Conditional PixelCNN++

Anonymous Author(s)
Affiliation
Address
email

Abstract

I present a conditional PixelCNN++, adapted from a non-conditional PixelCNN++ that uses discretized mixture of logistics, gated residual networks, and masked convolutions. This conditionality is introduced through a combination of early and fusion and residual block fusion, and achieves classification accuracy of 72% and FID score of 32.

1 Model

1.1 Baseline PixelCNN++ Recap

The baseline PixelCNN++ is an autoregressive model where the distribution of an image is the product of the distributions of pixels that came before. Each pixel has its distribution modelled with a discretized mixture of logistics, as shown by the formula below.

$$p(\mathbf{x}_i \mid \mathbf{x}_{<i}) = \sum_{k=1}^{K} \pi_k \,\mathrm{Logistic}\big(\mathbf{x}_i;\, \mu_k,\, s_k\big)$$

In order to only process preceding pixels, masked convolution with down and down-right kernels are used. This allows the model to not use explicit masking. Another key feature is the gated residual blocks used.

The loss function used for both the baseline PixelCNN++ and the new conditional version is as shown:

$$\mathcal{L}_{\text{NLL}} = -\frac{1}{N} \sum_{n=1}^{N} \sum_{i} \log p_\theta\big(\mathbf{x}_{n,i} \mid \mathbf{x}_{n,<i}\big)$$

1.2 Conditional Encoding

For my model, classes are embedded via a combination of early fusion and gated resnet fusion.

1.2.1 Early Fusion

Early fusion is used as a method to embed some class information into the input, and has negligible run-time and memory overhead. It's added by doing:

$$\mathbf{x}_{\mathbf{new}} = \mathbf{x} + \big(\mathbf{z}_c \otimes \mathbf{1}_{H\times W}\big)$$

1.2.2 Gated Residual Block Fusion

The next fusion integration is in the gated resnets, which allow for deeper feature integration. It's implemented by using FiLM, which learns a small MLP for each gated resnet, which outputs the β and γ used in the following:

$$h' = \gamma(\mathbf{c}) \odot h + \beta(\mathbf{c})$$

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

1.3 Classification

While the conditional PixelCNN++ model is a generative model, since it is fully tractable, it can also be used as a classifier. This is done by finding the likelihood of the model having generated an image embedded with a class c. Then, we pick the highest probability class as the classification result.

$$class \ = \ \arg\max_{c \in \mathcal{C}} \log p_\theta(\mathbf{x} \mid c).$$

Note that this formula requires a balanced class distribution. I made an assumption that the cpen455 dataset is balanced.

2 Experiments

2.1 Datasets

The model was trained and tested on the cpen455 dataset, though it can be adapted to use the mnist and cifar datasets instead.

The cpen455 dataset consists of 32x32 images of 4 classes: Hamburger, Panda, Koala, and Pizza.

2.2 Evaluation Metrics

The key evaluation metrics used are classification accuracy and FID score, with bits-per-dimension as a supplementary metric. The desired values for accuracy and FID are 75% and 30 respectively.

2.3 Key Model Choices

2.3.1 Architecture

I did not test many changes to the architecture, and so will not be including an ablation study. I thought that adding middle or late fusion may not be all too useful as I already included early fusion. The only changes I tested were the the complexity of the MLPs used in the resnet fusion. Initially, I had MLPs with two linear layers and a tanh non-linearity in between. I decreased this to just one linear layer later on, and saw no decrease in performance, so I kept it as the simpler version.

2.3.2 Hyperparameters

The key changes made during experimentation were on hyperparameters. The main hyperparameters were: nr_resnets, nr_filters, nr_logistic_mix, and batch_size.

2.4 Main Results

Here are some of the model results with varying hyperparameters.

2.4.1 Final Model

The final model used hyperparameters of nr_resnets = 5, nr_filters = 108, nr_logistic_mix = 10, and batch_size = 32, and ran for 300 epochs

On huggingface: it scored Accuracy: 72.20% and F1: 72.24%.

The Epoch and validation accuracy scores are shown in Figure 1. FID scores vs steps are shown in Figure 2.

It's interesting that the accuracy fluctuates so intensely across different amounts of training. We see the same for FID scores.

Some generated samples are shown in figure 3. It can be seen that class 0 (burger) is generated with a good amount of detail. The other classes are a bit lacking, however.

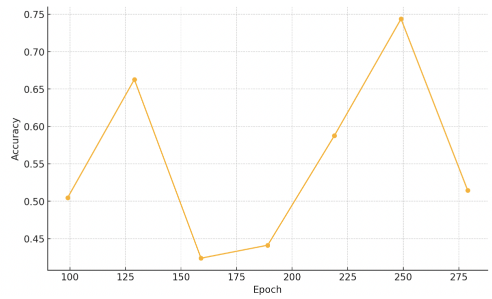

Figure 1: Final Model Epoch vs Accuracy

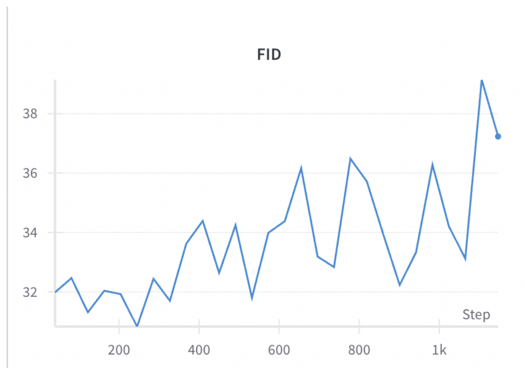

Figure 2: Final Model FID

2.4.2 Other models tested

At the beginning of training, I ran models with a RTX 2070 locally. With 8GB, I had to be selective with the specs I chose.

Some results:

First conditional model

epochs = 100 nr_resnet = 2 nr_filters = 64, nr_logistic_mix = 5, and batch_size = 32. Accuracy: 0.27167630057803466 Average fid score: 34.80098920866383

Third conditional model: epochs = 300 nr_resnet = 3 nr_filters = 140, nr_logistic_mix = 16, and batch_size = 32.

For this, I looked at TA Qi's WandB results, and saw that he used 1 resnet and 160 filters to get really good results. Unfortunately, it did not work for me.

The highest validation accuracy across 30 saved models (1 per 10 epochs) was 62%, shown in figure 5.

Other models were run, but none did nearly as well as the Final Model. They all had 50% accuracy, from 100 epochs till 300 epochs.

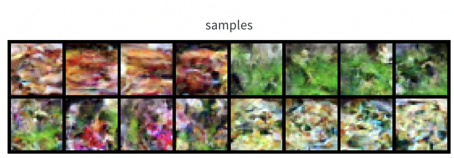

Figure 3: Final Model samples

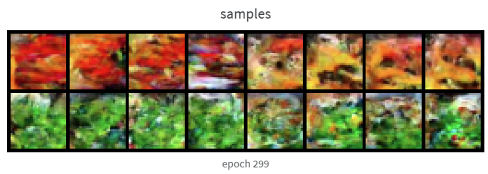

Figure 4: Other samples

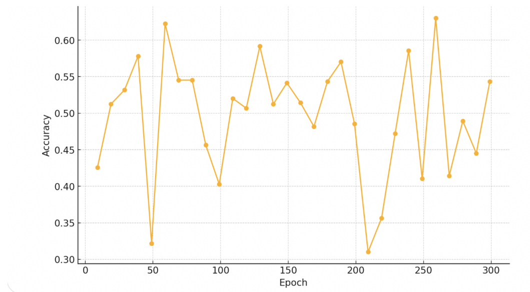

Figure 5: Fifth model accuracy

2.4.3 FID Scores Decreasing with Epochs

A key observation during experimentation was the relationship between increasing training epochs with FID score. For various models trained, FID score always tended to increase with more training.

3 Conclusion

3.1 Key Results

The model was successfully transformed into a conditional model.

For Epoch 249,

Accuracy: 0.74373795761079 FID: 38 Submission successful! Accuracy: 72.20%, F1: 72.24%

3.2 Limitations

The accuracy is worse than other classification methods like regular CNNs, and the FID could be improved. Also, this model has a very slow generation speed as it does it pixel by pixel.

3.3 Next Steps

A large limiter for this project was the physical specs available. With more memory and more time, it could have better performance. There are also things to test with optimizers and learning rates.