CPEN 455 Final Project Report Pixel CNN++ Conditional Classifier

Tayyib Chohan
Department of Electrical and Computer Engineering
University of British Columbia
tayyibchohan@gmail.com

1 Model

Pixel CNN++ is a conditional image generation model so the primary change to Pixel CNN is the ability to condition on images. To do this we can pass in label information to the model for it to condition on, in the form of a class embedding. I applied a variety of changes to the given Pixel CNN model to improve its performance on classification tasks. After experimenting heavily with embedding positions I used learnable class embedding at the center of the model as shown in the figure below. The embeddings were implemented with the nn.Embeddings module in pytorch. Next I applied data augmentation including random color jitter of the hue, saturation, brightness, and contrast by 10%. I also applied random horizontal and vertical flips to the data. I additionally changed the learning rate scheduler and optimizer to use AdamW with cosine annealing with warm restart. I was able to make the model large because I trained exclusively on an Nvidia RTX 4090 GPU. Further details about design choices are found in the experiment section. Overall I attempted to make the model large to induce over-fitting and attempted to incrementally reduce over-fitting afterwards [3]. The final model had a classification validation accuracy in the range of 82% to 89% with an FID somewhere in the 30s range.

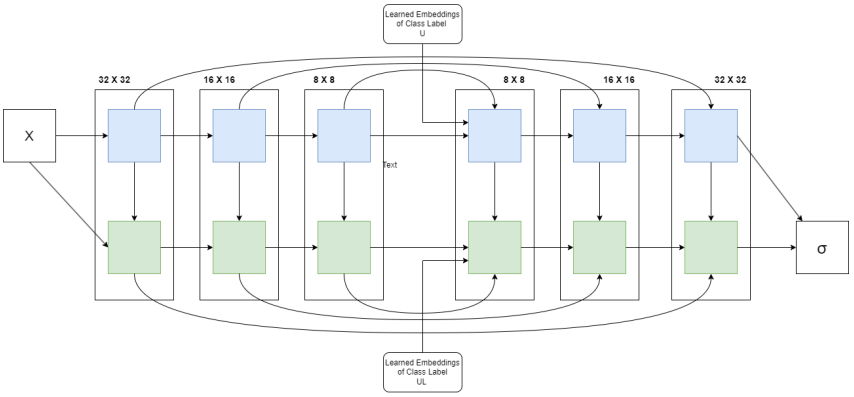

Figure 1: Final Pixel CNN++ Model Architecture

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

2 Experiment

2.1 Class Embedding

[1] [2] After reviewing the original Pixel CNN++ paper[6] I did not think it was worth exploring different types of embedding as the paper did not find a significant difference between absolute position encoding and learned parameters. Therefore I spent the majority of my time exploring embedding locations instead of ways to embed the class labels. I used learnable class embeddings through a pytorch nn.Embedding layer for all experiments in this section without data augmentation.

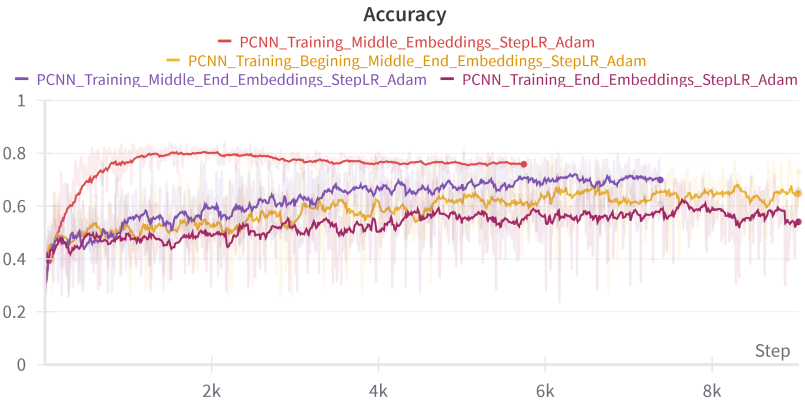

Figure 2: EMA of Classification Accuracy for different embedding positions

2.1.1 End Embedding

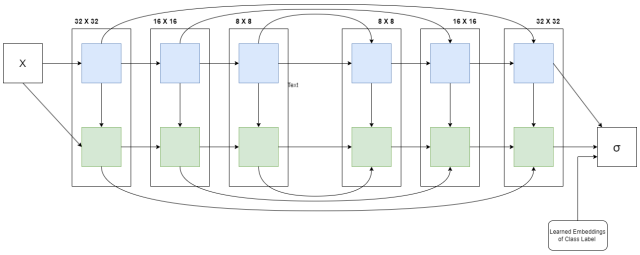

Figure 3: End Embedding Pixel CNN++ Model Architecture

This was the first embedding I attempted to implement. They performed poorly and I assume that the embedding likely did not encode very much information per pixel. This means that the label information may not have impacted the output logits heavily, which impacted the accuracy and lead to poor results. Refer to Figure 2 and appendix for further results A.

2.1.2 Middle End Embedding

I tried adding an embedding to the middle of the model because I thought the end embedding may not have enough influence over the model to affect the likelihood. This resulted in a significant improvement in performance. Refer to Figure 2 and appendix for further results A

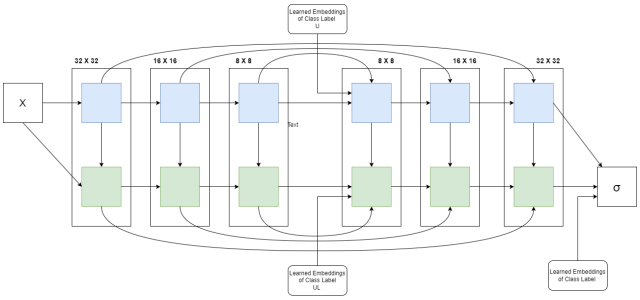

Figure 4: Middle End Pixel CNN++ Model Architecture

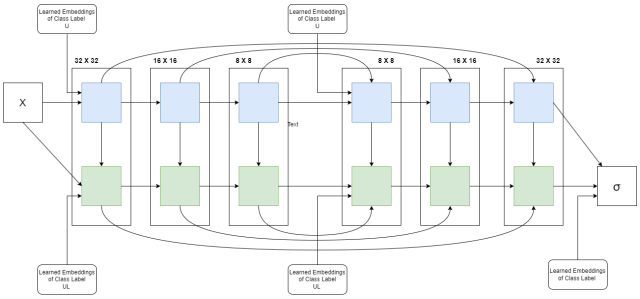

Figure 5: Beginning Middle End Pixel CNN++ Model Architecture

2.1.3 Beginning Middle End Embedding

I saw this as a logical extension of my previous experiment of middle and end embedding where I could influence the models likelihood even more with the class label. This unfortunately did worse than the middle and end embedding but better than just the end embedding. This led me to believe that it may be better to just encode the information in the middle of the model since adding another embedding may add unnecessary noise to the latent space. Refer to Figure 2 and appendix for further results A.

2.1.4 Middle embedding

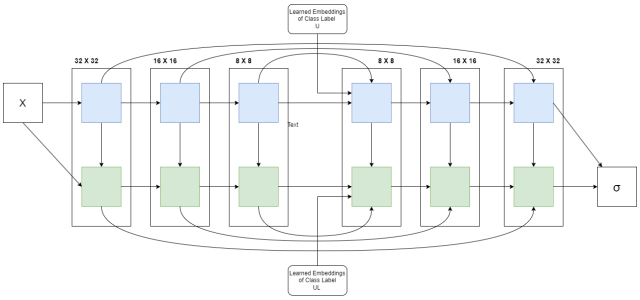

Figure 6: Middle Pixel CNN++ Model Architecture

This model performed significantly better than the previously attempted models but I noticed that the model may have been over fitting due to the behavior of the accuracy chart. After the validation accuracy peaked around 2k steps it began to slowly decrease. Nonetheless it outperformed the previous models considerably with an accuracy close to 80%. Refer to Figure 2 and appendix for further results A

[1]As shown in the diagrams the models use separate embedding layers for the U and UL path of the models respectively

[2]All accuracy charts in this paper depict validation set accuracy

2.2 NR Filters

I ran trials with varying nr_filters and determined that increasing the value resulted in an equal or better result so on every run I increased this value to the maximum that would run without failing due to a lack of VRAM.

2.3 Data Augmentation

I noticed signs of over fitting so I applied a random rotation of 10 degrees, random horizontal and vertical flips, and random color jitter. This gave me disappointing results which didn't make sense to me so I re-ran the training without the random rotation and I exceeded my previous results. I believe the random rotation deteriorated the images too much due to their small size so the quality of the data was worsened. When I removed the rotation I saw a significant improvement and the model reached a new peak accuracy. The results of this experiment can be found in the appendix A.

2.4 Learning Rate Scheduler and AdamW

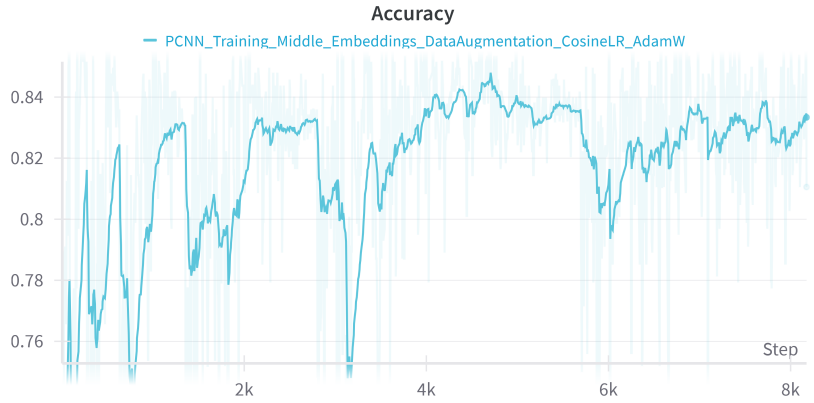

Figure 7: Middle Pixel CNN++ Model Architecture

I also explored a cosine annealing learning rate scheduler and AdamW on my best pre-trained model. Cosine annealing [7] uses a learning rate described with the following equation.

$$\eta_t = \eta_{\text{min}} + \frac{1}{2}(\eta_{\text{max}} - \eta_{\text{min}}) \times \left(1 + \cos\left(\frac{T_{\text{cur}}}{T_{\text{max}}}\pi\right)\right)$$

The reason for using this new LR function on my pre-trained weights was to attempt to explore new global minima. Unfortunately I found this to have similar accuracy to previous runs but all BPD measurements decreased further than before re-training. This is difficult to separate from the effects of adding weight decay so I'm uncertain how much these two changes contributed to the slight improvement in results. See figure 7 and relevant graphs in the appendix A.

3 Conclusion

I learned that in Pixel CNN++ class embedding placement has an extremely important effect on the result of the accuracy. Additionally I saw that data quality and quantity both positively contributed to the outcomes. The model used a limited dataset further work could be done with a larger dataset or additional augmentation techniques[2]. With more time I could have done a more controlled study to isolate effects of more variables and make a confusion matrix or do other types of analysis[3]. This could have led to further hyper-parameter tuning that could have further improved performance[1]. I may have also been able to explore more complex techniques for class embedding such as a deep network applied to the embedding or some other large architectural changes. I could have also adjusted the loss function to better account for classifications[3].

References

[1] https://github.com/google-research/tuning_playbook

[2] https://www.freecodecamp.org/news/improve-image-recognition-model-accuracy-with-these-hacks/

[3] https://neptune.ai/blog/iHowtoImprovetheAccuracyofYourImageRecognitionModelsmage-classification-tips-and-

[4] https://towardsdatascience.com/improve-image-classification-robustness-with-augmix-59e5d6436255

[5] https://neptune.ai/blog/how-to-choose-a-learning-rate-scheduler

[6] https://arxiv.org/pdf/1606.05328.pdf

[7] https://arxiv.org/pdf/1608.03983v5.pdf

A Appendix

A.1 Source Code

https://github.com/TayyibChohan/CPEN455HW-2023W2

A.2 Training Data

https://api.wandb.ai/links/ubccpentayyib/yph05puz

A.3 Additional Figures

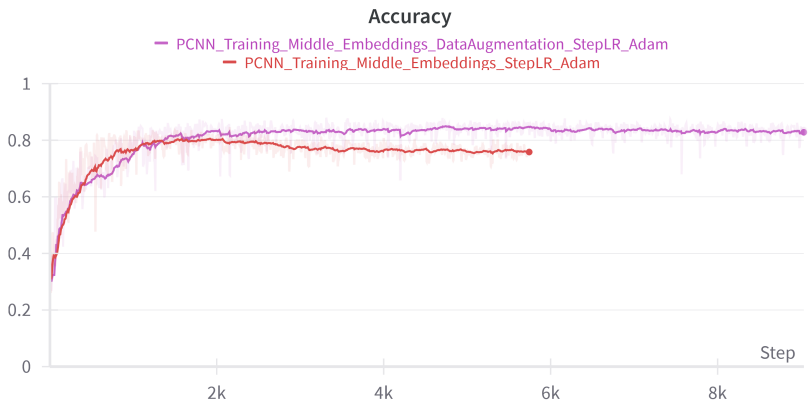

Figure 8: Graph showcasing the effects of data augmentation on accuracy

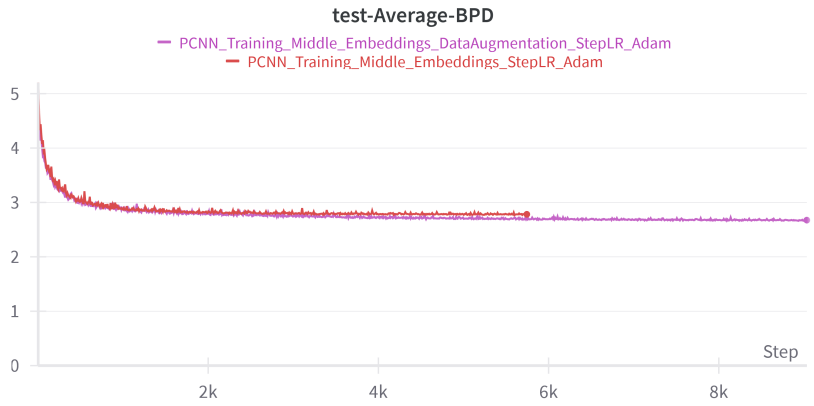

Figure 9: Graph showcasing the effects of data augmentation on test error notice how there is a divergence during the plateau around 4k steps

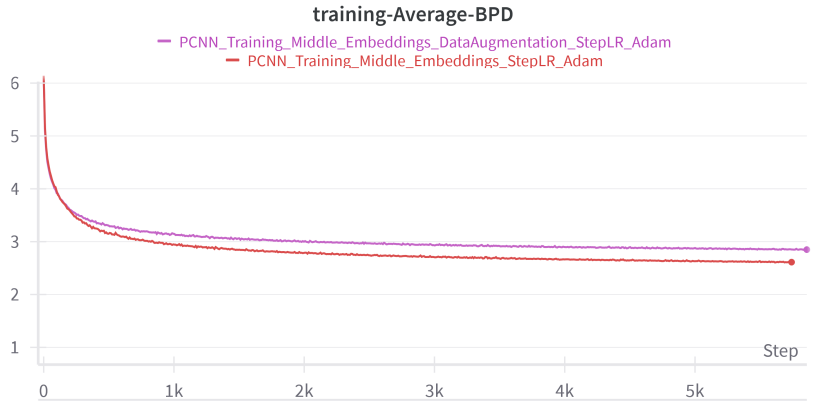

Figure 10: Graph showcasing the effects of data augmentation on train error notice how that despite the test error being better the train error is worse showcasing it as an imperfect approximation of the test error

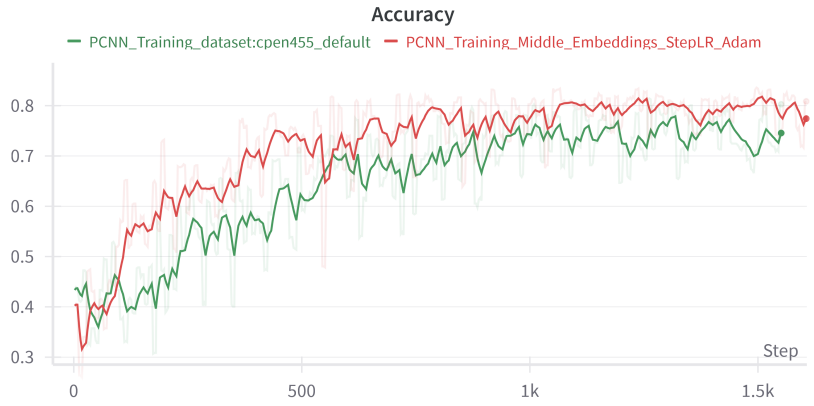

Figure 11: Graph showcasing the effects on accuracy of over augmenting the data with augments that erode data quality

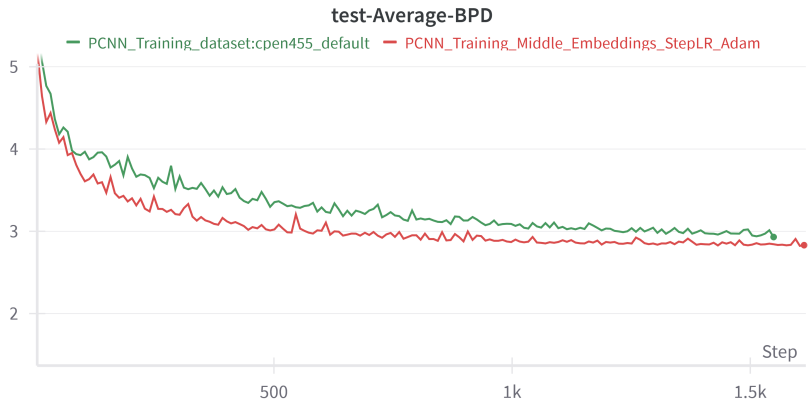

Figure 12: Graph showcasing the effects on test BPD of over augmenting the data with augments that erode data quality

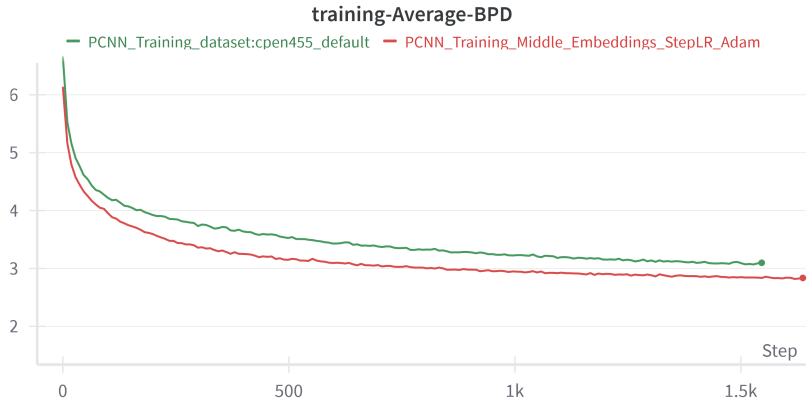

Figure 13: Graph showcasing the effects on train BPD of over augmenting the data with augments that erode data quality

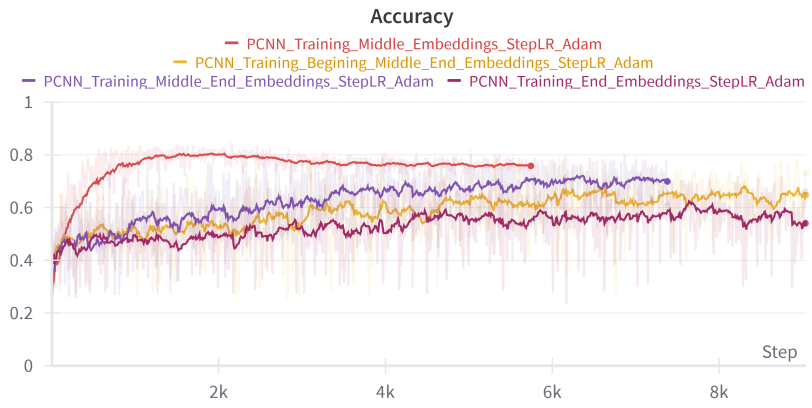

Figure 14: Graph showcasing the validation accuracy of different embedding locations

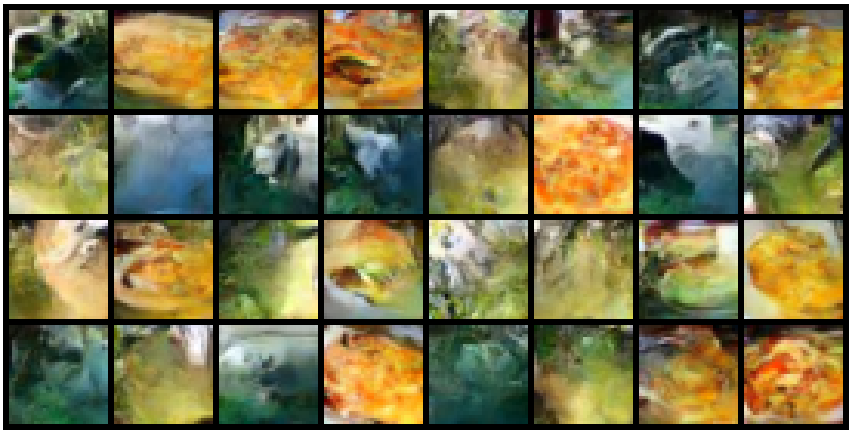

Figure 15: Middle End Pixel CNN++ Model Architecture

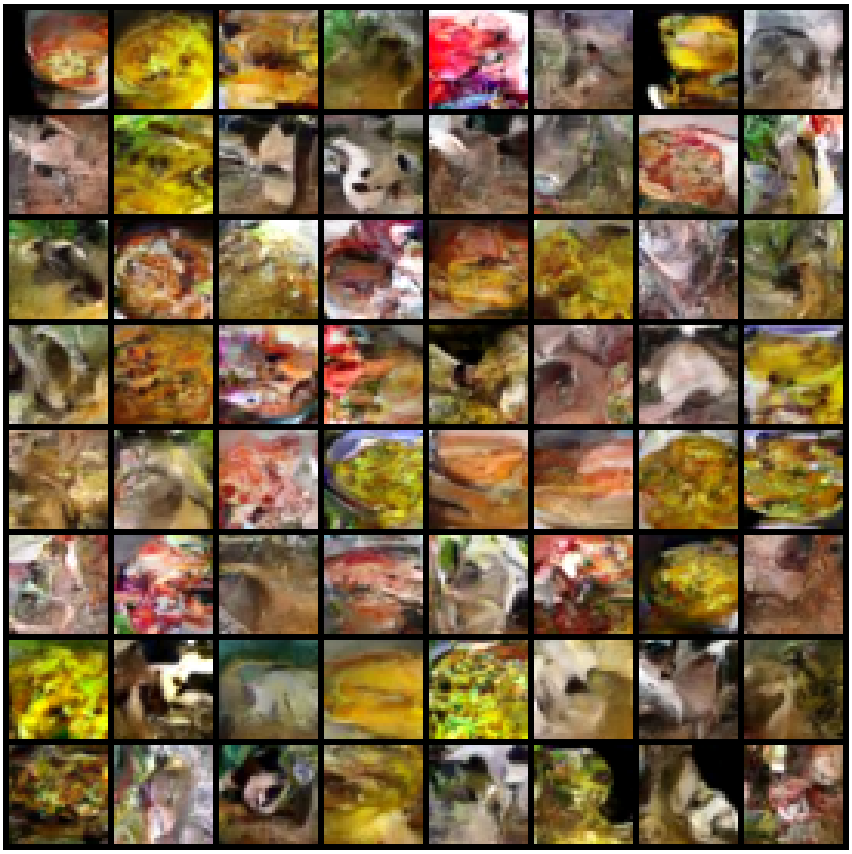

Figure 16: Samples from Final Model tuning (Cosine Annealing AdamW with Data Augmentation and Middle Embedding)

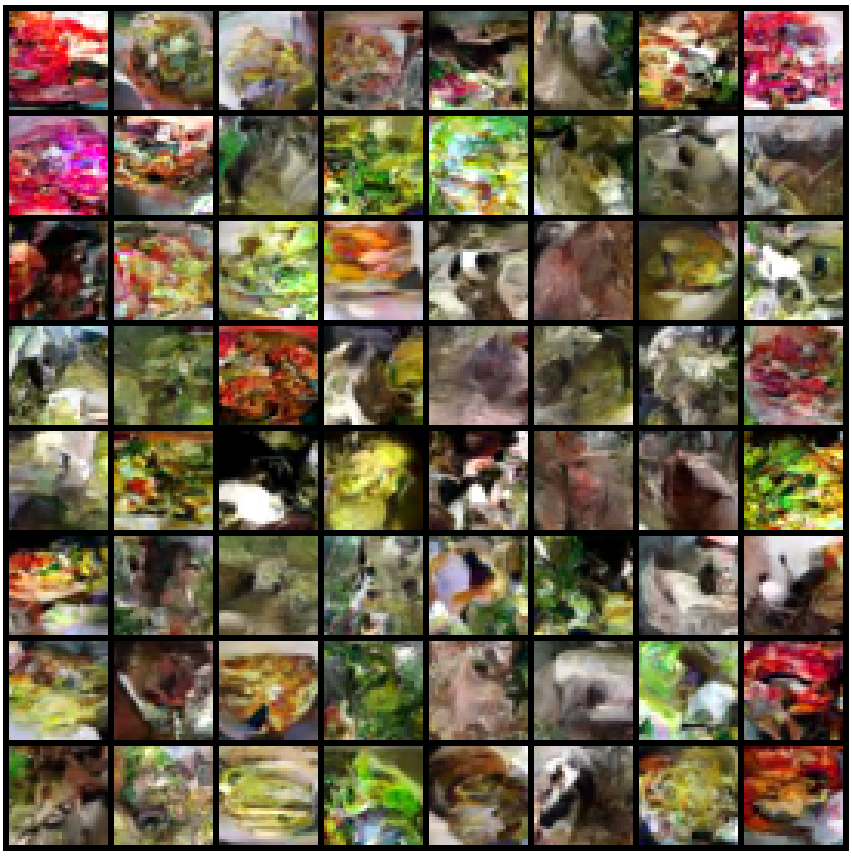

Figure 17: Samples from middle embedding with data augmentations

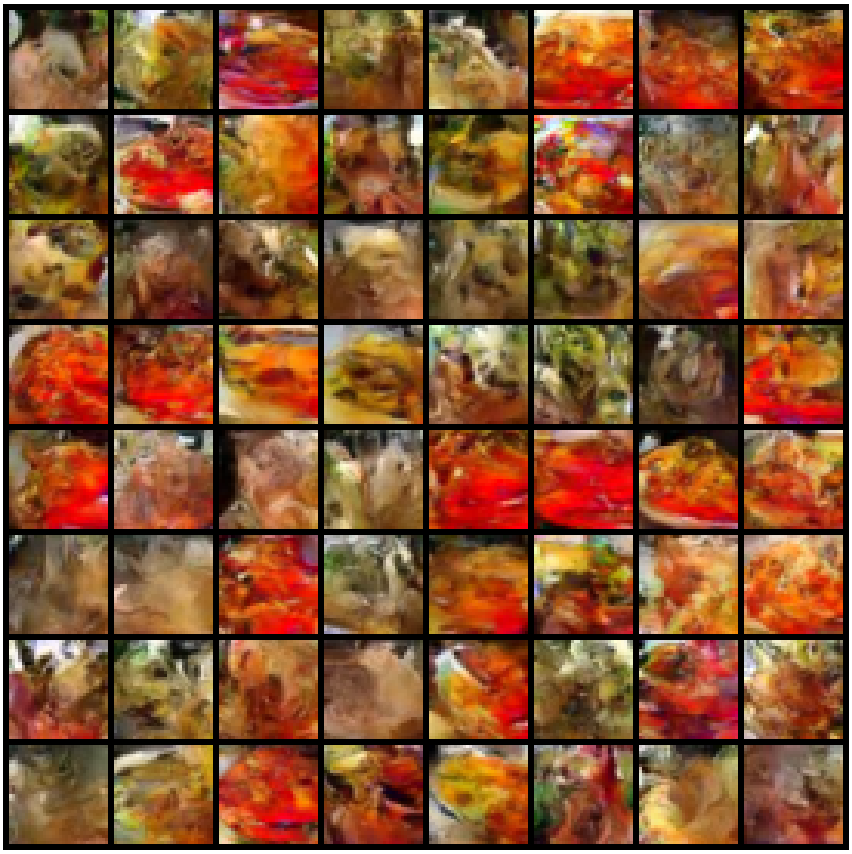

Figure 18: Samples from middle embedding

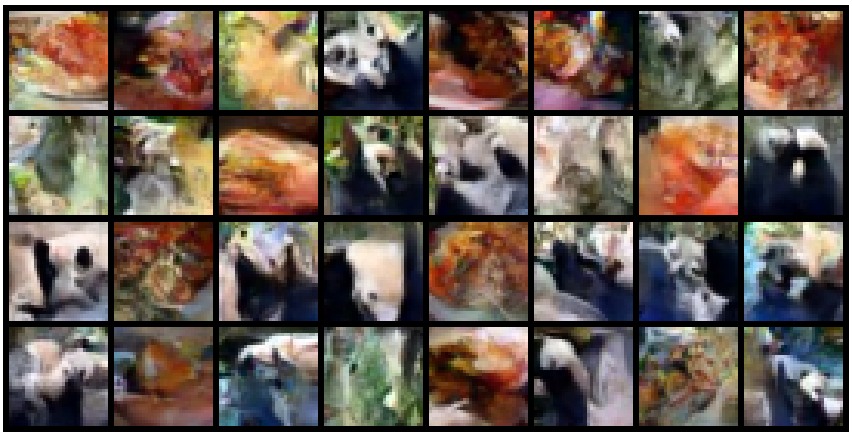

Figure 19: Samples from beginning middle end embedding

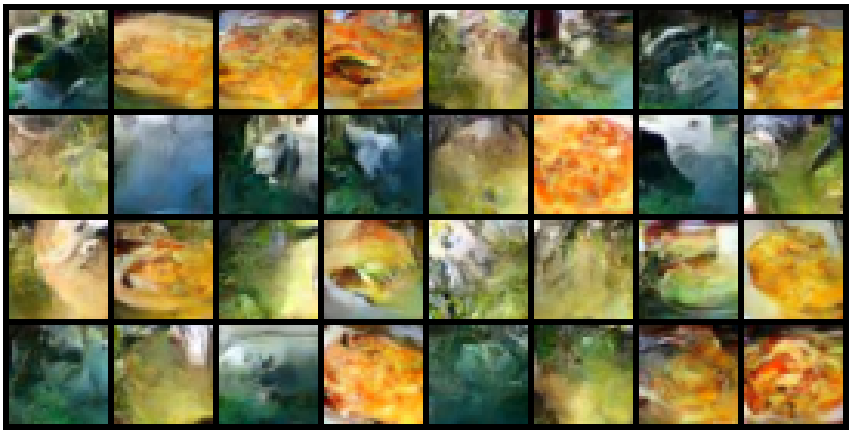

Figure 20: Samples from middle end embedding

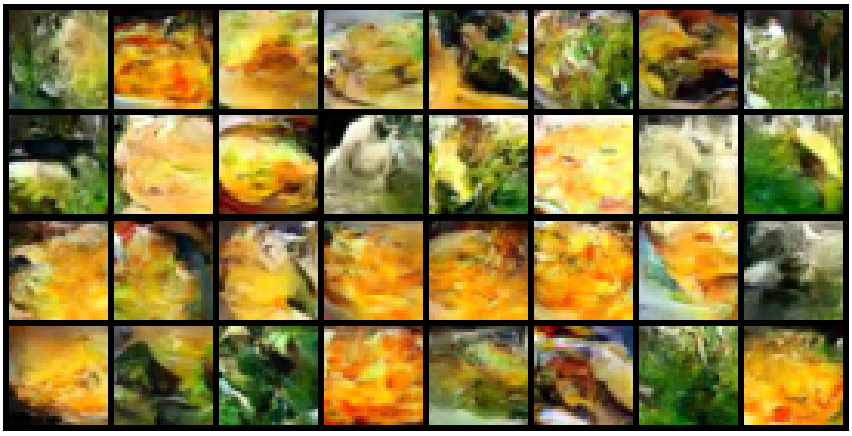

Figure 21: Samples from end embedding