GP-CONVCNP: Better Generalization for Convolutional Conditional Neural Processes on Time Series Data — Supplementary Material

Jens Petersen[1] **Gregor Köhler**[1] **David Zimmerer**[1] **Fabian Isensee**[2] **Paul F. Jäger**[3] **Klaus H. Maier-Hein**[1]

[1]Division of Medical Image Computing, German Cancer Research Center, Heidelberg, Germany
[2]HIP Applied Computer Vision Lab, Division of Medical Image Computing, German Cancer Research Center
[3]Interactive Machine Learning Group, German Cancer Research Center

A METHOD DESCRIPTIONS

Fig. A.1 shows schematic representations of the different methods used in this work, and a description is given in the figure caption. The MLPs in both NP and ANP have 6 hidden layers with 128 channels each, and the input and output sizes are adjusted to match the dimensions of data and latent representations. The latent representation in both models has 128 dimensions, so that the encoders for the NP and the NP path in ANP have 256 output channels to represent both the mean and the standard deviation of a Gaussian distribution (in practice, we predict the log-variance, not the standard deviation). The attention mechanism in ANP also uses 128 as the embedding dimension. These configurations follow Le et al. [2018], who evaluated several different configurations for NP and ANP.

CONVCNP and GP-CONVCNP both use a Gaussian kernel with a learnable length scale l to map the input to a continuous representation, given by

$$k(x, x') = \exp\left(-\frac{|x - x'|^2}{2l^2}\right) \tag{1}$$

The result is discretized onto a grid, which we obtain by taking the minimum and maximum of the target inputs as the value range, padded by 0.1 units. The grid is constructed over this range with a resolution of 20 points per unit. The discretized representations are projected to 8 channels before a CNN is applied. The CNN is a 12-layer residual network with ReLU activations. The number of channels in the convolutional layers doubles every second layer for the first 6 layers and is then decreased symmetrically, leading to 8 output channels. Residual connections are implemented via concatenation. Predictions are obtained by convolving the CNN output with a target input, followed by a final projection.

B OPTIMIZATION

Recall that our optimization objective is

$$\max_\theta \sum_{f \in \mathcal{F}} \log p_\theta(\mathbf{y_t}|\mathbf{x_t}, \mathbf{x_c}, \mathbf{y_c}) \tag{2}$$

which we can rewrite as

$$\max_\theta \sum_{f \in \mathcal{F}} \log p_\theta(\mathbf{y_t}|\mathbf{x_t}, Z) \tag{3}$$

where Z is given by the different E that *encode* the context introduced in Section 2. For CONVCNP this is deterministic, so we can maximize Eq. (2) directly. For the other methods we can again rewrite the summands as

$$\log p(\mathbf{y_t}|\mathbf{x_t}, \mathbf{x_c}, \mathbf{y_c}) = \log \mathop{\mathbb{E}}_{z \sim p(z|\mathbf{x_c}, \mathbf{y_c})} p(\mathbf{y_t}|\mathbf{x_t}, z) \tag{4}$$

where we now distinguish z as an expression of Z. In GP-CONVCNP, $p(z|\mathbf{x_c}, \mathbf{y_c})$ is given by the GP posterior, so for training we would need to integrate over the posterior. In practice, we just draw a single sample, which is common practice in stochastic mini-batch training. Approximating the expectation with this sample, we can also directly maximize the log-likelihood.

In contrast to the above, $p(z|\mathbf{x_c}, \mathbf{y_c})$ is an unknown or intractable mapping in NP and ANP, so we employ variational inference, i.e. we approximate $p(z|\mathbf{x_c}, \mathbf{y_c})$ with a member of some family Q that we can find by optimization. The log-likelihood then becomes

Supplement for the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI 2021).

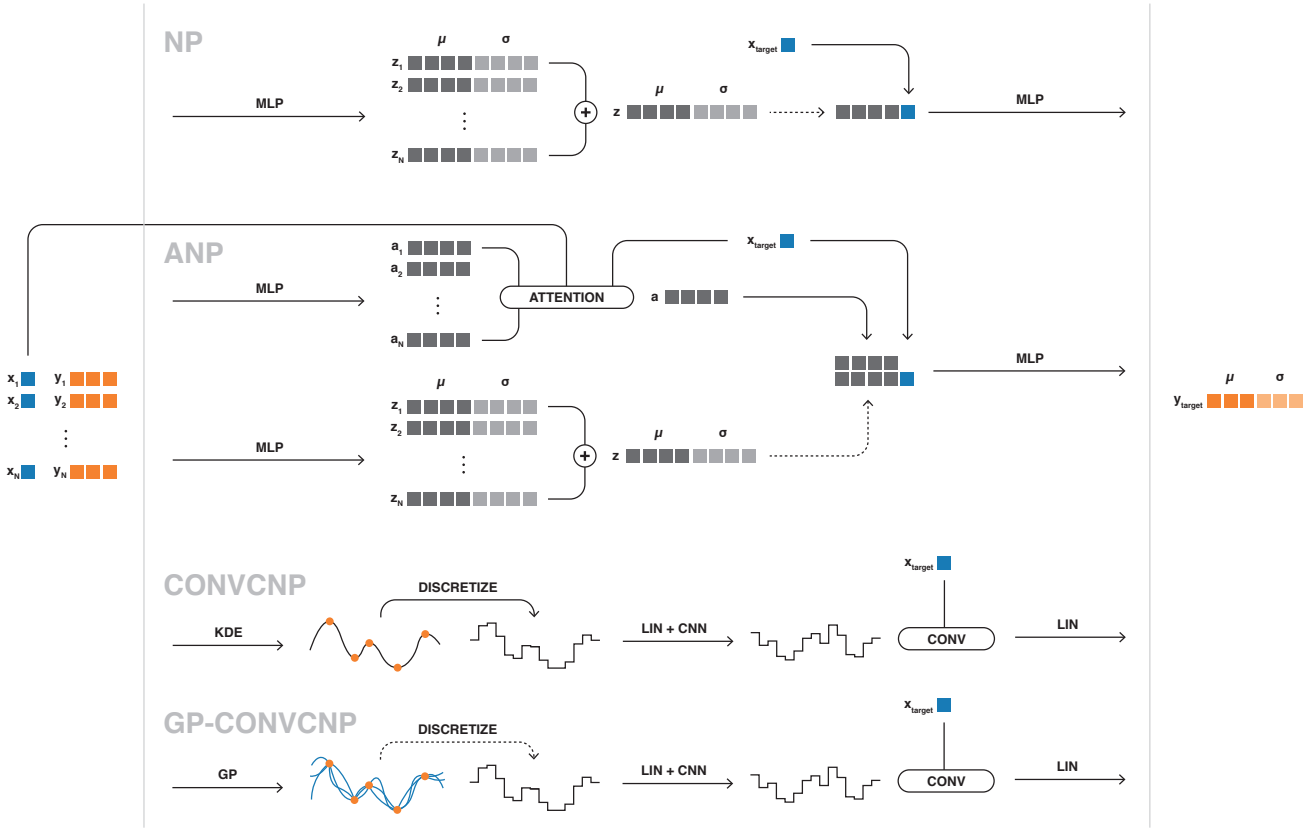

Figure A.1: Schematic overview of the different methods used in this work. Dotted lines indicate sampling and we use the following acronyms: multilayer perceptron (MLP), kernel density estimate (KDE), Gaussian Process (GP), linear layer (LIN), convolutional neural network (CNN). (*First row*) Neural Processes (NP) encode each context point (x_c, y_c) into a representation z_c. These are then averaged to form a global representation z. A sample from the global representation is concatenated with the target input x_t to predict the target output y_t. (*Second row*) Attentive Neural Processes (ANP) contain a NP, but have a second deterministic path. In this path, the context pairs are also encoded separately into representations $\mathbf{a_c}$. These are then combined via an atttention mechanism that uses x_t as the query, $\mathbf{x_c}$ as the keys and $\mathbf{a_c}$ as the values. The resulting representation a is concatenated with the representation from the NP path and the target input to predict the target output. (*Third row*) ConvCNP performs a kernel density estimate on the context observations $(\mathbf{x_c}, \mathbf{y_c})$, thus mapping to a continuous representation. This representation is evaluated on a grid, i.e. discretized, and a projection and CNN operate on the discretized representation. The result is evaluated at a target input x_t by performing a convolution with the discretized representation and finally projected to predict the target output. (*Fourth row*) GP-ConvCNP works similar to ConvCNP, but instead of a deterministic kernel density estimate a Gaussian Process is applied to the context. We sample from the GP posterior and discretize the result, continuing with the same operations as in ConvCNP. Note that for visual purposes, the KDE and GP outputs are one-dimensional, but in reality the output space can have any number of dimensions.

$$\log p(\mathbf{y_t}|\mathbf{x_t}, \mathbf{x_c}, \mathbf{y_c}) \geq \mathop{\mathbb{E}}_{z\sim q(z|\mathbf{x_t},\mathbf{y_t})} \log p(\mathbf{y_t}|\mathbf{x_t}, z) - D_{KL}\left(q(z|\mathbf{x_t}, \mathbf{y_t})||p(z|\mathbf{x_c}, \mathbf{y_c})\right) \tag{5}$$

$$\approx \mathop{\mathbb{E}}_{z\sim q(z|\mathbf{x_t},\mathbf{y_t})} \log p(\mathbf{y_t}|\mathbf{x_t}, z) - D_{KL}\left(q(z|\mathbf{x_t}, \mathbf{y_t})||q(z|\mathbf{x_c}, \mathbf{y_c})\right) \tag{6}$$

where the inequality follows from Jensen's inequality. To maximize the LHS it is sufficient to maximize the RHS, and Eq. (6) is what is being optimized in NP and ANP. q corresponds to what we designated as E in Section 2. Like for GP-CONVCNP, we approximate the expectation with a single sample during training.

In our implementation, we use Adam with an initial learning rate of 0.001. We train each model for $600\,000$ batches with a batch size of 256. We repeatedly multiply the learning rate by $\gamma = 0.995$ after training for 1000 batches.

C DATA & EVALUATION DETAILS

C.1 SYNTHETIC DATA

For all synthetic time series draws we define the x-axis to cover the interval $[-3, 3]$. As outlined in Section 3.2, we draw N context points randomly from this interval, with N a random integer from the range $[3, 100)$. We then draw M target points in the same manner, with M a random integer from $[N, 100)$. During training, we add the context points to the target set so that the methods learn to reconstruct the context. These are the different types we evaluate:

1. Samples from a Gaussian Process with a Matern-5/2 kernel with lengthscale parameter $l = 0.5$. The kernel is given by

$$k(x, x') = \left(1 + \frac{\sqrt{5}|x - x'|}{l} + \frac{5|x - x'|^2}{3l^2}\right) \cdot \exp\left(-\frac{5|x - x'|}{l}\right) \tag{7}$$

2. Samples from a Gaussian Process with a weakly periodic kernel that is given by

$$k(x, x') = \exp\left(-\frac{|x - x'|^2}{8}\right) \cdot \exp\left((\cos(8\pi x) - \cos(8\pi x'))^2\right) \cdot \exp\left((\sin(8\pi x) - \sin(8\pi x'))^2\right) \tag{8}$$

3. Fourier series that are given by

$$f(x) = a_0 + \sum_{k=1}^{K} a_k \cos(kx - \phi_k) \tag{9}$$

where K is a random integer from $[10, 20)$ and a_k (including a_0) as well as ϕ_k are random real numbers drawn from $[-1, 1]$.

4. Step functions, where we draw S stepping points along the x-axis, with S a random integer from $[3, 10)$. The interval between two stepping points is assigned a constant value that is drawn from $[-3, 3]$. We ensure that each interval is at least 0.1 units wide and that the step difference is also at least 0.1 units in magnitude.

C.2 TEMPERATURE TIME SERIES

The temperature dataset we work with is taken from `https://www.kaggle.com/selfishgene/historical-hourly-weather-data`. It consists of hourly temperature measurements in 30 US and Canadian cities as well as 6 Israeli cities, taken continuously over the course of ∼5 years. Occasionally there are NaN values reported in the dataset, we either crop those when at the begging/end of a sequence or fill them via linear interpolation. We use the US/Canadian cities as our training and validation set and the Israeli cities as our test set. For both training and testing we draw random sequences of length 720 (i.e. 30 days) from the corresponding set, and then draw N context points and M target points from the sequence, with N from the interval $[20, 100)$ and M from $[N, 100)$. The temperatures for each city are normalized by their respective means and standard deviations, and we define the time range for a given sequence to be $[0, 3]$, so that one time unit is equivalent to 10 days. We evaluate each seed for a model with 100 random samples and report the mean and standard deviation over 5 seeds for each model. For convenience, we include the data with our implementation.

C.3 POPULATION DYNAMICS

We simulate population dynamics of a predator-prey population with a Lotka-Volterra model. Let X be the number of predators at a given time and Y the number of prey. We draw initial numbers X from $[50, 100)$ and Y from $[100, 150)$. We then draw time increments from an exponential distribution and after each time increment one of the following events occurs:

1. A single predator is born with probability proportional to the rate $\theta_0 \cdot X \cdot Y$
2. A single predator dies with probability proportional to the rate $\theta_1 \cdot X$

3. A single prey is born with probability proportional to the rate $\theta_2 \cdot Y$
4. A single prey dies with probability proportional to the rate $\theta_3 \cdot X \cdot Y$

The rate of the exponential distribution we draw time increments from is the sum of the above rates. Each population is simulated for 10000 events, and we reject populations that have died out, populations that exceed a total number of 500 individuals at any given point, as well as those where the accumulated time is larger than 100 units. To get value ranges that are better suitable for training, we rescale the time axis by a factor 0.1 and the population axis by a factor 0.01. For each population we draw θ_0 from $[0.005, 0.01]$, θ_1 from $[0.5, 0.8]$, θ_2 from $[0.5, 0.8]$ and θ_3 from $[0.005, 0.01]$. These parameters result in roughly 2/3 of the simulated populations matching our criteria. We also tried the parameters reported in Gordon et al. [2020], but found that we had to reject more than 90% of populations, which meant an unreasonably long training time, as the simulation process for the populations is difficult to parallelize and thus rather slow. The N context points and M target points are again drawn randomly from a population, with N from $[20, 100)$ and M from $[\max(70, N), 150)$.

We evaluate models trained on simulated data on real world measurements of a lynx-hare population. The data were recorded at the end of the 19th and the start of the 20th century by the Hudson's Bay Company. To the best of our knowledge, the data represent recorded trades of pelts from the two animals and not direct measurements of the populations. There is no unique source for the data in a tabular format, but we used `https://github.com/stan-dev/example-models/blob/master/knitr/lotka-volterra/hudson-bay-lynx-hare` and include the data with our code for convenience. For evaluation, we normalize the data so that the mean population matches the mean of populations in the simulated data and the time interval matches the mean duration of a simulated population.

C.4 WASSERSTEIN DISTANCE

As outlined in Section 3, we seek to compare the distribution of samples from a model with a reference distribution, which we have access to for the synthetic examples sampled from a GP in the form of the prediction from the same GP. Comparing distributions is usually done with either some form of *f-divergence* (e.g. the Kullback-Leibler divergence) or with an *Integral Probability Measure* (IPM) f-divergences require evaluations of likelihoods in both distributions, while we can only evaluate those under the GP posterior but not in our models. IPM only compare samples from the distributions and are thus suited for our scenario.

Table A.1: This table corresponds to the rightmost column in Table 2, i.e. it shows results on the real world population dynamics data. In Table 2, the evaluation was performed as seen in Fig. 3, meaning one contiguous interval on the data was selected as the target region and the rest of the data is provided as context, following Gordon et al. [2020]. Here we instead sample the context and target points randomly from the entire interval, like we do in the other experiments as well. For each seed, we average over 100 random draws and report the standard deviation over 5 seeds as errors. While CONVCNP maintains leading performance in terms of reconstruction error, GP-CONVCNP significantly outperforms the other methods in predictive performance, similar to what we found in Table 2. All methods perform worse compared to the evaluation method used in Table 2.

	Predictive LL↑	Recon. Error↓
NP	-36.735 ± 4.137	0.952 ± 0.024
ANP	-38.717 ± 3.572	0.718 ± 0.018
CONVCNP	-28.762 ± 1.958	$\mathbf{0.272 \pm 0.008}$
GP-CONVCNP	$\mathbf{-19.252 \pm 1.846}$	0.343 ± 0.020

One of the more well-known measures from this group is the Wasserstein distance given by:

$$W_p(P, Q) = \min_{\pi} \left(\sum_{i=1}^{|P|} ||x_i - y_{\pi(i)}||^p \right)^{1/p} \tag{10}$$

where $P = \{x_i\}_i$ and $Q = \{y_i\}_i$ are collections of samples from the two distributions. In colloquial terms, the Wasserstein distance is the minimum overall distance between sample pairs, taken over all possible pairings between samples from the two distributions. For this reason the Wasserstein-1 distance is also called the *Earth Mover Distance*. p is the only hyperparameter we need to select, making this measure a very convenient choice. We set $p = 2$ so that the underlying distance metric becomes the Euclidean distance.

D ADDITIONAL RESULTS

In this section we show some additional results, specifically we show the performance as a function of the number of context points (Fig. A.2) as well as examples for NP and ANP on the temperature time series dataset in Fig. A.3 and on the population dynamics dataset in Fig. A.4. We also show results on the population dynamics dataset in Table A.1, using a different evaluation method compared to the main manuscript.

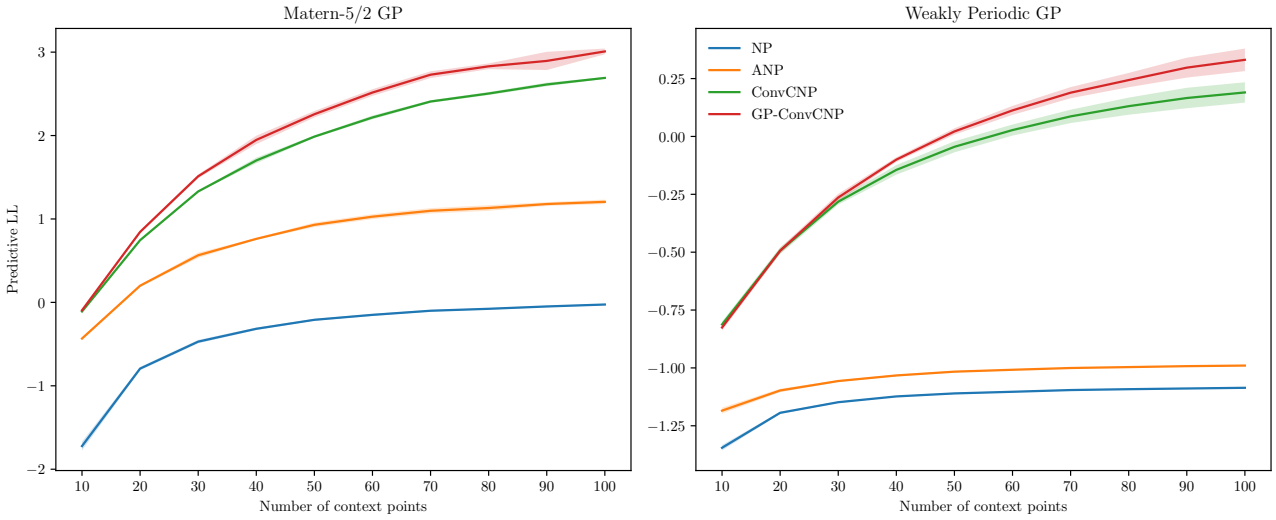

Figure A.2: Performance as a function of the number of context points for the two synthetic GP examples. For sparse context points, our model and ConvCNP are on par, while an increasing number of context points leads to an advantage for our model.

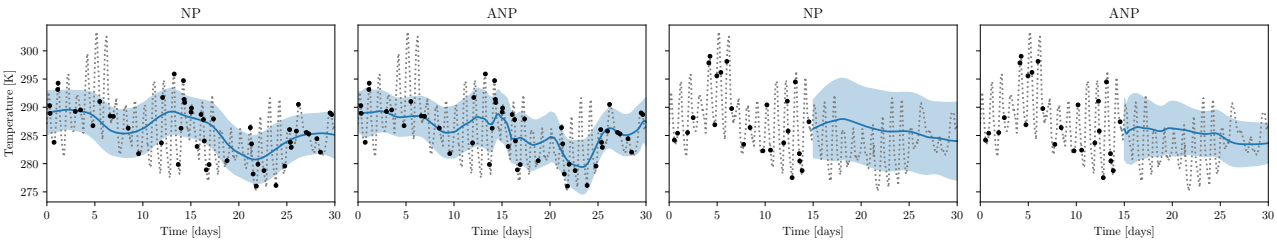

Figure A.3: Examples from the temperature time series test set for NP and ANP. For the interpolation task (left) we provide context points from the full sequence, for the extrapolation task (right) we provide context points in the first half of the sequence and evaluate the second. Both methods are unable to fit the context points, likely because the frequency is too high to be represented in the models.

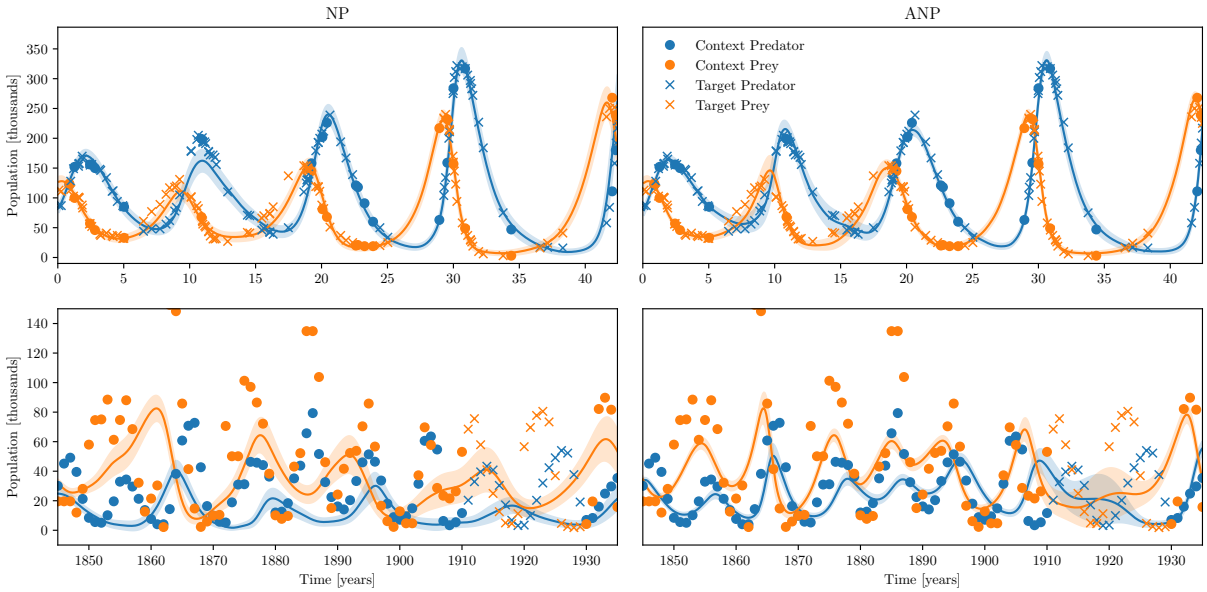

Figure A.4: Example of NP and ANP applied to the simulated Lotka-Volterra population dynamics (top) and to the real Hudson Bay Company lynx-hare dataset (bottom). Similar to ConvCNP and GP-ConvCNP, seen in Fig. 3, both work well on simulated data. On the real data, however, both struggle to fit the context points and produce a poor prediction for the test interval.

E IMAGE EXPERIMENTS

For a more complete comparison of our model with ConvCNP, we include image experiments, specifically MNIST, CIFAR10 and CelebA. For the latter two, we work with resampled images at 32^2 resolution. The context set has a size drawn from $[20, 400)$ ($[20, 300)$ for MNIST), the target set a size drawn from $[50, 400)$, and we reconstruct both target and context points during training. We evaluate the average log-likehood of the model predictions on the respective test sets, as seen in Table 3. The implementation of ConvCNP is again taken directly from the official repository, and we leave the architecture unchanged with the exception of swapping the kernel interpolation for a GP to make the comparison fair. All other hyper parameters are the same as in the time series experiments.

Examples for both ConvCNP and GP-ConvCNP can be seen in Fig. A.5, Fig. A.6 and Fig. A.7, with each example taken from the test sets. There is not noticeable visual difference between the two model, so we assume that the improved performance is due to better estimates of the predictive uncertainty (i.e. the standard deviation of the predicted Gaussian). In terms of performance, we found that inference takes roughly 1.5x as long for GP-ConvCNP as it does for ConvCNP, which we believe is still an acceptable tradeoff.

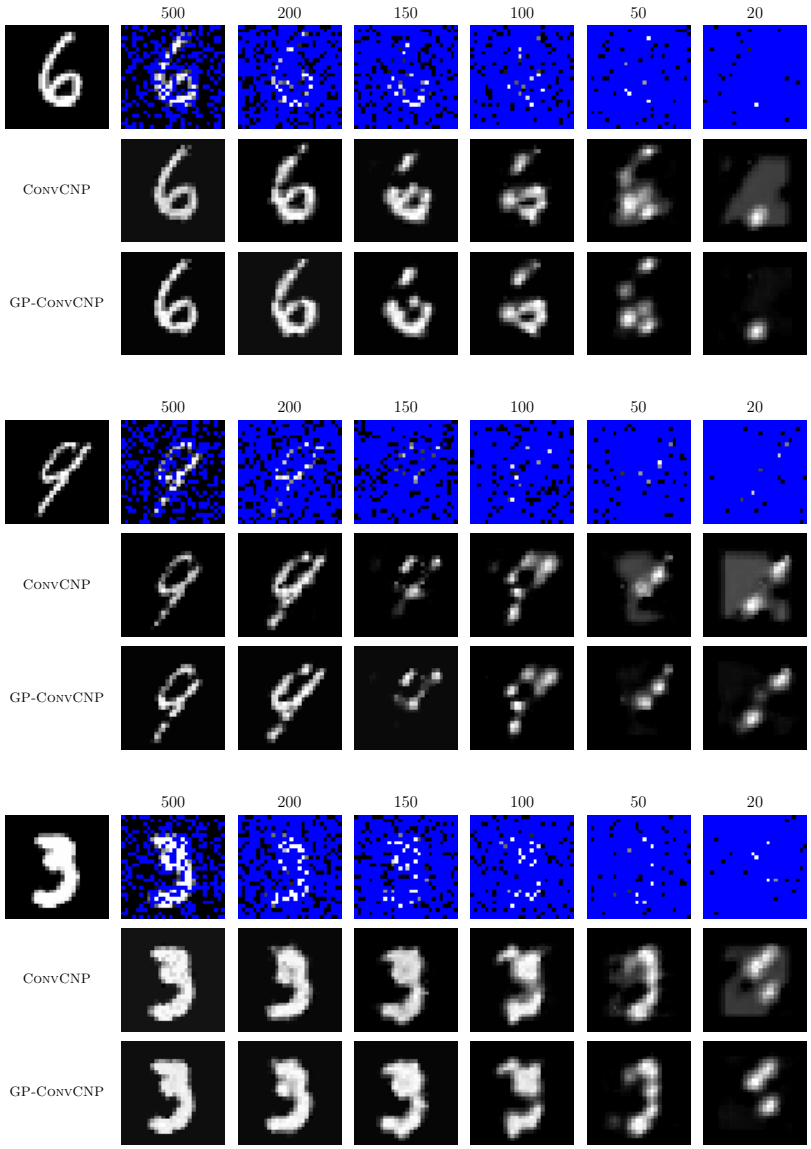

Figure A.5: Examples for ConvCNP and GP-ConvCNP applied on MNIST test data. Models were trained on the training set. Numbers indicate the number of context points and the top left panel shows the reference image for each case.

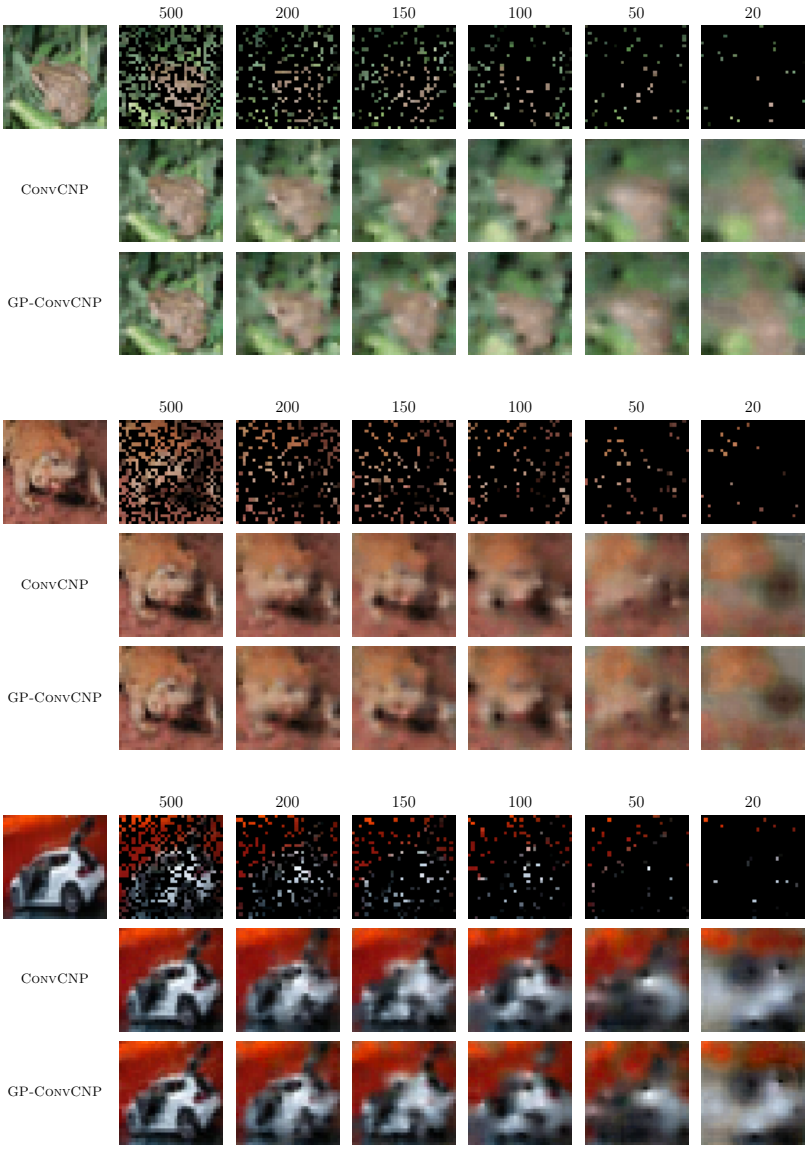

Figure A.6: Examples for ConvCNP and GP-ConvCNP applied on CIFAR10 test data. Models were trained on the training set. Numbers indicate the number of context points and the top left panel shows the reference image for each case.

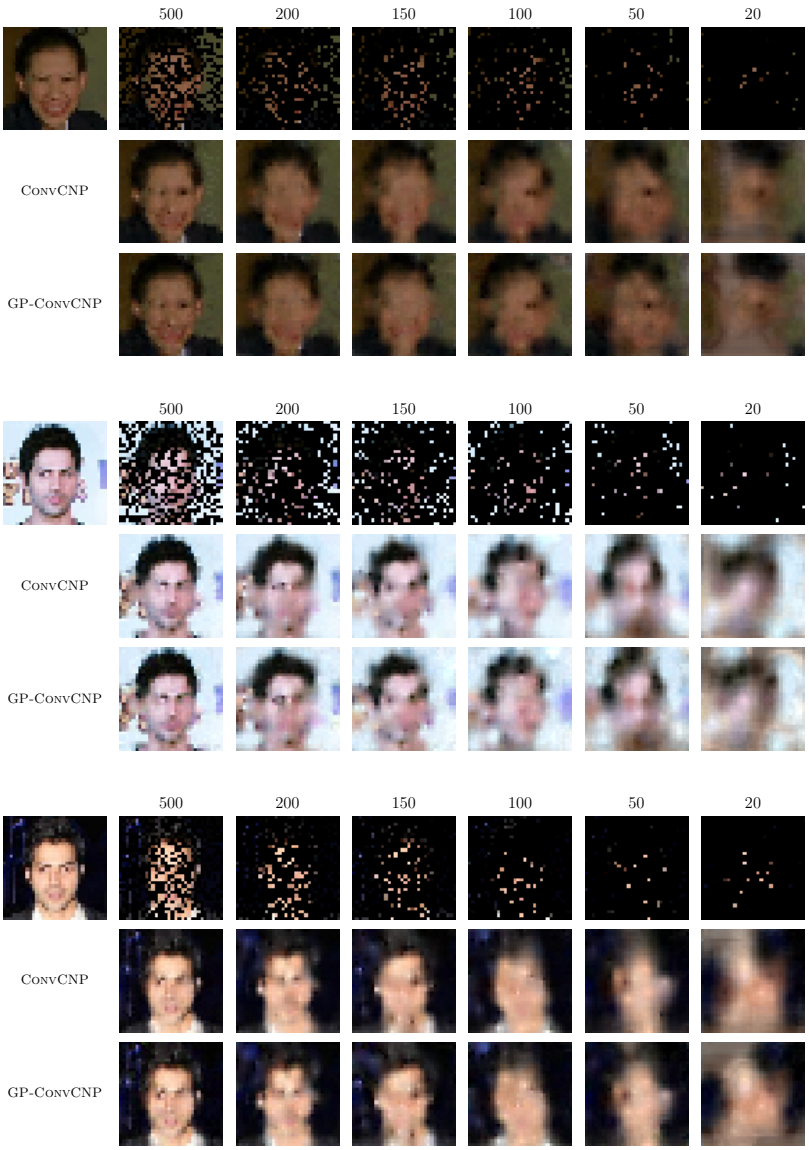

Figure A.7: Examples for ConvCNP and GP-ConvCNP applied on CelebA test data, resized to 32x32 resolution. Models were trained on the training set. Numbers indicate the number of context points and the top left panel shows the reference image for each case.