Augmenting the latent space of generative models

Kermedchiev, Anastas M.

Award date:
2022

Awarding institution:
Tilburg University

Augmenting the latent space of generative models
Anastas Kermedchiev 1282042

Data science, TU/e, Tilburg university
Bachelor end project report

Abstract- Autoencoders (AE) produce their results by computing an encoding of the underlining latent features of the data, in order to be able to recreate the original dataset. Ideally these encodings learn the most important underlining features of the data and output them in a latent space vector, describing each of the variables by their features. In this project a great effort was made to experiment and augment these encodings, in order to produce more unique and varied results than the original recreation of the AE. For this purpose, an Autoencoder was trained on 5 separate datasets, the latent space vectors of which have been used to train 3 different generative algorithms which attempt to recreate, and augment said latent space vectors. These algorithms include VAE-GANs, Sum-Product Networks and RealNVP Normalizing Flows. These augmentations are conducted in order to produce more unique and varied recreations and to check if it is feasible, beneficial or even possible to do so.

Keywords- latent space, autoencoders, sum-product networks, normalizing flows, VAE-GANs, generative models

I. INTRODUCTION

Generative models are statistical neural networks that learn a feature space of a dataset in order to be able to sample from it and output similar yet unique variables. These models are mostly applied to images, although they can be used in other fields as well. There are different types of generative models, such as ones that learn a probabilistic distribution, such as Sum-Product Networks (SPNs), Variational Autoencoders (VAEs), Normalizing flows (NFs), while others learn feature space vectors in a deep dense network, such as regular Autoencoders (AE) and General Adversarial Networks (GANs). Then there are models which combine both methods like the VAE-GAN network. All three types have one thing in common, which is they all compute a latent space representation that describes the features of the data given, in order to sample from it and produce unique recreations. This latent space is called a posterior estimation. It refers to how likely the latent variable is given the input, hence in training we want to learn a good posterior approximation, which explains the input dataset. The posterior is denoted by the probability of a latent variable z, given our input x variable – $p(z|x)$, which is approximated by the model. The prior on the other hand represents how the latent variables z are represented without conditioning on the input x, hence $p(z)$. In this project we would like to train an initial model that learns the posterior estimation of a dataset of images, then approximate and sample the distribution of this posterior estimation using generative models. This way we are altering the prior of the initial model to create new varying results. This method could also possibly be used to learn the prior of a generative model that doesn't define it's prior explicitly.

II. BACKGROUND

Autoencoders and Variational Autoencoders are a widely used unsupervised learning technique in which neural networks are used to learn a feature representation (latent space) of the input in order to recreate said input as closely as possible [1]. AEs and VAEs are composed of two neural network models, the encoder and the decoder. The encoder reduces the initial input's dimensions and learns a latent space, or a feature vector, for each of the variables in the sample input. The decoder on the other hand has the task of recreating the input data that was fed to the encoder as closely as possible, given the computed latent space. This latent space has special properties, like the fact that if points close to each other in the latent space are drawn, they will produce very similar results after going through the decoder. This is called the continuity property. Another property of the latent space is that every point in it produces a meaningful output after it has been decoded. This is called the completeness property. These properties are true for both AEs and VAEs. The differences between the two, however, lie in the way the latent space is computed and in their loss functions. AEs encodes the input data into a latent space vector, while VAEs encode the input data into a Gaussian distribution by computing mean and variance vectors. The loss of an AE I simply a reconstruction loss, which measures the difference between the initial input and the generated output. The loss of a VAE is the reconstruction loss combined with a KL-divergence term, which encourages the estimated posterior distribution $q_\phi(z|x)$ to match the prior distribution $p(z)$ for each z, which acts as a regularizer in training.

Generative Adversarial networks are probably the most popular generative models, because of their impressive and creative recreations as well as the recent popularization of deep fakes [5]. It is a general dense deep network that is composed of two separate models, those being the generator and the discriminator. The generator model's job is the same as an Autoencoder's one, which is to capture the distribution of the data and to recreate the input data as closely as possible, however it achieves this in a different way as opposed to the AE. The input to the generator is not the input dataset, instead it is random noise vector sampled from a Gaussian distribution. The generator then learns the input data distribution with the help of the discriminator. The discriminator is a classifier that predicts whether an image is a generated one or a recreation of one. If the discriminator is not able to distinguish between the real and generated images, then the generator has learned the features of the input and can recreate them well. The two models therefore play a complicated minimax game as they learn the input. A powerful variation of

the GAN model is the VAE-GAN [3]. By replacing the standard generator of a GAN with an Autoencoder learned representations by the discriminator can be used for the VAE reconstruction, replacing element-wise errors with feature-wise errors to better capture the distribution. This model can also learn embeddings of high-level abstract visual features, which can be modified.

Sum-Product Networks [2] are Directed Acyclic Graphs that represent complex probability distributions in a compact way. SPNs consist of the input variables as input at the leaves, sums and products as nodes and weighted edges. The product node simply takes the product of the nodes preceding it, while the sum node takes the sum of the nodes preceding it multiplied by the weights of each edge. The interesting part is that the leaves are actually probability densities and taking the product of two densities is the same as computing the factorization between them, while taking the sum of two densities multiplied by their respective weights is the same as taking the mixture of the two. This means that SPNs can describe very complicated distributions, very cheaply and efficiently. In order for SPNs to be valid they have to obey three rules. The first one is completeness, which implies that each sum node has children with the same scope. The second one is consistency, implying that no variable appears negated in one child of a product node and non-negated in another. The third and last one is decomposability, implying that the children of a product node cannot have the same scope. If these conditions are met the SPN is considered valid and its root note (distribution of the input variables) can be computed recursively. Training an SPN is similar to training any neural network, using gradients or Expectation Minimization on batches. Once an SPN has been trained it can also be sampled from by computing the Most Probable Explanation for all input variables, given a random draw from the density distribution at the root node.

Normalizing flow models are models that actually learn the true distribution of the input data [8][11]. Normalizing flows are very simple bijections, where invertible flow functions $f(x)$ are trained to map the complicated input distribution $p_x(x)$ to a pre-determined base measure $p_z(z)$, usually a Gaussian distribution. The flow functions need to be invertible and differentiable, in order to make it possible to train and sample from the base measure using said functions. Training of the flow functions is done by maximum (log-)likelihood. A linear transformation can be a flow function, however linear flow functions are inexpressive, and the determinant could be $O(d^3)$ and if d is large, like with pixels of images for example, this becomes very expensive to compute. For this reason, Coupling flows are preferred in generative normalizing flows. A Coupling flow is a general approach to construct non-linear flow functions, where the parameters are split into two disjoint subsets and invertible differential transformation is applied to only one of them. By changing the coupling splits and stacking a number of coupling flows one after the another we ensure full expressiveness. There are different coupling transformations, although the most significant one was first introduced in the RealNVP paper [9] and that is the affine transformation. With this transformation it is possible to learn the true distribution of complicated image datasets computationally inexpensively. Backpropagation is used to train these models which also makes them easier to implement than other generative models.

III. Methodology

Datasets

For the first half of the project the MNIST digits [17] dataset was used to get familiar with the techniques of generative models. This dataset contains handwritten numbers in the range of 0-9 in grayscale. It contains 70000 images with a 60000/10000 split for training and testing. The images have a shape of 28x28x1 pixels, each pixel having a value between 0 and 255, which is a black and white scale with 0 as black and 255 as white. This scale was normalized by dividing each pixel by 255, which produces float values between 0 and 1. The reason for choosing this dataset for the start of this project is because of its complexity, which makes it easier to train on and get quick results from simple tweaks in the models. In the second half of the project the models presented were tested on more complex datasets to see if they can handle variation bigger than the one present in the MNIST numbers dataset.

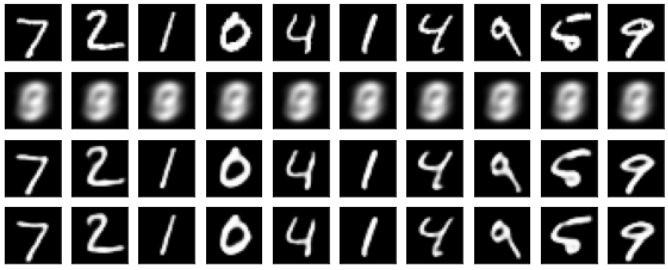
The first more complex dataset was the Olivetti faces, which includes ten different pictures of 40 distinct subjects with varying lighting, facial expressions and facial details, summing up to 400 unique pictures. Each image has a shape of 64x64x1 pixels, making them grayscale as well, however this time they come already normalized, with values between 0 and 1. The Olivetti faces dataset is a good representation of how the models handle human faces, without the added complexity of a complicated background.

The next dataset that was implemented was the fashion MNIST dataset, which includes 70000 images of 10 distinct classes split into 60000 training and 10000 test images. These 10 classes include T-shirts, pullovers, dresses, coats. Sandals, shirts, sneakers, bags and ankle boots. Each image has a shape of 28x28x1 pixels with each pixel having a value between 0 and 255, which were normalize to values between 0 to 1. This dataset is good for benchmarking how well the models perform for images portraying a large variety of different unique shapes.

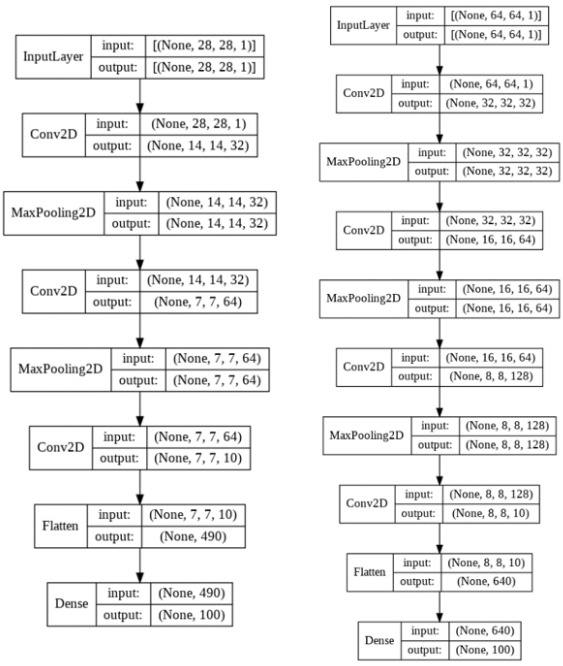
The second most complex dataset that was implemented is the CIFAR10 dataset [18], which includes 60000 images of 10 distinct classes split into 50000 training and 10000 test images. These classes include airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships and trucks. This is one of the hardest datasets to train on, because of the huge variety of each of the classes as well as the fact that the images are RGB, therefore it is expected for the results to be less accurate than the other datasets. The shape of the images is 64x64x3 pixels, with each RGB channel having values between 0 to 255, which were all normalized to values between 0 and 1. The last and definitely the biggest dataset implemented was the CelebA dataset [19], which contains more than 200000 high quality celebrity images each with a shape of 178×218x3 pixels. Needless to say, this is a huge step-up in computing complexity and for this reason a sample of 15000 images were taken with a smaller reshaped size of 64x64x3 pixels. Each image was also normalized to have values between 0 and 1 for each RGB channel instead of 0-255. Sadly, this is the only dataset for which the labels were not implemented, because the images were loaded from a mounted google drive folder containing the stored images, which doesn't include image labels.

INITIAL AUTOENCODER

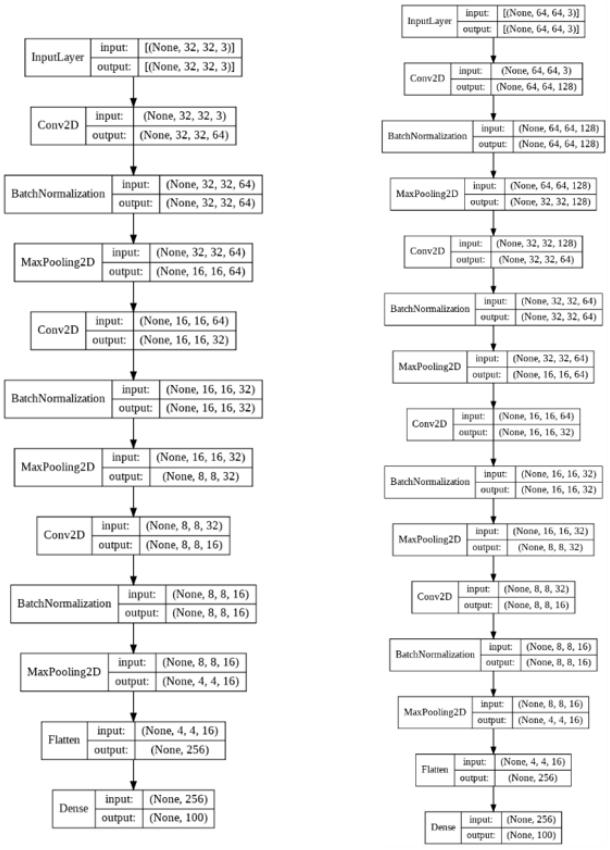
I order to produce the latent space vectors a convolutional autoencoder was trained to recreate the initial datasets as closely as possible. A great effort was made for the computed posterior to be as good as it can get before continuing to the augmentations. A big factor to consider is the size of the latent space vectors generated, since that plays a major role on the quality of the reconstructions. To demonstrate this an AE was trained on the MNIST digits dataset for different latent space sizes, namely 25, 100 and 490. The following results are ordered by rows, where the first row is the test set followed by recreations of the test set with a latent space vectors of size 25, 100 and 490 respectively:



The interesting thing is that the difference between recreations with latent space vectors of size 100 and 490 is not that big, which is very useful when trying to run complex models in reasonable time. For this reason, the initial Autoencoders have a bottleneck of 100. The AE is modeled in two parts, namely the encoder and decoder. The encoder is always set to reduce the shape of the input to a vector with earlier specified shape of 100, which is accomplished by taking advantage of the reduction of the image size and its transformation to a feature map as the input passes through the convolutional layers. This is illustrated in the architecture of the autoencoder. It is to be noted that a slightly different architecture was implemented to account for the larger input shapes of the Olivetti faces, CIFAR10 and CelebA datasets, as compared to those of the MNIST digits and fashion MNIST datasets. The architectures for the encoders are as follows:

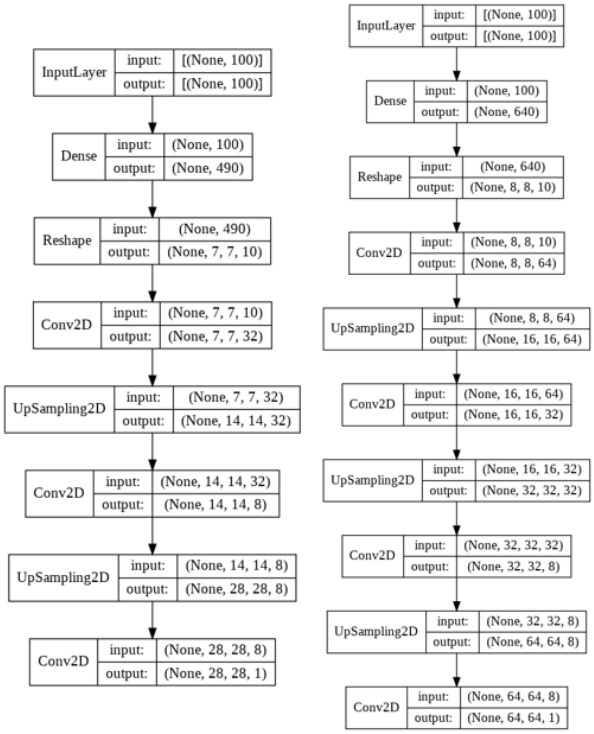


The encoders of the Olivetti and MNIST datasets,

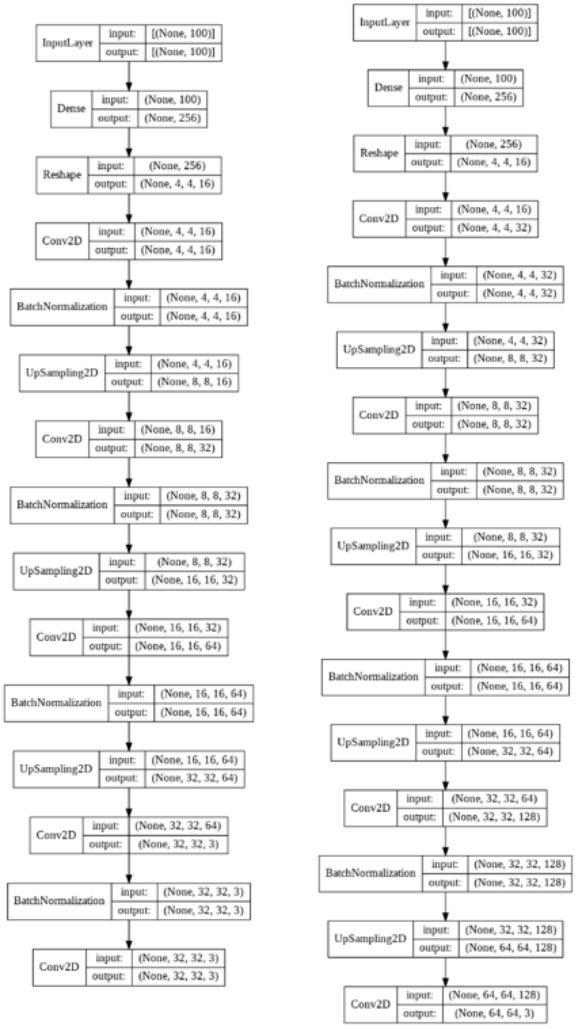


The encoders of the CIfar10 and CelebA datasets

Each of the Convolutional layers has is a 3x3 kernel with a stride of 2 and a ReLu activation. Each of the MaxPooling layers have a 2x2 kernel with padding to preserve the shape in the output. The last Dense layer of the autoencoders have a sigmoid activation. The reason for this is to make the input for the models in the latter half of this paper have values between 0 and 1, which greatly benefits the accuracy of those models. A hyperbolic tangent activation was also experimented with for the last layer. This greatly reduced the accuracy of the initial autoencoder, defeating the whole purpose.
The second half of the initial AE is the decoder. The decoder implements Transposed Convolutional layers that reverse the standard convolution by dimensions only. Alongside Up-sampling layers, which are simple layers with no weights that double the dimensions of input, the layers of the decoder have the task to restore the initial shape of the images. Like the encoder the shape of the decoder varies between datasets in a similar manner. The architectures of the decoders are as follows:



The decoders of the MNIST and Olivetti datasets



The decoders of the CIfar10 and CelebA datasets

In the decoder all the Up-sampling layers have a kernel size of 2x2 and all the Transpose Convolutional layers have a kernel size of 3x3 with padding and a Sigmoid activation. This is because the last Dense layer of the encoder has a Sigmoid activation and setting them up like this produced the best results. The last Convolutional layer of the decoder has a filter with the number of channels of the initial images, 1 for grayscale images and 3 for RGB ones. The reconstruction loss function of choice was Mean Squared Error. As opposed to Binary Cross entropy, which is often the default choice for AEs, optimizing for MSE means the generated output intensities are symmetrically close to the input intensities. Cross entropy on the other hand is asymmetrical, meaning that values far away from the true y will get penalized more harshly than with MSE. The Adam optimizer [7] was chosen for the initial AE

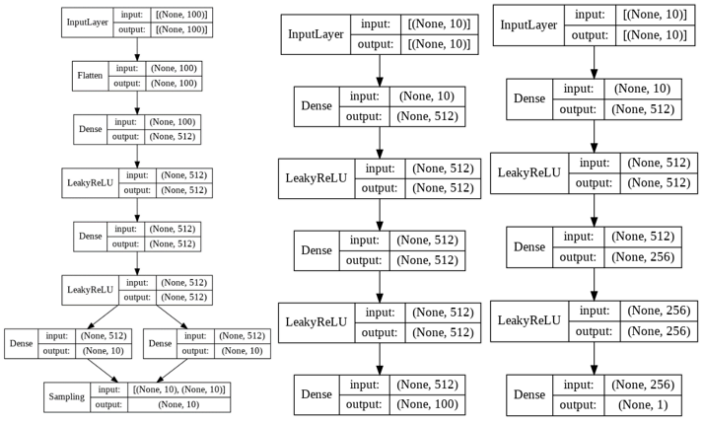
EVALUATING RECONSTRUCTIONS OF IMAGES

In order to be able to draw conclusions on the quality of results produced by generative models of any kind a metric that encompasses variability, creativity and similarity is needed. Mean Squared Error, Cross entropy and Euclidian Distance portray evaluate recreations only by the pixel difference between

actual image and reconstruction, failing to account for the fact that two images may be very similar in concept and shapes, yet very different numerically. For the purpose different versions of the Fréchet Distance were implemented for different generative models with the Fréchet Inception Distance (FID) [4],[10] being the most popular and widely used. Implementing the FID includes resizing, then computing embeddings of the two image datasets to be evaluated using the InceptionV3 model, then computing the Fréchet Distance of those two embeddings [12]. The FID was calculated for all models listed in this research, however there are some shortcomings to the FID. Firstly, it uses a pre-trained Inception model, which may not capture all features, resulting in a higher FID score. Secondly, the FID score is computed using limited statistics like the mean and covariance, which can result in two sets of images which look nothing alike to have very similar FID scores overall. And last, but not least, it can be computationally heavy, since a large sample of images is recommended, as well as the fact that the Inception model is trained on high quality images of 229x229, which can increase complexity for images much smaller like the MNIST dataset.

VAE-GAN

The VAE-GAN was the first model to be implemented on the learned posterior of the initial AE. The VAE-GAN was built from the ground up using TensorFlow. Because the inputs are not images, Dense layers were used for the encoder, decoder and discriminator of the model. Since a constant shape of 100 was decided for the latent space vectors for each dataset, the same VAE-GAN model was implemented for all datasets. The architectures for the encoder, decoder and discriminator respectively are presented as follows:



The generator (VAE) uses a Gaussian prior to estimate its posterior distribution, which is indicated as Sampling in the architecture displayed above. Leaky ReLu activations with an alpha of 0.2 are spread out throughout encoder, decoder and discriminator parts of the model, because those are the activations used in the literature as well. The last Dense layer of the generator (VAE) has a Hyperbolic Tangent activation, while the last Dense layer of the discriminator has a Sigmoid layer that predicts if the recreated image batches are indistinguishable from the original ones or not. The model trains only the generator, as per any model with an adversarial nature leaves the discriminator untrained. Random batches were also manually implemented for each epoch, which implies that a large number of epochs need to be completed in order to make sure all input data passes through the model. The generator uses a Binary Cross Entropy loss to optimize its weights, while the discriminator uses a Mean Squared Error loss. To evaluate this model the FID was computed between the decoded recreations of the test sample and the actual test sample, as well as between a decoded random sample generated from the model and the training sample.

SUM-PRODUCT NETWORK

In the context of this project the Sum-Product Network should first model the joint probability distribution of our input X, namely the latent space vectors. We then need to sample from our learnt probability distributions, which entails trying to find the Most Probable Explanation for the distribution, in terms of all input variables X_i. Most of the time for this project was spent on trying to implement this specific SPN. Firstly, an implementation was attempted using the libspn library [15], which failed, because it uses Tensorflow1.x in its base code. This meant that in order to use libspn the whole project needed to be converted for an older version of TensorFlow, which when tried produced sub-optimal results for some of the models because of the way they were implemented. Secondly, a very serios attempt was made to implement the desired SPN using the libspn-keras library [16]. Documentation for this library is sparce at best, although there were examples of sampling using a Convolutional SPN, which should imply that it is also possible to sample from a non-convolutional SPN. That ended up not being the case, however. It was possible to learn the joint probabilities of the latent space vector using a loss of NegativeLogJoint and a OnlineExpectationMaximization optimizer, both provided by the library. The problem was the sampling. Because of the way the libspn-keras library is currently set-up it favors Convolutional SPNs which work differently than the standard SPN and sadly the sampling did not work for our trained SPN, even after following the sampling example provided. Lastly, an implementation of the desired SPN was possible using the SPFlow library [14]. The SPFlow library is able to learn a random structure SPN [6] given a specified X vector, in this case the latent space vector. In this case every latent variable from the vector is modeled as a Gaussian. Given the class label Y of each input vector X, an SPN is trained to parametrically learn the joint distribution of the input variables X_i. This is done in order to be able to get samples with specific sets of classes, as well as the ability to create samples for each class separately. After the SPN is trained on each training sample of the dataset, 10 sets of samples with an identical class distribution to the test sample are computed. Each of the sample sets is first decoded through the original AE into images, then the FID between the recreated images from the sample set and the original test sample is computed. The sample set with the best FID score is returned from the network. This is done for two reasons. One, samples from an SPN can vary greatly in quality and this way of evaluating the sample sets reduces that variability. Two, and more importantly, is because of the shortcomings of the SPFlow library. The SPFlow library is slightly outdated, since it was developed for TensorFlow1.x, therefore some of the functionality

is also slightly outdated. One example is that it is impossible to optimize the weights of an SPN using TensorFlow, since the command uses functions of TensorFlow which are deprecated. Then there is the fact that the amount of time it takes to train a random SPN with input of 60000 samples of 100 latent space variables takes around 2 hours in Google Colab, which really stresses the importance of sampling repeatedly in order to produce the best results possible since the alternative entails many wasted hours. It is also to be noted that out of all of the datasets a different implementation of the SPN was implemented for the CelebA dataset, because of the lack of labels. Instead of the SPN learning the joint probability of each class, the SPN for this dataset learns a parametric SPN using just the latent vectors. Sampling is, therefore, completely random.

REALNVP

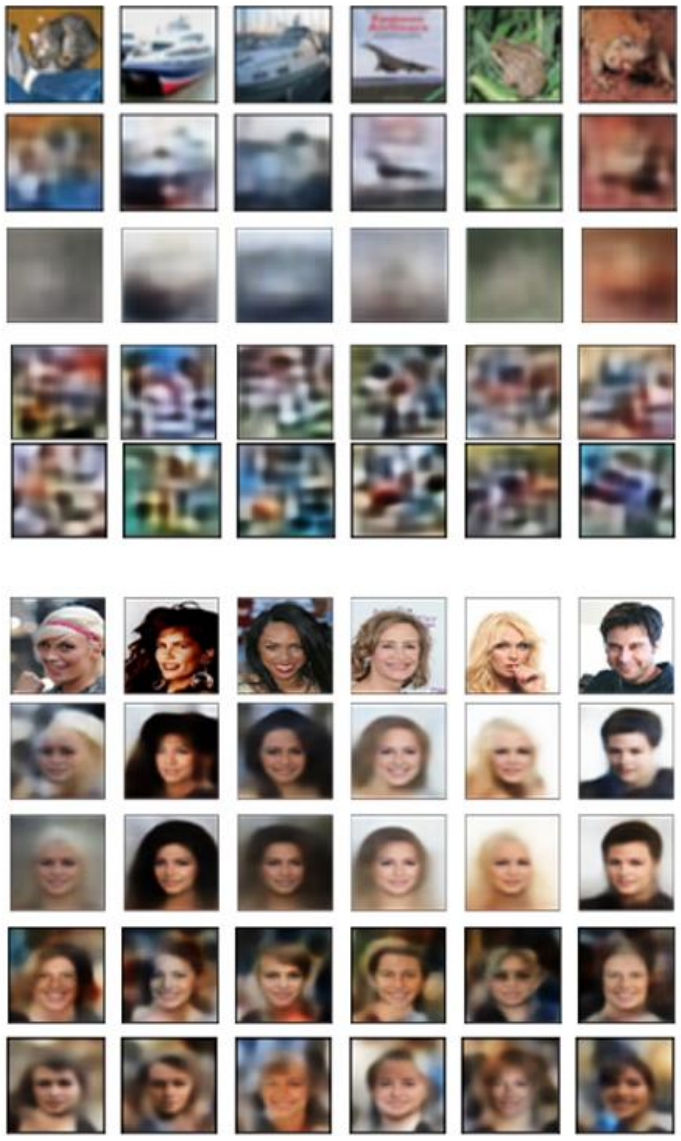
The RealNVP normalizing flow model was the last one to be implemented, therefore the least amount of time was spent on optimizing it. It was implemented from scratch from a keras example [13]. Still it is a far simpler model to implement, although the samples generated from it were random. The FID score will probably be affected by this, due to possible uneven distributions of classes between the test data and the generated samples

IV. RESULTS

After running all of 4 models on all 5 datasets the FID scores were computed, showed in the following table:

	MNIST Digits	Fashion MNIST	Olivetti	CIFAR10	CelebA
Initial AE	**7.1623804783**	**3.1717062403**	**0.90487221363**	1.5559486476	**4.068804989238**
VAE-GAN	10.004797824	5.2206130194	0.97929810575	3.4382171944	5.987437949494
SPN	17.452876828	6.1297250524	0.94721316302	**1.0089970150**	5.158556692801
RealNVP	22.338697662	8.6545136757	0.97229776987	6.1018636519	6.080173283719

Taking only into account the FID scores presented the initial AE is the model which recreates the initial test images more accurately than with the models that augment the latent space. This is to be expected, since the recreated latent space images cannot be better than the images produced by the original latent space. There are two outliers, however, namely the images decoded from the augmented latent space vectors of the SPN for the Olivetti faces dataset and those of the RealNVP for the CIFAR10 model. It is to be noted that the SPN-augmented latent variables generally produce good results according to the FID score and are most ofthen in second place for each dataset. The Normalizing flow also performs well in most cases. When checking to see what the recreations look like, however, the results don't seem as promising. Following are the recreations for all models and datasets, starting from the initial test images on the first row, followed by the recreations from the initial AE, the VAE-GAN, the SPN and the RealNVP in that order. It is to be noted that the SPN and RealNVP produce random samples as opposed to recreations of specific images and therefore should not be compared like strict recreations of the initial image column-wise.



As it is clearly shown by the sample recreations the VAE-GAN is the model that produces the images to the testing set from the augmentation models. This is predictable, given that the VAE-GAN has the task of recreating the latent vectors as close as possible and even though it does learn a latent representation as well it doesn't sample from it. That being said, the most interesting results come from the SPN and the RealNVP normalizing flow, since they produce the most varied and unique results. The SPN, given the labels of the images, is able to draw samples that are in most cases very similar and representative of the specific class. The recreations of the CIFAR10 dataset are the worst overall, which can be attributed to the very complex nature of the dataset itself. The SPN and RealNVP perform very similarly for this dataset. This could be attributed to the large variability of each class in the dataset, as well as the previously discussed complexity which is an obstacle for the initial AE and again, if the initial AE has a low accuracy, the augmentation models will follow suite. Lastly the CelebA recreations are surprisingly good, however the SPN suffers greatly from the lack of class labels and ends up producing similar results to the RealNVP model, which also cannot take advantage of class labels and both models end up generalizing to most-probable samples. For this reason, they have a bigger FID score as overall the samples they produce are closer to the average.

V. Discussion / Future work

Given the results presented in the research, evidence suggests that it is possible to improve the recreations of generative models by implementing augmentation models that take a latent space representation as input in order to produce unique image recreations. By learning the posterior distribution of an encoding wider control is given over the way these encodings are generated and therefore a possibility to learn the prior of the initial generative model better is also a given. And while I wish that the results presented are of higher quality, given the fact that the SPN and RealMVP models are not well optimized, they still show great promise of modeling the prior of the datasets, even in their unoptimized states. Again a huge amount of time was spent on trying to implement a more proper version of the SPN model and I truly believe that if successful an SPN model can perform this task of augmenting a latent space with high fidelity and the ability produce class specific output. Meanwhile, Normal flows have even more potential to be amazing augmentation flows since they can learn the true distribution of the latent space and I truly regret not spending the time I wasted on libspn-keras, researching all types of Normalizing flows that research is being done on at the moment. RealNVM is not the most recent Normalizing flow and I believe that, if constructed correctly, a Normalizing flow model can produce the best results for this research and if more time was given this would become the focus of this research. Lastly the VAE-GAN, while good at recreating input has the least amount of potential for learning the prior of the initial encoding. That being said, a possible continuation would be learning a VAE_GAN to distinguish features of the data in the encoding using class labels. Because of its great reconstruction, if trained properly, it would be possible to recognize which parts of the latent space generate specific features.

References

[1] Diederik P Kingma and Max Welling 2014, "Auto-Encoding Variational Bayes" https://arxiv.org/abs/1312.6114

[2] Hoifung Poon and Pedro Domingos,*"* Sum-Product Networks: A New Deep Architecture*"* University of Washington Seattle, WA 98195, USA

[3] Anders Boesen Lindbo Larsen and Søren Kaae Sønderby and Hugo Larochelle and Ole Winther, Autoencoding beyond pixels using a learned similarity metric, 2016, https://arxiv.org/abs/1512.09300

[4] Alexander Mathiasen and Frederik Hvilshøj, Backpropagating through Fr'echet Inception Distance, 2021

[5] J. Goodfellow and Jean Pouget-Abadie and Mehdi Mirza and Bing Xu and David Warde-Farley and Sherjil Ozair and Aaron Courville and Yoshua Bengio, Generative Adversarial Networks. 2014, https://arxiv.org/abs/1406.2661

[6] Gens, R. & Pedro, D.. (2013). Learning the Structure of Sum-Product Networks. *Proceedings of the 30th International Conference on Machine Learning, in PMLR* 28(3):873-880

[7] Diederik P. Kingma and Jimmy Ba, Adam: A Method for Stochastic Optimization, 2017, https://arxiv.org/abs/1412.6980

[8] Danilo Jimenez Rezende and Shakir Mohamed, Variational Inference with Normalizing Flows, 2016, https://arxiv.org/abs/1505.05770

[9] Laurent Dinh and Jascha Sohl-Dickstein and Samy Bengio, Density estimation using Real NVP, 2017, https://arxiv.org/abs/1605.08803

[10] Jean, Neal, Frechet Inception Distance, 2018

[11] Kobyzev_2020, Normalizing Flows: An Introduction and Review of Current Methods, IEEE Transactions on Pattern Analysis and Machine Intelligence, Kobyzev, Ivan and Prince, Simon and Brubaker, Marcus, 2020

[12] How to evaluate GANs using Frechet Inception Distance, https://wandb.ai/ayush-thakur/gan-evaluation/reports/How-to-Evaluate-GANs-using-Frechet-Inception-Distance-FID---Vmlldzo0MTAxOTI , accessed (06/14/2021)

[13] Density estimation using RealNVP, accessed (06/14/2021), https://keras.io/examples/generative/real_nvp/

[14] SPFlow: An easy and extensible library for Sum-Product Networks, https://github.com/SPFlow/SPFlow

[15] LibSPN, https://github.com/pronobis/libspn

[16] LibSPN Keras, https://github.com/pronobis/libspn-keras

[17] The MNIST database of hanwritten digits, http://yann.lecun.com/exdb/mnist/

[18] The CIFAR10 dataset, https://www.cs.toronto.edu/~kriz/cifar.html

[19] Large-scale CelebFaces Attributes (CelebA) Dataset, http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html