Conditional PixelCNN++ for Image Generation and Classification

Mahdi Shakouri
University of British Columbia
Student ID 65301327
`mmshg@student.ubc.ca`
April 18th 2024

Abstract

PixelCNN++ is a generative model that computes pixel dependencies through autoregressive distributions. This project presents an extension of the PixelCNN++ model to incorporate class-conditional information to enable both targeted image synthesis and zero-shot classification. By embedding class labels and integrating them via fusion strategies, our model can generate class-specific images and perform classification tasks. The project focuses on transforming the unconditional PixelCNN++ into a conditional model while preserving the generative capabilities of the original model. Our experimental results demonstrate the success of this approach, achieving a 76.64% accuracy and an Fréchet Inception Distance score of 30.80 on the CPEN455 test dataset in classification and generative tasks.

1 Model

1.1 Model Artitecture

The base architecture of PixelCNN++ is a convolutional neural network (CNN) that uses masked convolutions to model the joint distribution of pixels in an image. The model is designed to generate images by sequentially predicting the values of each pixel based on the previously generated pixels.

The conditional PixelCNN++ model computes the joint distribution of pixels in an image conditioned on a class label $y \in \{0, \dots, C-1\}$ while keeping the autoregressive factorisation 6[1] as follows

$$p_\theta(\mathbf{x} \mid y) = \prod_{i=1}^{H} \prod_{j=1}^{W} p_\theta\big(x_{i,j} \mid \mathbf{x}_{<i,<j}, y\big) \tag{1}$$

Here, $\mathbf{x}_{<i,<j}$ represents the raster-scan "past" pixels and it ensures that the output of the model at each pixel position is only a function of the preceeding pixels (to it's left and above it) [1]. The model employs a causal two-stream input mechanism. Down-shifted (u) and down-right-shifted (u_l) convolutions are used to generate two feature maps that only consider past pixels, ensuring causality. Each gated residual block, shared by both the up and down layers of the PixelCNN++ architecture, computes:

$$\mathbf{h}_{\text{out}} = \mathbf{h}_{\text{in}} + \text{Activation}\big(W_a * \mathbf{h}_{\text{in}}\big)\, \sigma\big(W_b * \mathbf{h}_{\text{in}}\big), \tag{2}$$

where W_a and W_b are masked 3×3 convolution kernels, $*$ denotes convolution, and $\sigma(\cdot)$ is the logistic sigmoid function.

The architecture follows a U-net hierarchy with three up-sampling stages (resolutions $32 \to 16 \to 8$) followed by down-sampling stages ($8 \to 16 \to 32$). Nearest-neighbor (de)convolutions are used for scale changes, and skip connections concatenate the u and u_l features at each stage.

The output head consists of a network-in-network layer that projects the final u_l feature map to the parameters of a K-component mixture of logistics (MoL) for each channel:

$$p_\theta(x_{i,j}) = \sum_{k=1}^{K} \pi_k \ \mathrm{Logistic}\big(x_{i,j}; \mu_k, s_k\big) \tag{3}$$

where $\sum_k \pi_k = 1$, and π_k, μ_k, and s_k are the mixture weights, means, and scales, respectively.

1.2 Class-Conditional Generation

By injecting a one-hot encoding of the class at various stages throughout the model's forward pass, the generation of images was conditioned on the class labels. This was achieved by concatenating the class embedding with the feature maps at different levels of the network as seen in Figure 1. The class label is accessed from a one-hot encoding of the class index, which was then passed through a linear layer to obtain a class embedding.

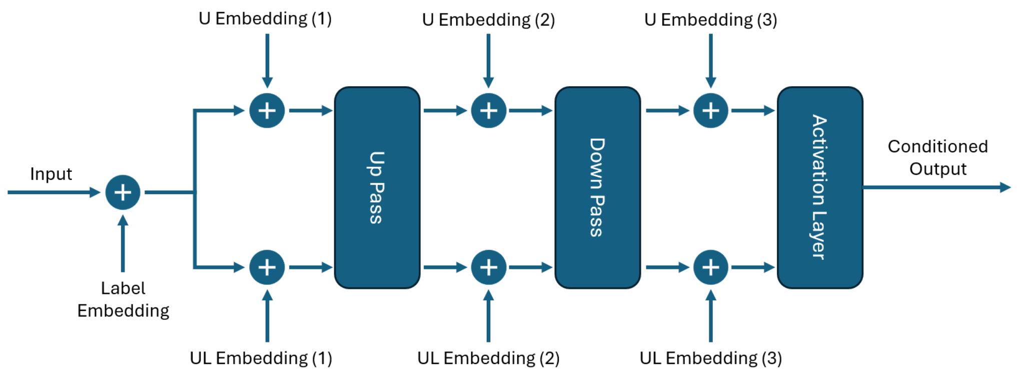

Figure 1: Class-conditional PixelCNN++ architecture. The class label is injected at various stages of the network to condition the generation process.

1.3 Fusion Strategy

The class embedding was injected into the model at different stages, and the fusion rules determine how the class information is integrated with the feature maps. The fusion rules are applied at the following stages:

Stage	**Spatial Size**	**Fusion Rule**
Early Fusion	32×32	$x \leftarrow x + e_y^0$
Up-stream initial	32×32	$u_0 \leftarrow u_0 + e_y^1,\ ul_0 \leftarrow ul_0 + e_y^2$
Middle Fusion	8×8	$u_b \leftarrow u_b + e_y^3,\ ul_b \leftarrow ul_b + e_y^4$
Late Fusion / logits	32×32	$\theta \leftarrow \theta + e_y^5$

Table 1: Fusion rules applied at different stages of the conditional PixelCNN++ model.

With these fusion rules laid in Table 1, the model can effectively incorporate class information with $\{e_y\}_\ell$ as the class embeddings injected at different stages of the network.

$$p_\theta(\mathbf{x} \mid y) = \prod_{i,j} p_\theta\big(x_{i,j} \mid \mathbf{x}_{<i,<j}, \{e_y\}_\ell\big), \tag{4}$$

1.4 Loss Function

For the generative PixelCNN++ model, the discretized logistic mixture (MoL) loss function [Salimans et al. 6] is utilized which is a common choice for generative models that output continuous values.

The model was trained using the negative log-likelihood (NLL) of discretized MoL bins which can be *simplified* to the following form:

$$\mathcal{L}_{\text{NLL}} = -\sum_{i,j} \log\Big[F_\theta(x_{i,j}+\tfrac{1}{2}) - F_\theta(x_{i,j}-\tfrac{1}{2})\Big] \quad \text{with} \quad F_\theta(x) = \sum_{k=1}^{K} \pi_k \ \text{Logistic}\big(x; \mu_k, s_k\big) \tag{5}$$

The original given logistic loss function was adapted to compute the loss for individual images instead of the entire mini-batches. This adjustment is crucial for classification tasks, as it allows the model to predict labels based on the loss of each image independently. Additionally, this approach improves the training process by providing a more precise evaluation of the model's performance on individual images rather than averaging over mini-batches.

2 Experiments

2.1 Training Methodology

The conditional PixelCNN++ model was trained end-to-end as a generative model on the *CPEN455* dataset provided in the repository. Each mini-batch of 32×32 RGB images were passed through the network, with class embeddings fused at multiple scales. The model parameters were updated by back-propagating the negative log-likelihood of a discretized mixture-of-logistics at each pixel. The loss was reported in bits per dimension (BPD) for interpretability, while the optimizer, Adam, with an initial learning rate of 2×10^{-4}, operated on the raw nats. A StepLR scheduler was given and employed to gradually decay the learning rate by a factor of 0.999995 after every epoch, enabling stable training over thousands of epochs.

During validation, the model's zero-shot classification capability was evaluated by computing the negative log-likelihood (NLL) for each image under all possible labels. The label with the lowest NLL was selected, providing a Bayes-optimal accuracy metric alongside BPD. Metrics were logged using the Weights-and-Biases **Wandb** platform, and checkpoints were saved at 5 epoch intervals. Training and validation losses for the final trial which yielded the best results are shown in the Appendix in Figure 3 and Figure 4 respectively.

To improve training speed, periodic image sampling and Fréchet Inception Distance (FID) computation was ommitted from later trials by commenting out the ralavent code in the training loop.

2.2 Training Workflow

Utilizing a Jupyter Notebook, training originally was set on Google Colab and then moved to Kaggle for better GPU support. The training process was initiated with a batch size of 16, and the model was trained for 50, 100, and then 200 epochs. Validation accuracy did not improve significantly with further epochs as seen in Figure 5.

2.3 Hyperparameter Tuning

Several hyperparameters were tested where the learning rate, batch size, number of filters, and number of logistic mixtures were varied. I studied the grid search approach [Graves 6] as part of my research and experimntation, but settled an manual adjustments of the hyperparameters based on the results of each trial. The final hyperparameters used in the Final Model are summarized in Table 2.

3 Discussions

This section is devoted to discussing the **[bonus]** questions. Detailed model description for first bonus is provided in the *Model* 1 section.

3.1 Why do maksed convolution layers ensure autoregressive property of PixelCNN++?

The masked convolution layers in PixelCNN++ ensure the autoregressive property by restricting the receptive field of each pixel to only include previously generated pixels, preventing information leakage from future pixels. This is imperative for maintainging the joint probability distribution of the image in a sequential manner. We can see this from the fact that PixelCNN++ finds the joint distribution of the pixels in an image by factorizing it into a product of conditional as detailed in equation 1.

The masked convolution layers applied at each step are according to the following equation:

$$\mathbf{y} = \text{Conv}(\mathbf{x}) \odot M, \tag{6}$$

The mask matrix M is designed such that it only allows information from previously generated pixels to be used in the convolution operation. It acts as a causal filter, ruling out connections to the current or future pixels, so every output only "sees" the past, exactly as required by an autoregressive model.

By using this masking technique, the model can generate images in a sequential manner, where each pixel is conditioned only on the pixels that have already been generated, ensuring the autoregressive property of the model.

3.2 What are the advantages of using a mixture of logistics in PixelCNN++?

Carefully inspecting the sample() function, we can see that the advantages of using a mixture of logistics in PixelCNN++ include:

- **Continuous Pixel Distributions:** The mixture of logistics allows the model to capture a wider range of pixel intensity distributions, which is particularly useful for complex images with diverse textures and colors. This leads to more realistic image generation.
- **Flexibility in Modeling:** The weighted sum in the mixed logistics (described in Eq. 3) lets the model approximate arbitrary multimodal, skewed, or saturated pixel value distributions with much fewer parameters than a softmax layer. This resluts in higher log-likelihoods (BPD) than a soft-max decoder.
- **Better Handling of Multi-modal Distributions:** The mixture of logistics can effectively represent multi-modal distributions, allowing the model to generate images with multiple distinct features or patterns.
- **Reduced Quantization Error:** The use of a mixture of logistics helps to reduce quantization error, leading to higher quality generated images with finer details. We can see this from the fact that the probability of a pixel value in the loss function in Eq. 5 is computed as the difference between the CDF values at $x + \frac{1}{2}$ and $x - \frac{1}{2}$:

  $$\log p(x = r) \;=\; \log\Big[F_\theta(r + \tfrac{1}{2}) - F_\theta(r - \tfrac{1}{2})\Big] \tag{7}$$

  The mixed logistics treats each integer pixel level $r \in \{0, \ldots, 255\}$ as the probability mass in a width-1 bin of a continuous distribution which results in a smaller quantization error and smoother gradients than a categorical cross-entropy.

4 Conclusion

Our class-conditional extension of PixelCNN++ effectively integrated label embeddings at multiple scales, achieving a zero-shot classification accuracy of 76.64% and a Fréchet Inception Distance (FID) score of 30.80 on the CPEN455 dataset, while preserving the autoregressive properties of the model.

However, several limitations remain despite these improvements. Training remains computationally expensive, with accuracy plateauing after approximately 200 epochs, suggesting that the current hyperparameter configuration may be near a local optimum. Additionally, the FID score is significantly behind state-of-the-art diffusion-based generators, and the model's applicability is constrained to 32×32 images, limiting its practical use. Future work on this project could explore adaptive learning rate schedulers and mixed-precision training to reduce computational costs, incorporate richer semantic embeddings (e.g., CLIP-derived) to enhance class conditioning, and investigate hybrid diffusion-autoregressive cascades to improve FID at higher resolutions. Techniques such as curriculum learning or dynamic masking could also be explored to accelerate convergence and improve training stability, paving the way for broader applications in controllable image synthesis.

5 Appendix

5.1 Appendix: Final Hyperparameters

Hyperparameter	Value
Learning Rate	0.002
Learning Rate Decay	0.999995
Batch Size	16
Number of Filters	100
ResNets	2
Number of Logistic Mixes	5

Table 2: Hyperparameters of the Final Model

5.2 Appendix: Performance Summary of the Final Model

Metric	F1 Score	Accuracy on Test Set (%)	FID
Model from Table 2	76.18	76.64	30.80

Table 3: Performance sommery of the Final Model on the test set as evaluated on the Hugging Face leaderboard.

5.3 Appendix: Sampling during Training

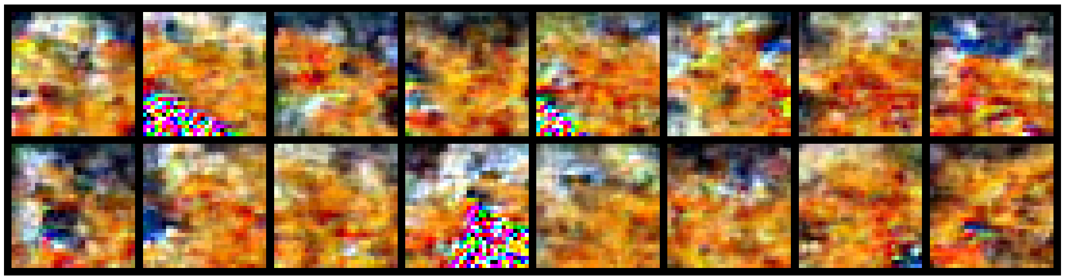

Figure 2: Sample generated images from class 3 during training.

5.4 Appendix: Training and Validation Losses

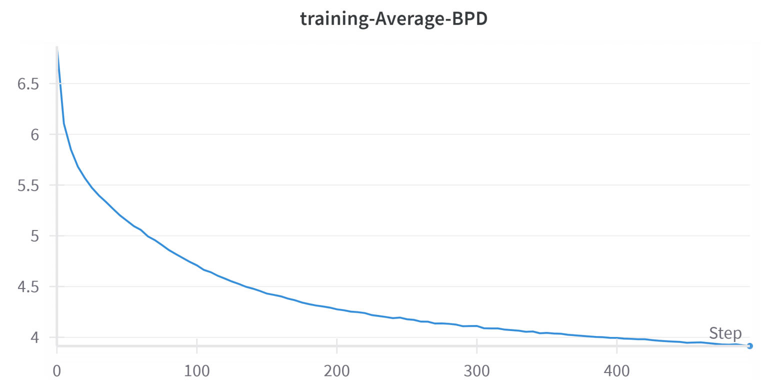

Figure 3: Training Loss (BPD) over 200 epochs.

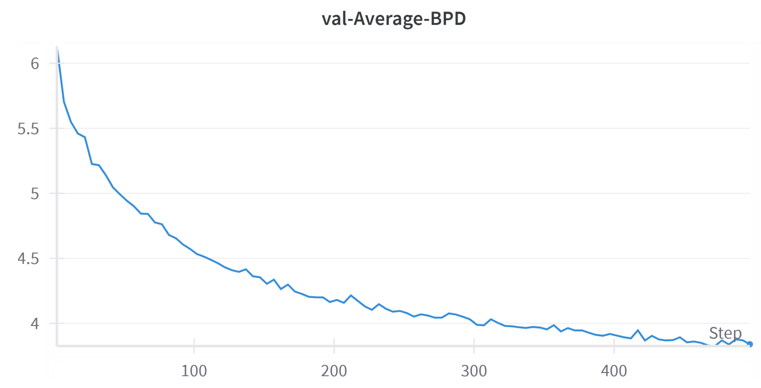

Figure 4: Validation Loss (BPD) over 200 epochs.

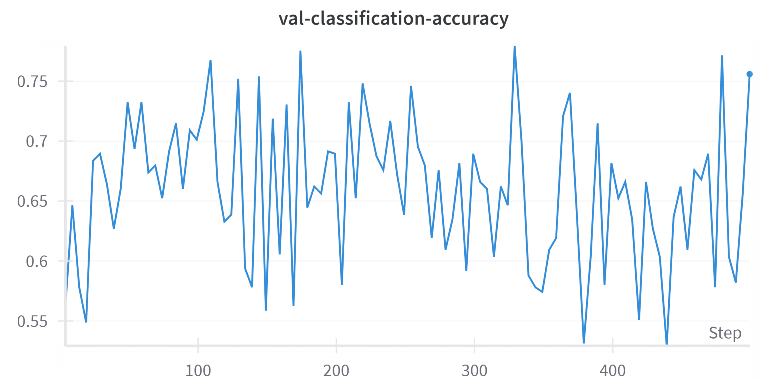

Figure 5: Classification Accuracy over 200 epochs.

5.5 Appendix: Sample Generated Images

Figure 6: Class 0 (a)

Figure 7: Class 1 (b)

Figure 8: Class 2 (c)

Figure 9: Class 3 (d)

Figure 10: Sample generated images from Final Model

6 References

[1] Graves, R. (2022). *Using a grid-search approach to validate the Graves–Pitarka broadband simulation method. Earth, Planets and Space (Online), 74(1), 186.* doi:https://doi.org/10.1186/s40623-022-01742-y

[2] Salimans, T. et al. (2017) *PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture.* arXiv preprint arXiv:1606.05328.

[3] Vanden Oord, A., et al. (2016). *Conditional Image Generation with PixelCNN Decoders. arXiv preprint arXiv:1606.05328.*