university of groningen

faculty of science and engineering

Introspective Energy Models: Outlier detection by failure of inpainting

Lukas Kinder

university of groningen

faculty of science and engineering

University of Groningen

Introspective Energy Models: Outlier detection by failure of inpainting

Master's Thesis

To fulfill the requirements for the degree of
Master of Science in Artificial Intelligence
at University of Groningen under the supervision of
prof. dr. Bart Verheij (Artificial Intelligence — Bernoulli Institute)
and
prof. dr. Marco Grzegorczyk (Stochastic Studies and Statistics — Bernoulli Institute)

Lukas Kinder (s3686566)

July 29, 2023

Contents

Abstract

Detecting outliers in images is a challenging task for deep-learning models, if the training data contains little or no examples of outliers. Generative adversarial neural networks (GANs) or autoencoders can be used for this purpose, but usually require a lot of training data. In the MVTec dataset, pure deep-learning models are outperformed by distance based models, which identify outliers based on their high distance from samples in the training set. However, a problem with them is that they rely on pretrained convolutional kernels and are not very explainable. This thesis presents the idea of an introspective energy model, that measures the success of inpainting to detect local anomalies in images. The approach involves a two-stage process. First, convolution is applied to generate feature maps of the image. Subsequently, the features of image regions are predicted using the features of surrounding regions. Inaccurate predictions indicate an outlier. This mechanism initially used Bayesian networks but was later refined using neural networks. The results demonstrate that introspective energy models can outperform the state of the art for certain object categories of the MVTec dataset. In a second experiment convolutional kernels pretrained on the ImageNet dataset were used in an attempt to improve the model further. However, in this case the model performed worse than the state of the art. This work is relevant because it uses a new mechanism to train convolutional layers one by one. The model can be trained with little training data and can explain which regions of an image causes it to be classified as an outlier.

1 Introduction

Models performing semi-supervised outlier detection aim to discriminate outliers based on the normal instances in their training data. This category of outlier detection can for example appear in industrial production, in which the manufacturing process is so optimized that the amount of defect instances is very small, compared to the defect-free ones. Even in a task in which there are some outliers that can be used to train a model, it is commonly the case that outliers are diverse and the examples for them may not be representative for all possible outliers that could occur.

Energy base models (EBMs) can be used for this task. They assign a low energy to an input, if the input is likely based on some learned distribution and they map an input to a high energy, if it is unlikely. State of the art energy based models for image data are for example deep convolutional models with a single unit in the output-layer that describes the energy [1, 2]. The problem with these models is that they require suitable high energy examples during training. It is usually the case that low-energy images lay on a very thin manifold and there are much more possible high-energy than low-energy samples in the image space. Therefore, it is usually not enough to present the models with random-noise images as examples with high energy. It is possible to generate more useful high energy examples using sampling with the current model. However, this is computational expensive and makes them difficult to train [3, 4]. In a similar fashion, generative adversarial networks (GANs) may also be used to train an energy based model. In this architecture there are two models, a discriminator model and a generator model. The task of the discriminator is to predict the energy of a presented sample and the task of the generator is to generate high energy examples [5]. In this case the high energy samples are created using one forward pass of the generator network, which makes the process more efficient. However, GANs usually need a lot of training examples and it is often difficult to ensure stable convergence [6].

Alternatively, there are a few types of EBMs that detect high energies indirectly and do not require any high energy examples during training. A straight forward method of determining the energy of a sample is to use neighbour distances between an instance and its closest neighbours in the training-set. A short distance means a low energy. This can work well if all low energy examples look very similar, but won't work well for more complex problems. A possible method to improve energy based models using neighbour distance is to first map them into lower dimensional space with the means of a pretrained neural network [7, 8]. Models using neighbour distance as a measure of energy perform very well in MVTec dataset [9] for outlier detection [10].

Finally, autoencoders may be used as energy based models as well. Autoencoders are neural networks that are trained to reconstruct their input. This can be a hard task because they usually have layers with a small amount of units as one of the hidden layers which acts as an information bottleneck. A compressed representation of the input must be learned to be able to reconstruct it. After training an autoencoder with a training-set of low-energy images, an energy is implicitly determined using the reconstruction loss [11, 12, 13]. In this case it is a common problem that the autoencoder is able to generalize too well and is able to reconstruct instances that should have a high energy.

This thesis presents an energy based model called introspective energy model (IEM). This model is systematically predicting each part of an instance based on the surrounding. The overall idea is to assign high energies to samples if there is a mismatch between the model's prediction and what is actually there. This is not only done for all regions of a sample but also over multiple levels of ab-

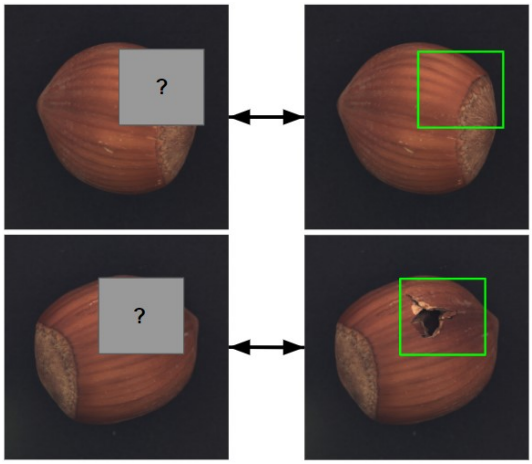


Figure 1: Two images of nuts, one normal and one with a defect. Predicting a normal region of an image should be easier than predicting a region with a defect.

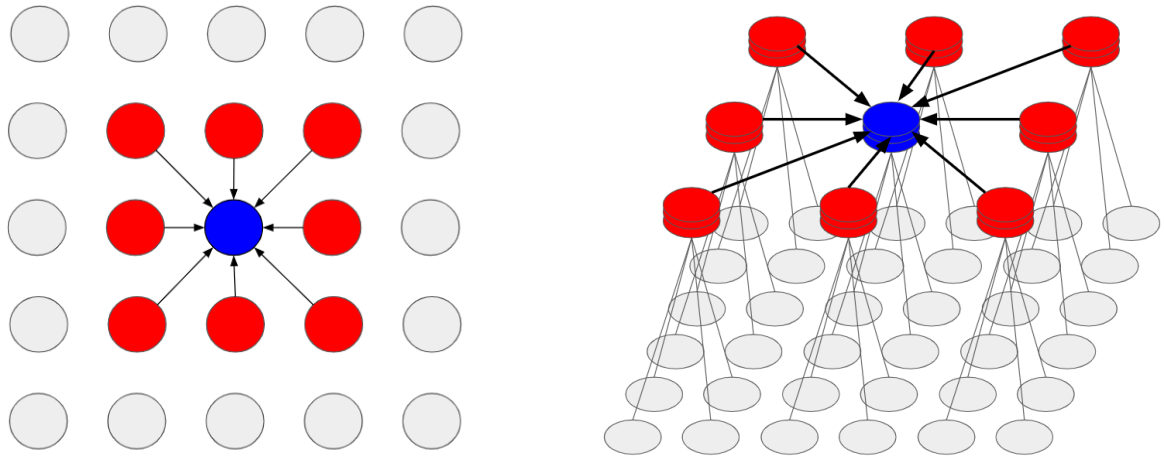
stractions all at once.

In many cases, image regions of a normal image should be easily predictable based on the surrounding image regions. On the other hand, an anomaly located somewhere on the image can not be predicted based on the surrounding image regions. Consider for example the task of predicting the masked regions in Figure 1. Assuming the model predicting the image region was trained well, the predicted and actual region should be very similar for the normal nut. If there is a damage located at the masked image region, the predicted and actual pixels are probably very different.

In an Introspective Energy Model, every image region is predicted using the surrounding. This way an energy map can be computed over the image showing where in the image an anomaly is located. A high energy corresponds to a big mismatch between what is predicted and what is there. This energy map is not only computed on the pixel level but also over multiple layers of higher level representation. These higher level representations are found through convolutional kernels. Higher level representations allow to exchange information over larger distances withing an image. By combining and summing up these energy maps it is possible to assign a hole image a single energy. This is different to an ”introspective neural network” as described by [14] in which a network is able of self-evaluation.

An easy method is a model that can predict the value of a pixel based on the surrounding 8 pixels. For some images it may be the case that the value of a pixel is independent of its position within the image given the surrounding pixels (often the case for textures or patterns). In this case the same model may be used for all pixels. If there is reason to believe that this assumption does not hold, a different models should be used for each pixel of the image. This method may work well for some problems, but long distance dependencies can not be modeled. For example, consider such a model trained on the handwritten digit dataset MNIST [15]. The problem is that there are some logical long-distance relationships. For example: If there is a loop in the upper part of the image then the lower part of the image is either part of an ”8” or ”9”. This type of dependencies can not be modeled by just predicting a pixel based on its immediate neighbours.

In order to capture long distance relationships, convolution can be used to get a map of higher level features of the image. These higher level features are used to predict each other over larger distances.



(a) Layer 0 / pixel representation.

(b) Layer 1

Figure 2: A visual representation of the sideward pass of nodes in different layers. **(a)**: An indication of a sideward pass of a single node in layer 0. Nodes represent the pixels. The blue node is predicted with a model that takes the neighbouring nodes (red) as an input. **(b)**: An indication of a sideward pass at a position in layer 1. The gray nodes represent the pixels in layer 0. The red and blue nodes encode the higher level representations that are generated using three convolutional kernels with size 2 by 2 and a stride of 2. Thin gray lines are used to indicate which image region maps to which nodes in the next layer. The blue nodes are predicted with a model that takes the neighbouring nodes of the same layer (red) as an input.

For example, a set of convolutional kernel of size $2 * 2$ generate super-features that correspond to a 4 pixel region within the image. Afterwards, a model is used to predict the values of the super-features at some position using the values of the surrounding super-features. See Figure 2 for a visualization of this.

We will refer to the process of determining the higher level features using convolution the *forward-pass* and the process of predicting the value of a node using its neighbours the *sideward-pass*.

This can be repeated over multiple layers. A 2D view of this process over multiple layers is shown in Figure 3. In general, it should not be the case that the values of a node, as determined by convolution, depend on the same pixels as their neighbours. This would enable the sideward-pass to reconstruct the value of a node directly without having to generalize. However, it may be fine if there is a small overlap in their receptive field.

This thesis discusses three experiments in which different versions of an introspective energy model are utilized.

- In an first experiment conditional probability tables were used for the sideward-pass. This means that the pixels and features generated through convolution are treated as variables of a Bayesian network. This thesis discusses algorithms to find suitable relations between variables and how the convolutional kernels can be trained in this case. After being trained with the MNIST dataset [15] the model is used for a semi-supervised outlier task. The results may be seen as a proof of concept but are not compared to state of the art methods, because training the model is too computational expensive to use for bigger datasets. The model was also not

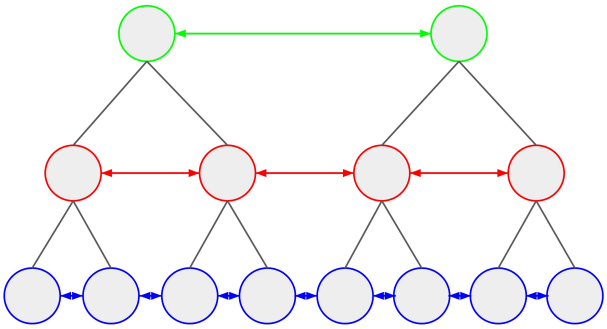


Figure 3: A view of an Introspective energy model from the side. The blue nodes represent the pixels, the red ones the second layer and the green ones the third layer. The activations of the higher layers are determined using convolution. This process is indicated with gray lines. The horizontal arrows indicate the sideward pass in which the activation of a node is predicted using its neighbours. A neighbour of a node is connected via a horizontal arrow and its value is determined using convolution on a completely different region of pixels.

able to create Gibbs samples that look like MNIST-images, suggesting that it failed to learn the distribution of the dataset well.

- In a second experiments, the sideward-pass was performed using small neural networks. It is discussed how to train the model using gradient descent and how extensive weight sharing can reduce the need for a lot of training data. With these algorithms, the model was able to outperform the state of the art in some object categories of the MVTec dataset [9].

- Lastly, an experiment investigates if the pretrained kernels of the AlexNet model [16] can be used to improve the performance. In this case, only the parameters performing the sideward-pass needed to be trained. This was improving the performance, but the model lacks behind compared to other models using transfer learning.

Introspective energy models have the property that they can be trained layer by layer, which makes learning very stable. They can also be implemented with a large amount of weight sharing which makes them applicable, even if there is little data. The outputs of an IEM are also relatively explainable, because the energy of an input can be attributed to a region of the input.

After going over the relevant background in Section 2, the main part of this thesis discusses three experiments. The first experiment in Section 3 discusses how to use Bayesian networks with hidden variables for the MNIST dataset. Afterwards IEMs are trained for the MVTec dataset from scratch in Section 4 and alternatively with pretrained kernels in Section 5.

2 Background

Section 2.1 elaborates Bayesian Networks for discrete data, in particular how previous work incorporated hidden variables. Hidden variables are important for image tasks, because of the emergent properties of pixels. Understanding introspective energy models requires a good understanding of how convolution generates higher level features of an image. Therefore, convolution as used in deep neural networks is discussed in section 2.2. Afterwards, section 2.3 explains different existing architectures for energy based models. Introspective energy models can be seen as a form of self-supervised learning, which is discussed in section 2.4.

2.1 Bayesian Networks

Bayesian networks were initially introduced by [17]. They model the joint probability distribution of a set of random variables $X_1, ..., X_n$ by using their local dependencies. A Bayesian network $\langle \mathcal{G}, \Theta \rangle$ consists of an acyclic directed graph $\mathcal{G}$ as well as a set of parameters Θ. The nodes in $\mathcal{G}$ represent variables that each have a domain of discrete states. The parameters Θ are used for the conditional probability tables (CPT) associated to each variable, that give the probability of the variables states, given the state of their parents. The state x_i of a variable X_i is independent of all its non-decedents given its parents (local Markov property). This allows to calculate the joint probability by taking the product of each variables states probability given its parents:

$$P(x_1, ..., x_n) = \prod_{i=1}^{n} P(x_i | Parents(X_i)) \tag{1}$$

The number of parameters in Θ needed is linear to the number of variables n but exponential to the maximum amount of parents k of a variable. This is because the number of rows needed for the CPT of a variable is equal to the number of unique combination of parent states.

Bayesian networks can predict the posterior probability of any set of variables, given the state of some other set of variables as evidence. However, inference in Bayesian networks is an NP-hard problem [18].

2.1.1 Training Bayesian networks

Training a Bayesian network requires some data that contains samples of the variables. When training a Bayesian network it might be that the graph $\mathcal{G}$ is already given or that the parameters as well as the structure of the graph need to be learned.

Ideally, expert knowledge can be used to determine the dependencies between variables. This means that only the parameters Θ need to be learned. This can be done simply by determining the probability of a variable X_i having state x_i given that its parents P have state p with:

$$P(X_i = x_i | P = p) = \frac{\#(X_i = x_i, P = p) + \lambda}{\#(P = p) + n\lambda} \tag{2}$$

Here $\#(\varphi)$ denotes the number of samples in the data that satisfy φ, n is the size of Xs domain and λ is the Laplace correlation. With $\lambda = 0$ we have the maximum-likelihood estimation. The maximum-likelihood estimation can assign a zero-probability to some states and may overfit if the data is not

perfectly representative. Common choices for λ are 1 or 0.5 [19].

If the dependencies between variables are not known, it is common to find a suitable graph using local search. This means taking a (random) starting point and adding/removing dependencies one by one. The local search should optimize some scoring function that measures the goodness of the model. A commonly used search method is simulated annealing [20]. Local search does not guaranteed to find the optimal model.

A high scoring model B should maximise the likelihood of the data D given the model: $P(D|B) = \prod_{x \in D} P(x|B)$. However, it should also not overfit. Therefore, a regulation term can be introduced that deducts from the score the more parameters are used.

A commonly used scoring function is the Bayesian information criterion (BIC):

$$BIC = k * ln(n) - 2 * ln(P(D|B^*)) \tag{3}$$

Here, k is the number of parameters used by the model, n is the number of samples in data D and B^* is the Bayesian network with maximum-likelihood parameter estimation. In an empirical evaluation by [21] it was reported that the BIC outperformed all other scoring functions.

Structure learning with many variables (hundreds or thousands) struggles based on the large search space of potential graphs. A method to reduce the search space is to initially reduce the number of possible parents of each node to a small candidate set [22].

2.1.2 Incorperating hidden variables in Bayesian networks

A hidden variable is a variable with unknown states for the samples in the training data. It is possible to just ignore them by removing them from the graph and adding a dependency from every parent of the hidden variable to all of its children. However, incorporating hidden variables can be useful in order to reduce the number of dependencies. An example that shows how a hidden variable can reduce the number of dependencies is shown in Figure 4. With reduced amounts of dependencies there are less parameters to learn which reduces the risk of overfitting.

A common task is to have a variable with a known position in the graph, but its states for the samples in the data is hidden. In this case the Gibbs sampling or the EM-algorithm can be for example used to find its states and the values in the CPT [18] [24].

In Gibbs sampling the states for unobserved variables are initially randomly chosen for each sample in the data D. Afterwards, the state of an unobserved variable in the data x_{il} (the state of variable X_i in sample l) is chosen and re-sampled based on the probability of the potential new resulting data D', given the structure of the graph. This step is repeated over a large amount of iterations. At the end, the data is assumed to be the average during these iterations.

In the EM-algorithm the parameters in the CPTs are initialized randomly. Afterwards, the expected data values of the hidden variables in the data are approximated using these parameters. This is the expectation step. Afterwards, during the maximisation step, the parameters are chosen to maximise the likelihood of the expected data. The expectation and maximisation step are repeated until convergence. The EM-algorithm is computationally cheaper than Gibbs sampling but is not guaranteed to

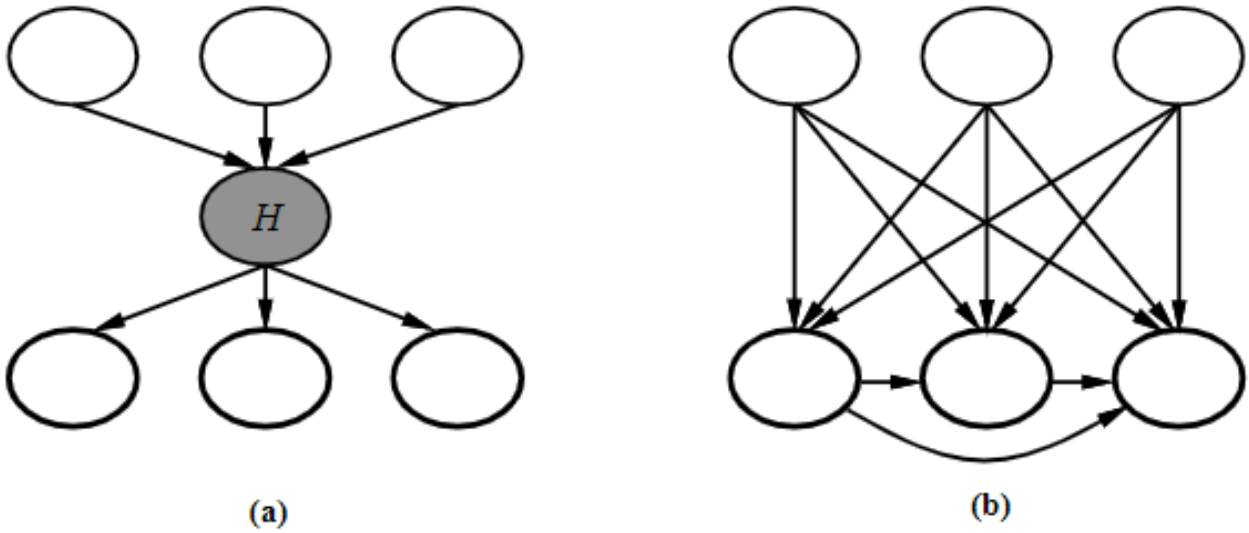


Figure 4: An example showing how hidden variables can simplify Bayesian networks. The number of relations become smaller with a hidden variable. Figure was taken from [23].

converge to the best possible solution [18].

There is much less work done in incorporating hidden variables in Bayesian networks if the position in the graph of the hidden variable is unknown [23]. This task is complicated because the search space of possible graphs needs to be expanded over a large set in which not only dependencies but also hidden variables are introduced. The number of graphs that can be constructed from n variables is super-exponential [18]. However, it is possible to reduce the search space by only considering graphs with a certain characteristic [25].

There has been some work done in finding single hidden variables using semi-cliques to help to guide the search [26][27]. A semi-clique is a group of nodes that have a lot of dependencies with each other (based on structure search using the data without hidden variables), which could indicate the presence of a hidden variable.

Learning a Bayesian network with large amounts of hidden variables with unknown structure is struggling with a large search space. A lot of recent real-world tasks that incorporate hidden variables in Bayesian networks did not use more than 3 hidden variables [28], [29], [30].

In a paper by [31] a model called ”hierarchical Bayesian network” (HBN) is described. Here, hidden variables are structured in layers such that the variables in upper layers represent a more compressed version of the data. Dependencies are going from each hidden variable in the direction of two variables in the layer below, as well as between variables in the same layer.

Their learning algorithm for HBN is as follows:

- The observed variables are paired, such that the pairs have a high mutual information.
- For each pair a hidden variable is introduced that has a dependency to both. The two most common value-combinations in the data of each pair is found. The hidden variable for a pair is

assign value 0 for samples in the data in which the pair has its most common value-combination and is assigned value 1 for samples in the data in which the pair has its second most common value-combination. Otherwise the value for the hidden variable is considered missing for a sample.

- The EM-algorithm is used to find the missing values in the hidden variables and the parameters that model the relationship between the pairs and the hidden variable.
- Additional layers of hidden variables are introduced, again having two children in the layer below each.
- Dependencies within layers are added using traditional structure learning.

This method enabled them to incorporate many hidden variables in a Bayesian network. However, they were only able to use it as a data analysis tool and not to improve the joint probability the Bayesian network assigns to inputs. They also do not utilize spacial structure in the data, like the distance between pixels in images.

2.2 Convolution

Convolution is a widely used method for deep-learning models [32]. It is primarily used for image tasks like image classification, object detection, image recognition and segmentation [33][34][35][36]. Convolution can also be used for 1-dimensional data like audio or temporal diagnostic data [37] or for 3-dimensional data like videos or point clouds [38][39].

Convolutional kernels for 2-dimensional data were first introduced by [40] as early as 1988. LeCun was the first using the term ”convolution” for a handwritten recognition task in 1989 [41].

The basic idea for convolution is to have a small feature detector called kernel that slides over the input. This way the output nodes can be organized as a feature-map. A kernel is usually just a matrix of numbers which are multiplied with their corresponding elements in the input and summed afterwards. The advantage of this is that features can be extracted while keeping information about their location and only a little amount of parameters is needed. Multiple kernels may be used for different types of features. Figure 5 shows the basic concept of applying a convolutional kernel on a 2-dimensional input. Convolution can also be performed sequentially over multiple layers. In this case the input of the next convolution is the output of the previous. This way, more and more refined features can be extracted.

In convolution ”padding” means adding values on the boarder of an input (usually the value 0). This makes it possible to also extract features right at the image boarders. A ”stride” is the amount of pixels a kernel is moved each step. This is a common method to make the feature map smaller which also reduces the number of parameters needed for consecutive layers and makes the process computational cheaper. A ”receptive field” of a node is the region in the original input that influenced its value. For example, in Figure 5 the receptive field of a node after the max pooling operation is a 5 by 5 region in the input.

Convolutional kernels can often be used for transfer learning [42]. This means, using the trained kernels from a model of an old task for a new task. This is for example useful if there few training

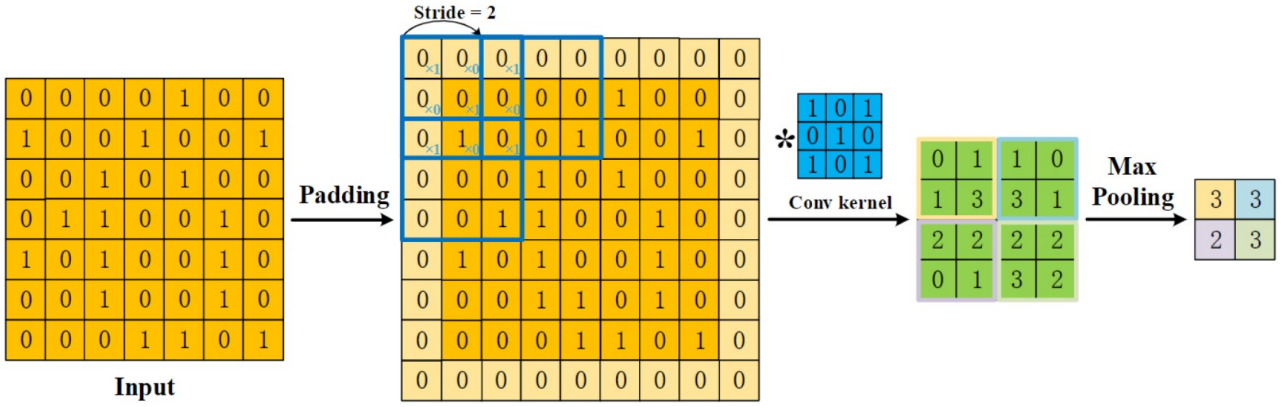


Figure 5: An example of applying a three by three convolutional kernel on an input. Initially the input is padded by extending it with zeros on all sides. Afterwards, the convolutional kernel is moved over the input in steps of two. An additional max-pooling operation is performed in which a four by four region is replaced by their max value. Figure was taken from [32].

data and the feature-maps created by the pretrained kernels are useful for the new task. Convolutional models pretrained for classification on large dataset are commonly used for transfer learning [43, 44, 45]. In particular the ImageNet dataset is used for training which has more than 50.000 citations [46].

2.3 Energy Based Models

An energy model $E_\theta(x) : \mathcal{R}^D \to \mathcal{R}$ models a data distribution by taking a D-dimensional input x and assigning low energies to it, if it is familiar and high energies otherwise. It parametrizes a density $p_\theta(x)$ as [47]:

$$p_\theta(x) = \frac{e^{-E_\theta(x)}}{\int_x e^{-E_\theta(x)}} \tag{4}$$

Here θ is the collection of the models parameters.

It is possible to map this energy to a probability. However, this requires to find the normalization constant $\int_x e^{-E_\theta(x)}$ which is often impossible if the state space is too large. Nevertheless, the energy can be used to reason about the input and to accomplish a variety of tasks.

This thesis only concerns energy based models that take images as input. The following sections describe different types of energy based models. Note that the term ”energy based models” is used broadly as any model that can take an input and maps it to a value that is low if and only if the presented input is likely or normal. This includes the discriminators of generative adversarial networks (GANs) and the loss of autoencoders even though they are not traditionally called ”energy based models”.

2.3.1 Direct Energy based models

The basic type is a (convolutional) neural network with one output unit [1]. The outputs activation is trained to correspond to the energy of the input. They require training-sets that include examples of high and low energies. Their performance can be increased by using pretrained convolutional kernels

[48].

These models must be presented with high-energy examples during training. It is often not enough to just use white noise images for this purpose. This is because the images that should have high energies outnumber the ones with low energies (the "normal" ones). The model needs to learn the very thin manifold on which these low energy instances are located. It is common practice to generate the high energy images using sampling (see section 2.3.5) with the model over multiple iterations. These high energy examples can be put in an replay buffer for the model to train with [2]. However, a problem with this is that it is very computational expensive to generate samples. This also makes the models notoriously difficult to train [3, 4, 2].

Despite these problems they have the advantage that the energy is calculated directly and there is no need to train multiple objects. This can mean that they require less parameters.

2.3.2 Autoencoders

An autoencoder, first introduced by [49], is a model that aims at reconstructing its input. Formally it consists of an encoder $A : \mathbb{R}^p \to \mathbb{R}^k$ and decoder $B : \mathbb{R}^k \to \mathbb{R}^p$. Usually p is (much) bigger than R. The task is to minimize:

$$E_{x \sim P_{data}(x)}[\mathcal{L}(x, B(A(x)))] \tag{5}$$

Here $E_{x \sim P_{data}(x)}[]$ indicates the expectations with samples x taken from the training data and $\mathcal{L}$ is a loss function that measures how well a sample is reconstructed [50].

Energy based models can be (convolutional) autoencoders [11, 12, 13]. A high energy can be associated with a high reconstruction error [51, 52]. These models do not necessarily require high-energy examples in their training-set.

A problem with using the reconstruction error as a measure for energy is that the autoencoder may generalize too well, causing it to reconstruct images well that should have a high energy. To avoid this, the autoencoder can be heavily restricted. For example by having a very low number of units in the latent layer.

Another possible method to improve the model is by performing an inpainting task [53]. For example, the RIAD model [54] is dividing an image x into k by k tiles and a random fraction of these tiles are obstructed to generate an image x'. Afterwards, the image is passed through an autoencoder and the energy is the reconstruction loss with respect to the original unmasked image. This process can be repeated and the energy averaged over multiple runs in which different regions are obstructed. The hope is that in some of these runs a defect region is completely masked which makes it very unlikely that the model has a low error for this region. See Figure 6 for a flowchart of this method.

2.3.3 GANs

Generative adversarial neural networks (GANs) [55, 56] are usually used to generate samples that mimic the instances in their training-set. They are divided into two (convolutional) neural networks, a generator G and a discriminator D. The generator is taking some noise-vector z as an input and is outputting an image. The discriminator has to decide whether a sample comes from the training-set

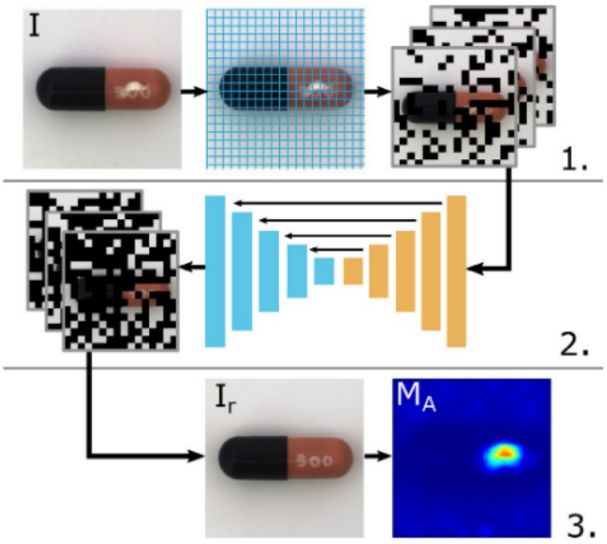


Figure 6: A Flowchart of how the RIAD model detects outliers. Step 1: An image I is divided into k by k squares and a set of images is created in which each instance of the set is the image with different squares being obstructed. Step 2: Each partially masked image is presented to an autoencoder that tries to generate the original image. Step 3: The generated images are combined to an image I_r. A local difference between I_r and I is indicated a defect. Figure as shown by [54].

or was created by the generator. Its output is a value between 0 and 1 indicating the probability that the presented sample comes from the real distribution. Both models are training at the same time with gradient descent with the objective function of a two player minmax game:

$$\max_D \min_G E_{x\sim P_{data}(x)}[log(D(x))] + E_{z\sim P_D(z)}[log(1-D(G(z)))] \tag{6}$$

Here $E_{x\sim P_{data}(x)}[]$ indicates the expectations with samples x taken from the training data and $E_{z\sim P_D(z)}[]$ indicates the expectation with z taken from some noise distribution.

After training, the discriminator can be used to determine the energy of an image as the network is trained to assign low values to images in the training-set and high values to images different to the ones in the training-set [5].

In practice the discriminator of a GANs is usually not performing very well for outlier detection with a small training-set, because they require a lot of training samples and the images that should be assigned high energies may be very different than the ones created by the generator. They also require a careful tuning of hyper-parameters to ensure stable convergence [6].

2.3.4 Distance based models

An easy method to determine if an image is unusual and should therefore have a high energy, is to compare it directly with the normal images in the training-set. This is possible by setting the energy to the distance between the image and the images in the training-set. This can be for example the distance with its closed neighbour, the distance to the k nearest neighbours or the Mahalanobis distance [7, 57].

There are some problems with this. For example, two images from the same object that is slightly rotated or shifted can have a large distance to each other. Or imagine the task of having a set of cat-images in the training-set and the model is supposed to assign high energies to dog-images during testing. Just comparing distance wont work well in this case, because different cat images may look very different and a dog and cat image can also look very similar.

This process can be improved by first mapping the images to a lower dimensional space. This can for example be done with the means of feature bagging, [58], principle component analysis [59], autoencoders [60], SVM [61] or by taking the encoding of a pretrained CNN [10, 8].

A bottleneck of this method can be the time complexity, as calculating the distance for a sample may require a comparison with each training sample. However, patch embedding for localization can be used to make the time complexity independent of the size of the training-set [57].

2.3.5 Generating samples

On possible use for an energy based model is to generate new samples that also have low energies and ideally share properties with the training instances. This is usually done using Markov Chain Monte Carlo sampling (MCMC) [1] like random walk, Gibbs sampling or Langevin sampling. In particular, Langevin sampling is used for quick convergence [62]:

$$x^{k+1} = x^k - \frac{\lambda}{2}\nabla_x E_\theta(x^k) + \omega^k, \omega^k \sim \mathcal{N}(0,\lambda) \tag{7}$$

Here, λ is a parameter that needs to be tuned and $\mathcal{N}(0,\lambda)$ is the normal distribution with mean 0 and standard deviation λ.

The idea is to start with a random image x^0 and iteratively change it in the direction of the gradient in respect to the energy plus some small noise. After enough iterations and a small enough λ the image will converge to a sample of the energy distribution [63]. The same method can also be used for image inpaining. In this case only the missing region of pixels is iterativly updated while the other pixels stay the same [2].

The quality of the generated samples can be evaluated and compared across different models using the fréchet inception distance(FID) [64] or inception score [65]. The inception score utilizes the Inception network which is pre-trained on the ImageNet dataset. Given an image the inception model outputs a vector that maps the image to the ImageNet classes. The idea is that the generated samples should be diverse, that is the entropy of the classes assigned by the inception model for the samples is high. Additionally, the samples should have a high quality. This is evaluated by measuring if the entropy of the classes the inception model assigns to a sample is high. The FID score also utilizes the Inception model and evaluates a set of samples by comparing the distribution of extracted features with the features of real images. The FID score can evaluate samples with respect to any target distribution, not only in respect to the ImageNet dataset.

Note that models using neighbour distance as described in section 2.3.4 can usually not be used to generate new images using MCMC sampling. This is because the samples would converge to an image in the training-set.

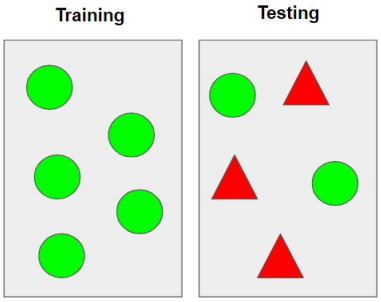


Figure 7: The concept of semi-supervised outlier detection. The training-set only contains normal samples and during testing the outlier need to be separated from the normal.

2.3.6 Outlier detection

Another use for energy based models is outlier detection. After training a model it should map normal images to low energies and abnormal images to high energies. For testing, an energy threshold can be chosen such that all images with a higher energy are labeled as outliers. Dependent on how the threshold is chosen the false positive and false negative rate can variate. A common practice to evaluate such models is by reporting the AUROC curve.

Outlier detection can be categorized into supervised, anomaly detection, semi-supervised anomaly detection and unsupervised anomaly detection [66]. In supervised anomaly detection the training data contains labeled defective and defect-free samples. In unsupervised outlier detection the training samples are not labeled and contain normal as well as outlier images. Semi-supervised learning methods are only provided with defect-free samples during training (see Figure 7).
Energy based models are especially useful because they also work for semi-supervised outlier detection where the training-set contains only normal images and the testing-set contains normal as well as outliers [67]. In this case it would not be possible to have for example a decision tree or a neural network that directly determines if something is an outlier, because the training-set does not contain examples of outliers.

Ideally, a model should also be able to distinguish between logical and structural anomalies. A structural anomaly is an image showing a wrong object. A logical anomaly is a valid object being located at a wrong position. Detecting logical anomalies is often harder [68].

2.4 Self-supervised Learning

A common bottleneck of deep learning models is the amount of labeled training data needed. However, it may be the case that there is more data available which is unlabeled. In this cases it may be beneficial to pretrain a model on tasks that only requires unlabeled data. While doing this the model may learn useful representations of the data. Afterwards, the model can be fine tuned by training for the actual task at hand. This is the idea of self-supervised learning [69, 70]. Self-supervised learning is different to supervised training in which the labels are given. In unsupervised learning, training is done without labeled data. Self-supervised learning can be seen as a category of unsupervised learning in which labels are created from the data to create so called pretext tasks.

A pretext task could for example be to use a convolutional neural network to find the correct rotation

of an image [71]. After the CNN was trained for this using a large set of unlabeled images it may have kernels that specialize on useful features of these images. Therefore, it may be much easier to further train it for other tasks like classification, object recognition etc. that require labeled data.

Another type of pretext task is inpainting [53]. In this case, an image is partially obstructed and the pretext task is to predict the obstructed region.

In recent years contrastive learning gained popularity [72]. In contrastive learning images are augmented and a model is trained to have similar representations for the different augmentations of the same image.

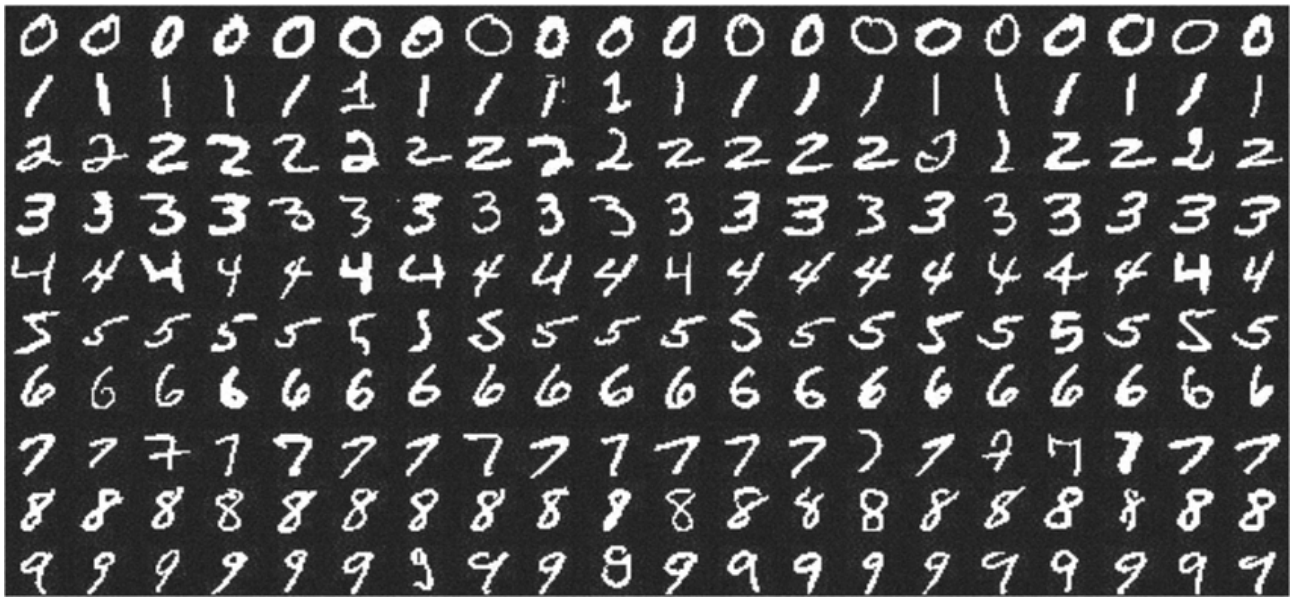


Figure 8: Examples of the handwritten MNIST images.

3 Introspective energy models with Bayesian networks

When developing introspective energy models, conditional probability tables (CPTs) were initially used for the sidewards pass. This method was later discarded in favour of neural networks which performed better. Nevertheless, this section will describe how they were used for the MNIST dataset [15] and the problems with them. The performed experiments will show the success of the model in an semi-supervised outlier task and in sampling instances based on the learned distribution.

The MNIST dataset contains grayscale images sized 28 by 28 pixels of handwritten digits between 0 and 9. The 60.000 training images are labeled with the digit shown on them and the test-set contains 10.000 images. In this experiment we do not care about the labels and ignore them. Examples of MNIST images are shown in Figure 8. In order to use a Bayesian network, pixels need to have discrete values. Therefore, they were binarized by setting their value to 1 or 0 depending on whether their grayscale value is above or below the threshold of 0.3. A threshold of 0.3 was chosen because it made reading the digits the easiest in a human inspection.

The following two subsections either one (just pixel representation) or multiple layers were used and problems with both methods are discussed.

3.1 Experiment without using high level features

As a first experiment only one layer is used, which means that each node in the layer corresponds to a single pixel and there is no convolution. In order to avoid having to search for a structure, hard-coded relations are used. The parents of a node are the nodes of the 4 pixels located north-west, north, north-east and west as shown in Figure 9. This is done, because the value of a pixel is probably a good predictor for its neighbouring pixels. It also ensures that the Bayesian network is acyclic and that each node has the same amount of parents. This is important as the number of data required grows exponentially with the number of parents. Each node is associated with a conditional probability table that predicts the probability of it having the value 0 or 1. This table has $2^4 = 16$ rows for each

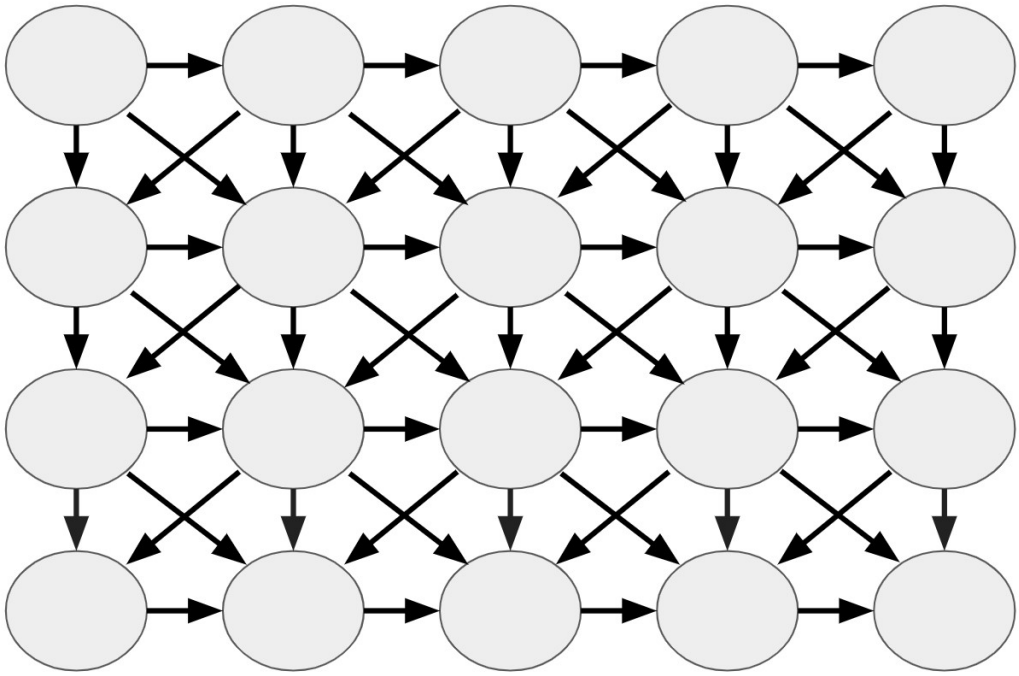


Figure 9: A structure for a Bayesian network for images. Nodes represent pixels. To keep the Bayesian network acyclic, parents are only located north-west, north, north-east and west of a node. Long-distance relations are not present.

unique parent configuration. Since MNIST has a set of 60.000 training images there are in average 3750 data-points per row. However, some configurations are of course much more likely or unlikely. Pseudo-counts of 1 were used for each row, since the data is limited and not completely representative.

After fitting the model to the data, it was tasked to generate new images of the same distribution as the training images. For this purpose, Gibbs sampling[73] was used as described in Algorithm 1. The idea is to repeatedly change a pixel and accepting this permutation according to the probability of the state before and after the change. In theory, as long as there are no states that are assigned zero probability and each variable is chosen infinitely often, the samples are independent and identically distributed samples of the Bayesian network [18]. In practice, good results can be achieved by having enough iterations between samples. For the MNIST samples, 100.000 iterations were used.

Algorithm 1 The pseudo-code for Gibbs sampling.

procedure GIBBSSAMPLING($bn, N, iterations\_between$)
  input: a Bayesian network bn, $N \in \mathbb{R}, iterations\_between \in \mathbb{R}$
  set the state of bn to a random image X
  for $1, ..., N$ **do**
    for $1, ..., iterations\_between$ **do**
      $X' \leftarrow X$ with a random pixel changed
      $X \leftarrow X'$ with probability $\frac{P(X'|bn)}{P(X|bn)+P(X'|bn)}$
    add X to *samples*
  return: *samples*

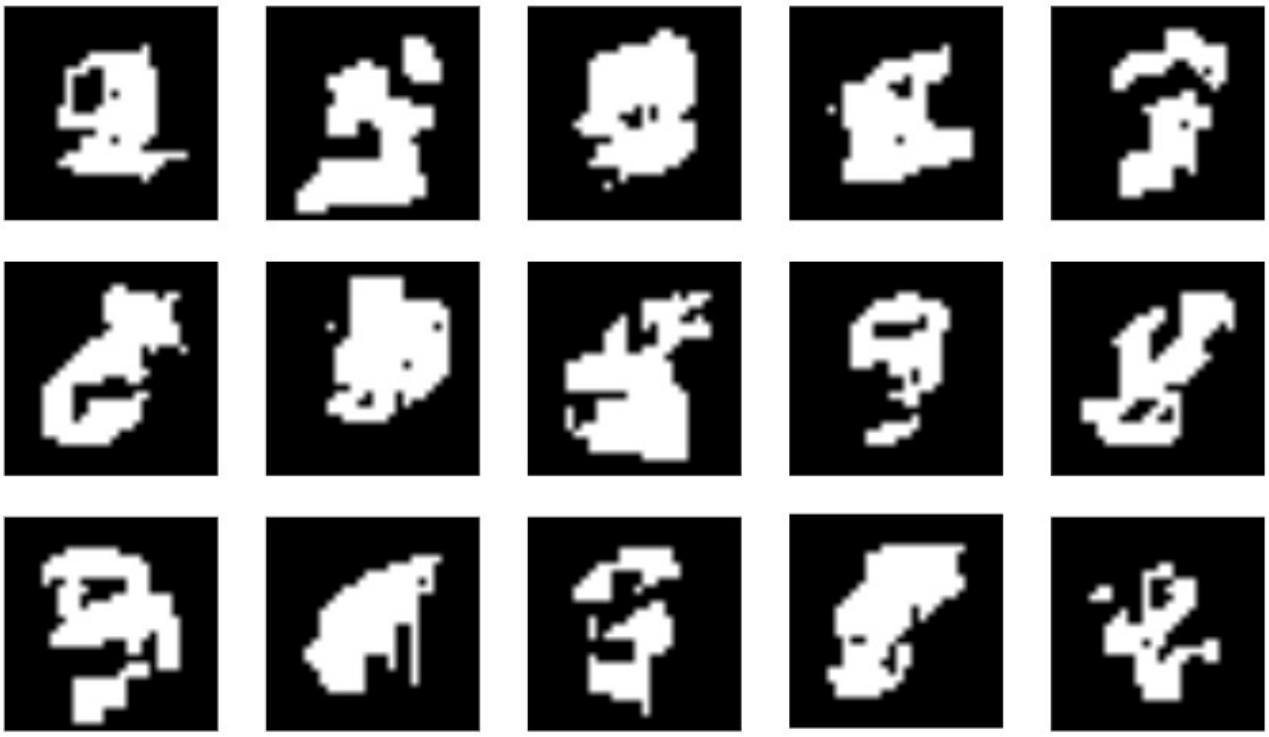


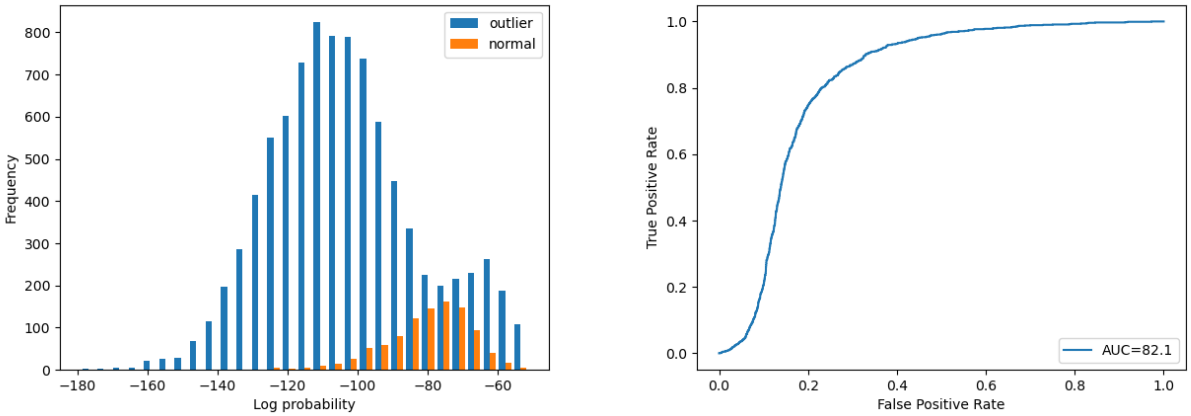
Figure 10: Samples generating using Gibbs sampling of a simple Bayesian network trained with the MNIST dataset.

The generated samples are shown in Figure 10. The model did learn that the pixels in the corners and boarders are usually black and it sometimes generates little lines or curves. However, they clearly do not look like handwritten digits.

In another experiment the Bayesian network was used for semi-supervised outlier detection. It was trained with images that only show one number. This means that instead of the 60.000 training images in the MNISt training-set it was only training with about 6.000 of them which show a specific number. During testing, it had to differentiate between images showing the number in the training-set and the other images (the ”outliers”). This type of task mimics a situation in which there is only one type of normal but a diverse set of outliers. The models find outliers by calculating the probability of each image with the hope that outliers have an especially low probability.

An example of the energy distribution of the outliers and normal images in the testing-set is shown in Figure 11a. A method to measure the goodness of a model performing outlier detection is the area under the receiver operating characteristic curve (AUROC), which is shown in Figure 11b. A %-area of 50 would be random guessing. The results for runs in which different numbers were in the training-set are shown in Table 1. The model performed best if it trained with images showing a 1 with an AUROC of 99.9. This makes sense because the centered line of a 1 is not commonly found in other images. After training with the number 8 the model performed performed the worse with an AUROC of only 76.6. This is probably because the lines that make up an 8 also exist in many other digits. The average AUROC for the ten runs was 89.4.

The results for this task of semi-supervised outlier detection shows a proof of concept, but is not a serious competitor with the state of the art. This specific type of semi-supervised outlier detection



(a) The probability for normal images showing the number two and outlier images that show other numbers.

(b) The resulting ROC curve. The false positive and false negative rate differs based on what threshold is chosen. The %-area under the ROC curve (AUROC) is 82.1.

Figure 11: An example of an probability distribution for normal and outliers images and the resulting ROC. The Bayesian network was trained with images showing the number two.

	0	1	2	3	4	5	6	7	8	9	Average
AUROC	94.9	99.9	82.1	84.2	85.0	89.2	95.1	94.8	76.6	92.5	89.4

Table 1: The AUROC scores for an semis-supervised outlier detection task using MNISt images. In each of the ten experiments the model was trained to consider one of the numbers as normal and the other abnormal. The introspective energy model used conditional probability tables for the sidewards-pass and used 1 layer.

it too unique of a task to compare it with any other models directly. It would be great to use this Bayesian network for semi-supervise outlier tasks that make it possible to compare it with other models. However, benchmarks that are used in recent years, like the MVTec dataset [9], consist of large rgb-images. Unfortunately, the Bayesian network would have the disadvantage that it needs change the pixel values to discrete states and it would also have a very large number of parameters, duo to the image size. This makes it impractical to train for such a task.

3.2 Incorporating higher level features in Bayesian networks for image data

The Bayesian network as implemented in the previous section was not able to generate realistic samples of the data. A problem may be the fact, that relations are only between neighbouring pixels. This may prevent the model to effectively exchange information over large distances within the image. For example, a line that is used to draw a digit is usually about 5 pixels wide. However, a pixel positioned somewhere within a line does not know where exactly in the line it is positioned. The CPT of the pixel only contains information about close neighbours and does not receive information from 5 pixels away. A possible solution for this is to add relations between pixels over large distances. However, this is problematic because there can not be too many relations, otherwise the CPTs become too large to be trained with the limited data.

Therefore, hidden variables may be used to solve this issue. A hidden variable could be a feature of an image region. These hidden variables can have relations with other hidden variables, allowing them to exchange information over large distances within the image. However, there are probably a lot of hidden variables needed, which rules out traditional methods used to find hidden variables in Bayesian networks. There is however a very effective method used by deep learning to extract features from an image: convolution. The new layer of hidden nodes would need to have discrete states. This can be achieved by performing convolution and thresholding the activations afterwards. Again, in order to avoid a computational expensive search it is beneficial to restrict the relations as follows:

- Relations between nodes should only exist within the same layer.
- Incoming relations should only come from north-west, north, north-east and west.
- Nodes that correspond to the same kernel must have incoming relations from the same directions.
- The receptive fields of nodes having a relation should not overlap.
- The receptive fields of nodes having a relation should be next to each other.

The first three restrictions are design choices to keep the model simple and to reduce the search space of possible relations. The reason for the third point is that nodes of the same convolutional kernel correspond to the same type of super-feature and may therefore be best predicted by the same type of surrounding super-features. The fourth point ensures that hidden variables can exchange information over large distances. The idea of the last point is that hidden variables that are features of neighbouring image regions are probably good in predicting each other. Information over even larger distances can be exchanged by adding additional layers (using convolution) with nodes having bigger receptive fields. Note that this limits the maximum amount of convolutional layers, since a node having a receptive field that is larger than half of the image can not have any relations with other nodes.

Even with these restrictions, it is not possible to add every possible relation if the number of kernels is high. Since relations can come from 5 directions, the number of possible incoming relations of a node is $5n$, with n being the number of convolutional kernels used. Structure search can be used to find suitable relations within the candidate-set. Relations that decrease the BIC the most are interactively added until there is no relation anymore that can decrease the BIC. Note that this method usually gives a good solution but may only find a local maximum [22].

The convolutional kernels need to be trained as well. Ideally, the nodes with values determined by a kernel are useful to predict other nodes that correspond to neighbouring image regions. Formally, given a Bayesian network BN we want to change the parameters θ of the kernels to minimize the following loss:

$$\mathcal{L}(\theta) = L(D|BN) - L(D|BN') \tag{8}$$

Here, $L(D|BN)$ is the likelihood of the data D given a Bayesian network BN that was trained on the data using maximum-likelihood estimation. BN' is the Bayesian network without any relations.

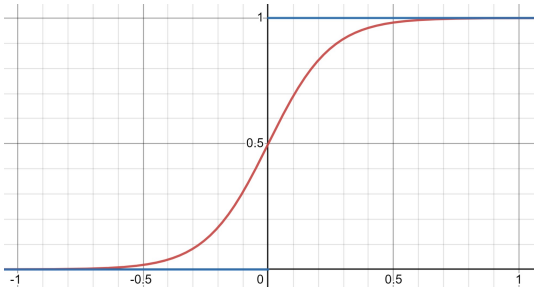


Figure 12: A comparison between the threshold function and the sigmoid function $\frac{1}{1+e^{-8x}}$. Both have a similar shape but the sigmoid function has a non-zero gradient. The threshold function is necessary, because the Bayesian network works with discrete variables. However, while calculating the gradient it is replaced with the sigmoid.

However, this loss can not be minimized by using gradient descent. Remember that convolutional kernels with parameters θ take some data x and multiply it by their weights and add a bias to get the higher level representations s. A threshold function is used on s to obtain data z that has either state 1 or state 0. Using the chain rule we get:

$$\frac{\partial L(D|BN)}{\partial \theta} = \frac{\partial L(D|BN)}{\partial z} * \frac{\partial z}{\partial s} * \frac{\partial s}{\partial \theta} \tag{9}$$

Since, z was obtained using the threshold function of s, it is the case that $\frac{\partial z}{\partial s} = 0$, which makes the hole gradient 0.

A way to avoid this problem is to replace the threshold function by a sigmoid while calculating the gradient. By using a sigmoid function it is possible to calculate a pseudo-gradient with a non-zero solution. Even though it is not the real gradient, it may still help to reduce our loss function $\mathcal{L}(\theta)$. The sigmoid function that worked well and was used to replace the threshold, function is $\frac{1}{1+e^{-8x}}$. The functions are plotted in Figure 12.

Overall, there are two optimization problems: The kernels need to be trained and the relations need to be trained. Training the kernels requires that there are already some relations and training the relations requires that there are already some kernels. To do this, an EM-algorithm is used. Initially the kernels and relations are random, but get iterativly updated until convergence. Note that this method may only find a local maxima. The pseudo-code of this algorithm is shown in algorithm 2.

To train a model, two convolutional transitions were used. Both transitions used five kernels of size 3 by 3 followed by a 2 by 2 max pooling with a stride of 2. This causes nodes in the two hidden layers to have a receptive field of size 4 by 4 and 10 by 10 respectively. Algorithm 2 shows how the parameters of the kernels and structure between hidden variables were trained in detail. It was used first to generate the first hidden layer and a second time afterwards to generate the second hidden layer. The EM-algorithm was run for 50 iterations each and the learning rate α to train the kernels was 0.008.

Gibbs sampling was once again performed to let the new model generate samples. The results are shown in Figure 13. Comparing them to the results shown in Figure 10, the images seem to have more pronounced lines. Otherwise they are at best vaguely resemble actual handwritten digits. The performance for the semi-supervised outlier task is shown in Table 2. The average AUROC score of

Algorithm 2 The pseudo-code to add an additional layer of hidden nodes to a Bayesian network.

procedure ADDLAYER(bn, n, D, α)
  input: a Bayesian network bn, a number of kernels n, a dataset D and learning rate α
  initialize n convolutional kernels randomly
  while no convergence **do**
    $optimizeStructure(bn, D, n)$
    $optimizeKernels(bn, D, \alpha)$

procedure OPTIMIZESTRUCTURE(bn, D, n)
  remove all relations in bn
  $dist \leftarrow$ the distance between two nodes such that their receptive fields are not overlapping.
  while BIC(bn,D) can be reduced **do**
    $k_1, k_2, dir \leftarrow$ parameters that result in the biggest bic reduction
    with $k_1, k_n \in [1, n]$, $dir \in \{north-west, north, north-east, east\}$
    for all nodes v of kernel k_1 **do**
      v' the node $dist$ away of v in direction dir of kernel $k2$.
      Add a relation from v' to v

procedure OPTIMIZEKERNELS(bn, D, α)
  $\theta \leftarrow$ the parameters of the convolutional kernels.
  $bn' \leftarrow$ bn without relations
  $\mathcal{L}(\theta) \leftarrow P(D|bn) - P(D|bn')$
  $\frac{\partial \mathcal{L}'(\theta)}{\partial \theta} \leftarrow$ the derivative of $\mathcal{L}(\theta)$ with respect to θ with the threshold function being replaced by a sigmoid function.
  $\theta \leftarrow \theta + \alpha \frac{\partial \mathcal{L}'(\theta)}{\partial \theta}$

93.0 is now a bit higher than without hidden layers. With the new learning algorithm training is quite slow and does not scale up well for larger image sizes.

	0	1	2	3	4	5	6	7	8	9	Average
AUROC	98.9	99.3	91.3	89.2	90.9	86.7	98.0	94.3	87.7	93.8	93.0

Table 2: The AUROC scores for an semis-supervised outlier detection task using MNISt images. In each of the ten experiments the model was trained to consider one of the numbers as normal and the other abnormal. The introspective energy model used conditional probability tables for the sidewards-pass and used 3 layers.

Overall, the results for generating samples are poor with and without hidden variables and lack behind the state of the art which can produce very realistic samples [62] [65]. A possible problem with this could be the generation method using Gibbs sampling, which may get stuck too easily in local maxima. Maybe using more iterations could increase performance.

However, the major problem with the hidden variables introduced is that the local Markov property is violated. The local Markov property demands that the state of a variable is independent of the other nodes given its Markov blanket. This is not the case in the presented model. A hidden variable that

Figure 13: Samples generating using Gibbs sampling of an Bayesian network with two hidden layers trained on the MNIST dataset.

encodes a feature of an image region causally depends on the nodes that encode pixels in this region. However, there are no dependencies between nodes of different layers. Similarly, hidden variables with the same receptive field that stem from different kernels also do not have relations but directly depend on each other. For example, a hidden variable may be a feature encoder for a horizontal line and another variable may be a feature encoder of a vertical line at the same region. Both features are mutually exclusive, which causes them to probably not be independent of each other given their parents in other image regions.

Unfortunately, there does not seem to be an easy solution for this issue. Adding these relations to the hidden variables would make them largely independent of the hidden variables further away in the image. But these relations are important and are the reason why hidden variables were added to the model at all.

All code used for the experiments was written in the C programming language using the OpenMP interface for multiprocessing. All files are available at https://github.com/LukasKinder/convolutionalBN.

4 Introspective energy models for semi-supervised outlier detection

In the previous section, the sidewards pass was performed using conditional probability tables. Now, small neural networks will be used instead. The nodes values do not need to be categorical anymore, which is why the threshold function is replaced by the ReLU activation function. Each node is associated with a small neural network that predicts the nodes activation using the activations of the surrounding nodes. A high energy is a big mismatch between the actual and predicted activations.

A possible application for this is semi-supervised outlier detection. After training the model with normal images, outliers are detected by their high energy. To asses the performance of the model, the MVTec dataset was used [9]. The MVtec dataset is subdivided into 15 categories of a different industrial manufactured object. For each category there are between 60 and 320 images of non-defect instances in the training-set. The test-set contain defects as well as non-defect instances that need to be identified. The rgb-images have a width and height between 900 and 1024 pixels. Examples of MVTec images are shown in Figure 14 and a statistical overview is shown in Table 3.

	Category	#Train	#Test (good)	#Test (defect)	Image side length
Objects	Bottle	209	20	63	900
	Cable	224	58	92	1024
	Capsule	219	23	109	100
	Hazelnut	391	40	70	1024
	Metal Nut	220	22	93	700
	Pill	267	26	141	800
	Screw	320	41	119	1024
	Toothbrush	60	12	30	1024
	Transistor	213	60	40	1024
	Zipper	240	32	119	1024
Textures	Carpet	280	28	89	1024
	Grid	264	21	57	1024
	Leather	245	32	92	1024
	Tile	230	33	84	840
	Wood	247	19	60	1024
	Total	3629	467	1258	-

Table 3: An overview of the MVTec dataset. For each object and texture the number of training images as well as the number of defect and normal testing images is given.

All code was written in the Python programming language using the PyTorch library. It is available under https://github.com/LukasKinder/Master.

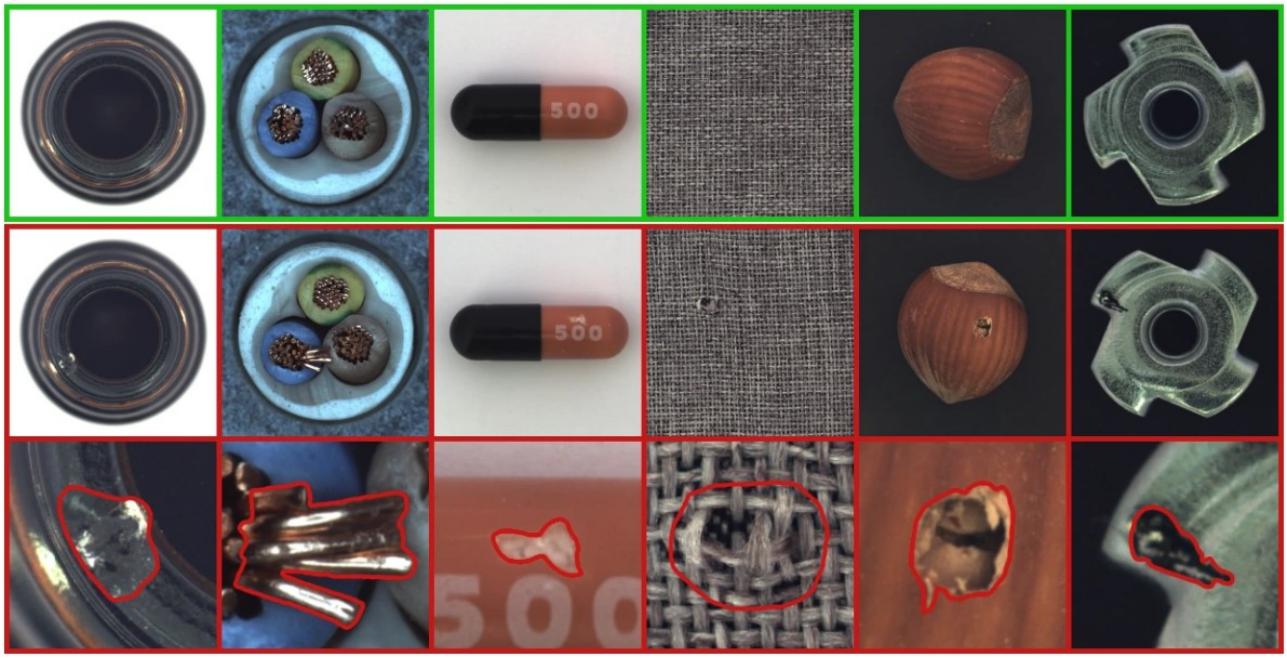


Figure 14: Example instances of the MVTec dataset. The first row shows defect-free samples, the second row shows defect samples and the third row shows the zoomed-in defects.

4.1 Methods

Formally, an introspective energy model is a convolutions network in which the activations in layer l are determined by: $a^{[l]} = conv(a^{[l-1]})$. With $a^{[0]}$ being the input image. The transition $conv()$ performs convolution, max pooling and applies the ReLU activation function. This is the forward-pass. The forwards pass generates a spacial map of nodes which corresponding to different regions of the image. There are as many nodes corresponding to the same region as there are convolutional kernels. During the sideward-pass the activation $a_i^{[l]}$ of a node $n_i^{[l]}$ at layer l is predicted using the activations of the neighbours of the node $neigh(n_i^{[l]})$. For this task there is a model $model_i^{[l]}()$ associated to this specific node:

$$\hat{a}_i^{[l]} = model_i^{[l]}(\ [a_j^{[l]} \mid n_j^{[l]} \in neigh(n_i^{[l]})] \ ) \tag{10}$$

Here, $\hat{a}_i^{[l]}$ is the prediction. The neighbours of a node $neigh(n_i^{[l]})$ are at the same layer and should be nodes with receptive fields that are adjacent to $n_i^{[l]}$. The difference between a nodes activation as determined by the forward-pass and its predicted activation during the sideward-pass is the error associated with this node.

$$err_i^{[l]} = \hat{a}_i^{[l]} - a_i^{[l]} \tag{11}$$

The overall loss in a layer l and the objective function is:

$$\mathcal{L}^l = \sum_i (err_i^{[l]})^2 \tag{12}$$

During training, the model aims to minimize this error by optimizing the weights of the convolutional kernels and the parameters of the models performing the sidewards-pass. This is done using stochastic gradient descent for the images in the training-set. However, the network may fail to learn meaningful

higher level concepts. For example because it always maps to 0 during the forward pass. If $a_i^{[l]} = 0$ for every image in the training-set, $model_i^{[l]}()$ will learn to output 0 without considering the input. The resulting error will be 0 as well.

To avoid this, while backtracking the gradient the actual activations are detached from the gradient.

$$err_i^{[l]} = \hat{a}_i^{[l]} - a_i^{[l]}.stopGradient() \tag{13}$$

In other words, while calculating the gradient during training, the nodes are trained to be good predictors of their neighbours and not to be easily predictable.

The models $model_i^{[l]}()$ for each node comes in the form of small neural networks. Since the training-set of the MVTec dataset is small, they may overfit. That means that they are good in predicting the activations for the limited training-set but fail to generalize for other (defect-free) images. To avoid this, weight sharing can be used. This means that (part of) the weights that made up a model $model_i^{[l]}()$ are the same in other models $model_j^{[l]}()$. This effectively allows the parameters of the weights to be trained with more data. For example, lets assume the number of training images n is quite small. The size of the images is 300 by 300 pixels. We would like to train the models that perform the sidewards pass in layer 0 that predict a pixel value. If in this case, the weights of all sidewards-models are shared, the weights are effectively trained with $n * 300 * 300$ samples. With this increase amount of data, the chance for overfitting is small, even if the models for the sidewards pass have many weights.

However, if all models share the same weights the position of the node is not incorporated in the prediction. This may not be a problem if the image shows a texture or pattern. For these types of images it is often the case that the information of the position within the image is not useful to predict a nodes activation. However, usually the position in the image is important. Many images in the MVTec dataset show objects that are centered, which creates an expectation for certain characteristics in different image regions. For example, in images showing a capsule we expect the outline of the upper part of the capsule to be located in the upper third of the image.

In order to incorporate information about the position of the node there are two methods. The model may receive the x and y position of the node as an input (both normalized as a value between 0 and 1). Or alternatively, the model associated to a node may only contain some shared weights and some weights that are specific to the node. Shared weights can come in two forms: Weights may be shared between all models associated to nodes in a layer or weights may be shared between models of nodes that corresponds to the same convolutional kernel. Overall, six different architectures were considered in which different types of weight sharing was used. They are shown in Figure 15. The models of type (a) uses the most weight sharing but no information about the position of the node. This caused a poor performance for some object categories of the MVTec dataset that are not textures. Model (d) uses no weight sharing at all, but had a poor performance, probably because of overfitting. Model (e) and (f) have a lot of weight sharing and are still able to incorporate information about the position of the node. However, they also did not perform very well. The reason is probably that in the MVTec dataset the objects are often position exactly in the center of the image and have the same size. This makes the exact position very important for the expected activation. Just giving the position as an input is not good enough, because small changes in position can have a big effect. It should be investigated if models of type (e) or (f) would perform better for other dataset, like for example ImageNet. Model type (b) and (c) are similar but (c) only shares weights between models of node that correspond to the

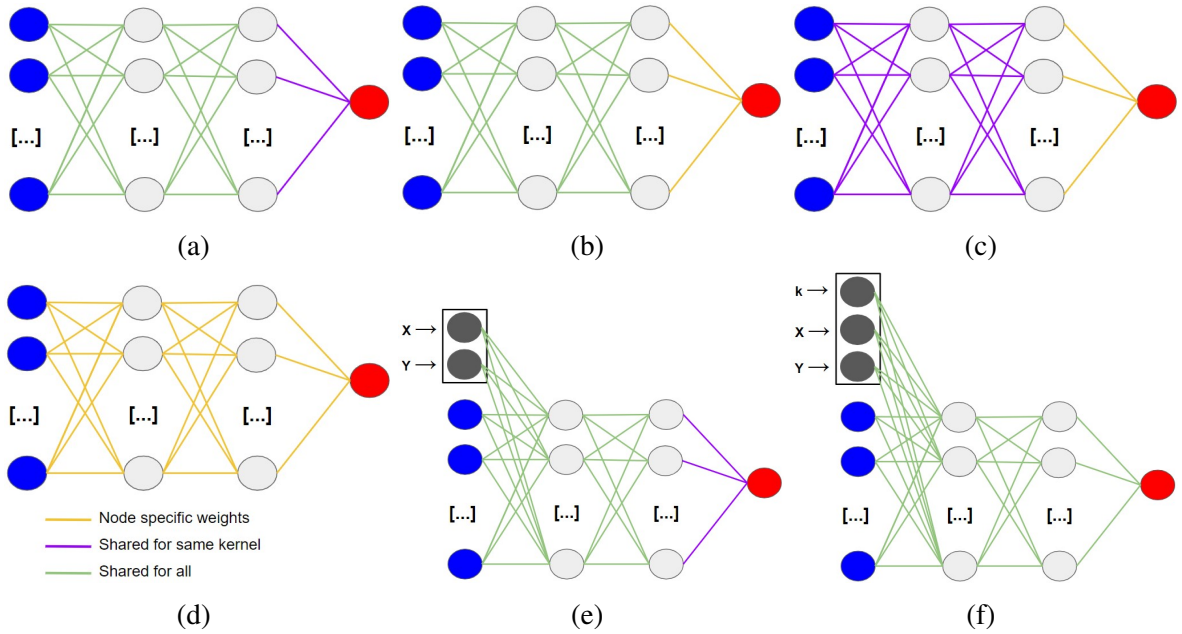


Figure 15: Six different designs for neural networks performing the sidewards-pass. The activations of the neighbours (blue) are used to predict the activation of a node (red). (a): The models share their weights except for the last layer, in which weights are shared for nodes of the same convolutional kernel.(b): Weights are shared between all nodes, except in the last layer. (c): Weights are shared between nodes of the same kernel, except in the last layer. (d): No weight-sharing at all. (e) The weights are shared with all nodes in the initial layers and shared between nodes of the same kernel in the last layer. Additionally, the x and y coordinate of the node is given as an input as well. (f) All weights are shared. The model receives the x and y coordinate of the node, as well as the kernel of the node.

same kernel. They performed the best and type (b) was chosen for the following experiments.

A new model was trained from scratch for each MVTec category. The architecture for the overall model for each category was the same and is shown in Table 4. To summarize: on top of layer 0 which is the pixel level there were four more layers that used 15,25,35 and 60 convolutional kernels respectively. Max pooling was always used with size 2 by 2 and a stride of 2. The receptive fields of a node went up to a size of 66 in the last layer. This is actually not a big fraction of the 300 by 300 pixel images. This was fine because defects usually impact only small regions. A node at position (x,y) was neighbour with a node (x',y') if and only if $max(|x-x'|,|y-y'|) = Neighbour\ Distance$. For example, in the first layer the neighbour distance is 1 which means that there are 8 surrounding positions with neighbours. The image has three channels (rgb) which means that there are $8*3=24$ neighbours in total. The neighbour distance was chosen such that the receptive fields between neighbours in layer 2,3 and 4 slightly overlap. The architecture of the sidewards model used 2 hidden layers with 32 and 12 hidden units, except in layer 4 were there were three hidden layers with 32, 32 and 16 hidden units each.

A node located close the edge of an image may not have neighbours in all direction. Since weight sharing was used it would still be good if the architecture of this nodes' sidewards model is still the same. This is why the activation of a missing neighbour was replaced by 0. However, the models

Layer	Convolution	Receptive Field Size	Neighbour Distance	Sidewards Models
0	-	1	1	[32,32,12,1]
1	conv:n=15,size=3 max-pool:size=2,stride=2	4	3	[384,32,16,1]
2	conv:n=25,size=5 max-pool:size=2,stride=2	14	3	[624,32,16,1]
3	conv:n=35,size=5 max-pool:size=2,stride=2	34	3	[864,32,16,1]
4	conv:n=60,size=3 max-pool:size=2,stride=2	66	2	[976,32,32,16,1]

Table 4: The cold-start architecture of the IEM used for the MVTec dataset. For each layer the convolutional transition, the resulting size of the receptive field, the distance of the neighbours and the number of nodes of the model performing the sidewards pass is shown. In the last column, the first number always indicates the number of inputs, followed by numbers describing the number of nodes in the hidden layers. The last number indicated the number of output nodes which is alway 1.

were also provided with a mask telling them if there does or does not exist a neighbour in a certain position. This is for example why a model performing the sidewards pass in layer 0 has 32 input. 24 of these inputs are the activations of the neighbours and another 8 inputs is the mask telling if at each of the 8 neighbour-position there actually is a node or not.

Training the convolutional kernels and models for the sidewards-pass was done layer by layer. That means that training the next layer was only done after the previous layer was completely trained. The pseudocode for this is shown in Algorithm 3. This method of training performed well and was a bit more stable than training all layers at once. The learning rates α_s and α_f were 0.001 and a batch size $N = 8$ was used. The images were all resized to 300 by 300 pixels, to make training computationally faster.

To calculate the energy after training, it was beneficial to normalize the errors of each node by assuming they are normally distributed. This is similar to the method proposed by [1] but is done over all nodes instead of over the pixels. We can normalize this error as follows:

$$err\_norm_{i,x}^{[l]} = \frac{|err_{i,x}^{[l]} - mean(err_i^{[l]})|}{std(err_i^{[l]})} \tag{14}$$

Here, $err_{i,x}^{[l]}$ is the error of node $n_i^{[l]}$ for image x. With $mean(err_i^{[l]})$ being the mean and $std(err_i^{[l]})$ being the standard error of the error of node i in layer l for the instances in the training-set.

Until this point each node is individually associated with an error. The final energy associated with a layer for an input is calculated via:

Algorithm 3 The pseudo-code to add additional layers of hidden nodes to an IEM. The layer are added one by one. The appendix ".detach" means that a variable is not backtracked while calculating a derivative,

procedure TRAINIEM($D, L, N, \alpha_s, \alpha_f$)
  input: Data D, number of layers L, batch size N, learning rates α_f and α_s
  for l = 0,...,L **do**
    if l = 0 **then**
      For each pixel, add sidewards models with parameters θ_s
      $\theta_f \leftarrow \emptyset$
    else
      Add a convolutional transition to the model with parameters θ_f
      For each new node, add sidewards models with parameters θ_s
    while no convergence **do**
      $B \leftarrow$ A batch of N images of D.
      $\mathcal{L} \leftarrow \sum_{b \in B} \sum_i (\hat{a}_i^{[l]} - a_i^{[l]}.detach)^2$
      $\theta_f = \theta_f + \alpha_f \frac{\partial L}{\partial \theta_f}$
      $\theta_s = \theta_s + \alpha_s \frac{\partial L}{\partial \theta_s}$

$$energy^{[l]} = \sum_i |err\_norm_i^{[l]}| \tag{15}$$

To associate an input with an energy we can combine the energies of each layer using:

$$energy = \sum_l w_l * energy^{[l]} \tag{16}$$

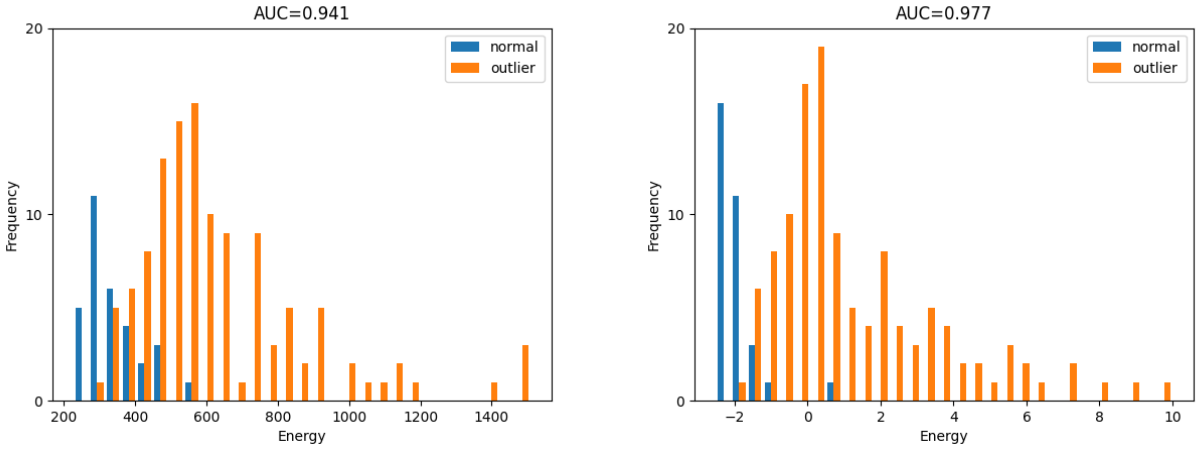
Here w_l is a weight associate with a layer that can fine tuned. However, for the experiments of the MVTec dataset they were set to 1 for all layers.

4.2 Results

An example of how error normalization is improving the models performance is shown in figure 16. Here, the distribution of normal and defect testing images of the "Zipper" object category is shown with and without error normalization. A much higher AUC of 0.977 was achieved with normalization which was considerably higher than without normalization. A similar trend was observed for other object categories.

An image is categorized by the model as normal or outlier based on its energy. A threshold must be chosen that indicates the maximum energy an image can have to be still considered normal. This can be visualized in a (receiver operating characteristic) ROC curve . A commonly used metrics to measure the performance for the MVTec dataset [9] is to use the area under the ROC curve (AUROC).

The results in terms of AUROC for every object and texture is shown in Table 5. The other models used for comparison were state of the art models trained from scratch and do not utilizing a pretrained model. Patch SVDD is a distance based model, ITAE and RIAD are autoencoders and GANomaly is



(a) No normalization

(b) With normalization

Figure 16: An example of the energy distribution for images of normal and defect instances. The IEM is able to separate normal from defect instances better if error normalization was performed. Images come form the ”Zipper” object of the MVTec dataset.

a GAN. The Introspective energy model has an average AUROC that is lower than RIAD and PATCH SVDD. However, this average is mainly low because of a low outlier for the ”Metal Nut” object category. The Introspective Energy Model performs better than any other model for the ”Hazelnut”, ”Carpet” and ”Wood” object category and is able to perfectly distinguish normal from defect screws.

4.3 Discussion

Note that the MVTEc dataset has quite small testing-sets for each category. Therefore the exact results should be seen with caution. During testing, the model is usually only misclassifying a handful of images. By performing many runs with different architectures and weights for the layers, it could be possible to ”randomly” get better results for a specific object category. This is why the weights for the layers were always 1 and the architecture was not fine tuned for different categories of the MVTec dataset.

The overall results are quite promising, even though the average performance was not as good as the state of the art. However, the reason the average is low, is manly because of some object categories for which the model had a very bad performance, like ”Metal Nut”, ”Capsule” or ”Tile”. The IEM did outperform all other models in some object categories.

A possible explanation of why the model performed poor for some categories, is that normal objects sometimes have random components. Examples of cases like this is shown in Figure 17. The IME tries to predict regions of these images using the surrounding region and assign a high energy if these predictions are bad. However, in this case accurately predicting an image region is impossible because of the random nature of the shown objects. The lines of the leather, the red dots of the pill and the dark dots on the tile are random. Therefore the model will assign high energies to normal instances. If normal instances are assigned high energies it is hard to distinguish them from abnormal instances that have high energies because of an defect.

	Patch SVDD[74]	ITAE[75]	GANomaly[5]	RIAD[54]	Introspective Energy Model
Bottle	98.6	94.1	89.2	**99.9**	99.0
Cable	**90.3**	83.2	75.7	81.9	78.5
Capsule	76.7	68.1	73.2	**88.4**	72.1
Hazelnut	92.0	85.5	78.5	83.3	**94.8**
Metal Nut	**94.0**	66.7	70.0	88.5	53.6
Pill	**86.1**	78.6	74.3	83.8	82.8
Screw	81.3	**100**	74.6	84.5	**100**
Toothbrush	**100**	**100**	65.3	**100**	95.8
Transistor	91.5	84.3	79.2	90.9	**95.7**
Zipper	95.1	87.6	74.5	**98.1**	97.8
Carpet	92.9	70.6	69.9	84.2	**94.1**
Grid	94.6	88.3	70.8	**99.6**	93.1
Leather	90.9	86.2	84.2	**100**	96.8
Tile	97.8	73.5	79.4	**98.7**	87.2
Wood	96.5	92.3	83.4	93.0	**97.0**
Average	**92.1**	83.9	76.2	91.7	87.9

Table 5: The AUROC scores in abnormally detection for different categories of the MVTec dataset. The introspective energy model was compared with state of the art models that also did not use pretraining using a different dataset.

The model performed particularly bad for the ”Metal Nut” object, which is shown on the right in Figure 14. The AUROC score of 53.6 is barley better than random. This could be because the object is also a bit random. The shiny and dim parts of the metal nut are often hard to predict exactly. Another explanation could be that the metal nuts are rotated differently in every image. This prevents to model to learn at which exact positions edges should be. However, the same is true for the ”Screw” object, for which the model performs very well. Maybe the difference is that the metal nuts do not have any meaningful textures or pattern if zoomed into the image. This could make it hard to learn meaningful higher level representation, as the model is only trained layer by layer.

A potential problem with the model is that the convolutional kernels are only trained with images of normal instances. This could mean that they are not able to pick on higher level features that only appear for defects. A possible solution for this could be to train the forwards pass with normal as well as defect instances and the parameters for the sidewards pass afterwards, with normal instances only.

Even without fine-tuning the hyper-parameters for the different object categories, it was a lot of work to find good settings. The design for the forwards and sidewards pass give many possible options. On top of this, the learning mechanism gives a big search space of hyper-parameters as well. This is in general a challenge for machine learning [76]. However, in this case the novel nature of the model makes it harder consult existing literature.

Figure 17: Some images of the MVTec dataset showing non-defect samples. A sample of the ”Leather” category is shown left, ”Pill” in the middle and ”Tile” on the right. Even though the objects are normal, they have random aspects that are hard to predict. this causes the IEM to assign them high energies, even though they do not have any defects.

The model presented here shows some similarities to the RIAD model, because both determine an energy by the success of an inpainting task. However, the IEM has the advantage that it only needs to be run once, while RIAD is run many times with different regions being obstructed. This makes the IEM computational cheaper.

An interesting aspect of the model is that it is able to train a convolutional network layer by layer. This is relevant as training a deep neural network often struggles with the vanishing gradient problem [77, 78]. In this case, the gradient that is backpropagated staring in the output layer becomes smaller and smaller for each layer before. This can prevent effective training. By training the layer one by one, the problem is overcome. Existing methods to train neural networks layer by layer is greedy layer-wise pretraining [79, 80]. Maybe future work could investigate further, if an IEM can be preferred over layer-wise petraining for some detests.

5 Introspective energy models with transfer learning

Even though the previous section showed promising results for the MVTec dataset, they lack far behind state of the art models that use transfer learning. Taking pretrained convolutional kernels from a model trained with the ImageNet dataset can achieve AUROC score of 99 or more for each of the 15 object categories.

A possible method to improve the Introspective energy model is to transfer the convolutional kernels of an already pretrained model. This section discusses how this is done in detail using the AlexNet model [16].

5.1 Methods

Since the MVTec dataset is very small, it could be very beneficial to pretrain it using a large dataset. For example, it may be trained on the ImageNet [46] dataset before fine-tuning it for the training images of the MVTec dataset. Even though this could work, the problem is that the ImageNet dataset is very big and pretraining a model with it would be very computationally expensive. Instead, it is possible to transfer the parameters of an already existing model. The hope is that the convolutional kernels of the pretrained model are useful for the task of predicting outliers of the MVTec dataset. In this case, only the parameters for the sidewards-pass need to be trained from scratch.

The pretrained model that was used for this task is the AlexNet [16] model. The reason it was chosen is that the first few layers have relatively little convolutional kernels and that the size of the convolutional kernels are relatively small. Only the parameters of the first four convolutional layers were used, because using more would make the receptive fields of the nodes too large. The exact architecture is shown in Table 6. The main difference to the architecture that was used before is that there are more kernels and that the receptive fields become larger in later layers. Since there are more kernels, the models performing the sidewards-pass have more inputs. To compensate for this, the models were made larger, with more hidden units. The images were resized to 224 by 224 pixels which is an requirement for AlexNet.

During training the weights of the convolutional kernels were not changed and only the weights of the models performing the sidewards pass needed to be trained. This performed better than taking the pretrained weights as a starting point and fine-tuning them. Because of this, it does not matter if the model is trained layer by layer or all at once. Otherwise, training is done as before by minimizing the objective function of equation 12 using stochastic gradient descent. The learning rate was 0.001 and the batch size was 8. The weights that were used for each layer as described in Equation 16 were always 1.

5.2 Results

The results in terms of AUROC for every object and texture is shown in Table 7. Using pretrained kernels increase the average AUROC from 87.9 to 90.8. However, there are some object categories for which the results were actually considerably worse, like for ”Grid” and ”Screw”.

The three state of the art models used for comparison were all distance based models that use pretrained kernels of a deep convolutional network trained on ImageNet. The overall performance of the

Layer	Convolution	Receptive Field Size	Neighbour Distance	Sidewards Models
0	-	1	1	[32,32,16,1]
1	conv:n=64,size=11,stride=4 max-pool:size=3,stride=2	19	1	[512,32,16,1]
2	conv:n=192,size=5 max-pool:size=3,stride=2	83	1	[1536,54,64,32,1]
3	conv:n=384,size=3	114	3	[9984,64,64,16,1]
4	conv:n=256,size=3	147	1	[2048,64,64,32,1]

Table 6: The architecture of the IEM using the convolutional kernels of the Alexnet Model. For each layer the convolutional transition, the resulting size of the receptive field, the distance of the neighbours and the number of nodes of the model performing the sidewards pass is shown.

IEM is considerably worse than the state of the art models for most object categories.

5.3 Discussion

State of the art models that detect defect instances for the MVTec dataset greatly benefit from pre-trained models. This is probably because the training-set of the MVTec dataset is very small. It seems like large the convolutional models trained on the ImageNet dataset come with higher level representations that are often useful for the MVTec dataset as well.

Using the convolutional kernels of AlexNet in the introspective energy model did lead to an improvement for a many of the objects. The reason that it did not help for all object categories could be that they require very specific kernels that are not part of AlexNet.

It is important to not that only the first four layers of AlexNet were used. This is only a small part of the parameters. Unfortunately it was not possible to use more layers, because otherwise the receptive fields become to large. The high amount of convolutional kernels may also prevented the model to have a better performance. During experiments without transfer learning a too high amount of convolutional kernels cause a bad performance. This is probably because the models performing the sidewards pass receive a too large input, making training slow and overfitting more likely.

Interestingly, a better performance was achieved by transferring the parameters of the convolutional kernels without changing the afterwards. It was not leading to an improvement to fine-tune the kernels with the images of the MVTec dataset. This is probably because the training-set of the MVTec only contains non-defect instances. By fine-tuning the kernels with images of non-defects. they may not be able pick up on features that only appear in images of defect instances. However, this is important to distinguish them.

	ReConPatch[81]	PatchCore[10]	MemSeg[82]	Introspective energy Model
Bottle	**100**	**100**	**100**	98.9
Cable	**99.7**	99.5	98.2	89.5
Capsule	99.8	98.1	**100**	78.3
Hazelnut	**100**	**100**	**100**	98.9
Metal Nut	**100**	**100**	**100**	79.2
Pill	96.21	96.6	**99**	85.5
Screw	**99.8**	98.1	97.8	88.7
Toothbrush	**100**	**100**	**100**	96.9
Transistor	**100**	**100**	99.2	89.3
Zipper	99.9	99.4	**100**	90.8
Carpet	**100**	98.7	99.6	91.0
Grid	99.5	98.2	**100**	80.1
Leather	**100**	**100**	**100**	96.8
Tile	**100**	98.7	**100**	99.1
Wood	99.47	99.2	**99.6**	99.2
Average	**99.6**	99.1	99.56	90.8

Table 7: The AUROC scores in abnormally detection for different categories of the MVTec dataset. The introspective energy model used pretrained kernels of the AlexNet model and was compared with state of the art models that also utilized pretrained models.

6 Conclusion

This thesis introduces the concept of an introspective energy model that uses a new architecture to assign energies to input images. A high energy is a measure of the disability to predict image regions based on the surrounding. One application for which it can be used is semi-supervised outlier detection, where the training-set only contains non-outliers. An introspective energy model that consists of layers of Bayesian networks was used for the MNIST dataset. Although it could be used for this task, the model struggled to learn the probability distribution of MNIST images effectively. The samples generated by the model using Gibbs sampling did resemble numbers very much. The main problem with using an introspective energy model with conditional probability tables is that it is not a true Bayesian network, because the local Markov property is violated. Moreover, the model's training process is computationally expensive, which makes it impractical for tasks involving large images.

A new version of the model was developed, utilizing small neural networks to predict image regions based on their surroundings. With this mechanism the model was able to outperform the state of the art in some categories of the MVTec dataset. However, an introspective energy model using pretrained kernels was not as good as distance-based models using pretrained models. A possible explanation for this is that the model used for transfer learning had too many convolutional kernels and the only the first few layers could be transferred.

There are image outlier detection tasks for which it is hard to use pretrained kernels. Very domain-specific datasets like x-ray, satellite, art or microscopic images usually do not benefit much from pretrained convolutional kernels trained with IamgeNet [83]. Therefore it is still relevant to explore models that do not depend on transfer learning.

In conclusion, introspective energy models are studied in this thesis and are worth to further investigated for the following reasons:

- Since every node of an introspective energy model has its own loss, the gradient does not need to be backtracked over multiple layers. This avoids the vanishing gradient problem.
- Introspective energy models are relatively easy to train, because layers can be trained one by one. This gives much control over the learning progress and makes it easier to pinpoint potential problems during training.
- An IEM is a gray-box model that can be much more explainable than traditional deep learning models. It can provide information about where in the input and in in which layer a high energy originates. It could potentially also be used to determine how the input should be different in order to reduce the energy.
- An IEM is an energy based model that can be trained without ever being presented with high-energy examples. For the MVTec dataset it was only outperformed by non-deep learning models that may not scale up to more difficult tasks.
- There is not much data required to train an IEM, because it is able to use a lot of weight-sharing. This is different to traditional deep learning models. With lot of weight sharing the number of parameters are reduced, which helps to prevent overfitting.

- Based on the task at hand, the architecture of an IEM can be chosen using expert knowledge. For example by using position-independent sidewards-models for texture inputs or by selecting receptive field sizes that match the size of important features.

It could also be interesting to use IEMs not only for image data but also for videos or audio data. For videos 3D convolution can be applied and the neighbours of a node may be nodes not only adjacent in space but also in time. When using an introspective energy model for audio data, a node encoding a feature of the audio at some time may be predicted using nodes that encode features of the audio before and afterwards.

Bibliography

[1] E. U. Genc, N. Ahuja, I. J. Ndiour, and O. Tickoo, "Energy-based anomaly detection and localization," *arXiv preprint arXiv:2105.03270*, 2021.

[2] Y. Du and I. Mordatch, "Implicit generation and modeling with energy based models," *Advances in Neural Information Processing Systems*, vol. 32, pp. 3603—-3613, 2019.

[3] C. Geng, J. Wang, Z. Gao, J. Frellsen, and S. Hauberg, "Bounds all around: training energy-based models with bidirectional bounds," *Advances in Neural Information Processing Systems*, vol. 34, pp. 19808–19821, 2021.

[4] Y. Song and D. P. Kingma, "How to train your energy-based models," *arXiv preprint arXiv:2101.03288*, 2021.

[5] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, "Ganomaly: Semi-supervised anomaly detection via adversarial training," in *ACCV.Springer*, pp. 622–637, 2018.

[6] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *International Conference on Learning Representations*, 2018.

[7] E. M. Knorr, R. T. Ng, and V. Tucakov, "Distance-based outliers: algorithms and applications," *The VLDB Journal*, vol. 8, no. 3, pp. 237–253, 2000.

[8] N. Cohen and Y. Hoshen, "Sub-image anomaly detection with deep pyramid correspondences," *arXiv preprint arXiv:2005.02357*, 2020.

[9] P. Bergmann, K. Batzner, M. Fauser, D. Sattlegger, and C. Steger, "The mvtec anomaly detection dataset: a comprehensive real-world dataset for unsupervised anomaly detection," *International Journal of Computer Vision*, vol. 129, no. 4, pp. 1038–1059, 2021.

[10] K. Roth, L. Pemula, J. Zepeda, B. Schölkopf, T. Brox, and P. Gehler, "Towards total recall in industrial anomaly detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14318–14328, 2022.

[11] D. Berthelot, T. Schumm, and L. Metz, "Began: Boundary equilibrium generative adversarial networks," *arXiv preprint arXiv:1703.10717*, 2017.

[12] J. Zhao, M. Mathieu, and Y. LeCun, "Energy-based generative adversarial network," in *5th International Conference on Learning Representations*, 2016.

[13] D. Zimmerer, F. Isensee, J. Petersen, S. Kohl, and K. Maier-Hein, "Unsupervised anomaly localization using variational auto-encoders," in *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part IV 22*, pp. 289–297, Springer, 2019.

[14] J. Lazarow, L. Jin, and Z. Tu, "Introspective neural networks for generative modeling," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2774–2783, 2017.

[15] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[17] J. Pearl, "Reverend bayes on inference engines: A distributed hierarchical approach," in *Proceedings of the Second AAAI Conference on Artificial Intelligence*, AAAI'82, p. 133–136, AAAI Press, 1982.

[18] D. Heckerman, "A tutorial on learning with bayesian networks," *Innovations in Bayesian networks*, pp. 33–82, 2008.

[19] J. Su and H. Zhang, "Full bayesian network classifiers," in *Proceedings of the 23rd international conference on Machine learning*, pp. 897–904, 2006.

[20] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning bayesian networks: The combination of knowledge and statistical data," *Machine learning*, vol. 20, no. 3, pp. 197–243, 1995.

[21] Z. Liu, B. Malone, and C. Yuan, "Empirical evaluation of scoring functions for bayesian network model selection," in *BMC bioinformatics*, vol. 13, pp. 1–16, BioMed Central, 2012.

[22] M. Scanagatta, A. Salmerón, and F. Stella, "A survey on bayesian network structure learning from data," *Progress in Artificial Intelligence*, vol. 8, no. 4, pp. 425–439, 2019.

[23] J. Binder, D. Koller, S. Russell, and K. Kanazawa, "Adaptive probabilistic networks with hidden variables," *Machine Learning*, vol. 29, no. 2, pp. 213–244, 1997.

[24] Z. Ghahramani, "Learning dynamic bayesian networks," *International School on Neural Networks, Initiated by IIASS and EMFCSC*, pp. 168–197, 1997.

[25] A. Anandkumar, D. Hsu, A. Javanmard, and S. Kakade, "Learning linear bayesian networks with latent variables," in *International Conference on Machine Learning*, pp. 249–257, PMLR, 2013.

[26] G. Elidan, N. Lotner, N. Friedman, and D. Koller, "Discovering hidden variables: A structure-based approach," *Advances in Neural Information Processing Systems*, vol. 13, 2000.

[27] G. Elidan, *Learning Hidden Variables in Probabilistic Graphical Models*. PhD thesis, Hebrew University of Jerusalem, 2004.

[28] N. Trifonova, A. Kenny, D. Maxwell, D. Duplisea, J. Fernandes, and A. Tucker, "Spatio-temporal bayesian network models with latent variables for revealing trophic dynamics and functional networks in fisheries ecology," *Ecological Informatics*, vol. 30, pp. 142–158, 2015.

[29] L. Uusitalo, M. T. Tomczak, B. Müller-Karulis, I. Putnis, N. Trifonova, and A. Tucker, "Hidden variables in a dynamic bayesian network identify ecosystem level change," *Ecological Informatics*, vol. 45, pp. 9–15, 2018.

[30] K. Masmoudi, L. Abid, and A. Masmoudi, "Credit risk modeling using bayesian network with a latent variable," *Expert Systems with Applications*, vol. 127, pp. 157–166, 2019.

[31] K.-B. Hwang, B.-H. Kim, and B.-T. Zhang, "Learning hierarchical bayesian networks for large-scale data analysis," in *International Conference on Neural Information Processing*, pp. 670–679, Springer, 2006.

[32] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, 2021.

[33] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[34] A. Dhillon and G. K. Verma, "Convolutional neural network: a review of models, methodologies and applications to object detection," *Progress in Artificial Intelligence*, vol. 9, no. 2, pp. 85–112, 2020.

[35] Q. Liu, N. Zhang, W. Yang, S. Wang, Z. Cui, X. Chen, and L. Chen, "A review of image recognition with deep convolutional neural network," in *Intelligent Computing Theories and Application: 13th International Conference, ICIC 2017, Liverpool, UK, August 7-10, 2017, Proceedings, Part I 13*, pp. 69–80, Springer, 2017.

[36] H. Ajmal, S. Rehman, U. Farooq, Q. U. Ain, F. Riaz, and A. Hassan, "Convolutional neural network based image segmentation: a review," *Pattern Recognition and Tracking XXIX*, vol. 10649, pp. 191–203, 2018.

[37] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1d convolutional neural networks and applications: A survey," *Mechanical systems and signal processing*, vol. 151, p. 107398, 2021.

[38] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 4489–4497, 2015.

[39] W. Wu, Z. Qi, and L. Fuxin, "Pointconv: Deep convolutional networks on 3d point clouds," in *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp. 9621–9630, 2019.

[40] W. Zhang, J. Tanida, K. Itoh, and Y. Ichioka, "Shift-invariant pattern recognition neural network and its optical architecture," in *Proceedings of annual conference of the Japan Society of Applied Physics*, pp. 2147–2151, Montreal, CA, 1988.

[41] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[42] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[43] M. Huh, P. Agrawal, and A. A. Efros, "What makes imagenet good for transfer learning?," *arXiv preprint arXiv:1608.08614*, 2016.

[44] C. A. Ferreira, T. Melo, P. Sousa, M. I. Meyer, E. Shakibapour, P. Costa, and A. Campilho, "Classification of breast cancer histology images through transfer learning using a pre-trained inception resnet v2," in *Image Analysis and Recognition: 15th International Conference, ICIAR*

2018, Póvoa de Varzim, Portugal, June 27–29, 2018, Proceedings 15, pp. 763–770, Springer, 2018.

[45] J. Chen, D. Zhang, Y. A. Nanehkaran, and D. Li, "Detection of rice plant diseases based on deep transfer learning," *Journal of the Science of Food and Agriculture*, vol. 100, no. 7, pp. 3246–3256, 2020.

[46] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[47] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, "A tutorial on energy-based learning," *Predicting structured data*, vol. 1, no. 0, 2006.

[48] W. Liu, X. Wang, J. Owens, and Y. Li, "Energy-based out-of-distribution detection," *Advances in neural information processing systems*, vol. 33, pp. 21464–21475, 2020.

[49] J. L. McClelland, D. E. Rumelhart, P. R. Group, *et al.*, *Parallel Distributed Processing, Volume 2: Explorations in the Microstructure of Cognition: Psychological and Biological Models*, vol. 2. MIT press, 1987.

[50] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, pp. 37–49, JMLR Workshop and Conference Proceedings, 2012.

[51] G. Williams, R. Baxter, H. He, S. Hawkins, and L. Gu, "A comparative study of rnn for outlier detection in data mining," in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pp. 709–712, IEEE, 2002.

[52] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang, "Deep structured energy based models for anomaly detection," in *International conference on machine learning*, pp. 1100–1109, PMLR, 2016.

[53] Z. Li, N. Li, K. Jiang, Z. Ma, X. Wei, X. Hong, and Y. Gong, "Superpixel masking and inpainting for self-supervised anomaly detection.," in *Bmvc*, 2020.

[54] V. Zavrtanik, M. Kristan, and D. Skočaj, "Reconstruction by inpainting for visual anomaly detection," *Pattern Recognition*, vol. 112, p. 107706, 2021.

[55] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[56] H. Alqahtani, M. Kavakli-Thorne, and G. Kumar, "Applications of generative adversarial networks (gans): An updated review," *Archives of Computational Methods in Engineering*, vol. 28, pp. 525–552, 2021.

[57] T. Defard, A. Setkov, A. Loesch, and R. Audigier, "Padim: a patch distribution modeling framework for anomaly detection and localization," in *Pattern Recognition. ICPR International Workshops and Challenges: Virtual Event, January 10–15, 2021, Proceedings, Part IV*, pp. 475–489, Springer, 2021.

[58] A. Lazarevic and V. Kumar, "Feature bagging for outlier detection," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 157–166, 2005.

[59] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang, "A novel anomaly detection scheme based on principal component classifier," tech. rep., Miami Univ Coral Gables Fl Dept of Electrical and Computer Engineering, 2003.

[60] H. Song, Z. Jiang, A. Men, and B. Yang, "A hybrid semi-supervised anomaly detection model for high-dimensional data," *Computational intelligence and neuroscience*, vol. 2017, 2017.

[61] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning," *Pattern Recognition*, vol. 58, pp. 121–134, 2016.

[62] E. Nijkamp, M. Hill, T. Han, S.-C. Zhu, and Y. N. Wu, "On the anatomy of mcmc-based maximum likelihood learning of energy-based models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 5272–5280, 2020.

[63] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 681–688, 2011.

[64] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in neural information processing systems*, vol. 30, 2017.

[65] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, 2016.

[66] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM computing surveys (CSUR)*, vol. 54, no. 2, pp. 1–38, 2021.

[67] J. Liu, K. Song, M. Feng, Y. Yan, Z. Tu, and L. Zhu, "Semi-supervised anomaly detection with dual prototypes autoencoder for industrial surface inspection," *Optics and Lasers in Engineering*, vol. 136, p. 106324, 2021.

[68] P. Bergmann, K. Batzner, M. Fauser, D. Sattlegger, and C. Steger, "Beyond dents and scratches: Logical constraints in unsupervised anomaly detection and localization," *International Journal of Computer Vision*, vol. 130, no. 4, pp. 947–969, 2022.

[69] L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 11, pp. 4037–4058, 2020.

[70] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, "Self-supervised learning: Generative or contrastive," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 1, pp. 857–876, 2021.

[71] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," in *International Conference on Learning Representations*, 2018.

[72] A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon, "A survey on contrastive self-supervised learning," *Technologies*, vol. 9, no. 1, p. 2, 2020.

[73] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the bayesian restoration of images," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 721–741, 1984.

[74] J. Yi and S. Yoon, "Patch svdd: Patch-level svdd for anomaly detection and segmentation," in *Proceedings of the Asian conference on computer vision*, 2020.

[75] C. Huang, J. Cao, F. Ye, M. Li, Y. Zhang, and C. Lu, "Inverse-transform autoencoder for anomaly detection," *arXiv preprint arXiv:1911.10676*, vol. 2, no. 4, 2019.

[76] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[77] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[78] B. Hanin, "Which neural net architectures give rise to exploding and vanishing gradients?," *Advances in neural information processing systems*, vol. 31, 2018.

[79] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, 2006.

[80] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training," in *Artificial intelligence and statistics*, pp. 153–160, PMLR, 2009.

[81] J. Hyun, S. Kim, G. Jeon, S. H. Kim, K. Bae, and B. J. Kang, "Reconpatch: Contrastive patch representation learning for industrial anomaly detection," *arXiv preprint arXiv:2305.16713*, 2023.

[82] M. Yang, P. Wu, and H. Feng, "Memseg: A semi-supervised method for image surface defect detection using differences and commonalities," *Engineering Applications of Artificial Intelligence*, vol. 119, p. 105835, 2023.

[83] S. Niu, Y. Liu, J. Wang, and H. Song, "A decade survey of transfer learning (2010–2020)," *IEEE Transactions on Artificial Intelligence*, vol. 1, no. 2, pp. 151–166, 2020.