LU TP 19-15
May 2019

Deep Learning techniques for classification of data with missing values

Leo Zethraeus

Department of Astronomy and Theoretical Physics, Lund University

Master thesis supervised by Patrik Edén

Abstract

Two deep learning techniques for classification on corrupt data are investigated and compared by performance. A simple imputation before classification is compared to imputation using a Variational Autoencoder (VAE). Both single and multiple imputation using the VAE are considered and compared in classification performance for different types and levels of corruption, and for different sample sizes for the multiple imputation. Two main corruption methods are implemented, designed to test the classifiers for the cases of data missing at random or data missing not at random. The MNIST data set is used for evaluating performance of the different techniques. It is shown that a Multilayer Perceptron (MLP) trained on VAE imputations outperform a MLP using a simple imputation for all tested levels of corruption. A Convolutional Neural Network (CNN) classifier trained with the simple imputation outperforms both the MLP and the VAE classifier on MNIST. This is expected since it is designed to perform well on data sets with high local correlation, like image data sets, whereas the MLP and the VAE classifiers can generalize to other types of data. The reasons for the observed performance of the different techniques and possible implications are discussed.

Popular Science Description

Abstract thought and creative computers – Do androids dream of electric sheep?

Imagine you find a puzzle in a bag in your attic. You don't know who left it there and the box is missing, so you have no idea what it should depict. Being an amateur detective, this only intrigues you more, and you try to piece it together to see what it resembles. After hard work you come to the conclusion that almost half of the pieces are missing. Can you still guess what the picture might be depicting? It should depend on how many and which pieces that are missing, but in many cases, you could probably do a decent job. Take a look at this picture:

You could probably tell that it shows a cup of coffee, even though half of it is missing. Human vision and object recognition is very good, indeed. In recent years, computers using artificial neural networks (abbreviated ANN), a technique inspired by how the human brain works, have shown good performance when it comes to recognizing objects in images – sometimes even on par with human performance - but if the object is seen from an unfamiliar angle, or just part of the object is shown, the task is much more difficult. The way humans and computers are able to relate new objects to familiar objects is through the process of abstraction. We find similar features and repeating patterns, and classify objects or impressions according to abstract categories. A certain kind of ANN, called an autoencoder, forces the computer to "think abstractly". That way it performs much better when classifying objects (a sheep doesn't have to have four legs, it might be enough that it's woolly). But what to do when some pixels in the image are missing (or pieces in the puzzle)? Should it try to classify it anyway or should it try to first fill in the blanks, and then determine the object? I work with a particular kind of autoencoder, a "variational autoencoder", that is able to not only recognize general features but also be "creative". After showing it many pictures of sheep, it can draw a picture that it hasn't seen before, but that still looks like a sheep. In a sense, the computer dreams of sheep. With such an imaginative tool, I intend to determine how puzzle-solving should be done even when pieces are missing. We humans are good at solving puzzles, often better than computers, but we lack something that computers have, endurance. Even the most persistent human puzzle solver would get tired after trying to classify thousands of images, but once an ANN has been properly trained for the task, it can run day and night without a break and give a systematic answer every time.

Contents

Abbreviations

ANN
Artificial Neural Network

MLP
Multilayer Perceptron

CNN
Convolutional Neural Network

VAE
Variational Autoencoder

dVAE
Denoising Variational Autoencoder

ELBO
Estimated Lower Variational Bound

KL-divergence
Kullback-Leibler divergence

1 Introduction

Background: Missing Data

Statistical models are ultimately limited by the quantity and quality of the data sets they are trained on. A common problem when dealing with large data sets are that some values are missing. It could be due to broken sensors leading to corrupted pixels in images or sensitive questions in surveys that respondents sometimes choose to leave unanswered. Whatever the reason, they affect the statistics of the data set and need to be dealt with. There are two ways to deal with missing data:

One way is to delete the samples that contain missing data (or the variable that contain missing data in many samples). Deleting rows or columns (corresponding to samples or variables) in the data set might be fine if the number of samples with corrupted data is small, or if the variable that contains missing data in many samples can be considered unimportant for the results. But if the corruption is high, deleting samples or variables can severely bias the computed statistic and reduce statistical power [1], making the model produce false predictions.

The other way to deal with missing values is to replace them, in one way or another, with some reasonable estimate of their value. The art of replacing missing values is called *imputation*, and can be separated into two main categories: single or multiple imputation. For single imputation, one value is set for each missing data point before calculating the statistic of interest. In multiple imputation, many different values are made for each missing data point, yielding multiple data sets. The statistic of interest of the data set is then calculated for each of the various imputations, and the mean of the statistic of interest for each imputed data set is calculated. The latter also has the advantage of allowing estimation of uncertainty in the computed statistic, from the induced variance of the multiple imputations. Because of the latter, multiple imputation is generally preferred when possible in practice.

In this thesis, the statistic of interest is classification performance of common deep learning models on a labeled corrupt data set. Two types of corruption methods are implemented to mimic two different categories of missing data: *missing at random* and *missing not at random*.

There are many methods for doing single and multiple imputation, for example k-nearest neighbors [2] and MICE [3], but the focus of this thesis is on methods using Deep Learning.

Deep Learning techniques for imputation and classification

In recent years, Deep Learning techniques have been successfully applied to a broad range of problems, including classification and imputation. The gold standard data set for comparing performance of different Deep Learning techniques is the MNIST data set, a set of images of handwritten digits [4]. For classification purposes, the task is to predict the corresponding digit using the pixel values. This is an example of *supervised learning*, where

the output of the model should predict target labels. An example of *unsupervised learning* would be imputation, as in imputing pixel values on a corrupted version of an MNIST image. Often, labels are the main restricting factor of data sets. Thus unsupervised learning has an appealing benefit: it can train on the data set itself, without requiring target labels.

The most common approach to imputation, in Deep Learning parlance called *denoising*, is to use different variants of the so called denoising autoencoder (dAE), which will be described in the theory section. The dAE is however, restricted to single imputation. The Variational Autoencoder (VAE) was invented by Kingma et al. [5] as a way to do variational inference on data distributions with the help of Deep Learning techniques. The VAE is similar to the autoencoder, but it can learn a distribution of hidden inferred variables from an observed data distribution, which enables multiple imputation to be performed. With the VAE as a tool for imputation, the next question is: how does multiple imputation affect classification performance on corrupt data? Can it improve classification performance compared to single imputation, and for what types and levels of corruption would it be useful for in practice? Finally, how should we actually implement and train a Variational Autoencoder for imputation and classification?

2 Theory

The theory necessary for understanding the main tools used in the thesis will be described throughout the following section. A short remark on notation: z, μ, σ, x and $\tilde{x}$ are all vectors of unspecified dimensions (either input dimension or a hyperparameter). If an index is written out, as in x_i, it is a scalar that represents the value of the i:th component (or pixel) of the vector x. In the context of MNIST, the vector x is the 784-dimensional vector representing the collection of pixels in one image. Basic overviews of the MLP and the CNN along with their associated terminology are presented in appendices A and B.

2.1 Autoencoders

Autoencoders are a special kind of ANN that are tasked with efficiently representing a high dimensional distribution on a lower dimensional *latent* space. The network consists of two parts; an *encoder* that is tasked with projecting the input to the latent space; and a *decoder* that reconstructs the input data given a position vector in the latent space. The objective, or *loss* function should measure how well the input is reconstructed by the output. This could in practice be the mean squared error between the input and output:

$$E = \frac{1}{N}\sum_{i}^{N}(x_i - \tilde{x}_i)^2 \tag{2.1}$$

or some other measure of distance between input x and output $\tilde{x}$, such as binary cross-entropy:

$$E = -\frac{1}{N}\sum_{i}^{N} x_i \log(\tilde{x}_i) + (1 - x_i)\log(1 - \tilde{x}_i) \tag{2.2}$$

The setup is shown in Figure 1.

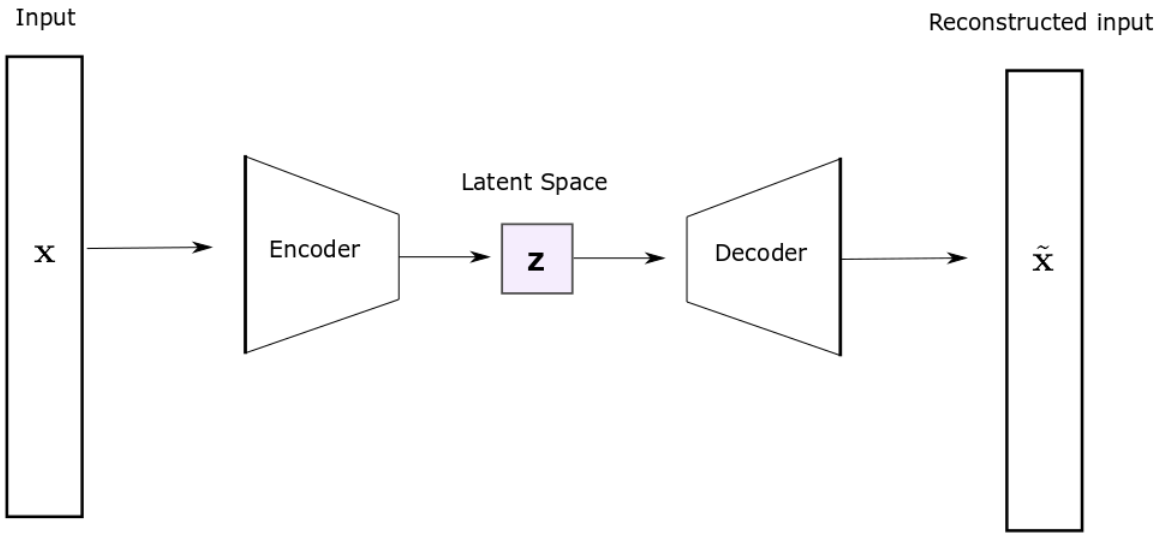


Figure 1: The structure of the Autoencoder.

The compression of the input to a lower dimensional space enforces the encoder to identify the most important features of the input. The dimension of the latent space vector z is a hyperparameter that can be chosen in order to maximize some objective, as long as it is smaller than the input, otherwise the encoder could simply learn an identity mapping. The encoder and the decoder could in principle consist of any kind of ANN architecture, as long as identity mappings are avoided.

2.1.1 Denoising Autoencoder

As the objective of the encoder is to extract important features from the distribution of the input data, it can be used as a tool to reconstruct clean input from a corrupted version. A denoising autoencoder does this, but rather than comparing the corrupted input $\hat{x}$ to output $\tilde{x}$, it compares the output with the clean input, and the loss function is adjusted accordingly. In this thesis *denoising* refers to imputation of data corrupted as described in section 3.2.

2.2 Variational Autoencoder

The Variational Autoencoder, first introduced by Kingma et al in [5], uses a similar setup as the *vanilla* autoencoder described in the previous section. It has an encoder part and a decoder part and it is tasked with reconstructing the input. A disadvantage of the vanilla autoencoder is that the latent space is not well structured, in the objective there is no reference to the structure of the latent space. The main motivation behind the VAE is to add some structure to the latent space in a way that makes sampling from it easy

and meaningful. The VAE should in practice make sure that similar images are located nearby in the latent space, and preferably, the distribution should be easy to sample from. That can be achieved if the true distribution is approximated by a well known parametric distribution, like a multivariate gaussian distribution, and the objective is to find the parameters (the mean and the variance) that best approximate the true distribution. The setup is illustrated in Fig. 2.

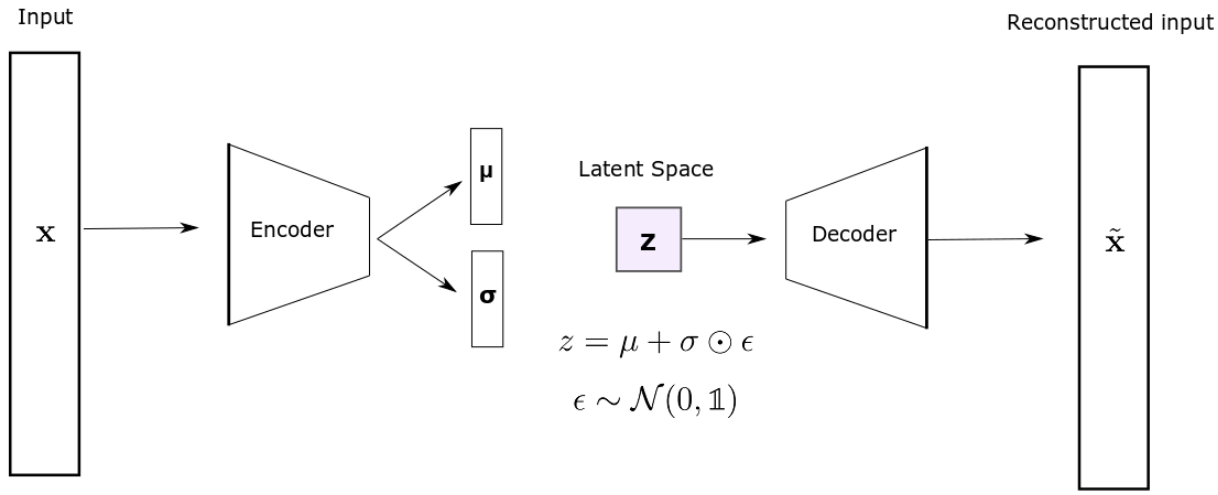


Figure 2: Variational autoencoder architecture. Note that μ, σ and ϵ are all L-dimensional, where L is the dimension of the latent space. The dot $\odot$ in this case means an element-wise product.

The VAE is a deep learning approach to Bayesian Variational Inference, in which you try to model an intractable posterior distribution $p(z|x)$ with latent variables z, by an approximate distribution $q_{\phi^\star}(z|x)$, where q is a parametric distribution that can be differentiated and easily evaluated at any point. The objective is to find the parameters $\phi^\star$ from a family of distributions q_ϕ that make $q_{\phi^\star}$ similar to the intractable true posterior distribution $p(z|x)$. The difference between the approximate q and the true posterior p is quantified by the Kullback-Leibler divergence, denoted as:

$$\mathcal{D}_{\mathrm{KL}}(q||p) = \mathbb{E}_{z\sim q} \log\left(q/p\right)$$

where $\mathbb{E}_{z\sim q}$ is the expectation value with respect to the distribution $q(z|x)$. With a quantification like this, the problem of finding the intractable posterior $p(z|x)$ can be turned into an optimization problem: finding the parameters $\phi^\star$ that minimize the KL-divergence between the approximate and true posterior. Hence the Variational part of the name.

If only samples from the true posterior distribution would have been of interest, some other method like Markov Chain Monte Carlo (MCMC) could be used in theory, but practically MCMC is very inefficient for high-dimensional distributions. Variational Inference overcomes this problem by instead finding a good approximate posterior distribution.

2.2.1 The VAE objective

The following derivation of the VAE objective loosely follows the one done in [5]. It is included here for completeness. The observed data x is assumed to be distributed according to a distribution $p(x)$. Following the inference procedure, it is assumed that there exists some latent variables z that describe the underlying process behind the observed data x so that the observed distribution can also be fully described by the joint distribution $p(x, z)$. The log-likelihood of observing the data x is then:

$$\log p(x) = \log \left[\int p(x, z) dz \right] \tag{2.1}$$

The expression above (2.1) can be rewritten by introducing another distribution $q(z|x)$, so far not specified, but it will be used to approximate the posterior distribution $p(z|x)$.

$$\log p(x) = \log \left[\int p(x, z) dz \right] = \log \left[\int p(x, z) \frac{q(z|x)}{q(z|x)} dz \right] \tag{2.2}$$

$$= \log \left[\mathbb{E}_{z \sim q(z|x)} \left[\frac{p(x, z)}{q(z|x)} \right] \right] \tag{2.3}$$

As $f(X) = \log(X)$ is concave, the expression (2.3) obeys *Jensen's inequality* [6]:

$$\log p(x) \geq \mathbb{E}_{z \sim q(z|x)} \left[\log \left[\frac{p(x, z)}{q(z|x)} \right] \right] \tag{2.4}$$

$$= \mathbb{E}_{z \sim q(z|x)} \left[\log \left[p(x, z) \right] \right] - \mathbb{E}_{z \sim q(z|x)} \left[\log \left[q(z|x) \right] \right] \tag{2.5}$$

Rewriting the joint distribution $p(x, z)$ in terms of a prior and a conditional on z,

$$p(x, z) = p(z)p(x|z)$$

the inequality becomes

$$\log p(x) \geq \mathbb{E}_{z \sim q(z|x)} \left[\log \left[p(x|z) \right] \right] - \mathbb{E}_{z \sim q(z|x)} \left[\log \left[q(z|x) \right] \right] + \mathbb{E}_{z \sim q(z|x)} \left[p(z) \right] \tag{2.6}$$

$$= \mathbb{E}_{z \sim q(z|x)} \left[\log \left[p(x|z) \right] \right] - \mathbb{E}_{z \sim q(z|x)} \left[\log \left[q(z|x)/p(z) \right] \right] \tag{2.7}$$

$$= \mathbb{E}_{z \sim q(z|x)} \left[\log \left[p(x|z) \right] \right] - \mathbb{KL} \left(q(z|x) \| p(z) \right) \tag{2.8}$$

were the definition of the Kullback-Leibler (KL) divergence between two distributions is used in the last step (2.8). The KL divergence is non-negative, and measures how different two distributions are. If $q(z|x) = p(z)$ then the KL divergence is zero. The right hand side of (2.8) can be interpreted as an estimated lower bound (ELBO) of the log-likelihood of the observed distribution. The distribution $q(z|x)$ is an approximate posterior and it can be chosen so as to maximize the ELBO in (2.8).

In practice, a differentiable parametric distribution is preferred, because then stochastic gradient descent methods can be applied. For the purpose of this thesis, a multivariate gaussian distribution is used. Using a gaussian approximate posterior is practical for two reasons: a parametric approximate posterior such as the multivariate gaussian enables the use of fast optimization techniques, as will be described in section 2.2.2. Finally, for the purpose of multiple imputation, a distribution with a clear interpretation of location and scale is useful.

If both the prior $p(z)$ and the posterior $q(z|x)$ are multivariate gaussian distributions (the prior being a gaussian $p(z) = \mathcal{N}(0, \mathbb{1})$), the KL-divergence can be solved analytically, allowing for fast computation. The derivation is shown in Appendix C, and the result is:

$$\mathbb{KL}\left(q(z|x)\|p(z)\right) = -\frac{1}{2}\sum_{l=1}^{L}\left(1 + \log\sigma_l^2 - \mu_l^2 - \sigma_l^2\right) \tag{2.9}$$

where L is the dimension of the latent space. Note that μ and σ are restricted by the KL-divergence on unequal terms, due to the presence of the $\log\sigma^2$ term. For a simple overview of how σ contribute to the KL-divergence for fixed μ, see Fig. 3.

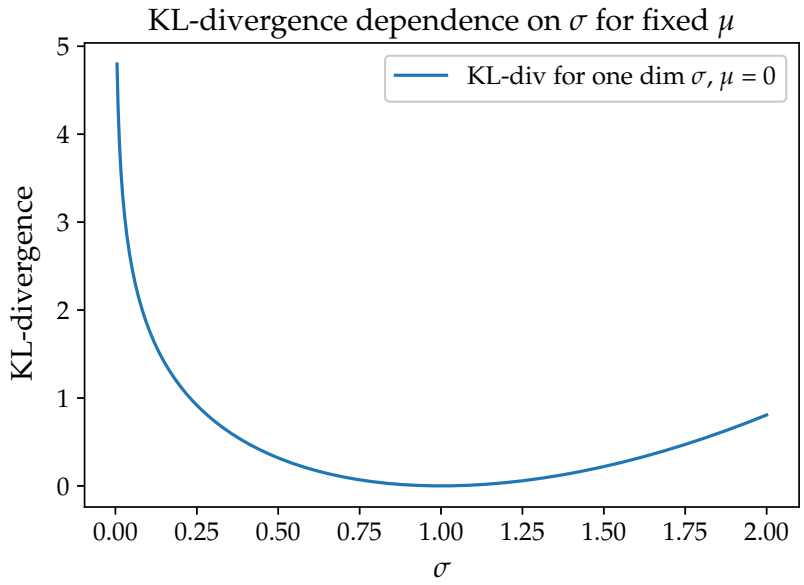


Figure 3: KL-divergence dependence on a single dimension of σ. The mean μ is held fixed at $\mu = 0$. Note that μ is allowed to take negative values and is symmetric around $\mu = 0$ (not shown in the figure, but its contribution to the KL-divergence is simply $\mu^2/2$), whereas low values of σ are greatly inhibited by the $\log\sigma^2$ term in the KL-divergence.

The first term in Eq (2.8) is a reconstruction loss that should measure how well the decoder manages to reconstruct the input x, for a set of sampled points z in latent space sampled from $\mathcal{N}(\mu, \sigma)$. If a single imputation is done per image, this is identical to the

objective in the vanilla autoencoder. The reconstruction loss can be measured in the same way as for the vanilla autoencoder, for example with a mean squared error or a binary-cross entropy loss function. In this thesis, the reconstruction loss used will be binary-cross entropy, defined in (2.1). As demonstrated in this section, the VAE objective has a clear probabilistic interpretation. The main difference between the vanilla autoencoder objective and the VAE objective, apart from the interpretation, is the addition of the KL-divergence, which acts as a regularizing term that inhibits the latent space structure from deviating too much from a prior unit variance gaussian. If only $z = \mu$ points are sampled, the VAE can be interpreted as a deterministic vanilla autoencoder, making it useful for comparing the two approaches to imputation.

2.2.2 The Reparametrization Trick

The fact that sampling is done in the middle layer leads to stochasticity in the network, which prevents training by back-propagation (each layer needs to be able to be expressed as a deterministic function of the previous layers in order to take chain-rule derivatives). The main contribution of [5] was the following observation: The VAE can be divided into two fully deterministic networks, and the stochasticity can be moved to the input of the second network (the decoder), so that back-propagation can be performed. Instead of sampling $z \sim \mathcal{N}(\mu, \sigma)$, z is *reparametrized* and calculated as $z = \mu + \sigma \odot \epsilon$ with $\epsilon \sim \mathcal{N}(0, \mathbb{1})$, which is equivalent due to the properties of the multivariate gaussian distribution, but it removes the stochasticity from the hidden nodes in the network, allowing for back-propagation.

2.2.3 Denoising Variational Autoencoder

To implement a denoising VAE, the reconstruction loss in the VAE objective is adjusted to compare the output to the uncorrupted original images, rather than the input. The KL-divergence term is unaffected.

2.2.4 Using the Denoising Variational Autoencoder for Classification

Once the dVAE is trained, the inference network, or encoder part of the dVAE, outputs a μ and a σ for each input:

$$\text{encoder}(\text{input}) \rightarrow (\mu, \sigma)$$

these may be thought of as the center and scales of an ellipsoid in the latent space manifold.

From here on, there are two options:

- Ignore σ and send $(\mu, 0)$ and thus $z = \mu$ through the generative network, or decoder and obtain one imputation I_0.

$$(\mu, 0) \rightarrow z_0 = \mu$$

$$\text{decoder}(z_0) \rightarrow I_0$$

- Sample one or many z from a multivariate gaussian distribution with mean μ and variance σ and send each one through the decoder to obtain many imputations corresponding to the same input I_1, I_2, I_3...

$$(\mu, \sigma) \rightarrow z_k \sim \mathcal{N}(\mu, \sigma)$$

$$\text{decoder}(z_k) \rightarrow I_k$$

  and repeat M times to obtain M imputations I_k.

With the ability to pick multiple samples from the μ and σ obtained from each input, the multiple imputations per input can be sent through a classifier of choice[1], to produce a predicted label, or a prediction vector of the possible labels.

Using the so called one-hot encoding, a prediction vector with ten labels can be represented as $p = (0, 0.2, 0, 0, 0, 0, 0.8, 0, 0)$ corresponding to 80 % probability of the label corresponding to index 8. In the case of MNIST, 0 is usually assigned to the first index, so this could correspond to an uncertainty between predicting the digit 7 or 1.

In the end, each input produces many output prediction vectors. Taking the mean of these new prediction vectors will produce a new output prediction. This method should be compared to simply taking the $\sigma = 0$ imputation for each input, corresponding to the best single imputation, analogous to the output of a vanilla autoencoder. In other words, the dVAE makes it possible to compare the mean of a classifier output to the classifier output of the "mean" ($z = \mu$) imputation. Case one corresponds to:

$$\text{classifier}(I_0) \rightarrow p_0$$

and case two:

$$\text{classifier}(I_k) \rightarrow p_k$$

and repeat M times to obtain M prediction vectors p_k. Then the multiple imputation prediction is calculated as:

$$p_{\text{multiple}} = \frac{1}{M} \sum_{k=1}^{M} p_k.$$

As the p_k are identically distributed and independent, by the central limit theorem the fluctuations in p_{multiple}, $\sigma_{p_{\text{multiple}}}$ should decrease as $\sigma_{p_k}/\sqrt{M}$, with σ_{p_k} being the fluctuations in a single sample (not necessarily $z = \mu$) prediction.

The two different approaches to classifying can be compared for different types and levels of corruption, and different sample sizes M. Also, the techniques are compared to classifying directly on the corrupted data,

$$\text{classifier}(\hat{x}) \rightarrow p_{\text{dc}}$$

for different types of classifiers.

[1] See details in section 3.3

Note that there are in principle other ways to construct the multiple imputation prediction. For example, the predictions of the sampled imputations could be calculated each one all the way to the end, computing the most probable label from each of their prediction vectors and then counting the number of occurrences of different labels, and assigning the most frequent prediction to p_{multiple}. A drawback with that approach is that information is lost when the most probable prediction from each prediction vector is singled out. Using the mean of the prediction vectors ensures that information about uncertainty is kept until the very end of the classification process.

3 Method

As the goal of this investigation is to evaluate and compare performance of classifiers on different types of imputations, care must be taken to avoid overtraining any of the models on the data set used.

The details are described in the subsections below, but a brief summary of different methods used to avoid overtraining the models can be presented:

- L2-norm regularization is added to the loss function for the Multilayer Perceptron (MLP) networks (or partial networks in VAE + MLP).
- Dropout is used on some layers in the Convolutional Neural Networks (CNN).
- Model selection is done on a large validation set, separate from the test set.

3.1 Data set

The MNIST data set [4] is used to evaluate performance of the different classifiers. Note that the methods used are more general and could be applied to data sets not consisting of images. Any kind of data set with missing data could be used, but for the sake of evaluating performance of ANNs, MNIST is practical because it is a large labeled data set, it is already stratified and it serves as the gold standard for evaluating performance of ANN classifiers in the deep learning community.

Nevertheless, the fact that MNIST is an image data set can affect which types of network architectures that perform better than others. For example the Convolutional Neural Network (CNN) is specifically designed to perform well for classification on images, but might not work well on other types of data.

3.2 Corruption methods

In real-world problems, missing data is usually categorized as either *missing at random* or *missing not at random*. To mimic and evaluate the effect of these two scenarios, two

corruption methods have been constructed and used to test the classifiers under investigation: *Random Pixels*: randomly chosen pixels set to white (1, as the values are normalized to be in the range $[0, 1]$ where 0 is black and 1 is white and any float value in between is allowed). The second corruption method, here called *Vertical Lines*, is where a number of random columns in the image are set to white. These corruption methods are illustrated in the figure below:

Figure 4: Left: original image, middle: *Random Pixels* 50% corruption; right: *Vertical Lines* 50% corruption.

A naive implementation of the corruption methods above would be to draw 1 pixel (or column) among the 784 N times and set the value to 1, but for large N, corresponding to more corruption, many pixels are drawn more than once, and so the fraction of corrupted pixels is not simply $N/784$. Each time a pixel is drawn, the probability that a chosen pixel is corrupted is $1/p$ with p the number of pixels in the image, in the case of MNIST $p = 784$. So the probability of a pixel remaining uncorrupted after D draws is:

$$\left(1 - \frac{1}{p}\right)^D \approx e^{-\frac{D}{p}}$$

In other words, to achieve a given corruption level cl (for example $cl = 0.5$ as in the images above (Fig. 4) with a corruption of 50 %), the number of random draws required on average is:

$$D = -p \log(1 - cl)$$

For the *Vertical Lines* method, p is simply replaced by the number of columns.

The corruption methods can also be regarded as a very simple type of imputation, as in a real world problem, missing data means there is no value at all. As the more advanced imputation methods using the VAE still need a value for all the pixels, this basic imputation is done before any other. In practice, as the full uncorrupted data set is available, the corruption and imputation is here done in one step. Still, it is important to note that an imputation is used on all the cases, and the value chosen for the corrupted pixels is quite arbitrary. For example: rather than setting random pixels to white, a mean value could be used. The same basic imputation is used for all models investigated in this thesis, so it should not affect the results.

3.3 Network Architecture

Two main types of classifiers are used throughout this thesis. The first uses a classifier immediately on the corrupted (and trivially imputed) input, from now on denoted Type 1 classifier, unless explicitly specified. The Type 1 classifier architectures under consideration are a Multilayer perceptron (MLP) and a Convolutional Neural Network (CNN). The specific architectures (the number of hidden layers and nodes in the MLP, for example) and various hyperparameters are determined by model selection, described in section 3.5. The other type of classifiers under consideration are the VAE + MLP or VAE + CNN, denoted Type 2 classifiers, whenever not explicitly specified. To clarify, these are using the imputations of the VAE and then classifying the VAE-imputed output using a Type 1 classifier, and both sorts are considered. See Fig. 7 for the Type 2 setups.

VAE

The specific architectures of the VAE under consideration are encoder and decoders with equal number of hidden dense layers with ReLu activation. The output layer uses a sigmoid activation function. The hyperparameters of the VAE are:

1. The number of layers in the encoder/decoder
2. The number of nodes in each layer
3. The learning rate of the Adam optimizer
4. An L2-norm parameter
5. The dimension of the latent space

the values of which are determined by model selection. An example architecture is shown in Fig. 5.

MLP

The MLP uses dense layers with ReLu activation and softmax in the output layer. The hyperparameters to be determined by model selection are:

1. The number of layers
2. The number of nodes in each layer
3. An L2 norm parameter

an example architecture is shown to the left in Fig. 6.

CNN

The CNN uses two convolutional layers with ReLu activation, the number of filters in each layer are held fixed. The two convolutional layers are followed by a dense layer with ReLu activation. Dropout is used before and after the dense layer. The output layer uses a softmax activation function. The hyperparameters for the CNN are:

1. Amount of dropout before the dense layer
2. Amount of dropout after the dense layer
3. The number of nodes in the dense layer

and are determined by model selection as well. The architecture of the CNN is shown to the right in Fig. 6.

Type 2

The Type 2 classifiers uses combinations of the architectures described above, VAE + MLP and VAE + CNN. The training procedure and model selection for the Type 2 classifiers are described in sections 3.4 and 3.5.

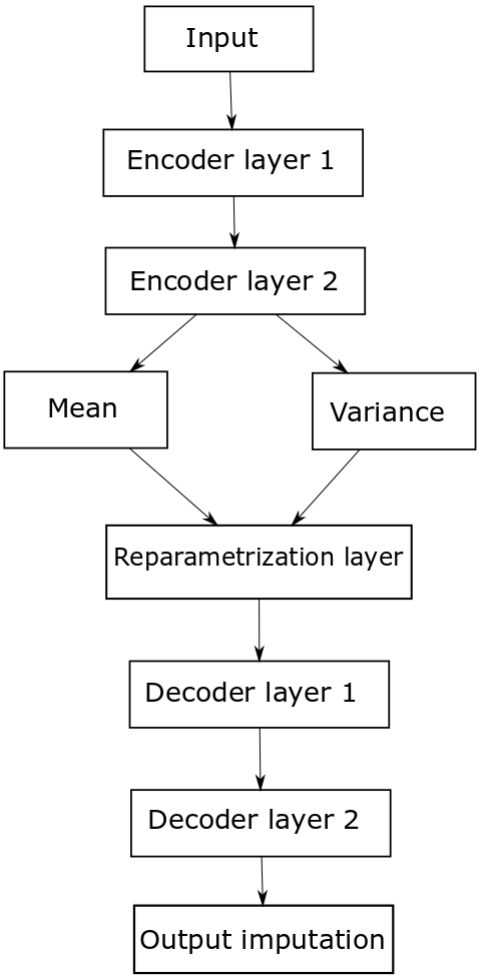
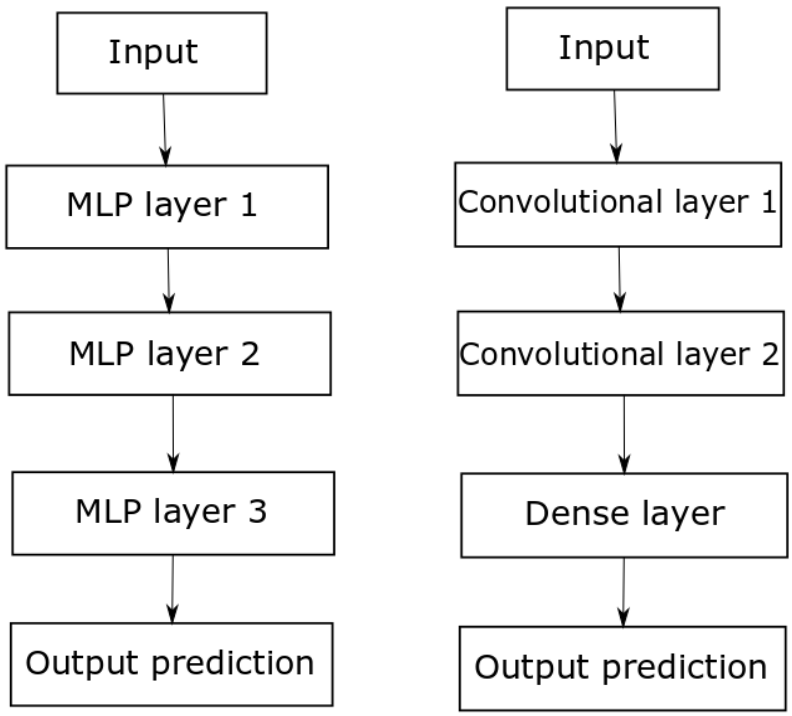
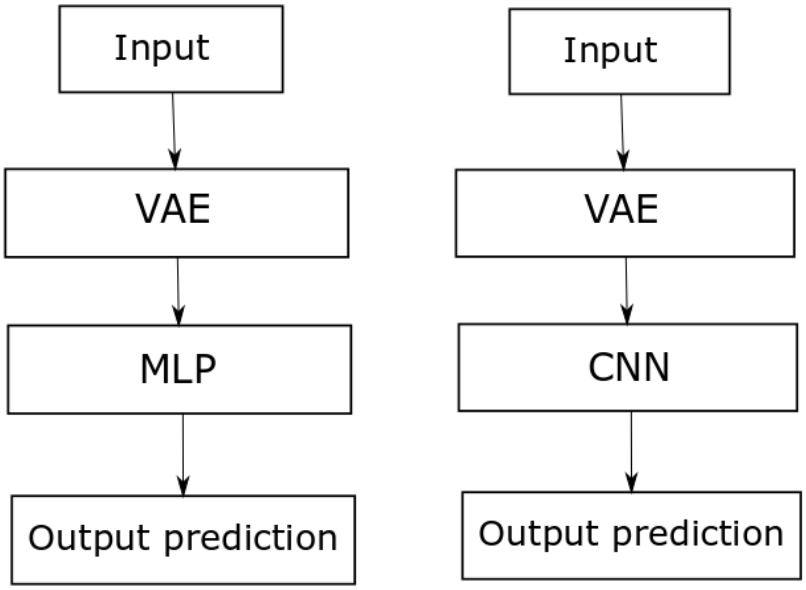


Figure 5: An example architecture of a VAE used, here with two dense layers in both the encoder and decoder.



(a) MLP classifier (b) CNN classifier

Figure 6: Examples of Type 1 classifier setups illustrated.



(a) VAE+MLP classifier (b) VAE+CNN classifier

Figure 7: The Type 2 classifier setups illustrated.

3.4 Training procedure

The training procedure for the Type 1 classifiers is straightforward, using the AdaDelta algorithm [7] for optimization of the network weights. The loss function used is categorical cross-entropy. For the Type 2 classifiers, the VAE is first trained using the VAE objective/loss function described in section 2.2.1, with the Adam optimizer [8]. The learning rate considered a hyperparameter to be determined during model selection. After the VAE has been trained, the weights are fixed, and a new network is constructed on top of the fixed VAE, making it a VAE with fixed weights plus a classifier with trainable weights. The VAE+classifier network is then trained in the same way as the Type 1, with categorical cross-entropy as the loss and AdaDelta as the optimizer. The hyperparameters like the number of layers and the number of nodes in each layer in a MLP classifier, the amount of dropout in CNN-layers etc are determined anew during model selection.

3.5 Model Selection

The four main types of classifiers come with many hyperparameters. It is desirable to be able to efficiently search through many different combinations of values. To this end, the random search algorithm is used. It is more efficient than the grid search algorithm, as demonstrated in [9], enabling search of a wider range of values in limited computational time. The random search algorithm consists of choosing allowed (or computationally feasible) ranges for the hyperparameters, and then rather than looping through all possible combinations of hyperparameters as in the grid search algorithm, the combinations are picked at random.

If each VAE with different sets of hyperparameters would be considered with model selection for each possible classifier attached to it (also with model selection of hyperparameters), the number of possible combinations and thus validation loops required would be very large, which would require a lot of computational time[2]. Therefore, a simplifying assumption is made: that the VAE with the best ELBO should correspond to the best imputator for classification. This assumption is reasonable as the ELBO quantifies how good the reconstruction is, and a bad reconstruction should lead to poor performance in classification. To further justify this assumption a short comparison for a given level of corruption of the performance of a few different models using VAEs with different ELBO's on classification is presented in Table 3 in the result section.

The model selection thus consists of the following steps for Type 1 classifiers:

- A few different architectures are considered, 2, 3 and 4 layers in the MLP, for each architecture a validation loop is performed.
- Ranges of the different hyperparameter values are set for a given architecture.

[2]One validation loop of a 100 random picks of sets of hyperparameters for a VAE takes a few hours on a GTX 1070 graphic card.

- A validation loop is performed: for each iteration, a randomly chosen vector in the hyperparameter space is picked, a model is trained with the corresponding values, and the performance is evaluated on a validation set, separate from the training and (the later used) test sets. A hundred iterations are used and early stop is used to reduce computational time as well as to observe the optimal number of training epochs (which can also be considered a hyperparameter).

- 50 000 images are used for the training set, 10 000 for validation and 10 000 for testing. The separation was stratified.

- After the validation loop, the model that performed best on the validation set is selected and trained again on both the training and the validation set, its final performance is evaluated on the test set.

For Type 2 classifiers the procedure is similar, with the following adjustments:

- The VAE are selected by their performance on the ELBO, as previously discussed. It is evaluated on a validation set of 10 000 images, separate from the training and test set (the same way as before for Type 1 classifiers).

- After a validation loop (100 iterations), the best VAE, judged by the ELBO evaluated on the validation set is trained anew, now including the validation set in the training.

- An architecture for the classifier is chosen, the same categories as described above are all checked (2, 3 and 4 layer MLP or CNN) in separate validation loops.

- The weights of the previously trained VAE are fixed, and a validation loop for the hyperparameters for the classifier part of the network, trained on the output of the fixed VAE, is performed.

- The VAE + Type 1 classifier that performs the best on accuracy with the target labels in the validation set is chosen, one for the MLP and one for the CNN.

- The now Type 2 classifier is trained on the training set plus the validation set and is finally evaluated on the test set.

- The finished Type 2 classifier is tested for both single and multiple imputation, taking 0 to 3000 samples, as described in section 2.2.4.

The same validation data that determined model selection for the VAE is used for validation of Type 2 classifiers. Thus only three data partitions are needed. Some arguments why this should not cause overtraining are:

- The objectives of the two are different, in the first validation loop, the VAE is evaluated based on its ability to perform imputation (quantified by the ELBO), whereas in the second loop the VAE + classifier is evaluated on the prediction accuracy of the target labels, which are not included at all in the first loop.

- The validation set is deliberately chosen to be large and stratified in order to prevent overtraining.
- The training of the VAE is carefully monitored in order to avoid overtraining, this can be seen by comparing the ELBO of the model on the validation set to the ELBO on the test set, and they were observed to always be similar.
- L2-norm is used in the classification process, which further reduces the risk of overtraining.

3.6 Evaluating performance

When two networks or imputation methods show similar performance on classification accuracy, a more rigorous approach to comparing them can be used.

If classifier (with or without VAE-imputation) 1 and classifier 2 have nearly the same accuracy, it is necessary to evaluate if the tiny difference between them is significant or due to inherent fluctuations. First, the images where their predictions differ are sorted out. The number of images on which their predictions differ is denoted N. The fraction of the correct predictions of classifier 1 on the images where they differ, to N is denoted x, and will be a stochastic variable from a binomial distribution so that standard deviation $\tilde{\sigma}_x$ and expectation value $\langle x \rangle$ are related by

$$\tilde{\sigma}_x = \frac{\sqrt{\langle x \rangle (1 - \langle x \rangle)}}{\sqrt{N}}$$

The null hypothesis is that $\langle x \rangle = 0.5$, i.e. their performance is equal. Thus

$$\tilde{\sigma}_x = \frac{\sqrt{0.5(1 - 0.5)}}{\sqrt{N}} = \frac{1}{2\sqrt{N}}$$

and the statistic computed for testing the null hypothesis will be:

$$z = \frac{x - \langle x \rangle}{\tilde{\sigma}_x} = \sqrt{N}(2x - 1)$$

Using a normal approximation of the errors, classifier 1 is considered to be better than classifier 2 at z standard deviations. Rather than choosing a specific significance level, the values of the test statistic z are presented in tables.

4 Results

Plots of classification accuracy for the different classifiers are shown in the subsections labeled by type of corruption. The plots depict relative performance of the different classifiers for various levels of corruption, for an increasing number of samples for the multiple

imputation case. Table 3 is included to test the assumption made during model selection that a lower ELBO corresponds to a worse classifier, for one type and level of corruption (*Random Pixels* 70 %), but using VAEs with different results in ELBO trained on the same corrupted data.

Absolute values of classification accuracy are of little interest for the purpose of this investigation, and any of the models could possibly be refined further by trying out even more architectures and hyperparameters during model selection. For this reason the y-axes are not held fixed, to more clearly see the fluctuations of the multiple imputation classifier for different number of samples. Also, if the accuracy of one classifier is severely above or below the accuracy of the VAE classifiers, it is omitted from the plot and stated in the following table.

Multiple imputation visualized

Figure 8 illustrates how a difficult imputation problem can result in multiple imputations with different results (in this case an ambiguity between digits 4 and 9). This clearly shows that *Random Pixels* corruption can make classification and imputation a non-trivial task.

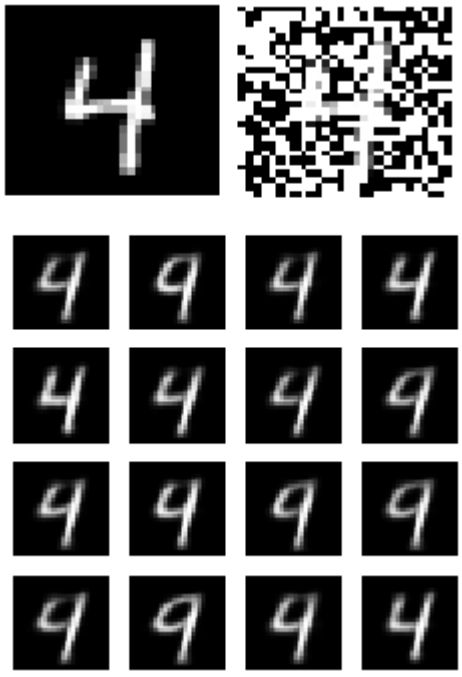


Figure 8: Top left: original image, Top right: the same image corrupted with 50 % *Random Pixels* corruption. Below: Sixteen example imputations from the corrupted version using the VAE for multiple imputation.

4.1 Random Pixels corruption

Three levels of *Random Pixels* corruption are illustrated in Fig. 9. Fig. 10 compares MLP classifiers directly on corrupted images ("MLP classifier, no dVAE"), one imputed image for $z = \mu$ ("Zero variance dVAE") and multiple randomly selected imputed images ("Multisample dVAE"), respectively. The three plots show results for various levels of *Random Pixels* corruption. The dimension of the latent space of the VAE used for imputation are specified for each corruption level. The same VAE are used for both VAE+MLP and VAE+CNN, as explained in section 3.4. The same comparison is shown in Fig. 11 for the CNN-type classifiers, but to resolve the differences between the single/multiple imputation VAE classifiers, the direct CNN accuracies are omitted. The accuracies are summarized in Tables 1 and 2 for a fixed number of samples in the multiple imputation case (3000 samples). The accuracies are the number of correctly classified images divided by the number of images in the test set, 10000. The distribution of digits in the test set according to the Type 2 classifiers are compared to the true distributions for different levels of corruption in Fig. 12.

Figure 9: Example image with *Random Pixels* corruption. First: Original image. Second: 50 % corruption. Third : 70 % corruption. Fourth: 90 % corruption.

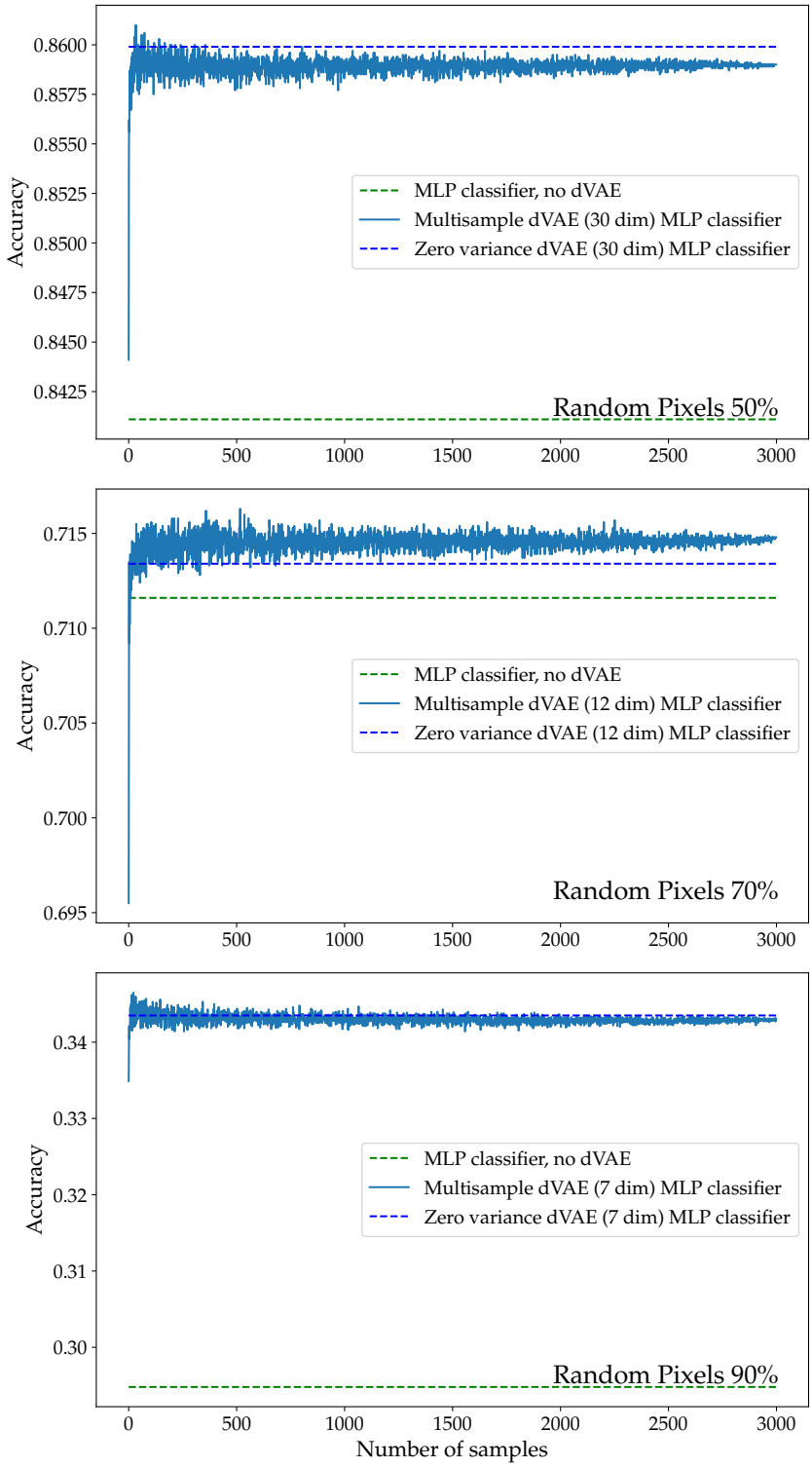


Figure 10: Performance of the direct MLP and the single/multiple VAE+MLP classifiers on *Random Pixels* corrupted images for three levels of corruption and number of samples on the x-axis. Top: 50%. Middle: 70%. Bottom: 90%.

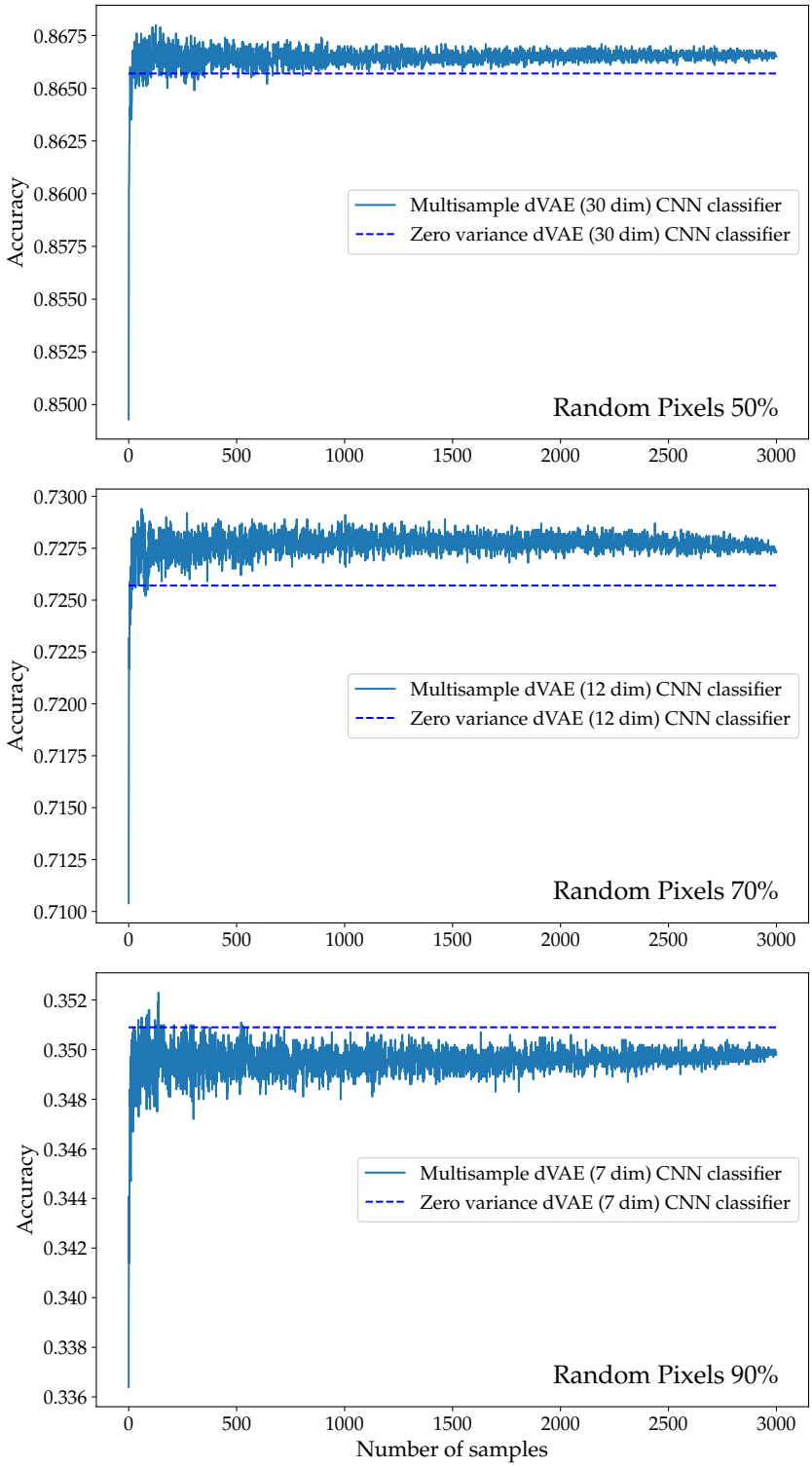


Figure 11: Performance of the single/multiple VAE+CNN classifiers on *Random Pixels* corrupted images for three levels of corruption and number of samples on the x-axis. Direct CNN accuracies are presented in Table 2. Top: 50%,. Middle: 70%. Bottom: 90%.

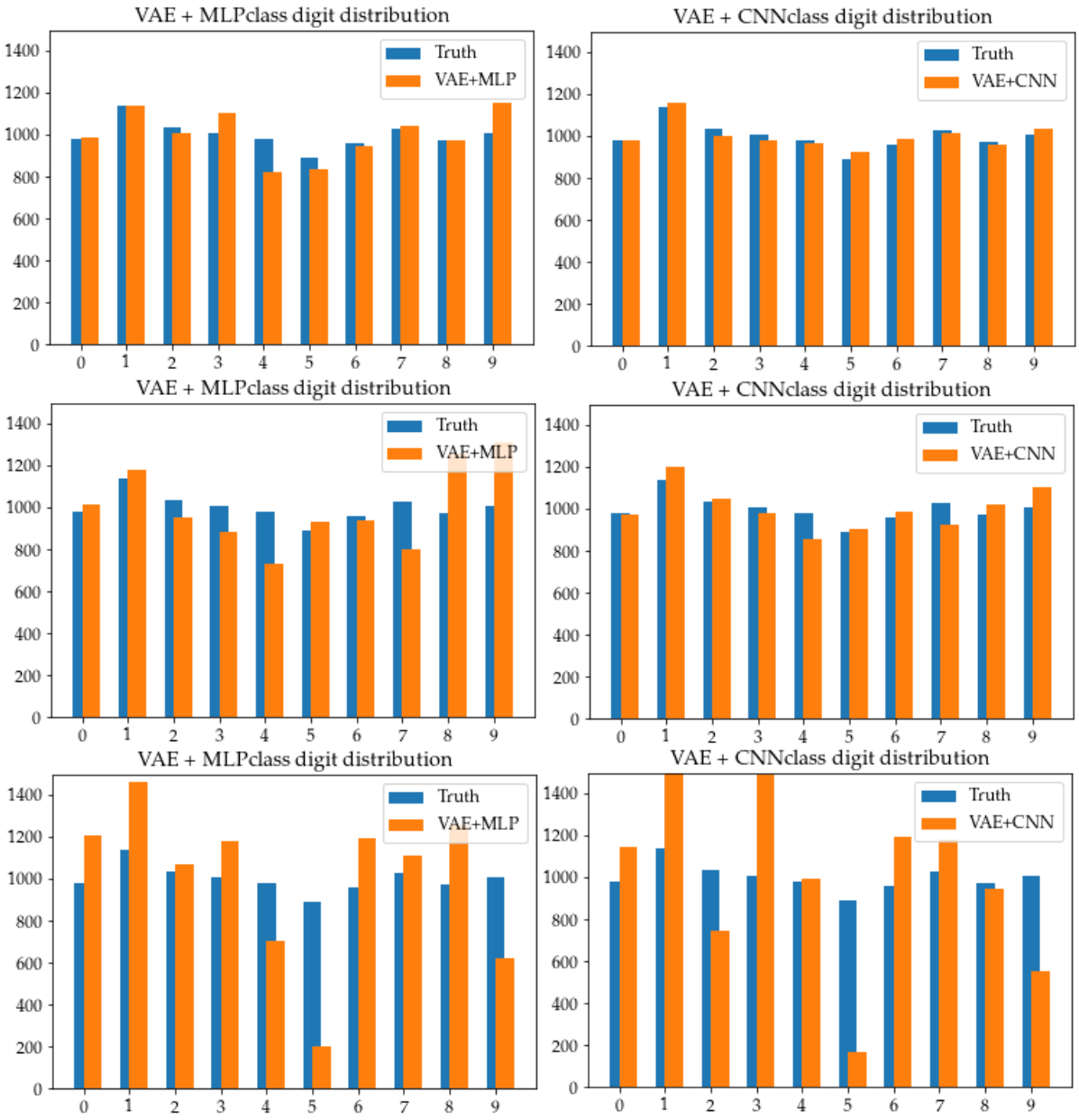


Figure 12: Digit distributions for the *Random Pixels* corruption. Top row represents 50 % corruption, middle 70 % and bottom 90 %. The first column shows the distributions according to the VAE+MLP classifier compared to the true distribution (in blue), and the second to the VAE+CNN classifier.

Table 1: Accuracies and significance tests ($z_{\text{multi/single}}$) for *Random Pixels* corruption levels cl $= 0.5, 0.7, 0.9$. MLP, MLP+VAE single vs 3000 sample multiple compared.

cl	MLP	MLP+VAE single	MLP+VAE 3000 samples	$z_{\text{multi/single}}$
0.5	0.8411	**0.8599**	0.859	-1.08
0.7	0.7116	0.7134	**0.7148**	1.32
0.9	0.2948	**0.3435**	0.3430	-0.36

Table 2: Accuracies and significance tests ($z_{\text{multi/single}}$) for *Random Pixels* corruption levels cl $= 0.5, 0.7, 0.9$. CNN, VAE+CNN single vs 3000 sample multiple compared.

cl	CNN	VAE+CNN single	VAE+CNN 3000 samples	$z_{\text{multi/single}}$
0.5	**0.9766**	0.8657	0.8665	0.96
0.7	**0.9336**	0.7257	0.7273	1.47
0.9	**0.7152**	0.3509	0.3498	-0.83

Clearly, the dVAE multiple/single classifiers outperform the direct MLP classifier. The single versus multiple imputation variants show similar performance, even for a large number of samples, as seen in the small deviations measured by $z_{\text{multi/single}}$ in Tables 1 and 2. The fluctuations in the multiple imputation case seem to decrease as $1/\sqrt{M}$, with M the number of samples, as expected. The performance does not seem to improve for a larger number of samples after a small threshold at around $10 \sim 50$ samples.

In Fig. 11, the same behavior is observed for the single/multiple performance for CNN classifiers. Overall CNN performs better than MLP, and the direct CNN outperforms any other classifier, as seen in Table 2 compared to Table 1. In order to resolve the differences between the single/multiple VAE classifiers, the direct CNN classifier was omitted from the plots in Fig. 11.

The digit distributions presented in Fig. 12 suggest that some digits are harder to reconstruct or classify than others (see digit 5 compared to 8 and 1 for example). Note that though different, the digit distributions of VAE+MLP and VAE+CNN both share the two least classified digits (5 and 9), suggesting that these digits are poorly reconstructed by the dVAE and thus hard to classify regardless the type of classifier. This is also suggested by the fact that classification accuracy is limited by the VAE performance on the ELBO, as seen in Table 3.

Table 3: ELBO and classification accuracy for three different VAE+MLP models, for *Random Pixels* corruption at 70 %.

ELBO	Classification Accuracy
150.7	0.7129
174.3	0.4629
205.8	0.1134

The dependence of classifiers on the VAEs performance as measured by the ELBO, demonstrated in Table 3 further justifies the assumption made during model selection (described in section 3.5): that lower ELBO corresponds to a higher possible classification accuracy. In fact, the classification accuracy seems to be severely limited by the VAE performance on the ELBO.

4.2 Vertical Lines corruption

The *Vertical Lines* corruption is illustrated in Fig. 13. Fig. 14 show the performance of the VAE+MLP classifiers for single versus multiple imputation for three levels of *Vertical Lines* corruption. The direct MLP classifier performs much worse on lower levels of corruption, as can be seen in Table 4. It is omitted from the first two plots to better resolve the relative performances of the single/multiple imputation VAE classifiers. In Fig. 15 the opposite scenario occurs: the CNN classifier accuracy is much greater than the VAE+CNN classifiers, so it is omitted from the plots. All fixed accuracies are presented in Table 5, with the multisample VAE classifier held fixed at 3000 samples. Fig. 16 shows the digit distributions according to the Type 2 classifiers for the *Vertical Lines* corruption, in the same way as described for the *Random Pixels* corruption in Fig. 12.

Figure 13: Example image with *Vertical Lines* corruption. First: Original image. Second: 50 % corruption. Third : 70 % corruption. Fourth: 90 % corruption.

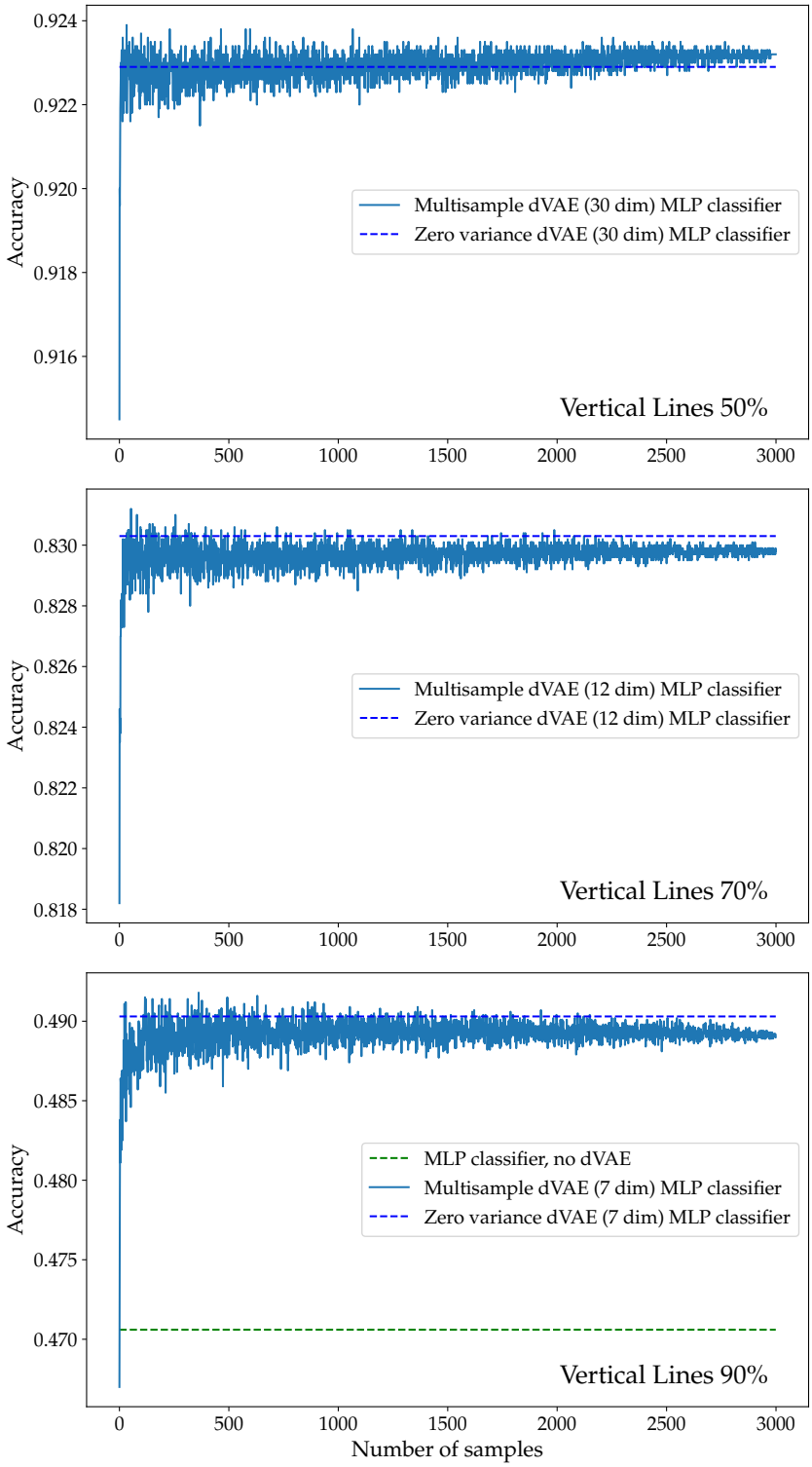


Figure 14: Performance of the direct MLP and the single/multiple VAE+MLP classifiers on *Vertical Lines* corrupted images for three levels of corruption and number of samples on the x-axis. Direct MLP accuracies are presented in Table 4. Top: 50%. Middle: 70%. Bottom: 90%.

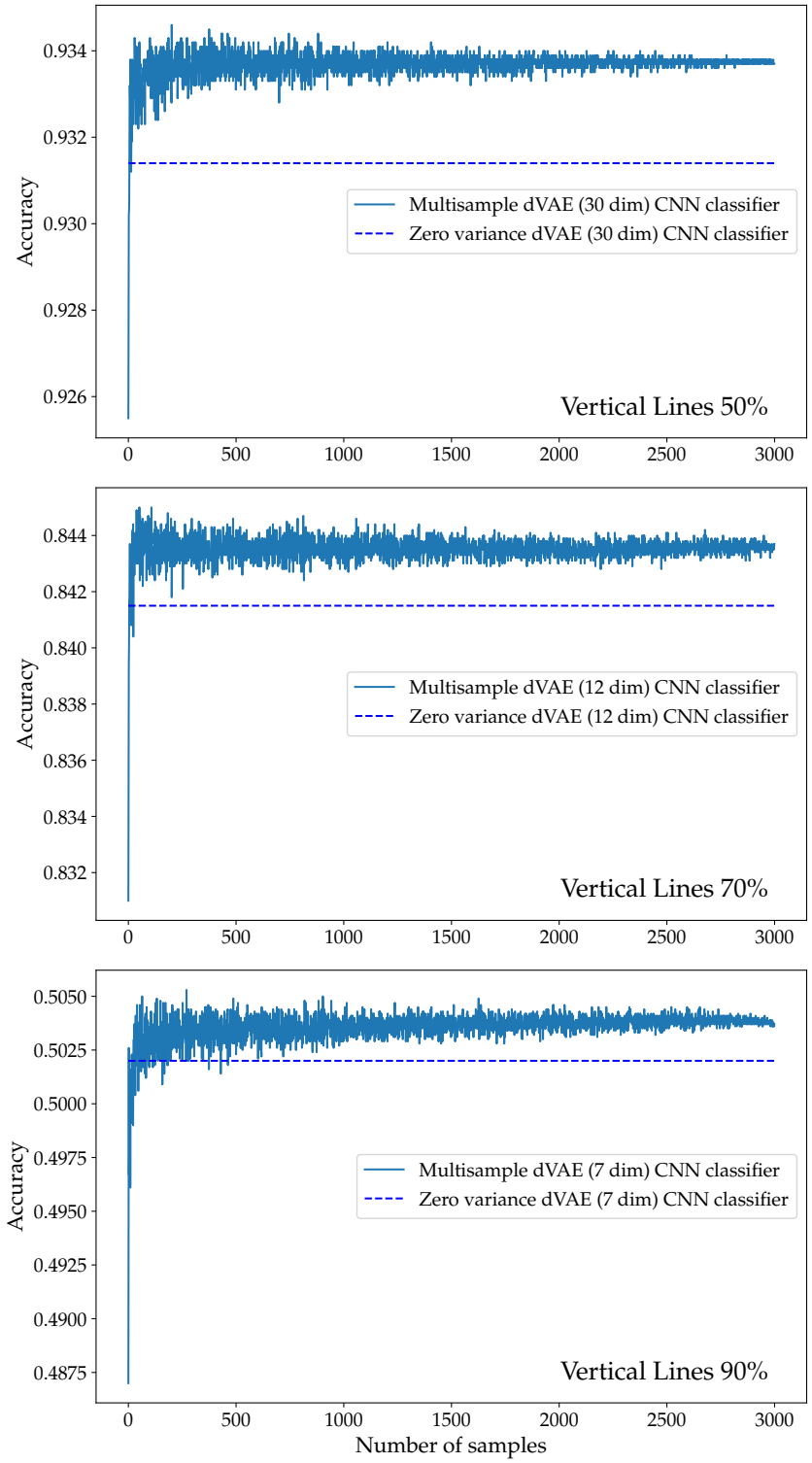


Figure 15: Performance of the single/multiple VAE+CNN classifiers on *Vertical Lines* corrupted images for three levels of corruption and number of samples on the x-axis. Direct CNN accuracies are presented in Table 5. Top: 50%. Middle: 70%. Bottom: 90%.

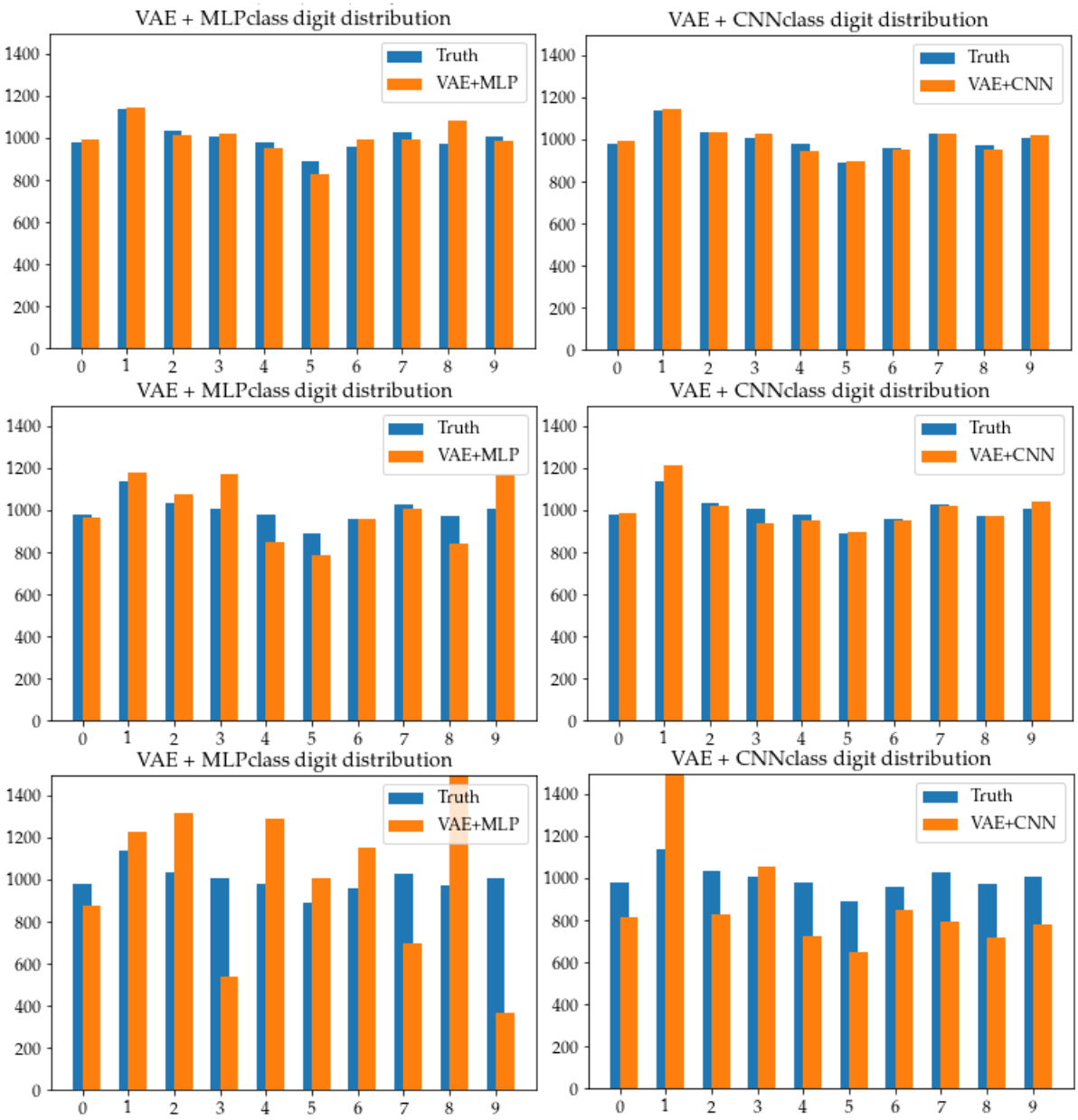


Figure 16: Digit distributions for the *Vertical Lines* corruption. Top row represents 50 % corruption, middle 70 % and bottom 90 %. The first column shows the distributions according to the VAE+MLP classifier compared to the true distribution (in blue), and the second to the VAE+CNN classifier.

Table 4: Accuracies and significance tests ($z_{\text{multi/single}}$) for *Vertical Lines* corruption levels cl $= 0.5, 0.7, 0.9$. MLP, VAE + MLP single vs 3000 sample multiple compared.

cl	MLP	VAE+MLP single	VAE+MLP 3000 samples	$z_{\text{multi/single}}$
0.5	0.8748	0.9229	**0.9232**	0.52
0.7	0.7682	**0.8303**	0.8298	-0.65
0.9	0.4706	**0.4903**	0.489	-0.63

Table 5: Accuracies and significance tests ($z_{\text{multi/single}}$) for *Vertical Lines* corruption levels cl $= 0.5, 0.7, 0.9$. CNN, VAE+CNN single vs 3000 sample multiple compared.

cl	CNN	VAE+CNN single	VAE+CNN 3000 samples	$z_{\text{multi/single}}$
0.5	**0.9755**	0.9314	0.9337	3.16
0.7	**0.9382**	0.8415	0.8437	2.18
0.9	**0.6638**	0.502	0.5037	1.13

As with the *Random Pixels* corruption, the direct CNN classifier performs best overall, but for the MLP-type classifiers a VAE imputation before helps. Interestingly, all classifiers perform better on *Vertical Lines* (Tables 4 & 5 compared to Tables 1 & 2), even though the same number of pixels are missing in both cases, for a given amount of corruption. This suggests that the random nature of *Random Pixels* only makes the problem harder. The difference between the multiple/single imputation is barely noticeable for the MLP-type classifiers in Fig. 14. For CNN-type classifiers, the larger and consistently positive z scores in Table 5 suggest that multiple imputation improves classification in this corruption scenario, as seen in Fig. 15. The overall behavior of the multiple sample VAE classifier for increasing number of samples is similar to what was observed for the *Random Pixels* corruption. In Fig. 16, note that the digit distributions for high corruption according to the VAE+MLP and VAE+CNN classifiers are not as similar as in Fig. 12, and the least classified digits are not the same.

Performance of classifiers trained on uncorrupted data

So far, all the classifiers have been trained and tested on data corrupted with the same type and level of corruption. To test if VAE-imputation can help in cases where labels are not as plentiful, so that classifiers cannot be trained on enough corrupted labeled data, the classifiers trained on uncorrupted data are tested on corrupted data, for both types and all levels of corruption, either directly or on an imputation from a VAE from the corrupted image. Note that the VAE has trained on the corrupted data, but the classifier attached to the end has not, so compared to the previous scenario, this time the VAE + MLP and the VAE + CNN are not trained on the VAE imputations. Exactly the same classifier (the same weights and hyperparameters) are used in both cases, but tested on either the corrupted images directly, or the VAE imputations. This is done in order to

test the usefulness of the unsupervised training of the VAE, and could correspond to a real world scenario: If a large data set is available, but only a small subset contain target labels, the VAE can be trained on the larger data set as an imputation tool, since the training of the VAE for imputation do not require target labels. The results are shown in Table 6 (for *Random Pixels* corruption) and Table 7 (for *Vertical Lines* corruption).

Table 6: Accuracies for Type 1 classifiers trained on uncorrupted data, with VAE trained on corrupted data, for *Random Pixels* corruption.

cl	MLP	MLP on one VAE imputation	CNN	CNN on one VAE imputation
0.5	0.2919	0.8394	0.1695	**0.8423**
0.7	0.1878	0.6882	0.1001	**0.6962**
0.9	0.1377	**0.2853**	0.0905	0.2775

Table 7: Accuracies for Type 1 classifiers trained on uncorrupted data, with VAE trained on corrupted data, for *Vertical Lines* corruption.

cl	MLP	MLP on one VAE imputation	CNN	CNN on one VAE imputation
0.5	0.2823	**0.9192**	0.2624	0.9182
0.7	0.1767	0.8159	0.1876	**0.8209**
0.9	0.1270	0.4158	0.1385	**0.4416**

The VAE models are chosen by performance on ELBO on the different corruption levels, and the MLP and CNN classifiers have been trained and model-selected by their performance on an uncorrupted validation set. The accuracies shown in the tables are for their classification performance on previously unseen corrupted data sets, varying in type and level of corruption. The classifiers clearly perform much better on the VAE imputations than directly on the corrupted test set, for all levels of corruption. Interestingly, in the vertical lines case the performance is almost on par with the performances of Type 2 classifiers in Table 4 and Table 5, in which the Type 2 classifiers were trained on the corrupted data set.

5 Discussion

Observations from the results are summarized here:

1. The difference in accuracy of multiple versus single imputation using the VAE is not very big, on one occasion 3.16σ but mostly around $\pm 1\sigma$ as seen in Tables 1, 2, 4 and 5.

2. The VAE + MLP classifiers outperform the MLP classifier at all of the tested types and levels of corruption.

3. The CNN applied directly after a trivial imputation, greatly outperforms any other classifier.

4. The VAE imputation in general performs better on images corrupted by the *Vertical Lines* method, used to model data missing not at random, than on images corrupted by the *Random Pixels* method, corresponding to data missing at random, suggesting that the former is an easier task for imputation.

5. For high levels of corruption, some numbers are harder to reconstruct and/or classify than others (see the digit distributions for 90 % corruption), which ones depend on the type of corruption.

6. The multiple imputation method does not improve in accuracy with more samples, after a low threshold (around ~ 10 samples), but the fluctuations get smaller. This behaviour is observed for all types and levels of corruption, in both types of multiple imputation classifiers, VAE+MLP and VAE+CNN.

Point 3, that the CNN outperforms any other classifier with the data set used is in itself not very surprising, as the MNIST data set is an image data set, with high local correlations between data points (pixels), a task for which the CNN is specifically designed for. The fact that the VAE+CNN classifiers perform worse, even with CNN used on the VAE imputations, could be explained by the fact that the quality of the imputed images by the VAE is measured by the reconstruction loss. In other words, the VAE could choose to mostly reconstruct *easy* images, the ones that have the most in common with other images. If it reconstructs an image of an 8, it would still get half of the reconstruction right if the uncorrupted image really was a 3, but for a classifier, that makes all the difference. At least for high levels of corruption, the digit distribution tables suggest an uneven distribution of reconstructed digits. For low levels of corruption, even if the distribution stays the same, the VAE imputations could reconstruct images that look like something in the middle between two classes, making the training for the classifier harder than to train directly on the corrupted data.

The most unexpected result is the observation that the multiple imputation classifier in many cases does not perform much better than the single imputation classifier, and it does not seem to improve much with more samples (after a low threshold). Intuitively,

you would expect the multiple imputation to get better than the single imputation as more and more samples are drawn (effectively integrating over the latent space). The hard cases, with corrupted images on the boundary between classes, should get better resolved as more and more area over the boundary is observed, at least that is the main point of doing multiple imputation for improving classification.

A possible explanation is that the unit variance prior used in the VAE objective puts a too severe restriction on the structure of the inferred latent space during training. The KL-divergence inhibits low σ, and tries to keep it not too far from $\mathbb{1}$. For multiple imputation to perform well, σ needs to be fairly unrestricted. If σ becomes too large, the latent space is over-regularized and overlapping between classes could occur. That is, when sampling $z \sim \mathcal{N}(\mu, \sigma)$ two images representing different digits may be projected to the same point in the latent space, leading to identical imputations but with different labels. That is an extreme case if the space is high dimensional, but it could suffice that points are mapped too close in latent space. If that happens, the training of the classifier on the VAE imputations gets an impossible task, as the classification boundaries randomly move about as nearby points in latent space are assigned different digits in each training iteration, due to the stochastic sampling of z. If on the other hand σ becomes too small, overlaps in latent space become less likely and the training of the classifier on the imputations should improve, but the performance approaches that of the single imputation, making the multiple sampling process less useful.

This would also explain the non-increasing accuracy of the multiple imputation with more samples. Once the VAE is trained and the weights are fixed, if the latent space is too tightly bound and contains overlaps between classes, any classifier trained on the VAE imputations is ultimately limited in performance, as it cannot change the structure of the latent space. More samples resolve the structure of the latent space, but if the structure itself is the problem behind poor classification accuracy, even an infinite amount of samples couldn't help.

Finally, a practical guideline for those seeking to know how to use deep learning imputation in practice, in light of the new results:

- If the data set under consideration is an image data set, and target labels are plentiful, it seems better to train a CNN on a simple imputation.
- If the data set is not an image data set, or supervised learning is difficult because of limited access to target labels, the VAE can be a useful imputation tool.
- Even in cases where multiple imputation perform slightly worse than single imputation (which can be done with a normal autoencoder), it allows for a better estimation of uncertainty in prediction.

Outlook

Further research could improve the effectiveness of the VAE as a multiple imputation tool. Of particular interest would be the use of more flexible priors, and the effect such a

modified objective would have on classification on missing data. The VAE classifier could be tested on other types of data sets, to see how classification performance of the different imputation methods are affected.

References

[1] Kang Hyun. The prevention and handling of the missing data. *Korean J Anesthesiol*, 64(5):402–406, 2013.

[2] Gerhard Tutz and Shahla Ramzan. Improved methods for the imputation of missing data by nearest neighbor methods. *Comput. Stat. Data Anal.*, 90(C):84–99, October 2015.

[3] Stef van Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software, Articles*, 45(3):1–67, 2011.

[4] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[5] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv e-prints*, page arXiv:1312.6114, Dec 2013.

[6] J.E. Peajcariaac and Y.L. Tong. *Convex Functions, Partial Orderings, and Statistical Applications*. Mathematics in Science and Engineering. Elsevier Science, 1992.

[7] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980, Dec 2014.

[9] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[11] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pages 319–, London, UK, UK, 1999. Springer-Verlag.

A Multilayer Perceptron

The MLP is the least restricted artificial neural network, and can be applied to a broad range of problems. Introducing it here can serve as a basic introduction to the core ideas behind ANN's, but for a proper introduction a textbook should be consulted, for example *Deep Learning* [10]. This appendix mainly serves as a brief overview of the subject, and also to clarify any terminology used in the thesis. The MLP is a graphical network consisting of nodes stacked up in layers that are connected by weights. It consists of an input layer, an output layer, and any number of layers in between, usually called *hidden* layers. If the MLP contains many hidden layers, it is called a *deep* network. The nodes in each following layer are connected to the nodes in the previous layer by weights, commonly all the nodes in the layer have connections to all the nodes in the previous layer, in which case it is called a fully connected, or dense, layer. The setup is shown in Fig. 17.

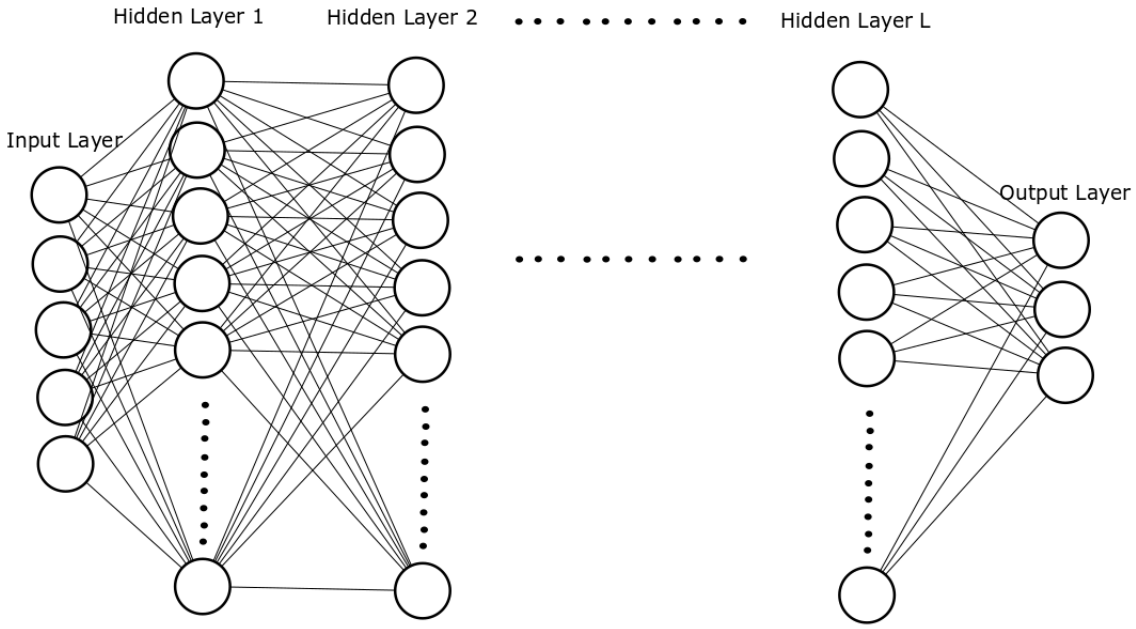


Figure 17: An example of a fully connected deep MLP, in this case with 5 input nodes, 3 output nodes and L hidden layers, each with an equal number of nodes. Note that the layers do not need to have the same number of nodes.

When values are assigned to the nodes in the input layer, they are sent forward through the network and each node in the new layer has an activation function f, which is a function

of the weights and the values of the input nodes. There are many choices of activation functions, but all have in common that they use the previous nodes as input, and the input are weighted.

The value h_{il} of node i in layer l is determined by a function of the weighted sum of all the nodes x_k in the previous layer $(l-1)$, weighted by their unique node-to-node connection weights.

$$h_{il} = f\left(\sum_k \omega_{ik} x_{k(l-1)}\right)$$

The training of an MLP consists of finding the optimal weights that produce the desired output for a given input, quantified by an objective, or *loss*, function. A problem with a fully connected MLP is the fact that the number of weights grow fast with increasing number of layers. For L layers with the same number of nodes N, the number of weights are N^L, this puts a limit on how deep MLP networks that can be used in practice.

B Convolutional Neural Network

A convolutional neural network [11] seeks to overcome the problem of the potentially huge number of weights in deep fully-connected layers, and is specially suited for image classification tasks. The design is inspired by insights in the workings of the human visual cortex [10]. The main idea is that for images, nearby pixels are often more correlated than pixels far apart. Therefore the weights are not fully connected, but only nearby input pixels have connections to a common layer, called a filter, the size of which can be decided for each problem. The input is mapped to many of these filters, and the filters are then mapped in a similar fashion to other filters, if the network is deep. It can be combined with a fully connected layer anywhere in-between, at the cost of the many weights it introduces.

C Kullback-Leibler Divergence between two multivariate gaussian distributions.

A proof for the analytic expression of the KL-divergence in section 2.2.1 for the special case of a gaussian prior with unit variance and a multivariate gaussian is presented here. Denote $p_1 = \mathcal{N}(\mu, \sigma)$, i.e. the multivariate gaussian. Also, it is assumed that:

$$\sigma = \text{diag}(\sigma_1, \sigma_2, ...\sigma_L) \tag{C.1}$$

i.e. the covariance matrix σ is diagonal, with L the dimension of the latent space. This is called a mean-field approximation in Variational Inference. The prior is $p_2 = \mathcal{N}(0, \mathbb{1})$. Under the mean-field assumption, the latent variables are independent and their distributions factor out:

$$p_1 = \mathcal{N}(\mu, \sigma) = \prod_{l=1}^{L} \mathcal{N}(\mu_l, \sigma_l) \coloneqq \prod_{l=1}^{L} p_1^l, \tag{C.2}$$

with

$$p_1^l(z_l;\mu_l,\sigma_l) = \frac{1}{\sqrt{2\pi\sigma_l^2}}\exp\left(-\frac{(z_l-\mu_l)^2}{2\sigma_l^2}\right). \tag{C.3}$$

The prior trivially factorizes as well. The KL-divergence between p_1 and p_2 is

$$\mathbb{KL}\left(p_1\|p_2\right) = \mathbb{E}_{z\sim p_1}\left[\log\left[p_1/p_2\right]\right] = \mathbb{E}_{z\sim p_1}\left[\log\left[p_1\right]\right] - \mathbb{E}_{z\sim p_1}\left[\log\left[p_2\right]\right]. \tag{C.4}$$

The first term in (C.4):

$$\begin{aligned}
\mathbb{E}_{z\sim p_1}\left[\log\left[p_1\right]\right] &= \mathbb{E}_{z\sim p_1}\left[\log\left(\prod_{l=1}^{L} p_1^l\right)\right] && \text{(C.5)}\\
&= \mathbb{E}_{z\sim p_1}\left[\sum_{l=1}^{L}\log p_1^l\right] && \text{(C.6)}\\
&= \sum_{l=1}^{L}\mathbb{E}_{z_l\sim p_1^l}\left[\log p_1^l\right] && \text{(C.7)}
\end{aligned}$$

and

$$\log p_1^l = -\frac{1}{2}\left(\log 2\pi + \log\sigma_l^2 + \frac{(z_l-\mu_l)^2}{\sigma_l^2}\right). \tag{C.8}$$

The first two terms are constants, and the expectation value of the third term in the bracket is:

$$\frac{1}{\sigma_l^2}\,\mathbb{E}_{z_l\sim p_1^l}\left[(z_l-\mu_l)^2\right] = \frac{\sigma_l^2}{\sigma_l^2} = 1$$

using the definition of the variance. So (C.7) becomes:

$$-\frac{1}{2}\sum_{l=1}^{L}\left(\log 2\pi + \log\sigma_l^2 + 1\right) \tag{C.9}$$

Now, the second term in (C.4) is just the expectation value of (C.8) for $\mu_l = 0$ and $\sigma_l = 1$, that is:

$$\mathbb{E}_{z_l\sim p_1^l}\left[-\frac{1}{2}\sum_{l=1}^{L}\left(\log 2\pi + z_l^2\right)\right] \tag{C.10}$$

as before, the first two are constants, but

$$\mathbb{E}_{z_l\sim p_1^l}\left[z_l^2\right]$$

needs to be evaluated.

Changing variables to $z_l' = z_l + \mu_l$, $z^2 = (z_l'-\mu_l)^2 = z_l'^2 - 2\mu_l z_l' + \mu_l^2$. $p_1^l(z_l)$ is shifted: $p_1^l(z) = p_1^l(z'-\mu) = \mathcal{N}(0,\sigma_l)$ so the expectation value becomes:

$$\mathbb{E}_{z_l\sim p_1^l}\left[z_l^2\right] = \mathbb{E}_{z_l'\sim\mathcal{N}(0,\sigma_l)}\left[z_l'^2\right] - 2\mu_l\,\mathbb{E}_{z_l'\sim\mathcal{N}(0,\sigma_l)}\left[z_l'\right] + \mu^2\,\mathbb{E}_{z_l'\sim\mathcal{N}(0,\sigma_l)}\left[1\right] = \sigma_l^2 + \mu_l^2 \tag{C.11}$$

where the definition of the variance has been used again, and the fact that odd moments of the gaussian distribution vanish. So (C.10) becomes:

$$-\frac{1}{2}\sum_{l=1}^{L}\left(\log 2\pi + \mu_l^2 + \sigma_l^2\right) \tag{C.12}$$

Finally, subtracting this from (C.9) gives:

$$\mathbb{KL}\left(p_1 \| p_2\right) = -\frac{1}{2}\sum_{l=1}^{L}\left(1 + \log \sigma_l^2 - \mu_l^2 - \sigma_l^2\right) \tag{C.13}$$

□