Generating surface topography of paperboard using diffusion models

Ale Loman

Master's thesis
2025:E5

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematical Statistics

Centrum Scientiarum Mathematicarum

Generating surface topography of paperboard using diffusion models

Ale Loman - ale.loman@hotmail.com

January 2025

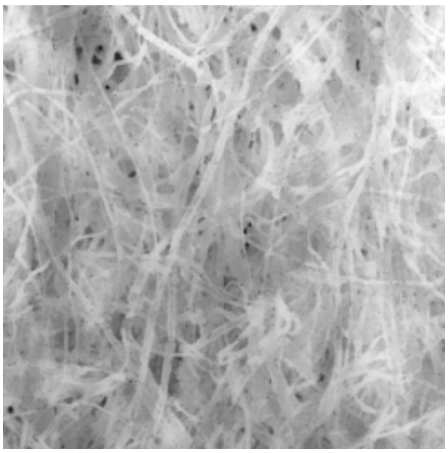
Academic supervisor: Linda Hartman
Tetra Pak supervisors: Magnus Arnér, Erik Bergvall, Torbjörn Hedberg

Abstract

This thesis uses the unsupervised deep learning method of generative diffusion models to generate images of paperboard surfaces. As the images are intended to be used as input to various simulations by food packaging company Tetra Pak, simply looking at them is not sufficient to determine their quality. For this reason, some new evaluation metrics, specific to paperboard images, are proposed and used. We present model architectures and training parameters that allow for sampling of new images, examples of generated images and the evaluation of them. The generated images bear a close resemblance to real images both when visually inspected and in terms of the used evaluation metrics. Finally, an alternative use of the models in this project is explained. It is possible to use a model to generate images larger than what it was trained on.

Populärvetenskaplig sammanfattning

Kartongen som utgör många av våra livsmedelsförpackningar har långt ifrån fullkomligt släta ytor. För att se detta kan man till exempel ta några bilder på kartong med ett kraftfullt mikroskop. Detta har Tetra Pak gjort och det visar sig att kartongytor består av fibrer som slingrar sig huller om buller. Utöver att beundras är dessa bilder användbara för att undersöka olika egenskaper hos kartongen. Specifikt är Tetra Pak intresserade av hur kartongen lämpar sig som förpackningsmaterial. Saker som att sätta dit tryck på kartongen och att laminera den med plast påverkas av fiberstrukturen. Tetra Pak använder kartongbilder för att simulera dessa processer, men för att simuleringarna ska ge tydliga resultat så behövs många bilder. Ett sätt att få fler bilder är självklart att ta dem med det ovan nämnda kraftfulla mikroskopet. Tyvärr är detta dyrt, så ett bättre sätt vore att generera artificiella bilder. Det är här som exjobbet kommer in i bilden.



Exempel på riktig kartongbild

Detta exjobb utgår ifrån verkliga bilder på kartong och tränar generativa diffusionsmodeller till att generera nya, liknande men inte exakt likadana, bilder. Tekniken bygger på att succesivt lägga till slumpmässigt brus till en riktigt bild och att sedan träna ett neuralt nätverk till att ta bort bruset. När man har tränat ett nätverk till att framgångsrikt ta bort brus från en bild så kan man använda det på en bild med bara brus, alltså utan någon riktig bild i grunden. Detta resulterar i en ny bild som är inspirerad av dem bilder nätverket är tränat på.

Resultaten från detta exjobb är modeller som kan generera bilder i två olika upplösningar, 64×64 och 256×256 pixlar. De mindre genererade bilderna går inte att skilja från sina verkliga motsvarigheter medan de större har vissa skavanker. I båda fallen är det dock inte bara de visuella intrycken som spelar roll. Eftersom att bilderna ska användas i simuleringar i syftet att bättre förstå förpackningsmaterial så vill vi också försäkra oss om de är lämpliga för det. För detta har några nya evalueringsmetoder tagits fram som visar att de genererade bilderna även stämmer överens med verkligheten när det kommer till simulering. Om så inte var fallet vore de genererade bilderna inte till särskilt stor nytta.

Sammanfattningsvis ger resultaten från det här arbetet förutsättningar för att bättre förstå egenskaper hos kartong när det används som förpackningsmaterial. I förlängningen kan detta leda till bättre livsmedelsförpackningar för oss alla!

Contents

1 Introduction

1.1 Background

Packages for various types of food, such as milk cartons, are made out of paperboard. Paperboard is, on a microscopic scale, a very uneven material as it is made out of wood fibres. This microscopic unevenness affects a large number of macroscopic properties of the package. For instance, the printability is affected by properties of the surface. The aim of this thesis is to generate artificial paperboard surfaces to be used in Monte Carlo simulations. This will allow for better understanding of certain characteristics of paperboard as a food packaging material. The surfaces modelled in this project can be seen as images and this turns the problem into one of image generation. Solving it involves implementing and training generative models that enable new images to be sampled as well as developing tools in order to analyse and characterise real and generated data.

This thesis was performed in collaboration with Tetra Pak. Tetra Pak uses paperboard surface topographies, which can be viewed as single channel images, as input to different types of simulations. The images are visually characterised by fibres going across the surface which can be seen in Figure 1.

The purpose of using images as input to simulations is to better understand the properties of paperboard when used as a food packaging material. Specifically, the structure and behaviour of the fibres are thought to play a key role. For the results of simulation to reflect real world variation, a lot of data is required. One approach is to scan vast amounts of paperboard surfaces to obtain more data. However, generating artificial data may prove to be a significantly more resource effective method if the generated data is of a good enough image quality.

Several things are included in what we mean by image quality. Firstly, the generated images should bear resemblance to real images when inspected by the naked eye. In many applications of image generation, this test is enough. If the images look good, they are good. For the purposes of this project, however, we need more. The generated images should match real images in terms of statistical measures such as image distributions and autocorrelation. Beyond that, we want images that yield results similar to real data when used as input to simulations.

1.2 History of generative diffusion models

The chosen method for images generation in this thesis is generative diffusion models (GDMs) which were first introduced by Sohl-Dickenstein et al. in 2015 [11]. This paper introduced the idea of gradually adding noise to data until it becomes Gaussian and then learning to reverse this process in order to recover the original data. These steps are generally called the forward process and the backward process. One of the core ideas behind GDMs is that while going from pure noise to a generated sample in one step is difficult, splitting the problem up into many, less challenging, steps makes the

de-noising feasible.

In 2020, Ho et al. [2] contributed significant improvements to GDMs. They introduced a simpler and more efficient training method. But, more importantly, their image quality was as good as or better than that of previous generative models. Their work ushered in a new era in which a plethora of articles have been written on the subject and diffusion has become the preferred method for image generation in commercial and open-source applications [9] [8]. The work in this thesis is primarily based on the formulation laid out in [2].

The main goal of this project is to implement and train GDMs that generate images that correspond to real paperboard images, both visually and in terms of statistics and for downstream simulations of packaging material properties. In order to achieve the main goal, different methods of analysis have to be considered and evaluated. Consideration must be taken to different limiting factors such as the available computational resources and, perhaps most importantly, the available raw data. Getting the most out of the available data will be crucial as GDMs are known to require a lot of data [9].

The implementation and training of GDMs is done in Python and specifically using the PyTorch library [7]. Neural networks are trained and used by connecting to Tetra Pak's in-house high performance computational cluster. Analysis is performed in Python and Matlab.

1.3 Previous attempts

The problem of generating images of paperboard topography has already been attempted at Tetra Pak. A first attempt was made with classical statistical methods in which the surface is modelled as a stationary process. The spectral density was estimated and then used to generate new surfaces. The generated data from these spectral models did not resemble real images visually. This is unsurprising as paperboard surfaces are obviously not stationary. Later, the model was expanded but became so complicated as to be impractical. Due to the complexity, it was decided to not pursue classical models further at the time. For this reason, a second attempt was made with the machine learning based method generative adversarial networks (GAN). This method was able to generate good quality images with 64×64 pixels, but struggled with finding a good training setup for larger images. [6]

With these previous attempts in mind, a goal of this project is to first of all produce good quality images with 64×64 pixel resolution. We also want to produce images with the larger resolution 256×256 pixels as this was not achieved when using GAN. To validate this project and the choice of GDMs for image generation, the results will be compared to those of previous attempts. At a minimum, we want to produce images which can be said be better than what classical methods were able to achieve.

2 Data

The problem tackled in this thesis is one of image generation. As is the case with all deep learning methods, GDMs require real data to learn from. Tetra Pak obtains this data through scanning paperboard with a coherence scanning interferometry microscope. For this project, 110 files that each contain the surface topographies of 1.56mm $\times$ 1.56mm areas of paperboard were made available by Tetra Pak. Each file contained height readings across the entire area along with the corresponding positions. This can be visualised as a greyscale image where the intensity of each pixel corresponds to the height at its position.

The 110 images come from 22 different types of paperboard i.e. we have 5 images from each type of paperboard. The paperboards differ in things like manufacturer, paperboard machine and stiffness. The exact points of difference are not important for this report. Rather, what we care about is the fact that we have 22 classes of paperboard. For the purposes of this report, we will name the classes $\{A, B, \ldots, V\}$. Figure 1 shows a few different examples of the images we are working with.

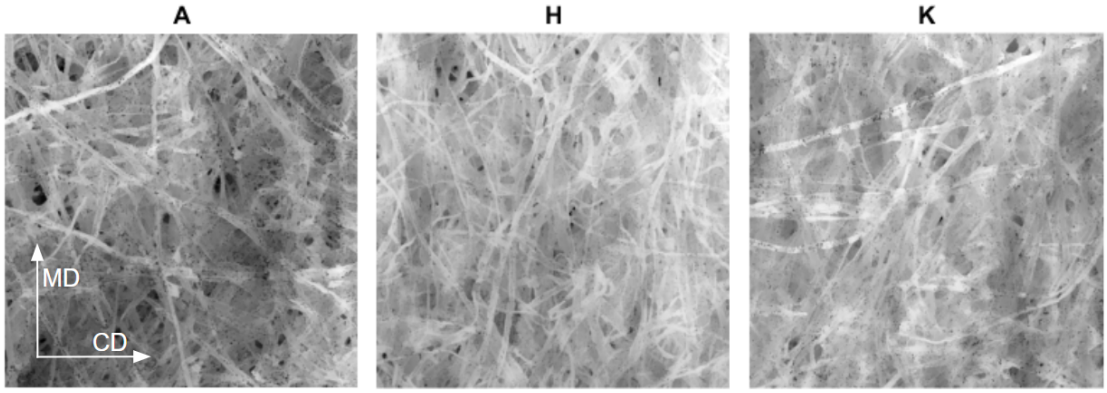


Figure 1: Unprocessed images and their corresponding classes. The directions of MD and CD, which correspond to the directions along and across a roll of paper are marked out.

When working with packaging material, like paperboard, it is common to not use x and y as directions. Instead, machine direction (MD) and cross direction (CD) are used as the direction in which material flows along the paper machine and the direction across the roll of paper respectively (Figure 2). The pieces of paperboard were scanned in such a way that MD corresponds to the vertical direction while CD is horizontal. All images displayed in this report follow this convention. It is noted that the materials studied are anisotropic. Fibres are stretched out more along MD which can be seen when analysing autocorrelations in different directions. Figure 3 shows the average autocorrelations in MD and CD of some randomly selected images.

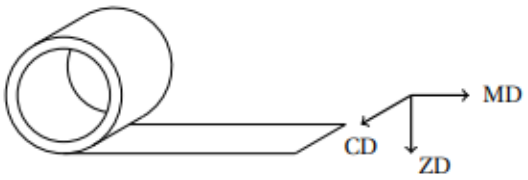


Figure 2: Machine and cross direction in relation to a roll of paperboard. ZD corresponds to height values in images.

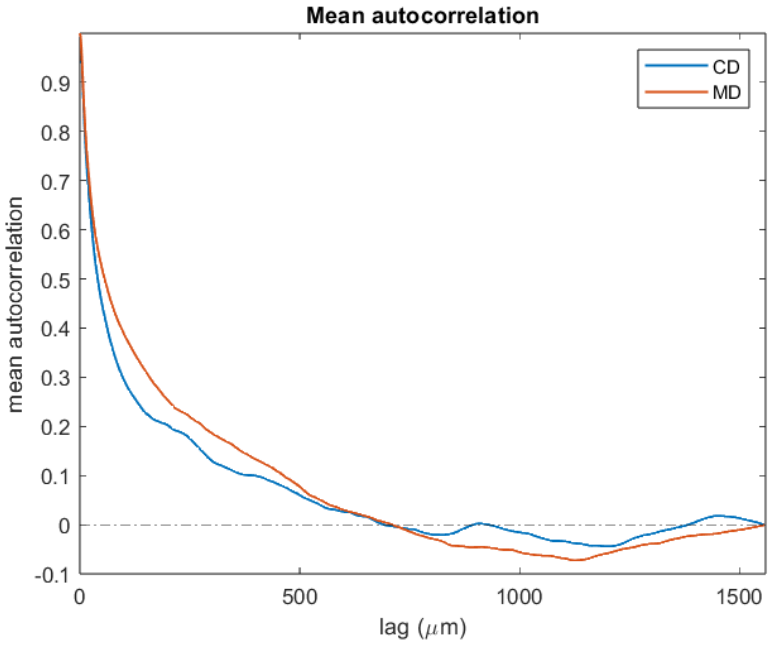


Figure 3: Average autocorrelation in MD and CD of raw images. Note that the correlation in the MD direction is more pronounced for small lag values compared to that of CD. This is due to the general orientation of fibres in the material.

2.1 Data pre-processing

Before using the raw data for training neural networks, some pre-processing had to be performed. We can see each image as a matrix

$$I_{raw} = \{z_{ij}\}, \qquad i, j \in [1, 1000]$$

where z_{ij} is the height reading in μm at position $[i, j]$.

Some of the images had missing values at a few points due to measurement issues. These NaN-values were handled by replacing the missing value with the average of all

surrounding non-NaN values. If there was a cluster of NaN-values, it was handled by recursively filling in the edges and then working toward the centre.

The images were also subjected to a high pass filter. This was achieved via convolution with a Gaussian kernel G_σ with standard deviation $\sigma = 100\mu m$ which corresponds to 64 pixels. The filtered image is then subtracted from the raw image which results in a high pass filtered image I_{hp}.

$$I_{hp} = I_{raw} - I_\sigma, \qquad I_\sigma = I_{raw} * G_\sigma$$

The reason for applying a high pass filter is to remove low frequencies in height difference which may exist across the surfaces. These variations in height over large distances may be due to the surface being tilted or unevenly placed during scanning. The purpose of the whole project is to analyse properties of paperboard which are related to the fibre structure itself and therefore we do not lose valuable information by filtering in this way. Conversely, removing low frequency variations allows the model to focus on capturing the fibre structure of the images. The value of $\sigma = 100\mu m$ was selected based on the typical length of autocorrelations in the images (Figure 3).

The final pieces of pre-processing relate to how the problem is approached in practice. Training directly on the raw data of 110 images with resolution 1000×1000 would prove to be impractical for two reasons. Firstly, size of the images would place too large demands on computational hardware. Secondly, using the raw data would mean that we only have 5 images images per class to train on. For these reasons, we divide the raw data into two datasets: **Data64** and **Data256**.

Data64 is made by taking square crops of the raw data with a crop size of 250 pixels and a step size of 17 in both MD and CD. These 250×250 crops are then scaled down to 64×64 using bi-cubic interpolation.
Data256 is made by taking crops of the raw data with a crop size of 256 pixels and a step size of 53. No interpolation is performed.

The resulting images in both datasets are also flipped along the MD and CD axis as well as rotated 180°. The point of these operations is to increase the amount of data that the models can train on. None of these operations violate the above described relationship between MD and CD in terms of autocorrelations. Here it is noted this processing reduces the amount of visual features included in each image. This could be detrimental when generated images are to be used as input to simulations. For this reason, it is important to strike a balance between cropping small (yielding more data) and cropping large enough (to preserve visual features. In Figure 4 we see that clear fibre structures are preserved in both datasets. Table 1 summarises the datasets used in this project.

Table 1: Datasets used in this project. Raw data is included for reference

	Raw data	**Data64**	**Data256**
Images per class	5	38720	4500

The height values z_{ij} in both datasets were rescaled in two ways in order to prepare the data for deep learning with neural networks. Firstly, the variance of height values within a given class of paperboard was calculated. Then each cropped image (in **Data64** and **Data256**) stemming from that class was divided by the corresponding variance. Secondly, the height values in each cropped image were adjusted such that their median was equal to zero. These modifications put the height values within a range that is suitable for image generation. This means that generated images will have height values in this modified scale. This must be considered when returning generated images to a realistic scale. Figure 4 shows a selection of the different images used in this project with a high pass filtered raw image as reference.

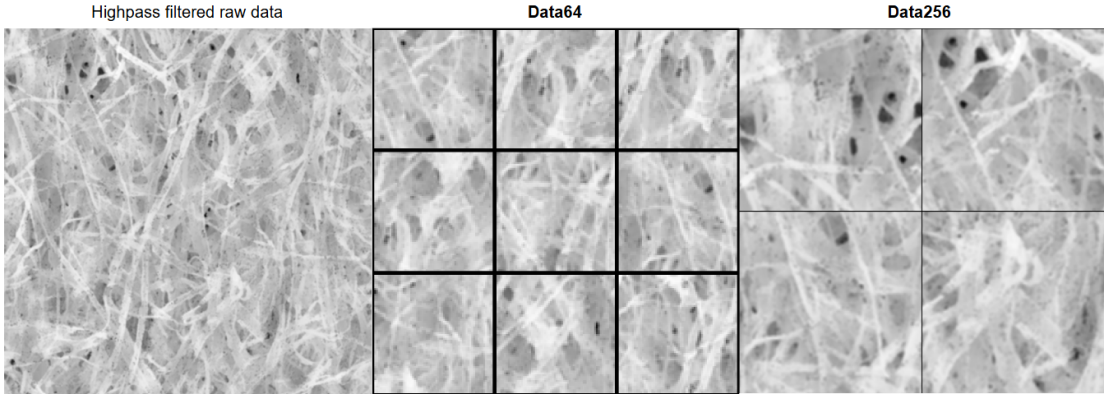


Figure 4: Samples from high pass filtered raw data, **Data64** and **Data256**. Note that the images are not to scale in MD and CD.

3 Theory of Generative Diffusion Models

GDMs are a form of unsupervised machine learning which model the underlying distribution of the data they are trained on. As stated in the introduction, the core idea revolves around gradually adding noise to training images and then training a network to undo that process. This section will describe the three main building blocks of GDMs and how we use deep learning for image generation. A method for controlling what type of paperboard gets generated is also described.

3.1 Forward process

The training data is assumed to follow an underlying distribution $q(\mathbf{x}_0)$ where $\mathbf{x}_0$ is a real image. In the case of an n×n greyscale image, $q(\mathbf{x}_0)$ will be an n^2-dimensional

probability density function. The ultimate goal of GDMs is to model this distribution. To start the forward process, an image $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ is sampled from the training data. Latent variables are then defined as $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$ where more and more noise gradually gets added. In practice, $\mathbf{x}_1$ will be very similar to $\mathbf{x}_0$ since only a small amount of noise has been added whereas $\mathbf{x}_T$, the final latent variable, will be almost pure noise. Noise is added in a cumulative manner and the transition probability is

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{1}$$

which allows us to express the transition $\mathbf{x}_t$ from $\mathbf{x}_0$ through

$$q(\mathbf{x}_t|\mathbf{x}_0) = \prod_{s=1}^{t} q(\mathbf{x}_s|\mathbf{x}_{s-1}) \tag{2}$$

In (eq. 1), $\mathbf{I}$ is Gaussian noise of zero mean and unit variance and $\beta_t, t \in [1, \ldots, T]$ is the *noise schedule* which sets the level of noise added at each timestep. In this thesis the total number of timesteps is set to $T = 1000$. In general, noise schedules are constructed such that small amounts of noise are added early in the forward process while more noise is added at later steps. The most common type of noise schedule is linear but other variations are possible. This thesis uses a cosine noise schedule where $\alpha_t = 1 - \beta_t$, $\overline{\alpha}_t = \prod_{s=1}^{t} \alpha_s$ and is defined in terms of $\overline{\alpha}_t$.

$$\overline{\alpha}_t = \frac{f(t)}{f(0)}, \qquad f(t) = \cos(\frac{t/T + k}{1+k} \cdot \frac{\pi}{2}) \tag{3}$$

The offset $k = 0.008$ is a standard choice in the literature [5].

Later during training, we will want to be able to directly calculate $\mathbf{x}_t$ from $\mathbf{x}_0$ without taking all intermediate steps that would be required with (eq. 2). $\overline{\alpha}_t$, which can be pre-calculated for all t, allows this through

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\sqrt{\overline{\alpha}_t}\mathbf{x}_0, (1-\overline{\alpha}_t)\mathbf{I}) \tag{4}$$

Some latent variables in the forward process are displayed in Figure 5 where we see that the last latent variable has enough noise to completely obscure the original image. This feature will be important in the sampling process.

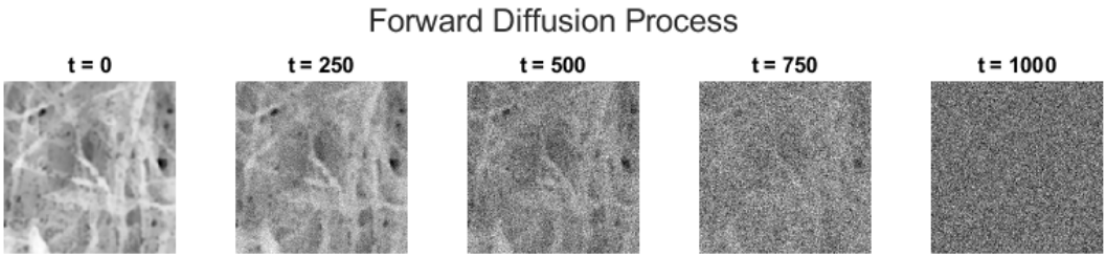


Figure 5: The forward process showed at 5 different timesteps. $t = 0$ represents the original image while $t > 0$ represents latent variables. Note that the final latent variable is indistinguishable from pure noise.

3.2 Backward process

The backward process describes transition probabilities in the opposite direction compared to the forward process. The key to GDMs is to estimate the backward transition probability $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$. In order to do this, we parametrise this conditional transition probability as

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \mathbf{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{5}$$

which introduces a dependence on t but crucially not on $\mathbf{x}_0$ as this is what we ultimately want to extract from the model. The two parameters $\mu_\theta(\mathbf{x}_t, t)$ and $\mathbf{\Sigma}_\theta(\mathbf{x}_t, t)$ are handled in different ways. The variance at each timestep is determined by the noise schedule in the forward process, so it will not be modelled. Instead it is taken to be $\mathbf{\Sigma}_\theta(\mathbf{x}_t, t) = \sigma_t^2\mathbf{I}$ where $\sigma_t^2 = \beta_t\frac{1-\overline{\alpha}_{t-1}}{1-\overline{\alpha}_t}$. This ensures that the modelling of the backward process is computationally efficient while maintaining consistency with the noise added in the forward process. The mean is modelled in terms of $\mathbf{x}_t$ and t as $\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\overline{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t))$. Where $\epsilon_\theta(\mathbf{x}_t, t)$ is a prediction of the actual noise that we have added to $\mathbf{x}_t$ [2]

This is where neural networks enter the picture. A convolutional neural network (CNN) will be trained to model $\epsilon_\theta(\mathbf{x}_t, t)$. A well trained network will be able to predict the noise well which in turn allows for precise modelling of the backward probability in (eq. 5). Conversely, a less effective CNN will lead to imprecise modelling and poor image quality within generated samples.

3.3 Sampling process

The final step of GDMs is the sampling process. The goal is to sample new, artificial, data from the underlying distribution of the training data $q(\mathbf{x}_0)$. This is done by starting the reverse process with pure Gaussian noise $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$. The random noise is then iteratively de-noised by going through the reverse process. The distribution of $\mathbf{x}_0$ when modelled by the neural network is:

$$p_\theta(\mathbf{x}_0) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t), \qquad p(\mathbf{x}_T) = \mathcal{N}(0, \mathbf{I}) \tag{6}$$

In practice, sampling is performed with the following algorithm from [2]

Algorithm 1 Sampling Algorithm

Require: Pretrained model ϵ_θ, noise schedule $\{\beta_t\}$, T, and $x_T \sim \mathcal{N}(0, I)$
Ensure: Sample x_0
1: **for** $t = T, T-1, \ldots, 1$ **do**
2: Compute $\mu_t = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right)$
3: Compute $\sigma_t^2 = \beta_t \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}$
4: Sample $z \sim \mathcal{N}(0, I)$ if $t > 1$, otherwise $z = 0$
5: Update $x_{t-1} = \mu_t + \sigma_t z$
6: **end for**
7: **return** x_0

3.4 Training

The goal of training is to learn the probability distributions of the backward process in (eq. 5). This is done with a CNN which takes a noisy image $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ and the timestep t as input. The expression for $\mathbf{x}_t$ comes from (eq. 4). The output of the CNN is a prediction of ϵ which is a scaling of the total noise that was added to get $\mathbf{x}_t$. An implication of this formulation is that the CNN actually tries to completely de noise the image from $\mathbf{x}_t$ to $\mathbf{x}_0$ rather than to go from $\mathbf{x}_t$ to $\mathbf{x}_{t-1}$. This may seem different from learning the backward transition probability (eq. 5) but, as we can see in the sampling algorithm above, the noise at step $t-1$ can be added back in to the predicted $\mathbf{x}_0$ yielding a predicted $\mathbf{x}_{t-1}$.

We get the natural training objective of the mean square error of the predicted noise and the actual noise $\mathcal{L} = \|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|_2^2$ which is minimized through the use of the Adam optimization method [3]. An example of what the training process looks like in included in the Appendix (7.3). The training process in summarized in the following algorithm [2].

Algorithm 2 Training Algorithm

Require: Training data $q(\mathbf{x}_0)$, noise schedule $\{\beta_t\}$, initialised neural network ϵ_θ
Ensure: Trained parameters θ
1: **repeat**
2: Sample $\mathbf{x}_0 \sim q(\mathbf{x}_0)$, $t \sim \text{Uniform}(\{1, \ldots, T\})$, and $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
3: Compute $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$
4: Optimize $\mathcal{L} = \|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|_2^2$
5: **until** converged

In practice, training is done by splitting the training data into batches of size S. On every iteration, S different values for t are sampled and a prediction of the total noise is made for each one. Optimization is done by taking the average gradient step from all S of the samples. Training is then continued over a preset number of epochs where all

training data is seen once during one epoch.

3.5 Convolutional Neural Network

This thesis uses two CNNs to model the reverse transition probabilities, one for each of the datasets **Data64** and **Data256**. The networks are named **CNN64** and **CNN256** and share the same basic structure. For clarity, we will describe **CNN64** and then provide an explanation of the modifications needed to get **CNN256**.

The network is inspired by the well know U-net architecture. This style of network was introduced in 2015 by Olaf Ronneberger et al. and was originally used for image segmentation [10]. However, the same type of network can also be used for GDMs, as is done here. The idea behind U-net is to first reduce the dimensionality of the input images though a series of convolutions. This is called encoding and it allows the network to first capture high resolution features of the data. As the image dimensions reduce further along the downsampling path, the convolutional kernels act over a larger area across the original image. This allows them to learn large patterns, e.g. fibres, across the image. When the image resolution drops, the number of channels is increased such that more convolutional kernels are available at deeper levels. The increasing number of channels is what preserves information as the images get smaller. The next step is called the bottleneck where data is processed at the lowest resolution. This allows the modelling of patterns that potentially stretch over the whole image. The encoded data is then fed into a decoder which recreates the original dimensionality of the image through a series of convolutions and convolutional transposes. The purpose of the upsampling path is to take all features learned in the downsampling and bottleneck stages and recreate an output with the same resolution as the input. To aid in this, another core feature of the U-net called skip connections is used. Skip connections copy the data at several points during downsampling and then concatenate this data to the upsampling path. This allows the upsampling to use a combination of more processed data (the main path of the network) and data with more fine details (skip connection) when creating the output image of the network. The network is constructed by blocks and its architecture is shown in Figure 6. Due to the skip connections, the upsampling block must have double the amount of input channels in order to be able to take the main path and skip connections. Both streams of data are combined and processed together within the upsampling blocks.

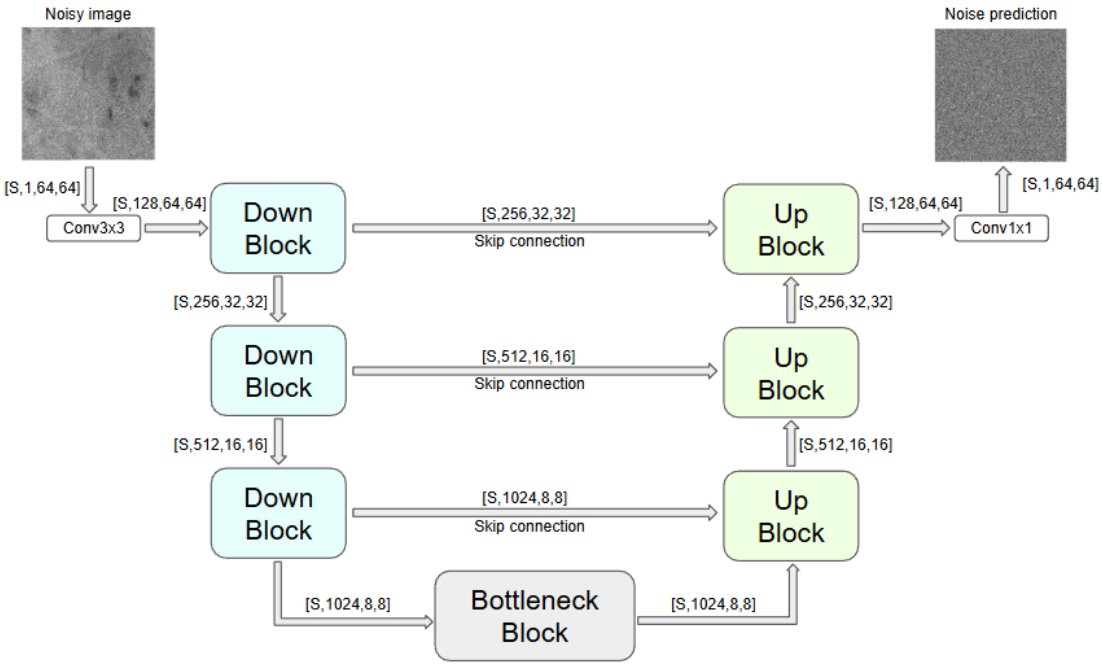


Figure 6: Network architecture for **CNN64**. The tensor (multi dimensional matrix) dimensions along every connection is included and should be read as [batch size, channels, height, width]. The batch size is here denoted as S. Note that due to skip connections, each upsampling block has two inputs.

As we see, the data is processed in modular blocks and the inner workings of these is what allows the network to perform its function. All blocks share the same general structure but differ slightly based on the type of block. Convolutional layers are used for learning features of the data. ReLU layers are used to introduce non-linearities without which training would not work. Batch normalisation layers are used to prevent instability during training. By unstable training we mean than the steps taken by the optimisation algorithm do not lead to convergence. The blocks also include a residual connection (the lower path in the blocks) which is inspired from the well known ResNet. These allow features from the input to flow freely further into the network and have been shown to greatly improve the performance of deep neural networks [1]. The three block types are described in Figure 7.

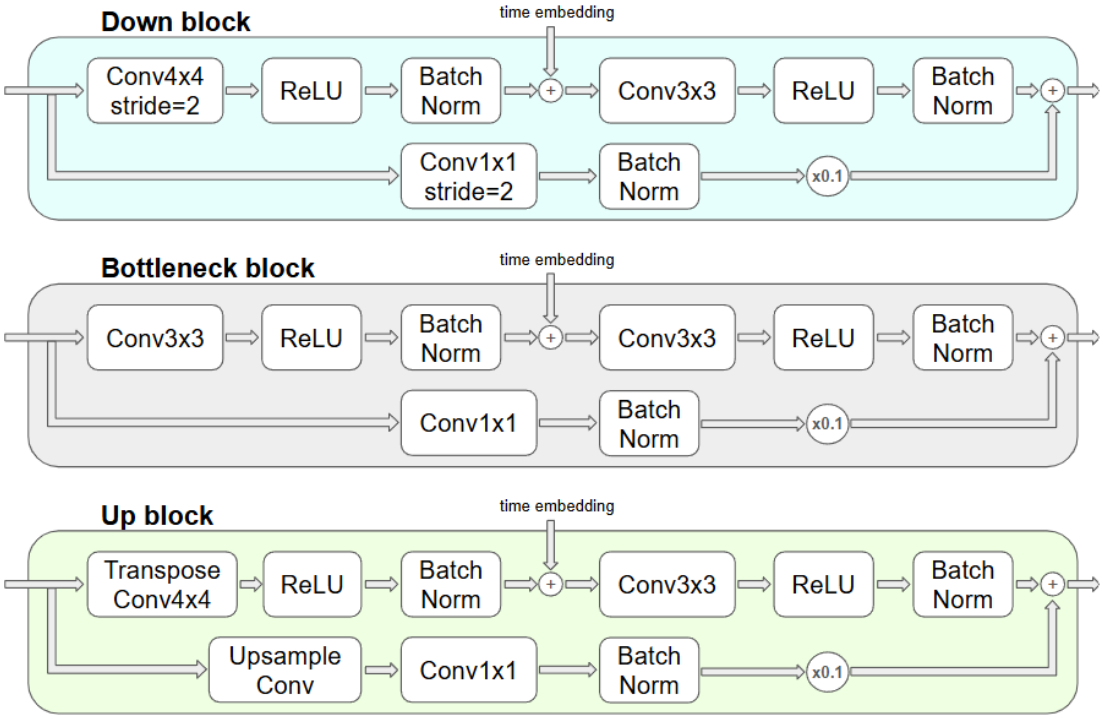


Figure 7: Components of blocks in neural network. All convolutions are padded such that image dimensions are preserved except where the convolution is intended to change the image resolution. In the upsample block's residual path, upsampling is handled separately with bilinear interpolation. Note that the residual connection is multiplied by a factor 0.1. This is to prevent unstable training. The time embedding will be explained in a later section.

For clarity it is noted that the overall structure of the network architecture (Figure 6) is taken directly from the original U-net paper [10]. The inner workings of the blocks (Figure 7) are designed with the data processing requirements of the network in mind. They are not taken directly from any source but are rather a collection of standard components for convolutional neural networks. The residual connections within the blocks are directly taken from ResNet [1].

3.6 Time embedding

What we have seen so far only concerns the manipulation of the images and their pixel data throughout the network. However, in order to predict the noise at a given timestep, information about the time must also be fed into the network. Just feeding a scalar into the network will not work. Instead, the time is embedded into a vector via an embedding function $f(t)$

$$f : \mathbb{Z} \to \mathbb{R}^{128}$$

The output dimension of 128 is an arbitrary choice and the details of the embedding function can be found in the Appendix (7.1). The purpose of the embedding vector is to encode the timestep for every block of the network (Figure 7). For this to be possible, the 128 dimensions of the embedding vector must be adjusted to the number of channels in a given block. This is achieved by running the embedding vector through a multi layer perceptron (MLP) which output matches the number of channels at the relevant block (Figure 8). The use of a MLP allows the time embedding to be added element wise to the image data being processed in the blocks of the network. In practice, every block of the network will have its own MLP and their weights will be learnable during training. This whole setup allows the network to account for the timestep at all levels of processing which is crucial for its functionality [12].

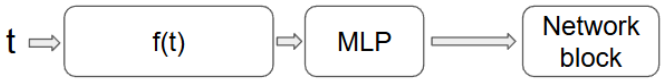


Figure 8: Encoding of a scalar timestep into each block of the network.

3.7 CNN256

For training on higher resolution images in **Data256**, the network needs to be adapted somewhat. The number of blocks and their corresponding channels are changed to meet the demands that larger images place on the model while still taking memory constraints into account. **CNN64** uses the channels $[128, 256, 512, 1028]$ with three down and up blocks. **CNN256** uses the channels $[128, 128, 128, 256, 512, 512]$ which require five blocks in the downsample and upsample paths respectively. The architecture of **CNN256** is chosen such that the image resolution in the bottleneck block is the same as for **CNN64**. Another difference in **CNN256** is that it does not use residual connections within the blocks. This is motivated by the fact that training on larger images, especially under memory constraints with small batch sizes, can be unstable. Removing the residual path contributes to stability during training and also frees up memory for larger batch sizes. For completeness, tables of the components of both networks used in this thesis are included in the Appendix (7.2).

3.8 Class conditioned sampling

This section concerns a method which was not ultimately used in the final models of this thesis. However, it is described here in order to be the subject of discussion later on.

The data used in this project is divided up into 22 different types of paperboard named $A, B, \ldots, V$. If we want to be able to control which type we generate, we must feed the network with class information. This is handled in a similar way to how information about timesteps was introduced. We make use of PyTorch's built in function *nn.Embedding* which allows us to map the class into a 128 dimensional vector. Similarly to the time embedding, this vector is also fed through a trainable MLP in order to

match the number of channels in a given block. The two outputs from the MLPs (time and class embeddings) are added together and then fed into the blocks of the network. This method allows the network to account for both timestep and class at all blocks of the network. The flow of data when this method is used can be seen in Figure 9.

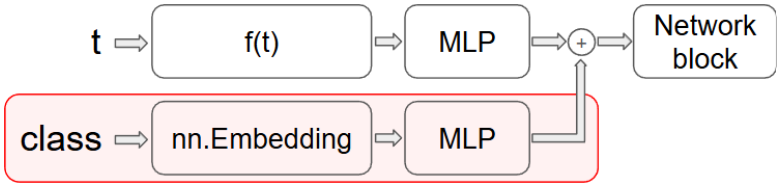


Figure 9: The flow of the timestep and class embeddings into each block of the network. This process takes place at every block of the whole network. Note that the class embedding path (marked in red) was ultimately not used in this thesis.

4 Method

In order to achieve the main goal of this thesis, to generate images of paperboard of good quality, the theory of GDMs described above was implemented into several different models. This section describes these models as well as some evaluation methods used to test the generated images.

4.1 Neural networks

The data in this thesis is split up into the two resolutions 64×64 and 256×256 pixels. The original aim was to use class conditioning and train one model for each resolution. This would allow for generation of all 22 different types of paperboard with a single model at a given resolution. This approach was attempted but ultimately abandoned due to difficulties with getting good image quality. The results of these attempts will be shown in a later section. Instead, it was decided to split the data up into smaller groups and train separate models on these. The splitting into groups was made based on the types of paperboard machines that the paperboards come from. E.g. classes A and B share the same paperboard machine so they were combined into a group. Table 2 shows the different groups of data modelled in this thesis.

Table 2: Classes of paperboard combined into groups. The classes combined share the same paperboard machine. Note that each groups correspond with a separate trained network.

Resolution	**Groups**		
64×64	A+B	C+L+M	N+O+U+V
256×256	A+B		

This way of splitting the data does not cover all available 22 classes. However, it is

sufficient for showing that this generation technique works as well as evaluating its results. It is trivial to use the methods laid out in this thesis to train models on other classes of paperboard in the future. In total, three neural networks called **CNN64-AB**, **CNN64-CLM**, **CNN64-NOUV** were trained on 64×64 images. One network, **CNN256-AB**, was trained on 256×256 images. All training was performed with the Adam training algorithm with training parameters according to table 3

Table 3: Training parameters used for the different networks.

Network	**Learning Rate**	**Beta1**	**Beta2**	**Batch Size**	**Epochs**
CNN64 (all)	0.001	0.9	0.999	128	300
CNN256	0.0005	0.9	0.999	32	300

4.2 Evaluation

As is stated in the introduction, we require generated images to not only to look like real images upon visual inspection. We need to characterise paperboard images in a quantitative manner and compare generated images on the same metrics. It is noted that merely analysing pixel value distribution is insufficient in this context because vastly different images can still have matching image distribution. E.g. randomly shuffling the pixels of an image will not affect the image distribution. Specifically, we are interested in how images behave when used in simulations. For this reason, the first two evaluation methods described in this section are based on a typical simulation. A third method based on classification is also described.

4.2.1 Contact mechanics simulation

One important feature of surfaces is how they deform when subjected to an external load. A simple case is when a solid plate gets pressed against the paperboard surface. The surface will deform due to being in contact with the plate and this will result in some force between to two objects. The situation is described in Figure 10.

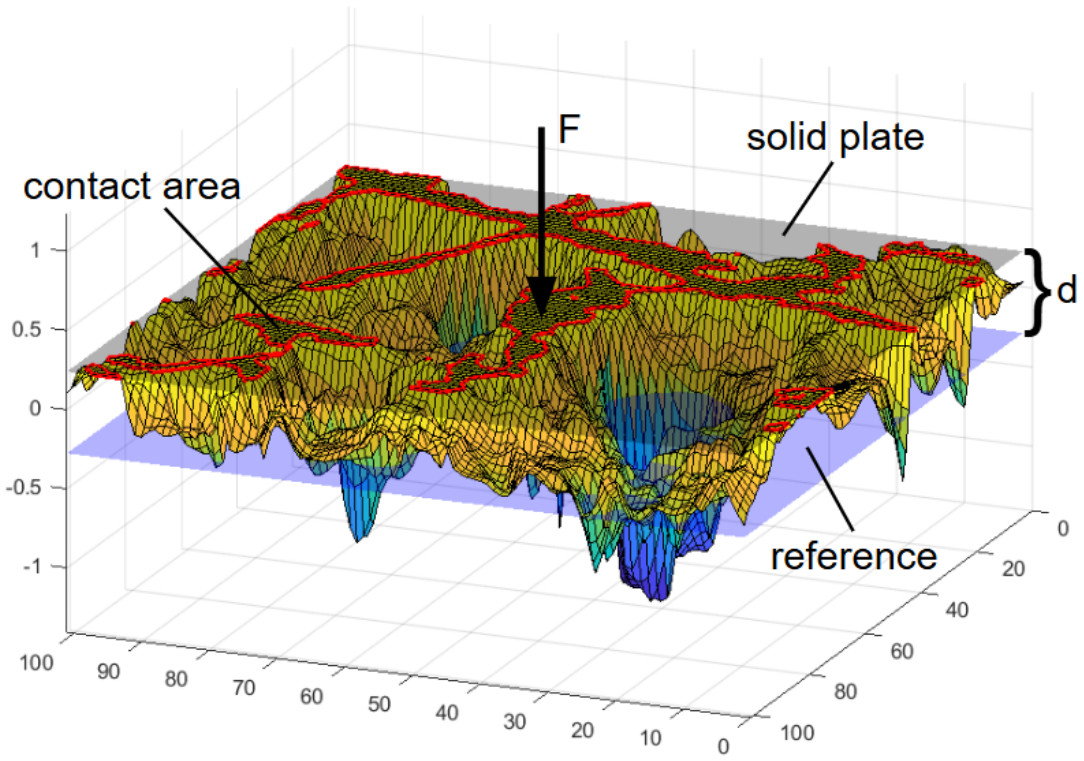


Figure 10: The solid plate is moved to some distance d from a reference plane located at the median height of the surface. This deforms the surface in and around the contact area (boundary highlighted in red). Some force F is required to keep the system in balance.

We are interested in how the relationship between F and the contact area changes with d. This can be found out by using the boundary element method. The details of this method are beyond the scope of this thesis. Tetra Pak provided a script that computes F and the contact area for a given distance d. The script is used by sweeping over a range of distances and plotting F and d. Figure 11 shows an example of this.

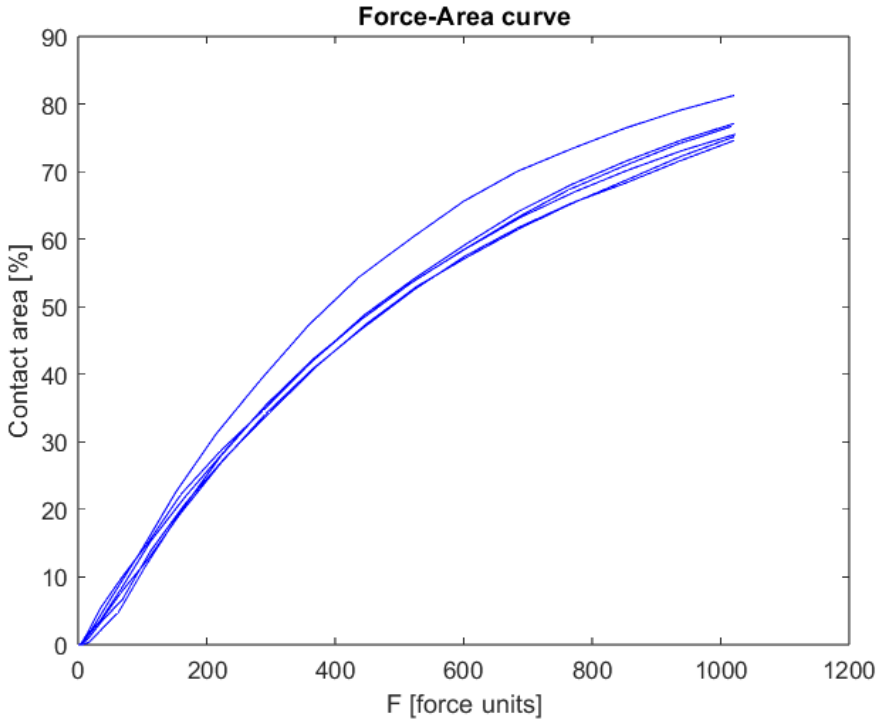


Figure 11: 6 examples of force-area curves coming from real surfaces. The script used expresses force in force units. Note that the unit is irrelevant for analysis of this graph.

Simulations like this are commonly carried out when studying properties of paperboard surfaces which is why it is used here. The force-area curve provides a relevant method for characterising surfaces and can therefore be used to validate generated surfaces. If the force-area curves of generated surfaces match those coming from real data, it supports the claim that the generated surfaces are suitable for further simulations.

4.2.2 Distance to contact area

The contact mechanics simulation described above can be used in another way to characterise surfaces. Instead of sweeping over different distances, we fix the distance and solve the deformation problem once for a given surface. This results in a binary image describing the contact area between the paperboard and the solid plate. For all non-contact points, the distance to the closest point is calculated. The distribution of these distances can be seen as yet another way of describing the original surface for some given distance d. For all analysis in this thesis, we fix $d = 1$ as this value seemed to generate the most expressive distributions. Figure 12 visualises the steps required for this evaluation method.

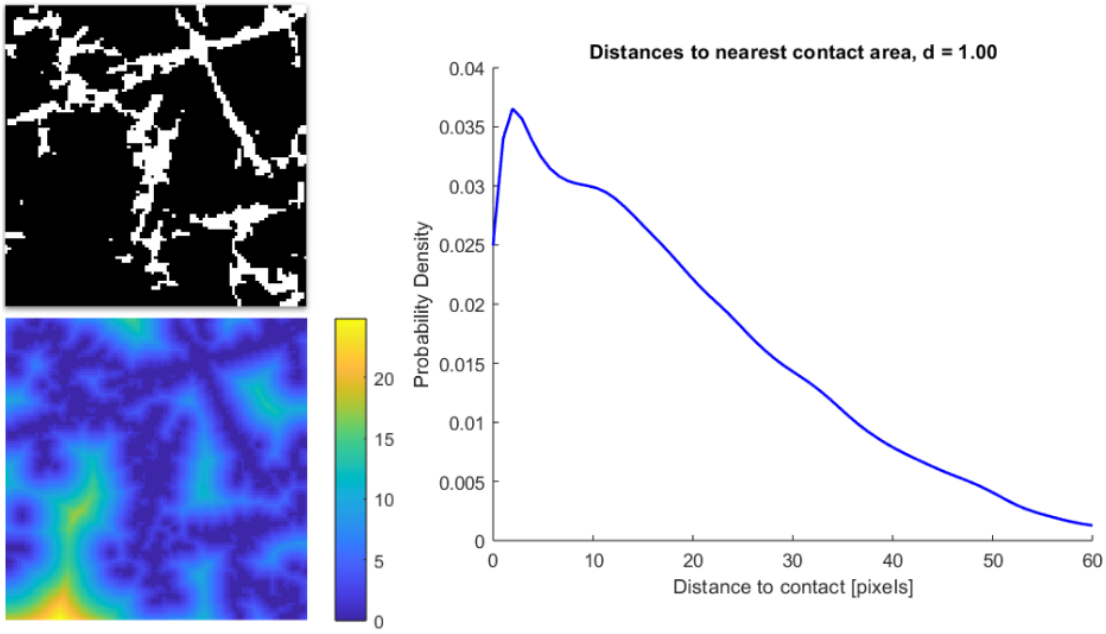


Figure 12: Contact area (top left) leads to distance to contact (bottom left). The distances are plotted as a distribution (right). The distribution is taken from many real images to show the general trend.

4.2.3 Classification network

The real images used in this thesis come from paperboards of different types. The data is ultimately combined into groups based on their paperboard machine and GDMs are trained on each group. We want to ensure that generated images share characteristics with the group that they were based on. This is tested by implementing neural networks for classification and seeing if generated images get classified similarly to real images. The networks for both resolutions were trained on 4 out of 5 of the original images from each class. After training, the left out data is classified to create a baseline confusion matrix (Figure 13) for how real images should get classified when subjected to the network [4]. The classification of generated images will be evaluated against this baseline. The architecture of the classification network is included in the Appendix (7.4).

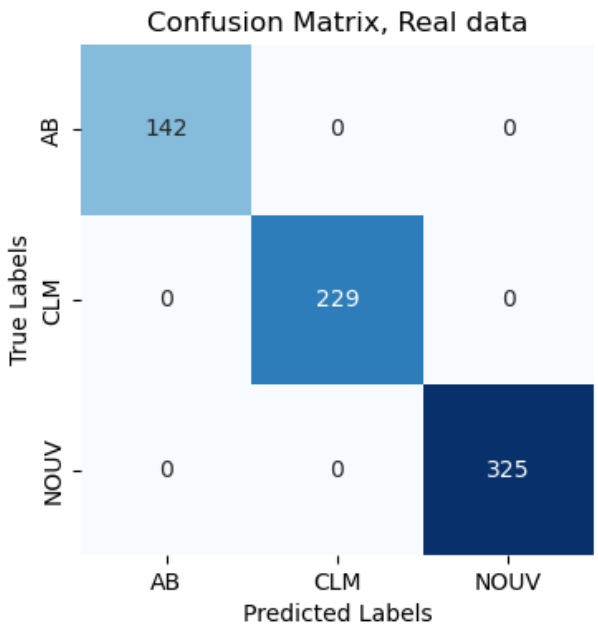


Figure 13: Base line confusion matrix from classification of real 64×64 images. We see that the network achieves perfect accuracy on the real images.

5 Results

This section presents generated images from the various networks and results from the evaluation methods described above. The section is ended with the results from attempting to train with class conditioning. A comparison between diffusion and spectral models as methods for generating paperboard surfaces is also included.

5.1 Visual results

A random selection of images from the 3 64×64-pixel models, together with examples of real images from the same groups, are displayed in Figure 14. Figure 15 shows the same for **CNN256**.

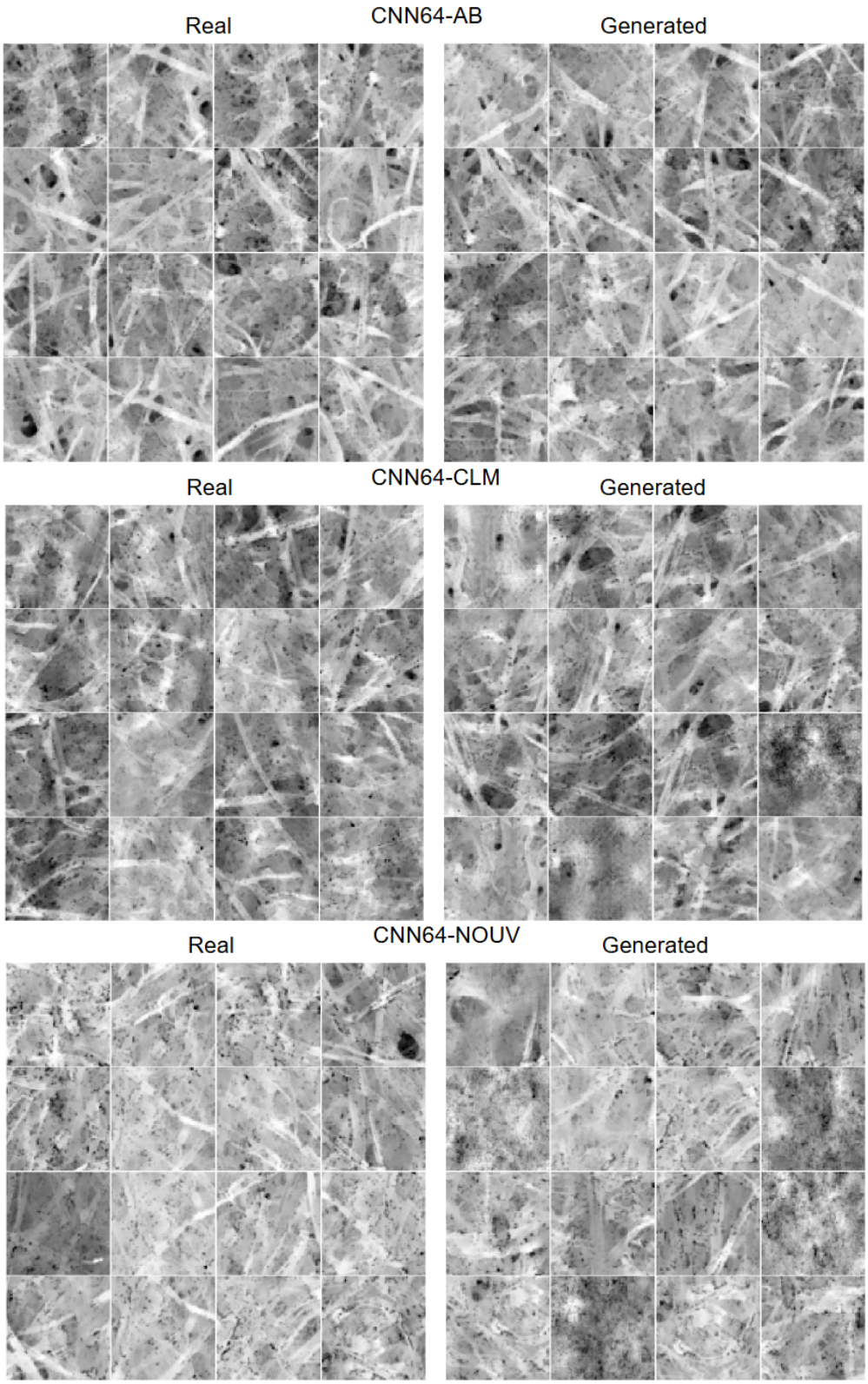


Figure 14: Real and generated images from **CNN64-AB**, **CNN64-CLM** and **CNN64-NOUV**.

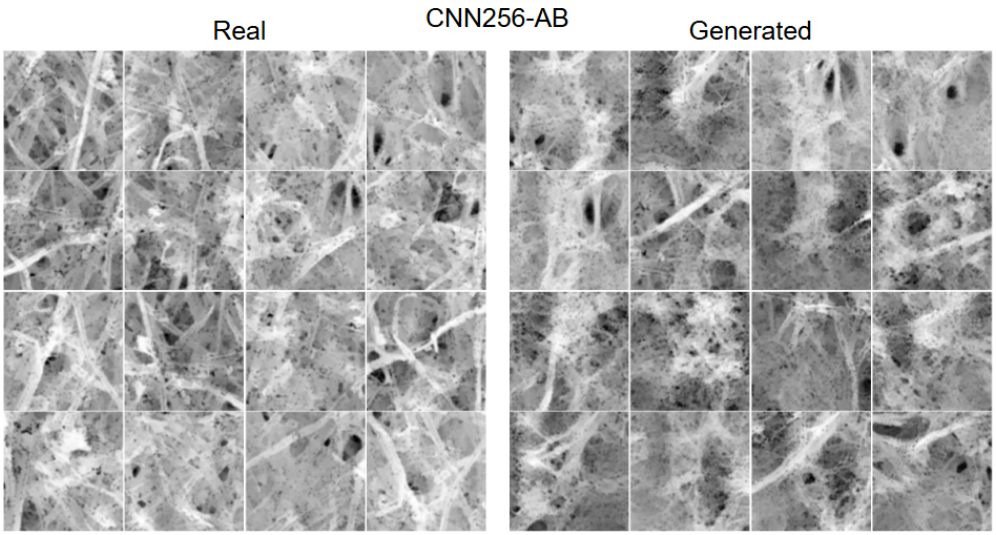


Figure 15: Real and generated images from **CNN256-AB**

5.2 Contact mechanics

The force-area curves from running the contact mechanics script on data from all models are displayed in Figure 16.

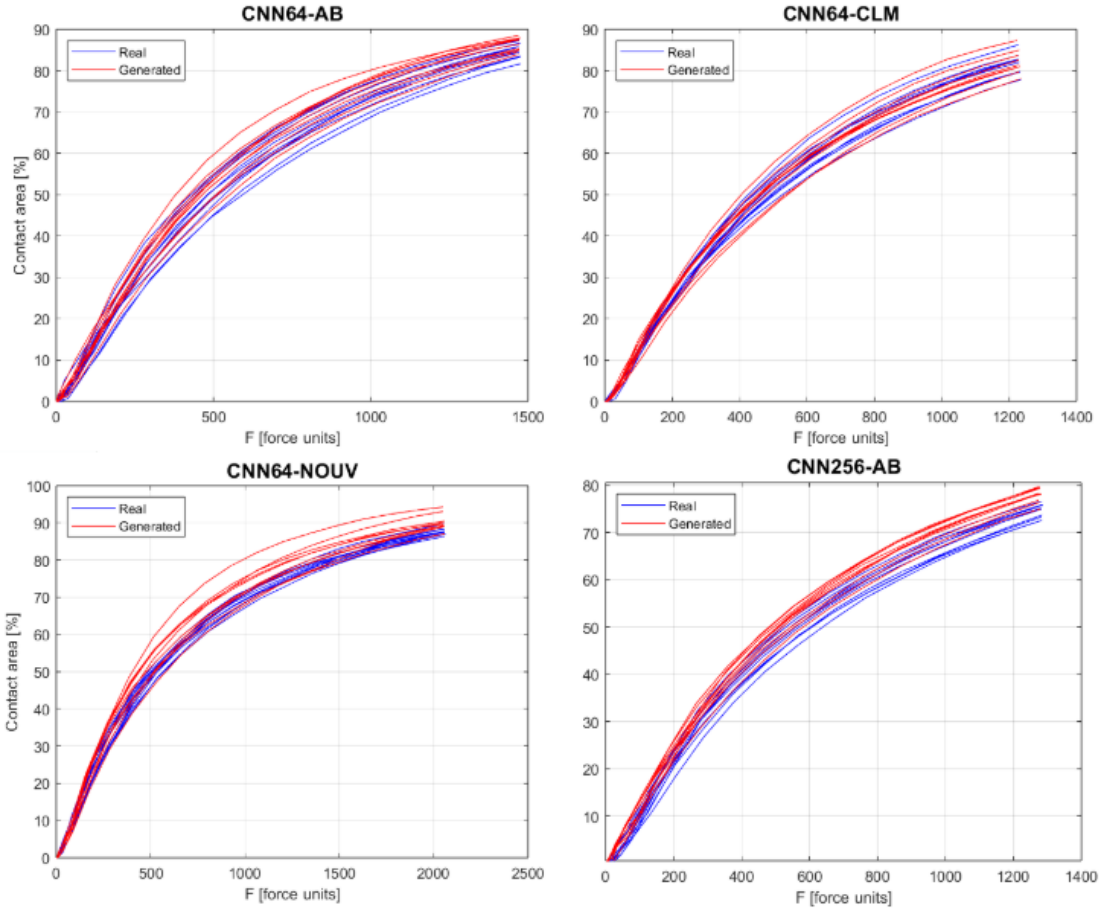


Figure 16: Force-area curves from **CNN64-AB**, **CNN64-CLM**, **CNN64-NOUV** and **CNN256-AB**. In each plot, 10 generated images are compared to 10 real images from the corresponding group.

5.3 Distance to contact area

Distributions of distances to nearest contact are displayed in Figure 17.

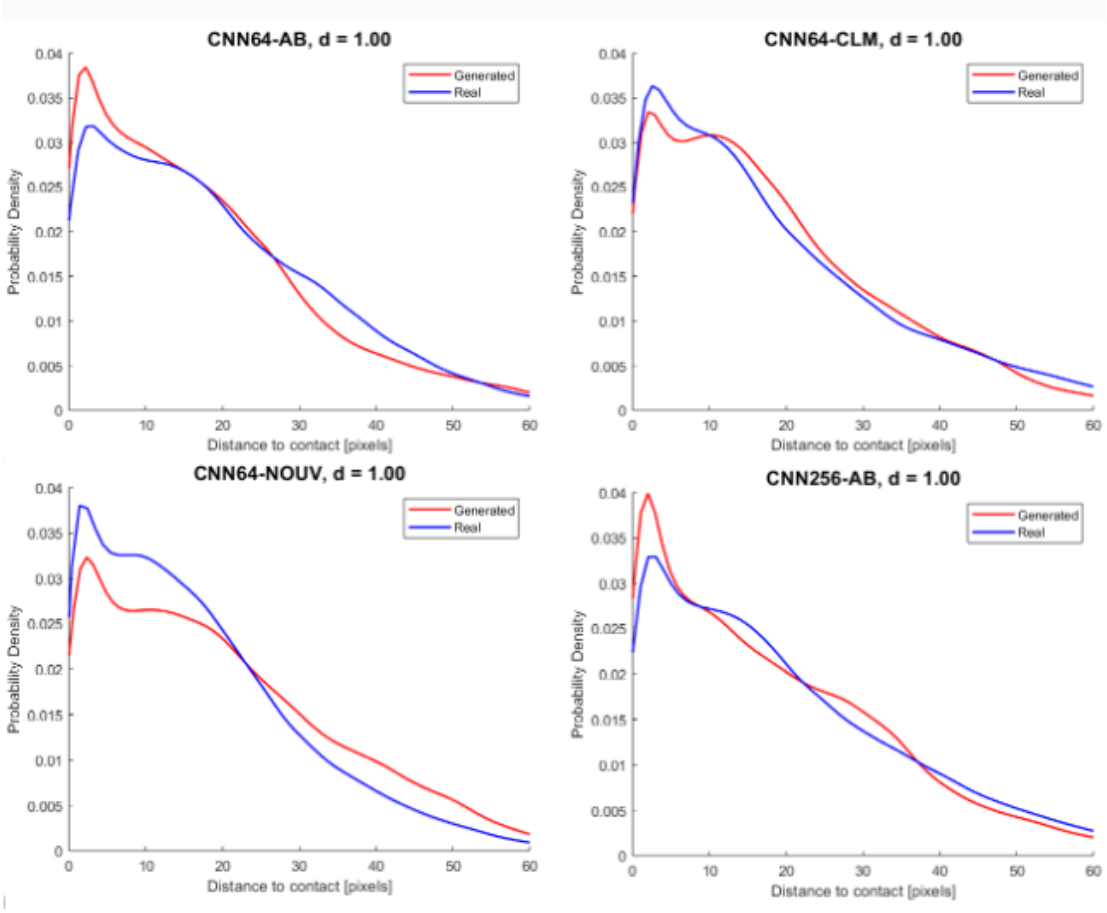


Figure 17: Distance to contact distributions from **CNN64-AB**, **CNN64-CLM**, **CNN64-NOUV** and **CNN256-AB**. In each plot, 16 generated images are compared to 10 real images from the corresponding group.

5.4 Classification

Generated images from all 64×64-pixel models were classified using the classification network. The resulting confusion matrix is in Figure 18. Since only one model was trained on 256×256 images, it is not possible to analyse those images in this way.

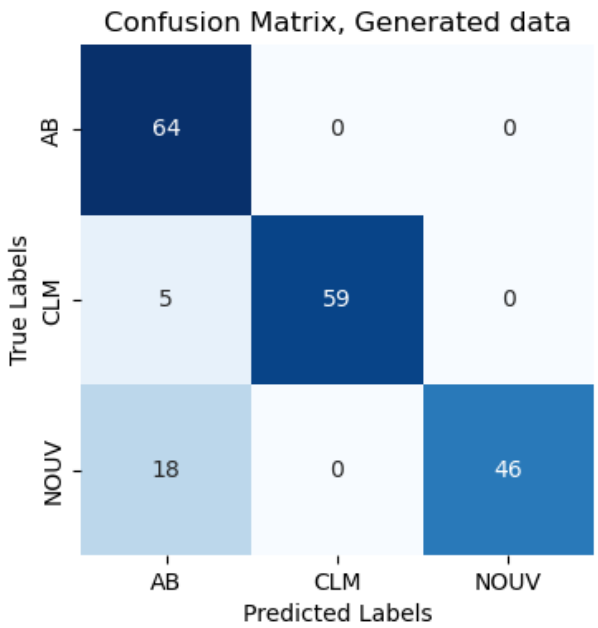


Figure 18: Confusion matrix when generated data from **CNN64-AB**, **CNN64-CLM** and **CNN64-NOUV** is classified with a neural network.

5.5 Results from class conditioned models

The method of class conditioning was used to train various models none of which feature as any of the final network of this work. The reason for this is that the image quality greatly suffered when class conditioning was used. Some example images from this type of model are displayed in Figure 19. The reasons for the lacking image quality are addressed in the discussion.

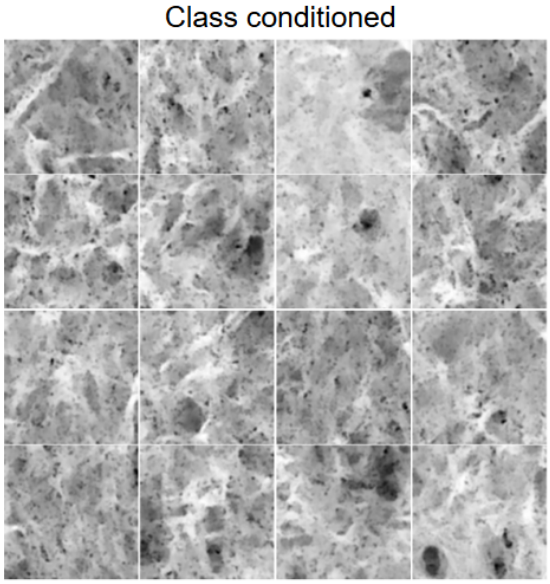


Figure 19: Generated images from model with class conditioning. Each of the images are conditioned on a random class.

5.6 Comparison to spectral models

As was stated in the introduction, this problem of generating paperboard surfaces has been previously attempted at Tetra Pak with spectral models. These are based on the naive assumption that the surface can be modelled as a stationary two dimensional random process. The spectral density of real images can be estimated and new images can be generated from this density. An example of one such image is shown in Figure 20. As we can see, the image does not resemble real paperboard visually. This is not surprising as the assumption is obviously wrong. Paperboard surfaces are not stationary due to the existence of e.g. fibres. However, for this project to be valid, the images generated with diffusion must be better than images from a spectral model when used as input to simulations. To check this, we take random crops from the spectral generated image and perform the simulation with these. The results are then compared with real data as is also seen in Figure 20. The generated images give rise to flatter curves than their real counterparts. When we compare this graph to those of Figure 16, it is noted that the spectral generated images perform less similar to real data than images generated with diffusion. This serves to further validate the use of GDMs to generate paperboard surfaces.

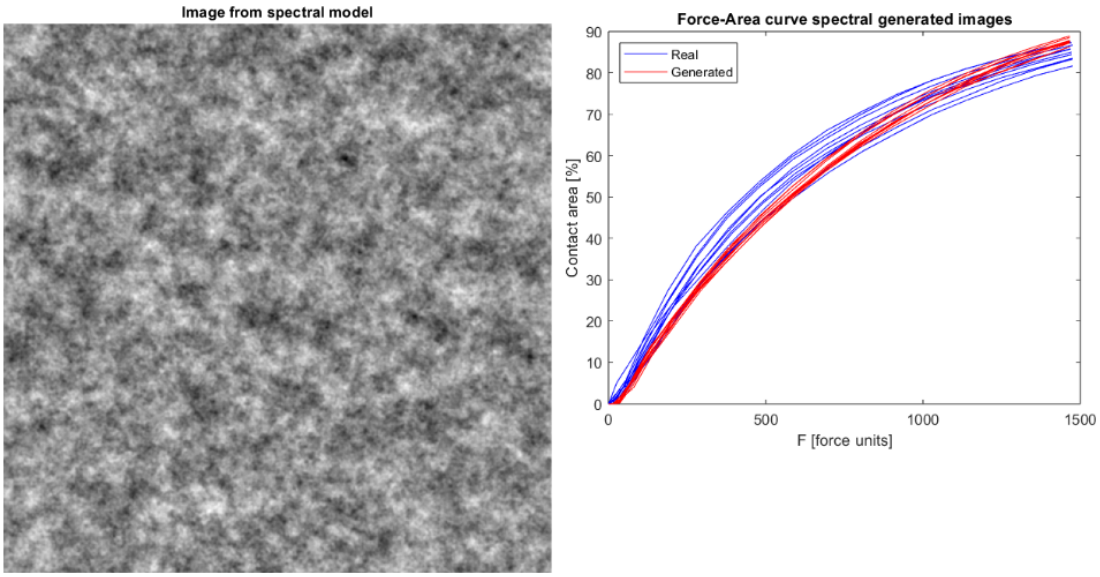


Figure 20: Example of image from spectral model which is based on paperboard class A (left). Force-area curves from a spectral generated image and real images (right).

6 Discussion

To conclude this project, we discuss its results and methodologies from a few different perspectives. At the end, an outlook of what further work can be done is provided.

The generated 64×64-pixel images (Figure 14), bear a close visual resemblance to the images their models were trained on. Fibre structures and holes, which are characteristic of paperboard surfaces, feature in the generated images. This result is not surprising as diffusion models have been shown to be capable of generating good quality images again and again. However, it is clear that the 256×256 images (Figure 15) leave something to be desired in their visual presentation. While fibres and holes are present in parts of images, other parts are noisy.

The worse results for larger images are likely due to issues with stability encountered during training and generation. The available GPU restricted the batch size for large images to 32 (compared to 128 with smaller images) which makes training less stable. Many attempts were made to find a combination of model and training setup which produces good, stable, results for larger images. Such a setup is surely possible, but we were not able to achieve it during the time-frame of this project.

Another challenge with this project was the limited amount of available raw data. Doing no pre-processing and training on five raw images per class would likely have resulted in a model at best capable of recreating the raw data i.e. extreme over fitting.

For this reason, some pre-processing had to be done. Taking different crops of lower resolution than the original image is a method of creating more data and to broaden the underlying distribution $q(\mathbf{x}_0)$ of the data. However, as the crops have to overlap each other, many of them will contain the same visual features. For example, the same fibre will feature in many overlapping cropped images. This means that the network will have a quite limited amount of different visual features to learn from. The network will generate images with learned visual features stitched together which can be seen in Figures 14 and 15. This can be seen as another form of over fitting where the model is unable to come up with many new features. It is noted that this type of over fitting is not necessarily a big problem in the context of paperboard images. The images are not made to be looked at. As long as they facilitate simulations, they have served their purpose.

Beyond visual results we have the evaluation metrics of contact mechanics to analyse. For all models, the force-area curves of generated images (Figure 16) correspond well to real data. At the very least the similarity is better than what the previously developed spectral models achieve (Figure 20). We can say with more confidence that images generated with diffusion suited to be used as input to simulations. The distances to contact areas (Figure 17) show a general resemblance between real and generated images which further validates the models. Both of these evaluation methods are developed with the specific use case of the images in this project in mind. They are a result of the inherent difficulty of evaluating images. Simply looking at metrics like image distributions or autocorrelations is not sufficient to characterise a paperboard image. In the case of image distributions, randomising the position of every pixel while keeping their value will not impact the distribution. In the case of autocorrelations, the images generated from spectral models have identical autocorrelations to real data while looking nothing like paperboard surfaces.

It is noted that the proposed evaluation metrics do not offer a complete characterisation of paperboard images. They are based on just one simulation and does therefore not guarantee that the generated images will perform similar to real ones when used in different ways. If the generated images from this project are used in other types of simulations, it could be discovered that they perform very differently in other contexts.

The classification network used on the 64×64 images is able to perfectly place real images to their paperboard group (Figure 13). This shows that there is a significant difference between different types of paperboard and validates the decision to split up data into different groups in the first place. When classification is performed on generated data, we see that the differences between groups is preserved by the models to some extent (Figure 18). Here, it is noted that the cases of classifying real and generated images are different. For real data, the classification network is evaluated on a board not seen during its training so we have a separation of data. When classifying generated images, the GDMs base their outputs on all the underlying data. In order to achieve separation of data with generated images, one would have to train GDMs on data not

seen by the classification network. Given the limited amount of data, this approach is seen as unfeasible.

The attempts to implement class conditioning into the models failed because it was not possible to generate good quality images with this method (Figure 19). The original plan was to have one model per image resolution capable of generating images of all classes. While this would have been elegant, the approach of training separate models for each group had to be settled for. In the network, class information was combined with time step information and it is possible that the interference of these two led to improper training. This combined with potentially sub-optimal network and training parameters led to poor images. Again, it is surely possible to find a network and training parameters that achieve a good class conditioned model. This is perhaps something to investigate further.

To conclude, this project has managed to produce good quality 64×64-pixel images and used some bespoke metrics to evaluate them. Some insights into the challenges of producing larger images with class conditioning have also been gained. The models presented can be used to generate more artificial data. It is also possible to train new models on other classes of data with the training methods provided.

6.1 Further work

Several topics in this project can be expanded upon. The most obvious thing to investigate further is a model and training setup that produces better 256×256-pixel images. It would also be interesting to see if class conditioning could be made to work. There exists several techniques for conditioning on classes with GDMs and only one was tried here.

Another thing to highlight here is an alternative use of the models developed in the project. While the models are trained on images of a certain size in terms of pixels, it is possible to use them to generate larger images. For example, a 256×256 Gaussian image can be de-noised by a model trained for 64×64-images. Four such images are displayed in Figure 21. This way of using the models should be investigated further as larger images are likely more useful to for some types of simulations.

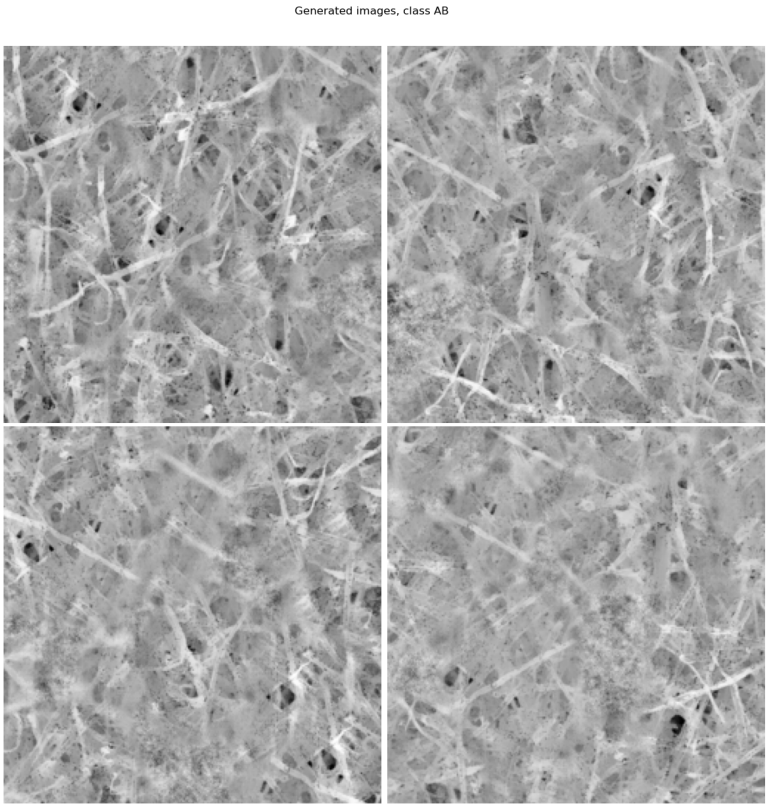


Figure 21: Four 256×256-images generated with **CNN64-AB**

Finally, it is noted that the method used in this thesis (GDMs) is not limited to image generation. It is applicable in other domains such as time series data or 3D data.

7 Appendix

7.1 Time embedding

The embedding has total dimensionality $D = 128$ which is split up into two equal parts of $D/2 = 64$ dimensions each. A frequency scaling factor is defined as

$$\text{scale} = \frac{\log(10000)}{D/2 - 1}$$

which allows us to define a vector of frequencies

$$\omega_k = \exp(-k \cdot \text{scale}), \qquad k = 1, 2, \ldots, D/2$$

where k is the index of each frequency. The final time embedding vector for a given timestep t is defined as

$$\text{embedding}_k = \begin{cases} \sin(t \cdot \omega_k), & \text{if } k \leq D/2 \\ \cos(t \cdot \omega_{k-D/2}), & \text{if } k > D/2 \end{cases}$$

7.2 CNN64 and CNN256 architectures

The architectures of the two networks used in this thesis are described in tables 4 and 5

Table 4: **CNN64** architecture.

Layer Type	**Input Shape**	**Output Shape**	**Details**
Initial Conv	$1 \times 64 \times 64$	$128 \times 64 \times 64$	Conv2D(kernel=3, stride=1)
Downsampling Blocks			
Downsample Block 1	$128 \times 64 \times 64$	$256 \times 32 \times 32$	Main path: Conv2D(4, 2), ReLU, BatchNorm, Time Embedding, Conv2D(3, 1), ReLU, BatchNorm Residual Path: Conv2D(1, 2), BatchNorm
Downsample Block 2	$256 \times 32 \times 32$	$512 \times 16 \times 16$	—”—
Downsample Block 3	$512 \times 16 \times 16$	$1024 \times 8 \times 8$	—”—
Bottleneck			
Bottleneck Block	$1024 \times 8 \times 8$	$1024 \times 8 \times 8$	Main path: Conv2D(3, 1), ReLU, BatchNorm, Time Embedding, Conv2D(3, 1), ReLU, BatchNorm Residual Path: Conv2D(1, 1), BatchNorm
Upsampling Blocks			
Upsample Block 1	$2048 \times 8 \times 8$	$512 \times 16 \times 16$	Main path: TransConv2D(4, 2), ReLU, BatchNorm, Time Embedding, Conv2D(3, 1), ReLU, BatchNorm Residual Path: UpsampleConv, Conv2D(1, 1), BatchNorm
Upsample Block 2	$1024 \times 16 \times 16$	$256 \times 32 \times 32$	—”—
Upsample Block 3	$512 \times 32 \times 32$	$128 \times 64 \times 64$	—”—
Output Layer			
Output Conv	$128 \times 64 \times 64$	$1 \times 64 \times 64$	Conv2D(1,1)

Table 5: **CNN256** architecture.

Layer Type	Input Shape	Output Shape	Details
Initial Conv	$1 \times 256 \times 256$	$128 \times 256 \times 256$	Conv2D(kernel=3, stride=1)
Downsampling Blocks			
Downsample Block 1	$128 \times 256 \times 256$	$128 \times 128 \times 128$	Conv2D(4, 2), ReLU, BatchNorm, Time Embedding, Conv2D(3, 1), ReLU, BatchNorm
Downsample Block 2	$128 \times 128 \times 128$	$128 \times 64 \times 64$	—"—
Downsample Block 3	$128 \times 64 \times 64$	$256 \times 32 \times 32$	—"—
Downsample Block 4	$256 \times 32 \times 32$	$512 \times 16 \times 16$	—"—
Downsample Block 5	$512 \times 16 \times 16$	$512 \times 8 \times 8$	—"—
Bottleneck			
Bottleneck Block	$512 \times 8 \times 8$	$512 \times 8 \times 8$	Conv2D(3, 1), ReLU, BatchNorm, Time Embedding, Conv2D(3, 1), ReLU, BatchNorm
Upsampling Blocks			
Upsample Block 1	$1024 \times 8 \times 8$	$512 \times 16 \times 16$	TransConv2D(4, 2), ReLU, BatchNorm, Time Embedding, Conv2D(3, 1), ReLU, BatchNorm
Upsample Block 2	$1024 \times 16 \times 16$	$256 \times 32 \times 32$	—"—
Upsample Block 3	$512 \times 32 \times 32$	$128 \times 64 \times 64$	—"—
Upsample Block 4	$256 \times 64 \times 64$	$128 \times 128 \times 128$	—"—
Upsample Block 5	$256 \times 128 \times 128$	$128 \times 256 \times 256$	—"—
Output Layer			
Output Conv	$128 \times 256 \times 256$	$1 \times 256 \times 256$	Conv2D(1, 1)

7.3 Training process

Figure 22 shows the loss function over the epochs for **CNN64-AB**. For reference, a training run like this (300 epochs) takes around 16 hours on the high performance cluster used in this project.

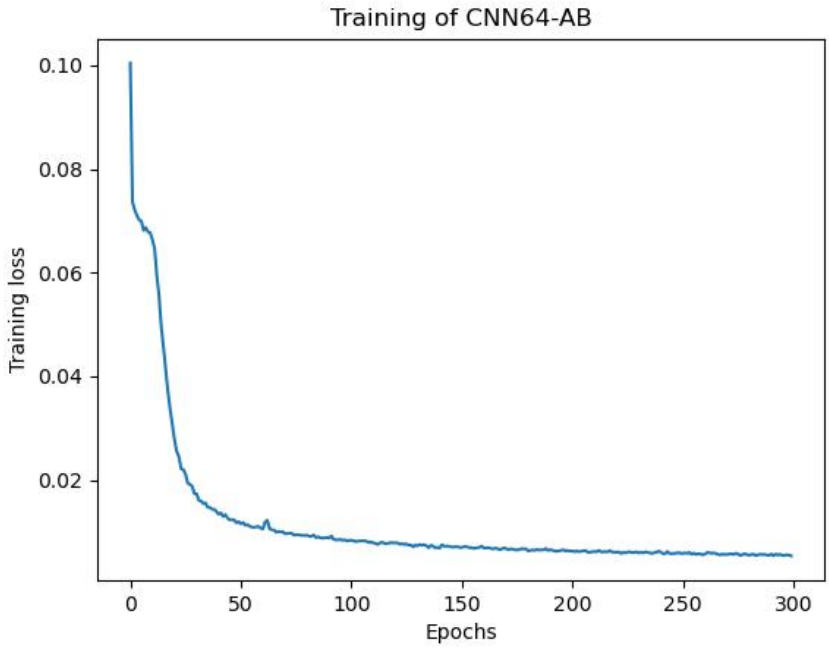


Figure 22: Training of **CNN64-AB**

7.4 Classification network

The architecture of the image classification network is described in table 6.

Table 6: Network architecture of the image classifier for 64×64 resolution

Layer	**Type**	**Parameters**
Conv1	Convolution (2D)	Kernel: 3×3, Padding: 1, Channels: $1 \to 32$
BN1	BatchNorm2D	Channels: 32
Pool1	MaxPool2D	Kernel: 2×2, Stride: 2
Conv2	Convolution (2D)	Kernel: 3×3, Padding: 1, Channels: $32 \to 64$
BN2	BatchNorm2D	Channels: 64
Pool2	MaxPool2D	Kernel: 2×2, Stride: 2
Conv3	Convolution (2D)	Kernel: 3×3, Padding: 1, Channels: $64 \to 128$
BN3	BatchNorm2D	Channels: 128
Pool3	MaxPool2D	Kernel: 2×2, Stride: 2
Conv4	Convolution (2D)	Kernel: 3×3, Padding: 1, Channels: $128 \to 256$
BN4	BatchNorm2D	Channels: 256
Global Pool	AvgPool2D	Global Average Pooling
Flatten	Flatten	-
FC1	Fully Connected	Input: 256, Output: 256, Dropout: $p = 0.5$
FC2	Fully Connected	Input: 256, Output: 3

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. URL: `https://ieeexplore.ieee.org/document/7780459`.

[2] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL: `http://arxiv.org/abs/1412.6980`.

[4] Scikit learn Developers. Confusion matrix, 2024. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html`.

[5] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8162–8171. PMLR, 18–24 Jul 2021. URL: `https://proceedings.mlr.press/v139/nichol21a.html`.

[6] Karin Pagels. A study of generative adversarial networks with applications to paperboard surfaces. *Stockholms universitet, Matematiska institutionen, Matematisk statistik*, 2023. URL: https://kurser.math.su.se/pluginfile.php/20130/mod_folder/content/0/Master/2023/2023_6_report.pdf.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL: https://arxiv.org/abs/1912.01703.

[8] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022. URL: https://arxiv.org/abs/2204.06125.

[9] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, June 2022.

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[11] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France, 07–09 Jul 2015. PMLR. URL: https://proceedings.mlr.press/v37/sohl-dickstein15.html.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Master's Theses in Mathematical Sciences 2025:E5
ISSN 1404-6342

LUTFMS-3510-2025

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/