Indian Institute of Technology, Gandhinagar

Probabilistic Machine Learning - Project Report

Conditional Neural Adaptive Processes (CNAPs)

Authors
Bhavesh Jain 20110038
Dhairya Shah 20110054
Lipika Rajpal 20110102
R Yeeshu Dhurandhar 20110152
Rahul Chembakasseril 20110158

21st November, 2023

Under the guidance of
Professor Nipun Batra
Madhav Kanda
Sarth Dubey

Contents

0.1 Introduction

0.1.1 Multi-Task, Few Shot Learning

In Machine learning problems, it is a common scenario not to have a lot of data for training a model from scratch. For a similar task, say, movie recommendation, there are some users who have a lot of data and some users who have very little data. In this case, can we use the data from users with a lot of data to help users with little data?

Well, in this case, we might think about few-shot learning. Few-shot learning means giving a few examples to the model; the model can learn from these examples.

Humans are very good at few-shot learning. We can detect the pattern encoded in the examples and apply the pattern to new examples. For example, if we are given a few examples of a cat, we can detect the pattern of a cat and apply the pattern to new examples.

Nowadays, LLM has been used to solve a few shot learning problems. LLM can be used to encode the pattern of the examples and apply the pattern to new examples. This has led to further developments and interest in these fields.

Now the question is, can we use something similar using Neural Networks to solve a few shot learning problems? The answer is yes. We can use Neural Networks to solve a few shot-learning problems. In this case, we can use Neural Networks to encode the pattern of the examples and apply the pattern to new examples.

0.1.2 Conditional Neural Processes

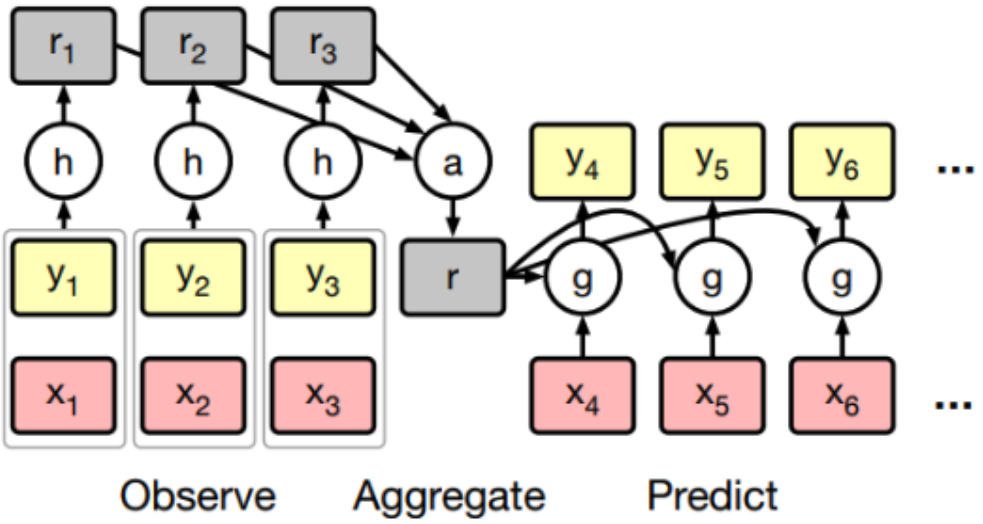
Conditional Neural Processes achieve the task of new shot learning by using the encoder-decoder architecture. We send the context points, i.e., the small amount of labeled data that we have, through the encoder. The encoder encodes the context points into a latent space. The decoder then uses the latent space in the prediction for the target points, i.e., the unlabelled data.

We can have a variable number of context points. Therefore, it is important to have a permutation-invariant encoder. The representation generated by the encoder should not depend on the number or order of the context points.

We pass each of the context points through the encoder. The encoder generates a representation for each of the context points. We then aggregate the representations of the context points into a single representation. We can use the mean or the sum of the representations of the context points as the representation of the context points.

We then concatenate the representation of the context points with the representation of the target points. We then pass the concatenated representation through the decoder. The decoder generates the prediction for the target points.

Both encoders and decoders are neural networks and can be trained using backpropagation. Now, once a CNP is trained, we can make predictions on a new unseen dataset of similar properties during the forward pass with the help of a few labelled data points.



$$p\left(\boldsymbol{y}^{*}|\boldsymbol{x}^{*},\boldsymbol{\theta},D^{\tau}\right)=p\left(\boldsymbol{y}^{*}|\boldsymbol{x}^{*},\boldsymbol{\theta},\boldsymbol{\psi}^{\tau}=\boldsymbol{\psi}_{\boldsymbol{\phi}}\left(D^{\tau}\right)\right).$$

Figure 1: Conditional Neural Process

Source: Conditional Neural Process

0.1.3 HyperNetworks

Hypernetworks are a type of neural network architecture that revolves around the idea of learning the parameters of another neural network, often referred to as the "target network" or "task-specific network." The primary goal is to enable the efficient adaptation of a neural network to multiple tasks, making it suitable for meta-learning scenarios.

We define a neural network called the "hypernetwork" that learns to predict the weights of the target network based on the task at hand. A set of context points(meta-input) is passed to the hypernetwork. Based on this context, the hypernetwork sets the weights of the target network. The target network then predicts the desired output on unseen/test points.

The meta-learning is executed as we try to minimize a pre-defined loss between the true values and the predicted output of the target network. This loss is used in back-propogation to update the weights of hypernetwork only. Note that the weights of the target network are set by the hypernetwork in each forward pass.

Here's an overview of the hypernetworks used in this paper:

- **Hypernetwork Overview:**
  This network has two hidden layers, each comprising 512 nodes. The input to the hypernetwork is the context(meta input), formed by concatenating the input features (in this case, the x-coordinate) and the corresponding output value. The hypernetwork's output is a tensor with a size matching the total number of parameters in the target network.
- **Target Network:**
  The target network adopted for this experiment is a straightforward Multilayer Perceptron (MLP)

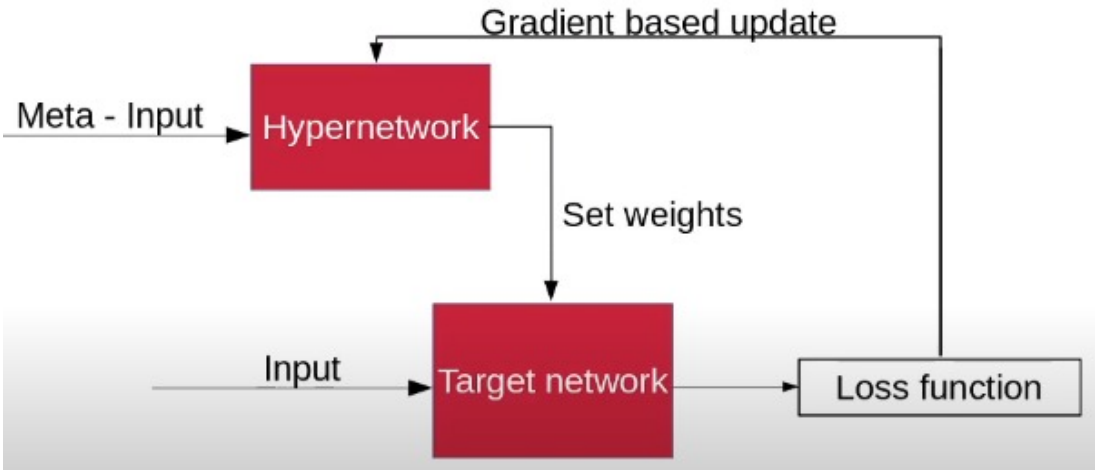


Figure 2: Hypernetworks for meta-learning

Source: HyperNetworks for meta-learning

with a single hidden layer of 16 nodes. The intentional simplicity of the target network prevents overfitting, as an increase in parameters might lead to the function attempting to simultaneously fit all tasks, resulting in learning an average of the tasks.

- **Forward pass:**
  During a forward pass, the context (meta-input) is fed into the hypernetwork, producing a flat tensor with a size equal to the number of parameters in the target network. A function is defined to reshape this tensor into the *state dictionary* of the target network, and these new parameters are assigned to the target network. Subsequently, the x-coordinate is passed through the target network, yielding the corresponding output (Y value).

0.1.4 How to train a meta-model?

Meta-learning is also known as learning how to learn. We can train a meta-model by using a lot of small datasets or tasks. For each task, we send a few context points to the model through which it captures the information about the task. Then, using this information, we can make predictions on the target points. We can then use the predictions to calculate the loss and use-back propagation to update the parameters of the model.

Therefore, we are trying to make our model learn how to learn from a few examples and then use that learning to make predictions on new examples.

During training, if we keep providing our model with a large variety of tasks with a few examples, then our model will learn to generalize and make predictions on a new task with a few examples.

0.1.5 Make-Moons example to show Meta Learning

e used the Make-Moons dataset to showcase meta-learning. It is an easy task to make moons classification using makemoons and can be achieved using a simple neural network. However, in this experiment, we used a makemoons dataset, which is randomly rotated between 0 to 360 degrees.

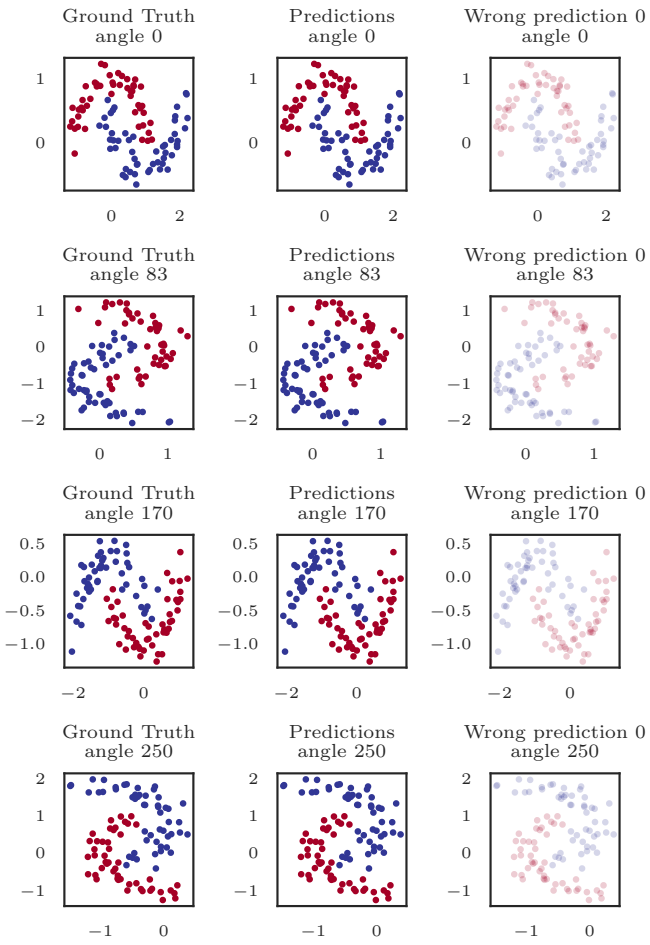


Figure 3: Predictions on Rotated Make-Moons Dataset on four random angles which it had not seen during the test time

During the forward pass itself, we will be sending a few context points to the model. The model will then learn to make predictions on the target points. The angle at which the dataset is rotated might be completely new for the model i.e., there is a high chance it has not seen this rotation angle during training. However, the model will still be able to make predictions on the target points.

During training, we took 100 make-moons datasets at random angles. We divided each dataset in the context set and target set by choosing 20 context points randomly. After training the model on these 100 datasets, we tested the model on new datasets with four new random angles. The model was able to make predictions on the target points of the new dataset.

Therefore the encoder was able to capture the information about the task from the context points and pass it on to the decoder, which uses that information to make predictions on the target points.

0.1.6 Conditional Neural Adaptive Processes

Conditional Neural Adaptive Processes are inspired by Conditional Neural Processes. These are used for classification tasks on new datasets with unseen tasks and a variable number of classes. They take in a few context points(in this case, labeled images) and try to predict the class of the target points(unlabelled images).

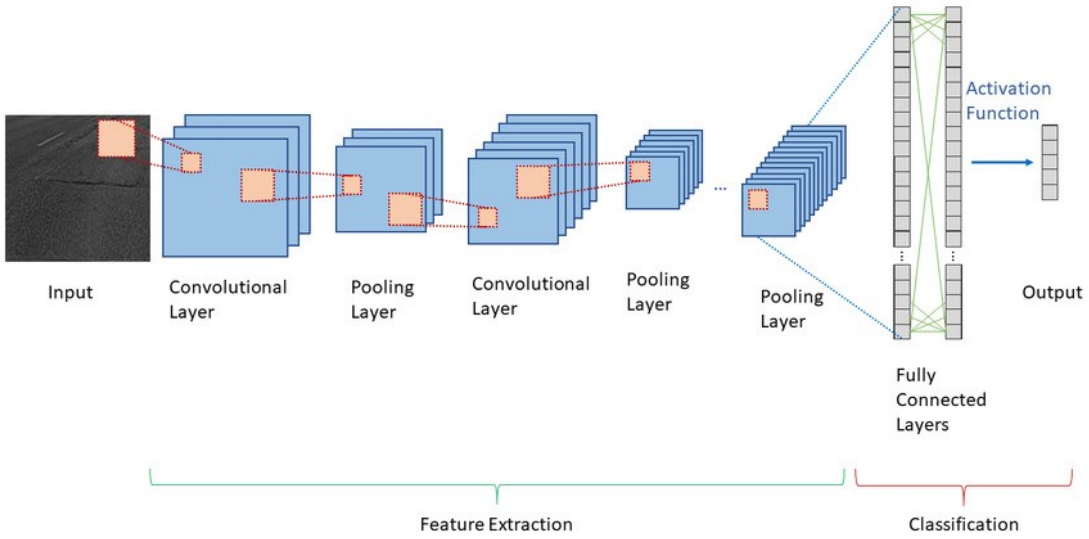


Figure 4: Convolutional Neural Network

Source: CNN

They differ from CNP in one major aspect. Instead of getting a vector representation from the context points, they use the context points to get weights for the model parameters. This is similar to hypernetworks.

In CNP, we know all the weights of the model before the forward pass. In CNAPs, we get some of the weights of the model during the forward pass. This enables us to adapt our architecture to variable numbers, outputs, or classes. While CNPs have a fixed architecture and are generally used for regression tasks, CNAPs can be used for classification tasks with a variable number of classes.

0.1.7 CNN VS CNAPs

Here is the figure 4 of traditional CNN architecture. It has two major parts: a feature extraction layer and a classification layer. The feature extraction layer is used to extract features from the input image. The classification layer is used to classify the image into one of the classes.

This architecture has provided us with amazing results on image classification tasks. However, it has a fixed architecture. It cannot adapt to a variable number of classes. Also, the feature extractor cannot adapt to new classes. If we want to add a new class to the dataset, we will have to retrain the whole model.

Therefore, if we need to make this architecture "meta," we would have to modify both these components such that they can adapt for the new classes according to the examples(few-shots) given to them. We will need to tweak the feature extractor layer to adapt to new features and will need the classification layer to adapt to a new and variable number of classes.

0.1.8 Adaptive Feature Extractor

In CNAPs we make the ResNet architecture adaptive by inserting FiLM layers in the network. The layers do affine transformations of feature maps and help them to adapt to a new task. The parameters of the

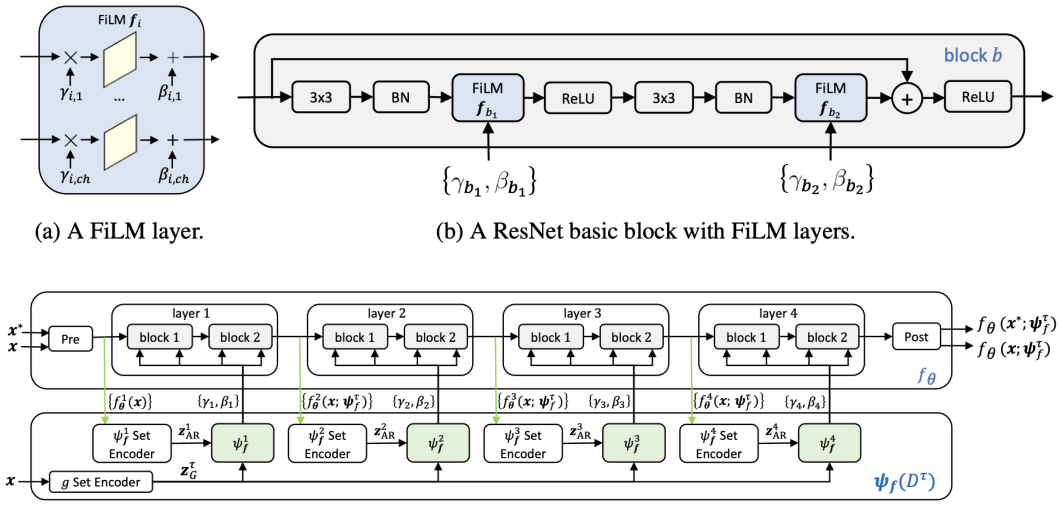


Figure 5: Adaptive Feature Extractor

Source: Original Paper

FiLM layer are task-specific and are generated from the context points. The FiLM layer is inserted after each convolutional layer. The FiLM layer takes in the feature maps from the previous layer and the parameters generated from the context points and generates new feature maps. The new feature maps are then passed to the next layer. This process is auto-regressive.

To enable this auto-regressive process, CNAPs initially generate a global encoding for the context images. Now, for each FiLM layer, it takes in the global encoding and the feature maps from the previous layer and generates new parameters. These parameters are then used to generate new feature maps. This process is repeated for each FiLM layer. This process can be seen in figure 5.

0.1.9 Linear Weights Classifier

In CNAPs, designing a classifier that can adapt to new classes is a bit tricky. We have a variable number of classes; therefore, the weights and biases of the models are not fixed. To generate weights for a particular class, we also need to know the output of the feature extractor layer for a particular class.

The outputs of the feature extractor layer will tell us about the features present in the class, and we can then decide the importance of each feature accordingly.

During the forward pass, when we get the outputs from the feature extractor for the context images, we segregate them by classes. Then, for a particular class, we take a mean over the representations. This aggregation ensures permutational invariance. The aggregated representation is then passed on to a hypernetwork to generate weights and biases for the classifier for that particular class.

0.2 CNAPs for Regression

Working with the Conditional Neural Adaptive Processes for classification motivated us to implement the same for the regression task. The implementation can be used in three different modes:

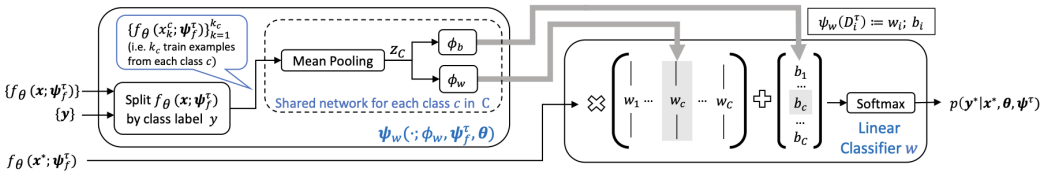


Figure 6: Adaptive Linear Classifier

Source: Original Paper

- Base Model
- CNAPs without Autoregressive
- CNAPs with Autoregressive

0.2.1 Dataset Used

For this task, we have taken datasets generated using sine functions with different amplitudes and phases, as shown in figure 7. Each of the sine function acts as a task.

$$y = A\sin(x + \phi) \tag{1}$$

where $0.5 <= A <= 5$ and $0 <= \phi <= 2\pi$.

0.2.2 Base Model

This is a simple Multi-Layer Perceptron (MLP) trained on one task (figure 8). Here, x is input, and y is output. As seen in figure 9, it predicts the same output for all the tasks because no adaptivity is added to the model.

0.2.3 CNAPs without Autoregressive

This is the model architecture in which we have added adaptive network 10. The context points are passed to the Context Set Encoder (CSE), which takes a context point and outputs a vector of size 128. It is done for all the context points present and then aggregated and then passed to the γ and β Adaptation Networks (AN) whose number of input nodes is 128 (same as the number of output nodes of Context Set Encoder). The γ and β Adaptation Networks give an output of size equal to the number of neurons in a particular hidden layer. In our case, we have taken the size of all hidden layers to be 128. Then, affine transformation (multiplication with γ and addition with β) is performed with the outputs/activations of each node of the base model. This affine transformation helps the base model change the output based on a given task and its context points.

During training, first, the base model is trained using a particular task, and then its parameters are kept frozen. Further, the adaptive networks are trained. In each iteration, the context set is passed through CSE, generates the γ and β parameters for all the layers of the base models, and the predictions are made by passing the target points. The loss is taken between predictions and ground truth and back-propagated,

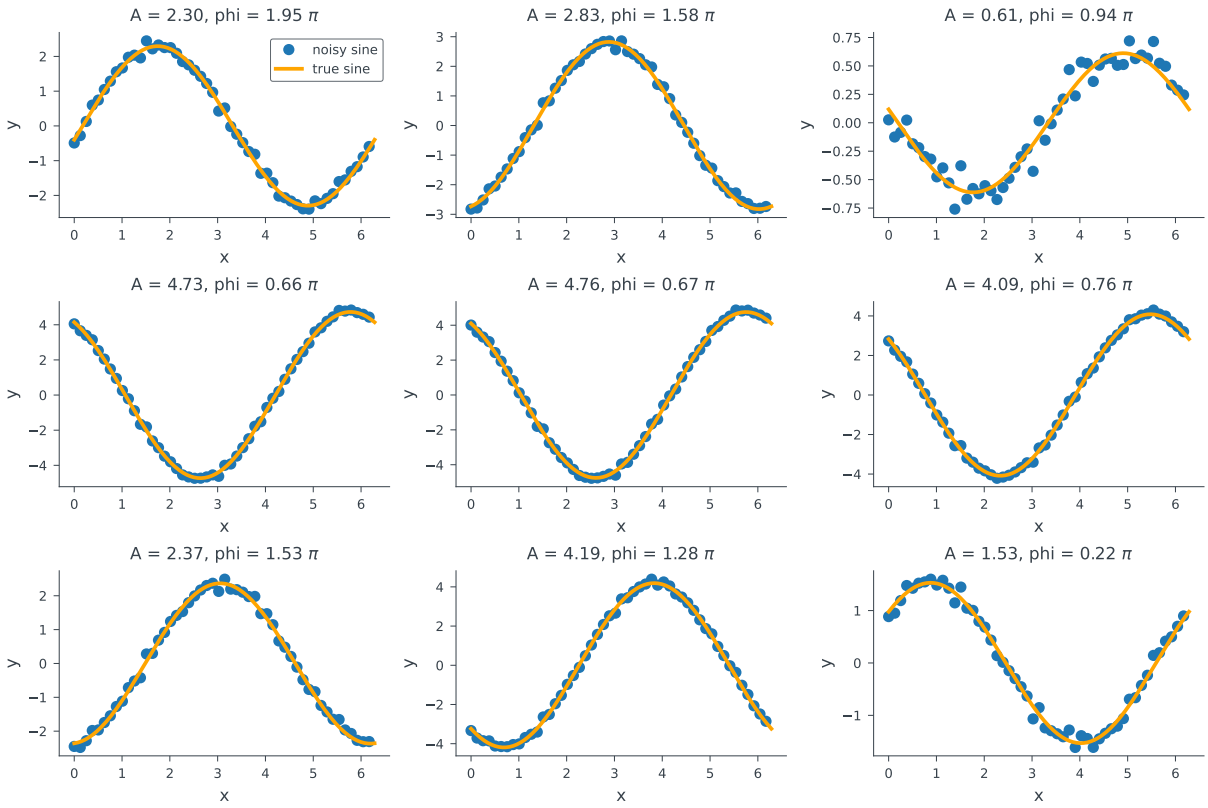


Figure 7: Some sine functions - dataset for CNAPs for regression

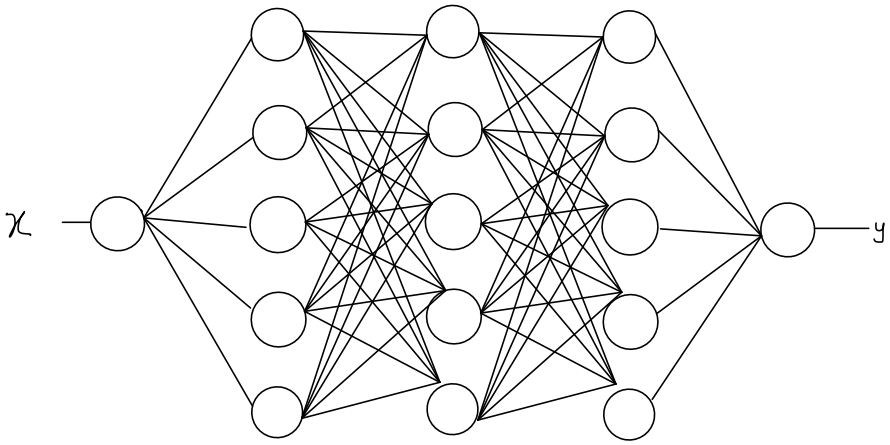


Figure 8: CNAPs for Regression: Base Model

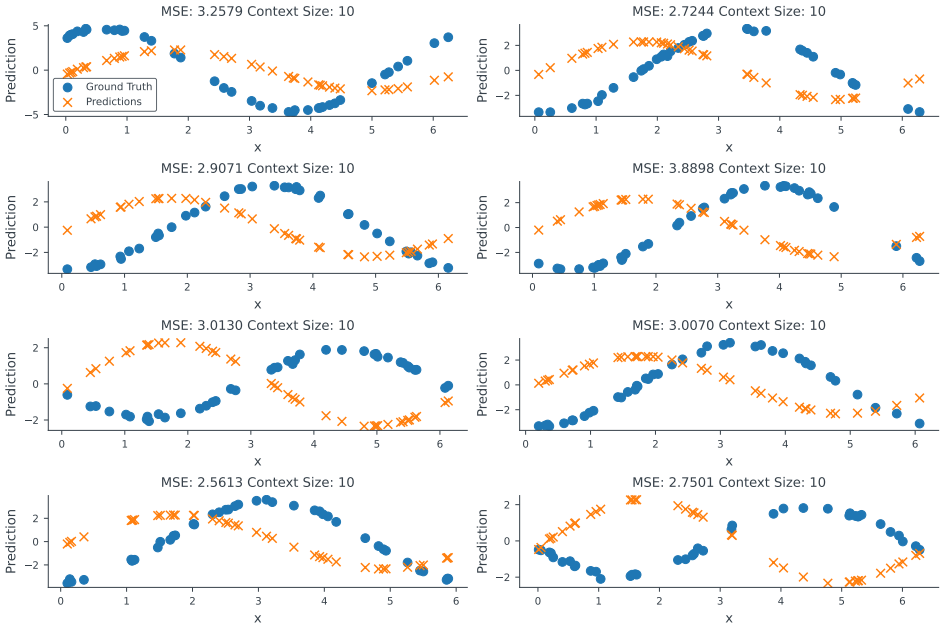


Figure 9: Predictions from base model

and all the parameters, including γ and β Adaptation Networks (AN) and Context Set Encoder, are modified.

During testing time, the context set is passed through the CSE, and predictions are made for target points.

It can be seen from figure 11, the model is adapting to the task, and predictions are better as compared to the predictions from base mode.

0.2.4 CNAPs with Autoregressive

In this case, an autoregressive part is added to the network as shown in figure 12. The activations of the previous layers of the base model are passed through Autoregressive Set Encoder (ArSE). The output of the ArSE is a vector of size 128. This vector, along with the output of CSE, is passed through the γ and β Adaptation Networks. In this case, the input size of the adaptation networks is 256 to accommodate the outputs of both the ArSE and CSE.

The training and testing process is the same as the without the autoregressive case, except that in this case, parameters of ArSE are also updated, and γ and β parameters are not generated at a time; on the contrary, they are generated after getting the output of ArSE of the previous activations.

As shown in figure 13, the predictions are very close to the true value. We can observe that as we move from the base model to CNAPs without Autoregressive to CNAPs with Autoregressive, the model is able to perform better for the meta-learning task. This is due to the increase in the ability to represent each of the context points as well as target points.

0.2.5 Effect of Context Size

From the figure 14, we can see that as context size increases, the predictions get closer to the true value, and the Mean Squared Error (MSE) reduces (as expected). The increased number of data points increases

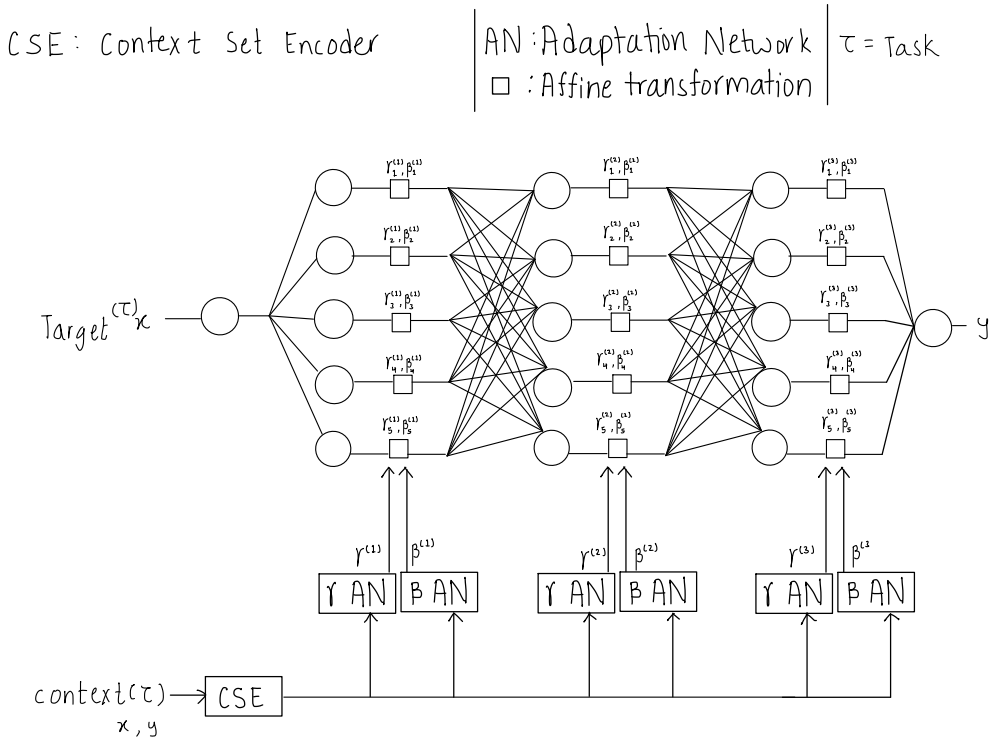


Figure 10: CNAPs for Regression: without Autoregressive

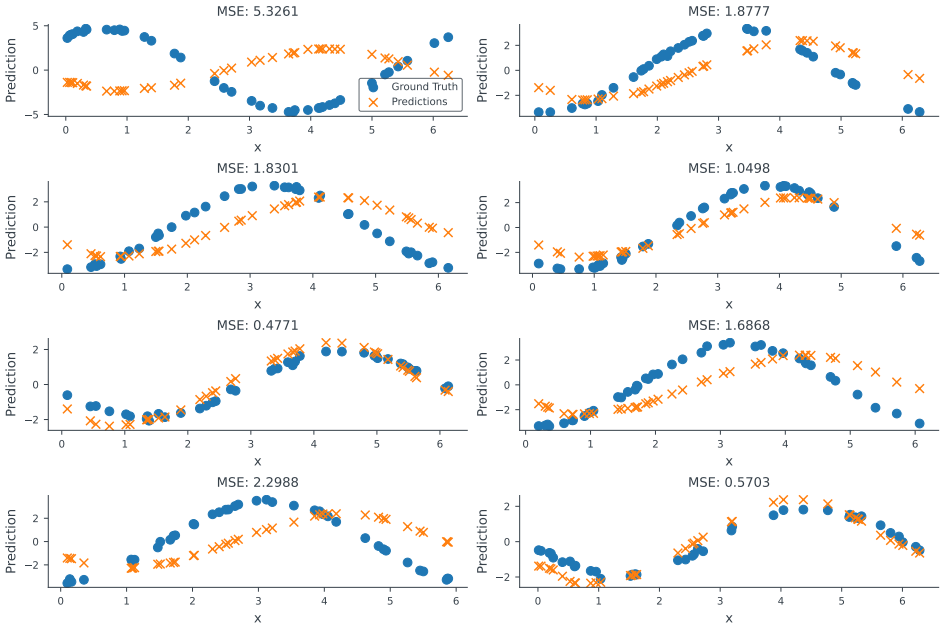


Figure 11: Predictions from CNAPs without Autoregressive

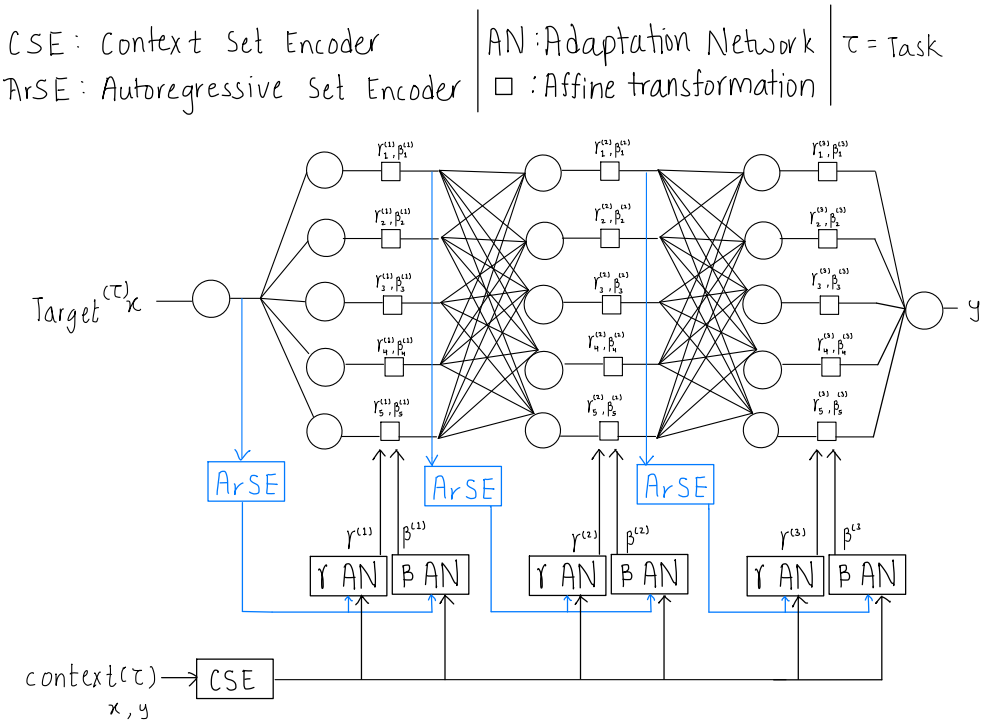


Figure 12: CNAPs for Regression: with Autoregressive

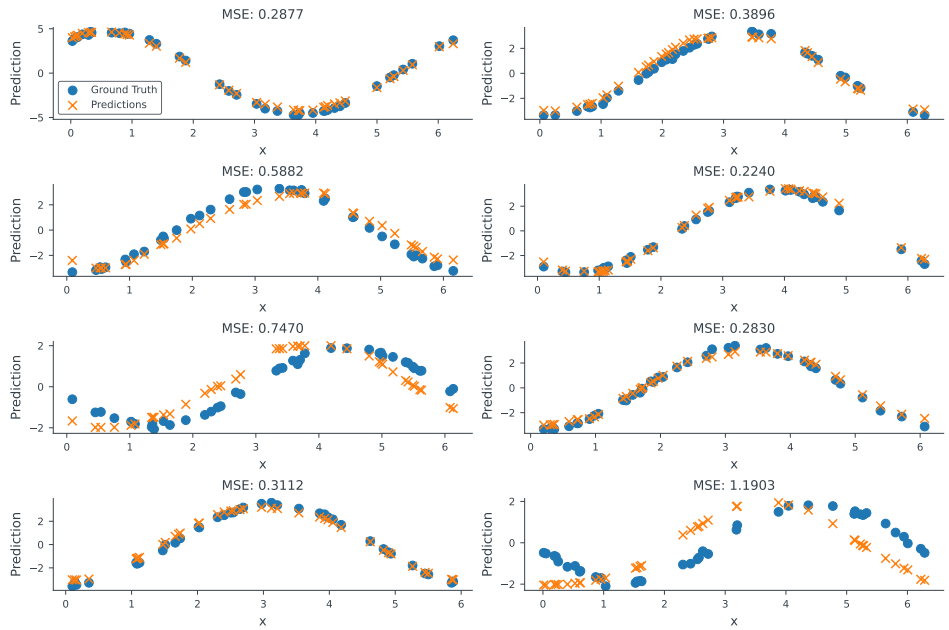
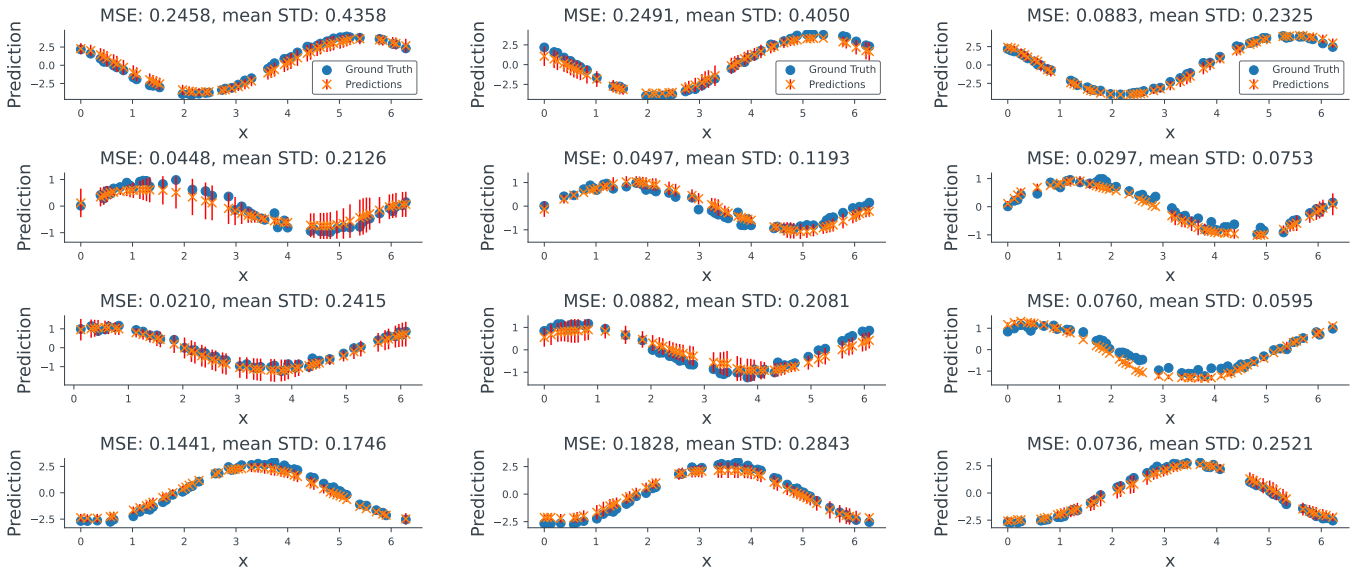


Figure 13: Predictions from CNAPs with autoregressive using context size = 10.



(a) Context size = 5 (b) Context size = 10 (c) Context size = 30

Figure 14: CNAPs for regression: as the context size increases, the prediction improves, and MSE and mean STD reduce. The red vertical lines show standard deviations.

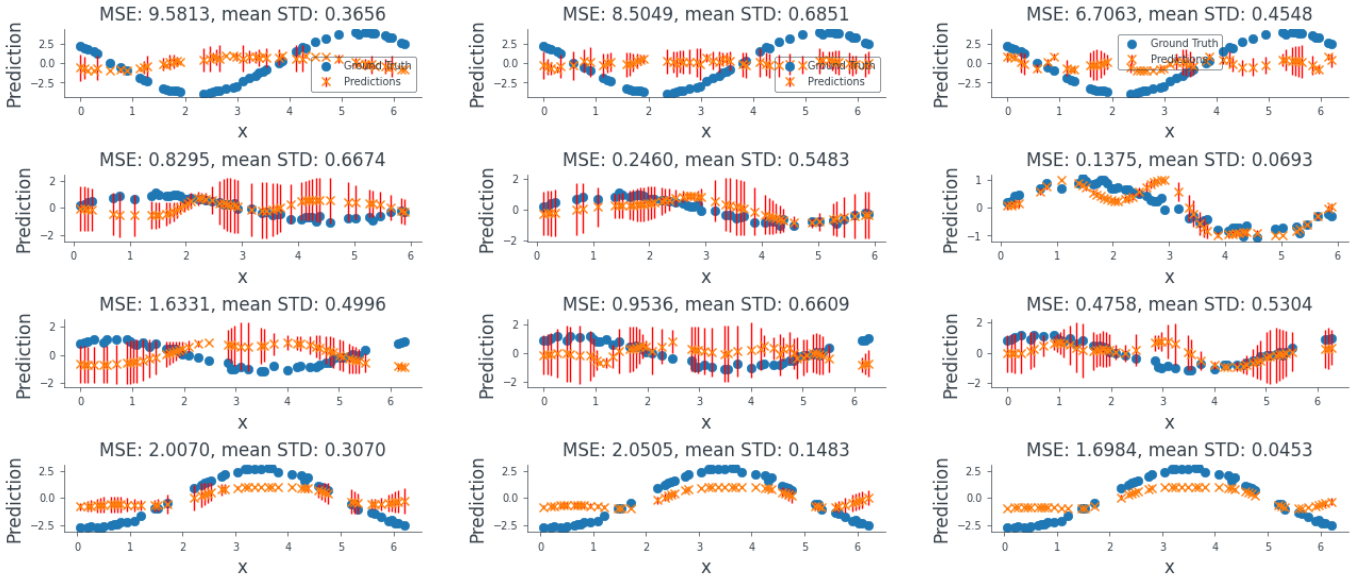
the information about the task at hand. Therefore, more information results in better representation and better performance. However, from the experiments, we observed that after some number of context points, the results did not change much, and sometimes, we observed reduced performance instead.

0.3 Baselines for comparison with CNAPs on Sine regression

0.3.1 HyperNetworks

The Sine-regression experiment incorporates hypernetworks as one of its baseline models. Here are the key details regarding this hypernetwork-based approach:

- **Meta Learning Tasks:**
  The experiment involves ten tasks to facilitate meta-learning, each representing different sine functions (7). These tasks are divided into training and test datasets. The training dataset comprises ten tasks, each with 50 points, while the test dataset has 50 points for each of the ten tasks.
- **Training:**
  The training process involves iterating through tasks in a specific manner. For each training epoch, the dataset is split in half, with one half serving as the context for the hypernetwork. A complete forward pass is then executed on the other half, and the loss is calculated. This process is performed iteratively for each task. The loss is determined as the Mean Squared Error between the predicted output and actual values. This training loop helps refine the hypernetwork and target network parameters to enhance the model's performance.
- **Results and observations:**
  We have observed that the hypernetworks try to average out all the tasks during training to minimize the loss. Due to this tendency, they could not clearly distinguish between different tasks



(a) Context size = 5 (b) Context size = 10 (c) Context size = 30

Figure 15: HyperNetworks: as the context size increases, the prediction improves, and the predictions align with the true functions more closely. The MSE loss over all tasks decreases upon increasing the context size, and the corresponding mean variance also decreases.

during the test phase. However, we have observed that as we increase the number of context points (from 5 to 30), the output function becomes better at predicting values closer to the actual values. In the following figure, we can observe that the predictions become smoother on increasing the number of context points and resemble the true function more closely. 15

0.3.2 Conditional Neural Processes (CNP)

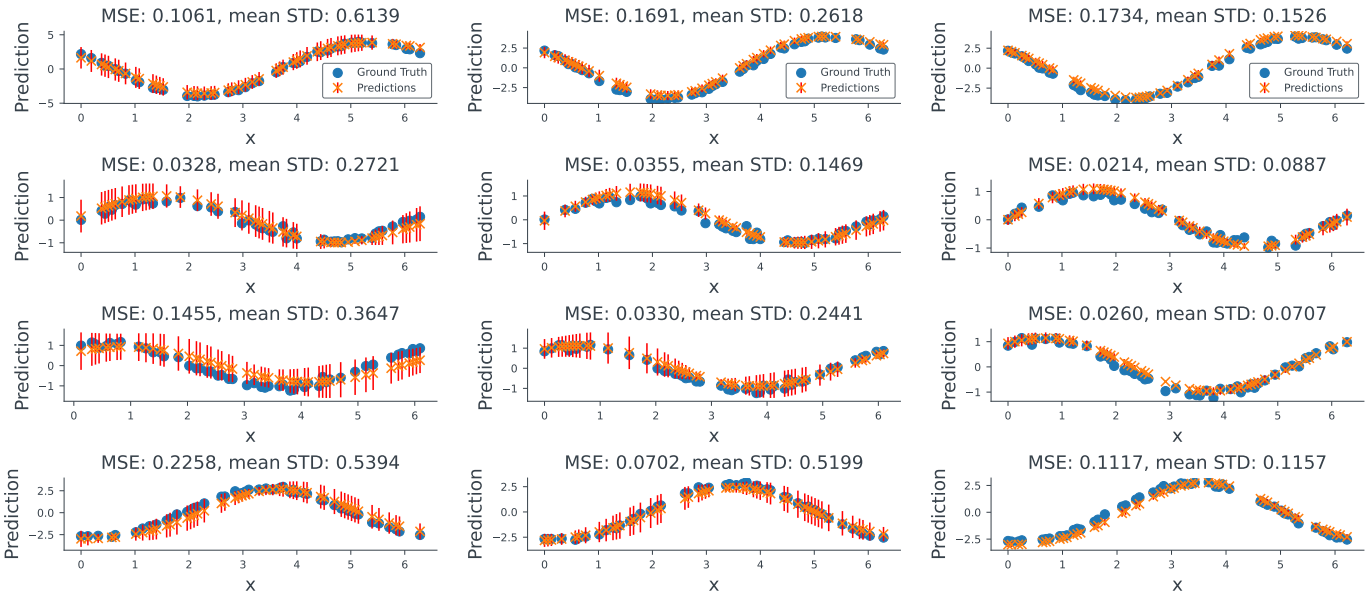
Conditional Neural Processes work very well on the regression problem. The number of parameters is also really low. We see that the standard deviation significantly decreases as we increase the number of context points.

0.3.3 Comparison of the performance of CNAPs with baselines

Model	Number of Parameters	Comparison
CNAPs	600,000	Good performance, a Large number of parameters
CNP	16,961	Good performance, Small number of parameters
Hypernetwork	552,034	Bad performance, Large number of parameters

Table 1: Comparison between CNAPs for regression, CNP, and Hypernetworks based upon the number of parameters and performance in terms of MSE and mean STD.

As it can be observed from table 1, both the CNAPs and CNPs are giving comparable and good results and outperforming the hypernetworks. Though hypernetworks have a similar number of parameters as compared to CNAPs, CNAPs have delivered far better results. CNPs, on the other hand, have competed in terms of performance with CNAPs with a smaller number of parameters.



(a) Context size = 5 (b) Context size = 10 (c) Context size = 30

Figure 16: CNP: as the context size increases, the prediction improves, and MSE and mean STD reduce. The red vertical lines show standard deviations.

This can be related to how these different architectures represent and extract relevant information from the context set. Hypernetworks have no direct representation of the context set, unlike the other two architectures. This plays a role in their poor performance and flexibility to meta-learn unseen tasks.

0.4 CNAPs for Multi-task, Multi-class Classification

0.4.1 Stage One - Work on Initial Datasets

Experimental Setup

We kicked off the project by reproducing the results from the original formulation by setting up the repository on our local device. We formulated the results for MNIST, CIFAR10, and CIFAR100 as facilitated by the source code.

The experiments on MNIST, CIFAR10 and CIFAR100 had the following setup-

- Split the dataset into Train, Validation, and Split in the ratio 60:20:20.
- During training, formulate *training tasks*, which consist of context images with their labels and target images with hidden labels.
- Provide context images and their labels to the CNAPs model so that it adapts to the classes from the context set.
- Compute loss over the classification of target images and backpropagate this loss through the network.
- After training is complete, run inference on unseen test images according to the same formulation: division of test set into *testing tasks*, which consist of context and target images.

- Compute loss and accuracy over the classification of target images.

Observations

- CNAPs perform well on all three datasets.
- The model's test accuracy is decreasing in the order MNIST, CIFAR10, CIFAR100.
- The reason behind this is that MNIST and CIFAR10 have only 10 classes and 1000 images per class, While CIFAR100 has 100 classes and only 100 images per class.
- MNIST is comparatively easier to classify than CIFAR10 and CIFAR100 due to less complicated features.

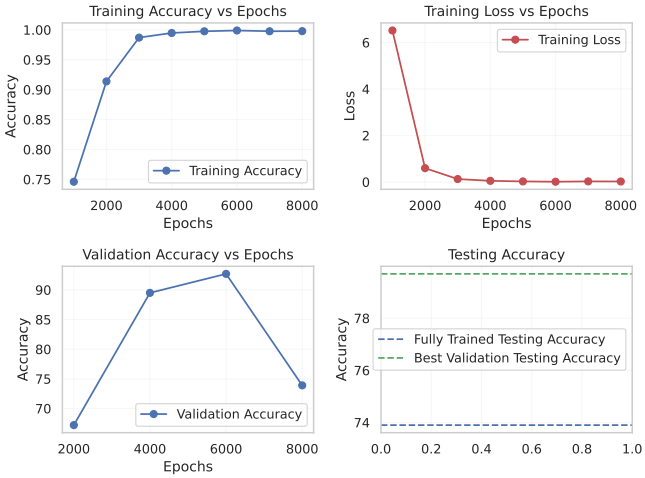
Inferences

- While CNAPs have proved to work well on multi-task, multi-class, and few-shot learning, their efficacy on more diverse datasets with more intricate training tasks was yet to be tested.
- The classes in MNIST are easy for a model like CNAPs to classify (training accuracy approaches 100 percent. The variation in features in different classes of CIFAR10 and CIFAR100 is very drastic. For example, CIFAR10 has the following classes- airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.
- The ability to correctly classify such separated classes does not test CNAPs ability to fine-grained learning from a few shots.
- A more complicated dataset is needed to experiment with CNAPs' ability to understand the specific features of similar classes from a few shots.

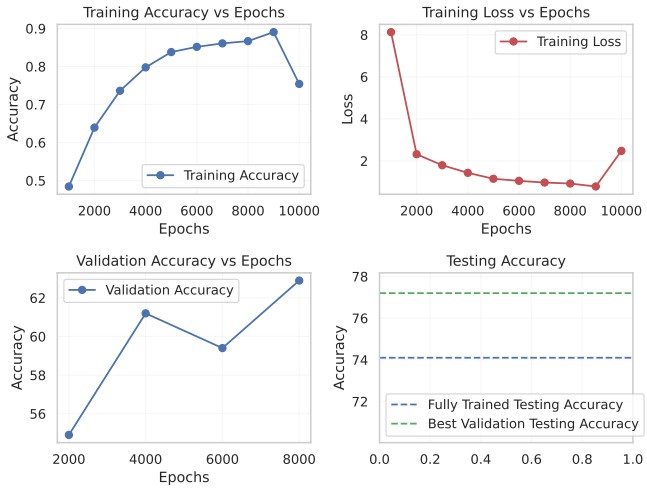
0.4.2 Stage Two - Working on CelebA Dataset

That's why we thought of implementing a new pipeline that supports new datasets. The CelebA dataset comprises over 200,000 images featuring the faces of 10,000 celebrities. It poses a challenging task with 40 facial attributes per image, including features like a *Big Nose*, *Bald*, and *Blond Hair*, among others. This dataset is not a straightforward multitask or multi-class classification problem, adding to its complexity and diversity. We formulated three experiments using this dataset- each having its own custom dataset derived from CelebA. In all the experiments on CNAPs, the context size was set to 1, and the target size was set to 10. The way (number of distinct classes in a particular *testing task*) was set to 5.

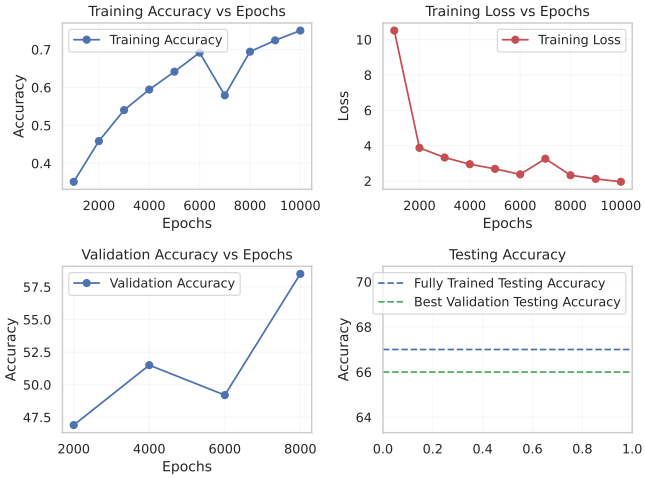
We also implemented a custom **FineTune** baseline. To compare the performance of CNAPs, we fine-tuned the CNN architecture as a baseline. We used the same dataset and same number of context points for both models. In the fine-tuning process, we did not change the ResNet block. We changed the linear layer depending on the number of classes in the task. For the context points, we backpropagated the loss only through the linear layer. In low data regimes, fine-tuning tends to overfit the training data. This results in low performance on the test data. We will see how the fine-tuning approach compares against CNAPs in the below experiments.



(a) Results on MNIST, with context size=1.

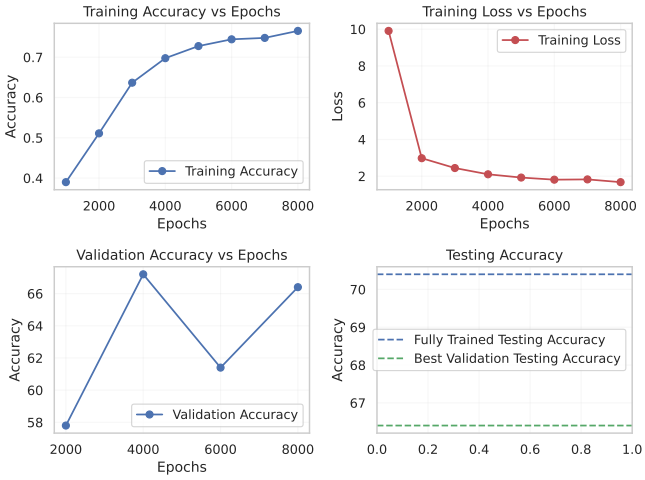


(b) Results on CIFAR10, with context size=1.

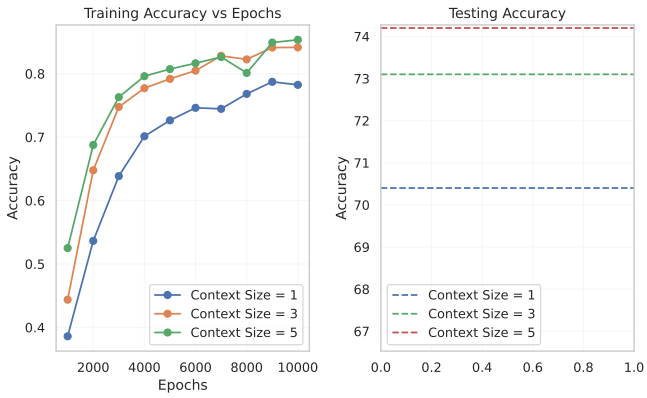


(c) Results on CIFAR10, with context size=1.

Figure 17: Initial Results



(a) Results on Celeb10, with context size=1.



(b) Results on Celeb10, with varying context sizes.

Figure 18: Results of CNAPs on Celeb10

Experiment One - Celeb10

Experimental Setup

Earlier experiments had proved that CNAPs works well when the images for different classes are quite distinctive (well-separated). The ability to differentiate between specific facial features is a more fine-grained task. This was the primary motivation behind curating **Celeb10**, and testing CNAPs' performance on Celeb10.

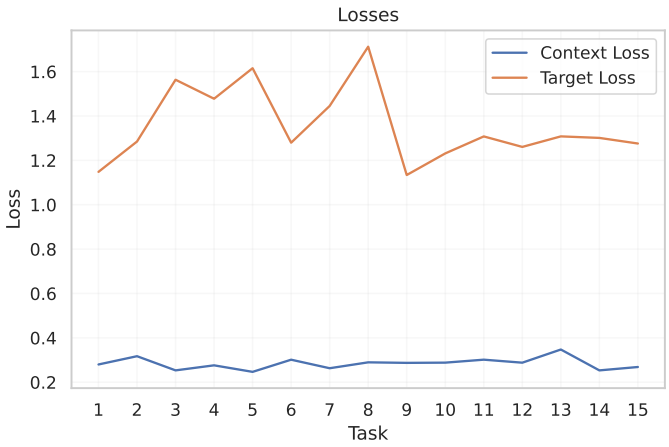
Celeb10 consists of 16,000 images- approximately 1600 images corresponding to 10 different classes. Each class corresponds to a distinctive facial feature, for example, Bald, Big Lips, Receding Hairline, Blond Hair, etc. This now becomes a 10-class dataset for CNAPs to be trained on.

We adopt a similar Train:Validation: Test split as described earlier. The evaluation and metric calculation strategy remains the same as well. In the training dataset, there are images labeled with attributes such as Bald, Big Lips, Receding Hairline, Wearing Hat, Brown Hair, and Pale Skin. On the other hand, the test and validation datasets consist of images labeled with the attributes Bangs, Blond Hair, Black Hair, and Eyeglasses.

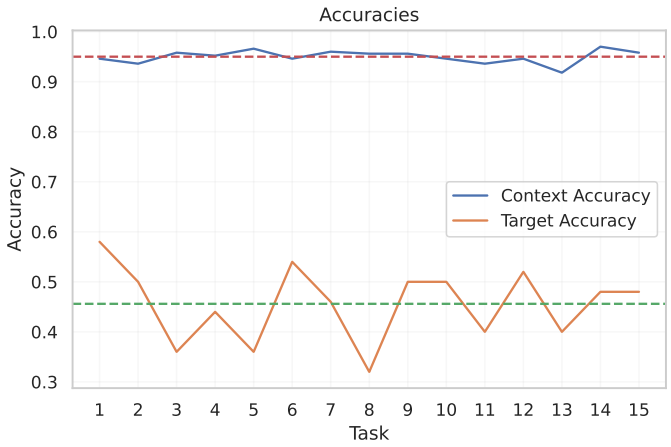
We also trained the fine-tuned models for 15 training tasks in order to measure the consistency of performance across tasks. Setting the context size equal to 1 led to extremely poor results; hence, we tried the experiment with 2 context images and 10 target images for **FineTune**.

Inferences

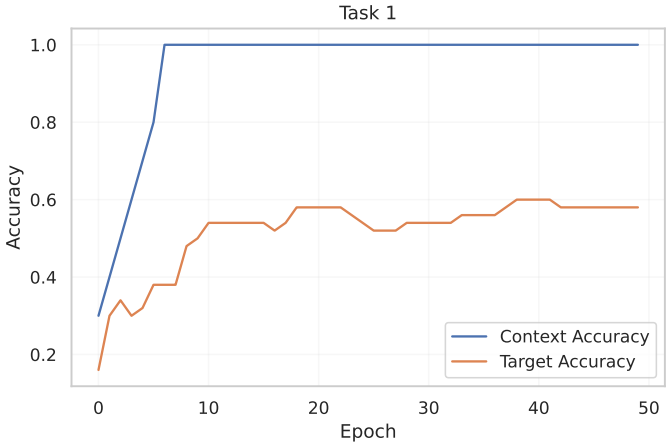
- The performance at test time with a single context image trends near an impressive 70 percent.
- Both training and testing accuracy increase with context size.
- Across 15 different tasks, **FineTune** consistently yields higher loss and lower accuracy values for the target set as compared to the context set.
- For individual tasks, we can observe that **FineTune** overfits on the context set and fails to generalize over the target set- which is evident in the saturating accuracies with epochs over these two sets.



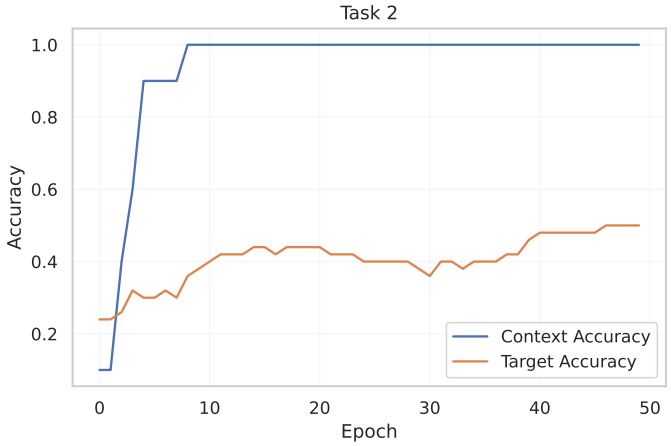
(a) Loss Comparison from FineTune on Celeb10, with context size=2.



(b) Accuracy Comparison from FineTune on Celeb10, with context size=2.

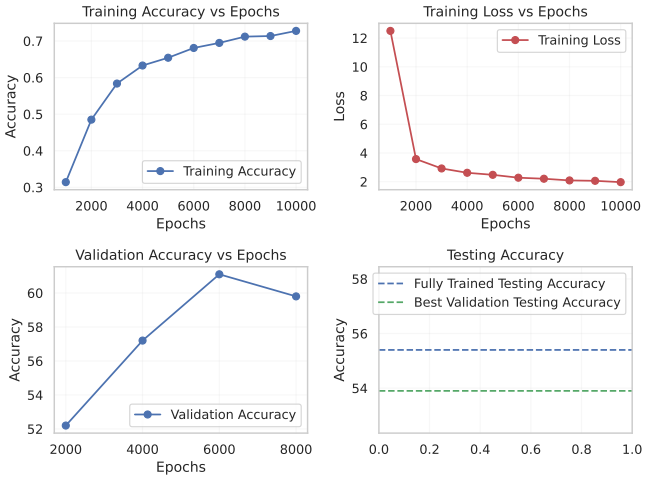


(c) Target vs Context Set Accuracy from FineTune on Celeb10, on Task 1

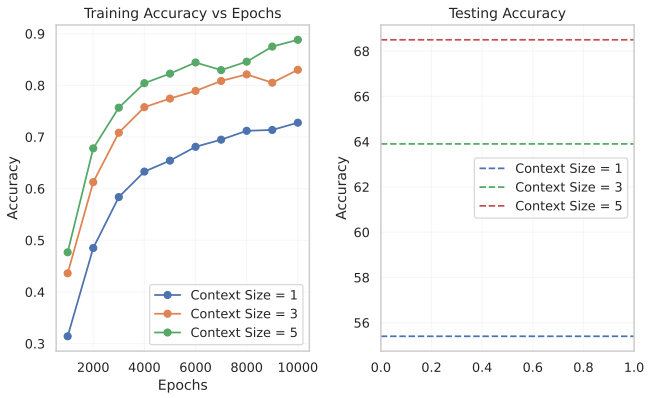


(d) Target vs Context Set Accuracy from FineTune on Celeb10, on Task 2

Figure 19: Results from FineTune on Celeb10



(a) Results on Celeb150, with context size=1.



(b) Results on Celeb150, with varying context sizes.

Figure 20: Results of CNAPs on Celeb150

Experiment Two - Celeb150

Experimental Setup

CNAPs prove to be good at classifying individual features (even unseen ones) from a few shots at test time. The next task we proposed was the classification of celebrities themselves. To this end, we curated another dataset, Celeb150- containing 30 images of 150 different celebrities.

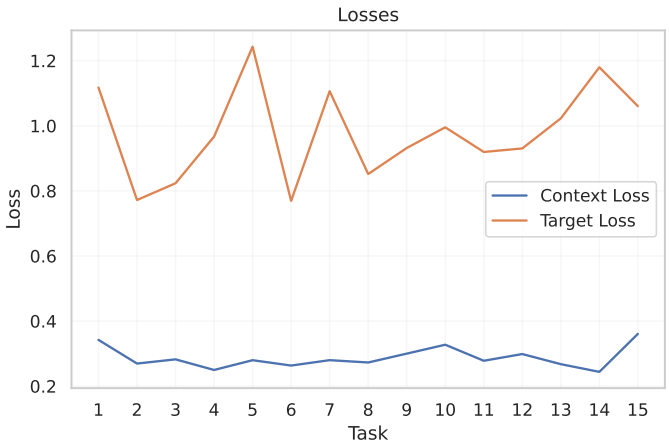
In order to perform well on this task, CNAPs would have to extract and learn from multiple facial features (and their relative proportions to each other), which will help uniquely identify a celebrity. All this from only 1 context image. Intuitively, this task seems more difficult than the first experiment.

The training set now contains 100 classes, and each class corresponds to images of a particular celebrity. Similarly, the testing and validation set contains 25 classes each. The split between context and target sets remains the same, along with metric calculations.

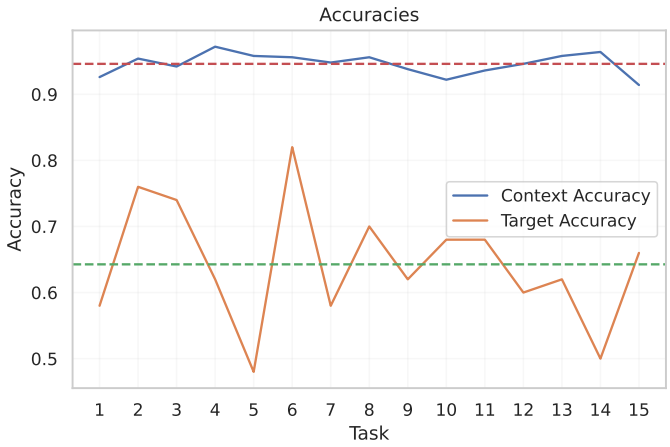
Fine-tuned models were also trained on 15 tasks, similar to the formulation in Celeb10.

Inferences

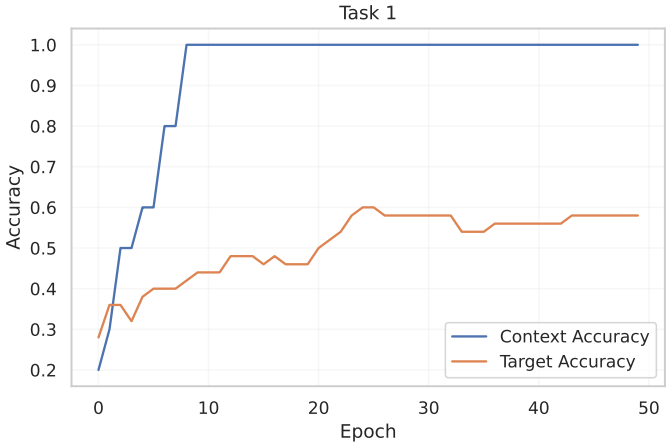
- As anticipated, the performance of CNAPs on Celeb150 is not as good as Celeb10. The accuracy drops, trending around 55 percent.
- Providing more context improves the performance of both training and testing data.
- Once again, across 15 different tasks, **FineTune** consistently yields higher loss and lower accuracy values for the target set as compared to the context set.
- For individual tasks, we can observe that **FineTune** overfits on the context set and fails to generalize over the target set- which is evident in the saturating accuracies with epochs over these two sets.



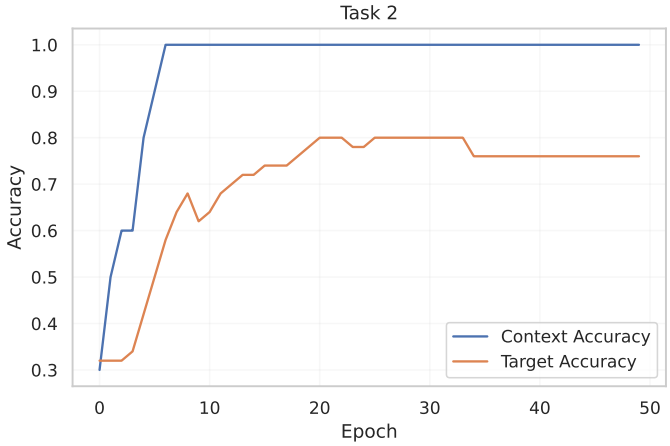
(a) Loss Comparison from FineTune on Celeb150, with context size=2.



(b) Accuracy Comparison from FineTune on Celeb150, with context size=2.

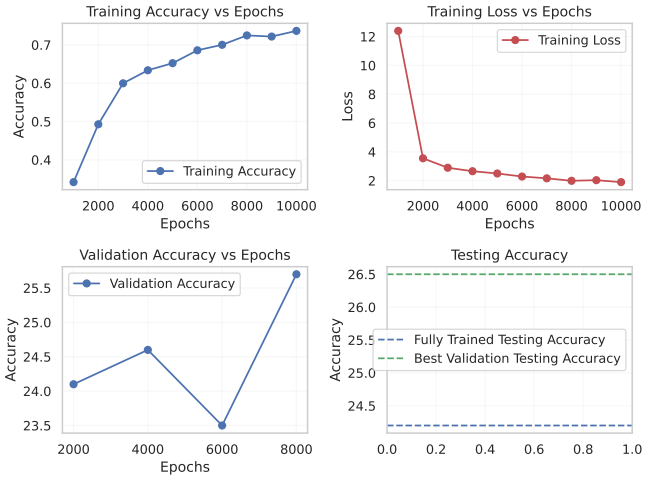


(c) Target vs Context Set Accuracy from FineTune on Celeb150, on Task 1

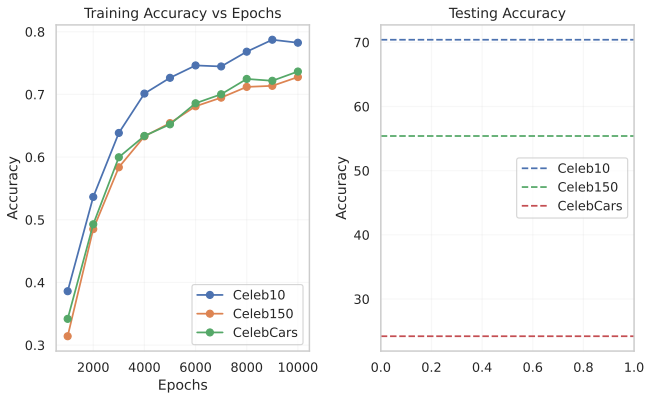


(d) Target vs Context Set Accuracy from FineTune on Celeb10, on Task 2

Figure 21: Results from FineTune on Celeb150



(a) Results on CelebCars, with context size=1.



(b) Results on differing Datasets.

Figure 22: Results of CNAPs on CelebCars

Experiment Three - CelebCars

Experimental Setup CNAPs prove to perform reasonably well on isolated facial features, while the performance drops on a more complicated task like celebrity recognition. In both these experiments, though, the train and test images were from the same underlying distribution/dataset.

This motivated us to formulate another experiment- where the distribution of test images is very different from that of the train images. Intuitively, CNAPs can generalize their learning from train images onto test images if they contain similar features/patterns. In order to test the performance of CNAPs when train and test distributions are very different, we curated another dataset- CelebCars. The training data remains the same as that of Celeb150. The test and validation set both contain 30 images each of 25 different car models.

The training procedure, testing procedure, and metric calculation remain the same as before.

Inferences

- The performance of CNAPs on CelebCars is extremely poor. While training accuracy is high, testing accuracy trends near 25 percent, barely above the random baseline of 20 percent.
- This proves that CNAPs cannot work well if the test images and train images are from completely different distributions.
- CNAPs' performance decreases in the order Celeb10, Celeb150, CelebCars due to an increasing level of complexity of tasks (Celeb10 vs Celeb150), as well as differing distributions of train and test data (in the case of CelebCars).

0.5 References

- Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D., Eslami, S. & Teh, Y. Neural Processes. (2018)

- Requeima, J., Gordon, J., Bronskill, J., Nowozin, S. & Turner, R. Fast and Flexible Multi-Task Classification Using Conditional Neural Adaptive Processes. (2020)
- Neural-Process-Family
- Cambridge mlg cnaps repository
- Chrhenning hypnettorch tutorials
- Meta Dataset by Google Research