PixelCNN++ Project

UBC CPEN 455 - Deep Learning

Idil Bil (21344189)

April 15, 2025

Table of Contents

1. Model

1.1 PixelCNN++ Description

PixelCNN++ is an improved autoregressive generative model for images, building on the original PixelCNN. It models the joint distribution of an image as a product of conditional pixel distributions, generating each pixel based on previously generated ones in a fixed order.

The core model used in this project is PixelCNN++. The architecture is composed of two main components: an upper layer that captures information from the top and left regions of the image and a lower layer that integrates context from deeper spatial levels. Class labels are embedded and incorporated into the network during training to enable conditional generation. This allows the model to generate images corresponding to specific categories. In my implementation, the model produces 25 samples for each class: Class 0, 1, 2 and 3.

1.2 Formula

PixelCNN++ models the joint distribution of image pixels in an autoregressive and class-conditional manner. Each pixel is generated one at a time, with the probability of each pixel depending on the values of all previous pixels and an external label. This is represented mathematically as [1]:

$$p(X|Y) = \prod_{i=1}^{N} p(x_i | x_{<i}, Y)$$

- X: Input image
- Y: Conditioning label (class information)
- $x_{i.}$: Value of the i-th pixel
- N: Total number of pixels

To compute these conditional distributions, PixelCNN++ uses convolutional neural network layers, which process the image through local receptive fields. The input to each layer is a 3D tensor with dimensions corresponding to image height, width and depth (channels). Convolutional layers apply learned kernels W^k and biases b^k to produce feature maps h^k, described by the formula [2]:

$$h^k = f(W^k * x + b^k)$$

- f: Non-linear activation function
- x: Input feature map
- *: Convolution operation

This operation is repeated through layers of the network, with the class conditioning information Y injected into the architecture, allowing the model to adjust its predictions based on class-specific context. The combined formulation enables PixelCNN++ to generate diverse and coherent images conditioned on labels, as well as evaluate likelihoods for classification.

1.3 Forward Pass Algorithm

The forward pass in PixelCNN++ starts with an empty image tensor, which is gradually filled by generating one pixel at a time in a fixed, top-to-bottom, left-to-right order. At each step, the model looks at the already generated part of the image and uses masked convolutional layers to predict the distribution for the next pixel. The class label is embedded and added to the model to guide it toward the correct output category.

The image then flows through a stack of gated residual blocks, which help the model learn non-linear patterns and capture relationships between pixels. Skip connections between layers help preserve important information and improve learning. The final output consists of parameters for a mixture of logistics, which define the pixel's predicted value. This step is repeated for every pixel until the full image is complete.

The forward pass enables the model to generate high-fidelity, class-specific samples while maintaining autoregressive consistency across pixels.

2. Experiments

2.1 Training Setup

Weights & Biases (wandb) was used to monitor the training progress, allowing real-time tracking of both the Bits Per Dimension (BPD) and Fréchet Inception Distance (FID) scores. These metrics offered key insights into the model's learning behavior across epochs. BPD was used as a proxy for likelihood estimation, while FID provided a quantitative measure of the generated image quality.

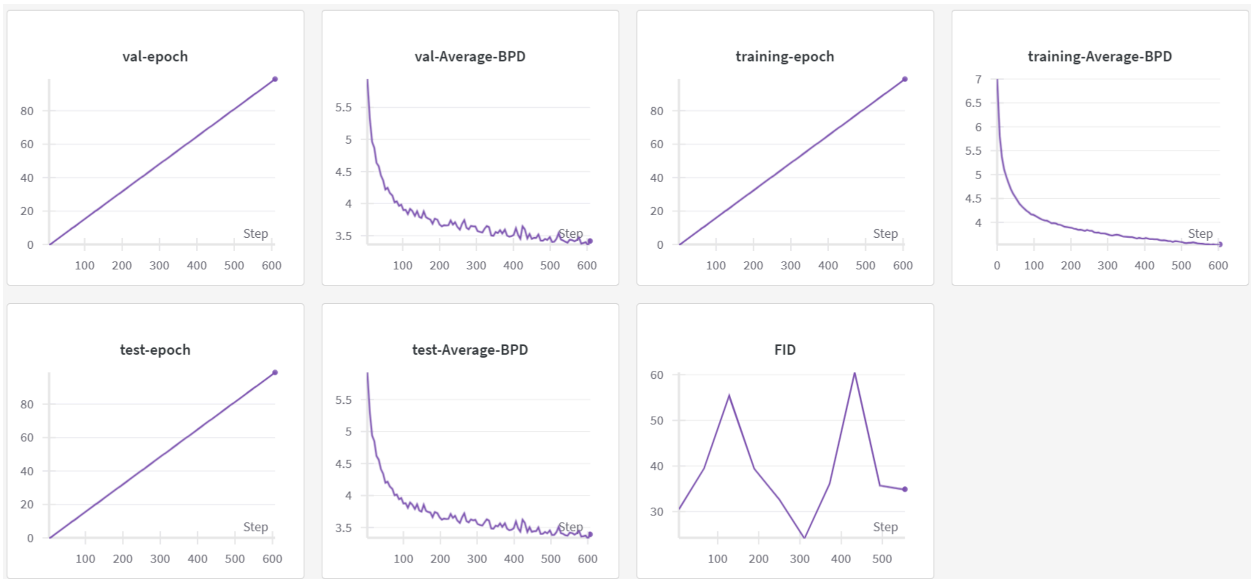


Figure 1: Wandb Training Models

2.2 Hyperparameter Adjustments

Due to limited time availability, only one set of hyperparameters were used for training. The chosen configuration aimed to strike a balance between computational efficiency and basic functionality, rather than optimizing for performance.

Hyperparameter	***Value***
nr_resnet	1
nr_filters	40
batch_size	16
sample_batch_size	16
lr_decay	0.999995
max_epochs	100
sampling_interval	10
save_interval	10
input_channels	3
nr_logistic_mix	5

Table 1: Hyperparameter Values

This single configuration was used throughout training and no additional tuning was performed. While this limits the ability to compare performance across setups, it provided a working baseline within the available timeframe.

2.3 Image Generation Result

Image generation was evaluated using the *generation_evaluation.py* script. A total of 100 images (25 per class) were generated and saved into labeled folders on Wandb.

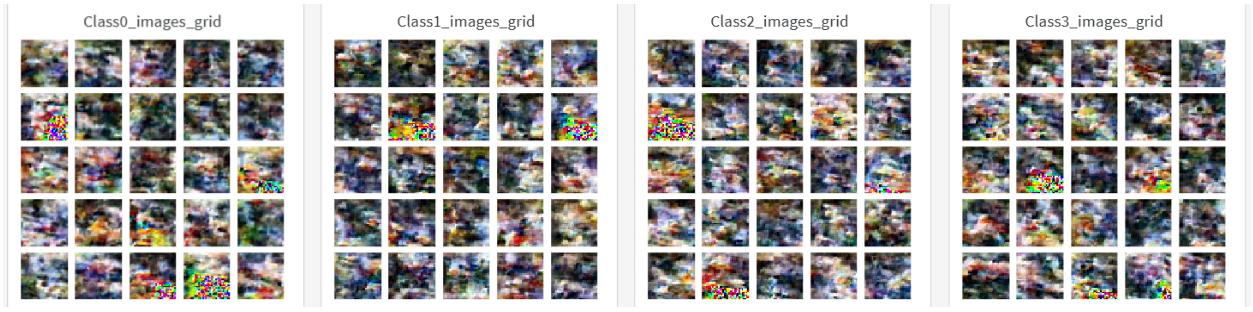
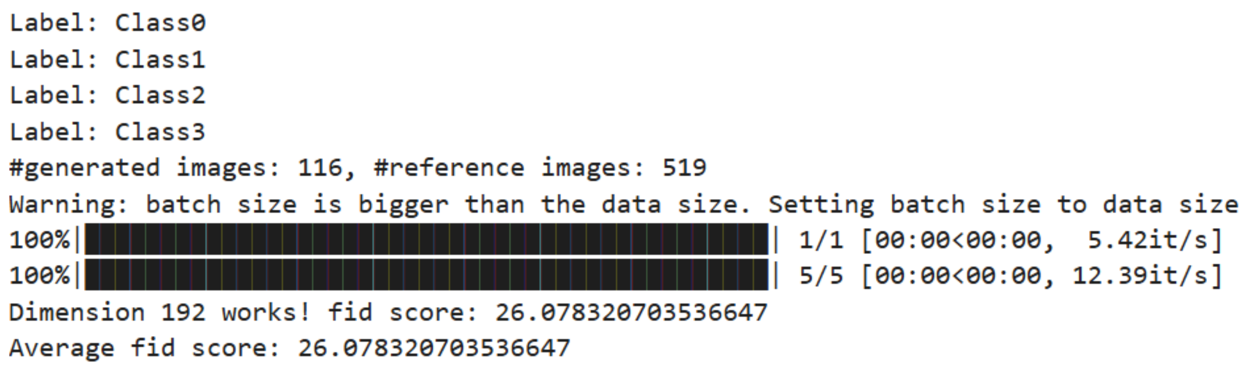


Figure 2: Generated Images Separated by Class

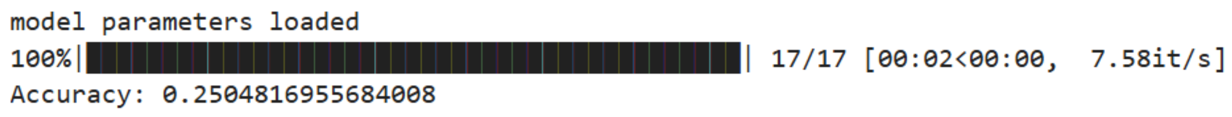


```
Label: Class0
Label: Class1
Label: Class2
Label: Class3
#generated images: 116, #reference images: 519
Warning: batch size is bigger than the data size. Setting batch size to data size
100%|██████████████████████████████████████████████| 1/1 [00:00<00:00,  5.42it/s]
100%|██████████████████████████████████████████████| 5/5 [00:00<00:00, 12.39it/s]
Dimension 192 works! fid score: 26.078320703536647
Average fid score: 26.078320703536647
```

Figure 3: Generation_Evaluation.py Console Output

2.4 Classification Results

To assess how well the model captured class-conditional structure, the *classification_evaluation.py* script was used to perform a likelihood-based classification.



```
model parameters loaded
100%|████████████████████████████████████████████| 17/17 [00:02<00:00,  7.58it/s]
Accuracy: 0.2504816955684008
```

Figure 4: Classification_Evaluation.py Console Output

3. Conclusion

This project focused on building a conditional PixelCNN++ model capable of generating images based on class labels and classifying them through likelihood evaluation. The model was trained using a single set of hyperparameters due to time constraints and the training was monitored through the Weights & Biases platform.

The final trained model achieved an average FID score of 26.08, indicating that the generated samples were reasonably similar to the real dataset in terms of distribution. However, the classification accuracy reached only 0.25, suggesting the model's conditional likelihood estimates were not sufficiently distinct between classes. This is likely due to limited training time and a relatively shallow model configuration.

Despite these limitations, the project successfully demonstrated the core mechanics of PixelCNN++, including autoregressive sampling, conditional modeling, and likelihood-based classification. With extended training and more advanced tuning, both generation quality and classification performance could be improved significantly.

4. Bonus Questions

4.1 Question 1

Why do masked convolution layers ensure the autoregressive property of PixelCNN++?

Masked convolution layers ensure that the prediction for a pixel does not depend on itself or any future pixels, only on previous pixels. This enforces the autoregressive property, where each pixel is generated sequentially based on already generated pixels. [3]

In PixelCNN++, masked convolutions:

- Use a mask to zero out weights that would allow a pixel to “see” future pixels.
- Maintain a strict scanning order.
- Ensure that the joint probability of the image can be decomposed into a product of conditional probabilities.

4.2 Question 2

Implement different fusion strategies in the architecture and compare their performance.

In this project, label conditioning was implemented using PyTorch’s *nn.Embedding* layer, which was injected into the network by adding the embedded label vector to intermediate feature maps. This approach provided a scalable and learnable way to incorporate class information during training.

Although other strategies, such as one-hot encoding or FiLM (Feature-wise Linear Modulation) could have been used, only label embedding was used due to time constraints. It was chosen over one-hot encoding for its efficiency and ability to capture continuous relationships between classes. Because no other fusion methods were tested, there is no comparative performance data, but embedding provided a functional baseline for class-conditional generation.

4.3 Question 3

Why are the advantages of using a mixture of logistics used in PixelCNN++? *(Hint: You will get the answer if you go through the sampling function, also the similar philosophy shared in deepseek-v2/v3)*

PixelCNN++ uses a mixture of logistics to model the conditional distribution of pixel values (instead of a softmax over 256 values per color channel, like in the original PixelCNN [4]).

Advantages:

- Continuous output modeling allows smoother gradients during training
- Better sample quality and faster convergence
- Mixture components can capture multimodal pixel distributions
- Sampling from a logistic distribution is efficient and differentiable, useful during backpropagation

4.4 Question 4

Read papers and reproduce their methods for inserting conditions, comparing them with common fusion implementations.

This project did not include the reproduction of external conditioning methods from literature due to limited time. Instead, a basic fusion approach using embedded labels was adopted and integrated into the model before the gated residual blocks. This method aligns with common practices in conditional generative models and provides a simple way to test class-conditioning.

Although methods such as FiLM layers or attention-based conditioning could potentially offer improved performance, they were not explored here. Future work could involve implementing and comparing such strategies to understand their effect on generation quality and classification accuracy.

4.5 Question 5

Compare the performance of your model with dedicated classifiers (e.g., CNN-based classifiers) trained on the same dataset. Think about the advantages and disadvantages of your model compared with dedicated classifiers.

The conditional PixelCNN++ model achieved a classification accuracy of 0.25, which is relatively low compared to typical CNN-based discriminative classifiers trained for classification tasks. While PixelCNN++ offers the advantage of modeling pixel-level distributions and generating realistic samples, it is not optimized for classification.

Compared to a dedicated CNN classifier:

- Advantages of PixelCNN++:
    - Can generate images in addition to classifying them
    - Learns a full probabilistic model, enabling uncertainty estimation

- Disadvantages:
    - Slower inference due to sequential pixel generation
    - Lower classification accuracy when not specifically optimized for that task

5. References

[1] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, “PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications,” arXiv.org, 2017. https://arxiv.org/abs/1701.05517 ((accessed Apr. 14, 2025).

[2] “Deep Learning,” Deeplearningbook.org, 2016. https://www.deeplearningbook.org/ ((accessed Apr. 14, 2025).

[3] Aaron, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, “Conditional Image Generation with PixelCNN Decoders,” *arXiv.org*, 2016. https://arxiv.org/abs/1606.05328 (accessed Apr. 02, 2025).

[4] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, “PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications,” *arXiv.org*, 2017. https://arxiv.org/abs/1701.05517 (accessed Apr. 02, 2025).