Tampereen yliopisto

Ossi Kankainen

VISUALIZATION OF SEMANTIC SEGMENTATION NETWORKS

Bachelor's Thesis
Faculty of Engineering and Natural Sciences
Examiner: Prof. Jouni Mattila
April 2020

ABSTRACT

Ossi Kankainen: Visualization of Semantic Segmentation Networks
Bachelor's thesis
Tampere University
Bachelor's Degree Programme in Engineering Sciences
April 2020

The development of visualization methods for deep convolutional neural networks supports their design and helps in their adaption also to critical applications. Semantic segmentation has many such heavily regulated application areas such as medical imaging and autonomous vehicles. Thus, there is a clear need to find visualization methods that can be applied to neural network models used in semantic segmentation.

In this thesis, solutions are sought to this need by studying methods that have been used with generative models having a similar network structure than semantic segmentation models. Two different structures, autoencoder and adversarial networks, are commonly used in semantic segmentation models. They both utilize a concept of latent space that is a compact representation of data. Due to its compactness, the latent space is also useful in visualization of models. Based on literature can be find five different latent space visualization methods for generative models. In the experiments of this work those methods are applied to two different semantic segmentation models to see how they adapt for them.

Received results show how latent space projections from different dimensionality reduction techniques can be used to illustrate what features a semantic segmentation model uses when it forms clusters of data. In addition, the capability of the model to generalize for new data can be assessed based on the compactness of the projections. Examining the predicted output masks of the training samples is a good way to get an initial view of the model performance. Also new samples can be interpolated from the latent space. By observing feature changes in the outputs that model gives to them, one can obtain a more accurate view of how features change between different areas in the latent space. However, a problem is that semantic segmentation models do not force latent variables to be meaningful for data generation like generative networks do. For this reason, latent space is typically sparser which appeared in the experiments so that the nearest neighbour was same for many interpolation points. Thus, examining nearest neighbours turned out not to be a useful visualization method for semantic segmentation models. Also attribute vector arithmetic cannot be applied directly to semantic segmentation networks since the definition of attribute vector is not straightforward for them.

Keywords: neural networks, semantic segmentation, autoencoder, latent space, visualization

PREFACE

I want to thank my instructors Prof. Jouni Mattila and research assistant Eelis Peltola who offered this topic to me and have given me guidance throughout this project. In addition, I am also grateful for my family and friends who have supported me during the writing process.

Tampere, 28 April 2020

Ossi Kankainen

CONTENTS

1. INTRODUCTION 1
2. NEURAL NETWORK STRUCTURES USED IN SEMANTIC SEGMENTATION 2
    2.1 Autoencoder 3
    2.2 Adversarial networks 4
    2.3 Latent space 5
3. LATENT SPACE VISUALIZATION 7
    3.1 Dimensionality reduction techniques 7
        3.1.1 t-Distributed Stochastic Neighbor Embedding 8
        3.1.2 Principal Component Analysis 9
        3.1.3 Uniform Manifold Approximation and Projection 9
    3.2 Visualization methods 11
4. CODE EXPERIMENTS 15
    4.1 Setup 15
    4.2 Results 16
        4.2.1 Results for SegNet 16
        4.2.2 Results for DeepLabv3+ 19
5. CONCLUSION 22
REFERENCES 23
APPENDIX A: EXPERIMENT CODES 25

1. INTRODUCTION

Deep convolutional neural networks that are used in computer vision applications are complex models with several layers and millions of parameters. Because of this complexity, the understanding of their learning processes is challenging. [1] Nevertheless, it is very important as it helps in the design of models that has so far been often based on empirical trial and errors. Another reason why neural networks should be interpreted is that the lack of transparency is an obstacle to their adaption especially in critical and heavily regulated applications such as medical imaging or autonomous vehicles. [2]

The development of visualization methods that explain how deep convolutional models make their decisions has been increasing over the past few years [1]. In this thesis visualization methods are studied to models that are used in a specific image recognition problem, semantic segmentation. To these models there are not yet well-established visualization methods so there is a clear need to study this topic.

Section 2 introduces at first the task of semantic segmentation. After that, sections 2.1 and 2.2 present two neural network structures that are commonly used in semantic segmentation. Then, in section 2.3 a concept of latent space that both of those structures utilize is defined because turns out that it is very useful in the visualization of semantic segmentation networks. Section 3.1 goes over three dimensionality reduction techniques that are needed in some of five latent space visualization methods that are presented in section 3.2. Those methods are listed based on literature concerning generative models that have similar structure than commonly used semantic segmentation models. Thus, section 4 goes over experiments that were made to try how these methods adapt to segmentation networks. Section 4.1 presents the used experiment setup and section 4.2 shows the experiment results. At last, section 5 concludes the most important observations about the usability of the experimented methods for the visualization of semantic segmentation networks.

2. NEURAL NETWORK STRUCTURES USED IN SEMANTIC SEGMENTATION

Neural networks have been used in many image recognition problems, such as classification, detection and segmentation. Whole-image classification involves assigning a label to an image by predicting the presence of object classes in the image. Object detection expands classification by also locating the objects. The location of an object is shown by drawing a bounding box around the object. Semantic segmentation is a natural expansion of detection that involves predicting each pixel of an image either to some object class or as a background. Thus, semantic segmentation is absolutely more demanding task than classification or detection. [3] For example, if there is a test image that contains multiple cars on a road, neural network model that has been trained to classify images gives a label ‘car’ to the image. For the same test image, an object detection model draws a bounding box around each car and a semantic segmentation model predicts each pixel to some object class such as ‘car’ or ‘road’. Another example of a semantic segmentation can be seen in Figure 1.

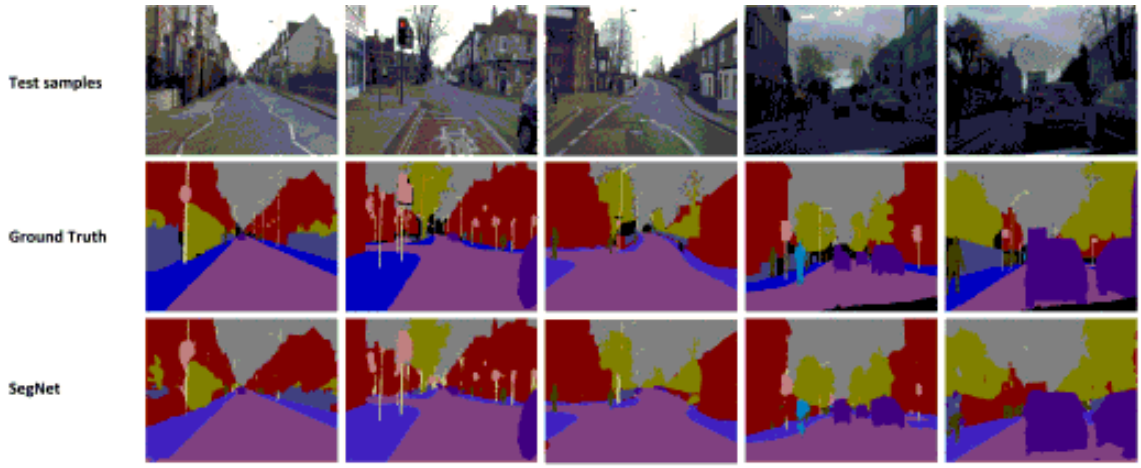

Figure 1. *An example of a semantic segmentation on road scene images. The top row shows five test images, the middle row shows manually given ground truth segmentation masks to them and the bottom row shows masks predicted by a SegNet model. Adapted from [4].*

In Figure 1 there are five test images on the top row. The second row shows manually given ground truth segmentation masks to these images, and the last row shows masks predicted by SegNet [4] that is an example of neural network model used in semantic

segmentation. From Figure 1 it can be seen that the result image of semantic segmentation has same width and height than the original sample image and that typically each object class is presented with a unique colour. For example, in these segmentation masks purple colour corresponds to the road and red colour corresponds to the buildings.

Semantic segmentation has many applications for example in medical imaging [5] and in autonomous vehicles [4]. The criticality and heavy regulation concerning these and many other applications, emphasizes the need of understanding the learning process of neural network models used in semantic segmentation. Thus, there is a clear need for visualization methods for semantic segmentation networks. However, understanding a model is not possible without having knowledge about its basic working principles. The next chapters 2.1 and 2.2 present two neural network structures; autoencoder and adversarial networks that are commonly used in semantic segmentation [4-7]. After that, the chapter 2.3 discusses in more detail a concept of latent space that both these network structures utilize.

2.1 Autoencoder

The idea of autoencoder was introduced in the 1980s when for example LeCun presented it in his thesis [8]. An autoencoder consists of two coupled models: an encoder and a decoder. The encoder is a feature-extracting function f_θ that transforms input data $\boldsymbol{x}$ to latent variables and delivers them to the decoder g_θ which produces reconstructions $\boldsymbol{r}$ from those variables. The autoencoder can thus be defined with the following equation

$$\boldsymbol{r} = g_\theta(f_\theta(\boldsymbol{x})). \tag{1}$$

The function composition emphasizes how the decoder takes the output of an encoder as its input. The autoencoder model is trained by minimizing reconstruction error between the inputs $\boldsymbol{x}$ and the reconstructions $\boldsymbol{r}$. [9] Models with autoencoder structure have been used also in supervised learning tasks such as semantic segmentation [4-6], even though originally autoencoder was designed to unsupervised tasks [9].

One later variation of the autoencoder is a variational autoencoder which was introduced for the first time by Kingma and Welling in 2013 [10]. It is originally motivated by generative modeling, meaning that a model is able to not only to reconstruct samples but also to generate new samples. This is possible because the variational autoencoder regularizes the training progress and thus ensures that the representations given by the encoder are meaningful for data generation. [11] In the meantime, the variational autoencoder also produces informative latent representations [12] because it learns a joint distribution

over all the input variables [11]. The variational autoencoder can be described also with probabilistic models as can be seen in Figure 2.

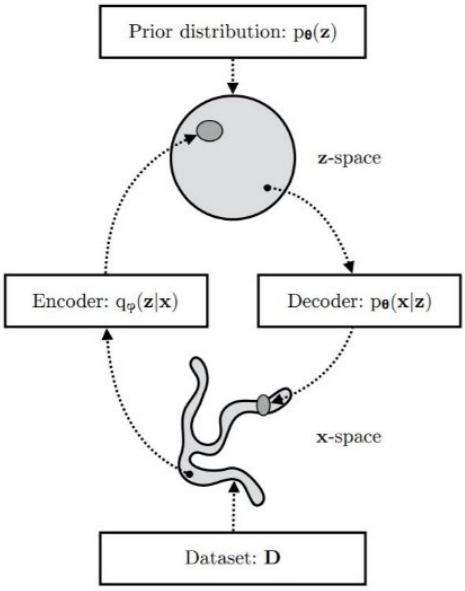

Figure 2. *A variational autoencoder is a process of probabilistic models that learns stochastic mappings between a dataset and a latent space. [11]*

The encoder is a parametric interference model $q_\varphi(\boldsymbol{z}|\boldsymbol{x})$ and its parameters φ are optimized so that the encoder approximates the posterior of the decoder. This means that

$$q_\varphi(\boldsymbol{z}|\boldsymbol{x}) \approx p_\theta(\boldsymbol{z}|\boldsymbol{x}). \tag{2}$$

The decoder meanwhile learns a joint distribution

$$p_\theta(\boldsymbol{x},\boldsymbol{z}) = p_\theta(\boldsymbol{z})p_\theta(\boldsymbol{x}|\boldsymbol{z}), \tag{3}$$

where $p_\theta(\boldsymbol{z})$ is a prior distribution over the latent space and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is a stochastic decoder. In that way, the variational autoencoder learns stochastic mappings between the observed data points and the latent space and after that it is able to generate new samples that look realistic. [11]

2.2 Adversarial networks

Generative adversarial networks was introduced by Goodfellow et al. in 2014. Like variational autoencoder, it was also originally motivated by generative modeling. [13] However, it has been applied also to semantic segmentation [7]. Another similarity with the autoencoder is that the generative adversarial network includes also two coupled models; a generator and a discriminator. Those models compete against each other. The

generator takes a point from a latent space as an input and generates a new sample. Meanwhile, the discriminator tries to distinguish whether an image is a real sample from training data, or the fake sample created by the generator. During a training generative adversarial network learns connections between the latent space points and the output images and after the training it is able to generate new realistic images. [14]

The training progress of generative adversarial network model can be described with a value function that the generator $G(\boldsymbol{z};\theta_g)$ tries to minimize and the discriminator $D(\boldsymbol{x};\theta_d)$ tries to maximize. This value function is

$$V(D,G) = E_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log\,(D(\boldsymbol{x})] + E_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}\big[\log\,(1 - D\big(G(\boldsymbol{z})\big))\big]\,, \qquad (4)$$

where both models D and G are multilayer perceptrons with corresponding variables, $p_{\boldsymbol{z}}(\boldsymbol{z})$ is a prior on input variables and $D(\boldsymbol{x})$ represents the probability that a sample comes from the training data. [13]

2.3 Latent space

As it has already been stated before, both variational autoencoder and generative adversarial networks utilize latent space when modeling the data [11,14]. Latent space is a continuous multi-dimensional vector space that forms a compact representation of data and has reduced dimensionality compared to the input space. Latent space helps a model to extract new more general features from the data. [12] An example of an autoencoder and a latent space can be seen in Figure 3. The data is compressed from the wide and thin input space to the narrow and thick latent space that is a bottleneck of the autoencoder.

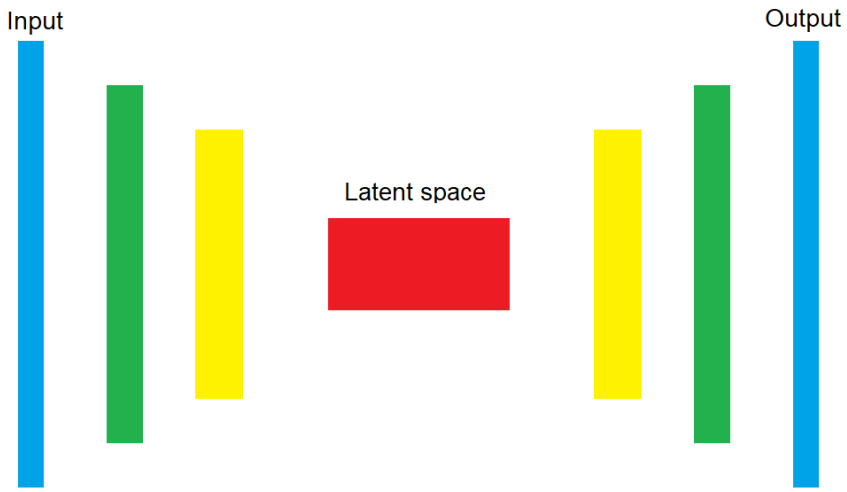

Figure 3. *A latent space is a bottleneck of an autoencoder that forms a compact representation of the input data.*

When a generative model learns to create new realistic samples it also produces a latent space representation that contains salient information about the input data [11]. This representation gives insights into the data and can reveal relationships that are ‘latent’ in the input space. The compactness of the latent space makes it also useful for the visualization of neural network model. [12] This thesis focuses on how semantic segmentation networks can be interpreted with the latent space visualization methods.

3. LATENT SPACE VISUALIZATION

Section 3.2 presents five latent space visualization methods for generative models based on literature. However, some methods demand that we have a way to plot multi-dimensional latent spaces with two-dimensional graphs. For this reason, at first section 3.1 discusses more about different dimensionality reduction techniques.

3.1 Dimensionality reduction techniques

Dimensionality reduction techniques are used in the visualization of latent space to reduce its dimensionality to two dimensions [12]. This is necessary so that visualization results can be plotted with 2D scatter plots like in Figure 4 that visualizes the latent space of an autoencoder trained with the MNIST dataset. In order to make visualization possible, the dimensionality of each latent vector was reduced using Principal Component Analysis (PCA). [15]

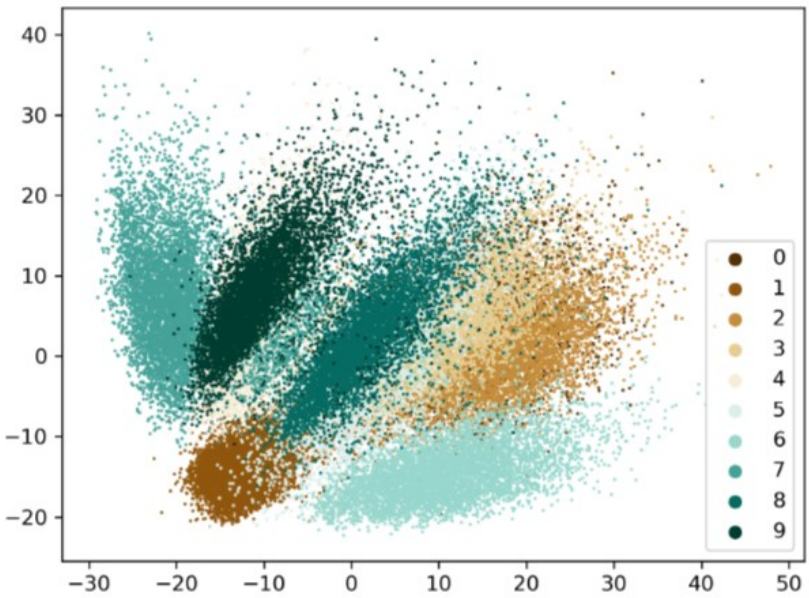

Figure 4. *The latent space of an autoencoder trained on the MNIST dataset visualized with 2D scatter plot. In order to visualize the latent space its dimensionality was reduced using principal component analysis. [15]*

In addition to PCA, two other dimensionality reduction techniques; t-Distributed Stochastic Neighbor Embedding (t-SNE) and Uniform Manifold Approximation and Projection (UMAP) are covered in this thesis. The following sections 3.1.1 – 3.1.3 present the basic principles of these techniques.

3.1.1 t-Distributed Stochastic Neighbor Embedding

Laurens van der Maaten and Geoffrey Hinton presented t-Distributed Stochastic Neighbor Embedding (t-SNE) in 2008 [16]. It is a non-linear dimensionality reduction technique that models neighbour samples in the original high-dimensional space close to each other also in the low-dimensional space. Non-linearity means that it is able to highlight cluster structures in the data, but it cannot preserve linear relationships between the data points. [12]

The first step of t-SNE algorithm is to calculate conditional probabilities $p_{j|i}$ between high-dimensional data points x_i and x_j with the following equation

$$p_{j|i} = \frac{\exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}\right)}, \tag{5}$$

where σ_i is the variance of the Gaussian normal distribution centered over x_i. These probabilities describe how similar data points are with each other. If data points are nearby, the conditional probability between them is relatively high but for separated data points it is almost infinitesimal. To prevent problems with outlier samples, joint probabilities p_{ij} are defined to be symmetrized conditional probabilities by setting $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$. Similar joint probabilities q_{ij} are calculated also to low-dimensional data points y_i and y_j with the following equation

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}. \tag{6}$$

After that t-SNE finds out the locations of data points in the low-dimensional space by minimizing the mismatch between these two joint probability distributions P and Q. This happens by minimizing Kullback-Leibler divergence

$$C = KL(P\|Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{7}$$

with a gradient

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) \left(1 + \|y_i - y_j\|^2\right)^{-1}. \tag{8}$$

During the training, a gradient descent is updated by adding the calculated gradient with some learning rate coefficient and a relatively large momentum term to the previous value of gradient descent. [16]

3.1.2 Principal Component Analysis

Principal Component Analysis (PCA) is a traditional and commonly used dimensionality reduction technique. It tries to find new variables, the principal components, that are linear functions of the original data variables and that minimize loss of information. This means that those variables should maximize the variance and they should be uncorrelated with each other. [17] PCA is a linear technique which means that unlike t-SNE it is able to preserve linear relations among the data but on the other hand it cannot highlight cluster structures as well [12].

Finding the principal components reduces to solving an eigenvalue problem. A data matrix $\boldsymbol{X}$ has n rows and p columns. PCA is looking for a linear combination of the columns of this matrix that maximizes variance. Such linear combination can be described with a matrix multiplication $\boldsymbol{Xa}$, where $\boldsymbol{a}$ is a p-dimensional vector of constants. The variance of this linear combination is defined in the following way

$$\mathrm{var}(\boldsymbol{Xa}) = \boldsymbol{a}'\boldsymbol{Sa}, \tag{9}$$

where $\boldsymbol{S}$ is the covariance matrix related to the dataset and $'$ denotes transpose. To find a well-defined solution for $\boldsymbol{a}$ that maximizes the variance, some additional restrictions are needed. The most common one is to suppose that $\boldsymbol{a}$ is a unit-norm vector. Then we can write the following equation

$$\boldsymbol{a}'\boldsymbol{Sa} - \lambda(\boldsymbol{a}'\boldsymbol{a} - 1) = \boldsymbol{0} \Leftrightarrow \boldsymbol{Sa} - \lambda\boldsymbol{a} = \boldsymbol{0} \Leftrightarrow \boldsymbol{Sa} = \lambda\boldsymbol{a}, \tag{10}$$

where $\boldsymbol{a}$ must be an eigenvector of the covariance matrix $\boldsymbol{S}$ and λ is the corresponding eigenvalue. This shows that the principal components can be found by solving the eigenvectors of the covariance matrix. As the covariance matrix is a pxp-dimensional real symmetrical matrix, it has exactly p eigenvalues. The dimensionality reduction happens by taking the first q principal components with the highest eigenvalue out of the original amount p. [17]

3.1.3 Uniform Manifold Approximation and Projection

Uniform Manifold Approximation and Projection (UMAP) is a more recent dimensionality reduction technique than t-SNE or PCA as it was for the first time introduced by McInnes et al. in 2018 [18]. It is a manifold learning technique which makes it also non-linear like t-SNE and thus it highlights cluster structures among the data [12]. However, it is also able to preserve the global structure of data better than the t-SNE and its better computational performance reduces running times and allows larger data sets [18].

UMAP algorithm consists of two phases. The first phase constructs a particularly weighted k-neighbour graph and the second phase transforms this graph to a low dimensional layout. At first a set of the k nearest neigbours is computed to each data point x_i in an input dataset $X = \{x_1, \dots, x_n\}$ using some dissimilarity metric d. A hyper-parameter k defines the amount of neighbours. At next, ρ_i is defined to each point as follows

$$\rho_i = min\left\{d\left(x_i, x_{i_j}\right) \middle| \, 1 \le j \le k, \; d\left(x_i, x_{i_j}\right) > 0\right\}, \tag{11}$$

and σ_i is set to be such value that the following equation

$$\sum_{j=1}^{k} \exp\left(\frac{-\max\left(0, \; d\left(x_i, x_{i_j}\right) - \rho_i\right)}{\sigma_i}\right) = \log_2(k) \tag{12}$$

is true. After these definitions, it is possible to define a weighted directed graph $\bar{G} = (V, E, \omega)$. The vertices $\bar{G}$ and V are simply the input dataset X, the set of directed edges is $E = \left\{\left(x_i, x_{i_j}\right) \mid 1 \le j \le k, \; 1 \le i \le N\right\}$ and the weight function is

$$\omega\left(\left(x_i, x_{i_j}\right)\right) = \exp\left(\frac{-\max\left(0, \; d\left(x_i, x_{i_j}\right) - \rho_i\right)}{\sigma_i}\right). \tag{13}$$

The symmetric adjacency matrix B of UMAP graph G can then be described with a formula

$$B = A + A^T - A \circ A^T, \tag{14}$$

where A is the weighted adjacency matrix of $\bar{G}$ and $\circ$ is a pointwise product. [18]

To compute a low dimensional layout, UMAP utilizes a force directed graph layout algorithm that applies a set of attractive forces along edges and a set of repulsive forces along vertices. The attractive force is given by the following equation

$$\frac{-2ab\left\|\boldsymbol{y_i} - \boldsymbol{y_j}\right\|_2^{2(b-1)}}{1 + \left\|\boldsymbol{y_i} - \boldsymbol{y_j}\right\|_2^2} \omega\left(\left(x_i, x_j\right)\right)\left(\boldsymbol{y_i} - \boldsymbol{y_j}\right), \tag{15}$$

where $\boldsymbol{y_i}$ and $\boldsymbol{y_j}$ are coordinates of two vertices and a ja b are hyper-parameters. The repulsive force is determined as follows

$$\frac{b}{\left(\varepsilon + \left\|\boldsymbol{y_i} - \boldsymbol{y_j}\right\|_2^2\right)\left(1 + \left\|\boldsymbol{y_i} - \boldsymbol{y_j}\right\|_2^2\right)}\left(1 - \omega\left(\left(x_i, x_j\right)\right)\right)\left(\boldsymbol{y_i} - \boldsymbol{y_j}\right), \tag{16}$$

where ε is a small number that prevents division by zero. [18]

3.2 Visualization methods

Liu et al. have made a literature review of latent space visualization methods as a part of their article [12]. This section presents five different visualization activities based on their results and also considers the usability of these activities for semantic segmentation models. Liu et al. analysed altogether 78 research papers of which 54 concerned generative modeling. The latent space visualization activities for generative models that they identified are:

- Viewing reconstructed samples
- Visualizing distributions
- Viewing interpolation results
- Examining the nearest neighbours
- Performing attribute vector arithmetic. [12]

Similar methods have also been represented in other research papers [15,19].

Viewing reconstructed examples is a simple way to see how visually good images a model is able to produce since the results can be compared to the corresponding original images. This method cannot be used directly with semantic segmentation models as they do not try to reconstruct input images. However, similar idea can be applied by comparing the predicted segmentation masks to the ground truth masks.

Visualizing distributions is also a good way to get initial understanding of the model [15]. Results given by dimensionality reduction techniques can reveal cluster structures and linear relationships among the latent variables which gives information about the ways in which the model processes data [12]. Distribution visualizations show also how sparsely the latent space of the model is populated which effects on its ability to generalize what it has learned [15]. Distribution visualization techniques can be applied directly also to semantic segmentation models.

Viewing interpolation results and examining the nearest neighbours expand the task of viewing reconstructed examples by creating new unseen results from the latent space. These tasks show if a model is able to create images that are indistinguishable from the training images. They also reveal how smoothly output features change in the latent space. [12] In principle these methods can be applied to semantic segmentation networks but it is not guaranteed that the results are reasonable. This is because non-variational autoencoders, that are used as segmentation models, do not ensure that the

latent space representations are useful for data generation. However, autoencoders have still given surprisingly good interpolation results. [15]

Latent space interpolation is performed by following a path between two latent points $\boldsymbol{z}_a$ and $\boldsymbol{z}_b$ and constructing samples at regular intervals [12]. Often the path is chosen to be linear since it is easy to understand and implement but also spherical paths are used especially for high-dimensional latent spaces [19]. Figure 5 shows an example of latent space interpolation. It includes three interpolations that were performed using spherical path [19] and they demonstrate how features change smoothly in the latent space.

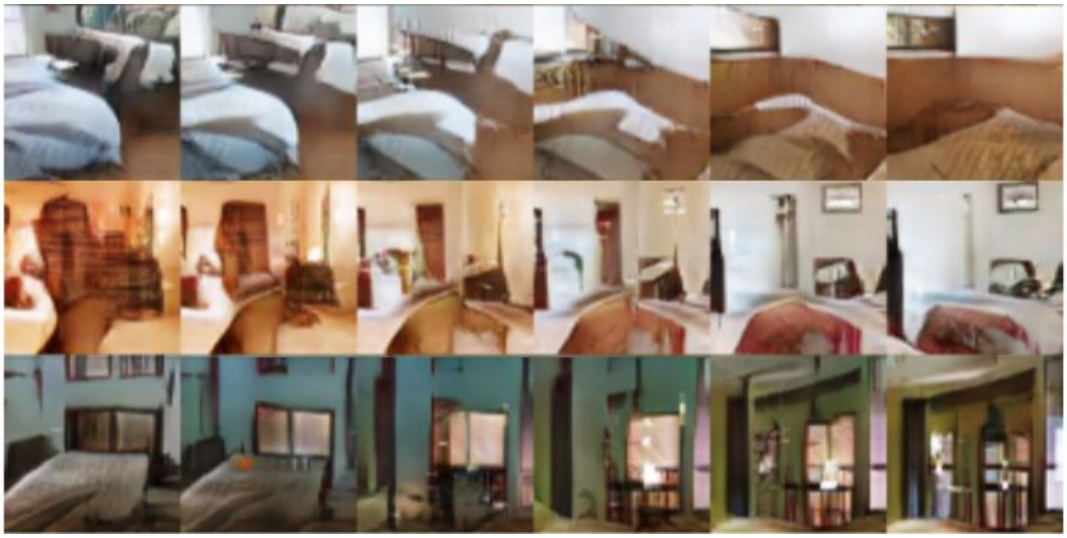

Figure 5. *An example of interpolation in latent space. Each row contains one interpolation that is performed with spherical path in a high-dimensional latent space. Adapted from [19].*

Linear interpolation line can be defined using vectors. The latent space is a n-dimensional vector space, so each latent point can be seen as a n-dimensional vector. Figure 6 illustrates how a linear interpolation line between latent space points $\boldsymbol{A}$ and $\boldsymbol{B}$ is actually a vector $\boldsymbol{AB}$ that can be calculated by subtracting the vector $\boldsymbol{A}$ from the vector $\boldsymbol{B}$. Any interpolation point $\boldsymbol{P}$ that is p percent of the way from $\boldsymbol{A}$ to $\boldsymbol{B}$ can be presented with the following equation

$$\boldsymbol{P} = \boldsymbol{A} + \alpha(\boldsymbol{B} - \boldsymbol{A}) \Leftrightarrow \boldsymbol{P} = \alpha\boldsymbol{B} + (1 - \alpha)\boldsymbol{A}, \tag{17}$$

where α is equal to $p/100$. This shows that any latent space interpolation point is a linear combination between the tail and head points of corresponding interpolation line vector. Typically, interpolation points are taken on regular intervals so that α gets evenly spaced values over its interval $[0, 1]$. The nearest neighbours of a point $\boldsymbol{P}$ can be found simply by calculating an Euclidean distance between this point and each training data point in the latent space [20].

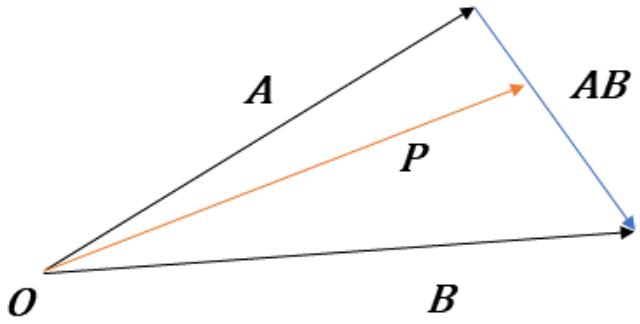

Figure 6. *Any interpolation point $\boldsymbol{P}$ that is located on a linear interpolation line $\boldsymbol{AB}$ is a linear combination of vectors $\boldsymbol{A}$ and $\boldsymbol{B}$.*

Performing attribute vector arithmetic is a good way to demonstrate that a latent space is able to produce samples with new attribute combinations [12]. A well-known example of attribute vector arithmetic with linguistic models shows that if vectors $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ correspond latent space attributes king, man and woman, then the result vector of calculation $\boldsymbol{A} - \boldsymbol{B} + \boldsymbol{C}$ should be close to queen [21]. This method has also been applied to images as can be seen in Figure 7. If the top left image is a source A and the other two corners are targets B and C, then the result image of operation $(B + C) - A$ can be seen in the bottom right image [19].

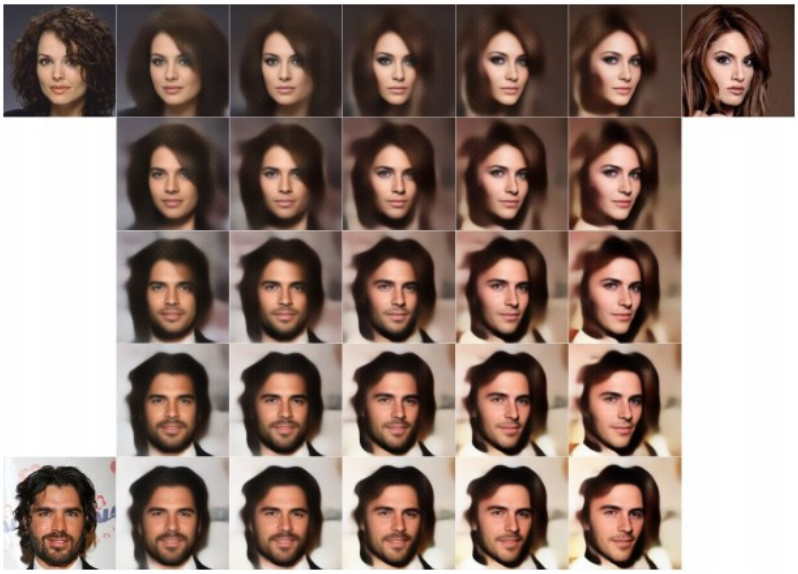

Figure 7. *Attribute vector arithmetic applied to images. The top left image is a source A and the other two corner images are attribute targets B and C. The result of operation $(B + C) - A$ is shown in the bottom right image. [19]*

Semantic segmentation networks have same issues with attribute vector arithmetic as with other generative tasks since their latent space representation is not optimized for data generation. Another problem is that semantic segmentation outputs have typically similar features with each other, but the difference comes from the locations of the objects. For this reason, the definition of attribute vector is not straightforward for semantic segmentation models and it is not covered in this thesis. Section 4 presents the code experiments that were made to the other four methods to see how they adapt for non-variational autoencoder models that are designed for semantic segmentation.

4. CODE EXPERIMENTS

The goal of code experiments in this thesis is to examine how latent space visualization methods that are commonly used with generative models apply to semantic segmentation models. The experiments focus on autoencoders that have similar structure than generative variational autoencoder models, but their latent space is not optimized for data generation [15]. Another difference is that semantic segmentation models use supervised learning, meaning that they learn mappings between the inputs and outputs of the training data while generative models learn unsupervised a joint distribution over all the inputs [11].

The previous section presented five different latent space visualization activities of which four were used in experiments. Latent space distributions were visualized with the results of two dimensionality reductions techniques, t-SNE and PCA, so that both linear and non-linear relationships of the latent space came out. The idea of viewing reconstructed examples was applied by viewing the predicted masks of the input data points. This task was combined with two other activities by constructing linear interpolations between reconstructed samples and showing their nearest neighbours from the original dataset. The following sections discuss the setup used in experiments and the received results in more detail.

4.1 Setup

Used codes were written in Python programming language because several useful machine learning libraries and frameworks have been developed for it. The experiment codes are attached in Appendix 1.

The experiments were made with two models that both have an autoencoder structure. The first model was SegNet [4] that is implemented in Image Segmentation Keras framework [22]. This model was chosen because it is a well-known network that was originally developed for road scene applications [4] and thus it should work quite well for the used dataset that consists of road scene images. The dataset includes 367 train images and 101 test images with corresponding ground truth masks and it can be downloaded from the model implementation web page [22].

The subject of this thesis is motivated by Eelis Peltola's ongoing master thesis concerning autonomous rough terrain mobile robotics in which he uses a network based on DeepLabv3+ [6]. Therefore, this model was chosen to be the second experimented

model. DeepLabv3+ has also an existing GitHub implementation [23] that can be used to build and train the model. The main difference between SegNet and DeepLabv3+ is, that unlike SegNet which utilizes only the output of encoder and max-pooling indices in decoding [4], DeepLabv3+ utilizes also the low-level features given by atrous convolution layers and concatenates them with the corresponding encoder features [6].

When a model is trained, its encoder and decoder parts are saved to different models. Thus, the encoder can be used to transform input data to latent variables and the decoder can be used to create output images from these variables. For the models that utilize also low-level features in decoding, it is important to define which input variables are given to the convolutional layers because they have a significant effect on the generated outputs. In this thesis, input variables were interpolated from the original input space in a similar way than latent variables were interpolated from the latent space. The more detailed explanation of interpolation in a vector space can be found from the section 3.2. In a case of DeepLabv3+ actually three different models were saved. The first one included convolutional layers and it could be used as an input to the encoder and the decoder.

4.2 Results

Results of the experiments for SegNet and DeepLabv3+ are presented and analysed in the following sections. The goal is to interpret and compare the results and based on those interpretations make conclusions about the suitability of these visualization methods for semantic segmentation networks.

4.2.1 Results for SegNet

Latent space of SegNet model was examined with 367 train images of the dataset. Figure 8 shows a 2D scatter plot of latent space that was obtained by reducing all encoded latent vectors to two dimensions with t-SNE algorithm. Close up view of one subpart in Figure 8 shows that all points are annotated with a sequence number corresponding to the original image. For example, a point number zero in Figure 8 was obtained by encoding the first image in the dataset to a latent vector and by reducing it to two dimensions. All the following figures can be zoomed in the electronic version of this thesis so that they can be examined in more detail.

From Figure 8 it can be seen that the latent variables are clustered so that neighbour samples in the original image space are close to each other also in the latent space representation. For example, images 0-61 that are taken from similar locations form a

cluster to the lower left corner of the figure. Images 281-366 are also logically on the left as they are also taken from a road that passes between buildings like images 0-61.

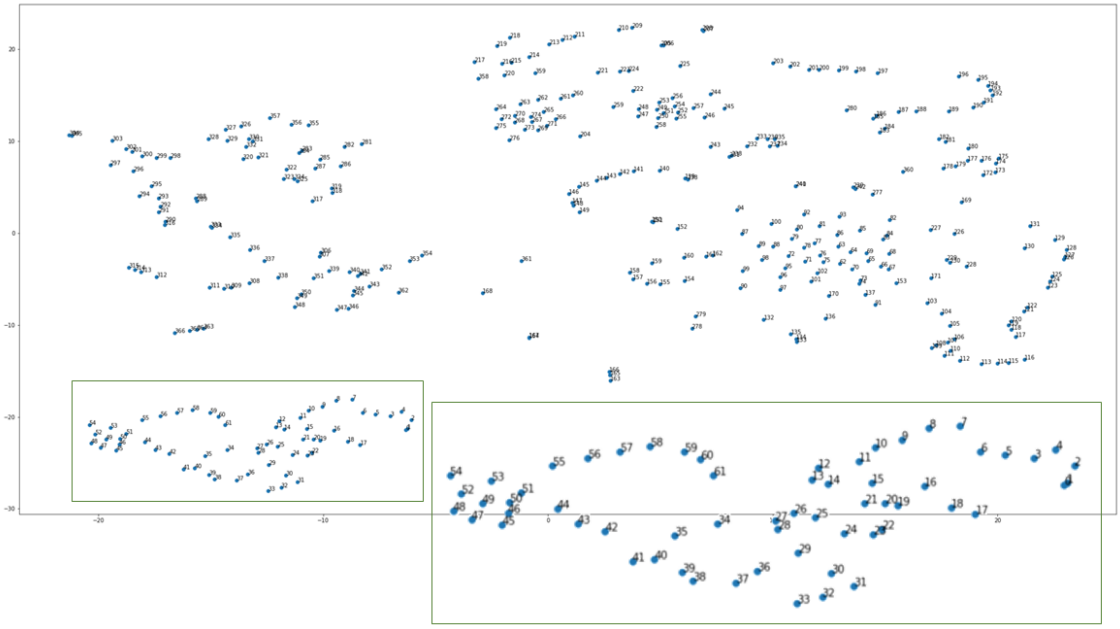

Figure 8. *Latent space of SegNet visualized with t-SNE projection.*

However, the clusters in Figure 8 are sparse and they are distributed over a large area. There are many 'chains' of points in the t-SNE representation but the distance to the next point after the chain is often huge. This indicates that the model has learned the mappings of the training data but it might not be able to generalize well for unseen data since there are lot of unknown areas in the latent space [15]. The disjointedness of the latent space comes out also in Figure 9 that shows a visualization of the same latent variables with PCA projection.

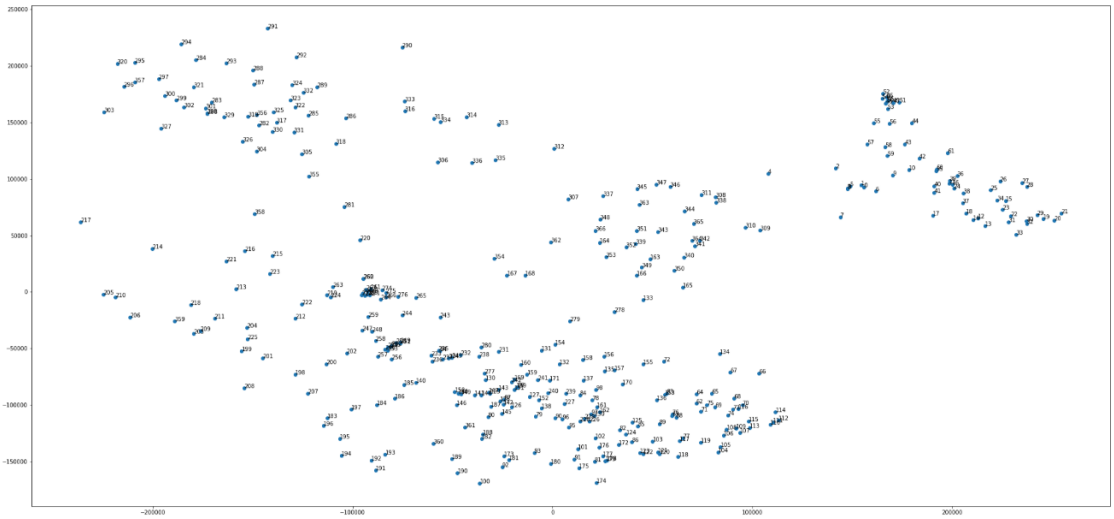

Figure 9. Latent space of SegNet visualized with PCA projection.

The coordinate values of the PCA projection vary in a large range from -236,000 to 254,400 and from -169,000 to 233,000. Thus, the latent space contains inevitably multiple locations where the model does not know how to make a prediction from the point. This problem should come out also in the next experiments where new samples are generated from the latent space.

Figure 10 shows interpolation results that are generated from the latent space and the nearest neighbours from the training data points for those interpolations. The first row contains interpolation results 1–6 and the second row shows their nearest neighbours. The third and fourth row show interpolation results 7–12 and their nearest neighbours. The tail and head points of an interpolation line vector are the data points 138 and 38. From their predicted masks it can be seen that the model is able to detect some objects from the images, but the object boundaries are unsharp.

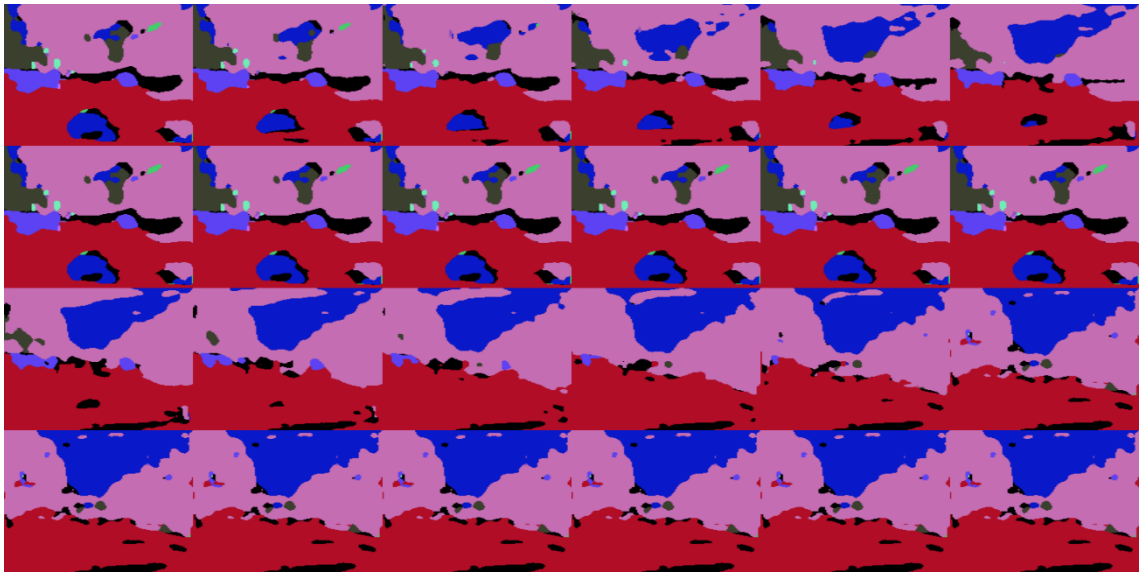

Figure 10. *Interpolation results and their nearest neighbours that are generated from the latent space of SegNet. The top row shows interpolation results 1-6 and the second row shows their nearest neighbours from the training data points. In a same way, the third and fourth row contain interpolation results 7-12 and their nearest neighbours. The tail and head points of interpolation line vector are the data points 138 and 38.*

It seems that the model is also able to generate new samples that cannot be distinguished from the training images. However, since the latent space is so sparse any other training samples than the head and the tail point do not appear in the nearest neighbours. The interpolation results are surprisingly good considering the disjointedness of the latent space. The feature changes are predictable and smooth. For example, the blue area at the top centre enlarges and the other blue area in the bottom centre changes to black and finally disappears during the interpolation. This indicates that latent space interpolations could be useful in the visualization of semantic segmentation networks.

4.2.2 Results for DeepLabv3+

For comparison, the same visualization methods were also experimented with another model, DeepLabv3+. Figure 11 shows a t-SNE representation over the latent variables for this model that was constructed in a similar way than in the previous section.

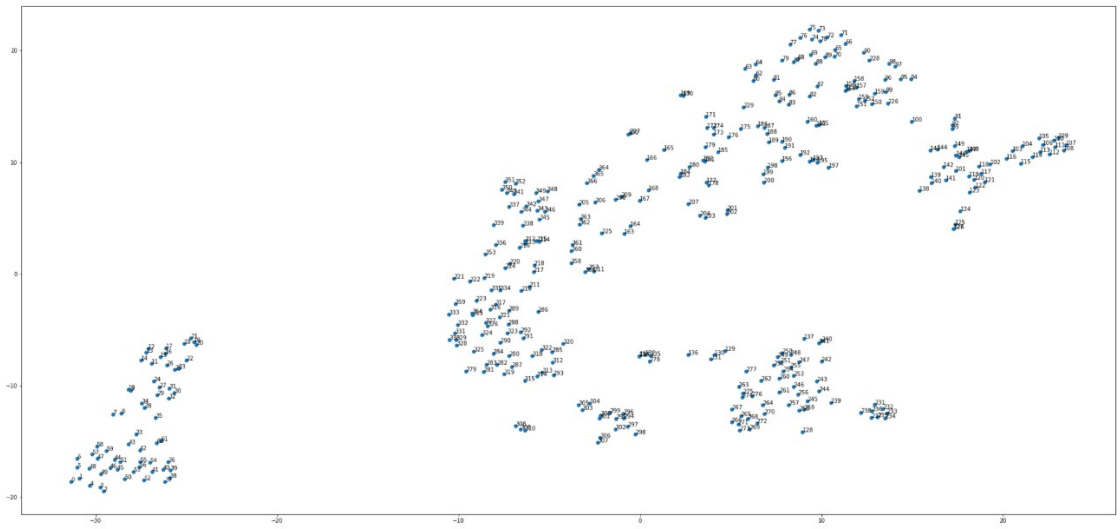

Figure 11. *Latent space of DeepLabv3+ visualized with t-SNE projection.*

Similar samples are again clustered near to each other which shows that both models utilize similarities to model the data. This time the latent space is more compact, and the latent points are centered around the origin, except the points 0–61 that form a separated cluster to the bottom left corner. However, the other clusters are even partly overlapping each other which reduces the amount of unknown areas in the latent space. It seems that all training images that contain a road passing between buildings are now in the bottom part of Figure 11. A PCA projection over the latent space is shown in Figure 12.

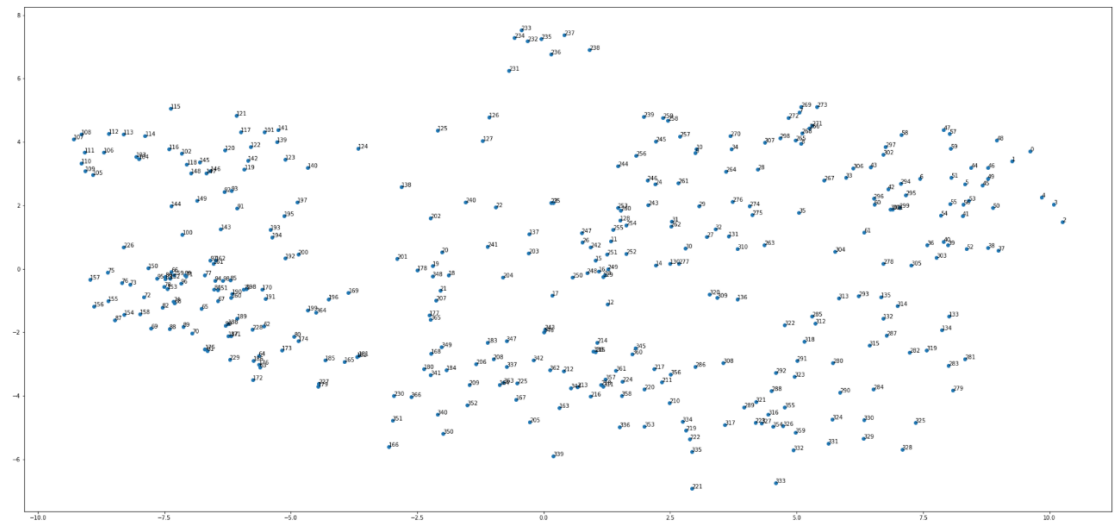

Figure 12. *Latent space of DeepLabv3+ visualized with PCA projection.*

From Figure 12 it can be seen that compared to the results of t-SNE, the coordinate values of PCA projection vary in a much smaller range that is now from -9.3 to 10.3 and from -6.9 to 7.5. This means that there are less areas in the latent space where the model does not know how to make a prediction. Based on these results, the DeepLabv3+ model should have better ability to generate new samples than the SegNet model.

Figure 13 shows generated latent space interpolation results and their nearest neighbours from the original image space for the DeepLabv3+ model. Images are organized in a same way than in Figure 10 so that the first and the third row include interpolation results and the second and the fourth row their nearest neighbours. The same tail and head points 138 and 38 than in the previous section are used to define an interpolation line vector. From the predicted masks of these points that belong to the original data space, it can be seen that the trained DeepLabv3+ model was able to create sharper boundaries between different objects than the SegNet model. There is for example a recognizable shape of tree in some of these constructed masks.

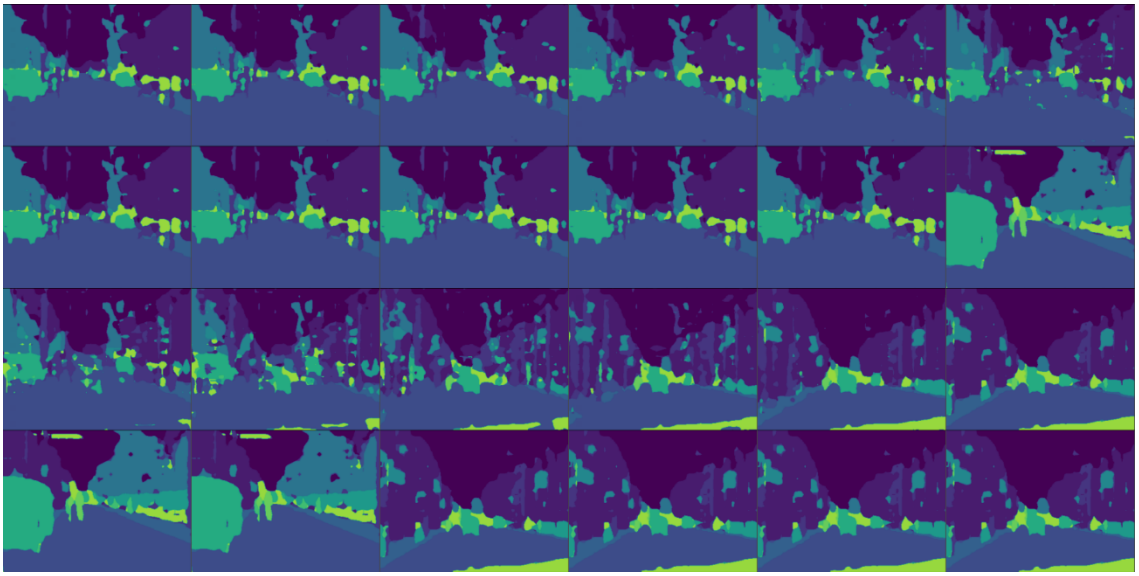

Figure 13. *Interpolation results and their nearest neighbours that are generated from the latent space of DeepLabv3+. Images are organized in a same way than in Figure 10. The top row shows interpolation results 1-6 and the second row shows their nearest neighbours from the training data points. In a same way, the third and fourth row contain interpolation results 7-12 and their nearest neighbours. The tail and head points of interpolation line vector are the data points 138 and 38.*

The examination of the nearest neighbours shows that the low-level features have a large impact on created outputs and thus samples 6–8 stand out of their nearest neighbour that was found outside of the interpolation path. All generated samples look realistic but even though the latent space is more compact, only three different neighbours are

still found for 12 interpolation points. Thus, it seems that the latent space of non-variational autoencoders is often too sparse so that the analysis of the nearest neighbour could be used in their visualization similarly than to generative models.

The generated interpolation results are again smooth and predictable. For example, the top left part changes gradually from green to dark blue and a yellow shred shows up at the bottom part. However, the interpolations seem to behave in a quite similar way for both models, which means that the distribution of latent space does not provide completely reliable information about when an autoencoder model is able to generate new samples, as based on distribution visualizations, the assumption was that DeepLabv3+ should produce distinctly better results. At least interpolations can probably be useful for the visualization of feature changes on small local areas of latent space where there are not large unknown areas. With models that utilize low-level features for decoding, sampling of them should be considered carefully because it has a very significant effect on the generated outputs.

5. CONCLUSION

At this thesis, visualization methods for semantic segmentation networks were sought by studying methods that have been used with generative models having a similar structure than semantic segmentation models. All these models utilize a latent space when modeling data. The latent space is a compact representation of data and due to its compactness, it is suitable also for the visualization of models. Based on literature was found five different latent space visualization methods for generative models that were examined with semantic segmentation networks to see how they adapt for them.

Results show that the projections of latent space with reduced dimensionality can reveal how a model uses clusters to understand data. This information can be used to show what features the model extracts from an input image so that it is able to segment it. The distribution visualizations of latent space also show how sparse the latent space of the model is which effects on its ability to generalize the learned mappings for new inputs. Examining the output masks of training data points is a good way to get an initial view of the model performance. Latent space interpolations proved out to be a good way to show how features change between different areas in latent space. However, semantic segmentation models do not regularize latent space representations to be meaningful for data generation like generative models do. For this reason, the results are not always necessarily good, and a problem is that there is not a reliable way to predict the quality of results as even the disjointedness of the latent space did not seem to always mean poor results. When generating outputs from new points with models that utilize low-level features in decoding, it is important to consider also the sampling of those features carefully since they have a significant effect on the result. Examining the nearest neighbours of the generated points did not work well for these semantic segmentation models since the latent space was so sparse that multiple interpolation points had the same nearest neighbour. Also, attribute vector arithmetic could not be applied directly to semantic segmentation models since the definition of attribute vector is not straightforward for them.

Both experimented models had an autoencoder structure. This work could be continued by applying same visualization methods also for models that belong to the other main semantic segmentation network type, adversarial networks. In a larger framework, it would be interesting to see what results can be achieved by visualizing other parts of semantic segmentation networks such as the convolutional layer weights of the encoder.

REFERENCES

[1] A. Mahendran, A. Vedaldi, Visualizing Deep Convolutional Neural Networks Using Natural Pre-images, International Journal of Computer Vision, Vol. 120, No. 3, 2016, pp. 233–255. Available (Accessed Apr. 23, 2020): https://arxiv.org/abs/1512.02017.

[2] L.M. Zintgraf, T.S. Cohen, T. Adel, M. Welling, Visualizing Deep Neural Network Decisions: Prediction Difference Analysis, International Conference on Learning Representations, 2017. Available (Accessed Apr. 23, 2020): https://arxiv.org/abs/1702.04595.

[3] M. Everingham, S. Eslami, L. Van Gool, C. Williams, J. Winn, A. Zisserman, The Pascal Visual Object Classes Challenge: A Retrospective, International Journal of Computer Vision, Vol. 111, No. 1, 2015, pp. 98–136. Available (Accessed Feb. 14, 2020): http://host.robots.ox.ac.uk/pascal/VOC/pubs/everingham14_bak.pdf.

[4] V. Badrinarayanan, A. Kendall, R. Cipolla, SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 39, No. 12, 2017, pp. 2481–2495. Available (Accessed Feb. 14, 2020): https://arxiv.org/abs/1511.00561.

[5] O. Ronneberger, P. Fischer, T. Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, International Conference on Medical Image Computing and Computer-Assisted Intervention, 2015, pp. 234–241. Available (Accessed Feb. 14, 2020): https://arxiv.org/abs/1505.04597.

[6] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, A. Hartwig, Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation, European Conference on Computer Vision, 2018, pp. 833–851. Available (Accessed Feb. 19, 2020): https://arxiv.org/abs/1802.02611.

[7] P. Luc, C. Couprie, S. Chintala, J. Verbeek, Semantic Segmentation using Adversarial Networks, Conference on Neural Information Processing Systems, 2016. Available (Accessed Apr. 20, 2020): https://arxiv.org/abs/1611.08408.

[8] Y. LeCun. Modeles connexionistes de l'apprentissage, Universite P. et M. Curie (Paris 6), 1987.

[9] Y. Bengio, A. Courville, P. Vincent, Representation Learning: A Review and New Perspectives, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 35, No. 8, 2013, pp. 1798–1828. Available (Accessed Apr. 20, 2020): https://arxiv.org/abs/1206.5538.

[10] D. Kingma, M. Welling, Auto-Encoding Variational Bayes, International Conference on Learning Representations, 2013. Available (Accessed Mar. 19, 2020): https://arxiv.org/abs/1312.6114.

[11] D. Kingma, M. Welling, An Introduction to Variational Autoencoders, Foundations and Trends in Machine Learning, Vol. 12, No. 4, 2019, pp. 307–392. Available (Accessed Feb. 14, 2020): https://arxiv.org/abs/1906.02691.

[12] Y. Liu, E. Jun, Q. Li, J. Heer, Latent Space Cartography: Visual Analysis of Vector Space Embeddings, Computer Graphics Forum, Vol. 38, No. 3, 2019, pp. 67–78. Available (Accessed Feb. 14, 2020): https://idl.cs.washington.edu/papers/latent-space-cartography.

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative Adversarial Networks, Advances in Neural Information Processing Systems, 2014, pp. 2672–2680. Available (Accessed Feb. 21, 2020): https://arxiv.org/abs/1406.2661.

[14] P. Bojanowski, A. Joulin, D. Lopez-Paz, A. Szlam, Optimizing the Latent Space of Generative Networks, International Conference on Machine Learning, 2017. Available (Accessed Feb. 21, 2020): https://arxiv.org/abs/1707.05776.

[15] T. Spinner, J. Körner, J. Görtler, O. Deussen, Towards an Interpretable Latent Space, Visualization for AI Explainability, 2018. Available (Accessed Apr. 21, 2020): https://thilospinner.com/towards-an-interpretable-latent-space/.

[16] L. van Der Maaten, G. Hinton, Visualizing Data using t-SNE, Journal Of Machine Learning Research, Vol. 9, 2008, pp. 2579–2605. Available (Accessed Mar. 2, 2020): http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf.

[17] I. Jolliffe, J. Cadima, Principal component analysis: a review and recent developments, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, Vol. 374, No. 2065, 2016. Available (Accessed Mar. 5, 2020): https://www.ncbi.nlm.nih.gov/pubmed/26953178.

[18] L. McInnes, J. Healy, J. Melville, UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, 2018. Available (Accessed Mar. 5, 2020): https://arxiv.org/abs/1802.03426.

[19] T. White, Sampling Generative Networks, 2016. Available (Accessed Mar 24, 2020): https://arxiv.org/abs/1609.04468.

[20] G. Poier, D. Schinagl, H. Bischof, Learning Pose Specific Representations by Predicting Different Views, IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018. Available (Accessed Apr. 22, 2020): https://arxiv.org/abs/1804.03390.

[21] T. Mikolov, Y. Wen-tau, G. Zweig, Linguistic Regularities in Continuous Space Word Representations, Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2013, pp. 746–751. Available (Accessed: Mar. 24, 2020): https://www.aclweb.org/anthology/N13-1090.

[22] D. Gupta, Image Segmentation Keras. Available (Accessed Apr 2, 2020): https://github.com/divamgupta/image-segmentation-keras.

[23] E. Zakirov, Keras implementation of Deeplabv3+. Available (Accessed Apr 14, 2020): https://github.com/bonlime/keras-deeplab-v3-plus.

APPENDIX A: EXPERIMENT CODES

This appendix includes experiment codes that that were used to get results in section 4.2.

```
# This function draws a plot and annotates it
#   results: 2 dimensional numpy array
def show_plot(results):
    plt.figure()
    plt.scatter(results[:,0], results[:,1])

    for i in range(0, results.shape[0]):
        plt.annotate(i, (results[:,0][i], results[:,1][i]))

    plt.show()
```

Program 1. *Function that is used to show projections with reduced dimensionality.*

```
# This function finds the nearest neighbour of a latent space point by
# calculating an Euclidean distance between it and each original data point
#   vec: a vector that represents a point in latent space
#   latent_variables: latent representations of original data points
def find_NN(vec, latent_variables):
    min_dist = None
    result_vec = None
    index = None
    for i in range(0, latent_variables.shape[0]):
        ref_vec = latent_variables[i,:]
        distance = np.linalg.norm(vec-ref_vec)
        distance = abs(distance)
        if min_dist == None or min_dist > distance:
            min_dist = distance
            result_vec = ref_vec
            index = i
    return result_vec, index
```

Program 2. *Function that is used to find the nearest neighbour of a latent space point.*

```
# This function reads images from a folder, preprocesses them for DeepLabv3+
# model and saves them
#   directory: path to the folder where the images are
#   type: image 'i' or annotation 'a' depending on the type of images that
#   folder contains
def read_from_folder_DeepLabv3(directory, type):
    trained_image_width = 384
    mean_subtraction_value = 255

    X = []
    for filename in os.listdir(directory):
        image = np.array(Image.open(os.path.join(directory, filename)))

        w, h, _ = image.shape
        ratio = float(trained_image_width) / np.max([w, h])
        resized_image = np.array(Image.fromarray(image.astype('uint8')).
                                 resize((int(ratio * h), int(ratio * w))))

        if type == 'i':
            resized_image = resized_image / mean_subtraction_value

        pad_x = int(trained_image_width - resized_image.shape[0])
        pad_y = int(trained_image_width - resized_image.shape[1])
        resized_image = np.pad(resized_image, ((0, pad_x), (0, pad_y),
                                               (0, 0)), mode='constant')

        X.append(resized_image)

    images = np.vstack([X])
    return images
```

Program 3. *Function that is used to read images from a folder to a numpy array and preprocess them for DeepLabv3+ model.*

```
# This code trains a SegNet model and saves the whole model, the encoder and
# the decoder

# Needed imports
from keras_segmentation.models.segnet import vgg_segnet
from keras.layers import Input
from keras.models import Model

# Create model using image-segmentation-keras framework
model = vgg_segnet(12, 384, 288)

# Train model
model.train(
    train_images =  "dataset1/images_prepped_train",
    train_annotations = "dataset1/annotations_prepped_train",
    epochs = 22
)

# Save full model, encoder and decoder
model.save("SegNet")

# The last 18 layers of model belong to decoder and the rest to encoder
encoder = Model(model.input, model.layers[-19].output)
encoder.save("SegNet_encoder")

# Shape of latent space is (24, 18, 512)
encoded_input = Input(shape = (24, 18, 512))
num_decoder_layers = 18
decoder_layer = encoded_input

# Go through all decoder layers and add them to model
for i in range(-num_decoder_layers, 0):
    decoder_layer = model.layers[i](decoder_layer)
decoder = Model(encoded_input, decoder_layer)

decoder.save("SegNet_decoder")
```

Program 4. *Code that trains a SegNet model and saves the whole model, the encoder and the decoder separately.*

```
# This code plots latent space dimensionality reductions received from t-SNE
# and PCA algorithms for SegNet

# Needed imports
import show_plot
import os
from keras.preprocessing import image
import numpy as np
from keras.models import load_model
from sklearn import manifold
import matplotlib.pyplot as plt
from sklearn import decomposition

# This function can be used to read images from a folder to a numpy array
#   directory: path to the folder where the images are
#   size: tuple of integers, size to which the image is resized
def read_from_folder(directory, size):
    X = []
    for filename in os.listdir(directory):
        img = image.load_img(os.path.join(directory, filename),
                             target_size = size)
        x = image.img_to_array(img)
        X.append(x)

    images = np.vstack([X])
    return images

# Load saved encoder
encoder = load_model("SegNet_encoder")

encoder.compile(loss = 'categorical_crossentropy',
              optimizer = 'adadelta',
              metrics = ['accuracy'])

# Read train images from a folder to a numpy array
directory = 'dataset1/images_prepped_train/'
target_size = (384, 288)

train_images = read_from_folder(directory, target_size)

# Encode inputs to latent variables and save them
latent_var = encoder.predict(train_images)
np.save("latent_variables_SegNet.npy", latent_var)

# Latent space has originally 4 dimensions, t-SNE and PCA tools accept
# only 2 dimensions so each sample is reshaped to one dimension
latent_var_2D = np.empty((0, latent_var.shape[1]*latent_var.shape[2] \
                          *latent_var.shape[3]))
for i in range (0, latent_var.shape[0]):
    a = latent_var[i,:]
    a = a.reshape((-1,))
    latent_var_2D = np.append(latent_var_2D, [a], axis = 0)

# Calculate t-SNE and PCA and display results
tsne = manifold.TSNE(n_components = 2, random_state = 0)
tsne_results = tsne.fit_transform(latent_var_2D)
show_plot(tsne_results)

pca = decomposition.PCA(n_components = 2, random_state = 0)
pca_results = pca.fit_transform(latent_var_2D)
show_plot(pca_results)
```

Program 5. *Code that visualizes t-SNE and PCA projections over the latent space of SegNet model.*

```
# This code creates interpolation results from the latent space of SegNet and
# shows their nearest neigbours from the original data points

# Needed imports
import find_NN
import numpy as np
from keras.models import load_model
import cv2

# This function makes a result image from the output of decoder
#   prediction: output of the decoder
#   id: identification number for saving
def show_output(prediction, id):
    # image-keras-segmentation has defined output to be shape (27648, 12) that
    # is reshaped to shape (192, 144, 12)
    prediction = prediction.reshape((192, 144, 12)).argmax(axis = 2)

    # Pixel values are ID numbers so they are changed to corresponding RGB
    # values
    output = np.zeros((prediction.shape[0], prediction.shape[1], 3))
    for c in range(np.amax(prediction)):
        pred_arr_c = prediction[:, :] == c
        output[:, :, 0] += ((pred_arr_c)*(colors[c][0])).astype('uint8')
        output[:, :, 1] += ((pred_arr_c)*(colors[c][1])).astype('uint8')
        output[:, :, 2] += ((pred_arr_c)*(colors[c][2])).astype('uint8')

    # Output is resized back to the image space size and saved to a file
    output = cv2.resize(output, (480, 360))
    cv2.imwrite('sample' + id + '.png', output)

# Load encoded latent variables and saved decoder model
latent_var = np.load("latent_variables_SegNet.npy")

decoder = load_model("SegNet_decoder")

decoder.compile(loss = 'categorical_crossentropy',
              optimizer = 'adadelta',
              metrics = ['accuracy'])

# Define a color space
colors = [(np.random.randint(0, 255), np.random.randint(
    0, 255), np.random.randint(0, 255)) for _ in range(5000)]

# 12 points are interpolated between latent points 38 and 138. Predicted masks
# of interpolation points and their nearest neighbours are shown
a = np.linspace(0,1,12)
v1 = latent_var[38,:]
v2 = latent_var[138,:]

for i in range (0,12):
    vnew = a[i]*v1 + (1-a[i])*v2
    NN, NN_index = find_NN(vnew, latent_var)

    predicted_interpolation = decoder.predict(np.expand_dims(vnew, axis = 0))
    predicted_neighbour = decoder.predict(np.expand_dims(NN, axis = 0))

    show_output(predicted_interpolation, str(i) + "a")
    show_output(predicted_neighbour, str(i) + "b")
```

Program 6. *Code that constructs new interpolated samples from the latent space of SegNet and shows their nearest neighbours.*

```
# This code trains and saves DeepLabv3+ model and saves its encoder, decoder
# and convolutional layers as separate models

# Needed imports. File model.py is cloned from
# https://github.com/bonlime/keras-deeplab-v3-plus
from model import Deeplabv3
import read_from_folder_DeepLabv3
import os
import numpy as np
from PIL import Image
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model

# Create model
model = Deeplabv3(input_shape = (384, 384, 3), classes = 14,
                  activation = 'sigmoid')

# Load training images and annotations
images_directory = 'dataset1/images_prepped_train/'
train_images = read_from_folder_DeepLabv3(images_directory, 'i')

annotations_directory = 'dataset1/annotations_prepped_train/'
train_annotations = read_from_folder_DeepLabv3(annotations_directory, 'a')

# Fit and save model
model.fit(train_images, train_annotations, batch_size = 6, epochs = 24)
model.save("Deeplabv3+")

# All but the last 18 layers belong to the convolutional layers that are saved
# to one separate model
convolutional = Model(model.input, model.layers[-19].output)
convolutional.save('Deeplabv3+_convolutional')

# Encoder takes output of convolutional layers as input and adds several layers
# over it
convolutional_output = Input(shape = (None, None, 320))

x = model.layers[-18](convolutional_output)
x = model.layers[-17](x)
x = model.layers[-16](x)
x = model.layers[-15](x)
x = model.layers[-14](x)
x = model.layers[-12](x)

encoder = Model(convolutional_output, x)
encoder.save('Deeplabv3+_encoder')

# Decoder is built of two different layer lines. The first one comes from the
# output of convolutional layers and the other from the output of encoder
encoder_output = Input(shape = (1, 1, 256))

x = model.layers[-13](convolutional_output)
x = model.layers[-11](x)
x = model.layers[-9](x)
y = model.layers[-10](encoder_output)
z = model.layers[-8]([y, x])
for i in range(-7, 0):
    z = model.layers[i](z)

decoder = Model([encoder_output, convolutional_output], z)
decoder.save('Deeplabv3+_decoder')
```

Program 7. *Code that trains a DeepLabv3+ model and saves the whole model, the convolutional layers, the encoder and the decoder separately.*

```
# This code plots latent space dimensionality reductions received from t-SNE
# and PCA algorithms for DeepLabv3+

# Needed imports
import read_from_folder_DeepLabv3
import show_plot
import os
from PIL import Image
import numpy as np
from keras.models import load_model
from sklearn import manifold
import matplotlib.pyplot as plt
from sklearn import decomposition

# Load saved encoder and model that contain convolutional layers
convolutional = load_model("Deeplabv3+_convolutional")
encoder = load_model("Deeplabv3+_encoder")

# Read train images from a folder to a numpy array
directory = 'dataset1/images_prepped_train/'
train_images = read_from_folder_DeepLabv3(directory, 'i')

# Encode inputs to latent variables and save them
latent_var = encoder.predict(convolutional.predict(train_images))
np.save("latent_variables_DeepLab.npy", latent_var)

# Latent space has originally 4 dimensions, t-SNE and PCA tools accept
# only 2 dimensions so each sample is reshaped to one dimension
latent_var_2D = np.empty((0, latent_var.shape[1]*latent_var.shape[2] \
                          *latent_var.shape[3]))
for i in range (0, latent_var.shape[0]):
    a = latent_var[i,:]
    a = a.reshape((-1,))
    latent_var_2D = np.append(latent_var_2D, [a], axis = 0)

# Calculate t-SNE and PCA and display results
tsne = manifold.TSNE(n_components = 2, random_state = 0)
tsne_results = tsne.fit_transform(latent_var_2D)
show_plot(tsne_results)

pca = decomposition.PCA(n_components = 2, random_state = 0)
pca_results = pca.fit_transform(latent_var_2D)
show_plot(pca_results)
```

Program 8. *Code that visualizes t-SNE and PCA projections over the latent space of DeepLabv3+ model.*

```
# This code creates interpolation results from the latent space of SegNet and
# shows their nearest neigbours from the original data points

# Needed imports
import read_from_folder_DeepLabv3
import find_NN
import numpy as np
import os
from PIL import Image
from keras.models import load_model
import matplotlib.pyplot as plt

# This function makes a result image from the output of decoder
#   prediction: output of the decoder
def show_output(prediction):
    prediction = np.argmax(prediction.squeeze(), -1)
    prediction = prediction[:-96]
    prediction = np.array(Image.fromarray(prediction.astype('uint8')). \
                          resize((480, 360)))
    plt.figure()
    plt.imshow(prediction)

# Load encoded latent variables, train images and both convolutional and
# decoder model.
latent_var = np.load("latent_variables_DeepLab.npy")

directory = 'dataset1/images_prepped_train/'
train_images = read_from_folder_DeepLabv3(directory, 'i')

convolutional = load_model("Deeplabv3+_convolutional")
decoder = load_model("Deeplabv3+_decoder")

# 12 points are interpolated between latent points 38 and 138. Predicted masks
# of interpolation points and their nearest neighbours are shown. Input images
# to convolutional layers are interpolated in similar way than latent points
# but in image space
a = np.linspace(0,1,12)
v1 = latent_var[38,:]
image1 = train_images[38,:]
v2 = latent_var[138,:]
image2 = train_images[138,:]

for i in range (0,12):
    vnew = a[i]*v1 + (1-a[i])*v2
    image = a[i]*image1 + (1-a[i])*image2

    NN, NN_index = find_NN(vnew, latent_var)
    NN_image = train_images[NN_index]

    predicted_interpolation = decoder.predict([vnew, convolutional.predict \
                                               (np.expand_dims(image, 0))])
    predicted_neighbour = decoder.predict([NN, convolutional.predict \
                                           (np.expand_dims(NN_image,
0))])

    show_output(predicted_interpolation)
    show_output(predicted_neighbour)
```

Program 9. *Code that constructs new interpolated samples from the latent space of DeepLabv3+ and shows their nearest neighbours.*