GANs vs Autoencoders: Comparison of Encodings

Giovanni Luca Favuzzi[*]
g.favuzzi@campus.tu-berlin.de

Niels Warncke[*]
niels.warncke@gmail.com

Thomas Michael Goerttler[†]
thomas.goerttler@tu-berlin.de

Binayak Ghosh[*]
binghosh@gmail.com

Mohannad Assad[*]
assad.mohannad@hotmail.co.il

ABSTRACT

Generative Adversarial Networks (GANs) learn how to sample from a high dimensional distribution, whereas Autoencoders (AE) are trained to compress samples from the distribution. Despite their different architectures and training objectives, the decoder of an AE can be compared to the generator of a GAN, since both learn mappings from a low dimensional space to the data space. This paper shows that it is possible to use GANs as AEs hitting acceptable performances, however, AEs cannot be applied to generate realistic samples. The analysis goes through the reason behind this and shows that only a tiny fraction of the encoding space is used by the AE to encode realistic samples. Such an insight is possible by means of a GAN-based outlier detection technique. The models are applied as preprocessing routines for image classification and editing; these experiments end-up showing that GANs work well for high-level image editing, but neither GANs nor AEs work too well as embeddings.

CCS CONCEPTS

• **Generative Neural Networks** → *GANs*; *Autoencoders*; • **Outlier Detection** → *GAN Outlier Detection.*

KEYWORDS

Generative Adversarial Network, Auto Encoder, Outlier Detection, PCA, Logistic Regression, Convolution

ACM Reference Format:
Giovanni Luca Favuzzi, Niels Warncke, Thomas Michael Goerttler, Binayak Ghosh, and Mohannad Assad. 2019. GANs vs Autoencoders: Comparison of Encodings. In *NIPP '19 Talks, 31 July 2019, Berlin, DE-BE.* , 10 pages.

[*]All authors contributed equally to this research.
[†]This author is the leader of the project.

1 INTRODUCTION

Generative Adversarial Networks (GANs) are very popular since the last years for learning how to sample from complex high dimensional distributions $P_X : \mathcal{X} \mapsto [0,1]$ such as images of faces [7]. It is achieved by training a generator $G : \ddagger \mapsto \mathcal{X}$ to map samples from a prior $P_z : \ddagger \mapsto [0,1]$ in a so-called latent space to images and training a discriminator $D : \mathcal{X} \mapsto [0,1]$ to distinguish these fakes from real samples. Since the introduction of the original GAN by Goodfellow et al [6], various architectures and applications have been explored. In this paper they are employed for generating samples, detecting outliers, for semantic interpolation and image completion. For many of these tasks, it is necessary to map existing samples to a latent code that represents them well. For many applications, it is required to encode a given sample x as well, which consists of finding the latent point z such that $J(x, G(z))$ is small for some loss function J, e.g. the mean squared error (MSE). A function that finds such an encoding is called encoder. Although the regular generator/discriminator architecture does not include an encoder, there are several ways to implement one for a given G: in this paper, we compare the use of a neural network encoder to a training free, gradient descent based approach and find that both techniques yield similar performances, but have different pros and cons depending on the use case.

A GAN combined with such an encoder can be seen as Auto-Encoder that is trained with an adversarial objective. AEs are neural networks consisting of a neural encoder network that maps high dimensional samples of an unknown distribution to a lower-dimensional representation, and a decoder that reconstructs the input from that compressed representation. Applications of AEs involve outlier detection, denoising input or dimensionality reduction for further processing. If we choose $\mathcal{Z}$ as a domain for the encoder, there is an obvious analogy to a GAN as shown in Figure 1: its generator would function as a decoder, since it is the component that transforms a vector of the latent space to an image. In the rest of this paper, we refer to the decoder part of an AE also as G or generator, and use $\mathcal{Z}$ or latent space both for the domain of the prior of a GAN and for the encoding domain of an AE.

The similarity of both models in terms of architecture and applications has inspired a number of mixed architectures, a not comprehensive overview can be found in section 2. In this work, we compare some of those models in terms of their generation power, their compression performance, how suitable they are for high-level image editing and with regard to their properties as embedding models. Experiments on the handwritten digits dataset MNIST [1] and on the related handwritten characters dataset EMNIST [3] show that we can use a pre-trained GAN successfully as Autoencoder with acceptable performance, provided the data that shall be encoded is sampled from the distribution of the train data. Autoencoders trained on handwritten digits can encode other handwritten characters better, at the cost of their suitability for tasks that involve generation of new data, including high-level image editing. As others have argued, this is due to the fact that Autoencoders use only a small fraction of the latent space to encode realistic images. We quantify this fraction using GANs for outlier detection and estimate that our GAN generates realistic samples in more than 83% of the time, while the Autoencoder did not produce any realistic digits when decoding 200 000 randomly sampled latent vectors. Variational Autoencoders (VAE) are a special kind of Autoencoders, trained in a way such that the distribution of encoded samples follows a predefined prior, usually Gaussian. Ideally, this enforces the decoder to also map any latent code samples from this prior to a realistic image. In practice, they can indeed be used for data generation, although images generated by a VAE tend to look blurry and not quite of the quality of GANs. For a detailed analysis of the connection of VAEs and GANs, we refer to other works such as [16], since VAE are not included in our experiments. While we do not find advantages in an architecture trained on a mixed objective, they report improvements with regards to mode collapse.

The rest of this paper is structured as follows: we give an overview of related work in section 2. In section 3, the architectures and training algorithms we used are explained in detail. It follows section 4, where different models are compared with regards to the previously mentioned applications. The details of the experimental setup and the results are described for each experiment. A conclusion including open questions for future work is given in section 5.

2 RELATED WORK

The comparison between AEs and GANs is not explored as much as other typical GAN problems like mode collapse and alike, anyhow there are several papers that either compare the two or a straight-up mix the techniques. For instance, Goodfellow et al. introduces in [11] Adversarial Autoencoders, which are AEs that use the GAN adversarial training procedure to perform variational inference by matching the aggregated posterior of the hidden code vector of the AE with an arbitrary prior distribution; this way the decoder learns a deep generative model that maps the imposed prior to the data distribution, thus it can be used as a generator. In [8] AEs are employed to learn a representation to better measure the similarities in the data space by combining a Variational Autoencoder (VAE) with a GAN discriminator as the basis for the VAE reconstruction objective; this way, instead of using the standard Euclidean distance as similarity, the new metric has invariance towards class-invariant transformations. AEs are combined with GANs in [13] exploiting their similar structure, as already done in the other papers, but trained over a unique objective (thus without alternating the training objectives of GAN and AE); this enables the mixed model to keep the strengths of both objectives. In [14] the mix between GAN and AE promotes convex latent distribution by training adversarially onto latent space interpolations: the AE is used as both generator and discriminator of the GAN and trained onto a pixel-wise loss; the generated samples are sharper than typical GAN ones. [18] provides theoretical reasoning behind why mixing AE and GAN alleviates problems like mode collapse, training instability etc. and tries to increase the stability of the adversarial training by applying an adaptive decay variable on the adversarial error. [10] tries to synthesize different Chinese calligraphy styles by means of a bi-module architecture that consists of a Transfer Network (TN) that transfers standard font images to calligraphy images and of a Supervise Network (SN) that helps the TN training process; the TN is modelled as a GAN Generator, while the SN is an AE. The result is an end-to-end model that outperforms its predecessors in transferring calligraphy styles. [9] proposes an AE-based inverse-mapping GAN: the GAN Generator is able to invert the data-samples into latent-samples since it is modelled as the Decoder of an AE, thus the actual inversion is done through the AE Encoder. This approach is much better than any constrained generator needed to represent a one-to-one invertible function, thus, it ends up outperforming the previous SotA models. AEs are employed in [17] as well, where they help to deal with the vanishing gradient in GANs by constraining the Discriminator: the real samples, that must be fed to the discriminator, are first encoded and decoded (thus reconstructed) by the AE. This couples the training convergence of both AE and Discriminator, effectively slowing down the Discriminator convergence, thus reducing vanishing gradient.

3 METHODOLOGY

This section gives an overview of the experiments performed for this paper. For all experiments, comparable models are trained on the MNIST dataset, which consists of 60000 greyscale images of handwritten digits.

The models we compare are a DCGAN equipped with an encoder neural network, the same architecture with a

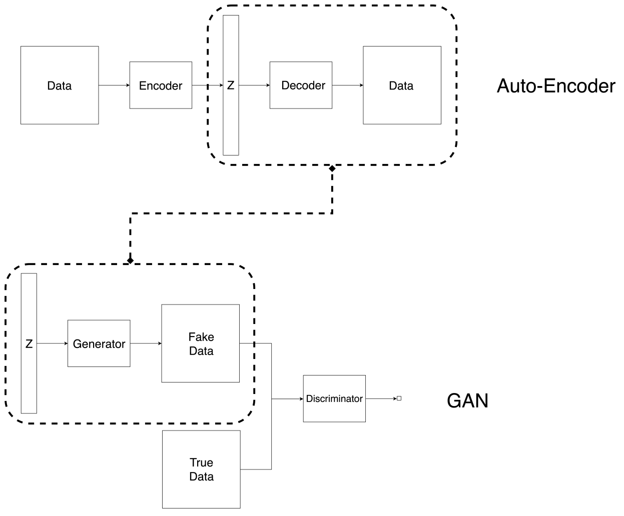


Figure 1: Similarity of the architectures of Autoencoders and GANs

probabilistic gradient descent based encoder and a convolutional autoencoder. Detailed descriptions of all architectures and training algorithms are given in Figure 3 and Section 3 respectively.

The quality of generated images is hard to measure without a better estimate for its true distribution, so this can only be done by human judgement on small sample size. Section 4 provides a brief comparison.

Other aspects on how these models approximate P_X can, however, be measured: adapted models are trained over a modified version of MNIST that has 1000 classes in order to evaluate how many of the modes are captured by the domain of G and the results are presented in Figure 6. The dataset, Stacked-MNIST consists of three randomly chosen digits, stacked into different channels. This experiment is employed in [16] to show that it is possible to improve a regular DCGAN architecture with respect to mode collapse by combining ideas from VAEs and GANs. Intuitively, this can be explained as follows: while the GAN objective punishes G if creates unrealistic samples, the inability to generate data samples of certain modes is only punished indirectly, because D can learn that fake samples are over-represented in the other modes. In a high-modal dataset, e.g. with 1000 modes, if one mode is never covered by fakes and D doesn't learn how to distinguish fake samples from real ones for that single mode, it is not really an issue. In contrast, autoencoders are explicitly trained for encoding every mode. Even though this argument is the main motivation for the experiments in this paper, the results yield much better performance for the DCGAN, therefore no true improvement is observable. This is directly related to the encoding capability of G, which can also be evaluated using the reconstruction loss $MSE(X, G(E(X)))$. In order to evaluate properly these capabilities, several encoding approaches are compared with respect to image samples from:

- MNIST test set
- EMNIST test set of handwritten letters
- grey image
- random noise.

These experiments show that a well trained GAN is able to encode most images well, although not as accurately as a vanilla AE. The next experiment estimates the ratio between the latent code that G decodes to realistic digits and the latent code that is decoded onto meaningless images. To do this, an independently trained GAN detects outliers in a process similar to proposed in [4] with the assumption that outliers correspond to unrealistic images, while non-outliers are reasonable digits.

After this extensive analysis of the representation power, the focus shifts towards the nature of the learned encodings and to what degree they disentangle latent variables. More specifically, the linear separability of digit classes is compared for the latent space of the DCGAN, the latent space of the AE and the pixel space. Surprisingly, the classes are best separable in pixel space. Moreover, a plot on a two-dimensional hyperplane in the latent space shows the entanglement of the sets that belong to the same digit. Other authors reported better performance in this regard. factors that affect this are an interesting research topic, but outside of the scope of this work.

Architectures

As baseline architecture a DCGAN [12] is equipped with two different encoders: a convolutional neural encoder, which is referred to as E_{NN} and which is trained after G, and a gradient-descent-based encoder E_{GD} described in 1. As [15] proposes, and advantage of E_{GD} is, that it can be easily modified to perform image completion: defining the objective over the set of known pixels only, it is possible to find an image similar to the original one (only in the specified region), while the remaining parts are predicted by applying G to the $E_{NN}(X)$ found. An example is presented in Figure 4.

This approach is compared to a regular convolutional AE, that uses the exact same generator architecture. Furthermore, is employed an architecture similar to [16],: encoder, generator and discriminator are trained simultaneously, alternating the train-steps between GAN and AE objective. Since the architecture is not perfectly equal, the latter will be referred to as AEGAN. These architectures are depicted in Figure 2 and Figure 3.

The latent space $\mathcal{Z}$ is the 100 dimensional cube $[-1, 1]^{100}$. P_Z is the uniform distribution over $\mathcal{Z}$. All encoding networks use a tanh activation in their last layer, such that the output is also restricted to this set.

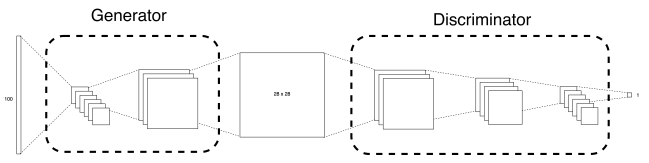


Figure 2: GAN Architecture

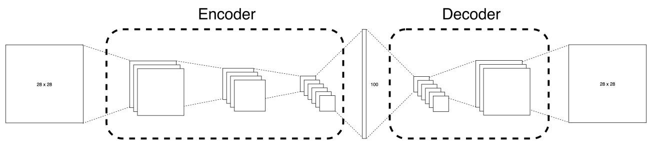


Figure 3: AE Architecture

```
Result: D-dimensional vector
$Z$ = 1000 samples of $P_Z$ ;
$z = argmin_{z \in Z} MSE(x, G(z))$ ;
for $E$ epochs do
    z -= lr $\cdot \delta MSE(x, G(z))/\delta z$ ;
    if $z \notin [-1, 1]^{100}$ then
        clip $z$;
    end
end
return $z$ ;
```

Algorithm 1: INVERSE(x): Encoding one sample for G. Gradient descent is performed in $\mathcal{Z}$ space with respect to $MSE(x, G(z))$. This objective is in general not convex, therefore it is possible to get stuck in local optima. To overcome this, we sample 1000 random vectors and take the one that minimizes our objective as the initial value for the optimization. Note that due to the randomness, this implements a probabilistic function.

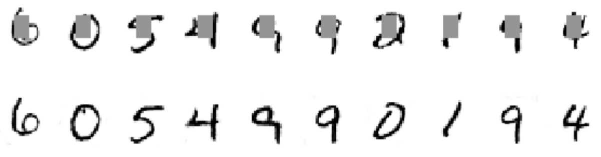


Figure 4: Using the models for image completion
The corrupted images (top) are inverted using a modified E_{GD} that optimizes only for the known pixels. The DCGAN (bottom) does a good job in this image completion task, although not explicitly trained for it.

Training

All models are trained on MNIST for 100 epochs and optimized following the Adam routine with `lr` = 0.002, $\beta = 0.5$ and a batch size of 128. The only exception are the models trained on Stacked-MNIST, which are trained for only 50 epochs.

The AE objective minimizes the loss function

$$J_{AE} := MSE(x, G(E(x))) \tag{1}$$

, whereas the GAN follows an adversarial procedure. The discriminator classifies real samples from generated ones (fakes), while the generator learns to fool the discriminator. The objective is to find $min_D max_G J_{GAN}(X, Z)$

$$J_{GAN}(X, Z) := min_D max_G \{KL(D(X), 1) + KL(D(G(Z), 0)\} \tag{2}$$

where $KL(\cdot, \cdot)$ is the Kullback-Leiber divergence.

For each train step of G, D is trained four times. Although it would have beed better to perform an extensive comparison of the training schedules, the limited access to GPUs does not allow that. Still, this schedule improves the training speed compared to a 1:1 ratio. As already mentioned, the AEGAN is trained by alternating between regular GAN train steps and AE train steps.

4 EXPERIMENTAL RESULTS

In this section, all experiments are explained in greater detail along with their results.

Generative Power

The generative power is a key property to distinguish the approaches. In order to visualize how the models do generate, 10 points are drawn randomly from the latent space and fed to the different generators, they can be found in Figure 5 and 6 for the Stacked-MNIST case. In Figure 7 the models have been used to interpolate between two samples from the test set. For this, the samples are encoded and the linearly interpolated vectors are decoded again.

Clearly, the AE does not yield legitimate digits. Even though it can represent almost any digit very well, only a tiny fraction of $\mathcal{Z}$ is decoded to realistic samples. The fake images generated by the DCGAN and the AEGAN look like digits, even if there is still room for improvement. A clear difference between the two GAN variations can't be found.

Mode coverage on Stacked-MNIST

A well-known problem of GAN training is mode collapse [16]: this phenomenon happens, when G creates insufficiently heterogeneous samples and neglects certain areas of the support of P_X. Therefore, a mode-collapsed-GAN can't encode all modes and is expected to have a high reconstruction loss when coupled with an encoder. [16] uses a dataset called

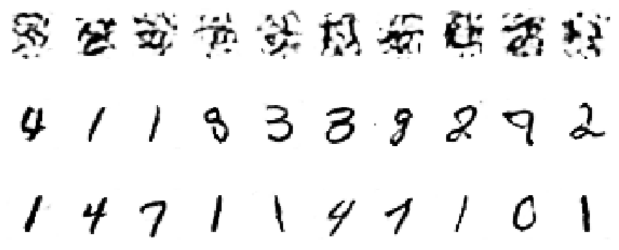


Figure 5: Random samples
AE [TOP], GAN [MIDDLE], AEGAN [BOTTOM]

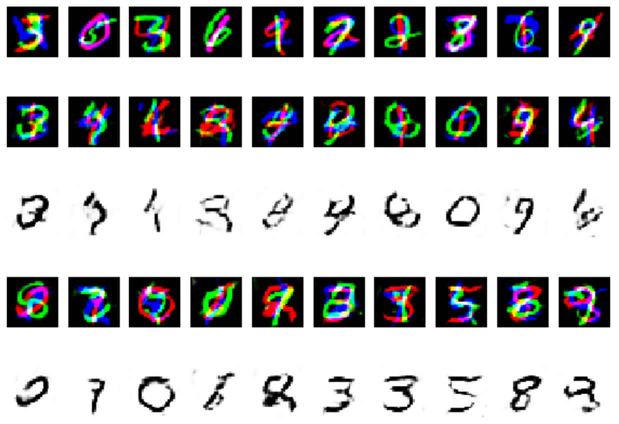


Figure 6: Stacked-MNIST
From top to down: (1) Samples of Stacked-MNIST, plotted as 3-channel RGB image. (2) Samples generated by a DC-GAN trained for 50 epochs. (3)A single randomly chosen channel of the samples above. (4) and (5) are the equivalent for an AEGAN.

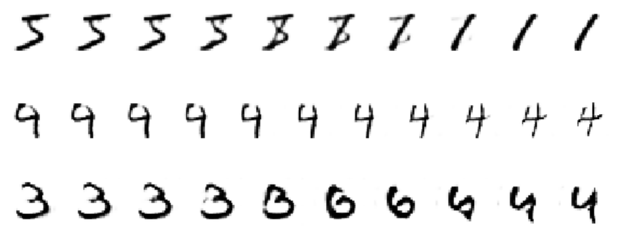


Figure 7: Latent space interpolation
AE [TOP], GAN [MIDDLE], AEGAN [BOTTOM]

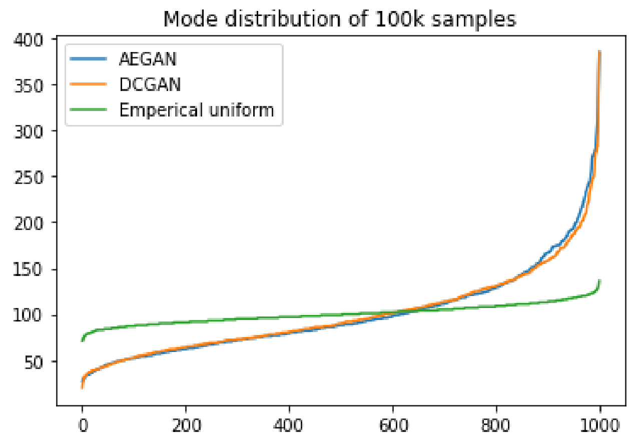


Figure 8: Mode coverage
10000 fake digits are generated with a DCGAN and AEGAN after 50 epochs of training, then a pre-trained classifier with over 99% test accuracy is used to label them. The plot shows the number of generated samples per class in ascending order. As a reference, the same number of labels are directly sampled from a uniform distribution. The plot shows that DCGAN and AEGAN have biases for and against certain modes, to almost the exact same extent.

Stacked-MNIST to compare the number of modes that different architectures can sample from. The dataset consists of three independently chosen MNIST samples stacked onto different colour channels, resulting in 10^3 classes. [16] reports that a regular DCGAN collapses onto 99 different modes only and that 150 modes can be achieved with their new architecture with a sample size of 26000. Surprisingly, the experiments for this paper show how the DCGAN generates samples from all the 1000 modes without any special modification. Nonetheless, the empirical distribution of labels in Figure 8 shows that biases for certain classes still exist and that, once again, no significant difference between AEGAN and DCGAN can be spotted.

Compressive Power

The compressive power measures the amount of information that is saved during the encoding. It is obtained computing the Mean Squared Error (MSE) between the original samples x and the reconstructions $G(E(x))$.

For the DCGAN there is no significant difference between E_{NN} and E_{GD} neither is terms human judgement, nor by MSE. The figures are made using E_{GD}. In Figure 9, the reconstructions are evaluated qualitatively not only for the MNIST test dataset but also for the related EMNIST dataset of handwritten characters, a grey image and some random Gaussian noise. While the reconstructed digit looks similar for all models, the AE is the only one that can compress a handwritten 'r' as well as digits. Unsurprisingly, on the grey and the noise image, all models fail, although the AE can produce some grey pixels, which neither DCGAN nor AE-GAN are able to do. A reason for this can be, that grey pixels hardly ever occur in the train set and would always be an indicator for images to be fake, but in the case of uncertainty between black and white, grey is the choice that minimizes the expected MSE, thus a reasonable output for the AE.

Figure 10 shows the expected reconstruction error for the models on digits versus letters: the AE has better performances than both DCGAN and AEGAN but, most importantly, the AEGAN worsens the reconstruction if the encoding is done by E_{NN} and only slightly improves it if the samples are encoded through Algorithm 1. Since hyperparameter evaluation is omitted (e.g. neither learning schedule nor weighting different objectives is performed), there may be room for improvement in this regard.

Since all architectures show a clear difference in the reconstruction error of the datasets, it is possible to use them for outlier detection, as done in [4] and [2]. This approach is employed for the following experiment.

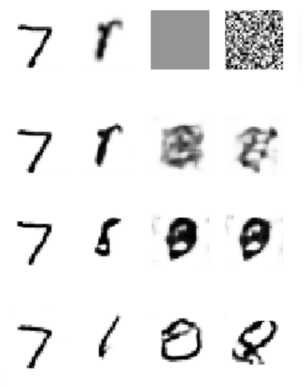


Figure 9: Reconstruction error samples
From top to bottom: [1] Original Samples, [2] AE Reconstruction, [3] AEGAN Reconstruction, [4] GAN Reconstruction

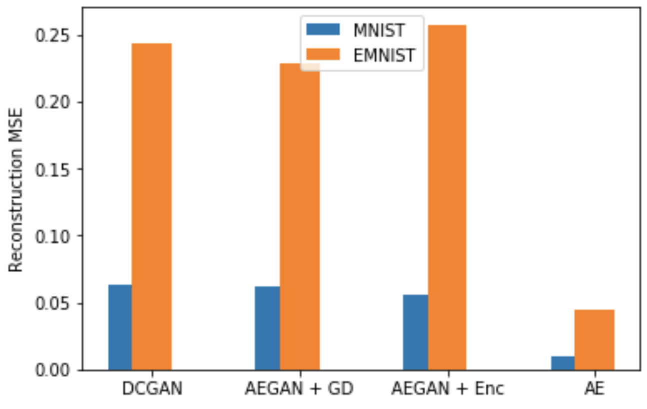


Figure 10: Reconstruction error bar-plot

Density. Dependently on their objective function, GAN and AE learn which parts of the latent space to use to encode realistic samples. Most likely, both latent spaces have some regions mapped onto realistic samples and other regions that lead to meaningless images. It is possible to understand how much of the latent space is used to encode proper samples using outlier detection techniques, therefore it is possible to estimate the density of encoding that each latent space sustains. To actually detect the outliers in the latent space, a fakeness threshold is found through an independently trained DCGAN. The workflow is rather simple: all the MNIST training samples are encoded in the latent space using Algorithm 1, then they are reconstructed using the GAN generator and the error is recorded for each point. Since the procedure in Algorithm 1 is not convex, there might be some extreme case leading to high reconstruction errors, so, to ignore these cases, the threshold is defined as the maximum value between the top 99 percentile of reconstruction error on the MNIST test set. All the images with a higher reconstruction error are then treated as outliers. This decision is taken based on Figure 12, which shows the reconstruction error of the outlier GAN for the MNIST test set. This technique shows how many randomly sampled latent vectors are decoded to realistic images.

If the DCGAN architecture is biased towards creating a certain type of unrealistic samples, these outliers will not be detected. However the outlier detector seemed to be coherent to the understanding of correct images, this it is possible to assume that the estimate is not too far away from the truth.

It is also worth mentioning that, this time, the reconstruction error is computed as the L1 norm of the difference between original and reconstructed samples: $\| X - X^{rec} \|_{L1} = \sum_i |x_i - x_i^{rec}|$.

The density of encoding is the ratio:

$$DoE = \frac{\#non\_outliers}{\#samples} \tag{3}$$

In a sample size of 200 000, 86% of random latent vectors are mapped onto legit digits by the DCGAN. Instead, the AE doesn't lead to even one single legit sample, thus DoE for the AE is capped by 0.0005% of non-outliers. This is summarized in Table 2.

A comparison between GAN and AE in Figure 11 shows the worst outliers of the former versus the best outliers of the latter. Since the difference between outliers is not that high, it can be stated that, despite the performance loss, GANs could be still be used for AE tasks but AEs cannot be used at all to for typical GAN tasks like generating samples.

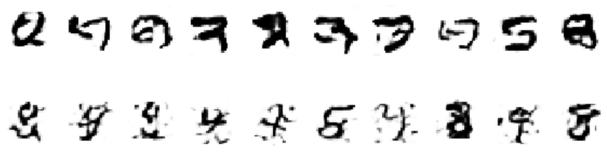


Figure 11: Outliers
GAN Worst [*TOP*], AE Best [*BOTTOM*]

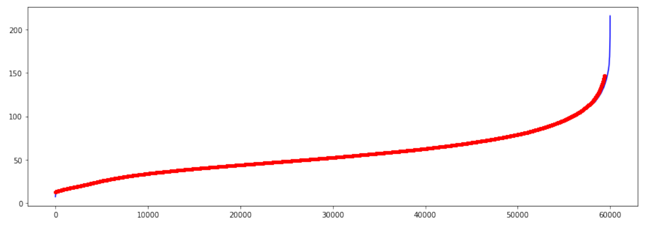


Figure 12: MNIST reconstruction errors
The red line corresponds to 99% of the samples.

Disentanglement

Different types of encodings have different properties such as ease of discrimination/classification. The latter is crucial for any kind of embedding space and measures how easy it is, for a general classifier, to correctly classify the examples. To answer this question is measured the linear separability for digits in pixel space and in the latent spaces of the DCGAN and the AE. The results are shown in Table 1.

The classes are more entangled in the latent spaces, i.e. the linear separability decreased by encoding the data. To understand the shape of the sets that belong to a class, the decision borders are plotted onto an arbitrarily chosen two-dimensional hyperplane within the respective latent spaces. Figure 13 and 14 shows that: each point on the grid is associated to a vector in $\mathcal{Z}$ by a linear transformation, which is then decoded by the respective G and fed to a classifier. The colours indicate the predicted labels. It is possible to see how the sets that belong to the same digit are entangled and non-convex, which means there are cases where the interpolation between two latent samples, leading to the same digit, leads to a different digit.

Table 1: Balanced Accuracy

Space	Balanced Accuracy
GAN	0.7908
AE	0.8732
Image	0.9215

Table 2: Density of Encoding

Model	DoE[%]
GAN	86.76
AE	<0.0005

5 CONCLUSION

This paper gives an in-depth comparison of the properties of both GAN and AE. The analysis of the encoding power concerning arbitrary samples from the training distribution shows how the DCGAN can learn how to generate 1000 classes, although they don't appear to be balanced. Moreover, given a pre-trained GAN, it is possible to find the encoding of a given image such that the reconstruction is similar. This suggests that GANs are applicable wherever autoencoders are applied, however, we AEs still yield more accurate reconstructions. Nonetheless, this performance decreases if an image is sampled from a different distribution. Using a GAN-based outlier detection technique, it is possible to estimate that a high percentage of the volume of the latent space is decoded to realistic images, whereas the density of encodings that belong to realistic digits is extremely small for the AE. Furthermore, a combined approach similar to [5] is evaluated, but no improvement in the behaviour of the DCGAN is found. Finally, despite the good generating power and the suitability for high-level image editing tasks such as image completion, the learned representation does not lead to linearly separable digit clusters.

Future Work

The finding that classes become less linearly separable in the latent space of a GAN is rather surprising, and yet it is hard to provide an explanation for this. To examine the factors that lead to a disentangled latent space is an interesting research question. For this, it would be nice to compare the performance of a Variational Autoencoder to the other models that are already explored in this paper.

ACKNOWLEDGMENTS

To the supervisor Thomas Goerttler, who assisted the team during the whole project, and all organizers of the module Neural Information Processing Project.

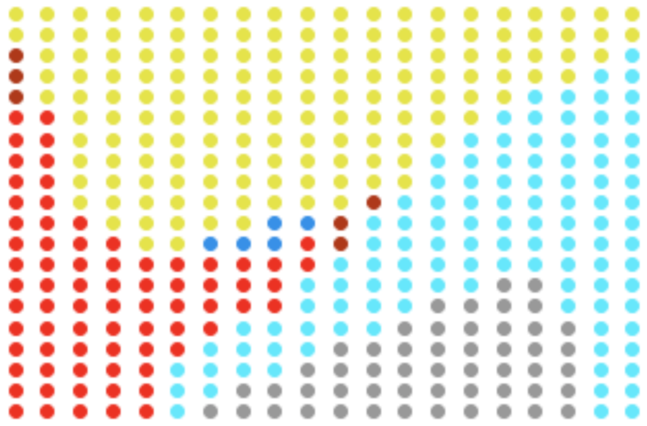


Figure 13: Decision borders on embedded 2D hyperplane in DCGANs latent space

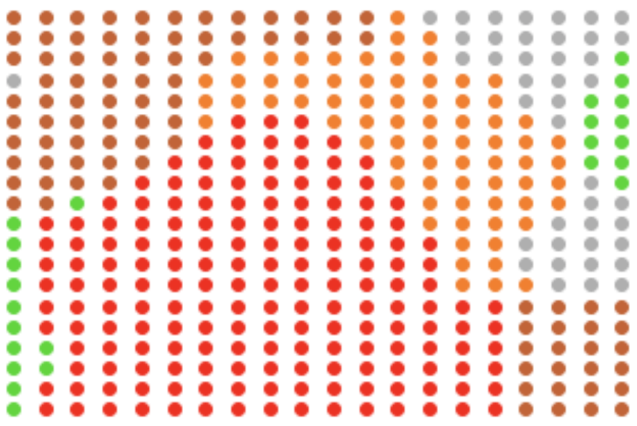


Figure 14: Decision borders on embedded 2D hyperplane in AEs latent space

REFERENCES

[1] [n.d.]. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. http://yann.lecun.com/exdb/mnist/

[2] Jinghui Chen, Saket Sathe, Charu C. Aggarwal, and Deepak S. Turaga. 2017. Outlier Detection with Autoencoder Ensembles. In *SDM*. https://doi.org/10.1137/1.9781611974973.11

[3] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and AndrÃľ van Schaik. 2017. EMNIST: an extension of MNIST to handwritten letters. *arXiv:1702.05373 [cs]* (Feb. 2017). http://arxiv.org/abs/1702.05373 arXiv: 1702.05373.

[4] Lucas Deecke, Robert Vandermeulen, Lukas Ruff, Stephan Mandt, and Marius Kloft. 2019. Image Anomaly Detection with Generative Adversarial Networks. In *Machine Learning and Knowledge Discovery in Databases*, Michele Berlingerio, Francesco Bonchi, Thomas GÃďrtner, Neil Hurley, and Georgiana Ifrim (Eds.). Vol. 11051. Springer International Publishing, Cham, 3–17. https://doi.org/10.1007/978-3-030-10925-7_1

[5] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. 2016. Adversarial Feature Learning. *CoRR* abs/1605.09782 (2016). arXiv:1605.09782 http://arxiv.org/abs/1605.09782

[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]* (June 2014). http://arxiv.org/abs/1406.2661 arXiv: 1406.2661.

[7] Ralph Gross. 2005. Face Databases. In *Handbook of Face Recognition*, A.Jain S.Li (Ed.). Springer, New York.

[8] Anders Boesen Lindbo Larsen, Soren Kaae Sonderby, Hugo Larochelle, and Ole Winther. 2016. Autoencoding beyond pixels uding a learned similarity metric. 2, 2 (feb 2016), 16. https://arxiv.org/pdf/1512.09300.pdf

[9] Junyu Luo, Yong Xu, Chenwei Tang, and Jiancheng Lv. 2017. *Neural Information Processing* (2nd. ed.). Lecture Notes in Computer Science, Vol. 10635. D. Liu et al., 207–216. https://link.springer.com/chapter/10.1007%2F978-3-319-70096-0_22

[10] Pengyuan Lyu, Xiang Bai, Cong Yao, Zhen Zhu, Tengteng Huang, and Wenyu Liu. 2017. Auto-Encoder Guided GAN for Chinese Calligraphy Synthesis. *IAPR* (apr 2017), 6. https://doi.org/10.1109

[11] Alireza Makhzani, Jonathon Shlens Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. 2016. Adversaria Autoencoders. 2, 2 (may 2016), 16. https://arxiv.org/pdf/1511.05644.pdf

[12] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv:1511.06434 [cs]* (Nov. 2015). http://arxiv.org/abs/1511.06434 arXiv: 1511.06434.

[13] Mihaela Rosca, Balaji Lakshminarayanan, David Warde-Farley, and Shakir Mohamed. 2017. Variational Approaches for Auto-Encoding Generative Adversarial Networks. 2, 2 (oct 2017), 21. https://arxiv.org/pdf/1706.04987.pdf

[14] Tim Sainburg, Marvin Thielk, Vrad Theilman, Benjamin MIgliori, and Timothy Gentner. 2019. Generative Adversarial Interpolative Autoencoding: Adversarial Training On Latent Space Interpolations Encourages Convex Latent Distributions. 3, 3 (apr 2019), 16. https://arxiv.org/pdf/1807.06650.pdf

[15] Christopher R. Sauer, Russell Kaplan, Alexander Lin Stanford, and Serra Mall. 2016. Neural Fill : Content Aware Image Fill with Generative Adversarial Neural Networks.

[16] Akash Srivastava, Lazar Valkov, Chris Russell, Michael U. Gutmann, and Charles Sutton. 2017. VEEGAN: Reducing Mode Collapse in GANs using Implicit Variational Learning. *arXiv:1705.07761 [stat]* (May 2017). http://arxiv.org/abs/1705.07761 arXiv: 1705.07761.

[17] Ngoc-Trung Tran, Tuan-Anh Bui, and Ngai-Man Cheung. 2018. *Computer Vision - ECCV 2018* (14th ed.). Lecture Notes in Computer Science, Vol. 11218. Springer, Cham, 387–401. https://doi.org/10.1007/978-3-030-01264-9_23

[18] Zhifei Zhang, Yang Song, and Hairong Qi. 2017. GANs Powered by Autoencoding - A Theoretic Reasoning. (2017), 8. https://web.eecs.utk.edu/~zzhang61/docs/papers/2017_ICMLWS_AEGAN.pdf

A ONLINE RESOURCES

The code is publicly available on GitHub at www.github.com/nielsrolf/GAN-TUBerlin.

B DETAILS ABOUT LINEAR SEPARABILITY

The linear classifier is trained using multinomial Logistic Regression, also known as Softmax Regression (SR), on MNIST, encoded in the latent space of both models and non-encoded as well. Its performance can tell us how well separated the latent points corresponding to different classes are.

Most importantly, even though SR is worse than other classification techniques, it is extremely robust due to its convex loss, which means that, given enough epochs and a proper learning rate, the global optimum is always reached, avoiding any oscillation in the performances. Considering K classes such that $y(i) \in \{1...K\}$ and the dataset $D = \{[\vec{x}_i, y_i]\}_{i=1}^{N}$, the Softmax Regressor is trained over the Cross Entropy objective in Equation 4 and optimized my means of simple GD.

$$J(\theta) = -\left[\sum_{i=1}^{N} \sum_{k=1}^{K} 1[y_i = k] \log \frac{\exp(\vec{\theta}_k^T \vec{x}_i)}{\sum_{j=1}^{K} \exp(\vec{\theta}_j^T \vec{x}_i)} \right] \tag{4}$$

The performances are measured through 4 metrics:

- Balanced Accuracy
- Multinomial Precision
- Multinomial Recall
- Multinomial F1

The SR classifier, tested on the encoded and non-encoded MNIST test-set, outputs the performances explained in the confusion matrices in Figures 15-16-17, for each space. The distribution of colours on the main diagonal shows that the error for each class is proportional in all three cases. Table 1 contains the balanced accuracy for each model and shows how the GAN embeddings are worse than both AE embeddings and image space ones.

These findings are explainable by looking at the latent spaces and image space, projected onto their two dimensions that explain the most variance, found by means of Principal Component Analysis (PCA); Figure 18, 19, 20. The GAN latent space is the most uniform one, while the other two spaces are much more scattered and, most importantly, do not use all the available space. This is expected since the GAN objective explicitly tries to encode the whole latent space onto meaningful digits, so to elude the discriminator check. Using the whole latent space, the latent classes are naturally more entangled, thus harder to separate linearly.

Table 3: Precision

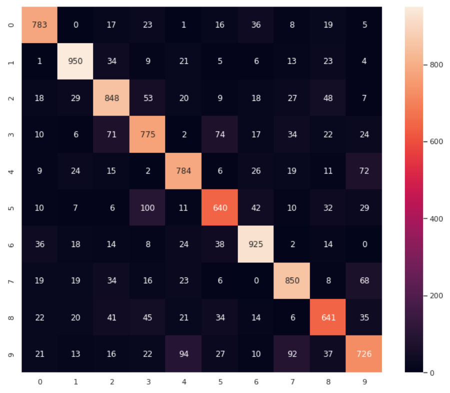
Class	GAN	AE	Image
0	0.8428	0.9025	0.9575
1	0.8747	0.9489	0.9585
2	0.7737	0.8707	0.9269
3	0.7359	0.8416	0.9034
4	0.7832	0.8745	0.9360
5	0.7485	0.8147	0.8960
6	0.8455	0.9155	0.9323
7	0.8011	0.8848	0.9328
8	0.7497	0.8335	0.8761
9	0.7484	0.8445	0.8976

Table 4: Recall

Class	GAN	AE	Image
0	0.8623	0.9448	0.9765
1	0.8911	0.9665	0.9779
2	0.7873	0.8488	0.8972
3	0.7487	0.8633	0.9079
4	0.8099	0.8940	0.9246
5	0.7215	0.7690	0.8699
6	0.8572	0.9164	0.9498
7	0.8149	0.8822	0.9192
8	0.7292	0.8121	0.8788
9	0.6862	0.8344	0.9127

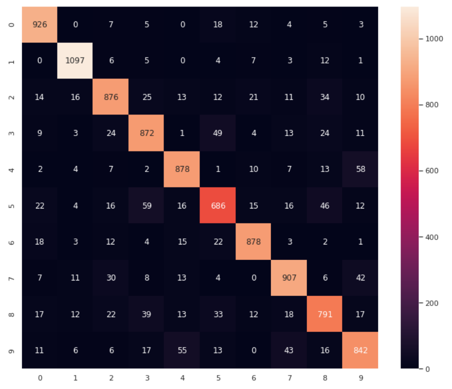
Table 5: F1

Class	GAN	AE	Image
0	0.8524	0.9232	0.9666
1	0.8828	0.9576	0.9681
2	0.7804	0.8596	0.9118
3	0.7423	0.8523	0.9056
4	0.7963	0.8841	0.9303
5	0.7347	0.7912	0.8828
6	0.8513	0.9160	0.9410
7	0.8079	0.8835	0.9260
8	0.7393	0.8226	0.8774
9	0.7159	0.8394	0.8774



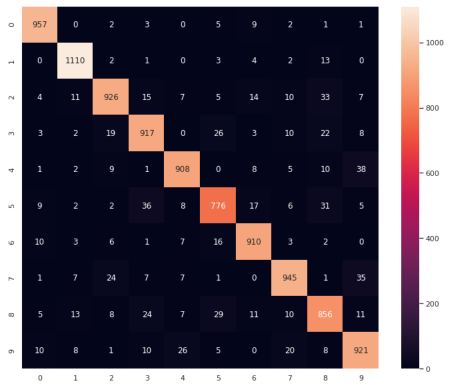
	0	1	2	3	4	5	6	7	8	9
0	783	0	17	23	1	16	36	8	19	5
1	1	950	34	9	21	5	6	13	23	4
2	18	29	848	53	20	9	18	27	48	7
3	10	6	71	775	2	74	17	34	22	24
4	9	24	15	2	784	6	26	19	11	72
5	10	7	6	100	11	640	42	10	32	29
6	36	18	14	8	24	38	925	2	14	0
7	19	19	34	16	23	6	0	850	8	68
8	22	20	41	45	21	34	14	6	641	35
9	21	13	16	22	94	27	10	92	37	726

Figure 15: GAN SR Confusion Matrix



	0	1	2	3	4	5	6	7	8	9
0	926	0	7	5	0	18	12	4	5	3
1	0	1097	6	5	0	4	7	3	12	1
2	14	16	876	25	13	12	21	11	34	10
3	9	3	24	872	1	49	4	13	24	11
4	2	4	7	2	878	1	10	7	13	58
5	22	4	16	59	16	686	15	16	46	12
6	18	3	12	4	15	22	878	3	2	1
7	7	11	30	8	13	4	0	907	6	42
8	17	12	22	39	13	33	12	18	791	17
9	11	6	6	17	55	13	0	43	16	842

Figure 16: AE SR Confusion Matrix



	0	1	2	3	4	5	6	7	8	9
0	957	0	2	3	0	5	9	2	1	1
1	0	1110	2	1	0	3	4	2	13	0
2	4	11	926	15	7	5	14	10	33	7
3	3	2	19	917	0	26	3	10	22	8
4	1	2	9	1	908	0	8	5	10	38
5	9	2	2	36	8	776	17	6	31	5
6	10	3	6	1	7	16	910	3	2	0
7	1	7	24	7	7	1	0	945	1	35
8	5	13	8	24	7	29	11	10	856	11
9	10	8	1	10	26	5	0	20	8	921

Figure 17: Image SR Confusion Matrix

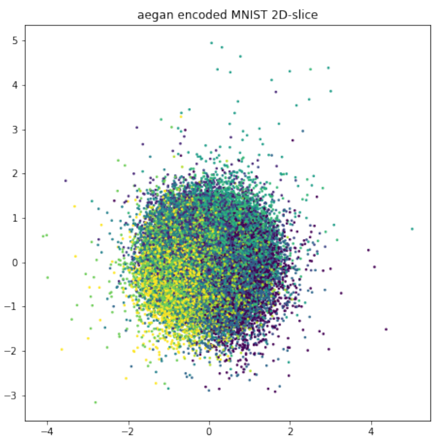


Figure 18: GAN Space PCA

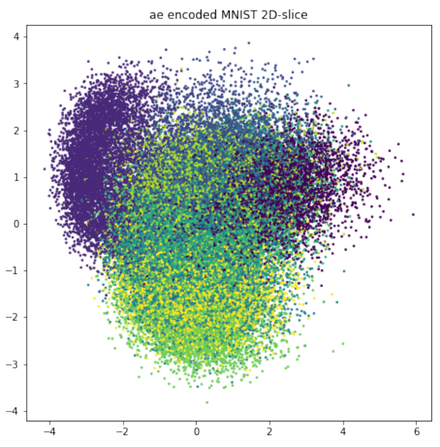


Figure 19: AE Space PCA

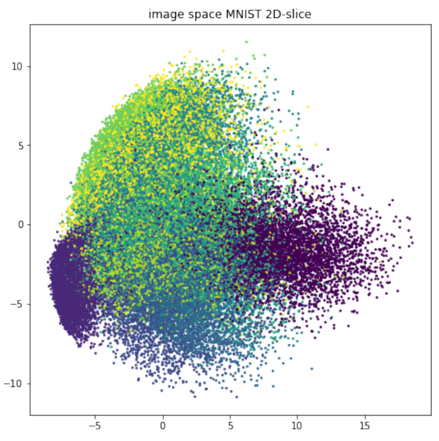


Figure 20: Image Space PCA