SemiDense Autoregressive Network with VAE

Piotr Piesiak

Master thesis

Supervisor: dr Marek Adamczyk

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

18 June 2024

Abstract

This thesis introduces a novel architecture called the SemiDense Autoregressive Network VAE (SarNET VAE), a generative model that synthesizes images patch by patch in a raster scan order. It is based on the autoregressive SemiDense layer, which uses a masked multi-layer perceptron. SarNET VAE is trained with a convolutional encoder and an autoregressive decoder consisting of SemiDense layers. Latent space vectors (produced by the encoder) enhance the autoregressive image generation process, combining the strengths of both autoregressive and variational models. The experiments demonstrated that SarNET VAE achieves the image generation quality of convolutional VAE. Additionally, SarNET VAE can reconstruct images with different styles conditioned on the latent space. The predictions of this model are easily interpretable and explainable by inspecting the influence of particular input patches on the outcome. SarNET VAE presents an alternative generative model architecture that offers efficient and high-quality solutions for image generation and reconstruction.

Poniższa praca wprowadza nową architekturę modelu generatywnego o nazwie SemiDense Autoregressive Network VAE (SarNET VAE), która generuje obrazy kawałek po kawałku, wierszami od góry do dołu. Bazuje ona na autoregresywnej warstwie SemiDense, która używa perceptrona wielowarstwowego z maskowanymi wagami. SarNET VAE jest trenowany razem z konwolucyjnym enkoderem i autoregresywnym dekoderem, który składa się z warstw SemiDense. Wykorzystanie wektorów z ukrytej przestrzeni (wyprodukowanych przez enkoder) usprawnia autoregresywny proces generowania obrazów poprzez wykorzystanie zalet modeli wariancyjnych i autoregresywnych. Eksperymenty pokazały, że SarNET VAE osiąga jakość generacji obrazów na poziomie konwolucyjnego VAE. Dodatkowo, SarNET VAE potrafi rekonstruować obrazy w różnych stylach warunkowanych wektorami z przestrzeni ukrytej. Predykcje tego modelu są łatwe do zinterpretowania i wyjaśnienia poprzez sprawdzenie, jaki wpływ na wynik miały poszczególne części obrazka z wejścia. SarNET VAE przedstawia alternatywną architekturę generatywnego modelu, która oferuje efektywne oraz wysokiej jakości narzędzie do generowania i rekonstrukcji obrazów.

Contents

Chapter 1

Introduction

In recent years, there has been a steep increase in interest in generative AI. This trend is most likely due to the huge success of the large language model GPT introduced by OpenAI. Generative AI has become accessible to regular people, not only to specialists. Generative models are built to create new data that has similar characteristics to the input training data. In other words, the generated data $\tilde{x} \sim p_\theta(\tilde{x})$ and the training data $x \sim p(x)$ should both come from the same distribution. These models are trained in an unsupervised way in order to learn patterns and distributions in a training dataset.

Generative AI is used in a wide variety of fields, such as text generation, translation, image synthesis, super-resolution, speech and music synthesis, and many more. These technologies can significantly reduce the costs of collecting datasets and speed up this process. Additionally, they are used by humans to boost their productivity and creativity. Therefore, it is worthwhile to delve into and explore the field of generative modeling.

In this thesis, we focus on image synthesis. Currently, the state-of-the-art image-generating models are GANs, Diffusion Models, and VAEs. These models produce diverse images of high quality, but each of them has its own limitations. GANs are very sensitive to hyperparameters, and their training can be unstable if the generator and discriminator are not balanced. VAEs tend to produce blurred images, and Diffusion Models are computationally intensive due to their iterative nature. The majority of research is centered around these models. In this research, we introduce a novel autoregressive architecture for image generation, highlighting the importance of exploring alternative approaches.

We refer to our method as SarNET-VAE (SemiDense Autoregressive Network with Variational AutoEncoder). This model uses a multi-layer perceptron with specially designed weight masks to achieve the autoregressive property. Unlike RNNs, the entire image can be fed into SarNET, resulting in faster training. The image is processed and generated patch by patch. Each patch is a contiguous square section of the image, and the square's dimensions can be adjusted before training. Additionally, SarNET can be fed with latent vectors, leading to better and more diverse image synthesis. The source code for SarNET VAE is publicly available [1].

The primary objectives of this research are:

- Create an autoregressive multi-layer perceptron with conditional generation ability (e.g., latent vector conditioning).
- Use SarNET as the decoder module in a Variational Autoencoder and train the designed VAE from scratch.
- Achieve image generation quality on the MNIST dataset similar to that of the Convolutional Variational Autoencoder.

Chapter 2

Literature Review

There are a lot of kinds of generative models in image-generation tasks. Some of the most well-known and most used are: GANs, VAEs, Autoregressive Models and Diffusion Models [2]. While they generate new data that is very similar to the training dataset, each works in a slightly different way. In this thesis, we focus on Autoregressive Models and Variational Autoencoders, since they are highly related to the SarNET VAE design.

2.1 Variational Autoencoder

Variational Autoencoders (VAEs) were introduced by Kingma and Welling in 2013 [3]. They are based on the autoencoder design, which first compresses the input into a latent space, and then reconstruct the data with minimum possible error. The main idea of the VAE is to use the hidden representation to generate new data, as the hidden space has far fewer dimensions than the input data. However, in Autoencoder there is no control over the latent space. The compressed data can be distributed in many ways. Kingma and Welling introduced the regularization of the hidden space, ensuring that the latent space has properties suitable for generative purposes.

Let's define the distribution of the encoded variable as $q_\phi(z|x)$ and the decoded variable as $p_\theta(x|z)$. In the VAE, we assume that $p(z)$ (the distribution of latent space) follows the standard Gaussian distribution. To ensure this property, the loss function includes a term for the Kullback-Leibler divergence [equation 2.1]. The following loss we aim to maximize.

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - D_{KL}\left(q_\phi(z|x) \| p(z)\right) \tag{2.1}$$

The first term is called the reconstruction loss and, in the case of binary images, can be substituted with Binary Cross Entropy. The second term is the Kullback-Leibler divergence, which ensures that the distribution of the latent space is close to $p(z) \sim N(0, I)$.

VAEs are widely used in the field of generative modeling. One of their main strengths is a well-organized latent space, which allows for interpolation between different data points. Additionally, the VAE loss function [equation 2.1] makes it straightforward to evaluate the quality of different models. The sampling process is also straightforward and fast. However, their biggest drawback is the tendency to generate slightly blurred images. This occurs because VAEs need to simultaneously organize the latent space and reconstruct the image, which can lead to less sharp generations.

2.2 Autoregressive Models

Autoregressive generative models condition their predictions on previous variables. The probability of an image x with n pixels can be expressed as the product of the conditional distributions over pixels:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, ..., x_{i-1}) \tag{2.2}$$

$$ln(p(x)) = \sum_{i=1}^{n} ln(p(x_i | x_1, ..., x_{i-1})) \tag{2.3}$$

Autoregressive models maximize the likelihood [equation 2.2] of the data. They are extremely powerful density estimators, but since the sampling process is iterative, it is resource- and time-consuming. While designing such a model, one needs to decide the order of the data. This is straightforward for text and audio, but in the image-generation task, there are many possibilities for processing pixels (e.g., from top to bottom, patch by patch, diagonally).

2.2.1 PixelCNN

PixelCNN is an autoregressive model developed by Google DeepMind [4]. It uses a stack of convolutional layers with specially designed masks to preserve the autoregressive property. PixelCNN uses two types of masks - *mask A* and *mask B*. Mask A is applied only to the first convolutional layer and allows connections only to previously processed pixels. Mask B is used in the deeper layers and additionally allows connections from the currently processed pixel. These masks are applied in a raster scan order. The advantage of this model over models using RNNs is that it can process all pixels at once, which speeds up training. The sampling process, however, is sequential, which is typical for autoregressive models.

2.2.2 Conditional PixelCNN

Conditional PixelCNN is a model that leverages the autoregressive PixelCNN architecture with additional conditioning using latent vectors. It was introduced by Google DeepMind in 2016 [5]. The authors model the conditional distribution of an image x depending on h as:

$$p(x|h) = \prod_{i=1}^{n} p(x_i|x_1, ..., x_{i-1}, h) \tag{2.4}$$

The h can contain only information about *what* should be in the image (e.g., a one-hot encoded vector describing a class) and not be dependent on the location of the pixels in the image. The authors also developed a variant in which h is the latent representation of an image and is dependent on the location of the pixels in the image.

The second design allows replacing the conventional decoder in the AutoEncoder model with the conditional PixelCNN. This approach results in image generation with very high visual quality. Additionally, the latent space learned with this architecture is organized differently than with the standard decoder (e.g., convolutional decoder).

2.2.3 NADE

Neural Autoregressive Distribution Estimator (NADE) was introduced in 2011 [6]. Similar to PixelCNN, it estimates the distribution $p(x)$ of an input image x. The basic NADE is a single layer feed-forward neural network that is not fully connected. Due to this approach, the model can be fed with a whole image and can be trained using simple backpropagation. The connections are designed to model conditionals as follows [figure 2.1], [equation 2.5]:

$$\begin{aligned} p(x_d = 1|x_{<d}) &= \sigma(V_d\mathbf{h}_d + b_d) \\ \mathbf{h}_d &= \sigma(W_{<d}\mathbf{x}_{<d} + \mathbf{c}) \end{aligned} \tag{2.5}$$

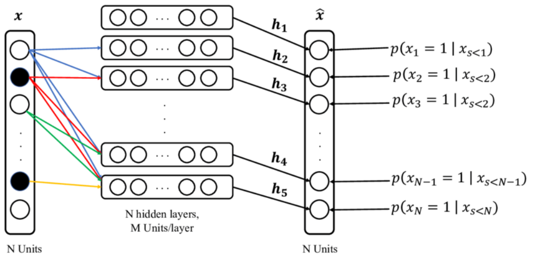


Figure 2.1: NADE architecture [6]. The prediction of a pixel x_d is dependent only on the previous pixels $x_{i<d}$.

Chapter 3

Methodology

3.1 Generative XGBoost

Our journey began with the ambition to develop a simple yet powerful generative model based on XGBoost. Since XGBoost is primarily designed for regression and classification tasks, we needed to adjust it for generative modeling. We decided to mention this approach only as an introduction and a source of inspiration for the SarNET architecture. We do not intend to compare its results to those of SarNET.

To use XGBoost as a generative model, the input data has to be appropriately preprocessed and normalized to the $[0, 1]$ range. We extract contiguous patches (squares) of dimension $2x2$ from the image and permute the pixels so that the pixels from each patch are adjacent to each other in a flattened 784 dimensional vector. Next, we extract the mean values from these patches as additional features.

Initially, the pixels were in order [equation 3.1]. After the permutation, in which we extracted 196 squares of dimension $2x2$, the pixels' order is [equation 3.2]. This permutation preserves the structural information of the image when we extract the mean values from the subsequent patches.

$$p_1, p_2, ..., p_{784} \tag{3.1}$$

$$p_1, p_2, p_{29}, p_{30}, p_4, p_5, p_{31}, p_{32}, ..., p_{756}, p_{757}, p_{783}, p_{784} \tag{3.2}$$

We created 195 f_i XGBoost regression models [algorithm 1] each one predicting patch i, $i = \{2, 3, ..., 196\}$. The i-th model was trained with a flattened vector containing pixels from patches $1, 2, ..., i-1$ and the means of each patch $m_1, m_2, ..., m_{i-1}$. Additionally, we created 195 r_i XGBoost regression models to refine the patches. The r_i model was trained with a slightly blurred $i - th$ patch (extracted from the image blurred with a Gaussian kernel $\sigma = 1$). It predicts the original (non-blurred) pixels of the i-the patch. There are 195 models instead of 196 because we assume the first patch is given and does not need to be predicted.

Algorithm 1 The training process of XGBoost generative models.

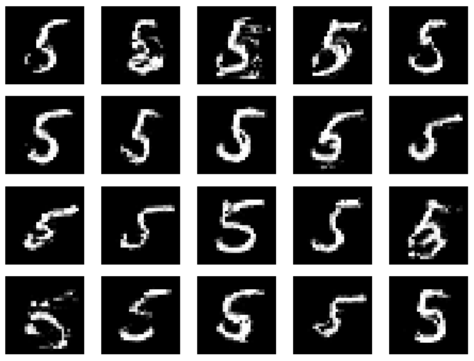
```
# extract patches
patched_data = get_patched_data(orig_data)
blurred_patched_data = get_patch_data(blurred_orig_data)

# train models to refine patch
refine_models = {}
for patch_idx in range(195):
    X = get_specific_patch(blurred_patched_data, patch_idx)
    Y = get_specific_patch(patched_data, patch_idx)

    model = XGBRegressor(n_estimators=40, max_depth=6)
    model.fit(X,Y)
    refine_models[patch_idx] = model

# train models to predict next patch
predict_models = {}
for patch_idx in range(195):
    X = get_previous_patches_and_means(patched_data, patch_idx)
    Y = get_specific_patch(patched_data, patch_idx)

    model = XGBRegressor(n_estimators=50, max_depth=7)
    model.fit(X,Y)
    predict_models[patch_idx] = model
```


(a) Images of the digit 5 generated with XGBoost. The models were trained on the MNIST dataset, which contains only images of the digit 5.

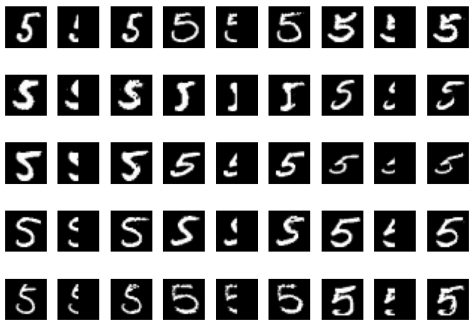

(b) Reconstruction of images of the digit 5 from the test MNIST dataset. Columns 1, 4 and 7 show the original images; columns 2, 5, and 8 show the images to be reconstructed; and columns 3, 6, and 9 show the reconstructed images.

Figure 3.1: Generation and reconstruction of images of the digit 5 using XGBoost. In the reconstruction task, it is evident that the models have learned the representations of different types of the digit 5 and reproduce the original images really well.

We synthesized images with this model by iteratively generating and refining patches with the appropriate models [algorithm 2]. To create more diverse images, the value of the pixel x_i is changed with a probability $p_s = (1 - \frac{patch_{idx}}{196})^{\alpha}$ and if the $i \leq i_{max}$ in the following way. When the model outputs the intensity $\beta \in [0, 1]$ of a pixel x_i, the value of the pixel x_i is set to 0 with a probability $p = 1 - \beta$ and to β with a probability $p = \beta$. This randomization occurs only in the first stage of the generation process ($i \leq i_{max}$) and only with a small probability p_s to achieve higher quality images. The reconstruction process is not randomized and is performed in a similar way, starting with the vector containing part of the picture to reconstruct.

Algorithm 2 Generation with 195 XGBregressors.

```
generated_image = get_initial_patch()
for patch_idx in range(195):
    X = get_previous_patches_and_means(generated_image, patch_idx)
    new_patch = predict_models[patch_idx].predict(X)
    new_patch = clip_values(new_patch)

    if patch_idx > max_p_idx or \
    rand(0,1) < (1 - patch_idx / 196) ** alpha:
        choices=1
    else:
        choices = [random.choice(2, p=[1-pix, pix])
                   for pix in new_patch]
    new_patch *= choices

    new_patch = refine_models[patch_idx].predict(new_patch)
    new_patch = clip_values(new_patch)

    generated_image += new_patch
```

3.2 SemiDense Layer architecture

3.2.1 SemiDense Layer

Inspired by the generative XGBoost, we aimed to replace each small XGBoost model with a neural network to take advantage of custom features that the network would learn. This led to the creation of the SemiDense layer, which combines each of these small neural networks into one large autoregressive layer. The SemiDense layer is a linear layer with masked weights, allowing it to achieve the autoregressive property. It takes a flattened image as an input and process it in parallel, pixel by pixel or patch by patch (depending on the configuration).

The patches are subsequent squares of the image and do not overlap. Additionally, connections from all the output neurons to the input latent vector can be added. The layer can be configured in two ways:

(a) The 0-th output patch depends on the 0-th input patch, and the i-th output patch (for $i > 0$) depends on the $0, ..., (i-1)$ input patches (used in the first layer).

(b) The i-th output patch depends on the $0, ..., i$ input patches (used in deeper layers).

The images [figure 3.2], [figure 3.3] illustrate the first (a) configuration with patches of size $4x4$ (49 patches). This design allows us to feed the entire flattened image into a network and obtain an output image of the same size. There is no need to "shift" the sequence to the right as in transformer models.

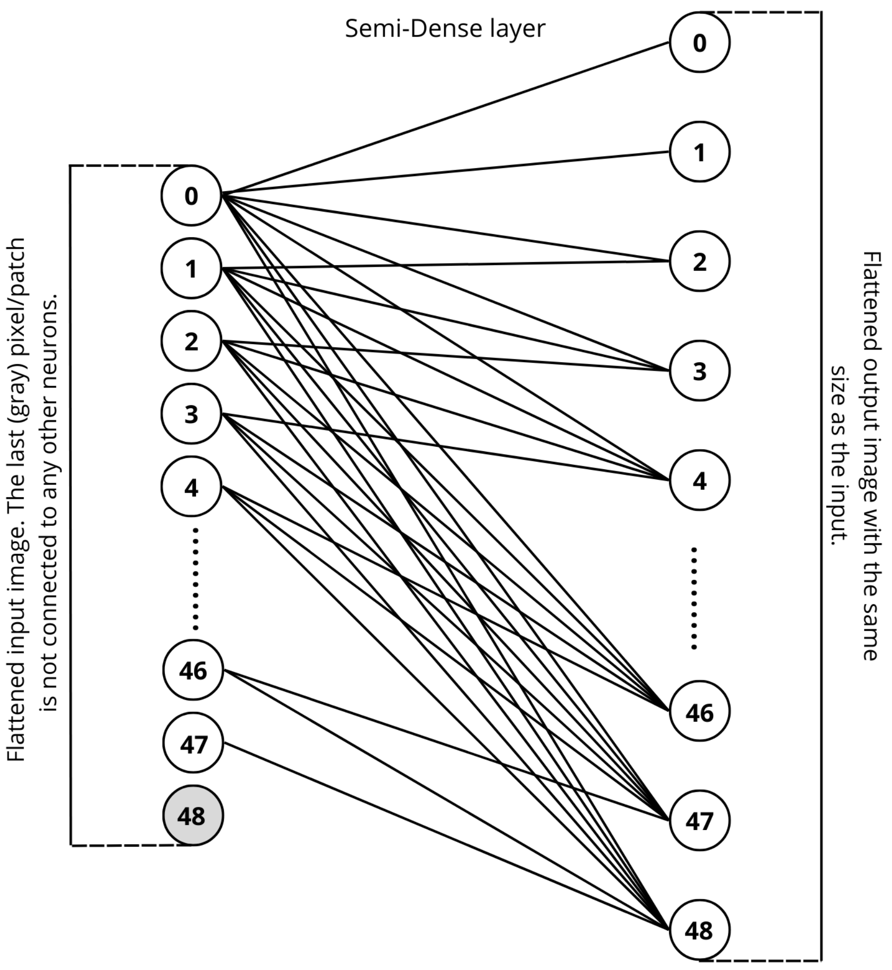

Semi-Dense layer
Flattened input image. The last (gray) pixel/patch is not connected to any other neurons.
Flattened output image with the same size as the input.
0
1
2
3
4
46
47
48
0
1
2
3
4
46
47
48

Figure 3.2: SemiDense architecture of type a with no latent input. The last input patch is not connected to any neurons, as the network learns to predict it. The first output patch is connected only to the first input patch and is expected to be identical. This design ensures that the input and output sizes are the same.

To preserve the autoregressive nature of the layer, the number of output pixels/patches should be the same as the number of input pixels/patches. The size of the input patches can vary from the size of the output patches. For example, we can compress the image from patches of size 4x4 to patches of size 2x2 or 1x1. Conversely, we can expand the patch size to, for example, 8x8. Additionally, we can add channels to the input and output image; the architecture remains the same, but instead of looking at/predicting the patch in a single channel, we look at/ predict the patch in every channel.

3.2.2 SemiDense Layer with latent input

The SemiDense architecture can be extended to use the latent vector as a conditional input while maintaining the same overall structure. The latent vector is integrated into the SemiDense layer by adding connections from all output neurons to the latent vector. This modification allows each output to be influenced not only by previous patches but also by the global context. The global context can include a one-hot vector indicating the class or a hidden representation capturing the style of the image [figure 3.3].

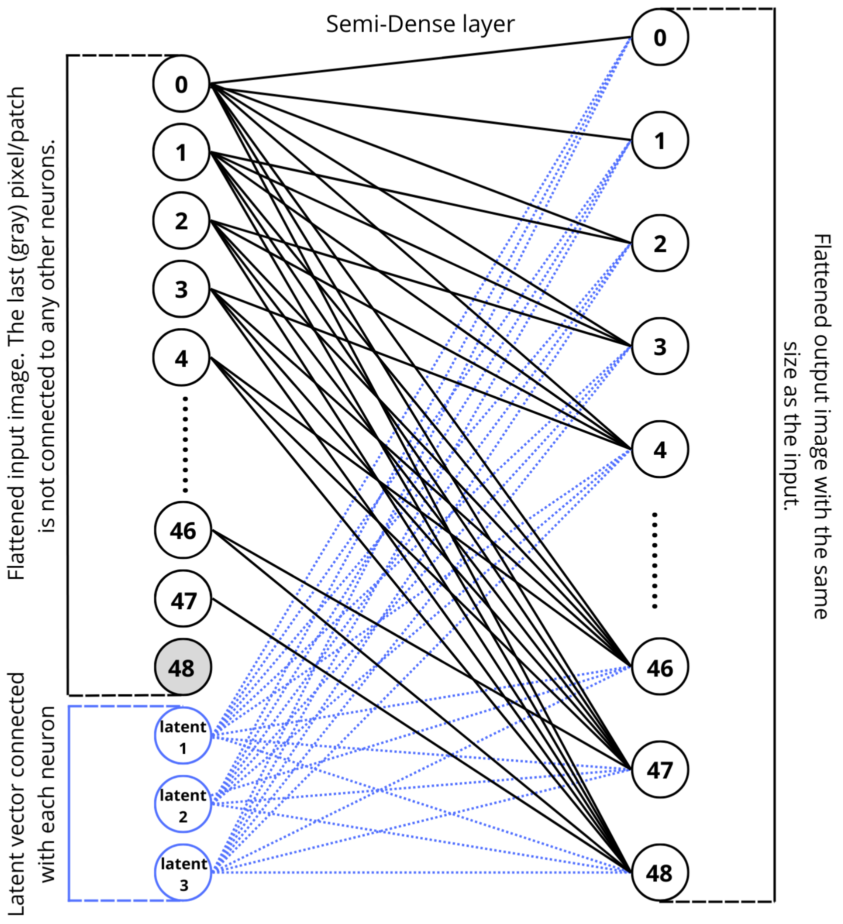
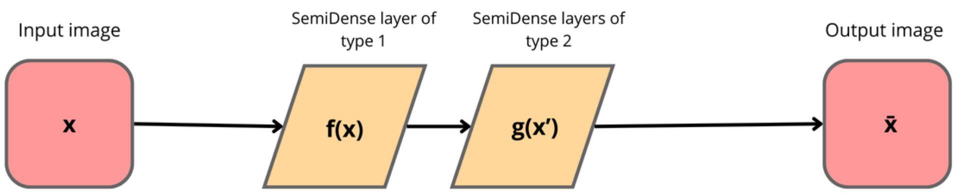

Semi-Dense layer
Flattened input image. The last (gray) pixel/patch is not connected to any other neurons.
Latent vector connected with each neuron
0
1
2
3
4
46
47
48
latent 1
latent 2
latent 3
Flattened output image with the same size as the input.

Figure 3.3: SemiDense with latent input, enabling conditional generation.

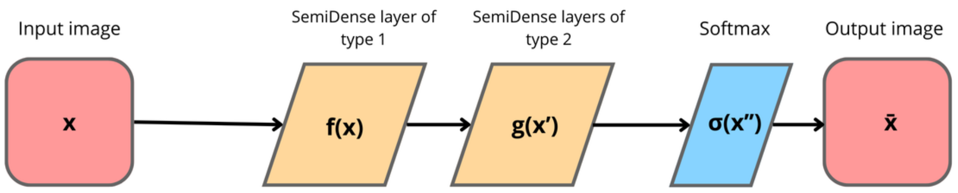
3.3 SarNET

SemiDense Autoregressive Network (SarNET) consists of a stack of SemiDense layers without latent inputs. Within the stack, the first layer is of type a. Subsequent layers are of type b. In our experiments, we decided to use three SemiDense layers. The first one compresses the single-channel input patches of size $4x4$ into two channels with patches of size $2x2$ and can be interpreted as a feature extractor. The second one expands the patches to size $4x4$ in two channels, and the last one maps the patches back to a single-channel.

SarNET can be used as a regression model [figure 3.4a] and trained to return the continuous values of pixels. It can also be used as a classification model [figure 3.4b], predicting the probabilities of a given class for each pixel. In the latter case, the Softmax activation function needs to be added at the end of the network.



(a) SarNET for regression.



(b) SarNET for classification.

Figure 3.4

The generation process with a classification model is straightforward. We generate patch by patch and sample the classes of pixels in each patch with the probabilities returned by the model. Additionally, we divide the returned probabilities by the parameter t (temperature). When $t < 1$, the model will be more certain about its predictions; if $t > 1$, the model will be less certain.

Plain generation patch by patch with the SarNET regression model leads to a single deterministic output. To obtain diverse images, we add a random vector V_R from the $N(0, 1)$ distribution to the image vector. The random vector has a size of $1x784$. We can parameterize this vector to control the degree of randomness [equation 3.3]. At the beginning of generation, we select the value of alpha based on the α_{max} parameter [equation 3.4].

$$noise = V_R \cdot \alpha,\ V_R \sim N(0, \alpha^2) \tag{3.3}$$

$$\alpha = \alpha_{max} * U[0, 1] \tag{3.4}$$

Adding even small noise to all the patches results in poor generation. To enhance this process, in each generation step, we randomly select previous patches (with probability p) to which we add the previously generated noise instead of taking all of them. We add noise only to the input patches because we do not want noise in the output image. We keep the generated patches in a separate vector.

In the end, we can refine the image by feeding it once again into the network. The output image usually is sharper and has less noise. To stabilize generation process, we black out the pixels below a certain threshold - typically, the pixels below a small constant (≈ 0.09) are set to black. Usually,s we set $p = 0.3$ and $\alpha_{max} = 0.65$.

3.4 SarNET VAE

SemiDense Autoregressive Network with Variational Autoencoder (SarNET VAE) is an extension of SarNET. It leverages the advantages of a Variational AutoEncoder to enhance the generation process. First, it organizes the latent space using a Convolutional Encoder. Then, it feeds the input image into a stack of SemiDense layers. Each of the SemiDense layers has additional connections to the latent vector produced by the Convolutional Encoder [figure 3.5]. The loss function consists of the KLD loss and the reconstruction loss, which is either Binary Cross Entropy or Mean Squared Error [equation 2.1].

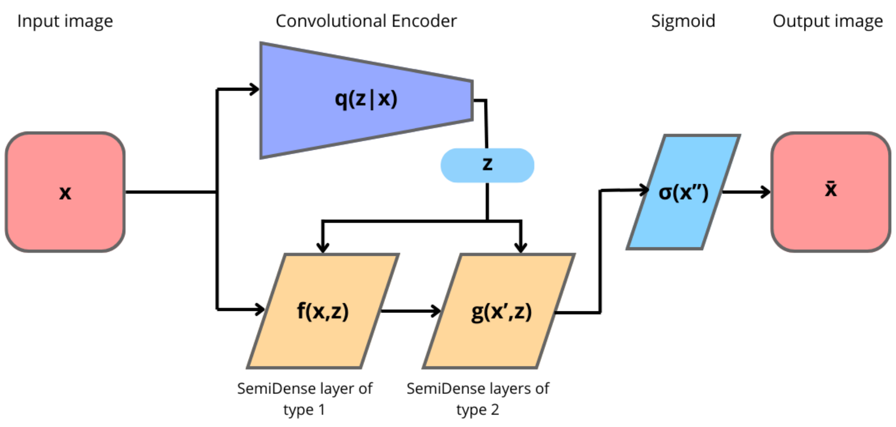


Figure 3.5: SarNET VAE architecture. We replace the conventional decoder (e.g., a convolutional decoder) with SarNET. This architecture is trained from scratch and does not require pretrained weights.

This architecture can conditionally generate an image from scratch (starting with a black patch) or reconstruct a part of an existing image. First, it generates a random latent space vector that defines the style of the image. Next, it synthesizes the image patch by patch using conditional information in the hidden space. The random vector can be modified to have less variance, resulting in output digits that are "sharper" but less diverse.

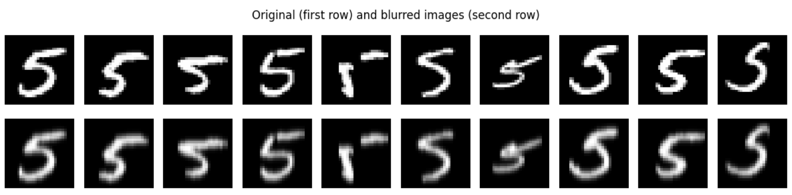
3.5 Evaluation - Frechet Inception Distance

Frechet Inception Distance (FID) is the metric used to evaluate the quality of images generated by generative models. It measures the similarity between the distributions of real and generated images. After passing images through a classification model, the statistics (means μ_{real}, μ_{gen} and covariances Σ_{real}, Σ_{gen}) of the features in the final feature layer are extracted. Then, the distributions of these features are modeled as the multivariate Gaussian distributions. The FID score is calculated as the Frechet Distance between these distributions [equation 3.5].

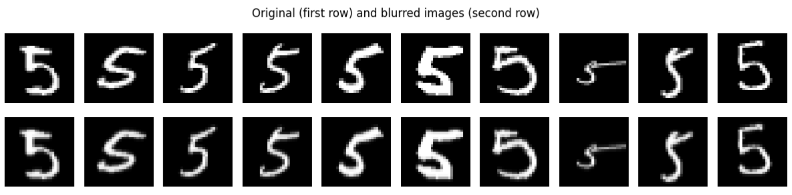
$$FID = \|\mu_{real} - \mu_{gen}\|^2 + \mathrm{Tr}\left(\Sigma_{real} + \Sigma_{gen} - 2\sqrt{\Sigma_{real}\Sigma_{gen}}\right) \tag{3.5}$$

The model used as a feature extractor is usually the Inception v3 model pre-trained on the ImageNet dataset. In our experiments, there was no need for such a sophisticated model, so we used a MNIST classifier developed by TensorFlow with 99% evaluation accuracy. This model allows for quicker calculations and is specifically suited for our tests. It was designed for the TF-GAN library, which provides functions for calculating FID. TF-GAN is widely used in projects and research at Google [7].

FID measures both the quality and diversity of generated images. The lowest possible value and the best, is 0, which occurs only when the two compared datasets are identical. The FID score is very sensitive. For example, after blurring images of the digit 5 from the MNIST dataset with a Gaussian filter, which applies a Gaussian kernel to smooth the images, the FID score was high - 37.053 for $\sigma = 1$ and 2.218 for $\sigma = 0.5$ [figure 3.6]. Other transformations, such as padding or random perspective transformation, yielded even higher scores.



(a) Images blurred with $\sigma = 1$



(b) Images blurred with $\sigma = 0.5$

Figure 3.6: Illustration of blurring images of the digit 5 with different values of σ. The FID is very sensitive; even for marginally blurred images with $\sigma = 0.5$, the score was 2.218.

Chapter 4

Experiments and Results

4.1 SarNET

We conducted several experiments to compare different SarNET architectures and different loss functions used during training. We found that the architecture introduced in the previous chapter (3 SemiDense layers, patch of size $4x4$ [figure 3.3]) gave the best results in our tests.

In the regression, we compared different loss functions of the form [equation 4.1]. Since the input data were standardized to the range $[0, 1]$, the larger the power α in the loss function, the more blurred the generated images became. When using only [equation 4.1] as the loss function, setting $\alpha = 1$ (L1 loss) yielded the best quality of generated images [figure 4.1].

$$L(x, y) = \frac{1}{n} \cdot \sum_{1}^{n} |x - y|^a \tag{4.1}$$

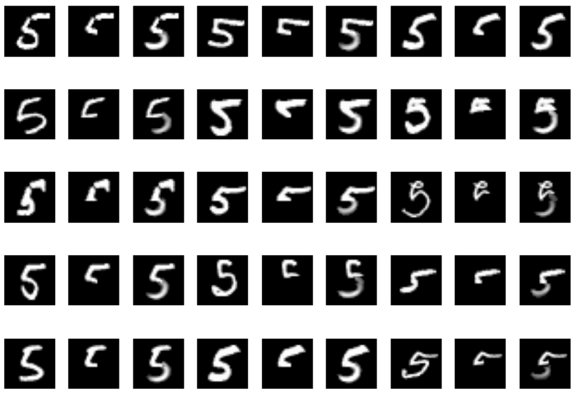


Figure 4.1: Reconstruction of images using SarNET with the L1 loss. Columns 1, 4, and 7 show the original images; columns 2, 5, and 8 show the images to be reconstructed; and columns 3, 6, and 9 show the reconstructed images.

To improve the model, we modified the loss function to include the Structural Similarity Index Measure (SSIM) [equation 4.2]. SSIM measures the similarity between two images based on structural information. It calculates three features from an image: brightness of an image (luminance), difference in luminance values across the image (contrast), and the spatial arrangement of the pixels (structure).

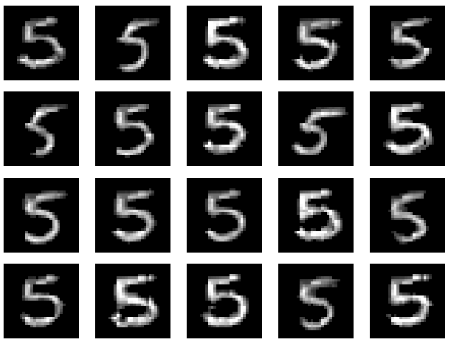
$$L(x,y) = \frac{1}{n} \cdot \sum_{1}^{n} |x-y|^a + (1 - SSIM(x,y)) \tag{4.2}$$

The SSIM formula is given by [equation 4.3], where σ_x^2, σ_y^2 are the variances of x and y. μ_x, μ_y are the means of pixel values in x, y and σ_{xy} is the covariance of x and y. l, c, s are the functions to compare luminance, contrast and structure, respectively. C_1, C_2, C_3 are constants to avoid division by zero. The value of this measure is between -1 and 1. The higher the score, the more similar the images are.

$$\text{SSIM}(x,y) = l(x,y) \cdot c(x,y) \cdot s(x,y) \tag{4.3}$$

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}, \; c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, \; s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}$$

Incorporating $SSIM$ into the loss function improves the speed of training and the quality of the model. Using $SSIM$ with Mean Squared Error improved MSE from 0.0354 to 0.0256, and when used with $L1$ $loss$, it improved $L1$ from 0.0902 to 0.0647. The quality of generation and reconstruction was similar for both losses. Even though the model learned the representations of the digit 5 [figure 4.2b] it was challenging to generate diverse images [figure 4.2a].


(a) Generation of digit 5.

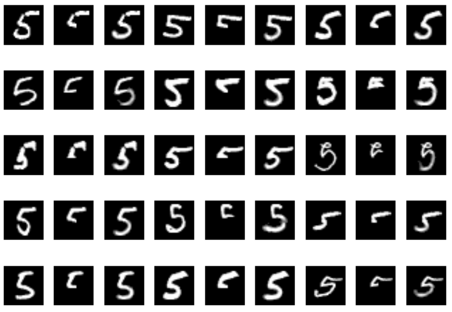
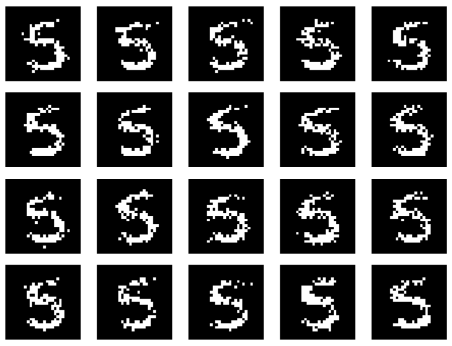

(b) Columns 1, 4, and 7 show the original images; columns 2, 5, and 8 show the images to be reconstructed; and columns 3, 6, and 9 show the reconstructed images.

Figure 4.2: Generation and reconstruction of the digit 5 with SarNET using $4x4$ patches, trained with MSE and $SSIM$ $loss$. The quality of the reconstruction shows that the model has learned the representations of the digit 5 very well.

We also experimented with different dimensions of the patches. We found that with small patches of size $2x2$, the MSE or $L1$ loss can be reduced even further during training. Unfortunately, despite having a smaller loss, the quality of the generated images was worse than when SarNET was trained with larger patch sizes.

SarNET with classification does not perform as well as the regression model. However, it is easier to generate more diverse images with this method because we can naturally sample from the returned probabilities. We found that the model gives the best results when trained with patches of size $4x4$ [figure 4.3a]. The model struggles to generate high-quality images as the number of classes increases.

The *SSIM loss* improved the regression models, so we considered incorporating it in classification. We created a model that takes pixel intensity in the range $[0, 1]$ as an input and interprets it as the probability of class 1. The target images in the training process are binary, but continuous in the range $[0, 1]$. This approach allows us to add *SSIM loss* to the Cross Entropy Loss. The model created this way generates far better results than other classification models [figure 4.3b]. Unfortunately, it has difficulties producing diverse images. While it can reconstruct images, the reconstructed parts are visibly blurred. We conclude that it is better to use the SarNET regression models for digit generation.


(a) Generation with model that uses binary target data.

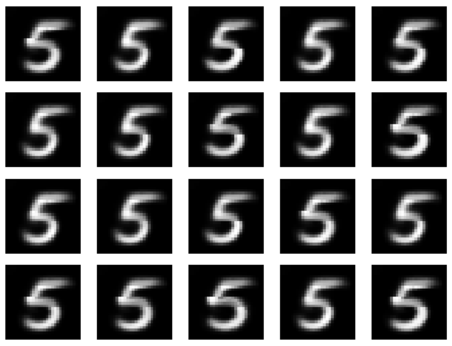

(b) Generation using a model that employs continuous target data and incorporates SSIM in its loss function.

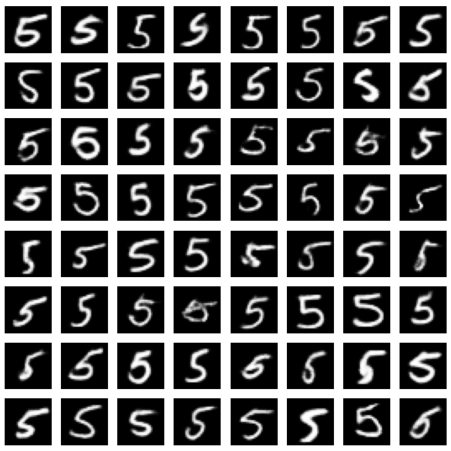
Figure 4.3: Generated images of the digit 5 using classification SarNET with $4x4$ patches.

4.2 Convolutional VAE

We used the implementation of a Convolutional VAE from the TensorFlow website as our baseline model. It uses convolutional layers for both the encoder and decoder networks. We trained this model with different sizes of latent vectors: $8, 16, 32, 64, 128$. The larger the latent space, the better the quality of images generated. *FID* scores of these experiments are presented in the Frechet Inception Distance scores subsection [subsection 4.4].

4.3 SarNET VAE

The inclusion of the latent space in the SarNET architecture was very helpful. It increased the quality of generated images [figure 4.4b] and, above all, simplified the generation process. SarNET VAE generates patch by patch based on the style given by the random latent vector. FID scores of these experiments are presented in the Frechet Inception Distance scores subsection [subsection 4.4].


(a) Generation with VAE.

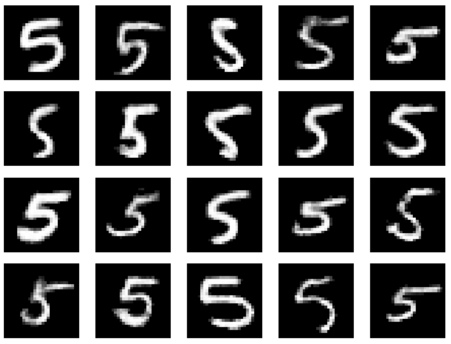

(b) Generation with SarNET VAE.

Figure 4.4: Comparison of the generation of the digit 5 using VAE and SarNET VAE with a latent space dimension of 128.

Additionally, thanks to the autoregressive nature of SarNET, it can reconstruct the image with a different style. We provided SarNET with half of the original image as input and generated the other half with the model. This resulted in different shapes for the bottom part of a digit 5 [figure 4.5].

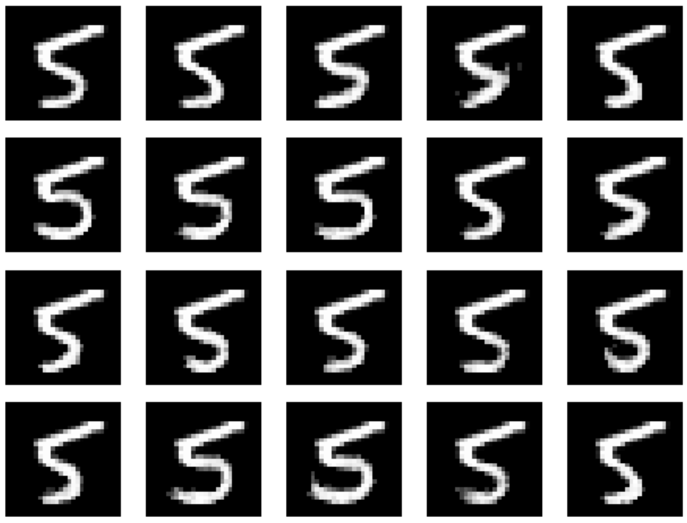


Figure 4.5: Different styles of the bottom half of the given image.

4.4 Frechet Inception Distance scores

We evaluated different configurations of our models using Frechet Inception Distance metric. To achieve reliable results, we generated 10 sets of 10 000 images of the digit 5 (50 sets for models with 128-dimensional latent space) and compared them with 10 000 real images of the digit 5 from the test dataset. We compared the averaged FID scores in [table 4.1].

Model	FID **Score**
SarNET VAE 128	**1.535**
SarNET VAE 64	1.595
SarNET VAE 32	1.580
SarNET VAE 16	1.614
SarNET VAE 8	2.125
VAE 128	2.018
VAE 64	**1.565**
VAE 32	1.611
VAE 16	1.716
VAE 8	2.876
simple SarNET	29

Table 4.1: Comparison of FID scores for SarNET, SarNET VAE and VAE.

The model with the best FID score of 1.535 was the SarNET VAE with a 128-dimensional latent space. The larger the dimensionality of the latent space, the better the results SarNET VAE obtained. This was not the case with the VAE model, where a too-large latent space led to overfitting and less diverse images (although an FID score of 2.018 is still a great score). Except for the latent dimension of 64, SarNET VAE obtained lower FID scores in all other cases. The simple SarNET without conditional generation had FID score of 29.

Chapter 5

Discussion

SarNET is a very powerful network capable of learning the representations of MNIST digits. When combined with a convolutional encoder, it achieves very high-quality generation. As discussed in the FID evaluation section [section 3.5], the FID score is very sensitive. Even for slightly blurred data, the metric was 2.218. Generated datasets with FID scores below 2 have images that are only marginally different from the real datasets. In 50 experiments, each generating 10 000 images, SarNET VAE achieved a mean FID score of 1.535, which was better than the VAE model.

The SarNET VAE model introduced in this thesis leverages the advantages of autoregressive models and variational autoencoders. The SarNET without latent space representations has difficulties in generating high-quality images, although it can reconstruct images decently. The SemiDense layer enables the creation of deep architectures and experimentation with different patch sizes and datasets. Our architecture provides flexibility and allows for the use of autoregressive modeling with latent space representations.

The experimental results demonstrate that SarNET VAE achieves the same or slightly better quality of generation than convolutional VAE. Additionally, thanks to its autoregressive nature, SarNET VAE can reconstruct the missing part of an image with different styles conditioned on the latent space. It is able to generate even more complicated datasets, as shown in [figure 5.2b]. The SarNET architecture makes it possible to calculate the importance of patches used in making a prediction [figure 5.1]. This allows us to detect patches that impacted the output and gain insights into how the model works.

5.1 Explainable SarNET

In the first SemiDense layer of our models, we calculate the activations of neurons in specific patches to determine the importance of each patch at the input. We achieve this by taking the product of the pixels in a patch and the weights connected to that patch. Then, we sum the absolute values and interpret the outcome as the activation of the given patch on the input. By iterating over each patch, we can rank them based on their importance. The architecture of the SemiDense layer allows the model to focus more on certain parts of images to make accurate predictions.

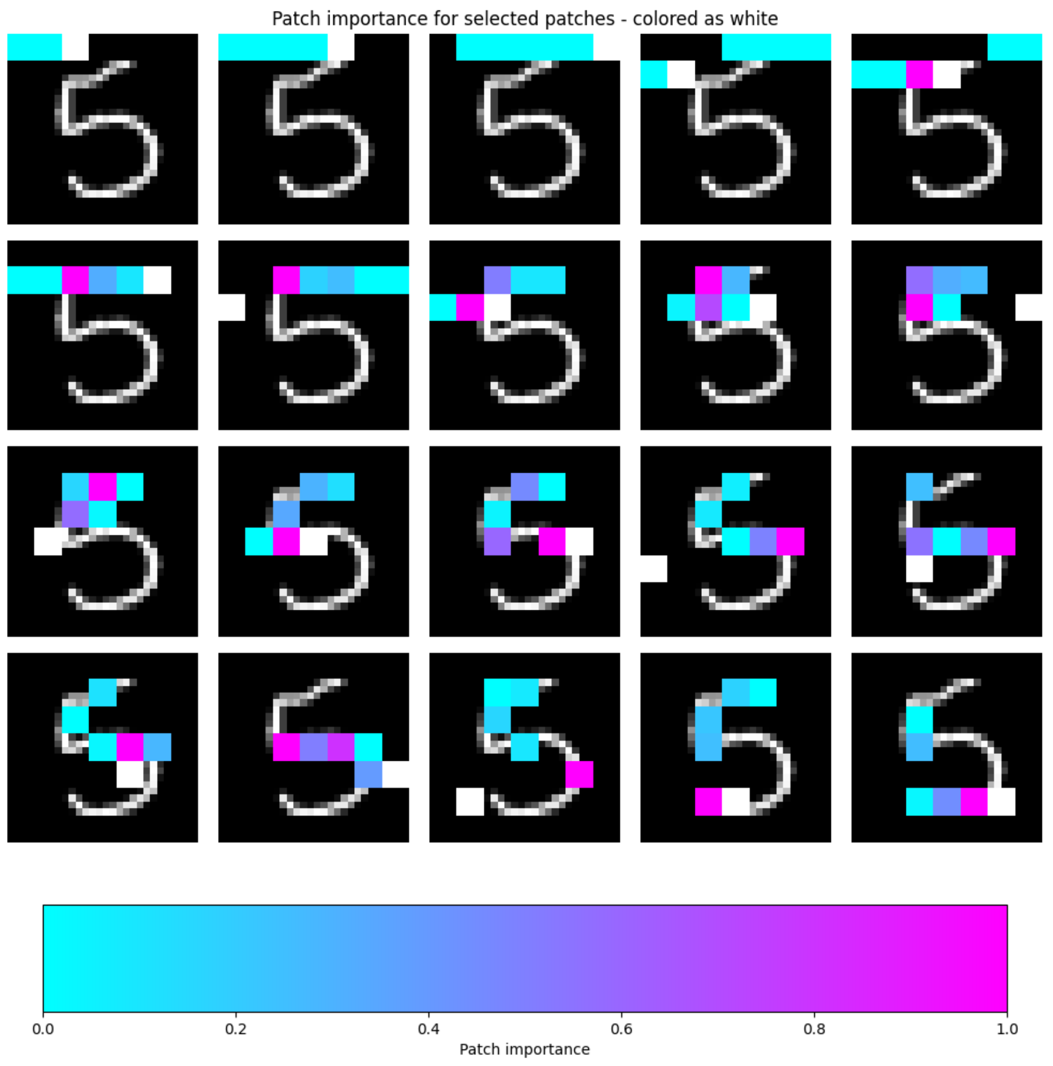
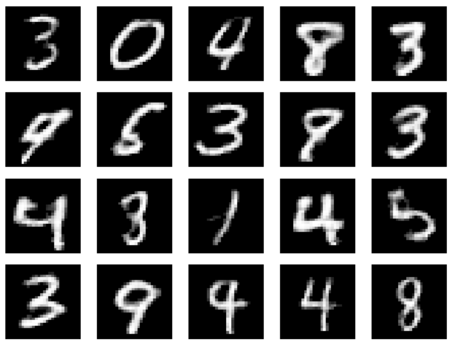


Figure 5.1: The importance of patches at the input. Each image shows the five most important patches that the model considers when predicting the value of the white patch. The importance of each patch was measured on a scale of $[0, 1]$. Patches were colored according to their importance.

5.2 SarNET VAE with different datasets

In this thesis, we focused specifically on generating one digit from the MNIST dataset and comparing it with the quality of the VAE model. In [figure [5.2]], we present the results of generation with the SarNET VAE model trained on the entire MNIST dataset (a) and on the FashionMNIST dataset (b). FFashionMNIST is a dataset of 10 types of Zalando's article images, consisting of a training set of 60,000 examples. The images are grayscale and have a size of $28x28$ pixels. SarNET produces realistic images for both datasets.


(a) MNIST generation.

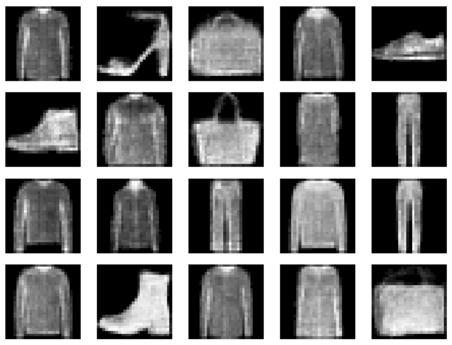
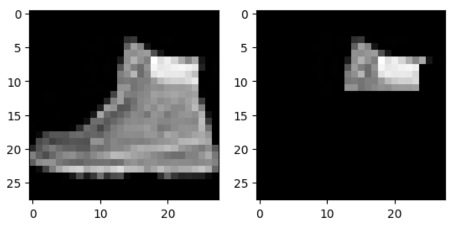

(b) FashionMNIST generation.

Figure 5.2: Generation with SarNET VAE trained on (a) the MNIST dataset and (b) the FashionMNIST dataset.

SarNET VAE perfectly reconstructs the images from the FashionMNIST dataset. It has the ability to condition its generation and creatively complete the images [figure 5.3].



(a) The original image (left) and the part of the image to be reconstructed (right).

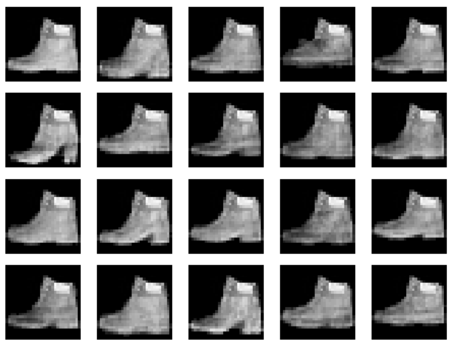

(b) Reconstruction of the boot in different styles.

Figure 5.3: Reconstruction of the boot based on the partially blacked-out image (a). SarNET VAE generated different heels, soles, and shoelaces.

In [figure 5.4], we present the importance of patches at the input for the image of a boot from the FashionMNIST dataset. Depending on the part of the shoe, SarNET focuses on different areas of the image.

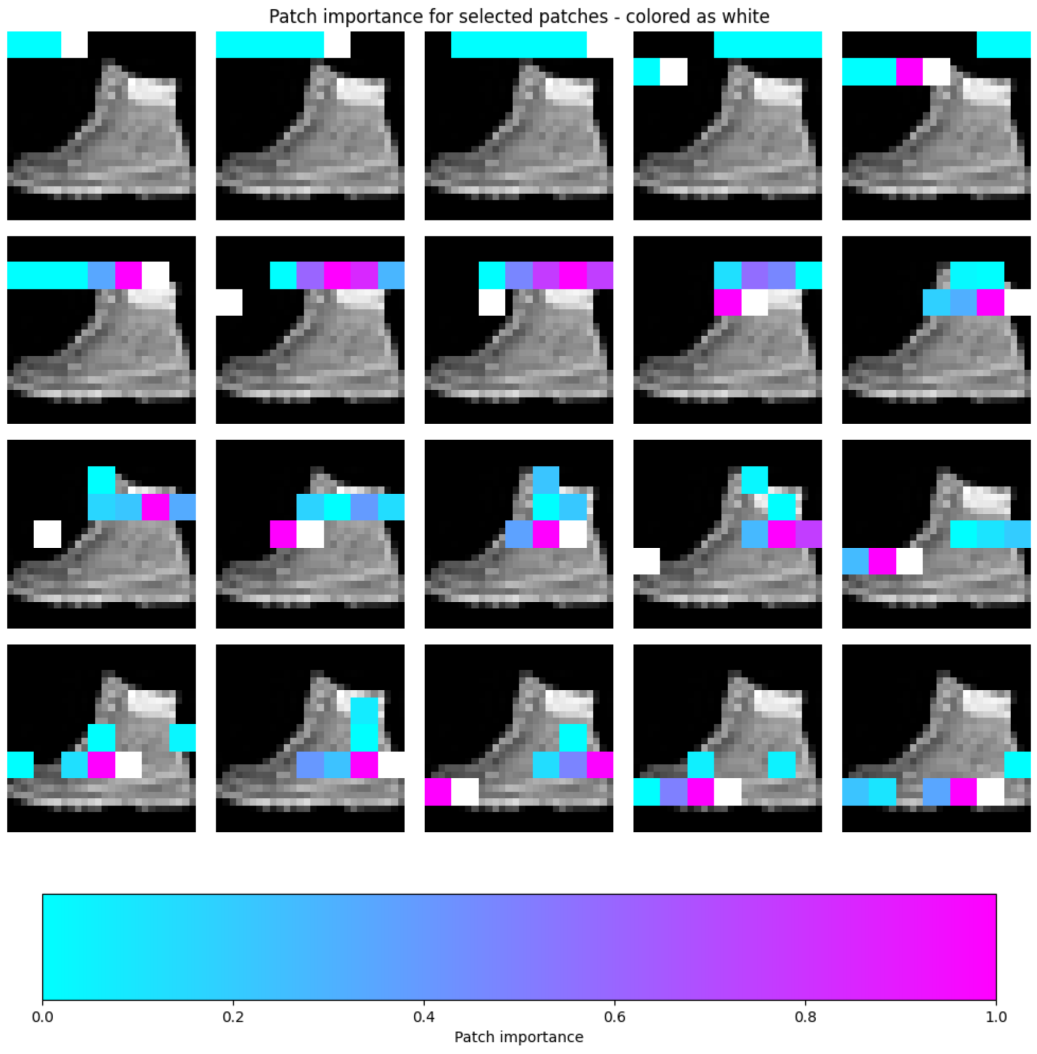
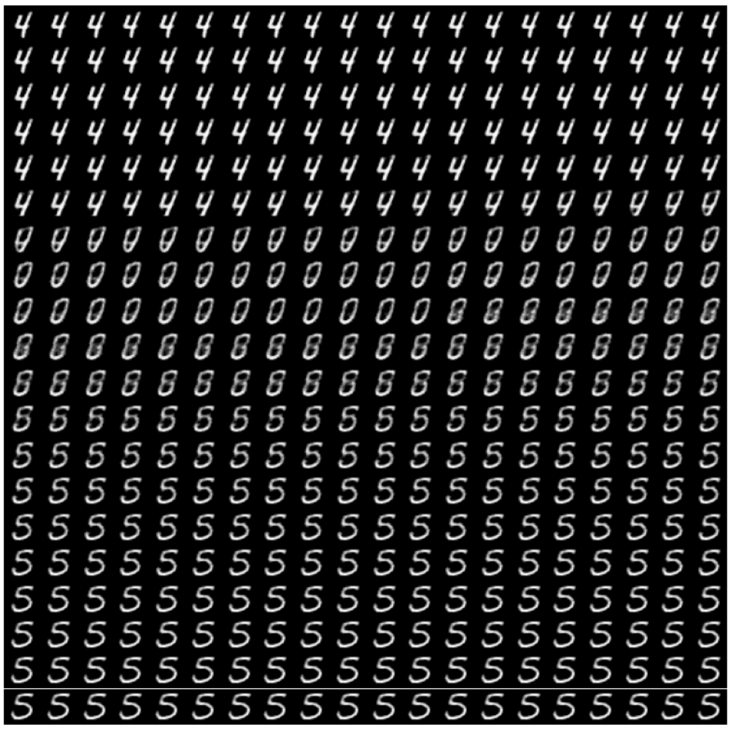


Figure 5.4: The importance of patches at the input. Each image shows the five most important patches that the model considers when predicting the value of the white patch. The importance of each patch was measured on a scale of $[0, 1]$. Patches were colored according to their importance.

In [figure 5.5], we present the interpolation over the latent space between the latent vectors of digits 4, 5, and 3, 4. Since sarNET is autoregressive, the initial pixels generated have the most significant impact on the model's output. The model generates images in a raster scan order (horizontally left-to-right), making digits with more apparent features at the top easier to generate.


(a) Interpolation over the latent space between digits 4 and 5.

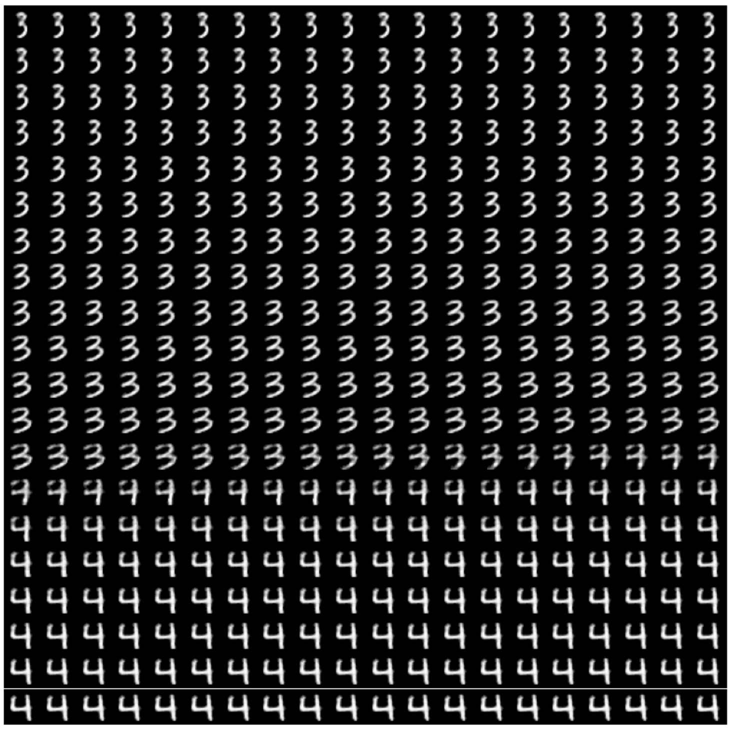

(b) Interpolation over the latent space between digits 3 and 4.

Figure 5.5: Interpolation between two randomly selected digits from the MNIST dataset.

Chapter 6

Conclusion

This thesis has introduced a novel architecture called the SemiDense Autoregressive Network (SarNET). It is based on autoregressive SemiDense layers, which use masked multi-layer perceptrons. This flexible and efficient framework can be used with an additional latent space input to condition generation. It can also be used and trained as the decoder module in a Variational Autoencoder.

Although in our thesis we focused mostly on images of the digit 5 from the MNIST dataset, our model can be trained on the entire MNIST dataset and achieve high-quality generation [figure 5.2a]. We also experimented with the more complicated FashionMNIST dataset and obtained good results [figure 5.2b].

All presented models have the capacity to learn the representation of the digit 5. However, it is difficult to generate high-quality, diverse images without a latent space input. We believe that the generation process in regression SarNET can be further explored. Future research could investigate the application of SarNET VAE to even more complicated datasets. Another area for future exploration is combining masked convolutions in the SemiDense layer. This approach could improve the ability to capture important image features.

The experimental results proved the effectiveness of the SarNET VAE framework, which achieved the quality of generation comparable to the convolutional VAE. Our research indicates that SarNET can be successfully used as a generative model. This model can also be used as a reconstruction tool that creatively completes parts of an image. This functionality is a significant extension over the convolutional VAE, which lacks this capability. Additionally, the outputs of SarNET can be easily explained by analyzing the importance of input patches used in making predictions. We believe that the insights gained from this work can help in further advancements in the field of image generation and contribute to the development of more efficient generative models.

Bibliography

[1] Piotr Piesiak (happypio). *SarNET: Source Code for SemiDense Autoregressive Network with Variational Autoencoder*. GitHub repository, `https://github.com/happypio/sarNET`, 2024.

[2] Sam Bond-Taylor, Adam Leach, Yang Long, Chris G. Willcocks *Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models*, arXiv preprint arXiv:2103.04922, 2021.

[3] Diederik P. Kingma and Max Welling, *Auto-Encoding Variational Bayes*, arXiv preprint arXiv:1312.6114, 2013.

[4] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu, *Pixel Recurrent Neural Networks*, arXiv preprint arXiv:1601.06759, 2016.

[5] Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu, *Conditional Image Generation with PixelCNN Decoders*, arXiv preprint arXiv:1606.05328, 2016.

[6] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, Hugo Larochelle *Neural Autoregressive Distribution Estimation*, arXiv preprint arXiv:1605.02226, 2016.

[7] Google, *TF-GAN: A Lightweight Library for Generative Adversarial Networks*, 2018. [Online]. Available: `https://github.com/tensorflow/gan`. [Accessed: 18-Jun-2024].

[8] VainF, *PyTorch MS-SSIM: Multi-Scale Structural Similarity for PyTorch*, 2019. [Online]. Available: `https://github.com/VainF/pytorch-msssim`. [Accessed: 18-Jun-2024].