

A Comprehensive Study of Autoencoders' Applications Related to Images

Volodymyr Kovenko, Ilona Bogach^a

Vinnitsia National Technical University, Khmelnytsky highway 95, Vinnitsia, 21021 Ukraine

Abstract

This article incorporates a comprehensive study of autoencoders' applications related to images. First of all, a vanilla autoencoder is described along with details of its architecture and training procedure. Secondly, main methods for regularization of it are exposed, such as dropout and additive gaussian noise. The applications of autoencoders such as image morphing, reconstruction and search are shown. Then, the VAE (variational autoencoder) is highlighted. Main applications of it such as outliers detection and image generation are described. Finally, it's shown that using warm-up for VAE with respect to KL loss gives much more plausible results in terms of image generation.

Keywords ¹

DNN, autoencoders, VAE, CNN, generative models

1. Introduction

Nowadays the family of machine learning algorithms called neural networks has become a first-variant solution for a large set of problems, starting from image classification and ending with voice generation. The advances in the subdomain of neural networks named deep learning made feature engineering much easier, as a big number of nonlinear transformations in DNNs (deep neural networks) serves as feature extractor. What is more, DNNs are capable of finding the hidden structure of the data and compressing it, saving the most relevant information. These capabilities made DNN's a nice choice for dimensionality reduction tasks. Moreover, if to compare dimensionality reduction of DNNs and PCA[1], the first outperforms the last, as the nonlinearity of DNNs helps them to compress data in a much more complex way. The capability of finding a hidden structure of the data comes with a possibility to reconstruct data from it. This particular feature is the main aspect which provides us with an opportunity to reconstruct corrupted data, generate new data samples and so on. In this article the applications of autoencoders concerning image data are discussed. As image data is quite complex in terms of the amount of features, the convolutional neural network architecture is used in autoencoder. Despite the fact that autoencoder is trained for image reconstruction, it can be used to tackle various domain tasks, such as image morphing, searching for similar images and transfer learning. However, the vanilla autoencoder is not capable of image generation, as it is trained only for data reconstruction from corrupted/non corrupted samples. For the purpose of image generation, VAE (variational autoencoder)[2] is used. Along with GAN[3], VAE has become a very popular choice for the generative procedure, as it manages to reconstruct the data with a high accuracy and also satisfies the condition that all the compressed representations of data samples are as close to each other as possible, while still being distinct. The mechanics of the autoencoder and VAE are exposed. The experiments were conducted on a publicly available dataset named "Anime-Face-Dataset"[4]. The up


IT&I-2020 Information Technology and Interactions, December 02–03, 2020, KNU Taras Shevchenko, Kyiv, Ukraine

EMAIL: urumipainblackreaper@gmail.com (A. 1); ilona.bogach@gmail.com (A. 2);

ORCID: 0000-0003-3825-1115 (A. 1); 0000-0001-9398-8529 (A. 2);

© 2020 Copyright for this paper by its authors.

Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

to date improvements and applications of generative models for images are left for further discussions.

Though we didn't focus on the comparative analysis of our model and existing implementations, the analysis was conducted in terms of choosing the hyper-parameters, types of AE's model architectures and training procedures.

2. Anime-Face-Dataset

The dataset, called "Anime-Face-Dataset", on which experiments were performed consisted of 63632 RGB images of anime characters faces (Figure 1).



Figure 1: Anime-Face-Dataset

The images were resized to 128x128 pixel size and normalized by max scaling in order them to be in range from 0 to 1. Then, the data was randomly splitted into two subsets: train (75%) and validation (15%), and used in the same way for all the experiments.

3. Autoencoder

3.1. General Overview

Autoencoder is an architecture of neural network which is used to reconstruct data. It consists of two parts: encoder and decoder (Figure 2).

The main goal of encoder is to compress the data in the way that main features of it are preserved, thus helping a decoder to reconstruct data from the compressed representation. Autoencoders are trained using back-propagation[5] algorithm in the way that the difference between original and reconstructed data is minimized.

The obvious choice for a loss function to train such type of networks is MSE (mean squared error) or MAE (mean absolute error) (1, 2).

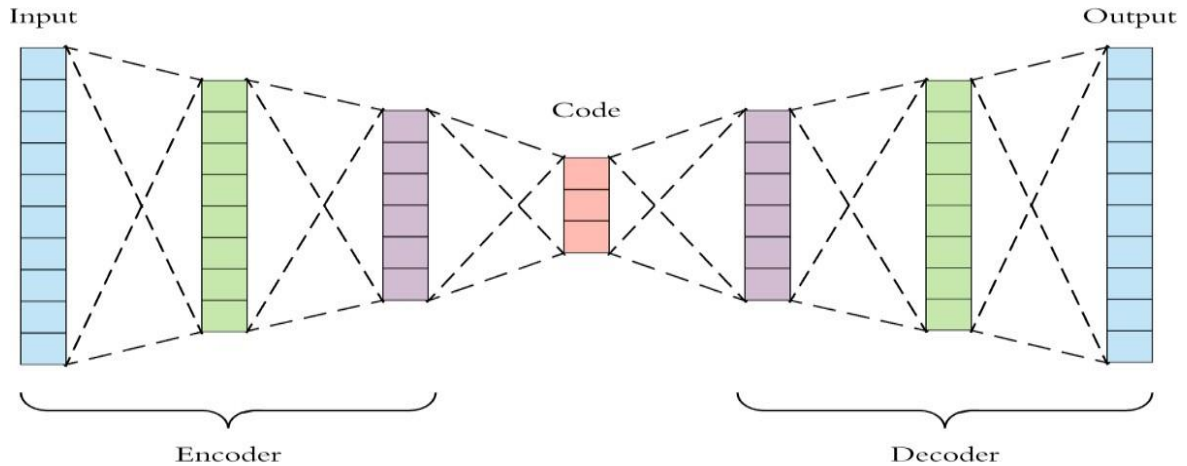


Figure 2: Autoencoder architecture

$$MSE = \frac{\sum_{i=1}^N (X_{true,i} - X_{reconstructed,i})^2}{N}, \quad (1)$$

$$MAE = \frac{\sum_{i=1}^N |X_{true,i} - X_{reconstructed,i}|}{N}, \quad (2)$$

where $X_{true,i}$ - is an input image and $X_{reconstructed,i}$ - is a reconstructed image.

Despite the fact that autoencoder resembles PCA algorithm, it's much more powerful when dealing with complex and nonlinear data due to nonlinearity of neural networks. While working with image data, the convolutional architecture of encoder and decoder is usually used. This choice is proven by the fact that convolutional layers tend to capture temporal and spatial dependencies in image through learning relevant filters. Also, by increasing the number of convolutional layers, the receptive field also increases, thus the last layers of convolutional neural network contain much more complex features (it will be shown further in the paper). Max pooling layer, which main goal is decreasing overfitting and feature map size and making model more robust to rotations of pixels, was used after each convolutional one. In order to reconstruct data decoder uses transpose convolutions that learn filters to perform upsampling operation (Figure 3).

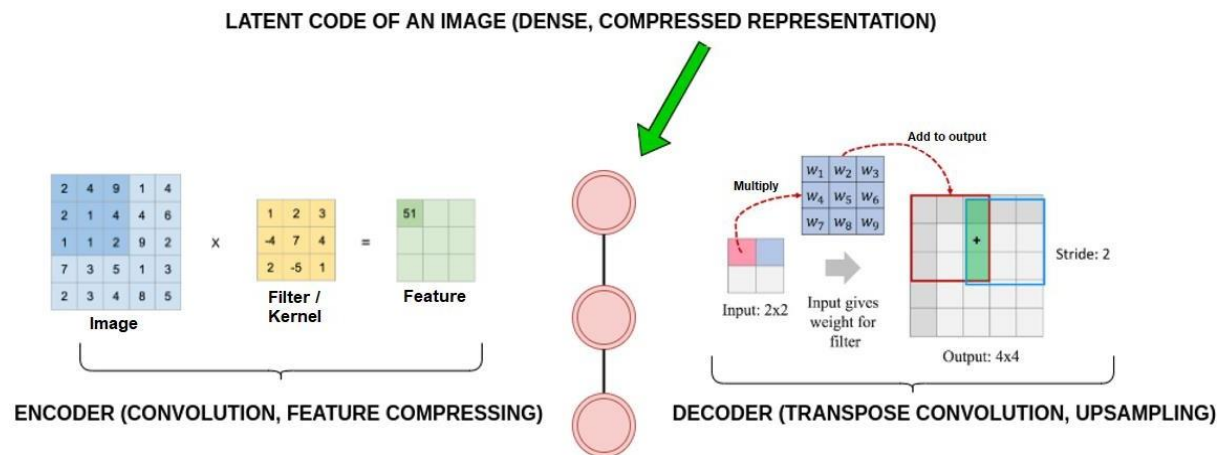


Figure 3: Architecture of convolutional autoencoder

3.2. Experiments and applications

As the data was normalized in the range from 0 to 1, the function used in the output layer is sigmoid. The vanilla autoencoder was trained for 10 epochs using Adamax[6] optimizer with batch size of 128 samples. The same settings of batch size and normalization are used throughout the paper. The results showed that vanilla autoencoder didn't experience overfitting (Figure 4).

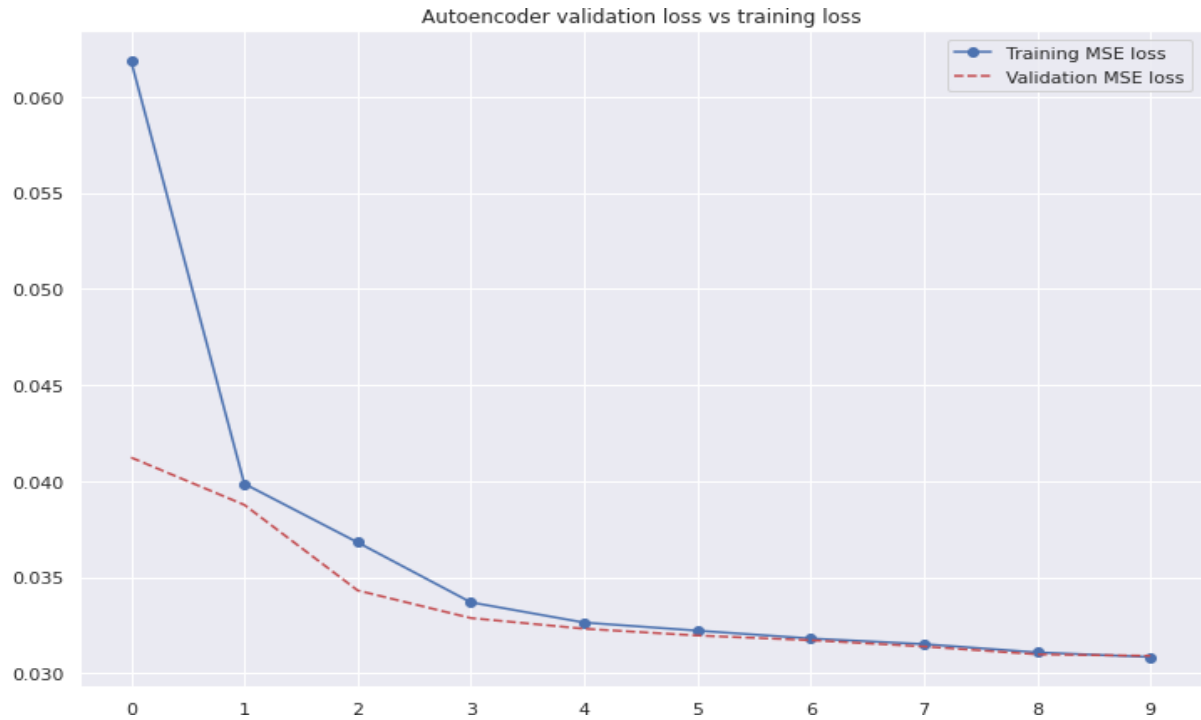


Figure 4: Comparison of MSE loss on training and validation subset

However, the model trained in such a way produces quite blurry reconstructions, it still preserves various attributes such as hair color, eyes size and so on. (Figure 5)



Figure 5: Real image vs reconstruction

The interesting and fun application of vanilla autoencoder is so called “image morphing”. The gist of it, is that the one can make the image which will have mixed attributes from two different images. To achieve this, the one makes the linear combination of two image codes from encoder and passes it to decoder (3) (Figure 6):

$$mixed\ image = decoder(code_1 * (1 - alpha) + code_2 * alpha), where\ 0 \leq alpha \leq 1, \quad (3)$$

where $alpha \in [0,1]$, $code_1$ and $code_2$ are outputs of encoder part of model for first and second image.

The one key property of autoencoder is that via training it for data reconstruction its encoder part learns complex feature representations (Figure 7). This property makes an encoder part of a model a nice candidate for such application as transfer learning[7]. What is more, using the encoder, the one can seek for similar images by calculating the distance between image codes using KNN[8] (Figure 8).



Figure 6: Image morphing with different values of alpha

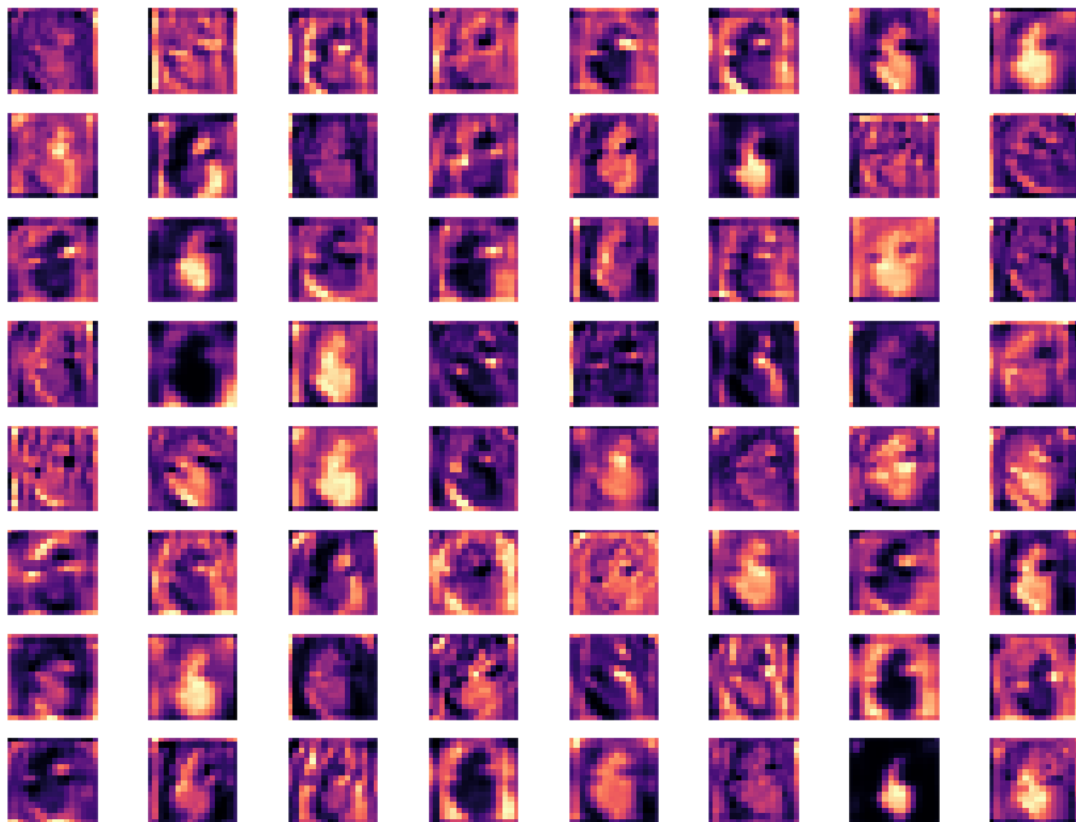


Figure 7: Visualization of last convolutional layer in autoencoder

Despite the fact that vanilla autoencoder has lots of applications, it's never used for the goal it was trained for: image reconstruction, as there is no logic in reconstructing an original good image into a blurred version of it. However, what one can do is to train the autoencoder to reconstruct a corrupted image into a good one. For achieving this goal, the data is usually augmented with noise/rotations/etc. This trick also boosts regularization of a model, as now it has to deal with much more difficult task. In this paper the denoising autoencoder is considered. It's trained to reconstruct images with additive gaussian noise with mean of 0 and standard deviation of 0.3 into original ones (Figure 9).

The other possible way to construct denoising autoencoder [9] is to corrupt data on the level of a model, using a dropout layer (with rate of setting units to 0 equal to 0.3) right after the input one in the encoder part of the network (Figure 10).



Figure 8: Example of finding similar images using encoder part of the network

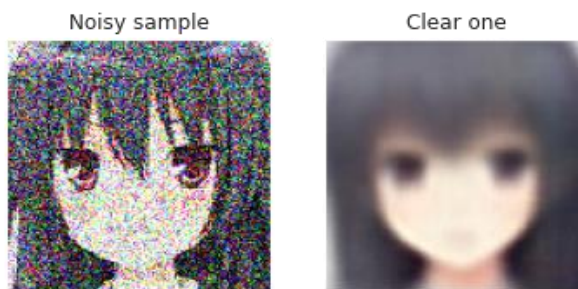


Figure 9: Denoising images with autoencoder



Figure 10: Example of finding similar images using encoder part of the network

Though autoencoder can be used to make fascinating things, its reconstructions aren't very detailed and this kind of model isn't able to generate new data samples. Further in the paper we will consider a probabilistic model which gives an opportunity to produce new data samples from gaussian noise.

4. VAE

4.1. General Overview

The task of understanding data is crucial for generative applications. If we think about generative process in probabilistic setting, our goal is to model the distribution of our data, mainly $p(x)$. Having the model of data distribution, we can then sample from it to generate new data and detect outliers. However, it turns out that modeling image data is quite a difficult task, as ordinary methods are infeasible, too restrictive or too slow in terms of synthesizing new data. If we represent the model of our data by marginalizing out latent variable t (where t is some latent variable, x is conditioned on), we will arrive at the following integral: $p(x) = \int p(x|t)p(t)dt$ which is intractable (thus, there is

no way we can evaluate marginal likelihood) and true posterior: $p(t|x) = \frac{p(x|t)*p(t)}{p(x)}$ is also intractable (thus, we can't use EM[10] algorithm). One workaround is to model image distribution using VAE. VAE is mainly a framework for efficient approximation of ML (maximum likelihood) or MAP (maximum a posteriori) estimation for the parameters w (parameters of the decoder part of a model), which gives a possibility to mimic the random process and generate artificial data. VAE fights the intractability by approximating the intractable true posterior using a probabilistic encoder: $q(t|x)$, which produces the distribution over latent variables t given data x . In a similar way it represents $p(x|t)$ using decoder, as given latent code t it produces the distribution of corresponding values of x . As VAE's decoder and encoder are neural networks of some architecture, the whole model should be trained using gradient based methods and back propagation algorithm w.r.t objective function which involves gradients computation. However, it turns out that it's infeasible to compute the gradient of $q(t|x, \phi)$, where ϕ is the parameters of the encoder part of the model. The solution proposed by Kingma is to use a so called reparameterization trick[11], that changes sampling t_i from $q(t_i|x_i, \phi)$ to $t_i = \epsilon \odot s_i + m_i$ (where ϵ is a random variable from standard gaussian distribution $N(0, I)$, s_i and m_i are standard deviation and mean modeled by encoder) which is now differentiable. The overall architecture of VAE looks the following way (Figure 11).

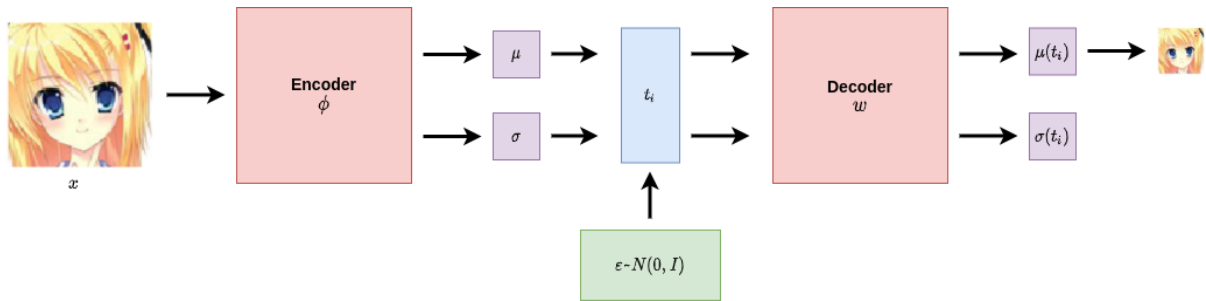


Figure 11: Architecture of VAE

The objective function of VAE consists of two parts: reconstruction loss and regularization (4):

$$L = \sum_{i=1} E_{q_i} \log p(x_i | t_i, w) - KL(q_i(t_i) || p(t_i)), \quad (4)$$

where $p(x_i | t_i, w)$ is a probability of image x_i given latent code t_i and decoder weights w , $p(t)$ is a prior distribution and $q(t)$ is a posterior approximation.

The first term in objective is often replaced with the reconstruction loss from vanilla autoencoder, MSE or MAE, however while experimenting with MNIST data, Kingma used to model reconstruction loss as Bernoulli distribution (which is actually a convenient choice, as MNIST data is grayscale and is often normalized between 0 and 1). For the case of RGB images, which is considered in this paper, the reconstruction loss is modeled as Gaussian distribution (5).

$$\begin{aligned} \log p(x | \mu(t_i), \sigma^2(t_i)) &= \log \left[\prod_{i=1}^N \frac{1}{\sqrt{2\pi\sigma^2(t_i)}} e^{-\frac{(x_i - \mu(t_i))^2}{2\sigma^2(t_i)}} \right] \\ &= -\sum_{i=1}^N \frac{(x_i - \mu(t_i))^2}{2\sigma^2(t_i)} - \frac{1}{2} \log(2\pi\sigma^2(t_i)) = \\ &= -\frac{N}{2} \log(2\pi) - \sum_{i=1}^N \frac{(x_i - \mu(t_i))^2}{2\sigma^2(t_i)} + \frac{\log(\sigma^2(t_i))}{2}, \end{aligned} \quad (5)$$

where $p(x_i | \mu(t_i), \sigma^2(t_i))$ is a probability of image x_i given $\mu(t_i)$ and $\sigma(t_i)$ which are dependent on latent code t_i

Different from Kingma and following Doersch approach[12], $\sigma^2(t)$ wasn't modeled by a decoder, but set as a hyperparameter which was then tuned in experiments. The second part of

objective corresponds to Kulback - Leibler divergence[13], which is a measure of how one probability distribution is different from a second (6).

$$KL(q(t)||p(t)) = \int q(t) \log \frac{p(t)}{q(t)} dt. \quad (6)$$

Assuming that both prior $p(t)$ and posterior approximation $q(x)$ are Gaussian ($p(t) = N(0, I)$, $q(t) = N(t; \mu, \sigma^2)$) the KL term can be integrated analytically (formula 7).

$$\begin{aligned} (1) \quad KL(q(t)||p(t)) &= \int q(t) \log \frac{p(t)}{q(t)} dt = \int q(t) (\log p(t) - \log q(t)) dt = \int q(t) \log p(t) dt - \int q(t) \log q(t) dt; \\ (2) \quad \int q(t) \log p(t) dt &= \int N(t; \mu, \sigma^2) \log N(0, I) dt = -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^J (\mu_j^2 - \sigma_j^2); \\ (3) \quad \int q(t) \log q(t) dt &= \int N(t; \mu, \sigma^2) \log N(t; \mu, \sigma^2) dt \\ &= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^J (1 + \log(\sigma_j^2)); \\ (4) \quad KL(q(t)||p(t)) &= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^J (\mu_j^2 - \sigma_j^2) + \frac{J}{2} \log(2\pi) + \frac{1}{2} \sum_{j=1}^J (1 + \log(\sigma_j^2)) \\ &= \frac{1}{2} \sum_{j=1}^J (1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2), \end{aligned} \quad (7)$$

where J - dimensionality of latent code t , μ and σ - mean and standard deviation which are produced by model.

KL term serves as regularization as it forces synthesized data to come from the same distribution (it encourages the approximate posterior to be close to the prior). As it will be shown further, with time KL part of the loss starts to increase as VAE generates data samples that differ from each other.

4.2. Experiments and applications

The experiments were conducted with respect to varying $\sigma^2(t)$. The architecture of VAE's autoencoder and encoder is the same as of discussed AE with small changes to fit the framework's mechanics. The results were validated due to the quality of VAE's generative process. The way to generate new samples using VAE (sample from $p(x)$) is fairly simple: during the inference only the generative part of the model (decoder that learned mapping from latent space t to data x) is used, t is modeled as standard normal distribution. Due to (6) $\sigma^2(t)$ is situated in the denominator, thus if its value is close to 1, the overall reconstruction loss will be a classical MSE + constant. Whereas if its value is close to 0, the loss will be infinitely large. According to these assumptions, $\sigma^2(t)$ is sometimes referred to as regularization hyper-parameter. Three different values of $\sigma^2(t)$ were tried out: 0.1, 0.01, 0.001; and each model was trained for 200 epochs (Figure 12).

Due to Figure 12 it's obvious that the best value for σ^2 is 0.01, as it gives the most plausible generation. With $\sigma^2 = 0.1$ the left part of the objective function resembles the classical MSE loss and the overall training converges to producing a picture which is just the average of samples in the dataset. With $\sigma^2 = 0.001$ the left side of objective is too huge and it takes a lot to optimize it.

The other experiment which was performed is training a model with so-called KL loss warmup or annealing suggested by Sønderby et al[14]. The gist of the approach is in adding new parameter to our objective called β , that is a multiplier of KL part (8).

$$L = \sum_{i=1} E_{q_i} \log p(x_i | t_i, w) - \beta * KL(q_i(t_i) || p(t_i)), \quad (8)$$

where β - annealing parameter.

Apart from the approach suggested by Higgins et al[15], during annealing β isn't set to constant, but is increased from 0 to 1 during warmup epochs. According to Sørnerby the main logic behind such an approach is the fact that variational regularization term causes some of the latent units to become inactive during training. To tackle this problem we can smoothly switch from deterministic autoencoder (with $\beta < 1$) to a variational one. Along with using KL warmup, batch normalization was added after each convolutional layer of both encoder and decoder. The warmup was done for 40 epochs, after which the model continued to train for 200 epochs with the original objective function. The comparison was done against the classical approach of training.



Figure 12: Sprite of generated images for each value of $\sigma^2(t)$

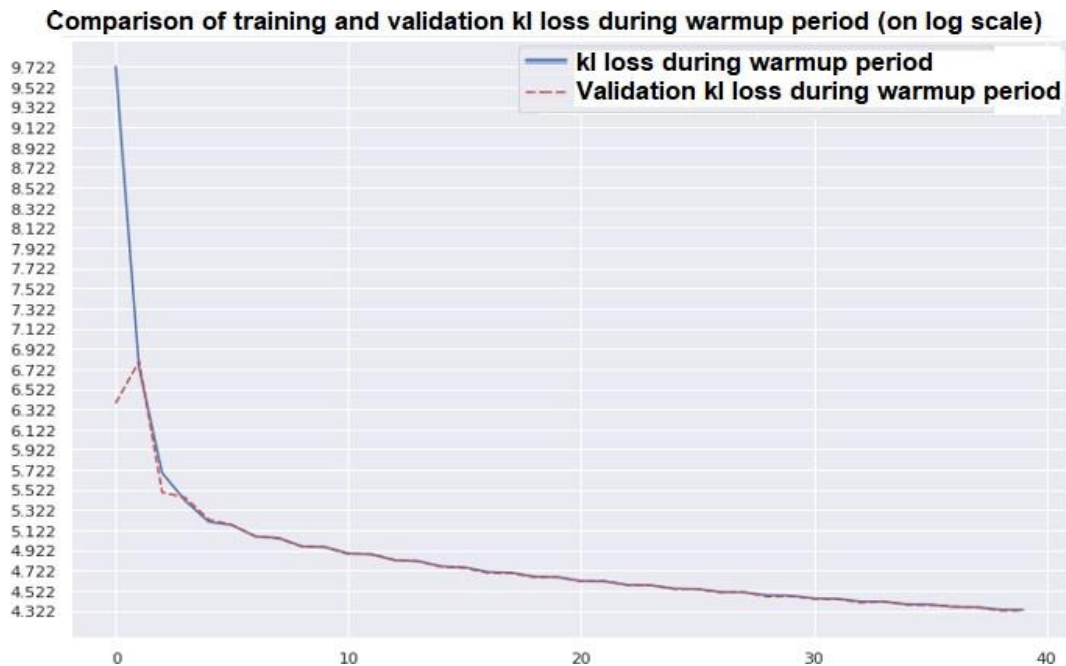


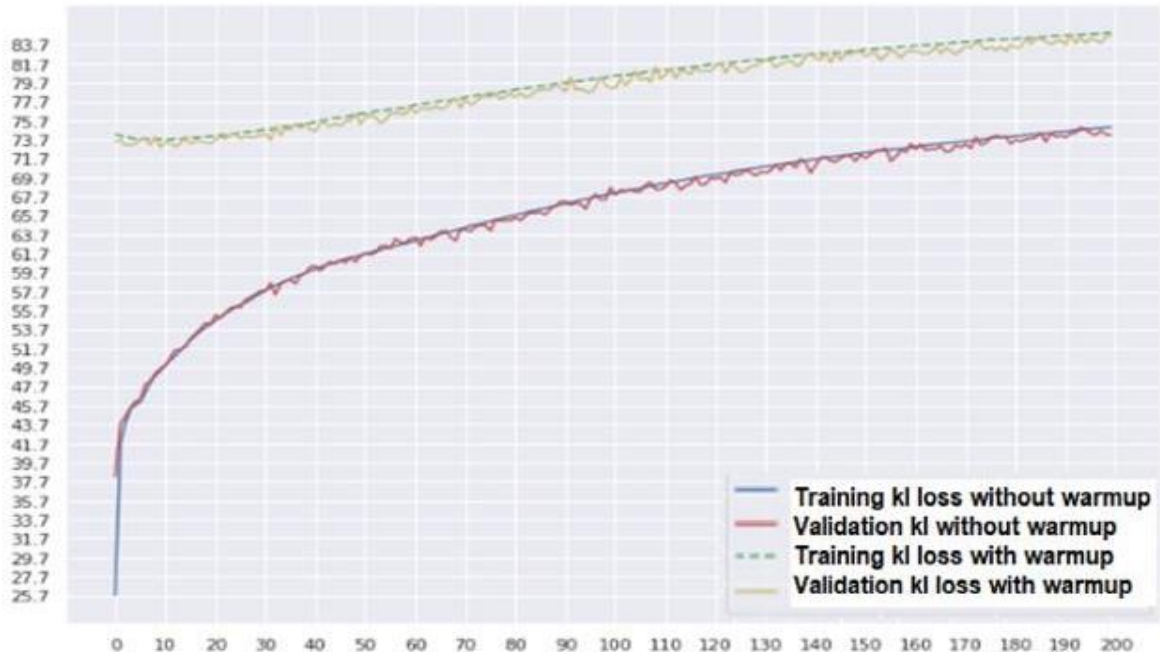
Figure 13: KL loss during warmup period on logarithmic scale

According to Figure 13, KL loss tends to decrease as along with increase of β , its contribution to the overall objective also increases. After β approaches 1, KL loss tends to increase as now our model tries to generate distinct samples. As it's shown on Figure 14 KL loss tends to increase both during classical training and training with warmup. Compared to reconstruction loss related to classical approach, the one related to training with warmup is lower on both training and validation subsets.

It's really hard to tell the difference between samples generated by VAE with batch normalization and warmup and classical VAE, but one can notice that samples generated by VAE with warmup and batch normalization are sharper a bit and contain less artifacts (Figure 15). Another application of VAE is outliers detection. By training VAE, we force it to learn the distribution of images in the

observed data. Thus, using the encoder part of the network we can generate a distribution parameters (mean and standard deviation) for any image, and then calculate the KL divergence for the distribution of particular image versus prior one. If KL divergence is bigger than some predefined threshold the image is an outlier. To calculate the threshold, the KL divergence for 10k samples from the dataset was calculated, the median of scores was denoted as outlier threshold.

Comparison of training and validation kl loss without warmup



Comparison of training and validation reconstruction loss with and without warmup

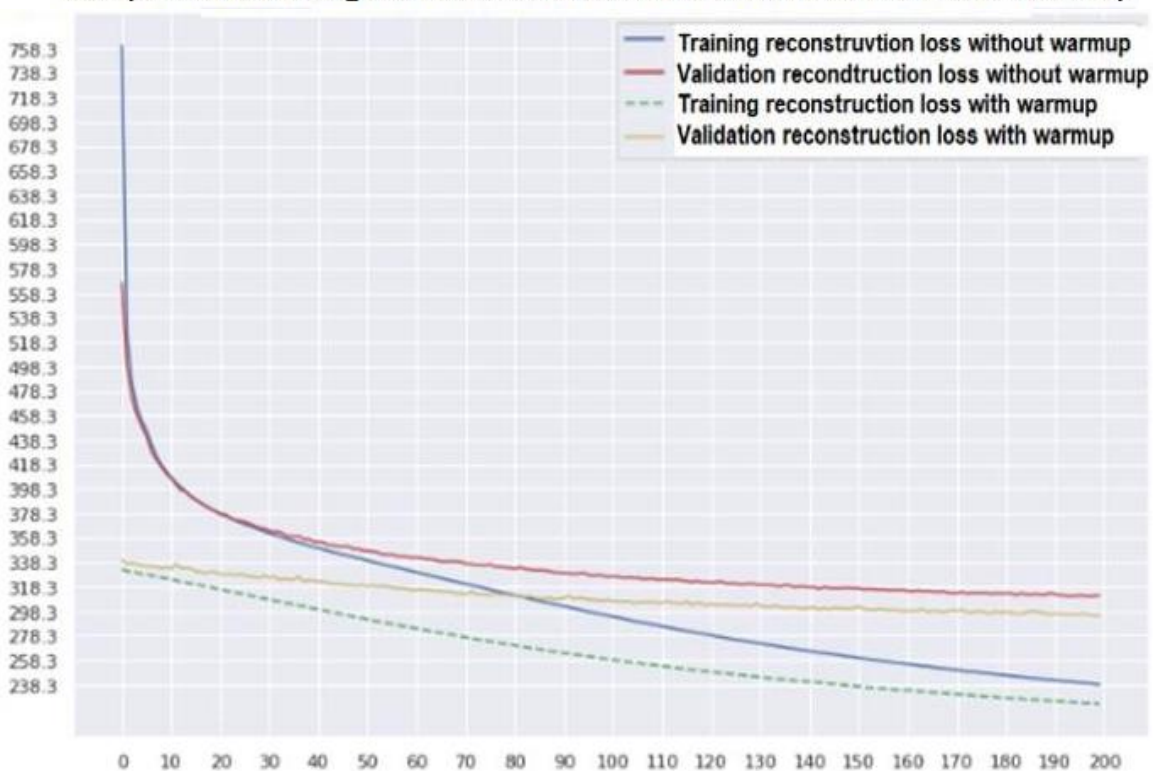


Figure 14: Comparison of warmup training with classical training

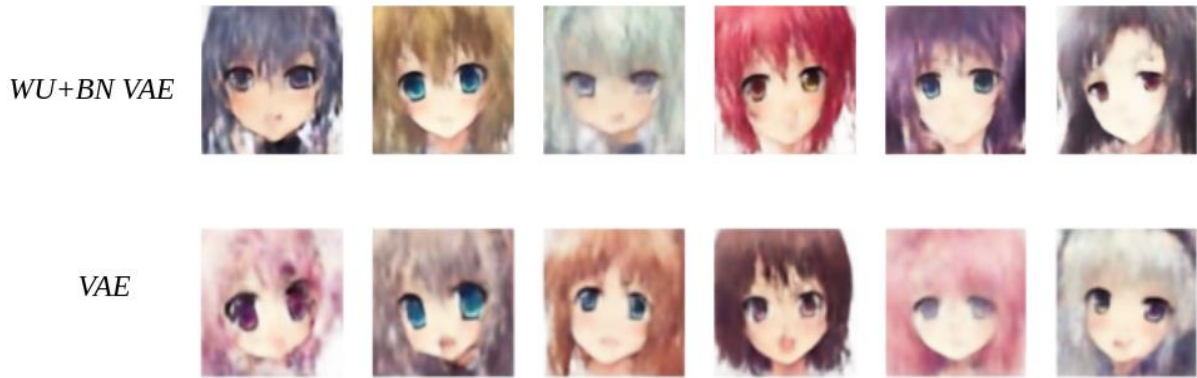


Figure 15: Comparison of data synthesizing of two approaches

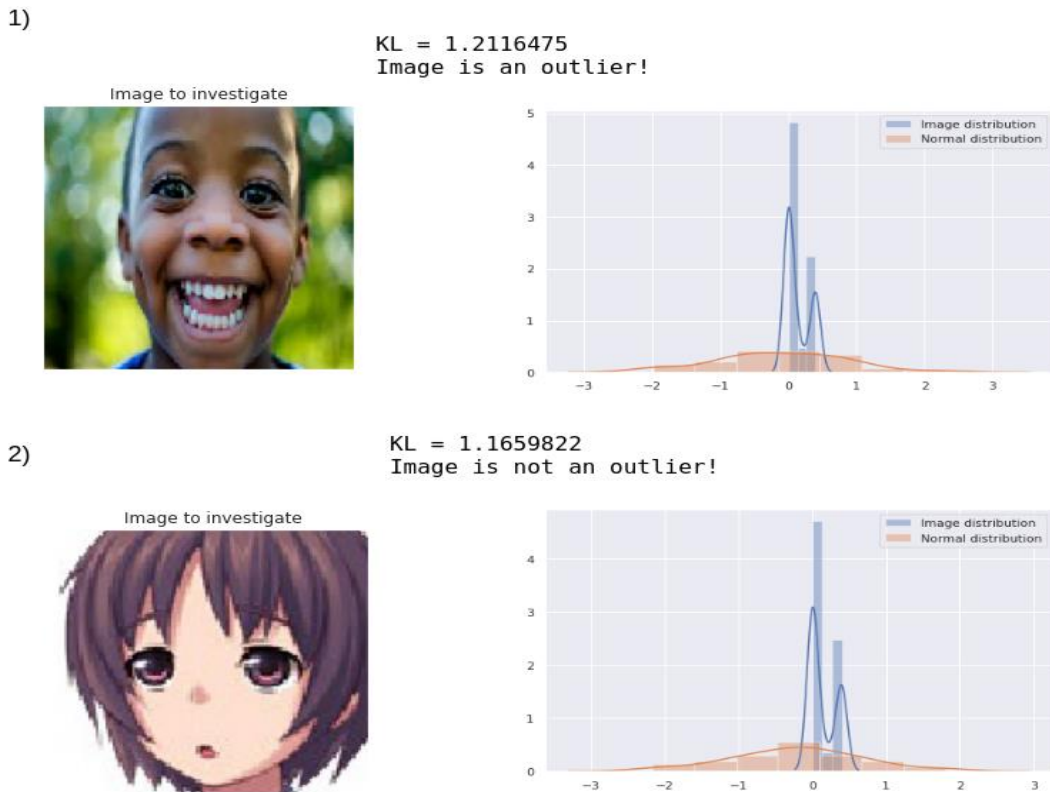


Figure 16: Outliers detection with VAE

5. Discussion and further work

In this paper main applications of autoencoder and VAE in terms of image data were discussed. The possibility of data reconstruction and modeling of its distribution opens doors to lots of interesting applications, such as image morphing, image reconstruction, image generation and outliers detection. Though the generative power of trained VAE isn't brilliant, as images still aren't that sharp and some artifacts persist, training it for longer should enhance results. It was shown that using a warmup of KL loss can give better results in terms of training and lead to better generative results in perspective.

Nevertheless, a big part of applications was highlighted, with the improvements of technologies, new usages of generative models arise. For example, Ma et al. uses a so-called Style-based VAE[16] to tackle the problem of super-resolution, Larsen et al. [17] mixes VAE and GAN to replace element-wise errors with feature-wise errors which leads to better data distribution learning. Recently, GANs have attracted lots of attention. A proposed by Nvidia Style-GAN[18] gives a possibility to generate

samples that are hard to discriminate from real ones even for a human. Although all the exposed usages of autoencoders are related to image data, this framework is often used with other types of data as text and signals.

The example of outliers detection with VAE is shown on Figure 16.

6. References

- [1] Jake Lever, Principal component analysis, 2017. URL: <https://www.nature.com/articles/nmeth.4346>.
- [2] Diederik P Kingma, Max Welling, Auto-Encoding Variational Bayes, 2014. URL: <https://arxiv.org/pdf/1312.6114.pdf>.
- [3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, Generative Adversarial Networks, 2014. URL : <https://arxiv.org/pdf/1406.2661.pdf>
- [4] Brian Chao, Anime-Face-Dataset, 2020. URL: <https://github.com/bchao1/Anime-Face-Dataset>.
- [5] Yann le Cun, A Theoretical Framework for Back-Propagation, 1988. URL: <http://yann.lecun.com/exdb/publis/pdf/lecun-88.pdf>.
- [6] Diederik P. Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization, 2014. URL: <https://arxiv.org/abs/1412.6980>.
- [7] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson, How transferable are features in deep neural networks?, 2014. URL: <https://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf>.
- [8] Zhongheng Gang, Introduction to Machine Learning : K-nearest neighbors, 2016. URL: https://www.researchgate.net/publication/303958989_Introduction_to_machine_learning_K-nearest_neighbors.
- [9] Dominic Mon, Denoising Autoencoders explained, 2017. URL: <https://towardsdatascience.com/denoising-autoencoders-explained-dbb82467fc2>.
- [10] Jason Brownlee, A Gentle Introduction to Expectation-Maximization (EM algorithm), 2019. URL: <https://machinelearningmastery.com/expectation-maximization-em-algorithm/>.
- [11] David M. Blei, Variational inference: A review for statisticians, 2016. URL: <https://www.tandfonline.com/doi/full/10.1080/01621459.2017.1285773>.
- [12] Carl Doersch. Tutorial on Variational Autoencoders, 2016. URL: <https://arxiv.org/pdf/1606.05908>.
- [13] Anna-Lena Popkes, Kullback-Leibler Divergence, 2019. URL: http://alpopkes.com/files/kl_divergence.pdf.
- [14] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe and others, Ladder Variational Autoencoders, 2016. URL: <https://arxiv.org/pdf/1602.02282.pdf>.
- [15] Irina Higgins, Loic Matthey, Arka Pal and others, beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework, 2017. URL: <https://openreview.net/references/pdf?id=Sy2fzU9gl>.
- [16] Xin Ma, Yi Li, Huaibo Huang and others, Style-based Variational Autoencoder for Real-World Super-Resolution, 2020. URL: <https://arxiv.org/pdf/1912.10227.pdf>.
- [17] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle and Ole Winther, Autoencoding beyond pixels using a learned similarity metric, 2016. URL: <https://arxiv.org/pdf/1512.09300.pdf>.
- [18] Tero Karras, Samuli Laine, Timo Aila, A Style-Based Generator Architecture for Generative Adversarial Networks, 2019. URL: <https://arxiv.org/pdf/1812.04948.pdf>.