0.1 Introduction

Convolutional Neural Networks (CNNs) are a class of neural networks that are renowned for be translation equivariant, they have been widely used in image processing tasks and have shown to be able to learn spatial patterns in the data via convolutional filters. CNNs are very powerful and have been the state of the art in many image processing tasks making them a desirable backbone for a Neural Process. Hence, the motivation for Convolutional Neural Processes (ConvCNPs) [Gordon et al. 2020]. We will breifly discuss the ConvCNP model and its architecture however for a more detailed explanation, we refer the reader to the original paper.
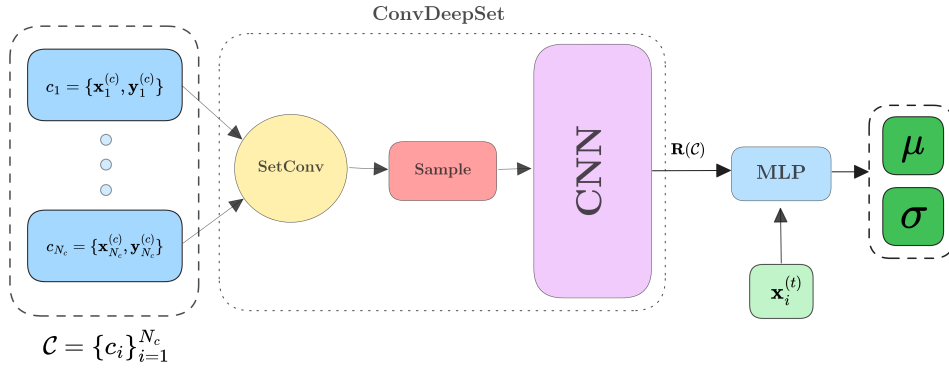
0.2 Model

Figure 0.2.1: ConvCNP Architecture. All the data points are converted into a continuous functional embedding using the SetConv, then discretely sampled to be fed into a CNN. The CNN processes the data and then the output is converted back to a continuous function using the SetConv operation.

ConvCNPs [Gordon et al. 2020] Figure 0.2.1 utilize CNNs, more specifically the UNet [Ronneberger, Fischer, and Brox 2015] architecture. However, a few hurdles are preventing us from directly plugging data into a CNN.

CNNs operate on **on-grid data**, meaning that the data is on a regular grid, for example, an image. However, the data we are working with is sometimes **off-grid data**, for example time series dataset with irregular timestamps. Furthermore, to 'bake-in' the TE property, we need to be able to shift the data in the input space and the output space in the same way. This is not trivial when using standard vector representations of the data. [Gordon et al. 2020] propose a solution to this problem by using **functional embeddings** to model the data. Functional embeddings are a way to represent data as a function that has a trivial translation equivariance property.

These functional embeddings are created using the **SetConv** operation. The SetConv operation takes a set of input-output pairs and outputs a continuous functional representation of the data. The SetConv operation is defined as follows:

$$\text{SetConv}(\{\boldsymbol{x_i}, \boldsymbol{y_i}\}_{i=1}^{N})(x) = \sum_{i=1}^{N} [1, \boldsymbol{y_i}]^T \mathcal{K}(\boldsymbol{x} - \boldsymbol{x_i}) \tag{0.2.1}$$

Where $\mathcal{K}$ is a kernel function that measures the distance between the query x and the data point $\boldsymbol{x_i}$.

This operation has some key properties:

- We append 1 to output $\boldsymbol{y_i}$ when computing the SetConv, this acts as a flag to the model so that it knows which data points are observed and which are not. Say we have a data point of $\boldsymbol{y_i} = 0$, then if we did not append the 1, the model would not be able to distinguish between an observed data point and an unobserved data point (as both would be 0).

- The 'weight' of the Kernel depends only on the relative distance between points on the input space which means that the model is translation equivariant.

- The summation over the data points naturally introduces **Permutation Invariance** to the model, meaning that the order of the data points does not matter.

Now that we have a function representation of the context set, we need to sample it/discretize it at **evenly** spaced points. Thus converting the data into a **on-grid** format. Now it can be fed into a CNN. Now that we have a CNN that operates on the data, we need to convert it back to a continuous function. This is done again by using the **SetConv** operation. The final output of the encoder is a continuous function that represents the context set which can be queried at any point in the input space using the target set.

$$R(\boldsymbol{x_t}) = \text{SetConv}(\text{CNN}(\{\text{SetConv}(\mathcal{C})(\boldsymbol{x_d})\}_{d=1}^{D}))(\boldsymbol{x_t}) \tag{0.2.2}$$

The decoder is a simple MLP that takes the output of the encoder and the target set as input and outputs the mean and variance of the predictive distribution $\boldsymbol{\mu}(\boldsymbol{x_t}), \boldsymbol{\sigma}(\boldsymbol{x_t}) = \text{MLP}(R(\boldsymbol{x_t}))$

Bibliography

Gordon, Jonathan et al. (2020). *Convolutional Conditional Neural Processes.* arXiv: 1910.13556 [stat.ML].

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation.* arXiv: 1505.04597 [cs.CV].