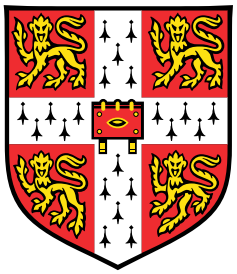
# Conditional Neural Processes and Semi-Supervised Learning



**Max-Olivier Van Bastelaer**

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
*Master of Philosophy in Machine Learning and Machine Intelligence*

Christ's College August 2021

I would like to dedicate this thesis to my loving family and partner, without whom my studies at Cambridge would not have been impossible.

# Declaration

I, Max-Olivier Christophe Van Bastelaer of Christ's College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

All software used in this thesis was written in Python from scratch for the purpose of this thesis, except where indicated otherwise. No proprietary tools were used.

Word count: 14798

Max-Olivier Van Bastelaer
August 2021

# Acknowledgements

First I would like to thank my supervisor Professor Richard Turner. I am profoundly grateful for your constant support and guidance. Your passion for machine learning inspired me at every meeting to work harder and search for better solutions.

I would also like to thank my co-supervisors James Requeima, Wessel Bruinsma and Andrew Foong. All three of you have been very helpful by sharing countless ideas with me. Thank you for taking the time to meet with me so often and for your suggestions on writing this thesis.

Finally, I would like to thank my coursemates for sharing my enthusiasm for machine learning. It was a special year and we were not able to meet as often as we would have wished, but I truly hope that we will transform this into long lasting friendships.

# Abstract

In applications of machine learning, we are often interested in a supervised classification task. It is typical to have a large pool of unlabelled data but few labelled samples available because labelling data could be expensive in terms of time or cost. In those settings, to achieve good results, it is key to leverage the unlabelled data to learn a representation that helps support the classification task. This area is in general called semi-supervised learning. A probabilistic approach to semi-supervised learning consists of jointly modelling the data and the labels with a distribution. In that case, the model should be designed such that the insights gained with the density modelling task help the model on the supervised task.

In this thesis, we will evaluate the use of Neural Processes for semi-supervised learning. Neural Processes are a new type of neural network architecture which provide new ways to couple modelling of the unlabelled data to modelling of the labelled data. They also offer a new capability for classification tasks to take as input subsets of data or to handle missing data.

We are going to take three lines of attack. First we look at a transfer learning approach where we first model the unlabelled dataset and then consider transferring information to the labelled set. Second, we investigate joint training and merge it together with consistency losses. Finally, we take a more sophisticated approach where we use latent variable models with a class-latent variable.

While we are able to improve over fully-supervised baselines by identifying a good model architecture, reaching accuracies competitive with recent algorithms from the semi-supervised literature requires combining our model with other standard semi-supervised techniques. Even though it would be convenient to fit the whole semi-supervised approach into a well-defined probabilistic Neural Process model, the results from this thesis suggest that it is not the only part of the solution and that other techniques are necessary to get competitive performances.

# Table of contents

# List of figures

# List of tables

# Nomenclature

**Acronyms / Abbreviations**

CNN Convolutional Neural Network

CNP Conditional Neural Process

ConvCNP Convolutional Conditional Neural process

DIGLM Deep Invertible Generalized Linear Model

ELBO Evidence Lower-Bound

EQ Exponential Quadratic

GCNN Graph Convolutional Neural Network

GMM Gaussian Mixture Model

GNN Graph Neural Network

GP Gaussian Process

JS Jensen-Shannon

KNN K-Nearest Neighbours

LNP Latent Neural Process

LR Logistic Regression

MLP Multilayer Perceptron

SVM Support Vector Machine

VAE Variational Auto-Encoder

# Chapter 1

# Introduction

While deep learning algorithms have repeatedly achieved human-level performances on supervised tasks by learning from very large labelled datasets, this approach is not easily scalable. Labelling data sets can require a large amount of human effort, risk (for example in medical data where invasive tests must be performed), or financial cost (for example to pay experts to label data) (Oliver et al., 2018).

When sampling data is inexpensive but labelling is expensive, semi-supervised learning can be a particularly powerful framework. The idea is to train a model to solve a given supervised task with a dataset split into a labelled and unlabelled part, where the unlabelled part is typically much larger than the labelled one. While semi-supervised learning is applicable to different supervised tasks, we restrict our work to classification tasks only. Chapter 2 is a review of semi-supervised learning and the types of methods which are currently available.

In this thesis, we will evaluate the use of Neural Processes (Garnelo et al., 2018a,b) for semi-supervised learning. Neural Processes, introduced in detail in chapter 3, are a novel type of neural network architectures which are used to model functions sharing some high level characteristics. In our semi-supervised setting, we are concerned with the case where every function has a class assignment. For example, if we view images of hand-written digits as 2d functions determining the pixel values at every location, then each of those functions would be associated to a label from 0 to 9 indicating the digit represented in the image.

Our target model is a Neural Process that extracts high quality classification features through the density modelling task. The intuition is that since the model will be exposed to many functions from the same family and it will be tasked to reconstruct them, it will need to extract high level information from those functions, which will be useful for classification. To try to reach that model, we present an analysis divided in three steps:

1. In chapter 4, different architectures are compared by using Neural Processes as pre-trained feature extractors and then testing how good the features extracted are with different classifiers.

2. In chapter 5, we evaluate whether jointly training the model on the density modelling task and the classification task helps to get better performance. We also try adding a consistency loss to the objective function, forcing functions sampled at different locations to have the same label. Furthermore, inspired from the self-supervised literature, we craft a new task which requires extracting good classification features, and we also add it to the training objective.

3. In chapter 6, we extend our model by introducing a class latent variable, so the classification task becomes equivalent to inferring the value of this variable. We consider two models, one deterministic and one where we include additional continuous latent variables. We will compare them on the MNIST (LeCun et al., 1998), SVHN (Netzer et al., 2011) and CIFAR10 datasets (Krizhevsky et al., 2009).

All models will be compared on semi-supervised image classification tasks, where we randomly sample some images to be labelled and keep the rest as unlabelled. The results achieved are not the best compared to state-of-the-art semi-supervised algorithms, but they are nonetheless interesting to evaluate how Neural Processes can contribute to solve semi-supervised problems.

## 1.1 Thesis contributions

The main contributions of this work are the following:

- An **updated taxonomy on semi-supervised learning algorithms**. We updated the classification of semi-supervised methods from Chapelle et al. (2009), which is outdated and misses the new deep-learning methods.

- A **thorough evaluation of the performance of Neural Processes for semi-supervised classification tasks**, including a distillation of the different parameters to identify the key components.

- A **new framework to model different families of functions with a single Neural Process**, by including a class-latent variable in the model.

- A **new method to weight losses in semi-supervised settings with multi-objective functions**. We modified the GradNorm method from Chen et al. (2018) to apply it to semi-supervised learning.

# Chapter 2

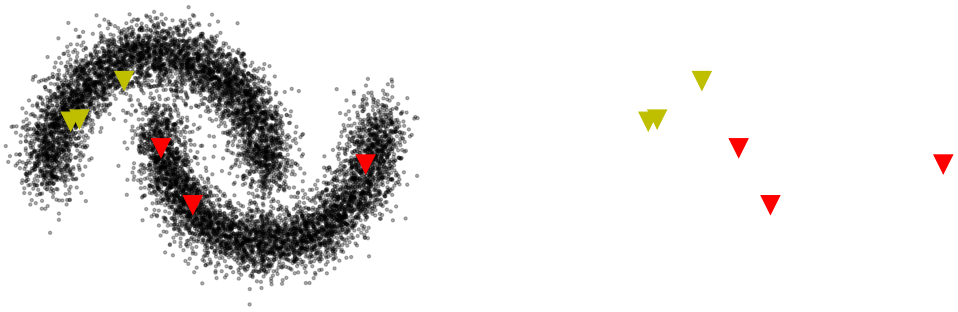
# Background on Semi-Supervised Learning

In this chapter, we give an introduction to semi-supervised learning. Section 2.1 introduces the problem formally, along with the notation used. We propose a taxonomy for semi-supervised methods in section 2.2, inspired by other reviews from the literature (Chapelle et al., 2009; Yang et al., 2021). The particularity is that we tie all categories by linking them to a single assumption (Assumption 2.1.1). The taxonomy groups semi-supervised algorithms into five main categories illustrated in figure 2.2. Self-supervised approaches are briefly mentioned in section 2.3. Finally, section 2.4 presents the difficulties that one encounters when evaluating semi-supervised algorithms.

To give a sense of the performance of the different algorithms presented in the taxonomy, we compare the performance of the different models on the MNIST hand-digit classification task (LeCun et al., 1998), with only 100 labelled training images and keeping the rest unlabelled. The results are presented at the end of section 2.2 in table 2.1.

## 2.1 Problem statement

Semi-supervised settings occur when solving a supervised task but with a training dataset $D = d_{1:l+u}$ split into two parts: the part $D_L = d_{1:l}$ for which labels $\lambda_{1:l}$ are provided and the part $D_U = d_{l+1:l+u}$ for which the labels are not known. The objective of semi-supervised learning algorithms is to leverage the unlabelled set to improve the performance on the supervised task. The condition for theses approaches to be useful is that the knowledge gained by elucidating the distribution of the unlabelled samples $p(d)$ is useful for the inference of $p(\lambda \mid d)$ (Chapelle et al., 2009). Figure 2.1a shows the common two-moons



(a) Labelled and unlabelled points (b) Labelled points only

Fig. 2.1 Two half moons. The grey dots are unlabelled points and the triangles are labelled points, where the class is color-coded. It illustrates how the unlabelled points help to build a better classifier.

classification problem. Most of the points are unlabelled (indicated as round grey dots) but six of them are labelled (triangles with a color-coded class). It is easy for a human to draw a decision boundary, because we identify two clusters by looking at the unlabelled points, and assign a class to each of the clusters. However, in figure 2.1b, the decision boundary is ambiguous without the unlabelled samples. In this case, the knowledge about $p(d)$ helps building a better classifier $p(\lambda \mid d)$. Semi-supervised learning algorithms rely on the following smoothness assumption (Chapelle et al., 2009).

**Assumption 2.1.1** (Smoothness assumption, taken from Chapelle et al. (2009))**.** *If two points $d_1$,$d_2$ in a high density region are close, then so should be their corresponding labels $\lambda_1$,$\lambda_2$.*

The notion of closeness is ambiguous, and it is its specific definition which will define the type of algorithm used. For example, if we consider distances in the input space, then by transitivity two points connected by a high density path will have the same label. So it is equivalent to say that points belonging to the same cluster will have the same label. This is known as the cluster assumption. Otherwise, if we assume that the data lie on a low dimensional (possibly curved) manifold, then closeness could be defined as the geodesic distances between the points. This accommodates for more complex concept of closeness, for example to say that two images under different lighting conditions are close, even though their pixels have very different values. Section 2.2 introduces different types of semi-supervised algorithms and links them to the definition of closeness that they use.

## 2.2 Taxonomy of semi-supervised learning algorithms

All semi-supervised algorithms can be interpreted as relying on a specific definition of closeness in assumption 2.1.1, to define a classification model which links the labelled and unlabelled samples. Based on the definition of closeness and type of model used, semi-supervised algorithms can be divided in five different classes (Chapelle et al., 2009; Yang et al., 2021). The taxonomy presented here is illustrated with a classification tree in figure 2.2. It is important to stress that the categories are not mutually exclusive and that some algorithms lie at the intersection of different classes. The algorithms presented below sometimes rely on advanced machine learning methods; instead of introducing them one by one which would make the taxonomy less concise and harder to follow, we refer the reader to good introductions from the literature, indicated with footnotes.

### 2.2.1 Change of representation

A simple framework for semi-supervised learning is to learn a change of representation with unsupervised learning, by using the whole dataset $D = D_L \cup D_U$ but ignoring the available labels. Then we perform plain supervised learning with the labelled points mapped the new representation. In this case, two points are said close in assumption 2.1.1 if their distance is small in this new representation. Examples of change of representation methods are PCA (Pearson, 1901), ISOMAP (Balasubramanian et al., 2002) and Variational Auto Encoders (VAE) (Kingma and Welling, 2013).[1] VAEs fit into this category and not into the generative model category introduced below when they are trained in a fully unsupervised fashion to just reconstruct the data, and then samples from the posterior latent distribution are used as classification features for the supervised task.

### 2.2.2 Low density separation

Low density separation algorithms assume that points are close if their distance in input space is small; they attempt to push the decision boundary away from the samples. The most common approach is to use classifiers which maximize the margin for unlabelled and labelled points, called transductive support vector machines (TSVM) (Gammerman et al., 1998). However, this problem is nonconvex so a range of algorithms have been proposed to optimize TSVMs (De Bie and Cristianini, 2003; Demiriz and Bennett, 2001; Joachims, 1998). Another method for low density separation is the null category noise model (NCNM) from Lawrence and Jordan (2004). This GP classifier forces the decision boundary to lie in a

[1] For an introduction to VAE, see (Kingma and Welling, 2013)

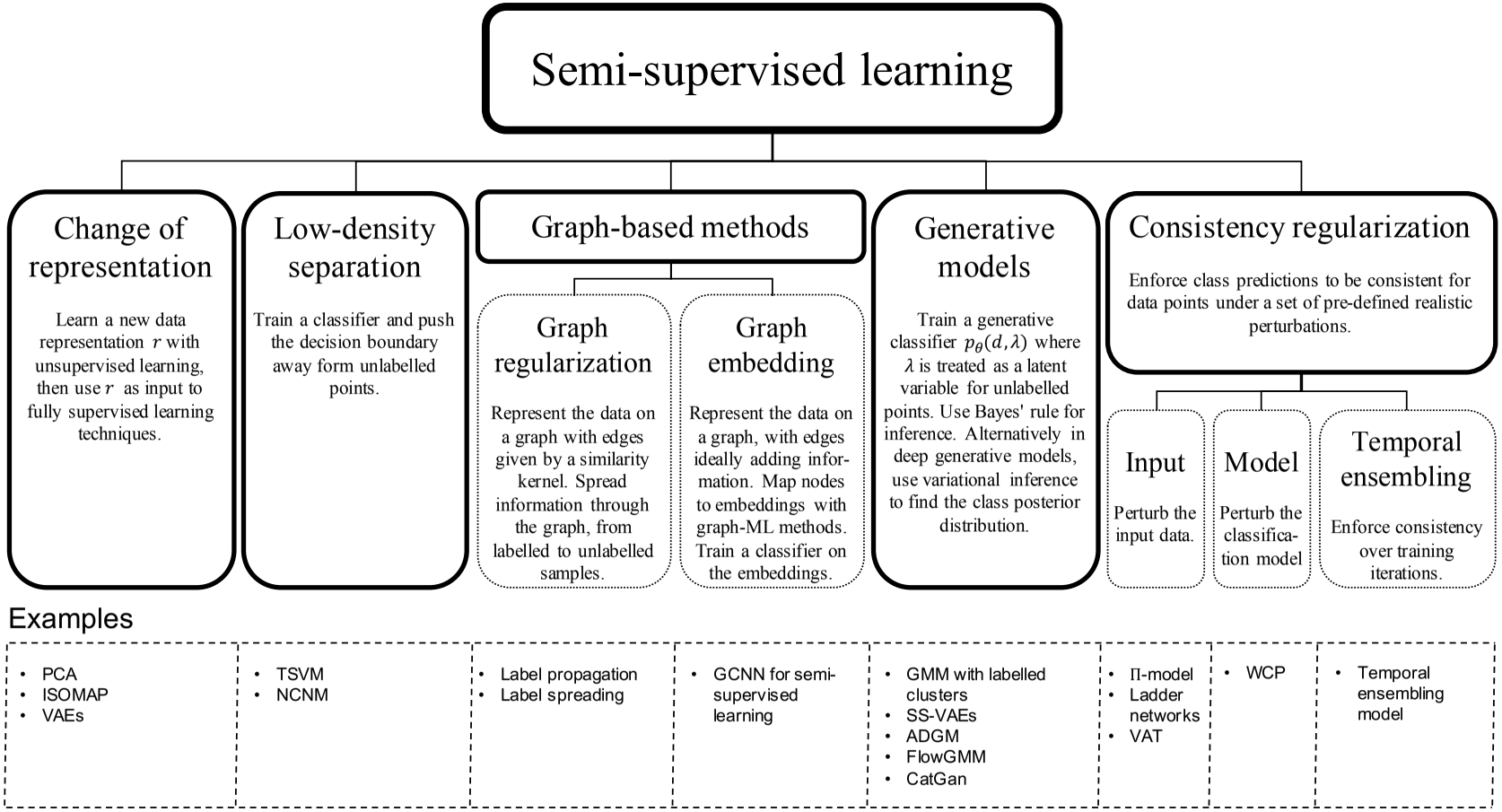



Fig. 2.2 Classification tree for the semi-supervised algorithms. Each category is introduced in section 2.2.

low density region by adding to the model an extra "null" category always situated between the different labelled regions, and saying that no data (labelled or unlabelled) comes from this region.

### 2.2.3 Graph based models

Graph based semi-supervised learning algorithms use a graph to model data; samples are nodes and edges encode some sort of similarity. The techniques are divided in two mains families: graph regularizations and graph embeddings.

Early research has mostly been on graph regularization. The idea is to rely on assumption 2.1.1 and define closeness as high similarity with respect to a kernel function giving the weights of the edges in the graph, stored in an adjacency matrix $W$. Common kernels include k-nearest neighbours: $W_{i,j} = w(d_i, d_j) = 1$ if $d_i$ is among the k-nearest neighbours of $d_j$ and 0 otherwise, or the squared exponential (of width $\sigma$): $W_{i,j} = w(d_i, d_j) = \exp(-\frac{\|d_i - d_j\|^2}{2\sigma^2})$. Graph regularization algorithms spread the information from the labelled points through the graph. It can intuitively be understood as iteratively updating the class probability of a node as the similarity-weighted average of the class probability of the neighbouring nodes. Such algorithms include for example label propagation (Zhu and Ghahramanih, 2002) or label spreading (Zhou et al., 2004).

More recent research has focused on graph embeddings, which is related to the change of representation methods. It consists in encoding each node to a representation $r$ using an encoder function which takes as input not only the data value $d$ but also its relation with neighbouring points, i.e. the value of the neighbouring nodes and edges. The state-of-the-art methods use Graph Neural Networks (GNNs) as encoders.[2] While they have more of a traditional fully supervised flavour since the models are simply trained to minimize the cross entropy of the class probability predictions with respect to the labelled samples, they qualify as semi-supervised techniques since the encoder extracts information from the neighbouring unlabelled points through the propagation/aggregation GNNs iterations. Intuitively, this method can be understood as learning a classifier taking as input the features of a data point $d$ augmented with the features of the neighbouring points and its connection to those points. The graph embedding methods work best when the edges add extra information not present in the features of the data. For example, in the Citeseer dataset (Sen et al., 2008) where the task is to classify bag-of-words vectors from scientific articles as belonging to certain classes, citations are available and can be used to define directed edges between papers, so they add extra information. In this setting, two points are said close in assumption 2.1.1 if they have a

[2] See Wu et al. (2020) for a detailed introduction to GNNs.

similar embedding $r$, which would be one of the hidden layers in the GNN. For an example of a graph embedding method, see the Graph Convolutional Neural Network (GCNN) for semi-supervised learning from Kipf and Welling (2016).

### 2.2.4 Generative models

Generative methods rely on a model of the joint data-label distribution $p_\theta(d, \lambda) = p_\theta(d \mid \lambda) p_\theta(\lambda)$, where the class assignment is treated as a latent variable for unlabelled points. It can be used as a discriminative classifier with Bayes' rule: $p_\theta(\lambda \mid d) = \frac{p_\theta(d|\lambda) p_\theta(\lambda)}{p(d)}$. Learning consists in finding the parameters $\theta$ which maximize the log probability of the observations,

$$\mathscr{L}(\theta) = \underbrace{\sum_{i=1}^{l} \log(p_\theta(d_i \mid \lambda_i))}_{\text{labelled points}} + \underbrace{\sum_{i=l+1}^{l+u} \log(\sum_{k=1}^{K} p_\theta(d_i \mid \lambda_k) p_\theta(\lambda_k))}_{\text{unlabelled points}}. \tag{2.1}$$

where $K$ is the number of classes. This objective can be optimized with the Expectation-Maximization algorithm or by gradient descent directly. Common generative algorithms include Gaussian Mixture Models (GMM) or hidden Markov Models. For generative models, closeness in assumption 2.1.1 means that two points are both in a high density region of the same mixture component, which for most models translates to being close in input space.

Research on semi-supervised generative models has recently shifted focus to deep learning based approaches. Kingma et al. (2014) introduced a semi-supervised VAE which has both a categorical class-latent variable $\lambda$ and a continuous latent variable $z$. They presented three models, the M1, the M2 and the M1+M2 models, which use different architectures and different number of stochastic layers. Maaløe et al. (2016) extended the M2 model with an auxiliary variable such that the posterior over the latent variables can fit more complicated distributions. The M2 model will be used as inspiration in chapter 6 to build a semi-supervised Neural Process using a very similar method. Normalizing flows have also been employed in the generative semi-supervised setting.[3] They are used to learn to map the input data distribution to more convenient distributions compatible with simple classifiers. For example, FlowGMM (Izmailov et al., 2020) maps the input distribution to a GMM with pre-defined class assignments and uses Bayes' rule as a classifier. Finally, semi-supervised Generative Adversarial Networks (GAN) have been introduced with the CatGAN (Springenberg, 2015).[4] The idea is to build a discriminator with $K+1$ output nodes, one for each class and one for fake data. The discriminator is then trained to maximize the probability of fake data being

[3] We refer the reader to Kobyzev et al. (2020) for a thorough introduction to normalizing flows.

[4] For an introduction to GANs, see Goodfellow et al. (2014).

classified as fake, true labelled data being classified as its corresponding target class, and unlabelled data as any of the true data classes (i.e. 1 minus the probability of being fake).

### 2.2.5 Consistency regularization

Consistency regularization based semi-supervised algorithms have recently achieved state-of-the-art performances. These methods assume that two points $d_i$, $d_j$ are close in assumption 2.1.1 if they are the image of the same point $d$ under a set of pre-defined realistic perturbations. This assumption is usually included in the model with a consistency loss, which is usually the KL divergence or the mean squared error between the class predictions for $d_i$ and $d_j$ (Yang et al., 2021). The perturbations can be divided into three categories: input augmentation, model perturbations and temporal ensembling. Note however that consistency regularization algorithms often combine ideas from more than one category.

Input augmentation consists in defining perturbations on the input which do not alter the semantics, for example the Π-model (Sajjadi et al., 2016) uses affine transformation of images, Ladder Networks (Rasmus et al., 2015) use additive noise and the VAT model (Miyato et al., 2018) relies on adversarial perturbations.

Model perturbations assume that the model should be robust to perturbations such as additive noise on the network weights or structural changes by dropping connections. For example, the novel Worst-Case-Perturbation (WCP) algorithm (Zhang and Qi, 2020) trains a model by finding the worst possible model perturbations and enforcing consistency with respect to them, kind of like an adversarial attack but on the model weights/connections.

Temporal ensembling methods constrain the model to make consistent class predictions over the training iterations. For example, the "Temporal Ensembling model" from Laine and Aila (2016) includes a consistency loss between predictions at the current iteration and an exponential moving average of previous predictions, combined with data augmentation.

## 2.3 A note about Self-supervised learning

Self-supervised learning algorithms are different than semi-supervised learning ones because they do not formalize the link between the labelled and unlabelled data through a model. It is a three step process. First, an expert defines a pseudo task which can be obtained from the data directly without human annotations. Second, a model is trained on this artificial task. Third, some part of the model are transferred and fine-tuned on the classification task, with the assumption that it will have extracted good transferable features. Many different and creative tasks have been considered to pre-train models; famous examples on images are

Table 2.1 Results achieved by the semi-supervised algorithms presented in the taxonomy in section 2.2, when tested on the MNIST classification task (LeCun et al., 1998) with 100 labelled images. The references point to the source where the result was found. For online Github repositories, the link is included as a hyperlink. "SS" is short for semi-supervised. "NA" means that the result was not available. All the results come from different sources, so they should not be interpreted as exact numbers because they might be the result of different computational budget/hyper-parameter optimization. They should rather be treated as an approximate indication of performance.

| Category | Model | Accuracy | Reference |
|---|---|---|---|
| Change of representation | PCA + SVM | 69.2 | Github |
| | ISOMAP | 81 | Bijral et al. (2012) |
| | VAE (M1) | 88.18 | Kingma et al. (2014) |
| Low-density separation | TSVM | 83.18 | Bachman et al. (2014) |
| | NCNM | NA | - |
| Graph based methods | Label propagation | 69.5[1] | Calder et al. (2020) |
| | Label spreading | 86.8[2] | Tudisco et al. (2021) |
| | GCNN + 5 nearest neighbour kernel | 82.1 | Li et al. (2018) |
| Generative models | GMM with labelled clusters | NA | - |
| | SS-VAE, M2 | 88.03 | Kingma et al. (2014) |
| | SS-VAE, M1+M2 | 96.67 | Kingma et al. (2014) |
| | ADGM | 99.04 | Maaløe et al. (2016) |
| | FlowGMM | 98.2 | Izmailov et al. (2020) |
| | CatGAN | 98.09 | Springenberg (2015) |
| Consistency regularization | Π-model | 99.45 | Sajjadi et al. (2016) |
| | Ladder Network | 98.94 | Rasmus et al. (2015) |
| | VAT | 98.64 | Miyato et al. (2018) |
| | WCP | NA | - |
| | Temporal ensembling | 98.38 | Github |

[1] with 50 labelled images
[2] with 60 labelled images

grayscale image colorization (Zhang et al., 2016), image inpainting (Pathak et al., 2016) or solving image jigsaw puzzles (Noroozi and Favaro, 2016).

While different, self-supervised learning is not incompatible with semi-supervised learning. In fact, the change of representations methods can be viewed as self-supervised tasks if we interpret the first unsupervised learning part as solving an artificial task.

## 2.4 Evaluation of semi-supervised algorithms

The evaluation of semi-supervised learning algorithms is typically done by measuring classification accuracy with a dataset such as MNIST (LeCun et al., 1998), SVHN (Netzer et al., 2011) or CIFAR10 (Krizhevsky et al., 2009), by only keeping a small portion of the data as labelled and treating the rest as unlabelled. The algorithms are typically compared with fixed number of labelled samples (traditionally 100 for MNIST, 1000 for SVHN and 4000 for CIFAR10), with every class equally represented. Oliver et al. (2018) raise the question whether those evaluation methods are coherent with the realistic applications of semi-supervised learning. We list here three remarks relevant for the evaluation of algorithms presented in this thesis.

First, it is important to spend some (computational) effort to obtain a high quality fully supervised baseline. The goal of semi-supervised leaning is to design algorithms which achieve high accuracies by using $D_U \cup D_L$ instead of only $D_L$. So, in order to make a fair comparison, one should try to optimize the performance obtained with $D_L$ only. This could include for example trying different classification models, different neural network architectures, or, if a suitable classifier pre-trained on a similar dataset is available, trying to fine-tune that model on the labelled data. Additionally, if hyper-parameters are tuned in the semi-supervised algorithm, a similar computational budget should be spent on optimizing hyper-parameters for the fully supervised baseline as well.

Second, in a realistic application of semi-supervised algorithms, the amount of labelled data available will be small, so extracting a validation set from the training data can be very expensive. In fact, a good validation set should be composed of a large enough number of labelled samples to accurately evaluate the algorithm, but that means less samples in the training set. Many papers disregard that and just use an unrealistically large validation set for hyper-parameter tuning, for example having only 100 labelled training images but using 1000 labelled validation samples (Miyato et al., 2018; Tarvainen and Valpola, 2017). For this reason, hyper-parameter tuning should be avoided as much as possible and, if necessary, should ideally be done with a realistically sized validation set.

Finally, the performance of semi-supervised algorithms might differ quite a lot with different number of labelled images. So when testing a model, one should test it on a large range of number of labelled samples, to highlight in what circumstances the model is best suited.

# Chapter 3

# Background on Neural Processes

Neural Processes (NPs) are used in the next chapters as a base for our semi-supervised models. They are used to extract features from the data, which are then mapped to class predictions by the classification models. Before diving into the details of our semi-supervised models, we provide some background on NPs in this chapter.

NPs incorporate ideas from Gaussian Process (GP) inference into a neural network training regime and model functions drawn from stochastic processes, by learning a distribution over functions given some observations. The Conditional Neural Processes (CNP) (Garnelo et al., 2018a) was the first type of NP introduced in the literature.

In section 3.1, we introduce the notation and the type of data processed by a NP. The Conditional Neural Process (CNP) architecture is covered in section 3.2 and section 3.3 explains the training procedure. Section 3.4 covers the Latent Neural Process (LNP) (Garnelo et al., 2018b), a type of NP which allows coherent sampling from the predictive distribution. The third type of model, in section 3.5, is the Convolutional Conditional Neural Process (ConvCNP) which supports translation equivariance. Finally, we illustrate in section 3.6 the output of the three models with examples on data generated from Gaussian Processes with different covariance functions and on images.

## 3.1 Data pipeline and notation

Let $\{f_k : X \to Y\}$ be a set of functions drawn from a stochastic process $P$; $P$ could for example be a Gaussian Process (GP) or an image generator. We define for every function $f_k$ a set of observed context points $C = (x_{1:n}, y_{1:n})$ and a set of target points $T = (x_{n+1:n+m}, y_{n+1:n+m})$, such that $y_i = f_k(x_i)$. During the inference process, the values $(y_{n+1:n+m})$ in the target set are unobserved, and we denote $\tilde{T} = (x_{n+1:n+m})$ the set containing only the input locations. The task of a NP is to make predictions for the value at the target points conditioned on

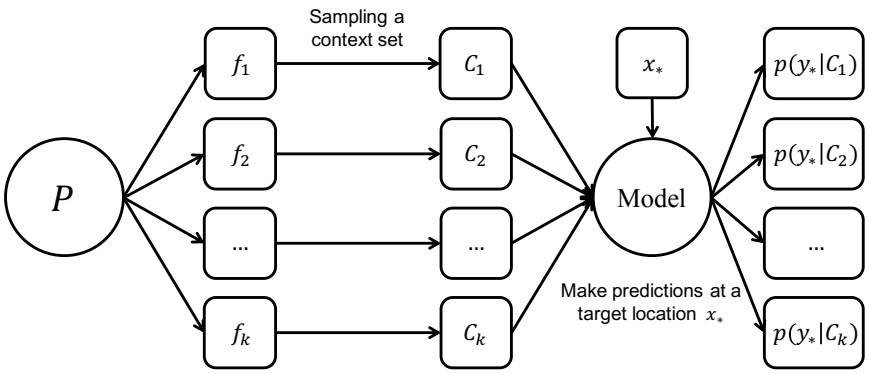



Fig. 3.1 Data pipeline for NPs. Batches of functions $\{f\}$ are sampled from the stochastic process $P$. A context set $C$ is extracted from the functions and then fed through the model together with a target input location $x_*$, in order to predict $p(y_* \mid C)$.

the context set, so it infers $p(y_{n+1:n+m} \mid x_{1:n}, y_{1:n}, x_{n+1:n+m}) = p(y_{n+1:n+m} \mid C, \tilde{T})$. Figure 3.1 illustrates how data is sampled and how it is then used by the NP to make predictions at the targets points.

## 3.2 Conditional Neural Process

To model functions drawn from a stochastic process, we desire that CNP satisfies two properties:

1. Invariance to permutations of $C$ and $T$,[1]
2. Ability to handle sets of different sizes.

The invariance to permutations of $T$ is satisfied by making the assumption that the predictive distribution, denoted by $p_\theta(y_{n+1:n+m} \mid C, \tilde{T})$, factorizes:

$$p_\theta(y_{n+1:n+m} \mid C, \tilde{T}) = \prod_{i=n+1}^{n+m} p_\theta(y_i \mid C, \tilde{T}). \tag{3.1}$$

One consequence of this assumption is that the function values at the target locations are conditionally independent given $C$, so it it not possible to obtain coherent samples from this predictive distribution. While the factorization assumption is not necessary, it simplifies the likelihood function as the product can then be simply transformed into a sum by taking the log-likelihood.

[1] Strictly speaking this is not a desire but a requirement for stochastic processes according the Kolmogorov Extension Theorem (Øksendal, 2003).

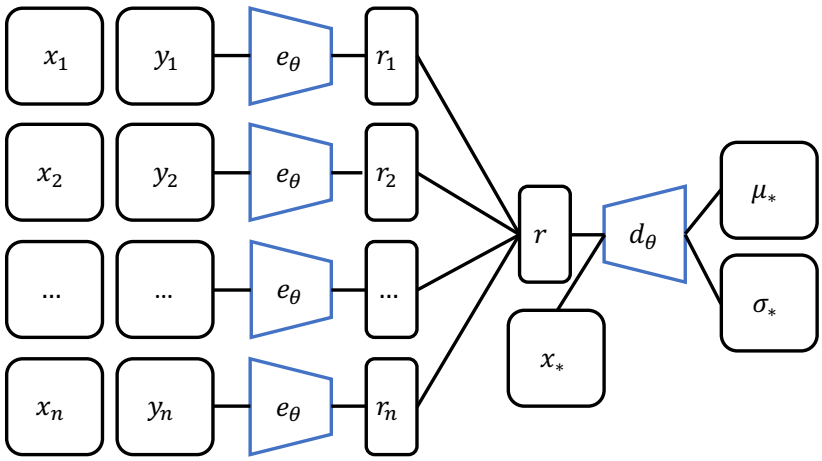



Fig. 3.2 CNP model (Garnelo et al., 2018a). Blue shapes indicate functions. The encoder $e_\theta$ maps the pairs of context points to vectors $r_i$ which are then aggregated to obtain $r$, a unique representation for the whole context set. $r$ is then passed through the decoder with the location of the target point $x_*$ to output a mean and standard deviation which are the parameters of the predictive distribution $p_\theta(y_* \mid C, x_*)$.

The CNP makes conditional predictions given the context set by embedding it to a representation $r$ of fixed dimensionality. Equations (3.2)-(3.4) summarize the architecture of the CNP, and figure 3.2 represents it visually. Garnelo et al. (2018a) propose to model $p_\theta(y_* \mid C, x_*)$ as a Gaussian predictive distribution, so the model outputs a mean and standard deviation for every target point $(\mu_i, \sigma_i)$.

$$r_i = e_\theta(x_i, y_i), \quad \forall (x_i, y_i) \in C \tag{3.2}$$

$$r = r_1 \oplus r_2 \oplus \cdots \oplus r_n \tag{3.3}$$

$$\mu_i, \sigma_i = d_\theta(x_i, r), \quad \forall x_i \in \tilde{T} \tag{3.4}$$

Equations (3.2)-(3.4) describe an encoder-decoder model. The encoder, $e_\theta : X \times Y \to \mathbb{R}^R$ maps each pair of context points $(x_i, y_i)$ to an embedding $r_i$ of fixed dimension $R$. Equation 3.3 aggregates all the embeddings by making use of a commutative operation $\oplus$, it results in a single element $r$ which represents the whole context set. In Garnelo et al. (2018a), $\oplus$ is the mean operation. The decoder $d_\theta : X \times \mathbb{R}^R \to \mathbb{R}^{2 \cdot C}$ maps every target point to parameters of a Gaussian distribution, based on the embedding $r$. If the predictive was modelled as another type of distribution, the output parameters would differ; for example, the model could output logits to parametrize a categorical distribution. Both the encoder and the decoder are neural networks to allow them to learn complex structures. The CNP encoder (and the models introduced in the next sections) is actually a particular type of the DeepSet architecture from

Zaheer et al. (2017) which defines what properties should be satisfied by functions with sets as input.

## 3.3 Training

For training, assume that we have access to some training data sampled from a stochastic process: $C, T \sim P$. The target values $y_i \in T$ are only used to evaluate the predictive distribution. Let the total number of points $n+m=N$ be fixed (record that $|C|=n, |T|=m$), but assume that $n$ is sampled uniformly as $n \sim \text{Uniform}(1,\ldots,N)$. Then the CNP is trained by minimizing the negative conditional probability,

$$\mathscr{L}(\theta) = -\mathbb{E}_n\left[\mathbb{E}_{C,T\sim P}\left[\log p_\theta\left(y_{1:n+m} \mid C, \tilde{T}\right)\right]\right]. \tag{3.5}$$

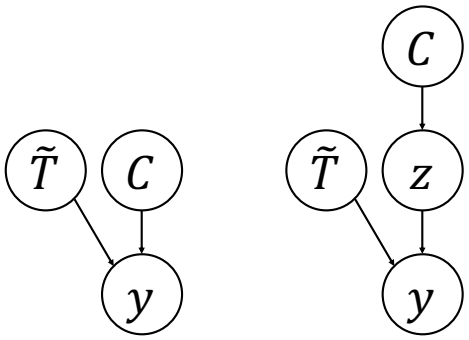
The model is trained by predicting the function values from $C \cup T$ instead of only the ones from $T$ because it was empirically found in Garnelo et al. (2018a) to lead to better performances. Since both expectations cannot be computed exactly, $\mathscr{L}(\theta)$ is approximated with Monte-Carlo estimations where $(C,T)$ and $n$ are jointly sampled. In practice, the Monte-Carlo estimation is done with mini-batches and the objective is optimized with stochastic gradient descent.

## 3.4 Latent Neural Process

Recall that the factorization assumption in equation (3.1) prevents coherent sampling from the predictive distribution. To accommodate for that, Garnelo et al. (2018b) proposed the Neural Process, here referred to as the Latent Neural Process (LNP) for better distinction with the other models. This model conditions the predictive distribution on a shared latent variable $z$. The generative model for the LNP is illustrated in figure 3.3b, together with the CNP model for comparison. It translates to:

$$p_\theta(z, y_{1:n+m} | \tilde{T}, C) = p_\theta(z \mid C) p_\theta(y_{1:n+m} \mid z, \tilde{T}), \tag{3.6}$$

where $p_\theta(z \mid C)$ is assumed to be a Gaussian distribution and $p_\theta(y_{n+1:n+m} \mid z, \tilde{T})$ is formed with a decoder function $d_\theta$ as in the CNP, to output a mean and standard deviation at every context point. This formulation is not exactly the same as in Garnelo et al. (2018b) where the prior is first introduced as $p(z)$, but they later use some training tricks which transform the model into the one from equation (3.6). Since $d_\theta$ is not linear, the posterior




(a) CNP model (b) LNP model

Fig. 3.3 Graphical model for the LNP and CNP models.

$p_\theta(z \mid C,T)$ is intractable and Garnelo et al. (2018b) use amortised variational inference to approximate it. The posterior is just like the prior but conditioned on a larger set, so it is reasonable to assume that it has the same form and to approximate it with another Gaussian distribution $q_\theta(z \mid C,T)$.[2] This variational approximation is characterized by a mean and standard deviation which are the output of a neural network encoder $e_\theta$. Since $q_\theta(z \mid C,T)$ and $p_\theta(z \mid C)$ have the same architecture, the same encoder as for the CNP is used for $q_\theta(z \mid C,T)$, but the representation $r$ is now split in two equidimensional vectors $\mu_z$ and $\sigma z$, so the parameters of the variational posterior are obtained as described in equations (3.7) and (3.8). The architecture of the LNP model is illustrated in figure 3.4.

$$r_i = e_\theta(x_i, y_i), \quad \forall (x_i, y_i) \in C \tag{3.7}$$

$$\mu_z, \sigma_z = r_1 \oplus r_2 \oplus \cdots \oplus r_n \tag{3.8}$$

An evidence lower-bound (ELBO) can be derived for the LNP, as for the VAE in Kingma and Welling (2013), by marginalizing over the latent variable $z$ and using Jensen's inequality with the posterior distribution $q_\theta(z \mid C,T)$,

$$\log p_\theta\left(y_{1:n+m} \mid C, \tilde{T}\right) \geq \mathbb{E}_{q_\theta(z|C,T)}\left[\sum_{i=1}^{n+m} \log p_\theta\left(y_i \mid z, x_i\right) + \log \frac{p_\theta\left(z \mid C\right)}{q_\theta(z \mid C,T)}\right]. \tag{3.9}$$

[2]The notation is overloaded on purpose here by using $\theta$ to denote the parameters of both the approximate posterior and the likelihood function $p_\theta$. We group all the parameters of the model into $\theta$.

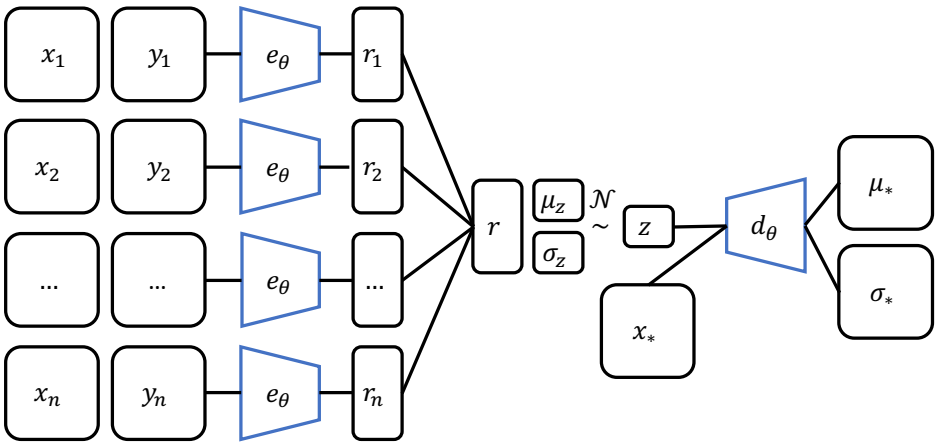



Fig. 3.4 LNP model (Garnelo et al., 2018b). Blue shapes indicate functions. The encoder $e_\theta$ maps the pair of context points to vectors $r_i$ which are then aggregated to obtain $r$, a unique representation for the whole context set. $r$ is then split in two to obtain a mean and standard deviation parametrizing a Normal distribution over the latent variable $z$. To output a mean and standard deviation which are the parameters of the predictive distribution $p_\theta(y_* \mid z, x_*)$, $z$ is sampled and fed to the decoder with the location of the target point $x_*$.

The prior $p_\theta(z \mid C)$ is intractable, so Garnelo et al. (2018b) propose to use a Gaussian variational approximation for the prior, $q_\theta(z \mid C)$, which gives,

$$\begin{aligned}\log p_\theta\left(y_{1:n+m} \mid C, \tilde{T}\right) \geq & \mathbb{E}_{q_\theta(z|C,T)}\left[\sum_{i=1}^{n+m} \log p_\theta\left(y_i \mid z, x_i\right) + \log \frac{q_\theta(z \mid C)}{q_\theta(z \mid C, T)}\right] \\ \geq & \mathbb{E}_{q_\theta(z|C,T)}\left[\sum_{i=1}^{n+m} \log p_\theta\left(y_i \mid z, x_i\right)\right] - KL(q_\theta(z \mid C, T) \| q_\theta(z \mid C)).\end{aligned} \tag{3.10}$$

The objective in equation (3.10) is not a valid ELBO anymore, but it was shown to work to train the model in Garnelo et al. (2018b). So the LNP is trained by minimizing the objective,

$$\mathscr{L}(\theta) = -\mathbb{E}_n\left[\mathbb{E}_{C,T\sim P}\left[\mathbb{E}_{q_\theta(z|C,T)}\left[\sum_{i=1}^{n+m} \log p_\theta\left(y_i \mid z, x_i\right)\right] - KL(q_\theta(z \mid C, T) \| q_\theta(z \mid C))\right]\right]. \tag{3.11}$$

$\mathscr{L}(\theta)$ is optimized with stochastic gradient variational Bayes (SGVB) (Kingma and Welling, 2013) by using a deterministic reparametrization of the stochastic sampling, called the reparametrization trick, combined with Monte-Carlo approximations of the expectation. The reparametrization trick consists in sampling a value $\varepsilon$ from a standard Gaussian distribution and then obtaining a sample $z$ form a Gaussian distribution with mean $\mu_z$ and standard

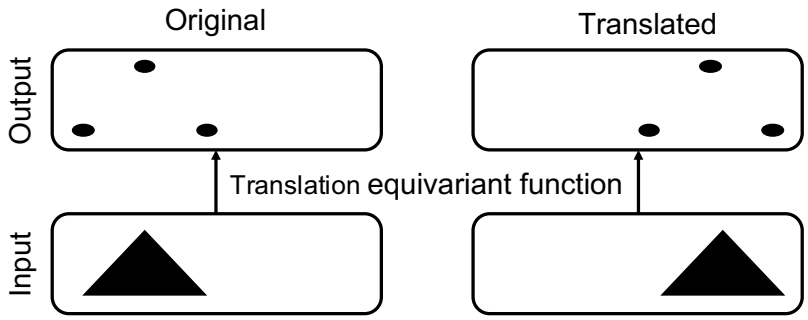



Fig. 3.5 Illustration of a translation equivariant function. The function detects the corners of the triangle. When the input is translated, the output is translated by the same amount.

deviation $\sigma_z$ as $z = \mu_z + \sigma_z \cdot \varepsilon$, such that the gradients can now be backgpropagated through $\mu_z$ and $\sigma_z$.

## 3.5 Convolutional Conditional Neural Process

A useful inductive bias when working with time series or spatial data is translation equivariance. A model satisfies translation equivariance if, when a translation is applied on the input locations, the same transformation is also applied to the predictions. This is illustrated in figure 3.5. The CNP or the LNP are not translation equivariant because the embeddings $r$ and $z$ are fixed dimensional vectors for which the notion of translation with respect to the input is not well defined. For this reason, Gordon et al. (2019) introduced the Convolutional Conditional Neural Process (ConvCNP).

The main difference between the ConvCNP and the CNP/LNP is that the ConvCNP embeds the context set into a function instead of a fixed-dimensional vector. In particular, the encoder maps the context set $C$ to a function space $H$, $e_\theta : \{X \times Y\} \to H$, where $\{X \times Y\}$ denotes the space of sets containing elements from $X \times Y$. To satisfy both translation equivariance and permutation invariance with respect to the input set, Gordon et al. (2019) prove that the encoder must be of the form,

$$h(x_*) = e_\theta(C)(x_*) = \sum_{i=1}^{n} \omega(y_i)\, \psi(x_* - x_i), \tag{3.12}$$

where $h$ is the notation for the functional representation of the context set, $x_*$ is any input location where $h$ is evaluated, $\psi : X \to \mathbb{R}$ is a continuous function and, for the applications

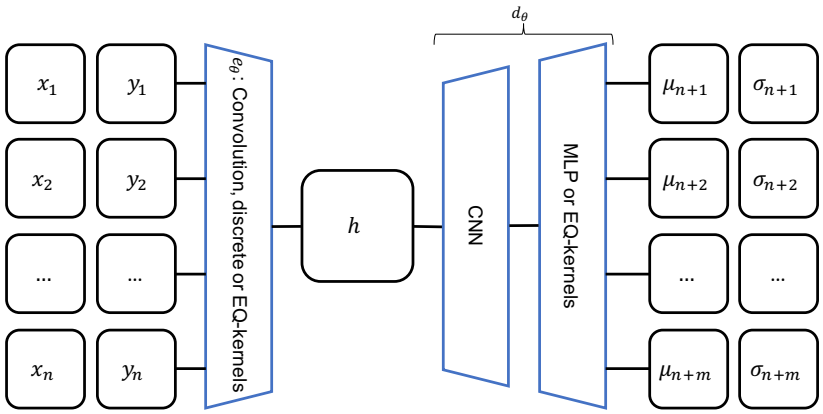



Fig. 3.6 ConvCNP model (Gordon et al., 2019). Blue shapes indicate functions. The encoder $e_\theta$ outputs a function $h$ with either a discrete convolution for the on-the-grid version or an EQ kernel for off-the-grid applications. $h$ is then mapped to a function outputting the mean and standard deviation at every target point, with the decoder $d_\theta$. $d_\theta$ is composed of two parts, first a CNN and then either a MLP applied on the CNN outputs for on-the-grid version, or evenly spaced EQ functions with weights given by the output of the CNN for the off-the-grid one.

considered in this thesis, $\omega : Y \to \mathbb{R}^2, \omega(y) = [1, y]$.[3] The decoder $d_\theta(h)(x_*)$ is a translation equivariant function mapping the functional representation $h$ to a function giving a mean and standard deviation for every target point,

$$\mu_i, \sigma_i = d_\theta(h)(x_i) \quad \forall x_i \in \tilde{T}. \tag{3.13}$$

Gordon et al. (2019) present two different architectures for the ConvCNP, one on-the-grid and one-off-the-grid. As the name suggest, the on-the-grid version handles data lying on a fixed grid, such as images. On the other hand, the off-the-grid version can handle inputs at any location, such as irregularly sampled time series. Figure 3.6 illustrates the general architecture of the ConvCNP.

In the off-the-grid version, $\psi$ in the encoder is implemented as an exponential-quadratic (EQ) kernel with a learnable parameter. Then $h$ is discretized by sampling the input $x_*$ on a uniform grid of specified density. The resulting tensor is passed through a Convolutional Neural Network (CNN) (LeCun et al., 1998), the output of which is then mapped back to a continuous function by using it as weights for evenly spaced EQ functions. This continuous function predicts the mean and standard deviation at every target points.

[3] In general $\omega(y) : Y \to \mathbb{R}^K = [1, y, y^1, \ldots, y^K]$ where $K$ is the multiplicity of $\{X \times Y\}$, i.e. the most times any input $x_i$ can repeated in the input set (see Gordon et al. (2019) for details).

The on-the-grid version is simpler because it is not necessary to use continuous functions and EQ kernels. Let $I \in \mathbb{R}^{H \times W \times C}$ be an image and $M \in \mathbb{R}^{H \times W \times C}$ be a mask matrix such that $M_{i,j} = 1$ if the pixel at location $(i, j)$ is in the context set.[4] The encoder applies a normalized convolution which consists in a convolution on the masked image $I \odot M$ divided by the same convolution applied on the mask $M$ (with $\odot$ the element-wise multiplication). The decoder has two stages, the first is a CNN as in the center part of the off-the-grid version, and the second is a multi-layer perceptron (MLP) applied independently to every pixel, to output the mean and standard deviation.

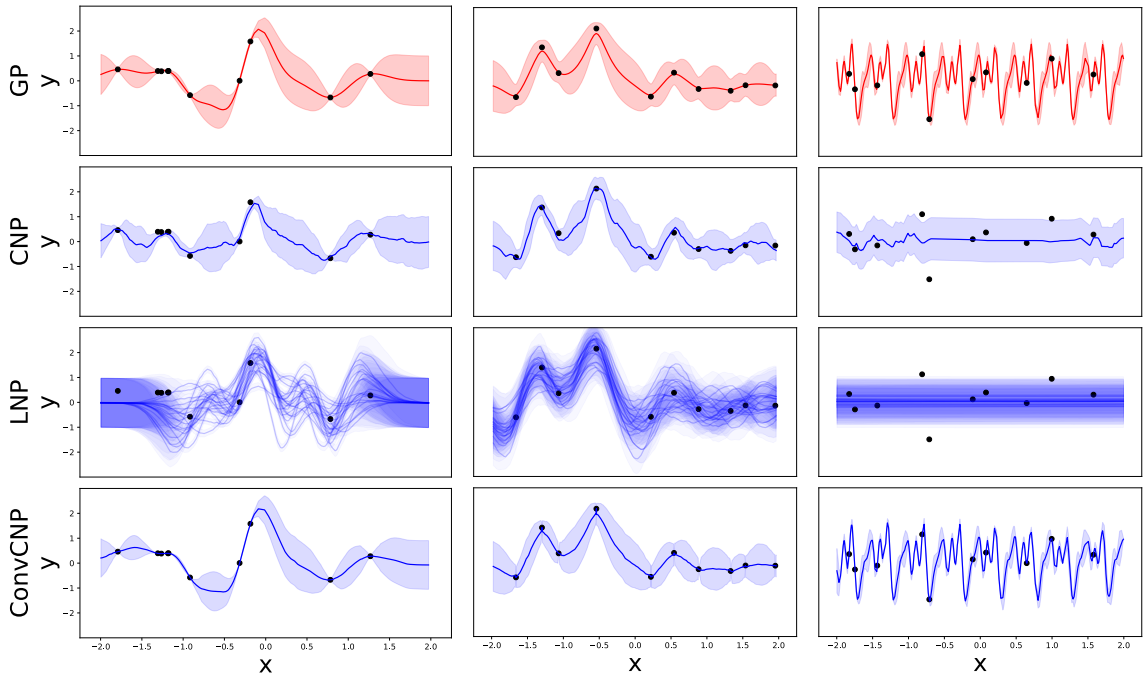
As for the CNP, the factorization assumption of the ConvCNP in equation (3.1) does not allow coherent sampling from the conditional predictive distribution. For this reason, Foong et al. (2020) introduced the Convolutional Latent Neural Process (ConvLNP, but called ConvNP in the paper). It uses the idea from the LNP to make the predictive distribution conditional on latent variables.

Since the embeddings in the ConvCNP are functions instead of fixed dimensional vectors, it is not directly obvious how to extract features representing the whole context set from them. In chapter 4, we consider how to reduce the embeddings to a fixed-dimensional feature vector representing the whole context set, which can then be used as features for classification.

## 3.6 Regression examples

NPs have traditionally been evaluated on functions generated by GPs because it is easy to generate GP data and because the predictions can be compared to the true GP conditional predictive distributions. Figures 3.7 and 3.8 illustrate the output of the CNP, LNP and ConvCNP for three different kernels: an EQ kernel with a lengthscale of 0.2, a 3/2-Matern kernel with lengthscale 0.2 and additive white noise of variance 0.1, and a periodic kernel of lengthscale 1 and period 0.5. The models were taken from Dubois et al. (2020). All models were trained for 100 epochs using 10,000 different context sets for every epochs. We refer to Dubois et al. (2020) for more details such as the exact architecture; the goal here is just to illustrate how the predictive distributions look like for the different models. The top row is the oracle GP which has the correct covariance function and hyperparameters; it constitutes and upper bound on the performance of the models. Figure 3.8 shows some typical behaviour for NPs; the CNP and the LNP underfit as their predictions don't pass exactly through all the context points and they don't extract the repetitve pattern from the periodic kernel. On

[4] $H$, $W$, $C$ are respectively the image width, height and number of channels. Even though the symbol $C$ was previously used for the context set, it is re-used on purpose here for clarity because it is very uncommon to use any other symbol to indicate the number of channels in an image.

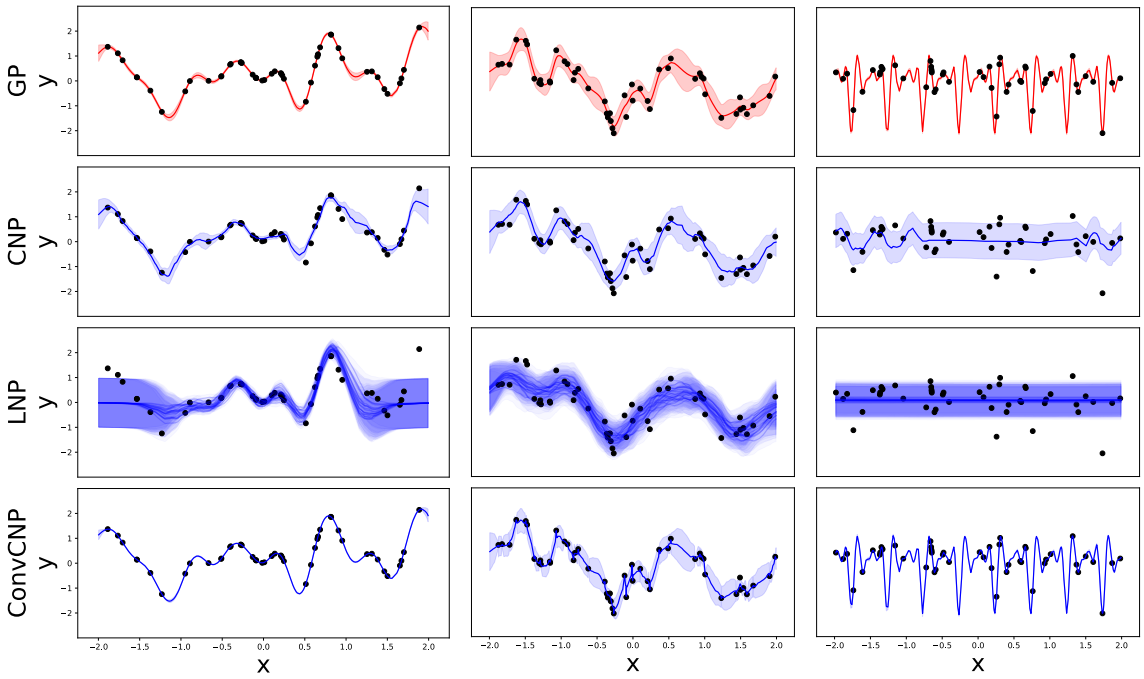



(a) EQ kernel (b) Noisy 3/2-Matern kernel (c) Periodic kernel

Fig. 3.7 Conditional posterior distributions with 10 context points, for a fully trained CNP, LNP and ConvCNP. The first row is the oracle GP with the correct covariance function and hyperparameters. The data was generated from three different Gaussian processes with an EQ kernel of lengthscale 0.2, a Matern-3/2 kernel with lengthscale 0.2 and additive white noise of variance 0.1, and an EQ periodic kernel with lengthscale 1 and period 0.5. Models were taken from the open-source code from Dubois et al. (2020) and trained for 100 epochs with 10,000 different context sets at every epochs.

the other hand, the ConvCNP does pass through all the context points and makes very good uncertainty predictions. It also correctly identifies the periodic pattern. In general, when the translation equivariance hypothesis is valid, the ConvCNP tends to show better results than the CNP/LNP.

In the later chapters, we will use the NPs to work with images, so we illustrate in figure 3.9 the application of the CNP and the ConvCNP to the MNIST dataset (LeCun et al., 1998) with 100 context pixels per image. We show the predicted mean and standard deviation for every pixel. The models used to obtain the results in figure 3.9 are the CNP and ConvCNP models that will be detailed in chapter 4.




(a) EQ kernel (b) Noisy 3/2-Matern kernel (c) Periodic kernel

Fig. 3.8 Conditional posterior distributions with 50 context points, for a fully trained CNP, LNP and ConvCNP. See figure 3.7 for the details.

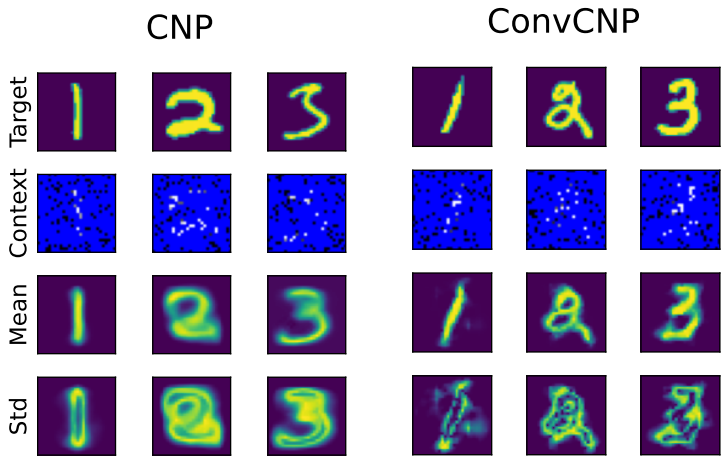



Fig. 3.9 Illustration of the image reconstruction task with NPs, using 100 pixels. We show the predicted mean and standard deviation for every pixel. The models used to obtained these results are the CNP and ConvCNP models from chapter 4.

# Chapter 4

# Neural Processes as feature extractors

Our goal is to build a NP model which extracts good classification features through the density modelling task. In order to do that, we build a NP model which jointly predicts the target values and the class assignments, and we share features between the two parts of the model.

In this chapter we use a two step learning procedure where we first pre-train the model on the density modelling task, and then we fine-tune it on the classification task; we will consider joint training in the later chapters. With this learning procedure, we use a probabilistic model which relies on the assumption that the predictive distribution for the target values $y_{1:n+m}$ and the classes $\lambda$ are conditionally independent given the context:

$$p_\theta(y_{1:n+m}, \lambda \mid C, \tilde{T}) = p_\theta(\lambda \mid C) p_\theta(y_{1:n+m} \mid C, \tilde{T}), \tag{4.1}$$

where $C$ is the context set and $\tilde{T}$ is the set of target locations. While this assumption is clearly false, it is convenient to start with a simple model. We will evaluate different architectures for our model, by comparing them on the MNIST digit classification task.

The semi-supervised method considered in this chapter falls into the change of representation category from section 2.2; we first use the unlabelled data to learn a mapping from input to embeddings, and then perform fully supervised learning on those new embeddings.

Section 4.1 covers some related work. Section 4.2 establishes some baselines for comparing the models. The different architectures considered are detailed in section 4.3 and then evaluated in section 4.4. Finally, we present a summary and discussion of the results in section 4.5.

## 4.1 Related work

As mentioned in section 2.3, the change of representation methods in semi-supervised learning can often be interpreted as self-supervised methods. In fact, in this case, since we will first pre-train the NPs to reconstruct missing values in images, it can be viewed as crafting an inpainting task to pre-train a model which is then used as an initialization for a classification task. However, we prefer to view as a joint probabilistic model $p_\theta(y_{1:n+m}, \lambda \mid C, \tilde{T})$ because the further developments in chapter 5 and 6 fit in this model.

## 4.2 Baselines

As stressed in section 2.4, it is important to construct good fully-supervised baselines to make a fair analysis of the performance of semi-supervised algorithms. For this reason, we present five different baselines and we cover a large range of model complexity. We include the WideResNet model as Oliver et al. (2018) have found that it generalizes very well in low data regimes. The baselines used are:

- A logistic regression (LR) classifier,
- A support vector machine (SVM) classifier with a linear kernel,
- A k-nearest neighbour (KNN) classifier,
- A LeNet classifier (LeCun et al., 1989) with different sizes: a large, medium and small version,
- A WideResNet (Zagoruyko and Komodakis, 2016) with one version using a random initialization and one pre-trained on the ImageNet dataset (Krizhevsky et al., 2012).

The exact architecture of the LeNet classifier is detailed in Appendix A.1. For the model details of the WideResNet, we refer to (Zagoruyko and Komodakis, 2016). We use the sklearn implementation (Pedregosa et al., 2011) for the LR, SVM and KNN classifiers.

## 4.3 Methodology

To extract features from NPs, we first train the models to predict $p_\theta(y_{1:n+m} \mid C, \tilde{T})$, by sampling images and randomly masking pixels at the locations in $\tilde{T}$. Then, we extract an embedding from the model and use it as classification features to feed to the classifier.

### 4.3.1 Architectures

The architecture of the model defines the inductive biases in the probabilistic model. It is therefore important to carefully choose the structure and where/how to extract the features. We will consider three different architectures in this chapter, the CNP, the ConvCNP and the UNetCNP. All models will be thoroughly investigated in section 4.4.4 as part of an experimental model selection. All the models detailed below and in the later chapters were independently (re)-implemented on Pytorch (Paszke et al., 2019). The code is available here ; some functions were copied form the repository from Gordon et al. (2019) and also from the one from Dubois et al. (2020) . All functions copied are referenced in the coder, just under the function definition.

#### CNP

The CNP architecture is the same as in the original paper (Garnelo et al., 2018a), the encoder $e_\theta$ is a 3 layer MLP with 128 units, and the decoder is a 5 layer MLP also with 128 units except at the last layer where it has two values per output dimensions for the mean and standard deviation. The embedding $r$ is 128 dimensional. The fixed-dimensional vector $r$ is the only variable in the CNP which represents the whole context set so it is an obvious choice as features for classification.

#### ConvCNP

Since the model will be applied on images, the on-the-grid version of the ConvCNP can be used. We implemented the same architecture as in Gordon et al. (2019). The encoder is a normalized $9 \times 9$ convolution, the central CNN is composed of 4 residual blocks each containing one $5 \times 5$ convolution, and the decoder is a 5 layer MLP with 64 hidden units, except for the last layer. All convolutions have 128 features and are depthwise separable (Chollet, 2017) for parameters efficiency. $e_\theta$ is restricted to be a non-negative kernel to satisfy the restrictions listed in equation (3.12). We consider extracting classification features by average-pooling over any of the layers in the CNN part of the network, to allow the model to be transferable to both the on-the-grid and off-the-grid versions.

#### UNetCNP

To help the model separate high-level from low-level features, we propose to use a UNet architecture (Ronneberger et al., 2015) as the CNN center part of the ConvCNP. The intuition is that the model will let local information pass through the top skip-connections and global

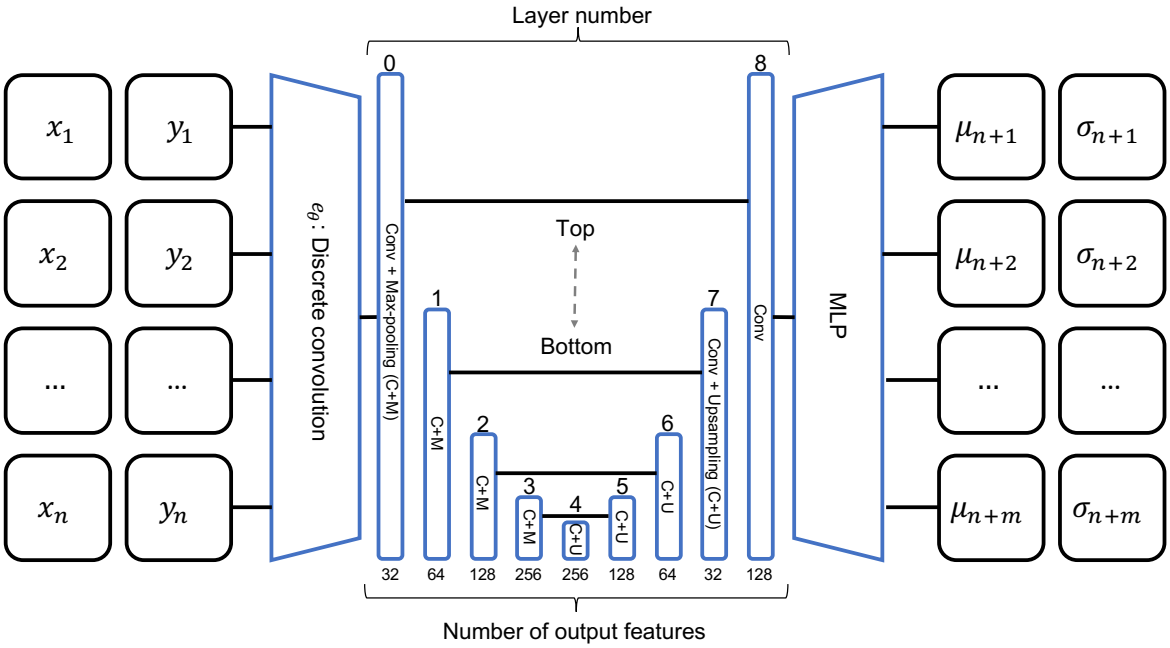
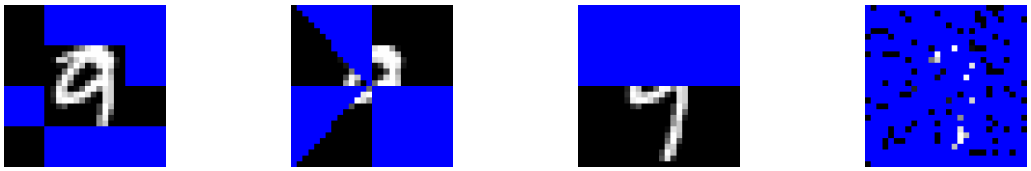



Fig. 4.1 Architecture of the UNetCNP model. Blue shapes indicate functions. We replace the CNN central part in the ConvCNP with a UNet. We define what top and bottom means in terms of the skip connections.

features would be extracted at the bottom to make predictions at locations far from context pixels. The model is illustrated in figure 4.1. The encoder and MLP part of the decoder are exactly the same as in the ConvCNP. On the way down, the UNet is composed of 4 blocks of $5 \times 5$ convolutions with doubling number of features from 32 until 256 and $2 \times 2$ max-pooling layers. The way up is symmetric with nearest neighbour upsampling and concatenation with the skip connections coming from the way down. The final convolution has 128 features to fall back on the MLP part of the ConvCNP decoder. Again, we consider extracting the embeddings by pooling over any of the layers in the UNet.

### 4.3.2 Type of context sets: uniform vs semantic

NPs are traditionally trained to make conditional predictions given independent random pixels. However, this makes the average gap between context values small, so the NPs can achieve very good performances by simply acting as smooth interpolation functions. That is not a desirable property for a model which should extract global features transferable to classification. Therefore, we modify the context point sampling procedure in the density modelling task such as to leave out whole regions of the input space, which encourages the model to extract semantic information to impute the missing values. With images, we add



(a) Blocks (b) Pizza (c) Cut (d) Random

Fig. 4.2 Different methods for sampling context sets in images. Blue pixels are masked. The three first methods (blocks, pizza and cut) encourage the model to extract semantic information to complete the image as it cannot rely on local smoothness properties.

three new ways of sampling random context pixels, which are illustrated in figure 4.2, along with the original random independent uniform sampling. For conciseness, we will refer to this new modified reconstruction task as the *semantic* task, and to the original one as the *uniform* task. The methods used to leave out chunks of the image are:

- Blocks: the image is divided in 16 squared blocks and every block has probability 0.5 to be included in the context set,
- Pizza: the image is divided in 8 triangular slices and every slice has probability 0.5 to be included in the context set,
- Cut: the image is cut in two along the vertical or the horizontal axis and every of the four possible context sets (left, right, top and bottom) have equal probability 0.25 to be selected.

### 4.3.3 Classification model

We select a number of classifiers to test how good our models are at extracting classification features; we use multiple of them to make the analysis as general as possible. We cover a range of complexity by including simple LR, KNN and SVM classifiers, and more complex MLP models. For the MLP, we work with three sizes again: small, medium and large. The number of nodes in the last MLP layer is fixed to one node for every possible class, but the other layers vary:

- Small: one hidden layer with 10 nodes.
- Medium: two hidden layers with 64 nodes.
- Large: four hidden layers with 128 nodes.

All layers are ReLU activated with 30% dropout rate and an L2 regularization term of $10^{-5}$ to counter over-fitting. For the MLP classifiers, both fine-tuning and freezing the NP weights will be considered.

## 4.4 Results

We test our models on the MNIST image classification task. To see how the performance varies with the size of the labelled set, we test them in situations where we have 10, 20, 40, 60, 80, 100, 600, 1000 and 3000 labelled images to train on, while ensuring that there is an equal number of samples for every class. For the model selection, we stick to the standard benchmark of 100 labelled images and we use a validation set for the evaluation.

In section 4.4.1, we first explain how we select a validation set. Section 4.4.2 presents the results achieved with the baselines. The model selection is detailed in section 4.4.4 and, finally, section 4.4.5 shows the results achieved by the selected models.

### 4.4.1 Validation set

As mentioned in section 2.4, choosing hyper-parameters is very expensive in the semi-supervised setting. Since we train models with a number of labelled images ranging from 10 to 3000, it is not possible to keep a realistic held-out validation set. Using cross-validation is not an option either as early stopping for neural networks requires having a validation set large enough to accurately detect over-fitting. For example, using 4 labelled validation images when doing 5-fold cross validation on 20 labelled images would not be sensible as the accuracy could only be measured in steps of 0.25. For this reason, we adopt the approach from Rasmus et al. (2015). We hold-out 1000 of the MNIST training set as validation data. Obviously this would not be possible in a real semi-supervised application. However, as argued in Rasmus et al. (2015), MNIST is such a simple task that 100 labelled images correspond to a far greater number of images in most other tasks, for which it would thus be realistic to assume that we have 1000 labelled images. Nevertheless, since we cheat with a large validation set, we will be careful not to do any extensive hyper-parameter tuning which would make the results unrealistic. If tuning is necessary, such as with simple the LR, KNN or SVM models, for which the performance depends a lot on the regularization term or the number of neighbours, we tune those parameters with a realistic set composed of 10% of the number of labelled images, and limited to a minimum of 10 images.

### 4.4.2 Baselines

Figure 4.3 shows the mean and standard deviation of the accuracies achieved over 10 runs by the different fully-supervised baselines when tested on the full MNIST test sets. To tune the hyper-parameters for the LR, SVM and KNN model, we use realistic validation set with 10% of the number of labelled images (with a minimum of 10). For the neural network architectures, we used the fixed size validation set with 1000 labelled images for early stopping, as explained in section 4.4.1. The training details such as the learning rate and batch size are given in appendix A.2. The WideResNet network is a larger network which takes much longer to train so it was not possible to run it multiple times.

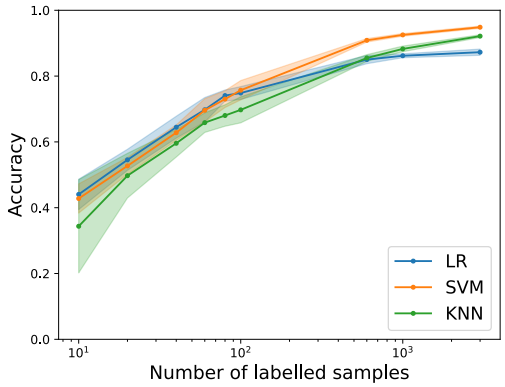
The poor results of neural network models with small number of labelled data are due to over-fitting, in fact the learning curves showed that the validation accuracy started decreasing after only few training epochs. We select the SVM classifier as a baseline for future comparisons, because it achieved the best results with both small and large number of labelled images.

### 4.4.3 Neural Processes training

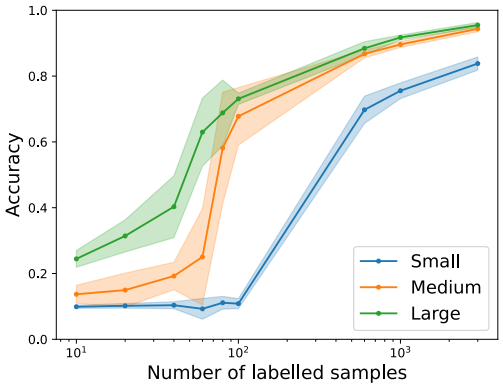
To use the NPs as feature extractors, it is first necessary to pre-train them on the density modelling task. We train the models on the whole MNIST training dataset by minimizing the NP training objective $\mathscr{L}(\theta)$ from equation (3.5). We use the adam optimizer (Kingma and Ba, 2014) for 400 epochs with a batch size of 64 and a learning rate of $10^{-4}$. Figure 4.4 shows the predicted mean and standard deviation for some example images when training the model with the semantic or the uniform task. More examples with one image per digit are shown in appendix B.1. As seen in section 3.6 with the GP data, the CNP underfits because it doesn't reconstruct the context pixels exactly.

### 4.4.4 Model selection

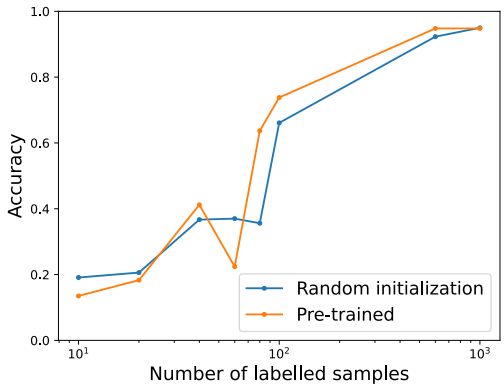
In this section we search for an architecture which best fits our target model: a NP which extracts good features for classification. For all the models, we investigate whether training the models on the semantic density modelling task improves compared to using the uniform task. For the ConvCNP and the UNetCNP, the layers from which the representation is extracted must also be selected. Since neural networks are hard to train on small datasets, we use an SVM classifier to make a robust evaluation of the quality of the features extracted for classification.




(a) LR, KNN and SVM

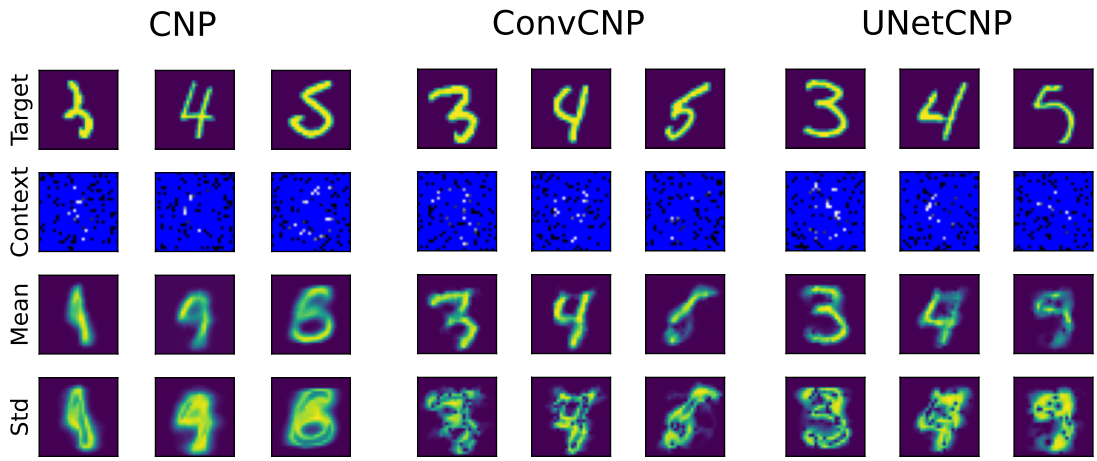



(b) LeNet

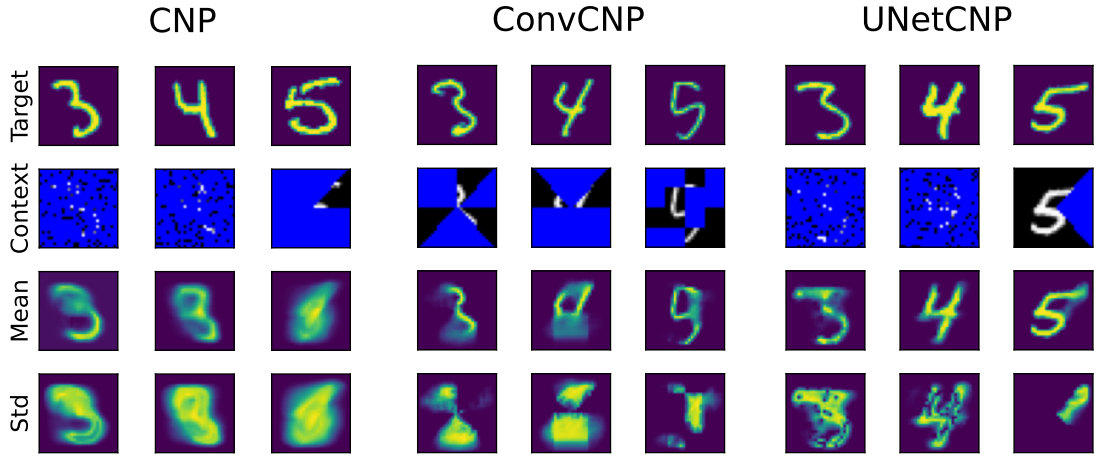



(c) WideResNet

Fig. 4.3 Fully-supervised baselines on the MNIST dataset. All classifiers are trained using only labelled images, so from only 10 up to 3000 images. All classes are equally represented in the training sets. The mean and standard deviation are computed over 10 runs. The WideResNet was run only once for computational reasons. The training details are given in appendix A.2. The pre-trained Wide ResNet was trained on the Imagenet dataset (Krizhevsky et al., 2012).

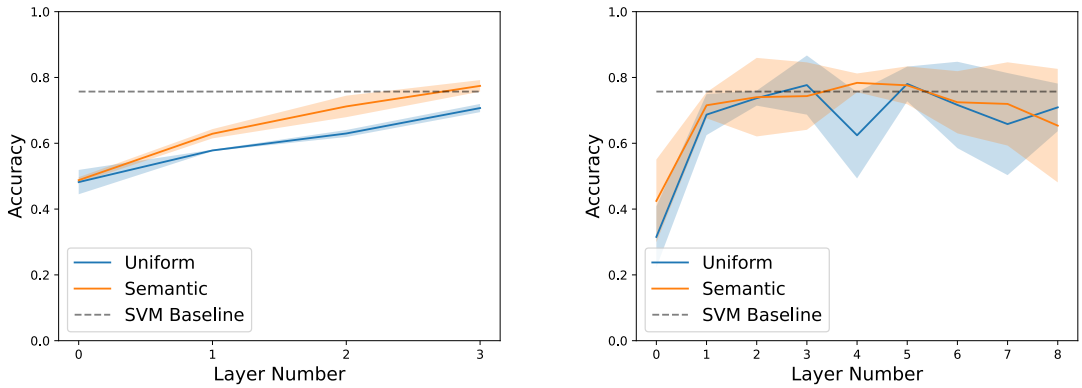



(a) Uniform task




(b) Semantic task

Fig. 4.4 Image inpainting results with the different Neural Processes which will be used to extract features for classification. All models were trained for 400 epochs on the entire MNIST training data with a batch size of 64 and a learning rate of $10^{-4}$.




(a) ConvCNP with average pooling

(b) UNetCNP with average pooling

Fig. 4.5 Results of the model selection with an SVM classifier applied on the features extracted. The accuracy is shown as the mean and standard deviation computer over 10 runs. The features are extracted by average-pooling over the UNetCNP or ConvCNP layers. The layer numbers correspond to the ordering in the computational graph, so for the UNet: 0-3 is the way down, 4 is the bottom, and 5-8 is the way up. We compare two density modelling tasks: one using only uniform independent sampling of context pixels (uniform task) and cropping out larger semantic blocks from the image (semantic task).

Figure 4.5 shows the accuracy achieved by the SVM classifier against the layer where features are extracted for the ConvCNP and the UNetCNP. The layer numbering corresponds to the ordering in the computational graph, so for the UNet: 0-3 is the way down, 4 is the bottom, and 5-8 is the way up. We report the average over 10 runs, with standard deviations.

For the ConvCNP, figure 4.5a shows that the optimal configuration is clearly obtained by pre-training the model on the semantic task and by average-pooling the features from the last CNN layer.

The drop in accuracy at layer 4 when using UNetCNP pre-trained on the uniform task shows that the model does not extract good classification features in the bottom layer. On the other hand, the UNetCNP pre-trained on the semantic task does not suffer from that problem and actually the features extracted in the bottom layer seem to be the best for classification as they yield the best accuracy over all configurations. Therefore, we select average pooling over the bottom layer and pre-training on the semantic task as our optimal UNetCNP model.

There are no architectural choices to be made in the CNP model because the only possibility is to use the global representation $r$ as classification features, but the two types of context pixel sampling procedures for the pre-training should be considered. Figure 4.6 shows a box-plot comparing the accuracy achieved when using a CNP pre-trained on the

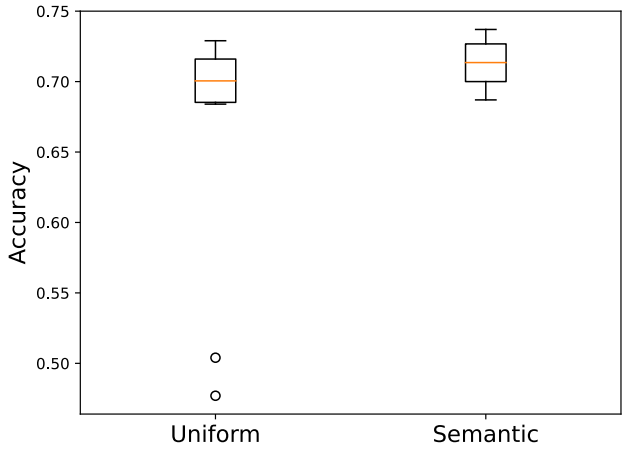



Fig. 4.6 Box-plot of the accuracy achieved by an SVM classifier on the representation $r$ of the CNP. The CNP was either trained with the uniform or the semantic task. See section 4.3.2 for the details of those two tasks.

uniform or the semantic task. Except from two outliers, there is no significant difference, so the semantic task is selected for consistency with the other models.

### 4.4.5 Results of the selected models

We now test the model configurations selected in section 4.4.4 with the different classifiers proposed in section 4.3. Figure 4.7 shows the results achieved with a number of labelled images ranging from 10 to 3000. The models are evaluated on the validation set instead of the MNIST test set because those results are used to select a model which will be extended in chapter 5. The neural network training details are given in appendix A.2. The LR, KNN and SVM results are averaged over 10 runs with standard deviations but, due to computational constraints, it was not possible to do the same with the neural network architectures.

The results in figure 4.7 show that the CNP and the ConvCNP models fail at extracting good features for classification since the lines are below or equal to the baseline.

On the other hand, the UNetCNP shows good results. Most classifiers reached accuracies better than the baseline with small number of labelled images. Besides, the fact that not updating (i.e. freezing) the NP weights with the MLP classifier improved the classification performance gives us further evidence that UNetCNP extracts useful classification features. Although the UNetCNP leads to results superior to our fully-supervised baselines, the results are still low compared to other standard semi-supervised techniques, for example the M1+M2 model from Kingma et al. (2014) achieves 96.7% accuracy with 100 labelled MNIST images.

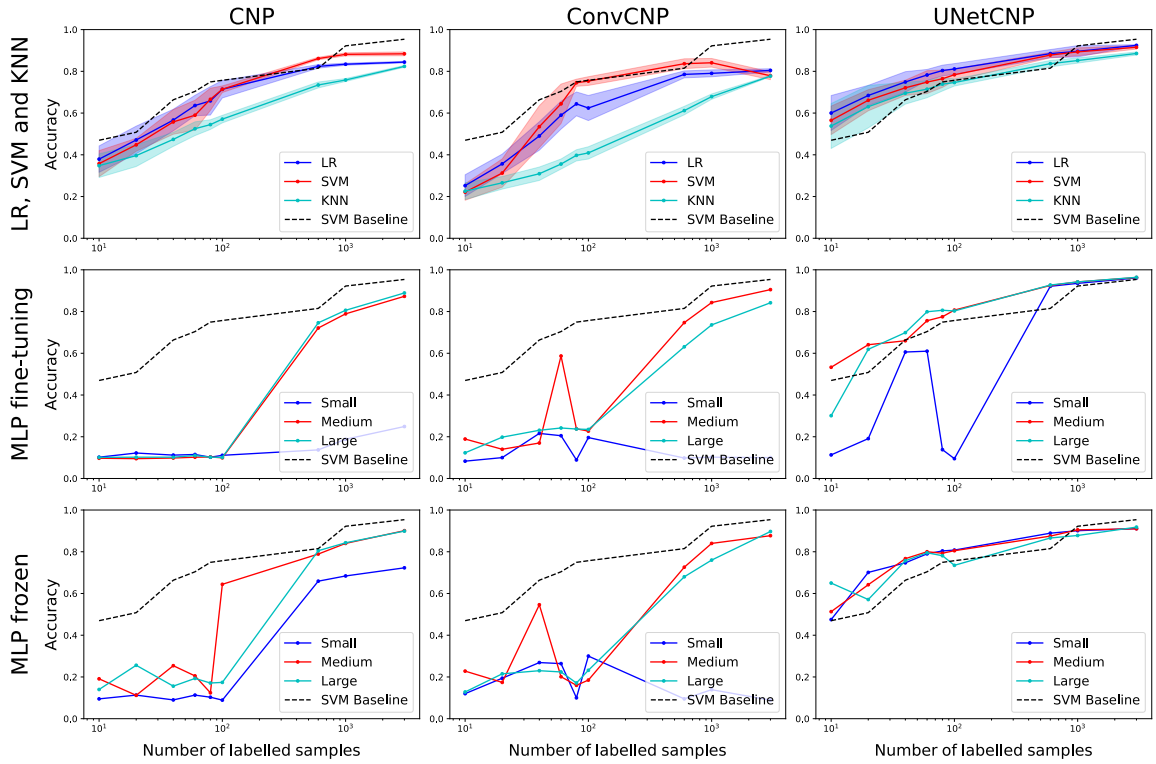



Fig. 4.7 MNIST digit classification accuracy against the number of labelled samples for the models selected in section 4.4.4. We include for comparison the SVM baseline classifier applied on the MNIST images directly. All NPs are pre-trained to reconstruct MNIST images with missing chunks (the semantic task), by using the entire training set. Then we train some classifiers with a varying number of labelled images, using as input features an embedding extracted from the NPs. We test LR SVM, KNN and MLP classifiers. The CNP embedding is extracted as the representation $r$ from equation (3.3). The ConvCNP representation is the result of an average pooling over the input dimensions on the last layer of the CNN in the center part of the model. Finally, the UNetCNP embedding is obtained with an average pooling on the layer at the bottom of the UNet.

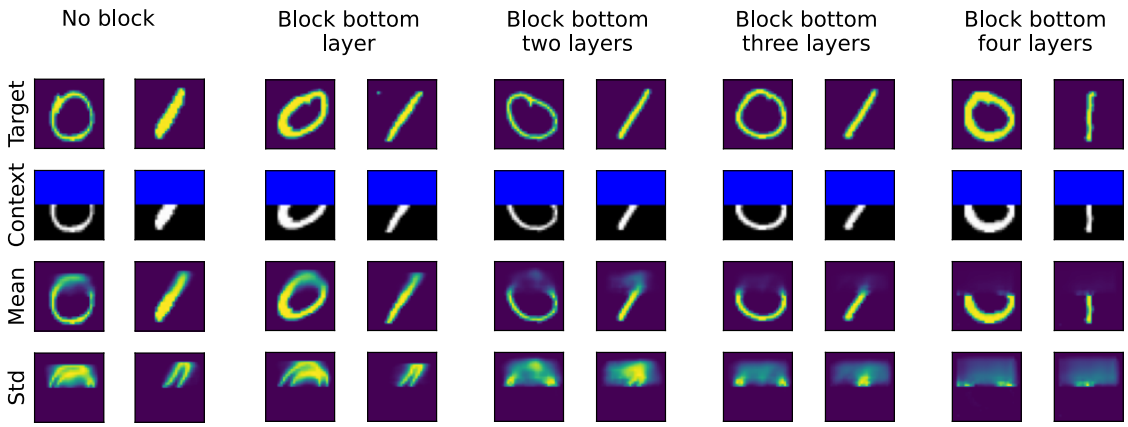



Fig. 4.8 UNetCNP predictive distribution (characterized by a mean and standard deviation) when blocking different residual connections. Blocking bottom $x$ layers means that the residual connections for the $x$ layers at the bottom of are multiplied by 0 at every entry, thus suppressing information. Keeping only the information from the top layers (i.e. blocking the three or four bottom layers) makes the model only reconstruct the context but not extrapolate to the missing parts, while including the bottom connections allows the model to make sensible predictions. It suggests that the UNetCNP extracts high level semantic information from the image in the bottom layers.

### 4.4.6 Investigation of the good performances of the UNetCNP

To understand what type of features are extracted by the different layers in the UNetCNP, we propose a simple ablation study where different skip connections are set to zero in the UNet, effectively suppressing the information passing through. We compare the predictions made with those models with missing connections to the normal UNetCNP where all connections are included. We choose to block all the connections up to a certain layer. We say for example "blocking bottom two layers" to say that we we block the skip connections for the two levels at the bottom of the UNet. The predicted mean and standard deviation when blocking different layers are shown in figure 4.8, with an input image where the top half of the pixels are missing. We do not re-train the models. Only keeping one or two skip connections at the top of the UNet, so blocking the bottom three or four layers, makes the model copy the context points but not extrapolate to the missing part. On the other hand, when all the layers are included, the UNetCNP is able to make sensible predictions for pixels even far away from context points. This supports our original hypothesis that the UNet architecture encourages the model to separate high-level from low-level information.

## 4.5 Summary and discussion

**Summary** In this chapter we tested the quality of the features extracted by the CNP, the ConvCNP and the UNetCNP by pre-training them on the density modelling task and using the features extracted as input to a classifier. This approach was superior to purely supervised learning when using the UNetCNP, but it was not competitive with the state of the art.

**Strengths** The accuracy achieved by using the UNetCNP was higher than the baseline by 10 to 20% for low data regimes, and the results from section 4.4.6 suggest that the UNetCNP separates high-level and low-level features through the different skip-connections.

**Limitations** One potential reason why the CNP and the ConvCNP do not extract good features is that the density modelling task is optimized when the models reconstruct the pixel values as precisely as possible, which encourages the extraction of lower level information that is useful for fine-grained predictions. Choosing a better architecture with the UNetCNP showed to be a solution to this problem. Nevertheless, the accuracy achieved with 100 labelled MNIST images was around 80%, which is in the lower range of the results achieved by the methods presented in the taxonomy of semi-supervised algorithms in table 2.1. So we have reasons to think that the features extracted by our UNetCNP can be further improved. In chapter 5, we will try to achieve that by jointly training the generative and the discriminative part of the model.

The best classifiers in figure 4.7 are the LR/SVM models and the medium MLP. In chapter 5, we will keep the MLP classifier as it is more flexible and it can easily be jointly trained with the UNetCNP by implementing both in a single Pytorch model (Paszke et al., 2019).

# Chapter 5

# Semi-supervised learning with the UNetCNP

In chapter 4, we built a model which was trained in two stages, by first pre-training a NP and then fine-tuning a classifier on the features extracted. The UNetCNP architecture proved to extract the best classification features. However, the accuracy achieved was around 80% with 100 labelled MNIST images, which is low compared to other semi-surprised algorithms, for comparison Kingma et al. (2014) report an accuracy of 96.7% for the M1+M2 model.

In this chapter we try to bridge the gap to other standard semi-supervised learning methods by jointly training the generative part $p_\theta(y_{1:n+m} \mid C, \tilde{T})$ and discriminative part $p_\theta(\lambda \mid C)$ of our model. We also experiment with two ideas drawn from the semi-supervised and self-supervised literature respectively. For the first, we include a consistency regularization loss into the training objective. For the second, we craft an additional task to train our model on, with the hope that it helps learning features transferable to the classification task.

Section 5.1 compares our method to another related model which is also trained by jointly optimizing a generative and a discriminative component. The joint training procedure and the extensions added to our model are presented in detail in section 5.2. In section 5.3, we discuss how to weight losses in multi-loss functions and we propose a new modified version of the GradNorm method from Chen et al. (2018). In section 5.4, we perform an ablation study to identify the key components to our model and we then test it on the SVHN and CIFAR10 datasets (Krizhevsky et al., 2009; Netzer et al., 2011). In section 5.5, we evaluate the ability of our model to handle subsets of data as input. Finally, section 5.6 discusses and summarizes the findings from this chapter.

## 5.1 Related

In the deep invertible generalized linear model (DIGLM) from Nalisnick et al. (2019), a normalizing flow is shared between a generative component for solving the density modelling task and a discriminative component for the classification modelling task. We use the same idea in our model because it shares a DeepSet architecture to also solve both a density and a classification modelling task. Besides, the DIGLM is also jointly trained to learn the best set of shared parameters. Our model is more flexible than the DIGLM because, thanks to the DeepSet architecture, it can also support subsets of data, such as images with missing inputs.

## 5.2 Methodology

### 5.2.1 Model

We keep the UNetCNP architecture from chapter 4 as it was shown to extract classification features of the best quality. To summarize the architecture, it is a ConvCNP model with a UNet as the central CNN part, and the classification features are obtained by average pooling over the bottom layer of the UNet. The classifier is a medium-sized MLP with dropout.

### 5.2.2 Training

To learn better weights for the shared part of the model, we propose to jointly train the UNetCNP on both the density modelling task and the classification task. The training objective is obtained by minimizing the negative log-likelihood:

$$-\log p_\theta(y_{1:n+m}, \lambda \mid C, \tilde{T}) = \underbrace{-\log p_\theta(\lambda \mid C)}_{L_{\text{sup}}(\lambda, C; \theta)} \underbrace{-\log p_\theta(y_{1:n+m} \mid C, \tilde{T})}_{L_{\text{rec}(C,T;\theta)}}. \tag{5.1}$$

We transform equation (5.1) into a weighted sum $w_{\text{rec}} \cdot L_{\text{rec}} + w_{\text{sup}} \cdot L_{\text{sup}}$, where the weights $w_{\text{rec}}$ and $w_{\text{sup}}$ are used to help the optimization converge to a shared local minimum for both losses. Since input images have missing pixels, not all supervised losses should have the same weight; when 99% of the pixels are missing, it is a much harder task for the classifier to make a correct prediction. Therefore, following Dubois (2019), we further weight the losses with a convex combination using the ratio of context pixels $n$ to the total number of pixels $n+m$. There are thus two type of weights, the loss-specific weights ($w_{\text{rec}}$ and $w_{\text{sup}}$) and the

convex weights which vary for every context set. The joint loss is

$$\mathscr{L}(\theta) = \mathbb{E}_{n,C,T,\lambda\sim P}\left[w_{\text{rec}} \cdot \frac{m}{n+m} L_{\text{rec}}(C,T;\theta) + \mathbb{I}(\lambda) \cdot w_{\text{sup}} \cdot \frac{n}{n+m} L_{\text{sup}}(\lambda,C;\theta)\right], \tag{5.2}$$

where $n, C, T, \lambda \sim P$ is short for saying that we sample the sets $C$ and $T$ with $n$ context points from a stochastic process $P$ and obtain the corresponding label $\lambda$, and $\mathbb{I}(\lambda) = 1$ if the sample is labelled and 0 otherwise. In practice, the expectation in equation (5.2) is approximated using a Monte-Carlo estimation with batches composed of randomly selected images from the training set, with a random number of context pixels. We again use the adam optimizer (Kingma and Ba, 2014) to minimize $\mathscr{L}(\theta)$, with a batch size of 64 and a learning rate of $2 \cdot 10^{-4}$.

### 5.2.3 Auxiliary consistency loss

Inspired by the state of the art results achieved by the algorithms under the consistency regularization category from the taxonomy in section 2.2, we consider adding a consistency loss to our objective function. Consistency losses are usually added as an ad-hoc loss but we propose one way to include it into our probabilistic model. We view it as an inductive bias which forces two context sets from the same image to be predicted as belonging to the same class. Encoding this bias into the model manually is a very hard task, so we use the consistency loss in order to constrain the optimization procedure to learn weights which make the model have consistent predictions.

The consistency loss should minimize the divergence between the class predictive distributions of two context sets from the same image. It is desirable for this metric to be symmetric since we have no prior knowledge to define the direction of the divergence when given two context sets. Following Dubois (2019), we select the Jensen-Shannon (JS) divergence because it is symmetric and bounded between 0 and $\log 2$.[1]. To prevent the model from falling in a potential local optimum where all samples have the same class predictions, we also maximize the divergence between the class predictions of two context sets coming from different images. If we define $(C^{(1)}, T^{(2)})$ and $(C^{(2)}, T^{(2)})$ as two different context and target sets from the same image and $(C^{(3)}, T^{(3)})$ as a context set from a different image, then

[1] if we use the natural logarithm

the consistency loss is

$$\begin{aligned}\mathscr{L}_{\text{cons}}(\theta) =& \mathbb{E}_{C^{(1)},T^{(1)},C^{(2)},T^{(2)},C^{(3)},T^{(3)}\sim P}\left[JS(p(\lambda \mid C^{(1)}\|p(\lambda \mid C^{(2)}) - JS(p(\lambda \mid C^{(1)}\|p(\lambda \mid C^{(3)})\right] \\ :=& \mathbb{E}_{C^{(1)},T^{(1)},C^{(2)},T^{(2)},C^{(3)},T^{(3)}\sim P}\left[L_{\text{cons}}(C^{(1)},C^{(2)},C^{(3)};\theta)\right] \\ =& \mathbb{E}_{C^{(1:3)},T^{(1:3)}\sim P}\left[L_{\text{cons}}(C^{(1)},C^{(2)},C^{(3)};\theta)\right],\end{aligned} \tag{5.3}$$

In practice we again approximate this with Monte-Carlo estimations by sampling batches containing two different context sets for every images and selecting $C^{(3)}$ as one context set from any other image.

### 5.2.4 Auxiliary siamese task

Self-supervised research is concerned about using the data to craft tasks which do not require human annotations but which require the model to learn good features transferable to downstream tasks. Such approach do not fit into our probabilistic modelling approach, since we would be creating a completely new task to solve. However, we do make an attempt to see if it can help and we include an additional loss in the objective function for a new task which consists in determining if two disjoint context sets come from the same image. We expect that being able to solve this task requires extracting high level properties from the image and determining whether they are compatible are not. Inspired by siamese networks (Koch et al., 2015), we use the absolute difference between the last hidden layer features in the MLP classifier as input to a linear model predicting whether two context sets are from the same image. We call this the *siamese* loss. If we write $p_\theta(C^{(i)},C^{(j)})$ the predicted probability that two context sets are from the same image, then the siamese loss is

$$\begin{aligned}\mathscr{L}_{\text{sia}}(\theta) =& \mathbb{E}_{C^{(1:3)},T^{(1:3)}\sim P}\left[-\log p_\theta(C^{(1)},C^{(2)}) - \log\left(1 - p_\theta(C^{(1)},C^{(3)})\right)\right] \\ :=& \mathbb{E}_{C^{(1:3)},T^{(1:3)}\sim P}\left[L_{\text{sia}}(C^{(1)},C^{(2)},C^{(3)};\theta)\right],\end{aligned} \tag{5.4}$$

where we use the same notation as in equation (5.3) but the pairs $(C^{(1)},C^{(2)})$ and $(C^{(1)},C^{(3)})$ are context sets at disjoint locations.[2]

[2]We overload the notation on purpose for conciseness by using the same notation as for the consistency loss. If the siamese loss is included then we assume disjoint locations for $C^{(1)}$ and $(C^{(2)},C^{(3)})$, otherwise not.

If we combine all the losses together, the general objective is

$$\begin{aligned}\mathscr{L}(\theta) =& \mathbb{E}_{C^{(1:3)},T^{(1:3)},\lambda^{(1:3)}\sim P}\Bigg[\sum_{j=1}^{3}\Bigg(w_{\text{rec}}\cdot\frac{m^{(j)}}{n^{(j)}+m^{(j)}}L_{\text{rec}}(C^{(j)},T^{(j)};\theta)\\ &+\lambda\cdot w_{\text{sup}}\cdot\frac{n^{(j)}}{n^{(j)}+m^{(j)}}L_{\text{sup}}(\lambda^{(j)},C^{(j)};\theta)\Bigg)\\ &+w_{\text{cons}}\cdot L_{\text{cons}}(C^{(1)},C^{(2)},C^{(3)};\theta)\\ &+w_{\text{sia}}\cdot L_{\text{sia}}(C^{(1)},C^{(2)},C^{(3)};\theta)\Bigg].\end{aligned} \tag{5.5}$$

## 5.3 Gradient weighting

The objective function $\mathscr{L}(\theta)$ from equation (5.5) combines different losses together. It is important to weight those losses to avoid situations where the gradients from one task dominate the gradients from the other tasks, because it could lead to a local minimum where one loss is minimized but the others are ignored (Chen et al., 2018). Most papers use an ad-hoc procedure where the weights are manually chosen; it is the case for example for the CatGAN (Springenberg, 2015), the Π-model (Sajjadi et al., 2016) and the semi-supervised VAE (Kingma et al., 2014). This is undesirable because, as discussed in section 2.4, the need for fine-tuning should be avoided as much as possible in semi-supervised learning algorithms because using a large held-out validation set is expensive in terms of labelled samples.

### 5.3.1 GradNorm

Chen et al. (2018) propose the GradNorm method to minimize multi-loss objective functions $L=\sum_i^T w_i\cdot L_i$. They propose to measure the impact of a loss on the learning procedure as the norm of its gradient with respect to the weights $W$ of the last layer shared among all tasks. Given the norm of the gradient for every loss $G_i=\|\nabla_W w_i\cdot L_i\|$, they compute an average gradient norm across all tasks: $\bar{G}=\mathbb{E}_i[\|\nabla_W w_i\cdot L_i\|]$. To measure how the $i$th task is training, an inverse training rate is computed as $\tilde{L}_i=\frac{L_i}{L_{i,0}}$ where $L_{i,0}$ is the (theoretical) initial loss.[3] Finally, the relative inverse training rate for task $i$ is $\gamma_i=\frac{\tilde{L}_i}{\mathbb{E}_i[\tilde{L}_i]}$, it compares how fast one task is training with respect to the others.

Chen et al. (2018) propose to balance the gradients by using $\gamma_i$: the higher the value of $\gamma_i$, the slower the task is training, so the larger its gradient should be. Therefore, the desired

[3] It could for example be $L_{\text{sup},0}=\log K$

gradient norm for each task is

$$G_i^{(\text{target})} = \bar{G} \cdot (\gamma_i)^{\alpha}, \tag{5.6}$$

where $\alpha > 0$ is a hyper-parameter setting the strength of a restoring force pulling gradients back to a common training rate. The weights are trained by adding a gradient loss to the objective function using a 1-norm distance to the target gradient norms,

$$L_{\text{grad}}(w_i; \theta) = \sum_i^T \left| G_i^{(\text{target})} - \bar{G} \times [\gamma_i]^{\alpha} \right|_1, \tag{5.7}$$

where $\bar{G} \times [\gamma_i]^{\alpha}$ is fixed to a constant. During the training procedure, $L_{\text{grad}}(w_i; \theta)$ is differentiated only with respect to $w_i$, and the computed gradients $\nabla_{w_i}$ are then used to update $w_i$ by gradient descent. Finally, the weights are standardized to sum to the number of tasks $T$, in order to avoid drifting to extreme values.

### 5.3.2 Modified GradNorm for semi-supervised learning

To scale the gradients in our semi-supervised objective loss $\mathscr{L}(\theta)$ from equation (5.5), we use a modified version of GradNorm.

First, the computation of the inverse training rate proposed in the paper ($\tilde{L}_i = \frac{L_i}{L_{i,0}}$) supposes that the losses are all positive and lower bounded by 0. Using a theoretical lower bound $L_{i,\text{LB}}$ the $i$th loss, we re-define it as $\tilde{L}_i = \frac{L_i - L_{i,\text{LB}}}{L_{i,0} - L_{i,\text{LB}}}$. Because we clip the variance at $0.01^2$ in the predictive distribution of the UNetCNP, the lower bounds for the losses in equation (5.5) are,

$$\begin{aligned}
&L_{\text{sup}} \geq L_{\text{sup,LB}} = 0, \\
&L_{\text{cons}} \geq L_{\text{cons,LB}} = -\log 2, \\
&L_{\text{sia}} \geq L_{\text{sia,LB}} = 0, \\
&L_{\text{rec}} \geq L_{\text{rec,LB}} = - \underbrace{(n+m)}_{\substack{\text{number of} \\ \text{pixels in} \\ \text{the image}}} \cdot \underbrace{\log\left(\frac{1}{2 \cdot (0.01)^2}\right)^{\frac{1}{2}}}_{\substack{\text{Gaussian distribution} \\ \text{with variance } 0.01^2 \\ \text{and squared error } 0}}.
\end{aligned} \tag{5.8}$$

Secondly, the random batch generation combined with the fact that many samples do not have labels means that some batches do not contain any labelled samples and thus that the supervised loss is not minimized at every gradient descent iterations. Therefore, if all

gradients were of the same norm, then the supervised task would train slower. To account for that, the weight of the supervised task should be rescaled with the ratio of the number of batches containing a labelled sample to the total number of batches per epoch.

Thirdly, if we use the GradNorm method where the weights are learned by minimizing the L1 norm in equation (5.7), the loss weights will take very long to reach sensible values. The convergence rate will be very slow because we can only update the weights when a batches contains a labelled sample (i.e. rarely), and because the derivative of the L1 norm in equation (5.7) doesn't scale with the distance between the current gradient norm and the target norm.

Instead of slowly learning the weights, we propose to make a hard update of the weights at every epoch. For every epoch, we compute the average gradient norm per task $\bar{G}_i$ by computing the gradient $G_i = \|\nabla_W L_i\|$ at every batch containing labelled samples. Then a target gradient norm $G_i^{(\text{target})}$ for every task is computed as in equation (5.6). The weights are then updated as

$$w_i = \frac{G_i^{(\text{target})}}{\bar{G}_i} \cdot a \tag{5.9}$$

where $a$ is a scaling factor accounting for the imbalance in labelled and unlabelled samples. The details on how to compute $a$ are given in the section "how to compute $a$?" below. Using this new method, the gradients are updated at every epoch and we do not rely on a slow learning procedure to find the right weights to make all losses train at the same rate.

The last thing to consider is the hyper-parameter $\alpha$ from equation (5.6). Recall from section 2.4 that having hyper-parameters is undesirable for semi-supervised settings, but Chen et al. (2018) showed that having any value $0 < \alpha < 3$ will improve over equal weights. The authors mostly set $\alpha = 1.5$ in the paper, so we stick to that value as well.

In section 5.4.1, we will compare the results achieved with and without using our GradNorm method. To make a fair comparison on an equal number of epochs, the objective loss trained without GradNorm uses $w_{\text{rec}} = w_{\text{cons}} = w_{\text{sia}} = 1$ and $w_{\text{sup}} = a$.

#### How to compute $a$ ?

In our training procedure, the expectations are approximated with Monte-Carlo estimations by taking the average loss over every element in the batch. For the supervised loss, the average is computed only over labelled samples. The scale factor $a$ in equation (5.9) should thus be the ratio between the total number of batches per epoch $b$ to the expected number of batches with at least one labelled sample $\mathbb{E}[b_l]$: $a = \frac{b}{\mathbb{E}[b_l]}$. Due to the linearity of the expectation, if $p$ is the probability that one batch has at least one sample, then $\mathbb{E}[b_l] = b \cdot p$. Now $p$ can be found as one minus the probability of the batch not having any labelled samples.

If $l$,$u$,$s$ are respectively the number of labelled samples, the number unlabelled samples and the number of samples per batch, then because $1-p$ follows a hyper-geometrical distribution, it follows

$$a = \frac{b}{\mathbb{E}[b_l]} = \frac{b}{b \cdot p} = \frac{1}{p} = 1 / \left( 1 - \frac{\binom{u}{s}}{\binom{u+l}{s}} \right). \tag{5.10}$$

## 5.4 Results

As in chapter 4, we compare our different models using the standard task of classifying MNIST images with 100 labelled images. We first use an ablation study in section 5.4.1 by removing the different losses from the objective function $\mathscr{L}(\theta)$ and see how our different models perform. After that, we test our best model with different number of labelled samples and we also evaluate its performance on the SVHN and CIFAR10 datasets (Krizhevsky et al., 2009; Netzer et al., 2011).

### 5.4.1 Ablation study

To test the different models proposed in section 5.2, we carry an ablation study where we evaluate the different possible combination of losses in $\mathscr{L}(\theta)$. Since the results differ depending on the set of labelled data sampled, we run every model 10 times and compare them with a box plot shown in figure 5.1. The same 10 sets of labelled images were used for all models. We use only 1/16th of the training set, precisely 3375 images, because we have to train a total of 100 models and using the full dataset would take between 4 to 6 hours per model on a V100 GPU. We also train a model with the consistency loss but without the regression loss. This model has the same architecture as the other models, but we drop all parts which are not used to make class predictions. All models were trained for 400 epochs with a batch size of 16 and a learning rate of $2 \cdot 10^{-4}$, except for the model without the reconstruction loss because it was very hard to train and to make it converge. For the latter, we eventually settled for a batch size of 256 and learning rate of $2 \cdot 10^{-4}$; the number of epochs was increased to 1600 to account for the difference in number of gradient descent iterations. GradNorm was only applied from epoch 50 onwards because the gradient norms were unstable at the beginning and stabilized after 30 to 50 epochs.

Unfortunately, we found out by looking at the learning curves that the models were not fully trained with 400 epochs, and it was the most obvious for models trained with GradNorm. Therefore, we trained (only once) four of the models for 2000 epochs (8000 for the model

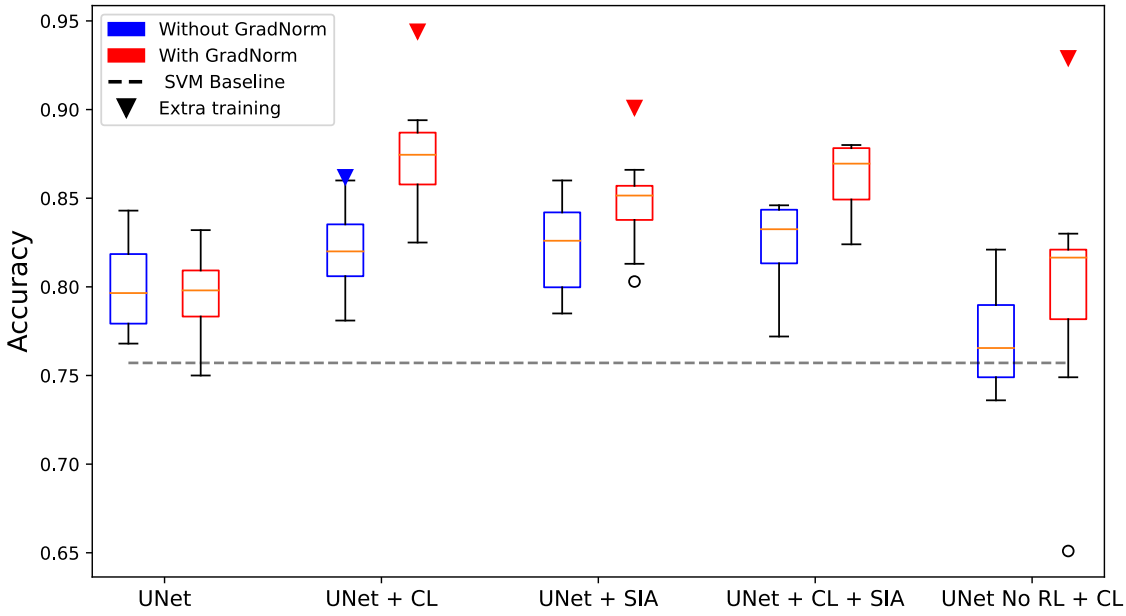



Fig. 5.1 Ablation study on the different joint losses. CL stands for consistency loss: $L_{\text{cons}}$, SIA for siamese loss: $L_{\text{sia}}$ and No RL for no reconstruction loss, i.e. models not trained on the density modelling task. The size of the training set was reduced to 1/16th of the original training set for computational reasons. All models were trained 10 times for 400 epochs and with a batch size of 16, except for the models without reconstruction loss which were trained for 1600 epochs with a batch size of 256 and the same learning rate. GradNorm is a method which scales the different losses to ensure that the norm of the gradients are equal (see section 5.3.2).

without the reconstruction loss) and we report their accuracy as triangles in figure 5.1. These results with longer training do not change the ordering in terms of performance, except for the model without density modelling, for which the accuracy reaches 92.9% which is very close to the best accuracy of 94.4% obtained when combining the reconstruction loss with the consistency loss.

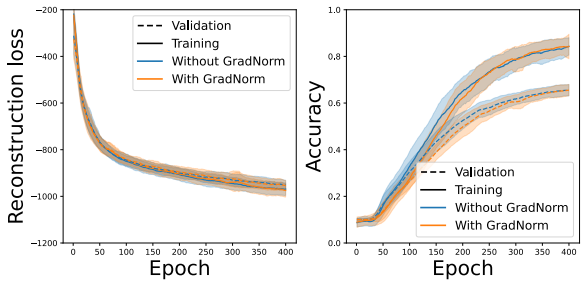
GradNorm seems to improve the performance achieved by all models, except for the joint UNetCNP without additional tasks for which it shows no significant difference. The effect of GradNorm on the training procedure is investigated in detail in section 5.4.2.

Adding extra losses to the objective shows improvements in accuracy. Particularly, the consistency loss brings better results than augmenting the training with the additional siamese loss, and including both together does not bring higher accuracies either. Although the model trained with the consistency loss and the density modelling task achieves the best accuracy, the good result achieved without the reconstruction loss seems to indicate that the density modelling task is not necessary to get good performances, and that the consistency bias might be the key component for good performance.

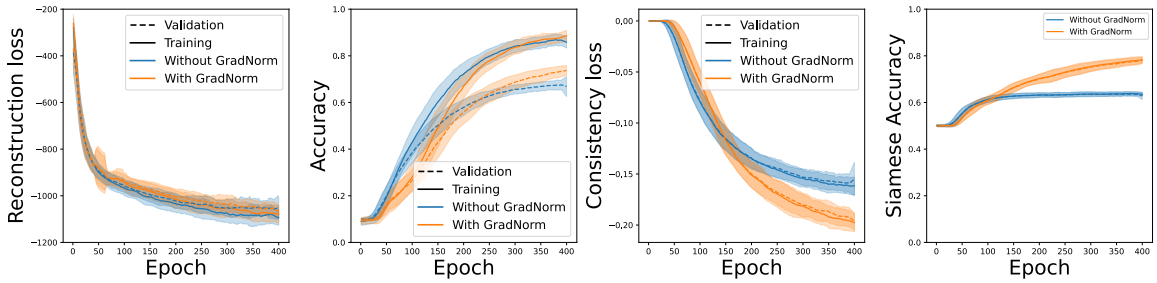
The fact that the model without reconstruction loss was hard to train is surprising because it essentially is a classification model. We believe that the gradients based on the consistency loss only (when a batch does not contain any labelled sample), were making the training unstable. In fact, we were able to make the model converge with a larger batch size effectively reducing the number of batches without labelled samples.

### 5.4.2 The effect of GradNorm

We investigate how GradNorm improves the accuracies by showing in figure 5.2 the learning curves for the different losses. The results are reported as the average and standard deviation over the 10 runs from the ablation study. The plots are further smoothed with a window 20 running average. In the case where only $L_{\text{rec}}$ and $L_{\text{sup}}$ are used, the effect of GradNorm appears insignificant. However, when the auxiliary losses $L_{\text{cons}}$ and $L_{\text{sia}}$ are included, the effect is obvious because the auxiliary losses reach lower values. GradNorm clearly helps the model by putting more emphasis on minimizing those additional losses, which in turn helps reducing the over-fitting. We can draw two important information from theses results. First, it confirms that carefully weighting gradients in multi-task losses is important and that GradNorm appears to be a sensible method for doing so. Secondly, the auxiliary consistency and siamese losses help the model learn better features because, when they are well trained, the over-fitting is lower.




(a) Objective function with the reconstruction and supervised losses only




(b) Objective function with all the losses

Fig. 5.2 Visualization of the effect of GradNorm on the learning curves for all the different losses. The curves show the the average and standard deviation of the losses at every epoch, computed over the 10 runs with different sets of labelled images. The plots are further smoothed with a running average of window size 20. GradNorm brings nearly no difference when only the reconstruction and supervised losses are used. On the other hand, when auxiliary losses are included, GradNorm helps the model by putting more emphasis on minimizing them. This in turn helps to increase the classification accuracy. The classification accuracies reported here are on images with missing pixels due to the training procedure of our models, so that explains why they are lower than in figure 5.1.

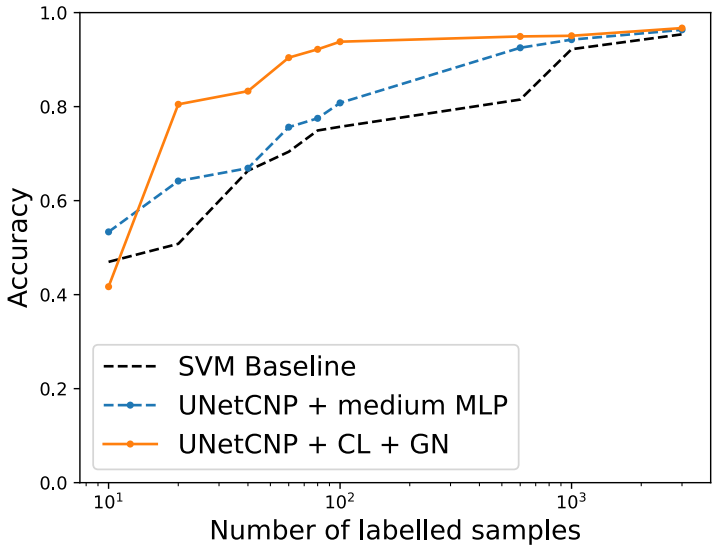



Fig. 5.3 Accuracy against number of labelled images for the UNetCNP with consistency loss and trained with GradNorm for 2000 epochs. The results are compared with the SVM baseline and with the best model from chapter 4. These results are from the full MNIST test set.

### 5.4.3 Accuracy with different number of labelled images

We now test the best model from section 5.4.1, the UNetCNP with consistency loss trained with GradNorm, by evaluating its performance with different number of labelled images. The models are trained for 2000 epochs, which is enough for the learning curves to show convergence. The accuracies are reported from the full MNIST test set. For computational reasons, we again only use 1/16th of the full unlabelled set for training. Figure 5.3 compares the results with the baseline and the best model from chapter 4.

We evaluated the model with 9 different number of labelled images and the models took around 2 hours to train, so it was not possible to train them multiple times to compute a mean and standard deviation. Most of the papers on semi-supervised algorithms report accuracies with standard deviations smaller than one when tested on the benchmarks that we use here (Kingma et al., 2014; Rasmus et al., 2015; Sajjadi et al., 2016). We therefore assume that our model behaves similarly, and that the measurements of the accuracy achieved by our models trained until convergence should have standard deviations of one or less.

The results from figure 5.3 are good with small number of labelled images. For example with 20 images, we achieve an accuracy of around 80%, against 50% for the baseline.

Table 5.1 Comparison of our model with other semi-supervised algorithms. Our UNetCNP model was trained with the full training dataset and by randomly selecting $l$ images for which to keep the labels. The results for the other algorithms are taken directly form the original papers.

| Model | MNIST ($l$=100) | SVHN ($l$=1000) | CIFAR10 ($l$=4000) |
|---|---|---|---|
| Π-model (Sajjadi et al., 2016) | 99.5 | 94.57 | 83.45 |
| CatGAN (Springenberg, 2015) | 98.09 | - | 80.42 |
| SS-VAE, M1+M2 (Kingma et al., 2014) | 96.67 | 63.98 | - |
| ADGM (Maaløe et al., 2016) | 99.04 | 77.14 | - |
| UNetCNP + CL + GN (ours) | 95.38 | 68.61 | 55.94 |

### 5.4.4 Comparison to the other semi-supervised algorithms

The MNIST digit classification task is a rather simple computer vision task, so to compare our model to the other semi-supervised methods, we consider two other standard benchmarks: the SVHN dataset (Netzer et al., 2011) with 1000 labelled images and the CIFAR10 dataset (Krizhevsky et al., 2009) with 4000 labels. Table 5.1 compares our model to some of the models presented in the taxonomy on semi-supervised algorithms in section 2.2. The results for our joint UNetCNP model presented in table 5.1 are obtained by using the whole training set.

The joint UNetCNP is clearly outperformed by the GAN or consistency-based (Π-model) methods. However, the results are not directly comparable since those two algorithms use data augmentation with affine transformation of the images, which effectively makes the training set larger and brings better performances. Using data augmentation could also be applicable in our setting but we try to keep the method as general as possible and not use any data-specific trick.

On the other hand, the performance of the jointly trained UNetCNP is in the range of the two semi-supervised deep generative models: M1 + M2 and ADGM. It even outperforms the M1+M2 semi-supervised VAE by approximately 5% on the SVHN dataset. The M1+M2 and ADGM models were not trained on the CIFAR10 dataset so it is hard to know what kind of performances to expect from a generative model. In fact, the images from the CIFAR10 dataset are much more complex than the ones from MNIST or SVHN.

## 5.5 Classification with missing inputs

The DeepSet architecture in NPs allows our models to handle sets of data as input, instead of fixed dimensional vectors. We test our model to see if it is an advantage for semi-supervised learning on images with missing pixels, by comparing it to an SVM baseline. We use the MNIST dataset with 100 labelled images and we vary the number of context pixels included in the test images from 1% up to 100%. We use the models from section 5.4.1. To train the SVM model on images with missing pixels, we repeat the 100 labelled images multiple times in the training set and mask different number of context pixels by setting them to 0. We use the same percentages as those tested in figure 5.4. For the SVM, we report the mean and standard deviation over 10 runs with different sets of labelled data. We cannot do the same for the UNetCNP models because we only have one fully trained model for both versions.

The results show that the UNetCNP keeps making very good predictions even with low number of context pixels, for example it achieves around 70% accuracy with only 10% of the pixels. The SVM is less robust to missing inputs because the accuracy falls more quickly when reducing the number of context pixels. Surprisingly, removing the reconstruction loss seems to improve the accuracies on low number of context points. While this difference could just be noise, it still suggests that if we were to transfer this model to situations where subsets of data are taken as input, the key component to keep is the DeepSet architecture, and not the density modelling task.

## 5.6 Summary and discussion

**Summary** In this chapter, we tested different models which were jointly trained on the density estimation task and the classification modelling task. Adding the consistency regularization loss and training with GradNorm was key to achieve good results. Our model showed very good performance for semi-supervised classification of images with a large number of missing pixels.

**Strengths** The UNetCNP model jointly trained with the consistency loss and GradNorm proved to extract good features on MNIST images as it achieved a test accuracy of 95.4% with 100 labelled images. The result on the SVHN dataset indicate that the quality of the classification features extracted by the model is similar to that of the other generative semi-supervised learning algorithms M1+M2 and ADGM (Kingma et al., 2014; Maaløe et al., 2016). Besides, the model was also able to generalize well with low number of labelled images as it reached for example 80% accuracy with only 20 labels, against 50%

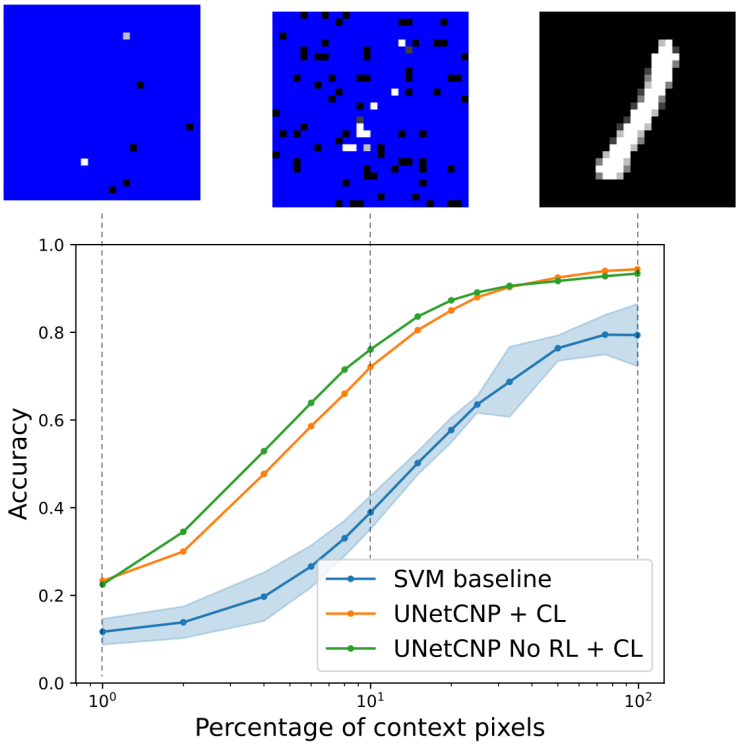



Fig. 5.4 Accuracy achieved by the UNetCNP with different number of randomly selected context points in the images classified. We show examples of an image with 1%, 10% and 100% of the context pixels. The models are discussed in section 5.4.1, they were trained with 100 labelled images using GradNorm. "CL" means consistency loss and "No RL" means that the model was trained without the reconstruction loss. We compare them with an SVM classifier for which we report a mean and standard deviation computed over 10 runs. The training set for the SVM was the 100 labelled images repeated every time with the different number of context pixels.

for the SVM baseline. An additional strength of our model is the DeepSet architecture which makes it transferable to settings where only subsets of data are taken as input. Finally, the modified version of GradNorm presented here is particularly interesting because it is applicable out-of-the-box to other semi-supervised settings with multi-loss objectives.

**Limitations** Although it would be convenient to fit the whole semi-supervised approach into a fully probabilistic NP model, the results in this chapter suggest that there are advantages to moving outside of the purely probabilistic NP approach, because the consistency loss was key to achieve competitive accuracies. Furthermore, training without the reconstruction loss but with the additional consistency loss revealed that the density modelling task was not even necessary to achieve good performances. Nevertheless, the density modelling task was shown useful to make the optimization smoother because removing the reconstruction loss required very large batch sizes for convergence. Training the model on the full MNIST dataset for 2000 epochs took one day on a V100 GPU from the Azure Virtual Machine. For a simple task like MNIST, this is very long training time, so the computational costs required by the NP might not be worth the smoother training.

# Chapter 6

# Correcting the conditional independence assumption in the UNetCNP

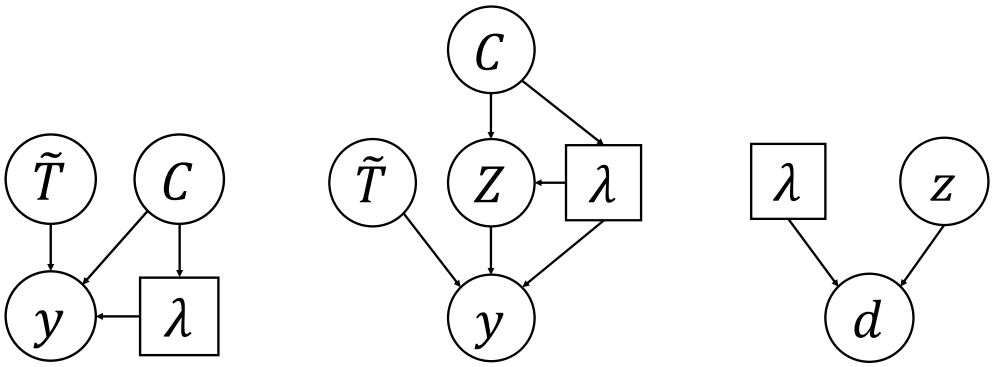
In chapters 4 and 5, we built models which relied on the improbable assumption that the predictive distribution for the density modelling task and the classification modelling task were conditionally independent given the context, so the probabilistic model was,

$$p_\theta(y_{1:n+m}, \lambda \mid C, \tilde{T}) = p_\theta(\lambda \mid C) p_\theta(y_{1:n+m} \mid C, \tilde{T}). \tag{6.1}$$

In this chapter, we remove that assumption by introducing a novel framework for NPs, where we include a class-latent variable in the model by making the predictive distribution for the density modelling task conditional on $\lambda$, such that the probabilistic model becomes

$$p_\theta(y_{1:n+m}, \lambda \mid C, \tilde{T}) = p_\theta(\lambda \mid C) p_\theta(y_{1:n+m} \mid \lambda, C, \tilde{T}). \tag{6.2}$$

In section 6.1.1, we use the class-latent variable to transform the predictive distribution into a mixture of Gaussians. In section 6.2.2, we further add a distribution over latent variables into the model. Section 6.2 presents a comparison of both models with the UNetCNP from chapter 5 on the MNIST, SVHN and CIFAR10 datasets. Finally, the results are discussed in section 6.3.




(a) LC-UNetCNP (b) LC-UNetLNP (c) M2 model from Kingma et al. (2014)

Fig. 6.1 Graphical models for the LC-UNetCNP, LC-UNetLNP and M2 model from Kingma et al. (2014). Squares indicate discrete variables and circles are for continuous variables. $Z$ is capital in the UNetLNP and UNetCNP models to indicate that it is a collection of latent variables for every input location due to the convolutional architecture of the model.

## 6.1 Methodology

### 6.1.1 LC-UNetCNP

We transform the UNetCNP by treating the class assignment as a latent variable. The graphical model is illustrated in figure 6.1a. If a sample is unlabelled, then we obtain the predictive distribution for the target points by marginalizing over $\lambda$:

$$p_\theta(y_{1:n+m} \mid C, \tilde{T}) = \sum_{\lambda=1}^{K} p_\theta(\lambda \mid C) p_\theta(y_{1:n+m} \mid C, \tilde{T}, \lambda), \tag{6.3}$$

where $p_\theta(y_{1:n+m} \mid C, \tilde{T}, \lambda)$ is a Gaussian predictive distribution and we keep the MLP classifier from the UNetCNP for the class inference $p_\theta(\lambda \mid C)$. The predictive distribution becomes a mixture of Gaussians where the weights of the components are the predicted class probabilities. We call this model the LC-UNetCNP for *latent-class* UNetCNP. To obtain the predictive distribution $p(y_{1:n+m} \mid C, \tilde{T}, \lambda)$ for every component, we add $\lambda$ as input to the network at the bottom of the UNet with a one-hot encoding. This LC-UNetCNP model is inspired by the FlowGMM model from Izmailov et al. (2020) which also ties the component weights in a GMM to the class predictions, but uses a normalizing flow for the generative part.

### 6.1.2 LC-UNetLNP

Kingma et al. (2014) present the M2 model which is a VAE model with a class-latent variable $\lambda$ in addition to the continuous latent variable $z$. The generative process for this model, illustrated in figure 6.1c, is

$$p(\lambda) = Cat(\lambda \mid \pi), \tag{6.4}$$

$$p(z) = \mathcal{N}(z \mid 0, \mathrm{I}), \tag{6.5}$$

$$p_\theta(d, \lambda, z) = p(\lambda)p(z)p_\theta(d \mid \lambda, z). \tag{6.6}$$

where $\pi$ is usually assumed to parametrize a flat prior.

We propose to further extend the LC-UNetCNP model by including a distribution over continuous latent variables as the M2 model from Kingma et al. (2014). Since convolutional NPs map context sets $C$ to functional embeddings $h$, the latent distribution is given by a stochastic process: an infinite collection of latent variables $Z$ for every input location. However, as explained in Foong et al. (2020), because we discretize the input space for the UNet in the central part of the model, sampling from the stochastic process is equivalent to sampling latent variables at the discrete input locations, so we have a finite collection $Z = \{z\}$. We call this model the LC-UNetLNP.[1] The generative model is illustrated in figure 6.1b and is given by

$$p_\theta(Z, \lambda, y_{1:n+m} | \tilde{T}, C) = p_\theta(\lambda \mid C)p_\theta(Z \mid \lambda, C)p_\theta(y_{1:n+m} \mid \lambda, Z, \tilde{T}). \tag{6.7}$$

The true posterior $p_\theta(Z, \lambda \mid C, T)$ is not tractable, so we use a variational approximation $q_\theta(Z, \lambda \mid C, T) = q_\theta(\lambda \mid C, T)q_\theta(Z \mid \lambda, C, T)$. We will use the inference network $q_\theta(\lambda \mid C, T)$ to classify the samples, so we model it with the MLP classifier at the bottom of the UNet. To sample $Z$, we use the same method as in the LNP by outputting a vector of means and standard deviations at every input location at the bottom of the UNet.

Following Kingma et al. (2014), when a sample is unlabelled, the log-marginal distribution is

$$\begin{aligned}
\log p_\theta(y_{1:n+m} \mid C, \tilde{T}) &= \log \sum_{k=1}^{K} p_\theta(\lambda \mid C) \left( \mathbb{E}_{p_\theta(Z|\lambda,C)} \left[ p_\theta(y_{1:n+m} \mid \lambda, Z, \tilde{T}) \right] \right) \\
&= \log \sum_{k=1}^{K} p_\theta(\lambda \mid C) \left( \mathbb{E}_{p_\theta(Z|\lambda,C)} \left[ p_\theta(y_{1:n+m} \mid \lambda, Z) \frac{q_\theta(\lambda \mid C, T) q_\theta(Z \mid \lambda, C, T)}{q_\theta(\lambda \mid C, T) q_\theta(Z \mid \lambda, C, T)} \right] \right),
\end{aligned} \tag{6.8}$$

[1] for latent-class Latent Neural Process

and the ELBO can be derived as,

$$\begin{aligned}\log p_\theta(y_{1:n+m} \mid C,\tilde{T}) &\geq \sum_{k=1}^{K} q_\theta(\lambda \mid C,T)\left(\mathbb{E}_{q_\theta(Z\mid\lambda,C,T)}\left[\log \frac{p_\theta(y_{1:n+m} \mid \lambda,Z,\tilde{T})p_\theta(\lambda \mid C)p_\theta(Z \mid \lambda,C)}{q_\theta(\lambda \mid C,T)q_\theta(Z \mid \lambda,C,T)}\right]\right)\\ &\geq \sum_{k=1}^{K} q_\theta(\lambda \mid C,T)\Big(\mathbb{E}_{q_\theta(Z\mid\lambda,C,T)}\left[\log p_\theta(y_{1:n+m} \mid \lambda,Z,\tilde{T})\right]\\ &\quad - KL(q_\theta(Z \mid \lambda,C,T)\|p_\theta(Z \mid \lambda,C))\Big)\\ &\quad - KL(q_\theta(\lambda \mid C,T)\|p_\theta(\lambda \mid C)).\end{aligned} \tag{6.9}$$

Using the same trick as with the LNP, we transform the non-tractable priors $p_\theta(\lambda \mid C)$ and $p_\theta(Z \mid \lambda,C)$ into a Gaussian and a multinomial variational approximations: $q_\theta(\lambda \mid C)$ and $q_\theta(Z \mid \lambda,C)$. The objective function to minimize is thus:

$$\begin{aligned}\log p_\theta(y_{1:n+m} \mid C,\tilde{T}) &\geq \sum_{k=1}^{K} q_\theta(\lambda \mid C)\Big(\mathbb{E}_{q_\theta(Z\mid\lambda,C,T)}\left[\log p_\theta(y_{1:n+m} \mid \lambda,Z,\tilde{T})\right]\\ &\quad - KL(q_\theta(Z \mid \lambda,C,T)\|q_\theta(Z \mid \lambda,C))\Big)\\ &\quad - KL(q_\theta(\lambda \mid C,T)\|q_\theta(\lambda \mid C)).\\ &\geq U(C,T;\theta).\end{aligned} \tag{6.10}$$

When a sample is labelled, we us a lower bound on the log-marginal joint distribution, with $\lambda$ fixed. Using the joint log-marginal distribution allows us to directly include the supervised loss in the objective instead of adding and ad-hoc classification loss as done in Kingma et al. (2014),

$$\begin{aligned}\log p_\theta(y_{1:n+m},\lambda \mid C,\tilde{T}) &= \log p_\theta(y_{1:n+m} \mid \lambda,C,\tilde{T}) + \log p_\theta(\lambda \mid C)\\ &\geq \mathbb{E}_{q_\theta(Z\mid\lambda,C,T)}\left[\log p_\theta(y_{1:n+m} \mid \lambda,Z)\right]\\ &\quad - KL(q_\theta(Z \mid \lambda,C,T)\|q_\theta(Z \mid \lambda,C))\\ &\quad + \log q_\theta(\lambda \mid C)\\ &\geq L(C,T,\lambda;\theta).\end{aligned} \tag{6.11}$$

Finally, the general objective function is obtained by combining $U(C,T;\theta)$ and $L(C,T,\lambda;\theta)$:

$$\mathscr{L}(\theta) = -\mathbb{E}_{C,T,\lambda\sim P}\left[(1-\mathbb{I}(\lambda))\,U(C,T;\theta) + \mathbb{I}(\lambda)\cdot L_\theta(C,T,\lambda;\theta)\right]. \tag{6.12}$$

We remove the skip-connections in the UNet, otherwise the NP could ignore the latent variables at the bottom of the UNet and only use the deterministic path at the top.

Table 6.1 Comparison of the accuracy of the LC-UNetCNP and the LC-UNetLNP, with the best UNetCNP model from chapter 5. Both model include a class-latent variables. The difference is that the LC-UNetCNP is deterministic and that the LC-UNetLNP includes a distribution over continuous latent variables. The same set of labelled images was used for the three models to make the results comparable.

| Model | MNIST ($l$=100) | SVHN ($l$=1000) | CIFAR10 ($l$=4000) |
|---|---|---|---|
| UNetCNP + CL + GN (chapter 5) | 95.38 | 68.61 | 55.94 |
| LC-UNetCNP + CL + GN | 93.6 | 62.3 | 52.1 |
| LC-UNetLNP + GN | 93.4 | - | - |

One interesting thing to note is that the minimization of $KL(q_\theta(z \mid \lambda, C, T) \| q_\theta(z \mid \lambda, C)))$ in $U(C, T; \theta)$ in equation (6.10) effectively acts as a consistency loss. It enforces context sets to have the same class predictive distribution as the full image without any missing pixels.

## 6.2 Results

Table 6.1 presents the results obtained with the two models introduced in this chapter: the LC-UNetCNP and the LC-UNetLNP. All models are trained with the same set of labelled images to make the results comparable. The LC-UNetCNP was trained with an additional consistency loss and GradNorm to weight the supervised loss as it showed to increase the performance in chapter 5. We did not add a consistency loss to the UNetLNP as it already includes one but we used GradNorm as well. We trained all models until the learning curves were flat, which was around 1000 epochs for the LC-UNetCNP and around 2000 for the LC-UNetLNP. We used batches of size 16 and a learning rate of $2 \cdot 10^{-4}$. We were not able train the supervised component of the LC-UNetLNP on the SVHN or CIFAR datasets, while the reconstruction loss did decrease, the supervised loss showed no progress after 2000 epochs (even when varying the batch size and the learning rate).

The results achieved by the LC-UNetCNP and the LC-UNetLNP are lower than that of the UNetCNP model from chapter 5. To identify if the models benefit in any way from the inclusion of a class-latent variable, we take a closer look at the results in sections 6.2.1 and 6.2.2.

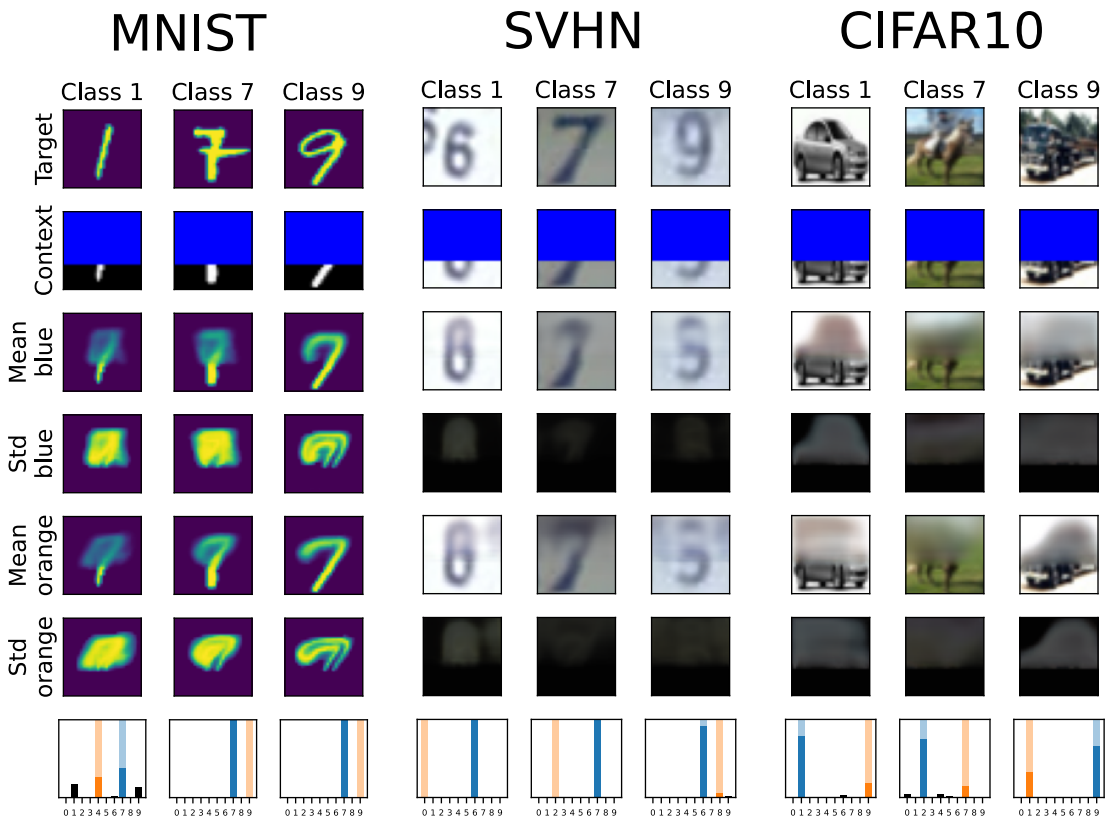



Fig. 6.2 Visualization of the output of the LC-UNetCNP model. We show the mean and standard deviation for the two components with the largest weights, indicated in blue and orange in the class probability distribution bar plot at the bottom.

### 6.2.1 Investigation of the results of the LC-UNetCNP

We desire the LC-UNetCNP to make multimodal predictions such that every mode corresponds to a specific class. In figure 6.2, we test the model in cases where the class is ambiguous by using only the bottom ten pixel rows as context. The goal is to see if the predictive distribution for every component in the mixture is consistent with the corresponding class-latent variable. The images are from the test set, so the models have never seen them. The bottom row corresponds to the class predictions. We pick the two components with the largest class prediction (indicated in blue and orange) and show their predicted mean and standard deviation.

Figure 6.2 shows that the means and standard deviations are very similar between the different components, so there is definitely no evidence that the predictions are consistent with the class assigned to the components. We show more examples in appendix B.2, with one image per class.

### 6.2.2 Investigation of the results of the UNetLNP

We visually evaluate the output of the UNetLNP by using the same method as in section 6.2.1. The results are shown in figure 6.3. Two samples for every context set are obtained by fixing the class-latent variable to one of the two classes with the largest predicted probability (blue or orange) and then sampling a continuous latent variable to output a mean and standard deviation. The SVHN and CIFAR10 models are the best models we were able to obtain after 2000 epochs; the model learned to improve on the density modelling task, but they showed no progress on the classification task. More examples are available in appendix B.3.

Even though the model achieves a classification accuracy of 93.4% on MNIST, the predictive distributions are not consistent with the class-latent variables. From the results presented here, we have no evidence that transitioning from a deterministic model to a model with distributions over latent variables improves the quality of the classification features extracted. Since we were unable to make the model converge on the SVHN or CIFAR10 datasets, the results suggest that the distribution over latent variables is more a of a hindrance to the classifier.

## 6.3 Summary and discussion

**Summary** In this chapter, we removed the conditional independence assumption that we used in the probabilistic model in chapter 4 and 5. To the best of our knowledge, we are the first to include a class-latent variable in NPs to make the predictive distribution class-conditional. We presented two models. The first is a LC-UNetCNP which outputs a mixture of Gaussians for the density modelling task, where every component is weighted by the predicted class distributions. The second is the LC-UNetLNP which includes both a discrete class-latent variable and a continuous latent variable.

**Strengths** Both models presented here do not outperform the UNetCNP from chapter 5. However, they allow semi-supervised VAE type of approaches to be leveraged in combination with NPs. Further work could build on top of these models, for example by leveraging recent advances in VAE models for images. A good starting point could be the BIVA model from Maaløe et al. (2019) because it is a skip-connected generative model which would allow us to restore the skip connections that we had to remove in the LC-UNetLNP. This is likely to improve the performance of the model because the results on the UNetCNP suggested that the skip connections were key to extract high quality classification features.

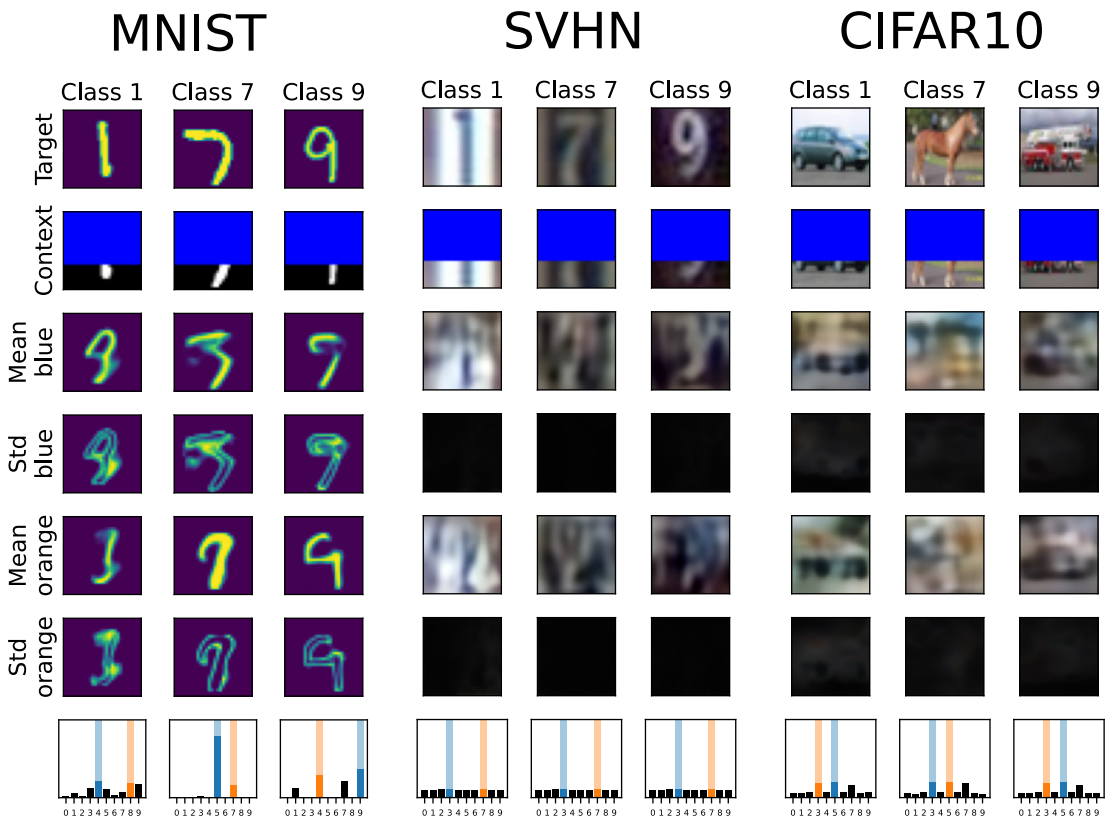



Fig. 6.3 Visualization of the output of the LC-UNetLNP. We show the predictive distribution when fixing the class-latent variable to one of the two classes with the largest predicted probability, indicated in blue and orange. We were not able to make the classifier of the models trained on SVHN or CIFAR10 converge, so we show the best models that we were able to obtain.

**Limitations** The first limitation of our model is their lower performances compared to the UNetCNP from chapter 5. The second is their computational cost. In fact, they require one forward pass for every class, so the models require up to $K$ times more computations with $K$ classes. With 25% of the training data, we had to train both of our models for 2 days on a V100 GPU, instead of 1 day for the UNetCNP with 100% of the data and a similar amount of training epochs. This could be improved, for example by combining our model with the truncation algorithm from Pal et al. (2005).

# Chapter 7

# Conclusion and future directions

The purpose of this thesis was to see if NPs could be useful for semi-supervised learning. To investigate that, we built Neural Processes modelling the context-conditional joint distribution of the class and target values. To perform a thorough analysis, we evaluated a number of models with different architectures, training procedures, and assumptions for the probabilistic model.

Sharing weights between the density modelling task and the classification task increased the classification performance over the fully-supervised baselines only when the architecture of the model encouraged the separation of high-level and low-level information. The problem is that an optimal model for the density modelling task will extract all the details from the context set to make fine-grained predictions, but those details are not relevant for classification. With the UNetCNP architecture, the model was able to make the low level information pass through the top connections in the UNet and extract higher level features at the bottom.

The UNet architecture was found to improve performance, however, the results were still low compare to state-of-the-art semi-supervised learning algorithms. We then experimented with a consistency regularization loss and found it was key to improve performance. Furthermore, training without the reconstruction loss but with this additional consistency loss revealed that the density modelling task was not necessary to achieve good performance. However, it was shown to make the training smoother; without it, training required a batch of size at least 256 to converge.

The first models relied on an improbable assumption of conditional independence between the predictions of the model for the density modelling task and the classification modelling task. We proposed two other models which corrected that assumption by making the predictive distribution dependent on a class latent variable. Although it was a more rigorous definition for our probabilistic model, we did not achieve better accuracies.

Finally, we made an attempt to include a distribution over latent variables in the model, to allow consistent sampling from the predictive distribution in order to help the model learn a consistent relation between the class latent variable and the predictive distribution. But this method also showed no improvements. We were even unable to make the model converge on the SVHN and CIFAR10 datasets. The randomness in the latent distribution seemed to be more of a hindrance to the model's classification abilities.

Even though it would be convenient to encapsulate the whole semi-supervised model into a clean probabilistic NP model, the results from this thesis suggest that it is not the only part of the solution. To achieve results competitive with recent algorithms from the semi-supervised literature, we must include additional components in the model by coupling it with consistency regularization methods.

To summarize what the experiments show, we list a number of points which we found to be important when using NPs for semi-supervised learning.

- It was particularly important to have some kind of hierarchical structure in the NP which isolates high-level features from low-level features.
- Adding a consistency loss to the model helps a great deal and it is rather easy to add as it only depends on the classification component.
- The unsupervised density modelling task helped stabilize the training and made it easier for it to converge.
- We recommend using our modified version of GradNorm when combining the different losses because it works out-of-the-box and it was key to get the full benefit out of the consistency loss.

## 7.1 Future directions

As suggested in chapter 6, further work could extend the class-latent NPs by leveraging advances in VAE for images. We had to remove the skip-connections from the LC-UNetLNP to prevent it from skipping the stochastic layer even though the results from section 4.4.6 suggest that those skip connections are important for extracting good classification features. To improve on that, we could draw inspiration from the BIVA model from Maaløe et al. (2016) to include stochastic skip-connections in the model.

A particular strength of the models presented in this thesis is that they are transferable to irregularly sampled data. This characteristic originates from the (Conv)DeepSet architecture, which does not depend on the probabilistic modelling concepts inherent to the

NP model. Therefore, further work on semi-supervised learning could consider applying the (Conv)DeepSet architecture to settings where it is particularly helpful, for example for classifying sporadically sampled time series. We could combine it with state-of-the-art semi-supervised algorithms to extend them to situations where sets are taken as input.

# References


Bachman, P., Alsharif, O., and Precup, D. (2014). Learning with pseudo-ensembles. *Advances in neural information processing systems*, 27:3365–3373.

Balasubramanian, M., Schwartz, E. L., Tenenbaum, J. B., de Silva, V., and Langford, J. C. (2002). The isomap algorithm and topological stability. *Science*, 295(5552):7–7.

Bijral, A. S., Ratliff, N., and Srebro, N. (2012). Semi-supervised learning with density based distances. *arXiv preprint arXiv:1202.3702*.

Calder, J., Cook, B., Thorpe, M., and Slepcev, D. (2020). Poisson learning: Graph based semi-supervised learning at very low label rates. In *International Conference on Machine Learning*, pages 1306–1316. PMLR.

Chapelle, O., Scholkopf, B., and Zien, A. (2009). Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542.

Chen, Z., Badrinarayanan, V., Lee, C.-Y., and Rabinovich, A. (2018). Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*, pages 794–803. PMLR.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258.

De Bie, T. and Cristianini, N. (2003). Convex methods for transduction. In *Advances in neural information processing systems*, pages 73–80.

Demiriz, A. and Bennett, K. P. (2001). Optimization approaches to semi-supervised learning. In *Complementarity: Applications, Algorithms and Extensions*, pages 121–141. Springer.

Dubois, Y. (2019). Extending neural processes to the wild. Master's thesis.

Dubois, Y., Gordon, J., and Foong, A. Y. (2020). Neural process family. http://yanndubs.github.io/Neural-Process-Family/.

Foong, A. Y., Bruinsma, W. P., Gordon, J., Dubois, Y., Requeima, J., and Turner, R. E. (2020). Meta-learning stationary stochastic process prediction with convolutional neural processes. *arXiv preprint arXiv:2007.01332*.

Gammerman, A., Vovk, V., and Vapnik, V. (1998). Learning by transduction. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, page 148–155.

Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. (2018a). Conditional neural processes. In *International Conference on Machine Learning*, pages 1704–1713. PMLR.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. (2018b). Neural processes. *arXiv preprint arXiv:1807.01622*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27.

Gordon, J., Bruinsma, W. P., Foong, A. Y., Requeima, J., Dubois, Y., and Turner, R. E. (2019). Convolutional conditional neural processes. *arXiv preprint arXiv:1910.13556*.

Izmailov, P., Kirichenko, P., Finzi, M., and Wilson, A. G. (2020). Semi-supervised learning with normalizing flows. In *International Conference on Machine Learning*, pages 4615–4630. PMLR.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., Mohamed, S., Rezende, D. J., and Welling, M. (2014). Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kobyzev, I., Prince, S., and Brubaker, M. (2020). Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Koch, G., Zemel, R., Salakhutdinov, R., et al. (2015). Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105.

Laine, S. and Aila, T. (2016). Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*.

Lawrence, N. and Jordan, M. (2004). Semi-supervised learning via gaussian processes. *Advances in neural information processing systems*, 17:753–760.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, Q., Wu, X.-M., and Guan, Z. (2018). Generalized label propagation methods for semi-supervised learning.

Maaløe, L., Fraccaro, M., Liévin, V., and Winther, O. (2019). Biva: A very deep hierarchy of latent variables for generative modeling. *arXiv preprint arXiv:1902.02102*.

Maaløe, L., Sønderby, C. K., Sønderby, S. K., and Winther, O. (2016). Auxiliary deep generative models. In *International conference on machine learning*, pages 1445–1453. PMLR.

Miyato, T., Maeda, S.-i., Koyama, M., and Ishii, S. (2018). Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993.

Nalisnick, E., Matsukawa, A., Teh, Y. W., Gorur, D., and Lakshminarayanan, B. (2019). Hybrid models with deep and invertible features. In *International Conference on Machine Learning*, pages 4723–4732. PMLR.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning.

Noroozi, M. and Favaro, P. (2016). Unsupervised learning of visual representations by solving jigsaw puzzles. In *European conference on computer vision*, pages 69–84. Springer.

Øksendal, B. (2003). Stochastic differential equations. In *Stochastic differential equations*, pages 65–84. Springer.

Oliver, A., Odena, A., Raffel, C., Cubuk, E. D., and Goodfellow, I. J. (2018). Realistic evaluation of deep semi-supervised learning algorithms. *arXiv preprint arXiv:1804.09170*.

Pal, C., Sutton, C., and McCallum, A. (2005). Fast inference and learning with sparse belief propagation. *Advances in Neural Information Processing Systems (NIPS)*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037.

Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., and Efros, A. A. (2016). Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544.

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Rasmus, A., Valpola, H., Honkala, M., Berglund, M., and Raiko, T. (2015). Semi-supervised learning with ladder networks. *arXiv preprint arXiv:1507.02672*.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.

Sajjadi, M., Javanmardi, M., and Tasdizen, T. (2016). Regularization with stochastic transformations and perturbations for deep semi-supervised learning. *Advances in neural information processing systems*, 29:1163–1171.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. (2008). Collective classification in network data. *AI magazine*, 29(3):93–93.

Springenberg, J. T. (2015). Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*.

Tarvainen, A. and Valpola, H. (2017). Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. *arXiv preprint arXiv:1703.01780*.

Tudisco, F., Benson, A. R., and Prokopchik, K. (2021). Nonlinear higher-order label spreading. In *Proceedings of the Web Conference 2021*, pages 2402–2413.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24.

Yang, X., Song, Z., King, I., and Xu, Z. (2021). A survey on deep semi-supervised learning. *arXiv preprint arXiv:2103.00550*.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R., and Smola, A. (2017). Deep sets. *arXiv preprint arXiv:1703.06114*.

Zhang, L. and Qi, G.-J. (2020). Wcp: Worst-case perturbations for semi-supervised deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3912–3921.

Zhang, R., Isola, P., and Efros, A. A. (2016). Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer.

Zhou, D., Bousquet, O., Lal, T. N., Weston, J., and Schölkopf, B. (2004). Learning with local and global consistency. In *Advances in neural information processing systems*, pages 321–328.

Zhu, X. and Ghahramanih, Z. (2002). Learning from labeled and unlabeled data with label propagation.

# Appendix A

# Neural network models details

## A.1 Architectures

### A.1.1 LeNet

Tables A.1-A.3 summarize the architecture of the different LeNet networks with small, medium and large sizes.

## A.2 Training details

### A.2.1 LeNet and WideResNet

Both the LeNet and the WideResNet networks are trained with a batch size of 64, or less if only fewer labelled images are available. With 100 labelled images or less, the learning rate

Table A.1 Details of the small LeNet networks used as a fully supervised baseline.

| | Layer description | Output tensor Dim. |
|---|---|---|
| 1 | $5 \times 5$ conv, 4 features, ReLU | H $\times$ W $\times$ 4 |
| 2 | $2 \times 2$ average pool | 1/2 H $\times$ 1/2 W $\times$ 4 |
| 3 | $5 \times 5$ conv, 4 features, ReLU | 1/2 H $\times$ 1/2 W $\times$ 4 |
| 4 | $2 \times 2$ average pool | 1/4 H $\times$ 1/4 W $\times$ 4 |
| 6 | $5 \times 5$ conv, 10 features, ReLU | 1/4 H $\times$ 1/4 W $\times$ 10 |
| 7 | 1/4 H $\times$ 1/4 W average pool | 10 |
| 8 | dense, 10 features, ReLu, dropout | 10 |
| 9 | dense, 10 features, ReLu, dropout | 10 |

Table A.2 Details of the medium LeNet networks used as a fully supervised baseline.

| | Layer description | Output tensor Dim. |
|---|---|---|
| 1 | $5 \times 5$ conv, 4 features, ReLU | H $\times$ W $\times$ 4 |
| 2 | $2 \times 2$ average pool | 1/2 H $\times$ 1/2 W $\times$ 4 |
| 3 | $5 \times 5$ conv, 8 features, ReLU | 1/2 H $\times$ 1/2 W $\times$ 8 |
| 4 | $2 \times 2$ average pool | 1/4 H $\times$ 1/4 W $\times$ 8 |
| 6 | $5 \times 5$ conv, 64 features, ReLU | 1/4 H $\times$ 1/4 W $\times$ 64 |
| 7 | 1/4 H $\times$ 1/4 W average pool | 64 |
| 8 | dense, 10 features, ReLu, dropout | 64 |
| 9 | dense, 10 features, ReLu, dropout | 10 |

Table A.3 Details of the large LeNet networks used as a fully supervised baseline.

| | Layer description | Output tensor Dim. |
|---|---|---|
| 1 | $5 \times 5$ conv, 4 features, ReLU | H $\times$ W $\times$ 4 |
| 2 | $2 \times 2$ average pool | 1/2 H $\times$ 1/2 W $\times$ 4 |
| 3 | $5 \times 5$ conv, 24 features, ReLU | 1/2 H $\times$ 1/2 W $\times$ 24 |
| 4 | $2 \times 2$ average pool | 1/4 H $\times$ 1/4 W $\times$ 24 |
| 6 | $5 \times 5$ conv, 128 features, ReLU | 1/4 H $\times$ 1/4 W $\times$ 128 |
| 7 | 1/4 H $\times$ 1/4 W average pool | 128 |
| 8 | dense, 10 features, ReLu, dropout | 128 |
| 9 | dense, 10 features, ReLu, dropout | 10 |

is $5 \cdot 10^{-3}$ and with more than 100 labelled images, it is $1 \cdot 10^{-3}$. This difference compensates for the different number of stochastic gradient updates.

### A.2.2 Fine-tuned MLP classifiers on features from the Neural Processes

Due to the low number of labelled images, some manual experimentation with the batch size and learning rate had to be done for the model training to converge. All models were trained with a batch size of 64 and a learning rate of $10^{-3}$, except for the small architectures with less than (and including) 100 labelled images, where a learning rate of $5 \cdot 10^{-3}$ was used. With 100 labelled images or less, the models were trained for a maximum of 400 epochs, using the validation set for early stopping. With 600, 1000 and 3000 images, the number of epochs was limited to 200 as it showed to be enough.

# Appendix B

# Additional results

## B.1 Neural Process training

Figures B.1 and B.2 illustrate the inpainting performance of the fully trained NPs, on the uniform and semantic task.

## B.2 LC-UNetCNP: more visualizations

Figures B.3 and B.4 show more visualizations of the output of the LC-UNetCNP from chapter 6.

## B.3 LC-UNetLNP: more visualizations

Figures B.5 and B.6 show more visualizations of the output of the LC-UNetLNP from chapter 6.

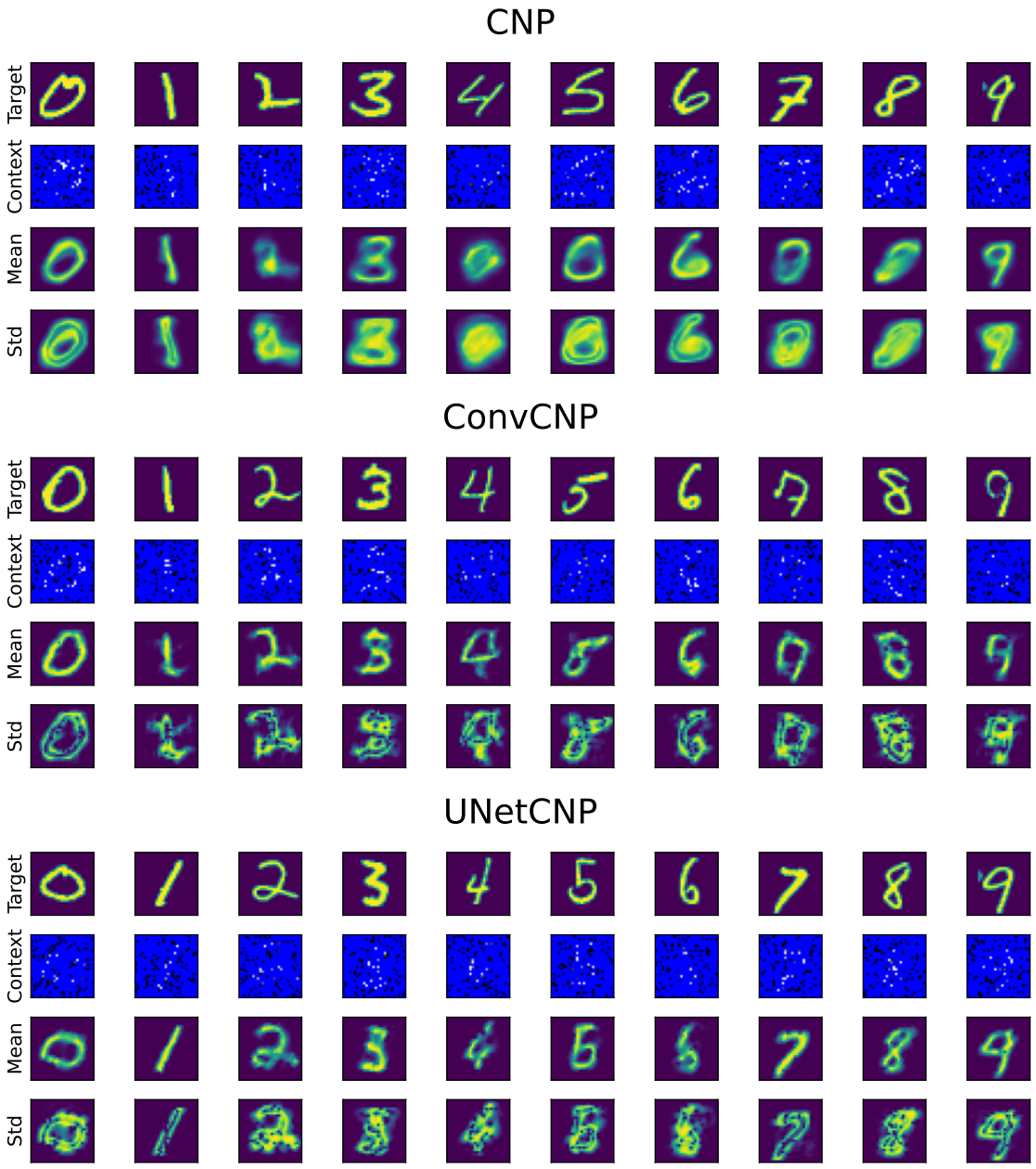



Fig. B.1 Image inpainting with the models trained on the uniform task.

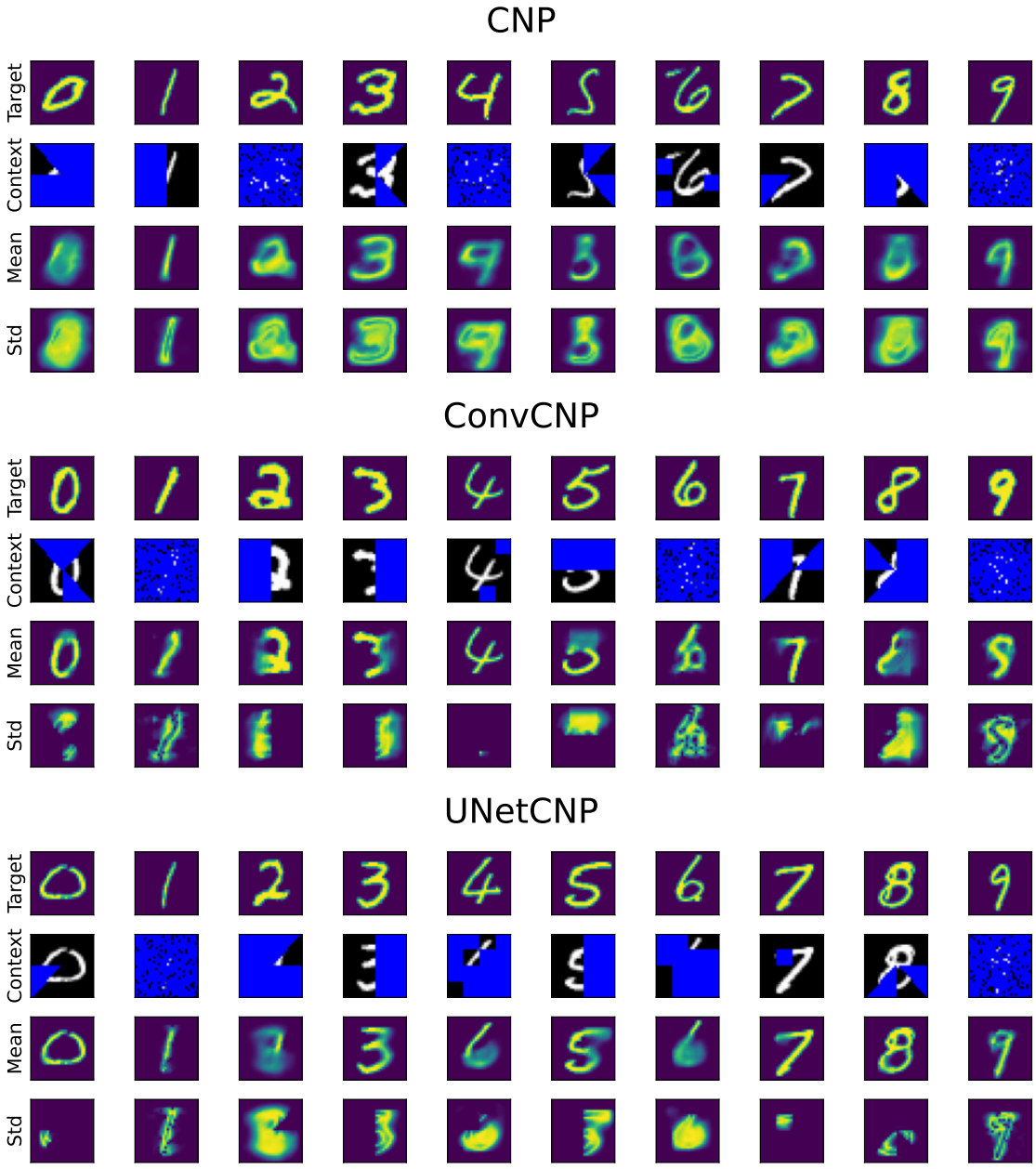



Fig. B.2 Image inpainting with the models trained on the semantic task.

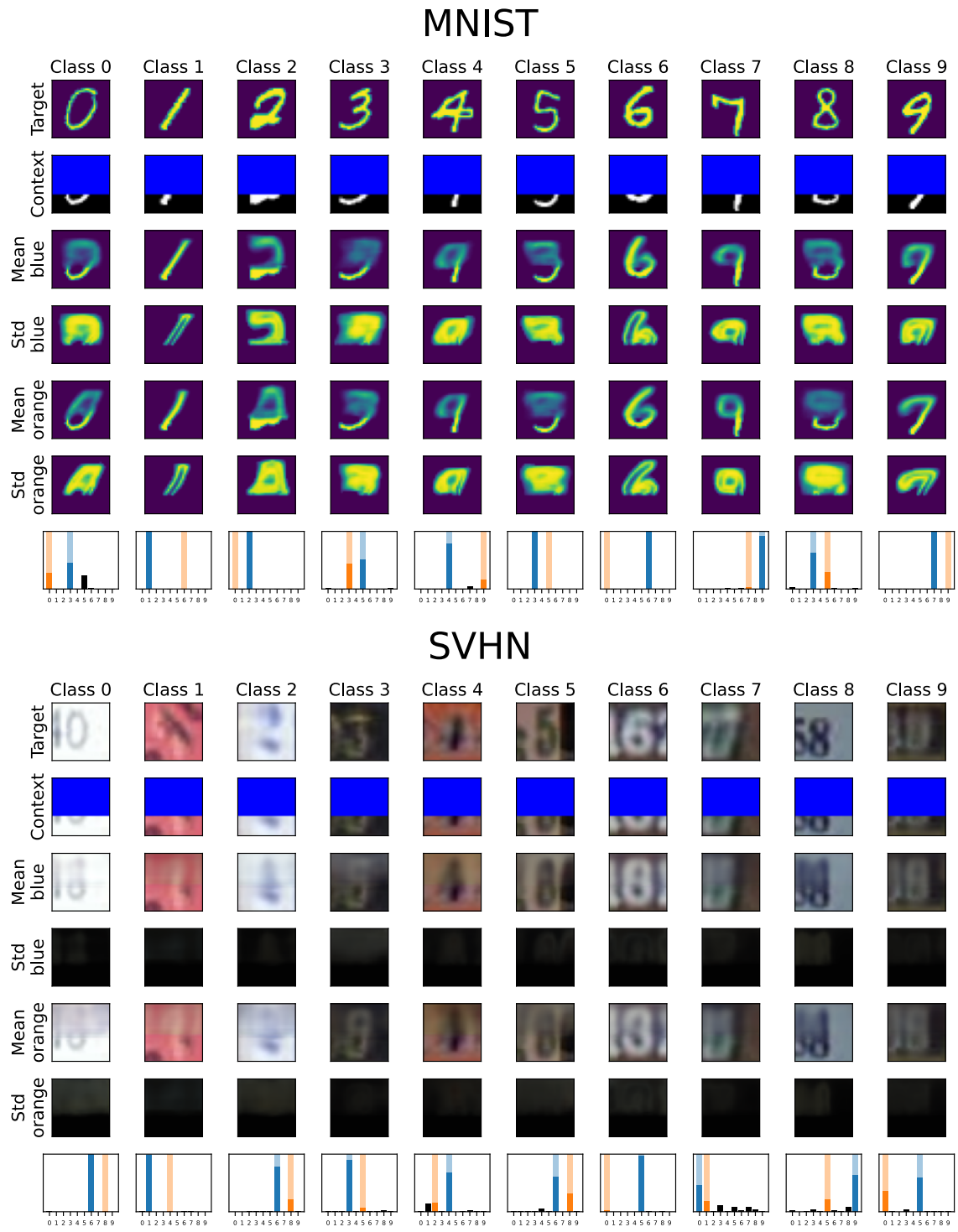



Fig. B.3 More MNIST and SVHN visualizations of the output of the LC-UNetCNP.

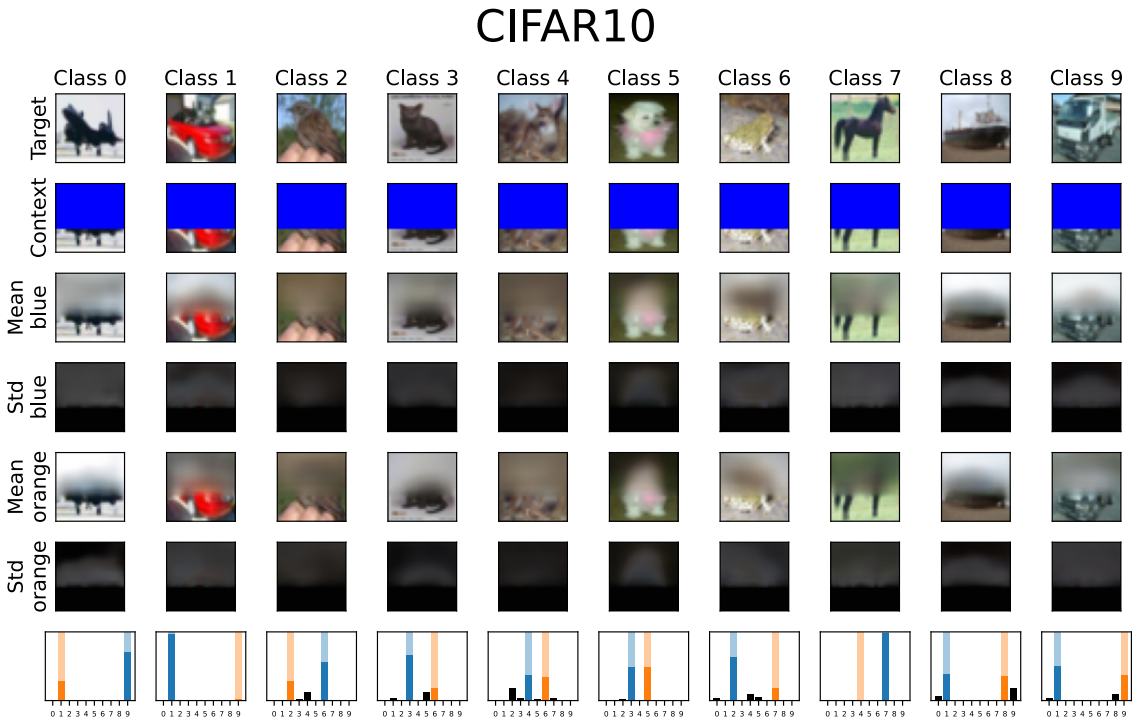



Fig. B.4 More CIFAR10 visualizations of the output of the LC-UNetCNP.

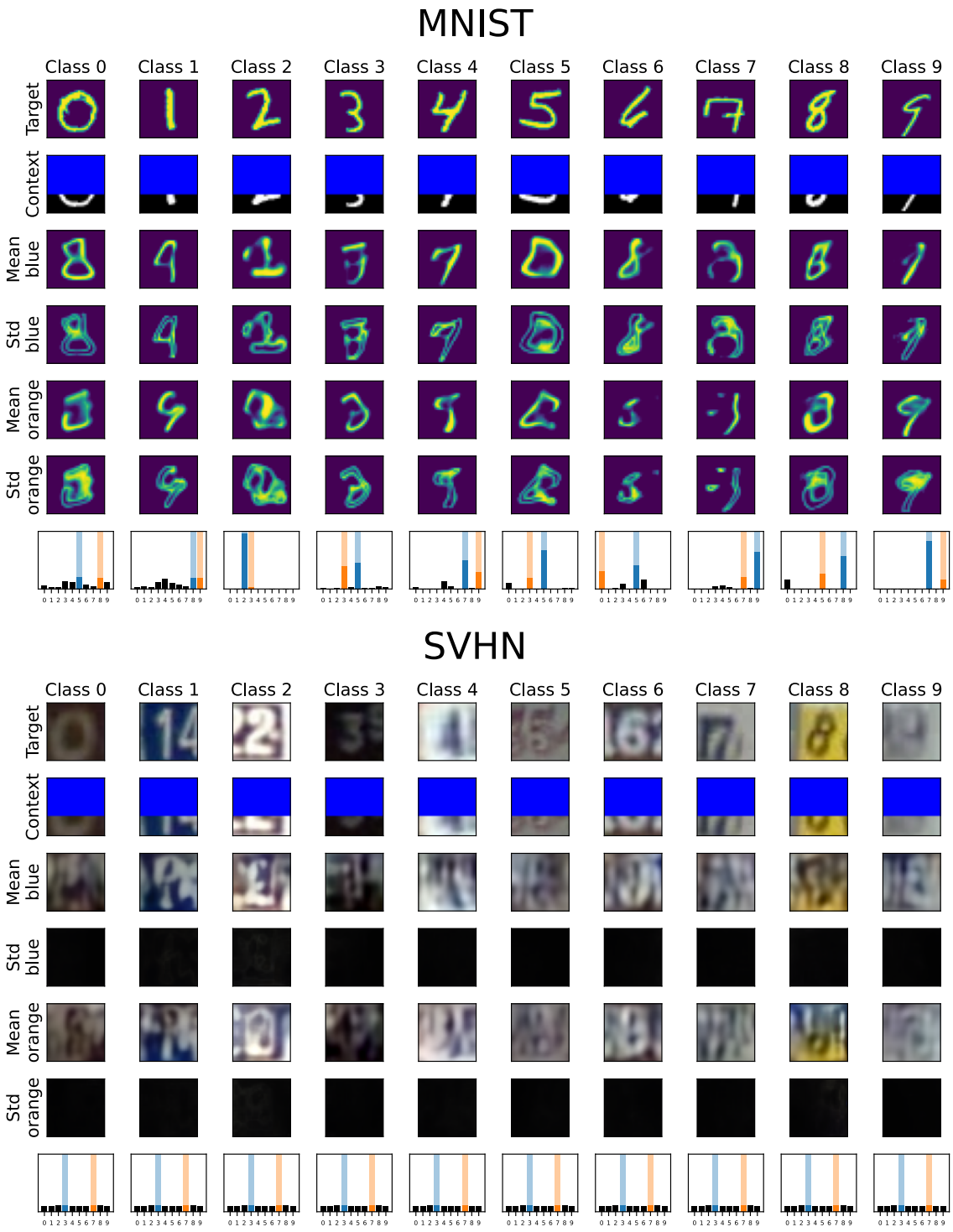



Fig. B.5 More MNIST and SVHN visualizations of the output of the LC-UNetLNP.

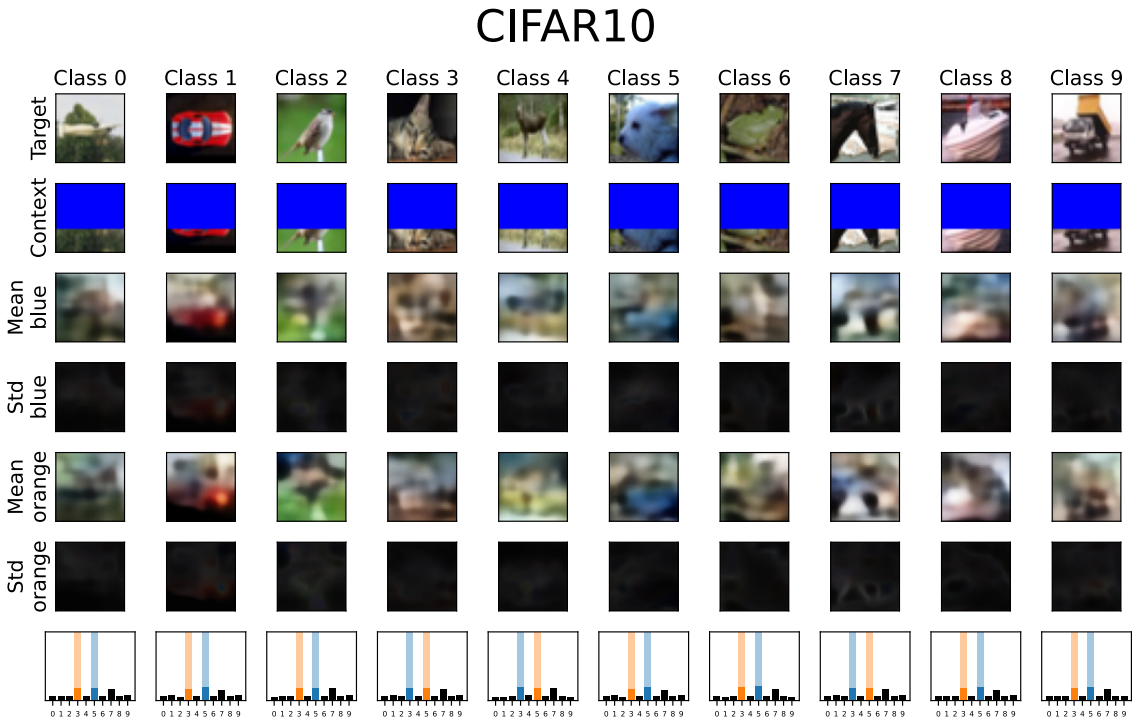



Fig. B.6 More CIFAR10 visualizations of the output of the LC-UNetLNP.