CPEN 455 Final Project

Yi Fan (Ivan) Guo
65994287
CPEN 455 - Deep Learning
ivanguo@student.ubc.ca

Abstract

This report focuses on the transformation of an unconditional PixelCNN++ model into a conditional one, and the process in enhancing its classification accuracy. The project involved a systematic exploration of architectural modifications and training methodologies to harness the benefits of conditional modeling while preserving the model's generative capabilities. The experimental findings showcase excellent classification and generative capabilities, underscoring the importance of incorporating conditional information into generative models to offer valuable insights for researchers and practitioners seeking to leverage conditional modeling techniques for image classification tasks.

1 Model architecture

1.1 Adding positional embedding

Fusing the positional embedding with the input embedding is necessary in the transformation process from an unconditional to a conditional PixelCNN++ model. Adding positional embedding allows the preservation of spatial information which is necessary during the generation process. In image generation tasks, spatial relationships between pixels are crucial in capturing fine-grained details and coherent structures.

The model was changed to take two additional inputs: The number of unique class labels and the labels embedding themselves. The positional embedding layer was created with an embedding number equaling to the total number of classes, and the embedding dimension to be the number of input channels multiplied by the dimensions of the image size. This allows the layer to be compatible with the input data dimensions and append to the input data.

Adding positional embedding at the input stage was a design choice made with efficacy and straightforwardness in mind. Embedding at the input stage allows the conditional information to propagate through the entire network, leading to more contextually relevant outputs. This means that the model can adhere to the modified input and positional embedding data earlier on and thus produce higher quality results in classification and generation tasks. Another benefit is that embedding at the input stage simplifies the implementation and training process. There is no need for additional conditioning mechanisms or data tensors throughout the rest of the network that must adhere to certain shapes of the data at certain times, which makes the model much more straightforward, less complex, and easier to interpret than fusing the positional embedding in other places.

1.2 Creating inference functionality

Creating the ability for the PixelCNN++ model to infer or classify images is central in converting an unconditional model to a conditional one. The image inference function runs a forward pass through the model on the data with each class label to obtain the loss of each individual image for each label, then selects the label with the least loss to be the predicted label.

1.3 Training procedural changes

There were several modifications that required to the training code to differentiate between training an unconditional PixelCNN++ model and a conditional one. The first major change was the addition of inputting both the class labels and the number of classes to the model. This allows the model to be conditioned on class information. The second major change involves modifying the model to train, validate, and test on data correctly for each epoch, fully utilizing the inference functionality listed above for testing and validation. The third major change was adding a validation accuracy tracker that is logged on **Wandb** in order to better monitor model performance over time and assist in hyper-parameter tuning.

1.4 Evaluation Changes

There were minor changes that were necessary after the conversion of the model. During the evaluation of the generation process, the sampling function invokes the model to generate images. This sampling function was modified to pass labels into the model to be used in generation. During the evaluation of the classification process, the inference function needed to be run for each image to compare the inferred label to their actual label.

1.5 Modifying the loss function

The logistic loss function was modified to calculate the loss of individual images instead of mini-batches. This is necessary because we need individual image loss for classification to determine which label to predict. This also enhances the training process as evaluating the model's performance on individual images provides a clearer showcase of its performance rather than over entire mini-batches.

2 Experiments

2.1 Individual hyper-parameter trials

Over 30 trials were conducted on hyper-parameter tuning in an attempt to improve the conditional PixelCNN+++ model. Several experiments were conducted by changing only one hyperparameter at a time while setting all other hyperparameters as control parameters. This was done over 10 epochs for faster training times.

For the trials, *nr_logistic_mix* was set to 5, *lr_decay* was set to 0.999995, and *max_epochs* was set to 10. Note that these hyperparameters were untested due to several limitations, including insufficient GPU memory and a lack of time to test each and every hyper-parameter.

Below are the main hyperparameters tested:

batch_size: In this test, *sample_batch_size* was fixed to 16, *nr_resnet* was fixed to 1, and *nr_filters* was fixed to 40. It appears at lower epochs, that generally an increase in *batch_size* is positively correlated with the FID and BPD score, while accuracy seems to vary with batch size, peaking at a batch size of 8 and 50. See (Appendix 1).

sample_batch_size: In this test, *batch_size* was fixed to 16, *nr_resnet* was fixed to 1, and *nr_filters* was fixed to 40. It appears that BPD is relatively unaffected by the *sample_batch_size* difference, whereas the FID score is lowest at a *sample_batch_size* of 50, and increases everywhere else. Accuracy also seems to vary, peaking when *sample_batch_size* is 50 and 80. See (Appendix 2).

nr_filters: In this test, *sample_batch_size* was fixed to 16, *nr_resnet* was fixed to 1, and *batch_size* was fixed to 16. It appears that BPD is inversely correlated with *nr_filters*, while the FID score is lowest at 60 and increases everywhere else. Accuracy seems to peak with a *nr_filters* value of 120, then drops off afterwards. See (Appendix 3).

nr_resnet: In this test, *sample_batch_size* was fixed to 16, *batch_size* was fixed to 16, and *nr_filters* was fixed to 40. It appears that BPD is inversely correlated with *nr_resnet*, while the FID score is lowest at a *nr_resnet* value of 4, and increases everywhere else. Accuracy seems to improve up to a *nr_resnet* value of 4 as well, then drops off afterwards. See (Appendix 4).

2.2 Combination hyper-parameter trials

Several hyper-parameter combination tests were also performed, but increased the training time exponentially. As a result, higher values of each hyper-parameter were not tested due to time constraints. The full trial data can be located in **trials.xlsx** in the submission.

The sample size was ultimately selected to be tested extensively over multiple epochs as it seemingly had the largest impact on classification accuracy without increasing training time as much as the other hyper-parameters.

2.3 Final hyper-parameter values

The final selected model has the following hyper-parameter values:

- batch_size = 8
- sample_batch_size = 16
- nr_resnet = 1
- nr_filters = 40
- nr_logistic_mix = 5
- lr_decay = 0.999995
- max_epochs = 100

The **Wandb** graphs for the running of the model can be found in the appendix. See (Appendix 5) for training-Average-BPD, (Appendix 6) for test-Average-BPD, (Appendix 7) for val-Average BPD, (Appendix 8) for val-Accuracy, and (Appendix 9) for FID.

The metric scores of the model are as follows:

- Average FID score: 13.2937490232047
- test-Average-BPD: 4.72321
- training-Average-BPD: 4.89579
- val-Accuracy: 0.75391
- val-Average-BPD: 4.756
- Classification accuracy: 0.753371868978805

The model scored an f-1 score of 0.7810672793324676 and an accuracy of 0.7861271676300579 on the public split of the test data. To see the trial compared to the other trials, see "Trial 27" in **trials.xlsx** in the submission.

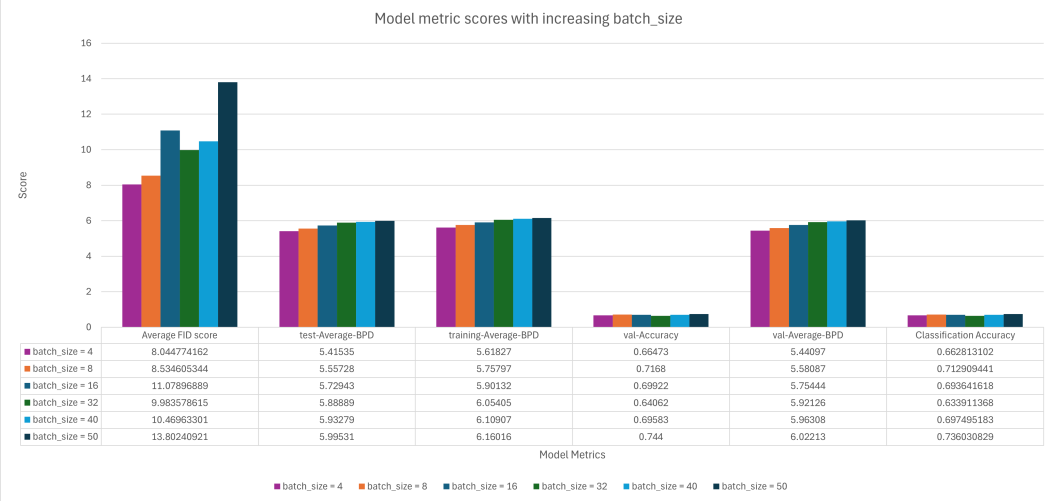
3 Conclusion

The key findings of the project was that BPD is inversely correlated with several hyper-parameters excluding sample batch size, while FID usually is the lowest at a certain hyper-parameter value, and higher everywhere else. Accuracy is also usually the highest at a certain point, while lower everywhere else. This is to be expected as we can assume that where accuracy is the highest is when the model is neither overfitting nor underfitting. Increasing the model complexity or runtime from that point can result in overfitting, while reducing the mode complexity or runtime can result in underfitting. This project contributed in transforming an unconditional PixelCNN++ model into a conditional one through positional embedding fusing, inference function, individual logistic loss calculation, and running the model on train/validation/test data with inference at the appropriate steps.

Some limitations of this project were time constraints and GPU constraints, as many hyper-parameter combinations or the utilization of hyper-parameter tuning algorithms like grid or random search were unable to be used due to the large amount of memory required to train an incredibly complex model over different hyper-parameters. Future avenues for improvement to this model could include a less naive process of fusing the positional embeddings with the model embeddings, such as within different stages of the model forward path rather than just the input embeddings. as well as training the model over more hyper-parameter combinations provided time and GPUs were not constraints.

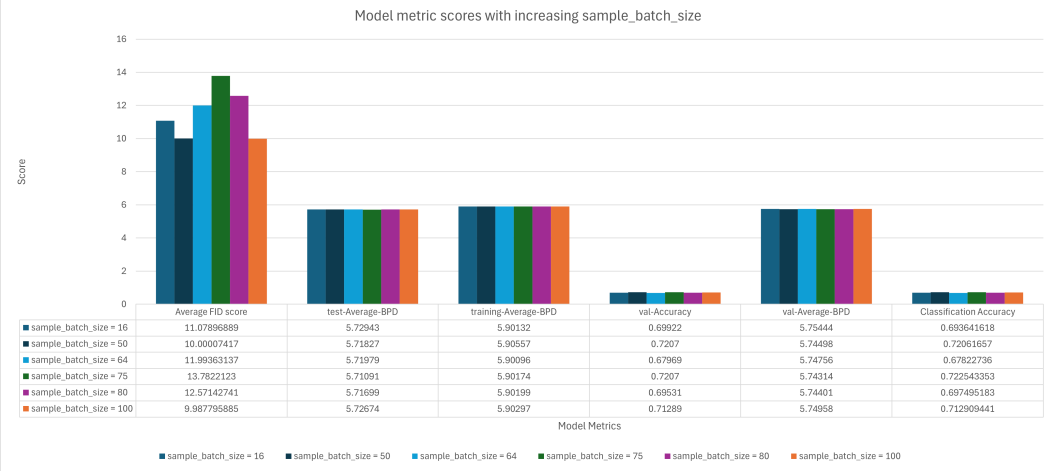
4 Appendix

Appendix 1:



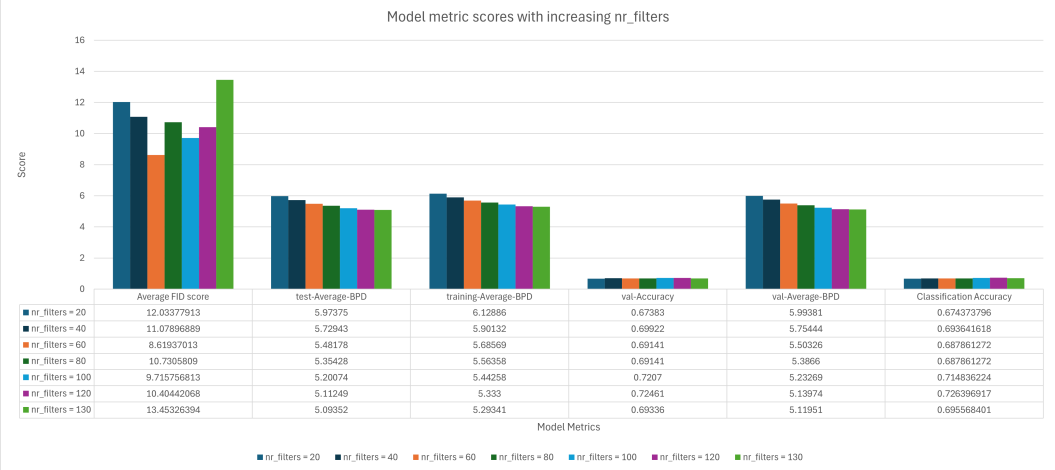
	Average FID score	test-Average-BPD	training-Average-BPD	val-Accuracy	val-Average-BPD	Classification Accuracy
batch_size = 4	8.044774162	5.41535	5.61827	0.66473	5.44097	0.662813102
batch_size = 8	8.534605344	5.55728	5.75797	0.7168	5.58087	0.712909441
batch_size = 16	11.07896889	5.72943	5.90132	0.69922	5.75444	0.693641618
batch_size = 32	9.983578615	5.88889	6.05405	0.64062	5.92126	0.633911368
batch_size = 40	10.46963301	5.93279	6.10907	0.69583	5.96308	0.697495183
batch_size = 50	13.80240921	5.99531	6.16016	0.744	6.02213	0.736030829

Appendix 2:



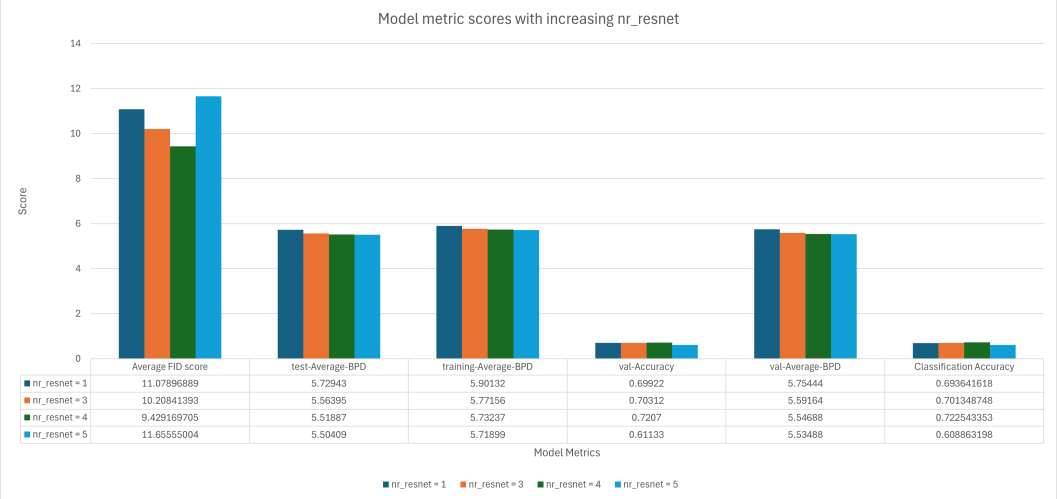
	Average FID score	test-Average-BPD	training-Average-BPD	val-Accuracy	val-Average-BPD	Classification Accuracy
sample_batch_size = 16	11.07896889	5.72943	5.90132	0.69922	5.75444	0.693641618
sample_batch_size = 50	10.00007417	5.71827	5.90557	0.7207	5.74498	0.72061657
sample_batch_size = 64	11.99363137	5.71979	5.90096	0.67969	5.74756	0.67822736
sample_batch_size = 75	13.7822123	5.71091	5.90174	0.7207	5.74314	0.722543353
sample_batch_size = 80	12.57142741	5.71699	5.90199	0.69531	5.74401	0.697495183
sample_batch_size = 100	9.987795885	5.72674	5.90297	0.71289	5.74958	0.712909441

Appendix 3:



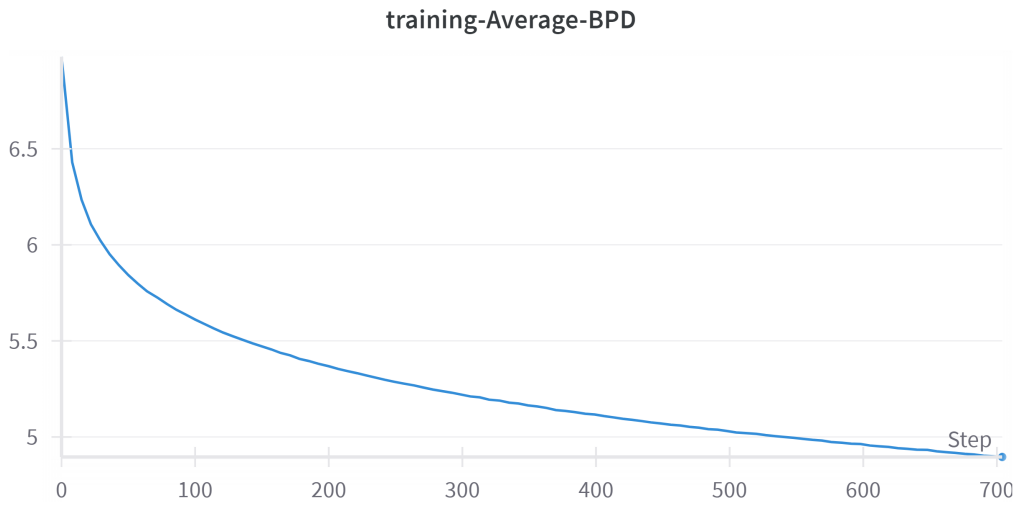
	Average FID score	test-Average-BPD	training-Average-BPD	val-Accuracy	val-Average-BPD	Classification Accuracy
nr_filters = 20	12.03377913	5.97375	6.12886	0.67383	5.99381	0.674373796
nr_filters = 40	11.07896889	5.72943	5.90132	0.69922	5.75444	0.693641618
nr_filters = 60	8.61937013	5.48178	5.68569	0.69141	5.50326	0.687861272
nr_filters = 80	10.7305809	5.35428	5.56358	0.69141	5.3866	0.687861272
nr_filters = 100	9.715756813	5.20074	5.44258	0.7207	5.23269	0.714836224
nr_filters = 120	10.40442068	5.11249	5.333	0.72461	5.13974	0.726396917
nr_filters = 130	13.45326394	5.09352	5.29341	0.69336	5.11951	0.695568401

Appendix 4:

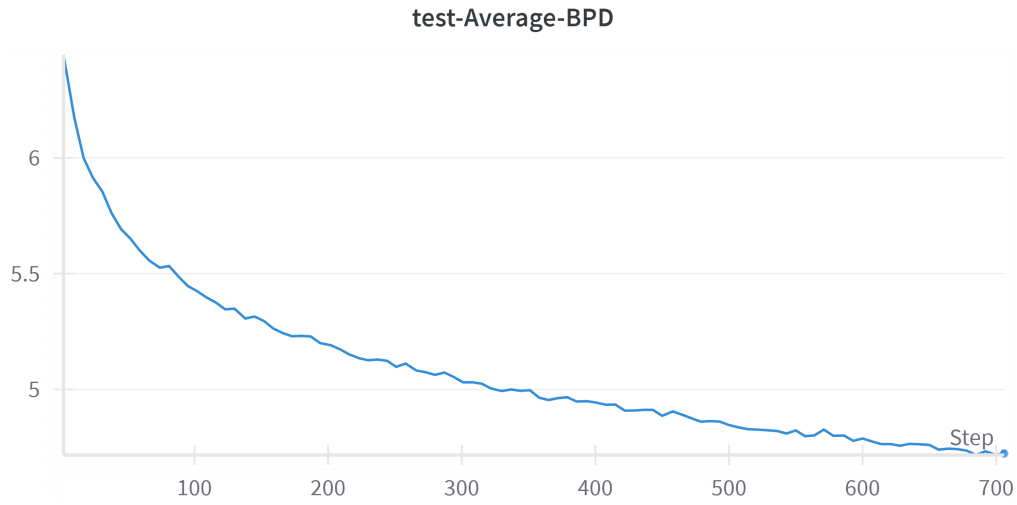


	Average FID score	test-Average-BPD	training-Average-BPD	val-Accuracy	val-Average-BPD	Classification Accuracy
nr_resnet = 1	11.07896889	5.72943	5.90132	0.69922	5.75444	0.693641618
nr_resnet = 3	10.20841393	5.56395	5.77156	0.70312	5.59164	0.701348748
nr_resnet = 4	9.429169705	5.51887	5.73237	0.7207	5.54688	0.722543353
nr_resnet = 5	11.65555004	5.50409	5.71899	0.61133	5.53488	0.608863198

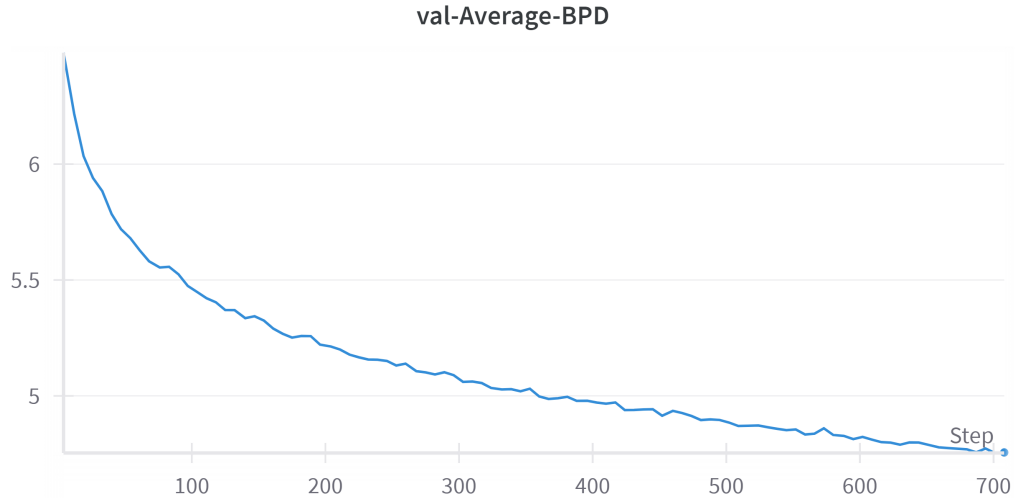
Appendix 5:



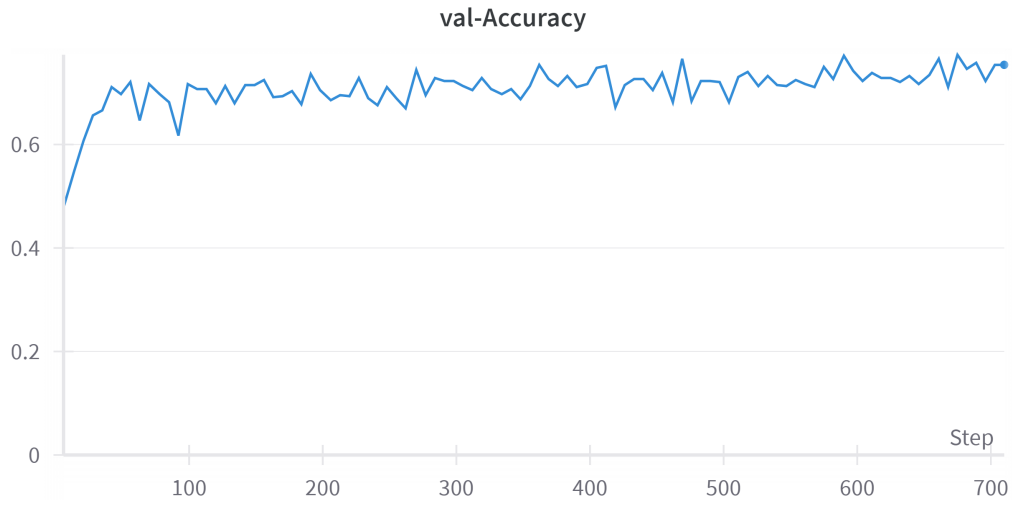
Appendix 6:



Appendix 7:



Appendix 8:



Appendix 9: