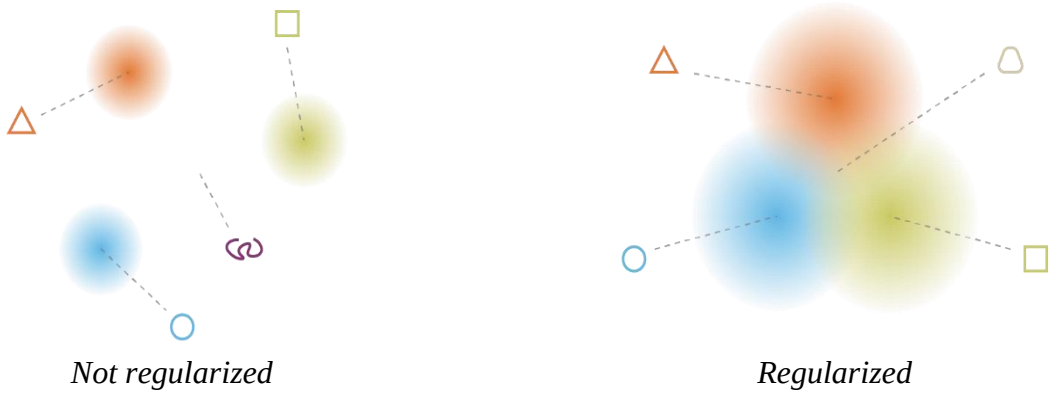


Figure 11: Regularized vs not regularized latent space

The stochasticity of the sampling layer means we cannot backpropagate gradients through it, as backpropagation requires deterministic nodes to be able to iteratively apply the chain rule through. This is solved by reparametrizing the sampling layer, by diverting the sampling operation to a new stochastic node ε, drawn from a normal distribution (Figure 12), which means that the stochastic sampling operation no longer happens at the bottleneck layer z, which becomes deterministic. More specifically, the operation that takes place is:

$$z = \mu + \sigma \odot \varepsilon$$

Where μ and σ are fixed, and $\mathcal{N}(0,1)$. This is called the reparameterization trick.

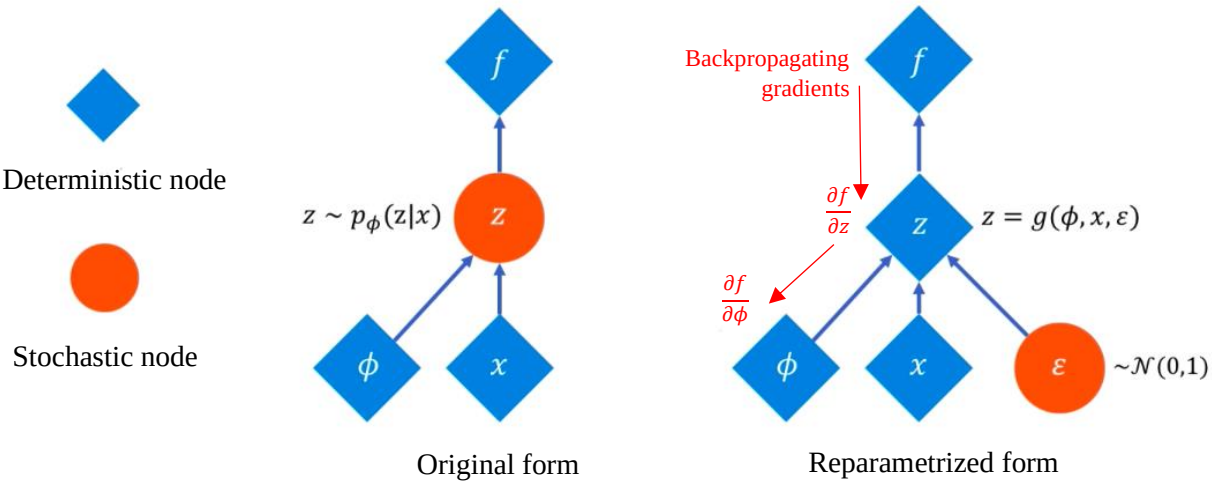


Figure 12: Reparameterization

2.1.6 Disentanglement

Our dataset consists of maps of cities and towns that have evolved over hundreds or thousands of years. They were formed under different cultures, different eras and responded to different pressures of the environment. This means that the data is incredibly rich in complexity and variability. This complexity has been thoroughly studied. Various theories have been proposed from many different fields. In such a complex dataset with so many different interdependent factors it is hard to find the most compact representation of the latent space possible, where the latent features are uncorrelated with each other. Nevertheless, we want to find features which are the most independent of each other as possible in order

to extract the most interpretable information out of our dataset. We try to achieve this by controlling the strength of the regularization, incorporating an extra parameter in the loss function. If each latent unit is sensitive only to a single generative factor and more or less invariant to perturbations in other factors, then that is a disentangled representation, which contribute to interpretability of the results, as they align with the factors of variation in the data, clearly highlighting which factors mean what.

2.1.6.1 BetaVAE

The most straightforward modification to the VAE, which achieves better disentanglement, is the β-VAE architecture, sharing the same incentive of generating samples of real data, but seeking to ensure that the learned latent representations capture the generative factors in the data in a disentangled manner. It modifies the loss function with an adjustable hyperparameter β on the regularization term in the loss function, which constricts the effective encoding capacity of the latent information bottleneck and encourages factorisation in the latent representation (Burgess et al., 2018):

$$\mathcal{L}(\theta, \phi; \boldsymbol{x}, \boldsymbol{z}, \beta) = \mathbb{E}_{z \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log (p_\theta(\boldsymbol{x}|\boldsymbol{z})] - \beta \, \mathbb{KL}((q_\phi(\boldsymbol{z}|\boldsymbol{x}) \,\|p(\boldsymbol{z}))$$

The network now feels two different pressures – the pressure on the encoding capacity of $\boldsymbol{z}$ and the pressure to maximise the log likelihood (first term in above equation) of the data $\boldsymbol{x}$ during the training of the model, encouraging the model to produce latent variables, which can efficiently reconstruct the input samples. The optimal thing to do in optimizing the above objective with $\beta >> 1$ is to encode only information about the data samples which can yield the most significant improvement in log-likelihood (Burgess et al.,2018.). The underlying assumption is that the real data $\boldsymbol{x}$ is generated using at least some conditionally independent ground truth factors and the Kullback-Leibler divergence term in the loss function encourages conditional independence in $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, hence higher β values should encourage learning a disentangled representation (Higgins, 2021). This assumption means that all the latent variables the model is trying to find are conditionally independent. In addition, these learned representations are continuous, which is not realistic when trying to model real world data. This concern is addressed by the JointVAE architecture described next.

Further, introducing β to the loss function has three general effects. First, it motivates reconstruction smoothness, meaning that there is a smooth transition between the reconstruction quality of different latent codes. Secondly, it encourages a compact representation of the input x – encoding its information into as few features as possible, which lowers the reconstructive quality of the output, as the network might omit a feature, which was adding some important detail, due to the restricted capacity on z. This is why it is important to find an equilibrium in the size of β. And thirdly, it incentivizes alignment between the main axis of variability in the data with the latent features, meaning that the capacity of the latent variables is aligned to their informative value (Burgess et al., 2018.). This is an area of active research, and many other algorithms exist, each with a different approach to improving disentanglement, mostly revolving around changing their approach to regularization.

2.1.6.2 JointVAE

This architecture tries to solve the problem of exclusively continuous latent representations modelled by BetaVAE. It is a framework also based on the VAE, so it shares its benefits, but also introduces the flexibility of learning a combination of continuous and discrete latent variables. On the MNIST digit dataset, for example, the JointVAE model would learn to disentangle each digit type from the its tilt, stroke thickness and width. The first of which is discrete, while the latter are continuous (Dupont, 2018). This is achieved by splitting the latent representation into a set of continuous latent variables $\boldsymbol{z}$ and the set $\boldsymbol{c}$ of discrete latent variables. The posterior distribution to be learned becomes the jointly continuous and discrete $q_\phi(\boldsymbol{z}, \boldsymbol{c}|\boldsymbol{x})$, while the prior $p(\boldsymbol{z})$ becomes $p(\boldsymbol{z}, \boldsymbol{c})$ and the log-likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{c})$. Updating the objective function accordingly, it becomes:

$$\mathcal{L}(\theta, \phi; \boldsymbol{x}, \boldsymbol{z}, \boldsymbol{c}, \beta) = \mathbb{E}_{z \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log (p_\theta(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{c})] - \beta \, \mathbb{KL}((q_\phi(\boldsymbol{z}, \boldsymbol{c}|\boldsymbol{x}) \,\|p(\boldsymbol{z}, \boldsymbol{c}))$$

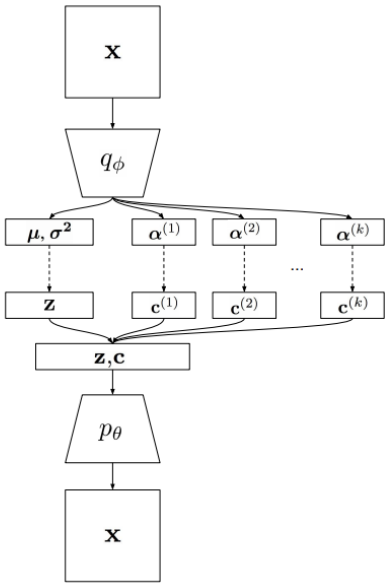
Similarly to BetaVAE, assuming both the continuous and discrete latent variables are conditionally independent in their respective distributions, we can further derive the objective function as:

$$\mathcal{L}(\theta, \phi; \boldsymbol{x}, \boldsymbol{z}, \boldsymbol{c}, \beta) = \mathbb{E}_{z \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log (p_\theta(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{c})] - \beta(\mathbb{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z}) + \mathbb{KL}(q_\phi(\boldsymbol{c}|\boldsymbol{x}) \| p(\boldsymbol{c}))$$

Separating the Kullback-Leibler divergence into continuous and discrete terms and summing them. Optimizing this objective function leads to the model simply ignoring the discrete latents $\boldsymbol{c}$ (Dupont, 2018). Two new parameters are introduced to combat this effect. A capacity term C, which acts as an upper bound on the mutual information between the encoder learned distribution q_ϕ and the prior p for each continuous and discrete variable. And a new constant γ forcing divergence term towards the capacity C. Increasing C gradually increases the amount of information that can be encoded in $\boldsymbol{z}$ or $\boldsymbol{c}$. In equation form this is (Dupont, 2018):

$$\mathcal{L}(\theta, \phi; \boldsymbol{x}, \boldsymbol{z}, \boldsymbol{c}, \beta) = \mathbb{E}_{z \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log (p_\theta(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{c})] - \gamma(\left|\mathbb{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z}) - C_z\right| + \left|\mathbb{KL}(q_\phi(\boldsymbol{c}|\boldsymbol{x}) \| p(\boldsymbol{c}) - C_c\right|)$$

As $\boldsymbol{c}$ is discrete, $q_\phi(\boldsymbol{c}|\boldsymbol{x})$ should be modelled by discrete distributions, but this would mean it could not be differentiated with respect to its parameters, so another reparameterization trick called Gumbel Max is employed, which allows sampling from a continuous approximation to the variable's discrete distribution, but we will not go further into detail.



The encoder q_ϕ hence takes the input data $\boldsymbol{x}$ and encodes them into parameters of the continuous distribution $\mu, \sigma^2$and the distributions approximating the discrete distributions $\alpha^{(i)}$ for each of the discrete latent variables i. The actual latent variables are then sampled (using the usual reparameterization trick) from each of these distributions and the values are then concatenated into a continuous and discrete latent vectors $\boldsymbol{z}, \boldsymbol{c}$, which are passed to the decoder p_θ(Dupont, 2018).

Figure 13: JointVAE architecture. Source

2.2 Related Work

Urban environments in all their larger-than-life complexity are incredibly hard to study. At the same time, their history, culture, innovation and development make them an appealing object of study for researchers from a diverse set of fields. Every city is unique, a reflection of their own history and environment. This is why historically, studies on cities have been qualitative and limited to small sample sizes or studies focused on individual cities and towns alone (Moosavi, 2017). With an ambition to capture some essential generalizable characteristics of cities and enable data-based comparisons, quantitative approaches have been made, however, these reduce cities to theoretic abstractions, which often fall short of capturing their true complexity (Kempinska and Murcio, 2019). A critique of these quantitative studies is that while trying to understand the big picture through developing universal equations, they fail to consider the qualitative factors, which though harder to quantify, often go a long way in urban design and planning. With the growing availability and increasing quality of urban data, due to advances in sensor technology and digitization, we are presented with an unprecedented

opportunity to use it as a real inquiry into many aspects of urban life. We believe creative research based on real world urban data can reduce this divide between the quantitative and qualitative methods in the field.

In our exploration of the current state of research in the field of livability and the form of the built environment, we grouped the research papers into two large groups in terms of how related their methods were to our research angle. The first group consists of research based on using machine learning methods and data to explore important questions in urban planning and design. The borders where the other group ends and begins are much more vaguely defined. It consists of a diverse range of interdisciplinary research with varying closeness to formal urban planning theory, but all of them rooted in computational and statistical analysis methods. We will start with an overview of the first group.

2.2.1 Machine Learning and Data in City Science and Urban Planning

The research directions and methods of the papers reviewed in this section differ greatly from each other, but they all make innovative use of machine learning techniques and real-world data. The table in (Table 1.) provides a visual overview of how the papers under review compare to each other in terms of machine learning technique and the datasets used. This figure alone highlights the diversity in approaches. The machine learning algorithms used range from deep architectures (GAN, VAE, CNN, FCN) to shallow architectures (SVM, kNN, Decision Tree, Naive Bayes), while in terms of data, the sources vary from governmental or municipal Geographic Information Systems to open source projects like OpenStreetMap and survey based approaches. The more significant heterogeneity in these papers, however, is in the research intention, which we will discuss here.

Paper	SVM	FCN	U-Net	GAN	CNN	kNN	Naive Bayes	Decision Trees	VAE	ANN
(Albert et al., n.d.)										
(Dubey et al., n.d.)										
(Oh, n.d.)										
(A.M. Chirkin and Reinhard Koenig, 2016)										
(Wijnands et al., 2019)										
(Kempinska and Murcio, 2019)										
(Liu et al., 2019)										
(Naik et al., 2014)										
(Porzi et al., 2015)										
(Zhou et al., 2014)										
(Arietta et al., 2014)										
(Yin and Wang, 2016)										
(Moosavi, 2017)										

Table 1: Research overview (to be changed)

Dataset	Satelite	GSV	Global Urban Footprint	Municipal GIS	OSM	Place Pulse	Places

GAN	Generative Adversial Network
FCN	Fully Convolutional Network
SVM	Support Vector Machine
kNN	k - nearest neighbors
ANN	Artificial Neural Network

Table 2: Acronyms

In their paper, Chirkin and Koenig (2016) address precisely this divide between quantitative algorithmic solutions to a complicated qualitative urban design problem. They developed a responsive reinforcement learning model and incorporated it into a human design process, to enable designers to explore a complex space of urban design solutions more efficiently. Similarly oriented towards using machine learning techniques to assist the urban design process, Oh (n.d.) highlight how computational models can complement designs by incorporating statistically justifiable insights instead of relying on the designers artistic intuition. They proved that classifying main streets is indeed a machine learnable

task, furthermore their system needed less features than are usually considered by designers and planners.

The other papers we have studied had less of an immediate focus on collaborative applications, but more on how to inform, evaluate and find interesting new applications for existing machine learning algorithms. Places is a scene recognition database consisting of 2.5 million labelled images, pulled from three image search engines (Google Images, Bing Images and Flickr), with 250 possible labels, introduced by Zhou et al. (2014). In addition, they trained a CNN (Places-CNN), which showed impressive performance on learning deep features for scene recognition tasks. Such research has a large array of applications, one of those being machine perception of urban scenery. Porzi et al. (2015) deal with exactly this. Using a crowdsourced dataset (Place Pulse 1.0) of pairwise comparisons of Google Street View (GSV) urban scenes, scored on safety, wealth, and uniqueness, they develop their own CNN architecture not only to predict the safety of an urban scene, but also to highlight which parts of the image correlate highly with the concept of perceived safety. However, Place Pulse 1.0 consists of scenes from only four urban areas, so their results are insufficient for extrapolation. Dubey et al. (2016) increase the number of cities to 56 from 26 countries covering 6 continents in Place Pulse 2.0. Compared to Porzi et al. (2015) they also expand on predicting perceptual attributes of street views to include *Safe*, *Lively*, *Beautiful*, *Wealthy*, *Boring* and *Depressing*, by developing two custom CNN architectures Streetscore-CNN and Ranking Streetscore CNN, which they also use for generating synthetic pairwise comparisons to predict urban appearance across countries. Again, this is done on a much smaller dataset of around four thousand GSV images of urban streetscapes in Buffalo, New York, though with a much narrower objective of objectively measuring key quantitative street level urban design features related to walkability. Yin and Wang (2016) used ANN and SVM algorithms to classify the enclosure (proportion of sky area) in streetscape images and correlate that against pedestrian volume showing the relationship between walkability, tree density, building heights and enclosure. Arietta et al. (2014) go deeper than this and explore how nonvisual city attributes such as violence crime and theft rates, housing prices, population density and tree presence can be predicted based on the visual appearance of the urban streetscape. They generated pairs of GSV panorama projections and (location, city-attribute-value) – from external data sources - and trained a predictor on these elements using non-linear Support Vector Regression. Supplementing a critical in-depth discussion of the results, they also use the predictors for various applications. For example, defining visual boundaries of neighbourhoods based on their unique visual appearance. Another is attribute-sensitive Wayfinding, which enables plotting city routes based on specific attributes such as safety or historic architecture for example.

Wijnands et al. (2019) too keep focus on machine perception of the urban environment, however, they instead take a generative approach. Their dataset matched 80,000 GSV images of Melbourne streets with 34,000 responses to a population-weighted health survey in the Melbourne metropolitan region and used a GAN to learn a style-transfer with style being defined in terms of the health or wellbeing. The network learned to translate GSV streetscape images from bad to good general health ‘style’. The image to image translations highlight the significance of sufficient urban green space, compactness of the built environment and natural surfaces in areas with good general health. Their findings complement the existing sustainable urban design theories. Albert et al. (2018) on the other hand demonstrate how modern generative machine learning is able to simulate realistic and diverse urban forms based on urban footprint data of 30,000 cities around the world. Modelling urbanization dynamics has long been an active area of research within urban morphology, though traditionally explored with Fractal based or Cellular Automata methods, which though impressively accurate in modelling the complexity of urban regions, they remain completely abstract, while employing techniques such as GANs enables us to study abstract spaces of urban forms rooted in reality. Moosavi (2017) and Kempinska and Murcio (2019) study urban form of cities around the world through their street networks. They employ another generative approach, namely Variational Autoencoders to extract the main spatial features of the street topology in the image data obtained from OpenStreetMap. As discussed in the background chapters above, these features are encoded in latent space vectors, which, in this case, can be used to compare urban forms in terms of the spatial characteristics of their street networks.

2.2.2 Urban Morphology and Computation

In this section we will focus on attempts to study urban form based on traditional computational quantitative methods not based on machine learning techniques, but rather on theoretical models of urban morphology such as fractals and other geometrical models or statistical analyses of spatial networks. In urban morphology the main object of study is the urban fabric of the city through neighbourhoods, street networks and buildings (footprints) and the interconnectedness of these. In this field, cities are studied as self-organizing complex systems[3], which is why their complexity is investigated with the same instruments as for similar objects occurring in nature and society often attracting physicists or mathematicians. An important open question in urban morphology is the classification of cities based on the characteristics of their urban form to eventually allow for large scale comparative analyses (Strano et al., 2013). The research papers described in this section propose different approaches to addressing this question.

Measure	Description	Measure	Description
embedding time series	examine variables to reveal deep structure and data	average streets per node	how well connected and permeable the physical form of the street network is, on average
Shannon entropy	how unpredictable a sequence is, based on number of types and proportional abundance	proportion of streets per node	characterizes the type, prevalence, and spatial distribution of intersection connectedness
mean information gain	how much new information is gained from each subsequent datum	average street length	how long the average block is between intersections; proxy for areal block size and grain
fluctuation complexity	amount of structure within a time series	node/intersection, edge/street density	how fine- or coarse-grained the street network is
urban transfer entropy	analytic tool for examining multi - scale urban patterns and flows	average circuity	how similar network-constrained distances are to straight-line distances
Ewing and Clemente field guide	set of methods for assessing the physical, visual complexity of the streetscape	diameter/periphery, radius/center	network complexity in terms of max/min size, structure, and shape
Cavalcante streetscape measure	image processing method to assess visual complexity on contrast and spatial frequency	node/edge connectivity	what is the minimum number of elements that must fail to disconnect network?
Simpson diversity index	assesses land use mix: how homogeneous or heterogeneous is the area of analysis	clustering coefficient	extent to which the neighbors of some node are connected to each other
dissimilarity index	how does the land use mix within a subarea relate to the mix across the entire area	multiple centrality assessment	uses primal, metric networks to examine multiple indices of centrality
Hausdorff fractal dimension	how a form's complexity changes with regard to the scale at which it is measured	betweenness centrality	the importance of an element in terms of how many shortest paths pass through it
destination accessibility	a function of land use entropy, amenity distribution, and network structure	space syntax	uses dual, topological networks to examine closeness centrality of a named street
closeness centrality	elements rank as more central if they are on average closer to all other elements	PageRank	ranking of node importance based on structure of incoming links

Table 3: Typology of urban form/design complexity measures. (Boeing, 2018)

A clear relationship between urban form and function[4] was discussed by Batty and Longley (1994). They recognized the fractal logic of cities and expressed it in mathematical terms in relation to population density functions (M. Alvioli, 2020). In Model Cities (Batty, 2007), Batty discusses and

[3] "Complex systems are large nonlinear systems of interacting components that can produce 'emergent' phenomena and allow structure to self-organize."

[4] Urban function can be conceptualized as function of city in relation to the society; as activities taking place inside of cities; or as a relation between urban (social) needs and urban (spatial) forms

defines the role and meaning of the model in relation to cities and our understanding of them. He classifies these into symbolic models, the focus of which is to simulate how logical function generates form usually based in analytical mathematics and iconic models, where the focus is on representing the geometry of form (Batty, 2007). Batty argues that these models are beginning to converge through digital representation. He goes on to describe models such as cellular automata and agent-based simulations as modern examples of symbolic models, which capture the spatial dynamics of self-organizing urban environments. Barthélemy & Flammini (2008) explore how studies of topological properties of spatial networks and network optimization are relevant for urban planning. They propose a model based on a local optimisation method reproducing certain statistical properties of complex transportation networks such as the number of edges (roads) versus the number of nodes (intersections). In discussing their results, they suggest that even in the absence of a global planning strategy, the evolution of city networks follow a universal mechanism (Barthelemy and Alessandro Flammini, 2008). Buhl et al. (2006) studied the planar graphs of 41 non-planned settlements from Europe, Africa, Central America, and India, including informal settlement types such as shantytowns. Among other significant results, they find that the studied street networks share most structural properties with ant tunnelling networks, exhibiting a similar relationship between cost and efficiency (J. Buhl et al., 2006). Strano et al. (2012) found that this general structural similarity between city networks can be attributed back to their planar representations. Their research proposed a classification method for cities based on performing a Principal Component Analysis on the distribution of street centrality[5] of 10 European cities. Performing a cluster analysis based on four different definitions of centrality and simple geometric properties, they characterized two clear separate clusters, which seemed to diverge in topographic and geographic local features like rivers, lakes, and hills.

Boeing (2018) was one of the first to emphasize that real world driveable and walkable street networks feature many overpasses or underpasses, that is, they are not planar, but exist in three-dimensional space. His research shows that modelling street networks as planar graphs can introduce bias into urban form analyses and by misrepresenting the real world, some statistical measures (overestimated intersection counts, underestimated average edge lengths, misrepresented connectivity) are made meaningless, hence weakening our arguments. In Boeing, 2018, they also studied the morphology and circuity of walkable compared to drivable street networks by simulating four million routes based on OSM data finding that driving routes are more circuitous than walking routes in most American cities, reaffirming that American cities were built around the automobile. In Boeing, 2020, they conduct a deeper empirical analysis into the evolution of American cities and the impact of car-dependence on street network design. They make an important point: “Street network design cannot merely reflect short-sighted transportation paradigms: due to its near-permanence, planners must carefully plan for decades or centuries of travel behavior, flexibly and sustainably,” highlighting the importance of empirical evidence in decision-making. Similarly, Palominos and Smith (2019) observe that street space is predominantly allocated to vehicles instead of pedestrians and cyclists.

Beyond the measures already mentioned, there are many more with which the adaptive complexity[6] of built environments can be assessed at many different scales, to enable urban planners and designers to evaluate the resilience, adaptability and even livability. As we are finding out for ourselves, these measures are scattered through different literature and research. Boeing (2018), again, compiled a toolkit of such indicators and metrics of complexity. Boeing uses some of these metrics to illustrate the differences between North American and European urban patterns. He empirically confirmed that North

[5] “Centrality measures serve to quantify that in a network some nodes are more important (central) than others.” (Paolo Crucitti, Latora and Porta, 2006)

[6] The term complexity refers to the higher-order phenomena arising from a system’s many connected, interacting subcomponents and describes both dynamics (i.e., processes) and structure (i.e., patterns and configurations) (Batty 2005). Complexity makes urban environments more resilient and robust, providing greater opportunities for social encounter, mixing, and adaptation through social learning. It entails greater connectivity, diversity, variety and sustainability.

American cities are more grid-like and found many interesting outliers, for example, Charlotte shows similar grid orientation and entropy values to countries in the European communist block, while Pittsburgh is clustered together with Munich and Vienna.

2.2.3 Our place

Inspired by Moosavi (2017) and Kempinska and Murcio (2019) we study urban form of cities around the world through their street networks by using VAEs to learn their characteristic structural features. Our research takes a different approach in two vital aspects. First, we build a dataset to specifically enable the VAE to extract features relevant to livability outcomes, i.e. instead of only extracting the street networks, we also include waterways, green spaces, building footprints and also put emphasis on the *walkable* street network. Doing this, we hope to describe the urban space as accurately as possible through a visual language (Figure 8) – we hope to capture as many descriptive features as possible in one comprehensive visual encoding. Next, we use a disentangled VAE architecture instead of a Vanilla VAE (VAE). This change improves on the mutual independence of the features extracted from the images, hence makes the results more interpretable and realistic. Ultimately the features we find should capture similar complexity indicators as those described in Table 3, but we also want to move beyond that and capture other meaningful features, not only based on the complexity of the street network, but rather based on the complexity of the symbiosis of the building footprints, green space, waterways and the walkable network all together.

Chapter 3

Methods

This chapter is devoted to explaining the technicalities in the development of our project, the engineering and processing of our dataset, the methods behind the analysis of the processed data, the external toolkits used and more detail explaining the implementation of our VAE architecture and how we use it on our dataset specifically.

3.1 Dataset

Our dataset is perhaps the most valuable and innovative part of our research. By carefully considering how we build our dataset, we set ourselves apart from existing research conducted in this space. This is why we will also allocate some space to describing our process in this paper. We start with generating a dataset of longitude and latitude values for every city and town around the world included in OpenStreetMap.

3.1.1 Extracting OpenStreetMap Data

OpenStreetMap (OSM) is a collaborative mapping project with a goal to create a free editable map of the world. Much like with Wikipedia, any registered user can contribute to the project. How the users collect their data varies widely. Some perform organized surveys, use satellite data, or use their local knowledge to provide additions. Collaboratively creating a detailed ever-growing editable map of the world, with daily updates, is an impressive achievement in itself, however, the primary output of this project is the underlying geodata, which the users contribute. This data is a rich and detailed representation of the built environment, which can prove to be invaluable in performing different urban planning analyses both at a global and municipal levels.

The data represents a diverse set of features on top of basic data structures, such as nodes, ways (links) and relations. These features are represented as tags, which describe a geographic attribute of the feature related to a specific node, way or relation. In order to extract longitude and latitude data for each city and town around the world, we needed to first process and filter (the feature tags) in the enormous raw OSM datasets encoded in the .pbf format, an alternative to XML.

Sub Region	Quick Links			
	.osm.pbf		.shp.zip	.osm.bz2
Africa	[.osm.pbf]	(4.4 GB)	✖	[.osm.bz2]
Antarctica	[.osm.pbf]	(29.1 MB)	[.shp.zip]	[.osm.bz2]
Asia	[.osm.pbf]	(9.3 GB)	✖	[.osm.bz2]
Australia and Oceania	[.osm.pbf]	(858 MB)	✖	[.osm.bz2]
Central America	[.osm.pbf]	(437 MB)	✖	[.osm.bz2]
Europe	[.osm.pbf]	(23.1 GB)	✖	[.osm.bz2]
North America	[.osm.pbf]	(10.3 GB)	✖	[.osm.bz2]
South America	[.osm.pbf]	(2.4 GB)	✖	[.osm.bz2]

Figure 14: File downloads hosted at Geofabrik.de

We downloaded a dataset for each individual continent (Figure 14) and processed it in three steps (Figure 15). First, we converted the datasets from

the .pbf format to .o5m[7] with a program called osmconvert, to enable us to in turn to filter it with another program called osmfilter and then again using osmconvert to produce a final output dataset in a .csv format containing the required fields (Figure 16).

```
C:\Users\Razer>osmconvert europe-latest.osm.pbf --out-o5m -o=europe-latest.osm_01.o5m
C:\Users\Razer>osmfilter europe-latest.osm_01.o5m --keep="place=city  =town =village"
  --drop-version --ignore-dependencies --drop-relations --drop-ways -o=europe-cities-towns.osm
C:\Users\Razer>osmconvert europe-cities-towns.osm --csv-separator="; "
  --csv="@lon @lat place name" > $europe-cities-towns.csv
```

Figure 15: Conversion and filtering of the latest North American dataset (in order of execution).

Through this process we produced a final .csv dataset for each continent, except Antarctica as the settlements there are too much of an outlier to represent the urban environment we are interested in exploring in this research. We appended all these datasets together into one large dataset containing every town and city worldwide (Figure 16). Out of all the tags and features, we decided to keep the longitude and latitude fields of each place, the town-city classification, and the city name. We omitted the country, because it was not included in the regular data structure, which would mean we would have to join the country field into our dataset from an external dataset, potentially introducing errors into our data. Instead, we decided to inject a country field into our data via the longitude and latitude fields at a later stage.

3.1.2 Generating maps with OSMNx

Having extracted datasets with longitude and latitude data of cities and towns, we went on to query OSM directly via the OSMNx Python package, which is a package for downloading, analysing and visualizing geospatial data made available by OSM. We used it to extract and visualise the geospatial geometries available for cities and towns in OSM in the form of a color-coded map of a 1km x 1km bounding box around the urban centre. We designed our queries based on the feature tags described in 3.1.1, namely, we were interested in constructing a comprehensive footprint of the urban environment also accounting for all possible forms of nature's presence. Our maps hence first included the walkable street networks and the building footprints. Next, we tried to consider every possible way nature can be present in an urban environment and tried to convert that into specific tags using OSM Tagfinder[8]. Using Tagfinder we could find specific tags related to a general search term. The tags we ended up finding fell into three high level tag categories, namely: '*Landuse*,' '*Natura,l*' and '*Leisure.' Landuse* and *Leisure* for example included tags such as 'Meadow', 'Forest' and 'Allotments,' while '*Natural*' included everything from 'Grassland' and 'Tree row' to 'Coastline,''beach' and 'water.' When building the maps, we plotted each group of geometries with their corresponding color. The street networks were black, the building footprints pink and everything else green, except for 'water' – blue and sandy[9] areas – yellow.

```
lat;lon;place;name
174.7772114; -41.2887953; city; Wellington
151.2164539; -33.8548157; city; Sydney
152.7959233; -31.6508445; town; Laurieton
152.4608481; -31.9128640; city; Taree
152.4982615; -32.1755506; town; Tuncurry
152.5131155; -32.1820852; town; Forster
151.2083011; -33.8354519; town; North Sydney
151.1818571; -33.7944797; town; Chatswood
151.0026660; -33.8139843; town; Parramatta
151.9533510; -27.5610193; city; Toowoomba
150.8938500; -34.4243941; city; Wollongong
150.6092697; -33.7460755; town; Blaxland
150.5655467; -33.7017613; town; Springwood
150.5355990; -33.7005603; town; Faulconbridge
150.3317665; -33.7156888; town; Leura
150.3697595; -33.7156389; town; Wentworth Falls
150.3121633; -33.7137590; town; Katoomba
151.2634168; -27.1822593; town; Dalby
153.3971541; -28.3267617; town; Murwillumbah
153.5761111; -28.2566667; town; Kingscliff
153.5469621; -28.1817080; town; Tweed Heads
153.0033506; -28.6209746; town; Kyogle
152.6122222; -28.3913889; town; Woodenbong
152.2957331; -28.3353902; town; Killarney
144.7772156; 13.4966854; town; Tamuning
```

Figure 16: Sample dataset

[7] .o5m format tries to combine advantages of both .pbf and .osm, that is, small file size, faster processing speeds and built-in compression.

[8] Because OSM is an open-source project, anyone can contribute, which means that there is a rich list of defined tags, yet there is no official way to search the tags, which are not defined on the OSM Wiki. Tagfinder is a search website, where a user can simply search for a general term and receive a list of all related tags.

[9] While the tag 'coastline' does not necessarily represent sandy areas, it doesn't matter as we know which specific tags the yellow color does represent – 'sand', 'coastline', 'beach'.

```
23          landuse = ox.geometries_from_point(place_point, tags={
24            'landuse':
25              ['meadow',
26               'forest',
27               'orchard',
28               'farmland',
29               'vineyard',
30               'farmyard',
31               'recreation_ground',
32               'allotments',
33               'reservoir',
34               'greenfield',
35               'plant_nursery',
36               'grass']
37          }, dist = dist)
```

Figure 17: Querying landuse geometries

More specifically, generating a map for a city consists of two steps. First, pulling all the required geospatial geometries (Figure 17) from OSM based on a pre-defined bounding box and the location values (longitude, latitude). After this, the geometries are plotted in layers stacked on top of each other to create the map image, which is saved at a file location (Figure 18). We perform this process for every row (location) in our dataset of cities and towns worldwide (Figure 16) and append a new column with the path to the image. All code is written in Python and heavily relies on the use of Pandas dataframes. Due to the size of the dataset, and the time to generate a single map image, the dataset was split into batches and the images were generated in parallel by running the script for different batches on multiple Jupyter notebooks simultaneously.

```
41          #plot geometries
42          fig, ax = ox.plot_footprints(leisure, bgcolor=bgcolor, bbox=bbox, color='green',show=False)
43          fig, ax = ox.plot_footprints(natural, ax=ax, bgcolor=bgcolor, bbox=bbox, color='green', show=False)
44          fig, ax = ox.plot_footprints(landuse, ax=ax, bgcolor=bgcolor, bbox=bbox, color='green', show=False)
45          fig, ax = ox.plot_footprints(waterways, ax=ax, bgcolor=bgcolor, bbox=bbox, color='#89cff0', show=False)
46          fig, ax = ox.plot_footprints(sandy, ax=ax, bgcolor=bgcolor, bbox=bbox, color=sand_color, show=False)
47          fig, ax = ox.plot_graph(G, ax=ax,bgcolor=bgcolor, node_size=0, edge_color=edge_color, show=False)
48          fig, ax = ox.plot_footprints(fp, ax=ax, bbox=bbox, color='#F73C6E', save=False, show=True, filepath=file, dpi = dpi)
49          required_data.at[x, 'map'] = file
```

Figure 18: Plotting geometries (notice landuse geometries at line 44)

A significant problem we faced in generating the maps was that of clipping geometries when plotting them. This occurred because often some of the geometries within the bounding box extended far beyond it and were plotted as a whole instead of being clipped at the border of the bounding box (Figure 19). Solving this problem for each individual map would be rather difficult, we would need to somehow determine the center the relevant area of each plot, as for example including a river in the geometries meant that a large part of the river geometry was plotted, meaning that the actual urban centre was minimized and pushed out of the centre to accommodate for the river. Luckily, we managed to find a solution within the OSMNx package.

Figure 19: Unclipped (left) and clipped map (right)

3.1.3 Data Properties

Our source data comprised of 175,937 cities and towns. There were 15,782 cities and the rest were towns. Here, we adopted the definition of town and city from OSM. In OSM, the tag *town* is defined as "an important urban centre, between a village and a city in size", their population (also a tag) ranging from 1000 to 70000 for 95% of the town entries in the OSM database. Similarly, according to OSM, cities are "the largest urban settlement or settlements within the territory". The median population tag value of cities in the OSM database is 130,000, however 13% of the entries have a population tag value between 20,000 and 50,000. OSM is free and under an open licence to anyone, so these kinds of errors

are bound to happen. We could filter them out based on the *population* tag when designing our queries. We did not, however, as we want to cover the widest possible range of cities worldwide and these entries could perhaps include under-developed areas where the population monitoring is not as precise, however the settlement is still considered a city. We realise that 13% is not insignificant and including this might impact the interpretability of our results.

In our exploration of the data we have found even more irregularities. Again, due to the openness of the OSM dataset, imperfect data is to be expected for certain geographical locations. This is also an inevitability of conducting research at such a massive scale. The particularly worrying case was the empty geometries being returned from OSM for specific tags, which resulted in nothing being drawn in their place and hence an image, which was falsely representing the physical attributes of a certain location. These irregularities included everything from missing building footprints to bodies of water not being drawn (Figure 20). Though, the street grid was fully drawn in a majority of the cases, we believe a large proportion of the dataset contained at least some irregularities, from less (a few missing small building footprints) to more (large bodies of water not being drawn) significant. However, quantifying the proportion of such corruption in the data would make for an extensive research problem in itself, and it would be especially difficult determining the ground truth. This is why we first acknowledge the impact this might have on what and how our model learns, but carry on with our work in order to look for interesting properties in the data, which is representative of the real world.

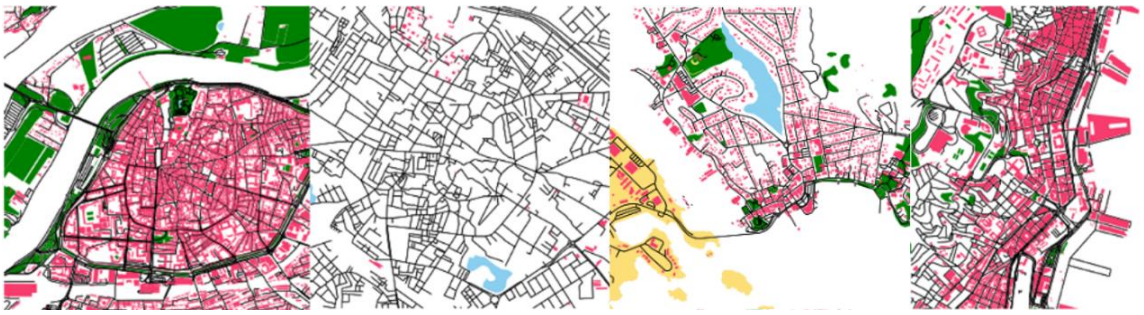


Figure 20: Data irregularities

3.2 Training the Models

This chapter covers the implementation details of our models. For all the model variations, in training them and further experimentation we have made extensive use of Pytorch.

3.2.1 BetaVAE

As discussed in section 2.1.5, our modified (Beta) VAE comprised of two networks; the encoder, which translates the high-dimensional input into the low-dimensional latent vector and the decoder, doing the

```
ConvVAE(
  (enc1): Conv2d(3, 8, kernel_size=(4, 4), stride=(2, 2), padding=(2, 2))
  (enc2): Conv2d(8, 16, kernel_size=(4, 4), stride=(2, 2), padding=(2, 2))
  (enc3): Conv2d(16, 32, kernel_size=(4, 4), stride=(2, 2), padding=(2, 2))
  (enc4): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(2, 2))
  (enc5): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(2, 2))
  (fc1): Linear(in_features=128, out_features=256, bias=True)
  (fc_mu): Linear(in_features=256, out_features=50, bias=True)
  (fc_log_var): Linear(in_features=256, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=128, bias=True)
  (dec1): ConvTranspose2d(128, 64, kernel_size=(6, 6), stride=(2, 2))
  (dec2): ConvTranspose2d(64, 48, kernel_size=(6, 6), stride=(2, 2))
  (dec3): ConvTranspose2d(48, 32, kernel_size=(6, 6), stride=(2, 2))
  (dec4): ConvTranspose2d(32, 16, kernel_size=(6, 6), stride=(2, 2), padding=(2, 2))
  (dec5): ConvTranspose2d(16, 8, kernel_size=(6, 6), stride=(2, 2), padding=(2, 2))
  (dec6): ConvTranspose2d(8, 3, kernel_size=(6, 6), stride=(2, 2), padding=(2, 2))
)
```

Figure 21: Convolutional VAE

exact opposite – recovering data from the low-dimensional latent representation. In turn, the encoder network consisted of 5 layers of CNNs (conv2d), each followed by the Relu activation function. Instead of pooling methods, the input data was down sampled using a combination of strides and padding parameters. The decoder network included one more CNN network in order to recover the full size of the original output from the 50-dimensional latent vector. The output of the final encoder (enc 5) is then fed to the first fully connected (dense) hidden layer, which maps the encoded features from the encoder into a hidden vector of 256 features, which are passed to another two dense linear layers, which compute the parameters of the latent distribution (a vector of means μ (mu) and a vector of log – variances $\log(\sigma^2)$ (log_var)). The relatively high number of these features will hopefully capture some of the complexities in the dataset and enable the model to learn more efficiently. These two parameters are used for the reparameterization (Figure 20) producing the latent space vectors z (50 dimensions), which is fed to another dense network, inflating the latent vector samples to features (128) used by the decoder network.

```
def reparameterize(self, mu, log_var):
    """
    :param mu: mean from the encoder's latent space
    :param log_var: log variance from the encoder's latent space
    """
    std = torch.exp(0.5*log_var) # standard deviation
    eps = torch.randn_like(std) # `randn_like` as we need the same size
    sample = mu + (eps * std) # sampling
    return sample
```

Figure 22: Reparameterization function

To begin the training, we converted our dataset into batches of tensors, representing the images and split it into a training and validation sets reserving 30% of the dataset for validation. In most of our experiments we used a batch size of 128 and a learning rate of 0.001. During the model training and validation, we used the modified loss function with the β-hyperparameter described in section 2.1.6, the implementation shown in Figure 23.

```
def final_loss(bce_loss, mu, logvar, beta):
    """
    This function will add the reconstruction loss (BCELoss) and the
    KL-Divergence.
    KL-Divergence = 0.5 * sum(1 + log(sigma^2) - mu^2 - sigma^2)
    :param bce_loss: recontruction loss
    :param mu: the mean from the latent vector
    :param logvar: log variance from the latent vector
    """
    BCE = bce_loss
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + 1 + beta * KLD
```

Figure 23: Loss function

We experimented by training three models, all based on the same architecture described above, but trained with different β – starting values.

- $\beta = 1$ – regular VAE
- $\beta_1 = 5$
- $\beta_2 = 250$

Decreasing the values to $\beta_1 = 1$ and $\beta_2 = 50$ over 750 epochs, which is the duration of the training.

3.2.2 JointVAE

As discussed in Section 2.1.6.2, JointVAE is based on the VAE, just as BetaVAE is. This means that in terms of the fundamental architecture, they are very similar. The encoder layer is made up of four convolutional layers followed by ReLU activation function and a fully connected hidden linear layer, which does the same job as in BetaVAE – encodes the features into a latent vector, which will be used in the reparameterization. The reparameterization layer in this model (Figure 23), however, includes one more layer, namely the alpha layer, for the discrete distribution approximations, from which the discrete latent variables are sampled from (Figure 24).

```
(fc_mean): Linear(in_features=289, out_features=10, bias=True)
(fc_log_var): Linear(in_features=289, out_features=10, bias=True)
(fc_alphas): ModuleList(
  (0): Linear(in_features=289, out_features=10, bias=True)
)
```

Figure 24: JointVAE reparameterization

While the continuous latent variables are sampled from a distribution parameterized by the mean (μ) and the log-variance ($\log(\sigma^2)$). In the configuration (Figure 24) 10 discrete and 10 continuous latent variables have been specified as out_features, which we concatenated and passed to the decoder, starting with another linear hidden layer followed, again by four transpose convolutional layers.

To train the JointVAE model, we again split our dataset into separate training and validation sets, reserving 30% for the validation with batch size remaining at 128. On top of the reconstruction error, we compute the Kullback-Leibler divergence separately for continuous and discrete functions. Here the continuous loss is the KL divergence between a normal distribution with diagonal covariance and a unit normal distribution – hence the _kl_normal_loss takes only the mean and the log-variance as parameters (Figure 26), whereas for the discrete variables the KL divergence is calculated between a set of discrete distributions (alphas) and a set of uniform discrete distributions. With every iteration, we also increase both the discrete and continuous capacities towards a set maximum. Training this model was much more resource intensive compared to training BetaVAE. Therefore, working in a computationally constrained environment, we limited our training to 2500 epochs.

```
def reparameterize(self, latent_dist):
    """
    Samples from latent distribution using the reparameterization trick.

    Parameters
    ----------
    latent_dist : dict
        Dict with keys 'cont' or 'disc' or both, containing the parameters
        of the latent distributions as torch.Tensor instances.
    """
    latent_sample = []

    if self.is_continuous:
        mean, logvar = latent_dist['cont']
        cont_sample = self.sample_normal(mean, logvar)
        latent_sample.append(cont_sample)

    if self.is_discrete:
        for alpha in latent_dist['disc']:
            disc_sample = self.sample_gumbel_softmax(alpha)
            latent_sample.append(disc_sample)

    # Concatenate continuous and discrete samples into one large sample
    return torch.cat(latent_sample, dim=1)
```

Figure 25: JointVAE reparameterization trick

In (Dupont, 2018), the model on CelebA was trained over 100000 epochs, so we did not expect to find significant results, given the complexity of our dataset.

```
# Calculate KL divergences
kl_cont_loss = 0  # Used to compute capacity loss (but not a loss in itself)
kl_disc_loss = 0  # Used to compute capacity loss (but not a loss in itself)
cont_capacity_loss = 0
disc_capacity_loss = 0

if self.model.is_continuous:
    # Calculate KL divergence
    mean, logvar = latent_dist['cont']
    kl_cont_loss = self._kl_normal_loss(mean, logvar)
    # Linearly increase capacity of continuous channels
    cont_min, cont_max, cont_num_iters, cont_gamma = \
        self.cont_capacity
    # Increase continuous capacity without exceeding cont_max
    cont_cap_current = (cont_max - cont_min) * self.num_steps / float(cont_num_iters) + cont_min
    cont_cap_current = min(cont_cap_current, cont_max)
    # Calculate continuous capacity loss
    cont_capacity_loss = cont_gamma * torch.abs(cont_cap_current - kl_cont_loss)

if self.model.is_discrete:
    # Calculate KL divergence
    kl_disc_loss = self._kl_multiple_discrete_loss(latent_dist['disc'])
    # Linearly increase capacity of discrete channels
    disc_min, disc_max, disc_num_iters, disc_gamma = \
        self.disc_capacity
    # Increase discrete capacity without exceeding disc_max or theoretical
    # maximum (i.e. sum of log of dimension of each discrete variable)
    disc_cap_current = (disc_max - disc_min) * self.num_steps / float(disc_num_iters) + disc_min
    disc_cap_current = min(disc_cap_current, disc_max)
    # Require float conversion here to not end up with numpy float
    disc_theoretical_max = sum([float(np.log(disc_dim)) for disc_dim in self.model.latent_spec['disc']])
    disc_cap_current = min(disc_cap_current, disc_theoretical_max)
    # Calculate discrete capacity loss
    disc_capacity_loss = disc_gamma * torch.abs(disc_cap_current - kl_disc_loss)

# Calculate total kl value to record it
kl_loss = kl_cont_loss + kl_disc_loss

# Calculate total loss
total_loss = recon_loss + cont_capacity_loss + disc_capacity_loss
```

Figure 26: Computing losses

3.3 Analysis

In order to try to understand our results, what kind of features are being encoded, and what patterns emerge, we needed to analyse and explore them. To gain an intuitive understanding of what the model is learning, we mostly relied on visual inspection heuristics of the latent space. In further research it would be interesting to compute a disentanglement metric, which quantifies the degree of disentanglement in the latent representation, by running inference on reconstructed images, generated with a single fixed latent variable and all others randomly sampled (Burgess et al., 2018.). Instead, we used clustering algorithms to try to visualise the data, especially since our goal was to build a system in which the data can be explored interactively.

3.3.1 Uniform Manifold Approximation and Projection (UMAP)

UMAP a novel technique for dimension reduction based on topological data analysis of large, high dimensional datasets. Unlike the widely popular tSNE, based on pure Machine Learning techniques. UMAP has been shown to outperform tSNE on a many metrics like preserving global structure in the data and significantly increased speed (3 minutes compared to 45 minutes projecting the MNIST dataset). Like t-SNE, UMAP is a graph layout algorithm, but unlike t-SNE its advantage is in its strong mathematical foundation, providing it with some formal postulations on how well a graph represents the real data. First UMAP constructs a weighted graph in high dimensions, the weights representing the

closeness of the high dimensional data points[10]. Points within an overlapping radius r of each other are connected, forming connected bodies which capture the fundamental topology of the data (see footnote 10). Choosing the right radius is the most important step determining whether small isolated clusters will emerge or just a single large one. This is solved by setting a point-based variable radius depending on the distance to its neighbours. The edge weights are turned into probabilities of two points being connected, preventing points far-away from each other from being connected, but a constraint is also introduced, so that a point needs to be connected to its nearest neighbour in order to prevent isolated outliers. This high dimensional representation of data is then projected into a lower dimensional space by a force directed graph layout algorithm. The advantage to t-SNE is in the step constructing the high dimensional data where more of the global structure is captured (Pearce and Coenen, 2021).

```
clusterable_embedding = umap.UMAP(
    n_neighbors=10,
    min_dist=0.1,
    n_components=2,
    random_state= 14,
).fit_transform(z_images)
```

Figure 27: UMAP embedding

We encoded our data into its latent representations (z_images) using the encoder of a previously trained network and transformed it into 2 dimensions (n_components) using UMAP (Figure 27). The basic parameters of the UMAP embedding can produce wildly different results with the smallest perturbation in the parameters controlling the transition between local and global structure. This is why it is important to try to understand what exactly they are doing:

- n_neighbors – used in constructing the initial weighted high dimensional graph. This is the most important parameter for controlling the relationship between local and global structure. Lower values will preserve detail in local structures, while larger values push the algorithm towards representing the larger patterns.
- min_dist – how close together the points will be in the lower dimensional space. Lower values will make distinct clusters by tightly packing points close together, while larger values will make UMAP preserve the overall topology with a looser structure of points.
- n_components – the dimensionality of the lower dimensional space.

Another important parameter is the metric, but we have left it unchanged at the 'Euclidean' for this project.

3.3.2 Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN)

In order to label our data into clusters, we used HDBSCAN, a popular hierarchical density-based clustering algorithm with an efficient implementation in Python. Unlike flat, centroid based algorithms like k-means, which rely upon certain properties (like equally sized and dense round clusters) of the data in order to find an appropriate clustering, HDBSCAN makes few assumptions about the clusters making it better suited for real world data exploration. It does make the assumption that noise is present, which is exactly the assumption making it more suitable for work with real data, which is often messy, corrupt and noisy. It looks for a hierarchy of clusters shaped by the underlying multivariate distribution. Instead of looking for certain properties in the shape of the clusters like k-means, it simply focuses on denser regions in the data, trying to separate denser regions from the sparser noise. Just like with UMAP, local density is determined by computing an inexpensive metric, namely the distance to the kth nearest neighbour – called core distance. Points with low density and consequently high core distance are spread apart, to be at least their core distance apart from any other point. This effectively makes the low-density regions even sparser, while preserving the denser regions. I am omitting the formal equation, but this is achieved by comparing the core distances of two points with the Euclidean distance between them and taking the maximum out of these three. This is called the mutual reachability distance, which is used to

[10] A representation of the topology of the data is captured by using the geometrical concept of a simplex, a k-dimensional object, made by connecting together k+1 points, points become 0-simplexes and a line is a 1-simplex.

treat the data as a weighted graph with the mutual reachability distance as the edge weights (McInnes et al, 2016).

The graph is dissected into connected components using a threshold value on the mutual reachability distance. A threshold value is found, which gives the minimum spanning tree[11] of the graph. The minimum spanning tree is converted into a hierarchy of connected components. This is done in a top down manner. The edges are sorted based on distance in an ascending order. The algorithm iterates through them, adding each edge into a new merged cluster. This hierarchy is of a very fine resolution accounting for each single point, so it needs to be condensed into a smaller tree with a larger cluster of points at every node. How these clusters are formed is determined by a parameter called minimum_cluster_size (Figure 28). At every split the size of the two new clusters is compared to the minimum cluster size. If the cluster is smaller, that it will eventually dissipate, while the larger cluster retains its label. If two clusters of sizes greater than the minimum size are created, then those clusters are separated and two new nodes are created in the tree. The most stable and persistent clusters in the tree – the ones which retained most of the points with every iteration are then extracted (McInnes et al, 2016). Another important parameter is the min_samples, which controls the degree to which the algorithm tends to classify points as noise and restrict clustering to denser areas, resulting in fewer clusters with larger values. We used this algorithm to cluster the 2-dimensional embedding of our encoded data produced by UMAP weights.

```
clusterable_embedding = umap.UMAP(
    n_neighbors=10,
    min_dist=0.1,
    n_components=2,
    random_state= 14,
).fit_transform(z_images)

labels = hdbscan.HDBSCAN(
    min_samples=5,
    min_cluster_size=100,
).fit_predict(clusterable_embedding)
```

Figure 28: HDBSCAN labels

3.3.3 Latent Space Vector Arithmetic and Interpolation

In previous sections on VAEs, we have discussed how the encoder network takes an image input and learns to encode it in a lower dimensional latent space representation. These representations take the form of vectors, which, in our case, live in a 50-dimensional hypersphere. Still, it is a vector space, so the operations of subtraction and addition are well defined on it, meaning we can also explore our dataset by means of simple vector arithmetic.

Using this, we performed two different kinds of visual inspection experiments. First, a latent traversal. Taking a random input sample, feeding it through the encoder and obtaining its encoded latent vector $\boldsymbol{z}$. If we perturb a single latent variable of $\boldsymbol{z}$, while fixing the rest of the variables, we can generate sample variations similar to the original representation $\boldsymbol{z}$. If we pass these through the decoder, we get generative reconstructions of the latent variations. We visually inspect these reconstructions in order to see what kind of generative factor of the data this single fixed latent variable represents. In one such pass, fixing a single latent variable, we are in effect performing a single latent traversal, but repeating this process with other latent variables, we are navigating the latent space, slowly obtaining an intuitive feel for which generative factors of the data the model has managed to learn.

In the second experiment, we took two images from the dataset encoded them into their respective latent vectors, just like in the traversal experiment, but instead interpolating between these vectors, by computing their difference vector and splitting it into n points and then generating n new vectors on the line between the two. These intermediate points, along with the original latent vectors were again fed through the decoder generating synthetic reconstructions of the latent space between the two original latent representations, resulting in a fluid visual transition from one data point to the other (Figure 29).

[11]The minimum spanning tree (a subset of graph, which has all the vertices covered with minimum possible number of edges) of a weighted graph is a set of n-1 edges of minimum total weight which form a spanning tree of the graph.

This was an interesting tool to analyse our dataset with. It allowed us to take two cities and see exactly how one transforms into the other, which features of the urban environment appear and which disappear. Another fun, interesting application of this would be to take the latent representations of two cities and add or subtract them generating new urban forms out of a combination of existing ones. Some sort of city arithmetic.

```
def interpolate(idx1,idx2):
  npoints = 10
  points = []
  gif_images = []
  with torch.no_grad():
      z1 = model_loaded.encode(train_loader.dataset[idx1][0].unsqueeze(0))[0].numpy()
      z2 = model_loaded.encode(train_loader.dataset[idx2][0].unsqueeze(0))[0].numpy()

      # Interpolation
      diff = z2-z1
      for i in range(0,npoints+1):
          point = z1 + i*diff/npoints
          points.append(point)

      # Plot each interpolated point
      for i,point in enumerate(points):
          image_tensor = model_loaded.decode(torch.from_numpy(point))
          gif_image = make_grid(image_tensor.cpu())
          gif_images.append(gif_image)
      image_to_vid(gif_images)
```

Figure 29: Interpolation experiment

3.3.4 Similarity Analysis with Euclidean Distance

As the latent space is a vector space of 50-dimensions, and the latent representations are points in this space it is natural to wonder about the closest neighbours of some point in this space. A data point's closest neighbours should be the latent vectors (and with it the original input images) most similar to it in terms of the encoded values of the latent variables. We measured this similarity or distance between two latent vectors $e = (e_1, e_2, \dots, e_{50})$ and $f = (f_1, f_2, \dots, f_{50})$ as the Euclidean Distance:

$$d(e,f) = \sqrt{\sum_{i=1}^{50}(e_i - f_i)^2}$$

3.4 Interactive Visualization

We visualised the clustered data in an interactive web format using Three.js, a JavaScript library which generates low level code for WebGL and React, a JavaScript library for building user interfaces. For each trained model, we encoded all of our data into their latent representations, projected it onto two dimensions using UMAP and generated cluster labels for it with HDBSCAN. We dumped both the UMAP embeddings and the HDBSCAN labels into a JSON file in order to easily process it in Javascript.

We rendered the points in 2D using the Three.js renderer and two simple vertex and fragment shaders written in GLSL. To try to make the visualisation as interactive as possible, we decided to visualise points as image sprites instead of basic geometry. The sprite geometry required its own vertex and fragment shaders. The images are square, so the sprite geometry included 4 vertices, so each point had an associated 4 – amounting to a geometry of 60,2048 vertices being rendered smoothly in a web browser environment.

Our dataset was not overwhelmingly large (15,062 images, see section 4.0), however, for each point geometry in the scene, the renderer will make a separate "draw" call to render its texture with every

frame of the animation and each such call requires the CPU the browser is running on to send all geometry data the host GPU. This transaction will happen many times per second, depending on the frame rate at which the scene is being rendered (sometimes up to 60). It becomes apparent that individual textures are not feasible if a visualisation is to run smoothly. This is why we transformed our dataset of 15,062 city centre images into 59 texture atlases made up of 16 x 16 images reduced to 128 x 128 pixels each, meaning that each texture atlas (Figure 30) was of size 2048 x 2048 pixels (which is the limit for the size of a texture on many devices) and contained 256 images, except the last one containing the remaining 214 images. The benefit of texture atlases is that they bundle the images together and hence allow us to reduce the amount of textures, but they also make it easy to extract an image based on an index in the dataset, by cropping and offsetting the texture elements and extracting each individual one when required.

In order to make the clustering truly interactive and not just a plain geometry of points, we added a sprite highlighting functionality activated on hover. Hovering over a point produced an enlarged semi-transparent monochrome sprite over the it, making it clearly visible through contrast with the neighbouring points, while also rendering the original image in a sidebar on top of the city name, also appearing on hover.

Figure 30: Texture atlas

Chapter 4

Results

4.0 COVID-19 Impact

I proposed this project to the School of Computer Science towards the end of Summer in 2020, perhaps blissfully ignorant of the COVID-19 pandemic and with every intention of returning to the University campus to conduct my research. This did not happen and I stayed at home in Slovenia. In terms of the impact of COVID-19 on my life and studies, I consider myself to be among the very lucky ones. Even though I did not return to the University, I managed to continue my studies with little to no impact. At the same time, as per Slovenian governmental advice, I could still go outside and enjoy the nature I have at my doorstep. I feel very privileged in saying that the pandemic has not had a noticeable impact on my mental health.

This project, however, was significantly impacted by the pandemic due to the constrained computational resources available to me at home. The whole of the dataset I generated – cities and towns consisted of 175,937 images or around 35 GB of data to train a model on, something not feasible on my home computer. For a long time, my work was thus constrained to small experiments (5,000 images - ~3% of my dataset) in the environment of Google Colab. Seeing as the results of those experiments would not get me far in terms of reaching my objective, I contacted the Reasoning & Explainable AI Lab in order to ask for some direction regarding acquiring computational resources. They liked my project idea and the fact that I had already generated a dataset and tested some model architectures, so they invited me to collaborate and remotely use their much more powerful machines for the training of my models. After a month of back and forth communication, I was granted access, but the machine to be used by me was not yet set up. I waited for around a month longer, but due to the approaching project deadline, and the uncertainty regarding the state of the machines available, I decided to commit to working in Google Colab PRO. Google Colab PRO is an environment mostly meant for small scale experiments and tutorials and not for larger research projects. I reduced my dataset to 15,062 cities. Even with this reduced dataset size and a Google Colab PRO subscription, I managed to train my models for at most 1000 epochs, not nearly enough to obtain meaningful results. It became clear that due to limited access to computational resources, this project has become more of a feasibility study than an actual research project.

4.1 Training

In our experimentation, we trained four models, three variations of the BetaVAE with different magnitudes of the beta parameter and a model based on the JointVAE architecture. Both architectures were described in Sections 2.1.6.1 and 2.1.6.2 with their implementations in 3.2.1 and 3.2.2. All of the models based on BetaVAE were trained over 750 epochs with a learning rate of 0.001, which took around 18 hours per model. On more than 750 epochs, Colab became unexpectedly unreliable, with the session being terminated for no apparent reason.

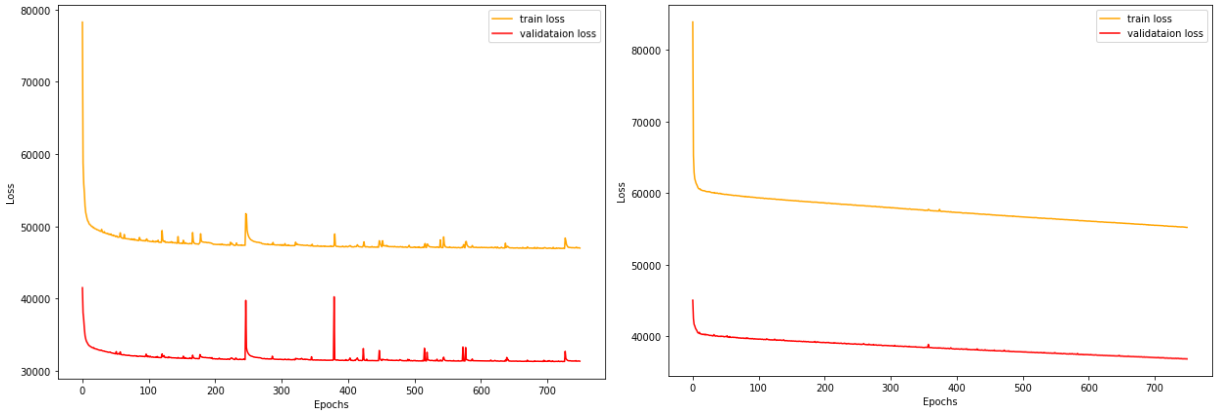


Figure 31: Training and validation loss graph for VAE (β=1) – left and BetaVAE ((β = 250)- right (train loss - orange, validation loss - red)

Observing the loss plots for the BetaVAEs (Figure 31) it seems that the increased pressure on the encoding capacity of the latent representation and a higher penalty on the reconstruction both enforced by β stabilise the training process. The model trained with β = 1 is showing some spikes in the loss plot, perhaps a consequence of batch gradient descent with a batch-size of 128 and the model encountering an unlucky batch significantly different to what it has seen up until the spike. The model trained with β, starting at 250 does not seem to show such spikes, but this could mean that the model is learning the noise in the data or that the restricted capacity of the latent variables is failing to capture enough information to distinguish between data samples – this effect can also be seen in the poor and noisy reconstruction quality (see Section 4.2). However, out of all the BetaVAE variations, it seems to be the only one showing a slight decline in its loss curves, while both the training and validation curves are almost levelling off for two other BetaVAE models, implying that they are failing to learn any significant characteristics of the complexity of the data. Their initially sharply decrease, but quickly slow down, almost levelling off. This kind of behaviour in training could suggest that the models start with overfitting on the initial data and fail to generalize any further, i.e. underfit on the rest of the data. The large gap between the training and the validation loss also suggests underfitting.

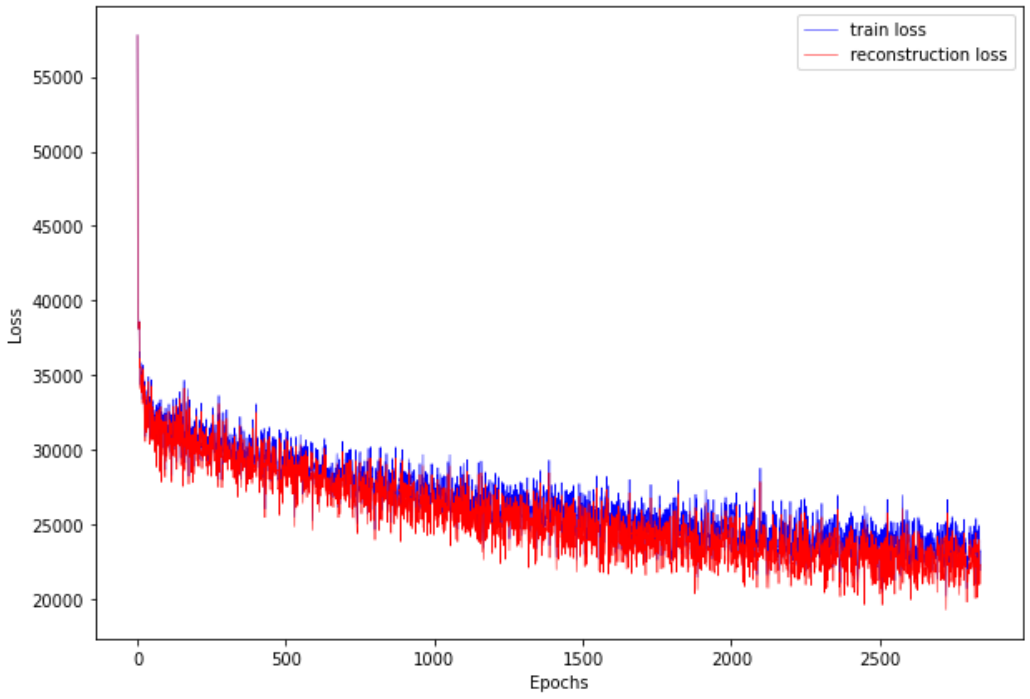


Figure 32: Training and validation loss for JointVAE

The training of the JointVAE model was less stable compared to the other ones (Figure 32). It seemed to have performed better in learning the complexity of the data, with the noisy loss curves being a reflection of that compared to BetaVAE ($\beta = 250$). In training the JointVAE model, we had control over the capacity of the latent variables, acting as an upper bound against the pressure of the β parameter to decrease the dependencies between the latent variables.

4.2 Reconstruction Quality

The objective of a regular VAE in terms of its loss function is to try to minimize the reconstruction loss conditional on the choice of a regularizer. As discussed, with adding the β parameter to the VAE's loss function the model's objective becomes to learn an efficient factorised encoding of the input data, sacrificing reconstruction quality by restricting the amount of information able to flow through the latent bottleneck. It is possible for the VAE to learn a perfect reconstructive mapping, whereas this is much harder achieved by the BetaVAE.

In our experiments, we too have observed this reconstructive trade-off of disentanglement. The reconstructed images generated from the VAE (β=1) managed to preserve some spatial information (Figure 33). Larger green areas were reconstructed almost perfectly, while the smaller ones remained accurate in terms of position in the image. The pink buildings, representing denser data became a smear of blurry pink color in the image. The intensity of the pink often encoded some information about the density of the building footprints in the original image, as denser areas with many smaller buildings became a almost a solid shape of pink color in the reconstructed image. It also managed to reconstruct a "shadow" of the street network, preserving some information about the connectedness of the grid, while losing almost all of the detail. The model struggles with reconstructing bodies of water, perhaps because of the lower color contrast or a rarer presence in the training data. These results suggest that the VAE performs better with reconstructing wider, uniform patches of pixels rather than exact narrow lines or line connected shapes.

Figure 33: VAE (β=1) originals (above) and reconstructions (bellow)

Increasing the β parameter to 5, does not drastically lower the reconstruction quality, though some subtleties are visible we believe. For example, the images with mostly roads and little greenery or building footprints highlight the increased blurring, as the model struggles to capture the characteristics of the street network (Figure 34) (most visible in third picture from the right).

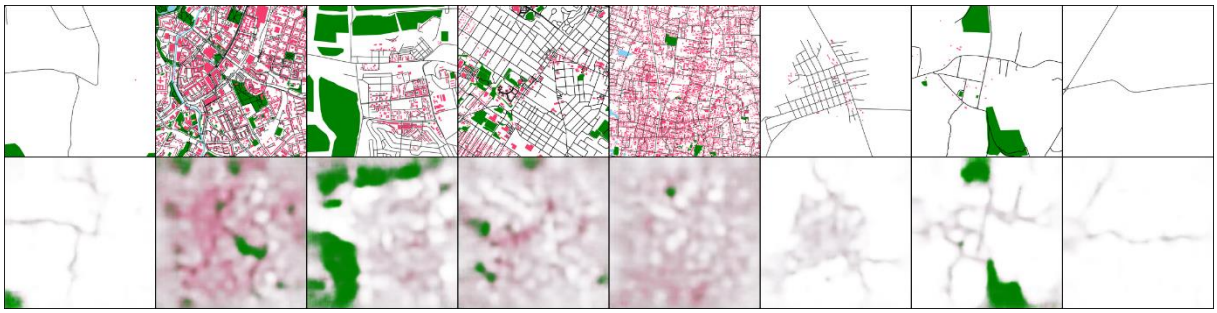


Figure 34: BetaVAE (β=5) originals (above) and reconstructions (bellow)

We also observe that the boundaries between the green areas and the surroundings become more blurred and less defined (second picture from the left).

The consequences of increasing β to 250 has on the reconstructive quality speak for themselves (Figure 35). Almost all spatial information, except for the green areas, is lost. The street network is hard to make out and denser images with building footprints, green areas and roads begin to blend into one pixelated painting. There are many techniques for improving the reconstructive ability a BetaVAE model, while still enforcing disentanglement. One such technique is introducing a capacity term into the objective function, similarly to JointVAE (see Section 2.1.6). These results highlight the undeniable trade-off between straightforward disentanglement attempts and reconstruction quality.

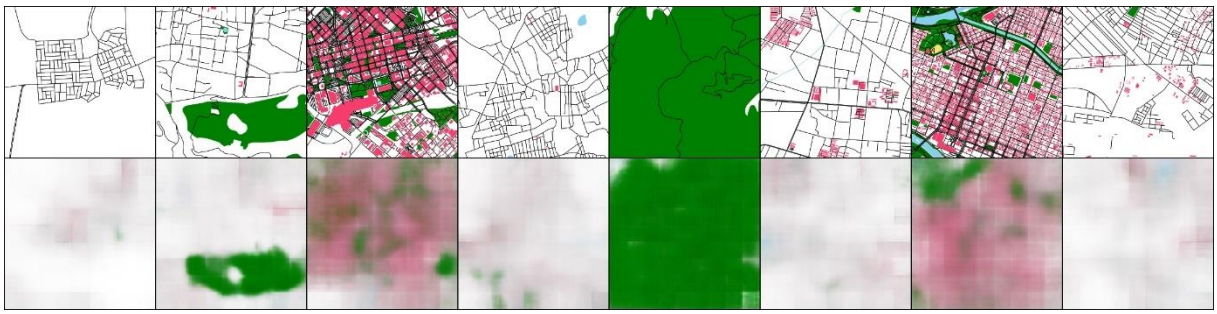


Figure 35: BetaVAE (β = 250) originals (above) and reconstructions (bellow)

Looking at the JointVAE reconstructions (Figure 36), we observe some obvious differences compared to the Beta models. First, the JointVAE model performs worse on reconstructing the street network on a blank background (no building footprints or other geometries). If you look at the last image on the right, the reconstruction looks very chaotic and noisy in a way that we have not seen with the Beta models. Encoding information into discrete variables perhaps leads it to reconstruct some images with features that were not in the original image. For example, in the second image from the left, there is a lot more pink than the few dots in the original image, perhaps because the model learned to correlate between dense street networks and building footprints. The overall reconstruction performance of the JointVAE does not seem to be as consistent as compared to the Beta models, there are more edge cases on which the model performs a lot worse. We think this means that the number of discrete and latent variables is too small for the model to learn an efficient representation.

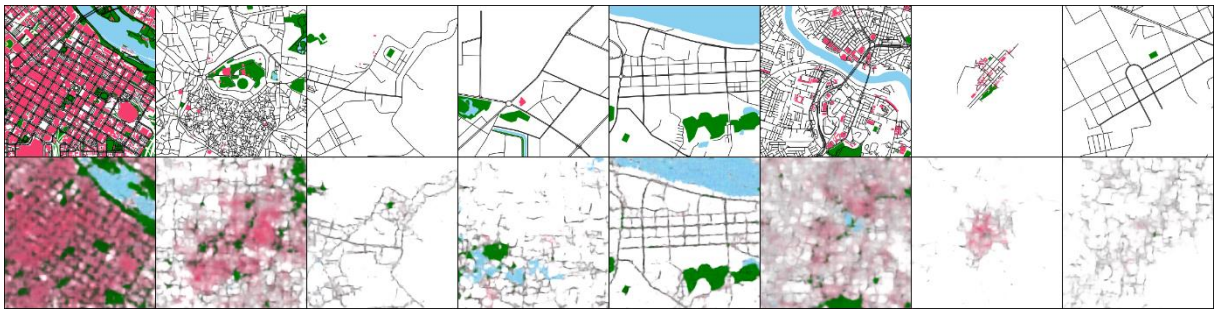


Figure 36: JointVAE originals (above) and reconstructions (bellow)

4.3 Latent Space

In chapter 3.3 we discussed various techniques we implemented in order to explore the latent space. In this chapter we will discuss the experiments we conducted using those techniques. Some experiments, like interpolation adding the latent vectors of two cities are better experienced animated, so we will reserve some of the discussion in this chapter for the screencast presentation. First, we discuss latent traversals. The latent traversals for the beta models were uninterpretable. Even after a lot of experimentation, we did not learn the effect of perturbing a single variable means, as the perturbed

reconstructions (Figure 37) were too chaotic and did not make sense compared to the original input images. Though, we again notice the blurring effect of the increased β parameter (Figure 35).

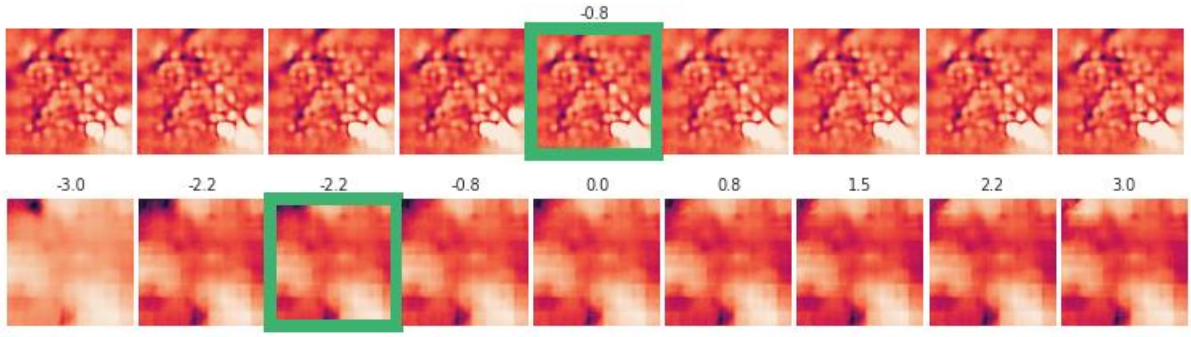


Figure 37: Traversal of one latent variable. VAE (β = 1) above, BetaVAE (β = 250) bellow.

Experimenting with different latent variables simultaneously, we have not found any evidence of disentanglement. The reconstructive differences between the decoded perturbed latent vectors were too insignificant to claim any disentanglement. We propose three interdependent reasons behind these poor results. First, the data is incredibly complex, so the number of generative factors is unknown and hence we have no idea about the ideal number of the latent variables. Second, the training was too short and there was not enough training data. And third, we postulate that to efficiently learn and disentangle 50 latent variables we would need to train the models for a significantly longer training period than 750 epochs. As a result the models learned a 50-dimensional latent representation, but the information was inefficiently factorised. In this case, larger values of β might have been hurting the encoding, as it only restricted the amount of information that could be encoded in the 50 dimensions, meaning that it did not have the desired factorising effect, but it only restricted what the 50 latent variables could represent entangled together.

The JointVAE model performed better on latent traversals. Though the traversal reconstructions were far more familiar than those generated by the beta models, we still found it difficult to interpret what the latent variables represented. Nevertheless, the transitions between the reconstructions with the discrete traversal (Figure 38) seem to be discrete compared to the continuous traversal (Figure 39). These are affirmative results that we can observe and explain, but not much more.

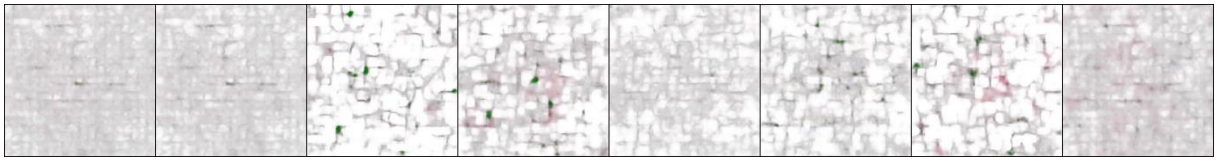


Figure 38: Latent traversal over the first discrete latent variable.

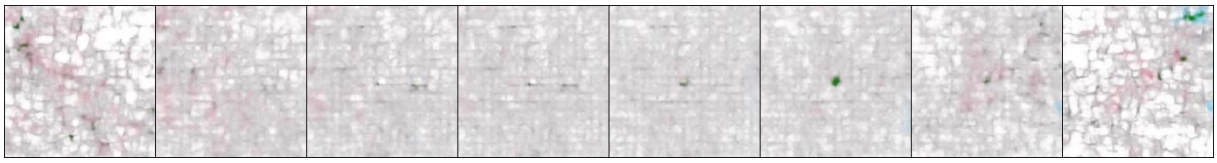


Figure 39: Latent traversal over the first continuous latent variable

In section 3.3.4 we discussed another method of exploring the latent space, that is, similarity analysis based on the Euclidean distance between data points. The sequence of steps is as simple as can be. Given a latent representation of an image in our dataset, we sort our latent space (vectors) based on the Euclidean distance to the latent vector of the chosen image. We take the first n (in our case 5) such latent vectors and find their associated image. Figure 40 shows a grid of randomly chosen cities (top row) and their most similar urban forms based on the closeness of their latent vectors. The technique indeed manages to find urban forms with similar properties. For example, we can observe that the urban forms

are matching in terms of density of both the building footprints (third and fourth column) and density of the street network. Further, network structure and orientation are preserved between the found closest neighbours in all samples. It can again be observed just how varied and complex the dataset is. The unique nature of every point in the dataset makes it hard for the model to learn general trends. We observe this in how it arranges cities based on similarity. It does not, for example, disentangle the street network from the building footprints, which means that a grid populated with building footprints is different to an empty grid, which can be seen in comparing the columns in (Figure 37). In the third and fourth column, where the data points are denser with building footprints, the grid becomes less significant than in the first, second and fifth columns, where the grid is clearly present in every single image, while even preserving some notion of the grid orientation.

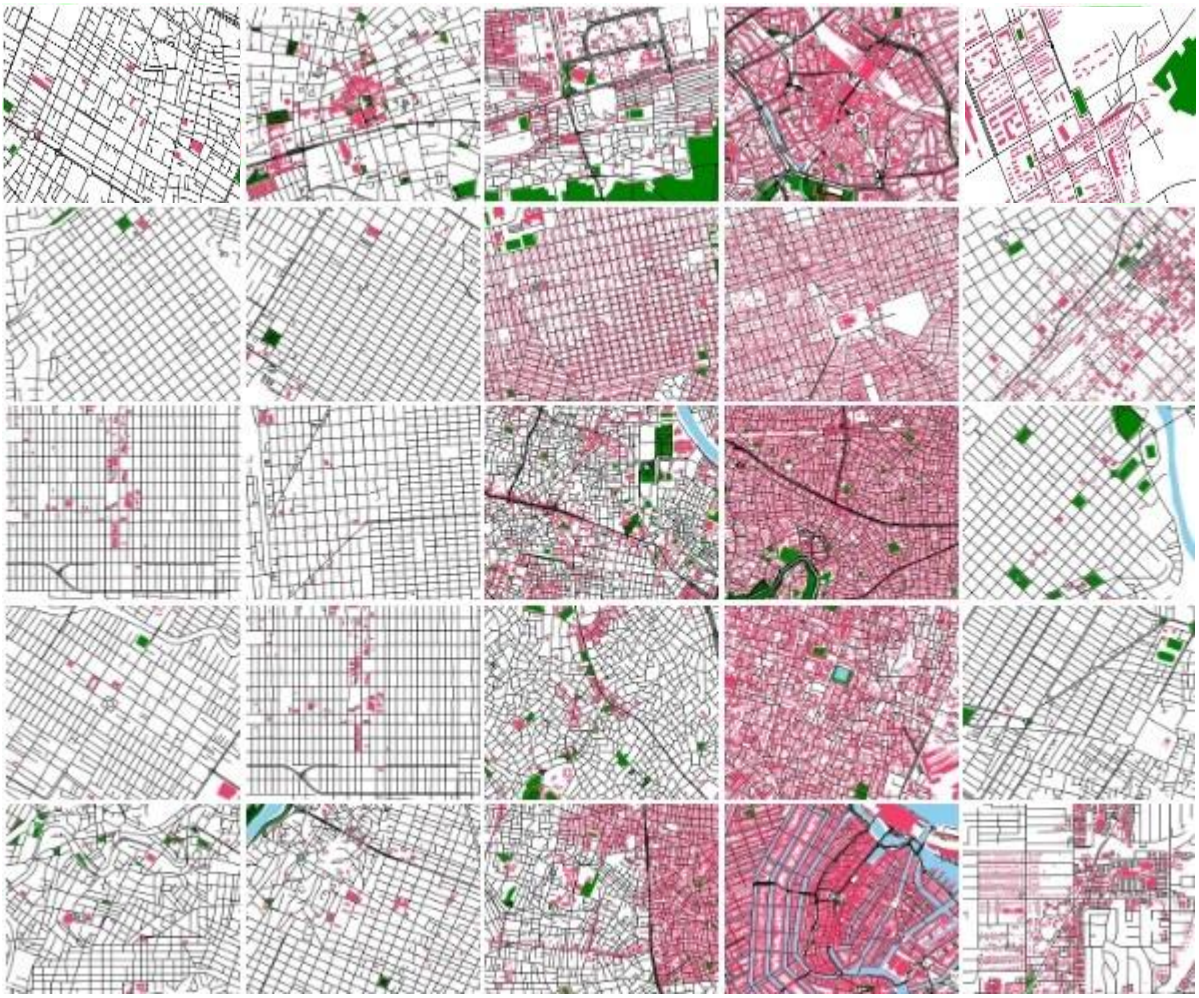


Figure 40: City samples (top row) and their associated closest neighbours (rows 2-5)

4.4 Clustering

Clustering the latent representations was among the main objectives of our research. We expected to use clustering techniques in combination with information condensed in the latent vectors to organise our data in a way that would reveal spatial patterns and relationships between the cities that go beyond what is currently possible clustering directly on the image data. We also developed a web visualisation tool that makes it easier to perform a visual exploration of the clustering, by making the clustering interactive enabling direct and immediate inspection of the cluster members.

Our trained models, however, as the training plots show, did not manage to encode enough meaningful information about the urban forms, which is also visible in the clusters (Figure 41). The UMAP

dimension reduction algorithm projects the data into two-dimensional connected shapes and within these shapes distinct clusters are found by HDBSCAN, implying smooth transitions between the data points and the clusters, which is not surprising given the interconnected complexity of the dataset. We chose the HDBSCAN parameters using the condensed tree method mentioned in section 3.3.2, giving us the most resilient clusters in the data. Upon visual inspection of the cluster members, we did not observe any obvious distinct features of the cluster members and in many of the cases the clustering was almost arbitrary, especially for the beta models. Granted, the diversity of the data elements makes it challenging to discern characteristic features of the clusters, as we need to consider such a large number of possible features.

One of the abilities of HDBSCAN, which we used to generate the cluster labels, is to refuse to assign points to found clusters, but rather to label gray as noise. According to such clustering, noise was present in all of our encoded data for all of our trained models. Comparing the BetaVAE (β = 250) clustering with the other beta models, we see that the clusters are more defined with the larger β value, however another consequence is that more data is labelled as noise. This further implies that the increased independence pressure exerted on the latent variables by β, introduces more noise into the data, by restricting the flow of perhaps vital information through the latent bottleneck.

The clustering of the data encoded by JointVAE is certainly distinct from the beta models. There is not much noise classified by HDBSCAN and the UMAP no longer projects the data into a connected shape, but rather into two large clusters, one relatively homogeneous and the other a conglomeration of smaller clusters. This might be due to the significantly lower number of latent variables used in the encoding and also due to the fact that they were split into continuous and discrete, perhaps making it easier for HDBSCAN to classify compared to the 50 continuous latent variables learned by the beta models.

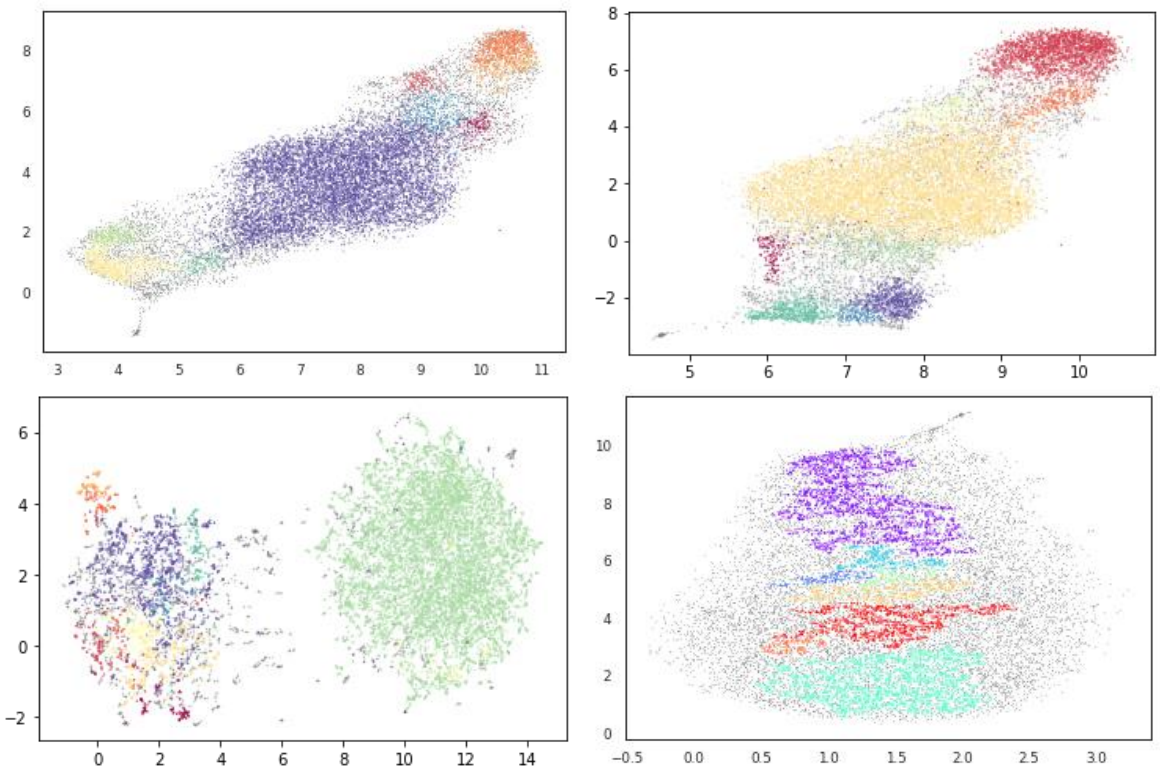


Figure 41: Clustering - VAE (β = 1) top left, BetaVAE (β = 5) top right, JointVAE lower left and BetaVAE (β = 250) lower right.

Chapter 5

Reflection and Conclusion

In this last chapter of the report we evaluate and reflect on the project as a whole, from the research and planning phase to the experimentation and development. In the end we also provide a conclusion and discuss possible next steps.

5.1 Planning

One of the main challenges with this project was its interdisciplinary nature. I started my scramble for a research topic well before Summer, during my industrial placement. Having decided on a broad topic, I started exploring this research space between urban planning and computation. I read through many papers, one more specific than the other in order to find a research angle that would inspire me. Ironically, I came across one of the main papers inspiring this project in a Facebook group I follow. This paper was "Urban Street Network Orientation" by Geoff Boeing, the creator of OSMNx. This is how I started experimenting with OSMNx and I knew I could generate an interesting dataset with it. Thus, I started the year with a concrete plan on how I am going to build a dataset and a clear idea of what I want to do with it, but with a much less clear plan for the specific implementation. At that time I wildly underestimated this implementation step, especially implementing and experimenting with the various different VAE architectures. I treated it as a software engineering exercise, but with a hacking approach. After some failed experimentation I saw this approach was not working, so I diverted my attention to learning more about the theory of autoencoders, as well as the current research on the topic. I also contacted experts in our school and managed to grab the attention of some, as I already mentioned in section 4.0. Starting to collaborate with experts brough some more guidance to my project, but it also changed it yet again to include disentanglement. My initial project idea was purely application oriented, but at the end of semester 1 it was starting to look more and more like a novelty research project. I took up an iterative and fluid approach to project planning, but I did not re-evaluate what I plan to achieve and how with every such project change. Instead, I let my own excitement about the project and what it could become get the better of me. The project has become a bit of everything, but nothing in particular.

The uncertainty due to COVID-19 only added to all of this. A lot of my research and time were allocated to testing different online and local machine learning environments, like Anaconda and Google Colab. This is also why I became reliant on the Reasoning & Explainable AI Lab group in order to perform my experiments as I initially planned them, unconstrained by resources. Up until about a month and a half before the deadline, I was committed to running my experiments remotely on my university machines.

I was committed to carrying out my project to be as true to my initial, perhaps naïve idea, as possible. Throughout the whole academic year, the project was constantly changing under different forces, both my own aspirations and the constraints of the unprecedented state of the current academic environment.

5.2 Conclusion

This project is an early exploration into studying urban form through deep learning models such as variational autoencoders. It tries to fill the gap between network science-based approaches to studying urban forms and the well accepted link between city livability and city characteristics such as urban greenspace, walkability, density and compactness, by generating a dataset, which visually captures all of these features. The encoded latent representations contain the information required to reconstruct the original images, effectively acting as a condensed representation of the urban form preserving a lot of its spatial and connectivity information. Due to the lack of well-defined similarity metrics of urban forms based on urban features mentioned above, our approach of using the learned latent representation as vectors of such features is a well-rounded technique for analytical studies of urban forms. We believe using high-dimensional urban imagery in this way could open new avenues in understanding the urban environment (Kempinska and Murcio, 2019).

We generated a dataset of 175,937 city and town images around the world but had to limit our experimentation to 15,782 cities. Our trained models successfully encoded the images into their low-dimensional urban vectors. The results showed they did not manage to disentangle the latent space, but rather only learned to preserve some spatial data from the original images and generate new synthetic urban form images with similar spatial properties to real world data samples. Due to the relatively small dataset size and short training periods, the models failed to reconstruct the images at a good resolution. In our future work we plan to experiment with the full size of the dataset. We also plan to include a controllable capacity parameter to the BetaVAE objective function (mentioned in section 4.2) and experiment with the different sizes of the latent bottleneck (currently at 50 dimensions). Our next step would be to experiment with a significantly smaller and hopefully disentangled latent space in order to see which generative factors the model would learn if we enforced it to learn a factorized representation of 5 latent variables, for example. In addition, our goal in future work is to make the training process more adaptive to real time changes. For example, changing the beta parameter in response to real-time training loss data to speed up the experimentation.

Finally, this was the largest and the most ambitious personal project I ever took on. Through purely independent study, and with only my mathematical background and little experience with machine learning to rely on, I learned so much about deep learning, data engineering, even using geographical information systems. I followed my own passion outside of the world of computer science and managed to translate that into an interesting project idea. By developing an interactive data visualization in three.js I incorporated another deep interest of mine into the project. Doing this, the project has become a medium for my own development as a computer science student. I managed to turn it into an accurate reflection of my current interests. This is something I am immensely proud of.

Bibliography

1. Burgess, C., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., Lerchner, A. and London, D. (n.d.). *Understanding disentangling in β-VAE*. [online]. Available at: https://arxiv.org/pdf/1804.03599.pdf.

2. A.M. Chirkin and Reinhard Koenig (2016). *Concept of Interactive Machine Learning in Urban Design Problems*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/303099121_Concept_of_Interactive_Machine_Learning_in_Urban_Design_Problems

3. Albert, A., Strano, E., Kaur, J. and González, M. (n.d.). . [online] . Available at: https://arxiv.org/pdf/1801.02710.pdf.

4. Arietta, S.M., Efros, A.A., Ramamoorthi, R. and Agrawala, M. (2014). City Forensics: Using Visual Elements to Predict Non-Visual City Attributes. *IEEE Transactions on Visualization and Computer Graphics*, [online] 20(12), pp.2624–2633. Available at: https://ieeexplore.ieee.org/document/6875954

5. Batty, M. and M. and Longley (1994). *Fractal Cities - A Geometry of Form and Function*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/30867789_Fractal_Cities_-_A_Geometry_of_Form_and_Function

6. Bérenger, V. and Verdier-Chouchane, A. (2007). Multidimensional Measures of Well-Being: Standard of Living and Quality of Life Across Countries. *World Development*, [online] 35(7), pp.1259–1276. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0305750X07000563?via%3Dihub

7. Boeing, G. (2018). The Morphology and Circuity of Walkable and Drivable Street Networks. [online] Available at: https://arxiv.org/ftp/arxiv/papers/1708/1708.00836.pdf
8. Boeing, G. (2020). Off the Grid… and Back Again? The Recent Evolution of American Street Network Planning and Design. *SSRN Electronic Journal*. [online] Available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3665445.

9. Boeing, G. 2018. Planarity and Street Network Representation in Urban Form Analysis. Environment and Planning B: Urban Analytics and City Science. doi:10.1177/2399808318802941

10. Deng, L. (2014). Deep Learning: Methods and Applications. *Foundations and Trends® in Signal Processing*, [online] 7(3-4), pp.197–387. Available at: https://www.nowpublishers.com/article/Details/SIG-039.

11. Deng, L. (2014). Deep Learning: Methods and Applications. Foundations and Trends® in Signal Processing, [online] 7(3-4), pp.197–387. Available at: https://www.nowpublishers.com/article/Details/SIG-039.

12. Dubey, A., Naik, N., Parikh, D., Raskar, R. and Hidalgo, C. (n.d.). *Deep Learning the City: Quantifying Urban Perception At A Global Scale*. [online]. Available at: https://arxiv.org/pdf/1608.01769.pdf

13. Dupont, E. (2018). *Learning Disentangled Joint Continuous and Discrete Representations*. [online] arXiv.org. Available at: https://arxiv.org/abs/1804.00104

14. Ecsdev.org. (2021). *View of A Critical Review of Urban Livability*. [online] Available at: https://ecsdev.org/ojs/index.php/ejsd/article/view/772/767

15. Emanuele Strano, Matheus Palhares Viana, Alessio Cardillo and Latora, V. (2012). *Urban street networks: a comparative analysis of ten European cities*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/232743659_Urban_street_networks_a_comparative_analysis_of_ten_European_cities

16. Geoff Boeing (2018). *Measuring the Complexity of Urban Form and Design*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/316711249_Measuring_the_Complexity_of_Urban_Form_and_Design

17. Geoff Boeing. (2017). *The Relative Circuity of Walkable and Drivable Urban Street Networks*. [online] Available at: https://geoffboeing.com/publications/relative-circuity-walkable-drivable-street-networks/ [Accessed 25 Feb. 2021].

18. Higgins, I. (2021). *beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework*. [online] OpenReview. Available at: https://openreview.net/forum?id=Sy2fzU9gl

19. J. Buhl, Gautrais, J., N. Reeves and Theraulaz, G. (2006). *Topological patterns in street networks of self-organized urban settlements*. [online] ResearchGate. Available

at:https://www.researchgate.net/publication/225755643_Topological_patterns_in_street_networks_ofself-organized_urban_settlements

20. Kempinska, K. and Murcio, R. (2019). Modelling urban networks using Variational Autoencoders. *Applied Network Science*, [online] 4(1). Available at: https://appliednetsci.springeropen.com/articles/10.1007/s41109-019-0234-0 [Accessed 6 Mar. 2021].

21. Liu, W., Yue, A., Shi, W., Ji, J. and Deng, R. (2019). An Automatic Extraction Architecture of Urban Green Space Based on DeepLabv3plus Semantic Segmentation Model. *2019 IEEE 4th International Conference on Image, Vision and Computing (ICIVC)*. [online] Available at: https://ieeexplore.ieee.org/document/8981007

22. M. Alvioli (2020). *Administrative boundaries and urban areas in Italy: A perspective from scaling laws*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/343152178_Administrative_boundaries_and_urban_areas_in_Italy_A_perspective_from_scaling_laws

23. Martino, N., Girling, C. and Lu, Y. (2021). Urban form and livability: socioeconomic and built environment indicators. *Buildings and Cities*, [online] 2(1), pp.220–243. Available at: https://journal-buildingscities.org/articles/10.5334/bc.82/

24. McInnes, L., Healy, J., Astels, S. (2016). *How HDBSCAN Works — hdbscan 0.8.1 documentation*. [online] Available at: https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

25. Moosavi, V. (2017). *Urban morphology meets deep learning: Exploring urban forms in one million cities, town and villages across the planet*. [online] arXiv.org. Available at: https://arxiv.org/abs/1709.02939

26. Naik, N., Philipoom, J., Raskar, R. and Hidalgo, C. (2014). Streetscore -- Predicting the Perceived Safety of One Million Streetscapes. *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. [online] Available at: https://ieeexplore.ieee.org/document/6910072

27. Oh, J. (n.d.). *Finding Main Streets: Applying Machine Learning to Urban Design Planning*. [online] . Available at: http://nyc.lti.cs.cmu.edu/IRLab/11-743s05/jeanoh/ir-report.pdf.

28. Paolo Crucitti, Latora, V. and Porta, S. (2006). *Centrality in Network of Urban Streets*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/7179558_Centrality_in_Network_of_Urban_Streets

29. Pearce, A. and Coenen, A. (2021). *Understanding UMAP*. [online] Available at: https://pair-code.github.io/understanding-umap/supplement.html

30. Porzi, L., Samuel Rota Bulò, Lepri, B. and Ricci, E. (2015). *Predicting and Understanding Urban Perception with Convolutional Neural Networks*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/305208849_Predicting_and_Understanding_Urban_Perception_with_Convolutional_Neural_Networks

31. Rocca, J. (2019). *Understanding Variational Autoencoders (VAEs) - Towards Data Science*. [online] Medium. Available at: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73#:~:text=variational%20autoencoders%20(VAEs)%20are%20autoencoders,order%20to%20ensure%20a%20better

32. Senbel, M., Giratalla, W., Zhang, K. and Kissinger, M. (2014). Compact Development without Transit: Life-Cycle GHG Emissions from Four Variations of Residential Density in Vancouver. *Environment and Planning A: Economy and Space*, [online] 46(5), pp.1226–1243. Available at: https://journals.sagepub.com/doi/10.1068/a46203

33. Strano, E., Viana, M., da Fontoura Costa, L., Cardillo, A., Porta, S. and Latora, V. (2013). Urban Street Networks, a Comparative Analysis of Ten European Cities. *Environment and Planning B: Planning and Design*, [online] 40(6), pp.1071–1086. Available at: https://journals.sagepub.com/doi/10.1068/b38216

34. UCL (2019). *CASA Working Paper 212*. [online] The Bartlett Centre for Advanced Spatial Analysis. Available at: https://www.ucl.ac.uk/bartlett/casa/publications/2019/sep/casa-working-paper-212

35. UN DESA | United Nations Department of Economic and Social Affairs. (2018). *68% of the world population projected to live in urban areas by 2050, says UN | UN DESA | United Nations Department of Economic and Social Affairs*. [online] Available at: https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html#:~:text=News-,68%25%20of%20the%20world%20population%20projected%20to%20live%20in,areas%20by%202050%2C%20says%20UN&text=Today%2C%2055%25%20of%20the%20world's,increase%20to%2068%25%20by%202050

36. Urban form and infrastructure: a morphological review. (n.d.). [online] . Available at: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/324161/14-808-urban-form-and-infrastructure-1.pdf.

37. Wijnands, J.S., Nice, K.A., Thompson, J., Zhao, H. and Stevenson, M. (2019). Streetscape augmentation using generative adversarial networks: Insights related to health and wellbeing. *Sustainable Cities and Society*, 49, p.101602.

38. Yamashita, R., Nishio, M., Do, R.K.G. and Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, [online] 9(4), pp.611–629. Available at: https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9

39. Yin, L. and Wang, Z. (2016). Measuring visual enclosure for street walkability: Using machine learning algorithms and Google Street View imagery. *Applied Geography*, [online] 76, pp.147–153. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0143622816304581

40. Zhou, B., Lapedriza, A., Xiao, J., Torralba, A. and Oliva, A. (2014). Learning Deep Features for Scene Recognition using Places Database. *Advances in Neural Information Processing Systems*, [online] 27. Available at: https://papers.nips.cc/paper/2014/hash/3fe94a002317b5f9259f82690aeea4cd-Abstract.html