Conditional PixelCNN++

Tareq Alansari
73265217
UBC CPEN455
tareq.ayz@gmail.com

Abstract

This paper presents a conditional PixelCNN++ model designed for both image generation and classification. Given an initial unconditional PixelCNN++ model, we were tasked to modify the model and make it conditional. We will present our findings and explore possible implementations, depicting their results. We finally settled on a gated residual block fusion model that linearly projects a one-hot encoding of a class label into unique bias vectors, which are added within the block. The biases vectors correspond to an input bias, gate bias and output bias, the goal being to incorporate class conditioning where deep feature integration occurs in the PixelCNN++ model.

1 Model

The Conditional PixelCNN++ build upon the unconditional PixelCNN++ model by incorporating class-specific information into the models' layers. At its core, the model follows a U-Net structure with gated residual blocks, but is modified to condition all computations on a label c. This is achieved by fusing a class embedding into the network. The tricky part is fusion can be done in many different places of the model (early, middle, gated residual blocks, late) and in many different ways (additive, concatenation, FiLM). In my exploration, I was able to implement two different early fusion models, and one gated residual block fusion model. For submission I ended up submitting the model and epoch that performed the best, directly in terms of metrics (FID and test/validation accuracy).

1.1 Conditional PixelCNN++ Architecture

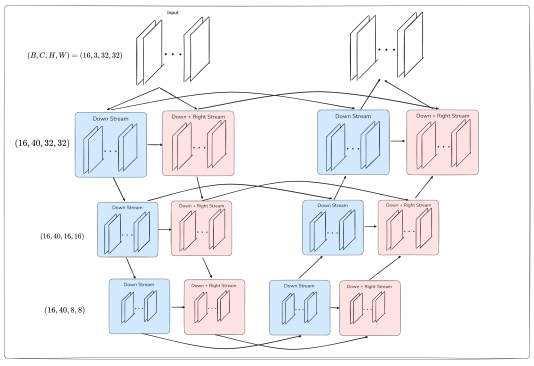


Figure 1: PixelCNN++ High-level architecture

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

The network comprises two primary stages: an Up-Pass and a Down-Pass.

During the Up-Pass (Decoding stage), the input is progressively downsampled, so there's a reduction in spatial resolution. At its core, feature maps are sequentially passed through three Up-Pass modules, each containing a number of gated residual blocks arranged into two streams. a *down-stream* (vertical) and a *down-right stream* (diagonal). These streams use specialized down-shifted and down-right shifted convolutions to ensure the autoregressive property is preserved.

In the Down-Pass (Encoding), the features extracted in the Up-Pass are refined and upsampled to recover the original spatial resolution. This stage uses deconvolution operations along with skip connections that merge features from the Up-Pass layers. The skip connections help retain fine-grained details that might otherwise be lost during the downsampling and upsampling process.

1.2 Gated Residual Block

Let's define the gated residual block as it's a key component of the model's architecture.

First we pass the input $\mathbf{x}$ through a nonlinearity function denoted as $f(\cdot)$. For our implementation we use the Concat-ELU function. Then we pass the result through a convolution operation, which is either a down-shifted or down-right-shifted 2D convolution. If an optional skip connection, $\mathbf{a}$, is given then it is also passed through the same nonlinearity function and subsequently passed through a network-in-network transformation.

$$\mathbf{z}_1 = \text{conv\_input}\big(f(\mathbf{x})\big) + \begin{cases} \text{nin\_skip}\big(f(\mathbf{a})\big) & \text{if } \mathbf{a} \text{ is given} \\ 0 & \text{otherwise} \end{cases}$$

Next, the modified input $\mathbf{z}_1$ is passed through the nonlinearity function again, after which we perform dropout with a dropout rate of 0.5. We do this to regularize the neural network and prevent the model from overfitting.

$$\mathbf{z}_2 = \text{Dropout}\big(f(\mathbf{z}_1)\big)$$

The dropout result is then convolved by another convolution operator. conv_out is designed so that the output has twice as many channels as the number of filters used (i.e. $\mathbf{y} \in \mathbb{R}^{2C \times H \times W}$).

$$\mathbf{y} = \text{conv\_out}\big(\mathbf{z}_2\big), \quad \mathbf{y} = [\mathbf{y}_1, \mathbf{y}_2]$$

The tensor $\mathbf{y}$ is split into two equal parts along the channel dimension so that $\mathbf{y}_i \in \mathbb{R}^{C \times H \times W}$. Then, a gate is applied via an element-wise sigmoid on the second half and multiplying it with the first half:

$$\mathbf{z}_{\text{gate}} = \mathbf{y}_1 \odot \sigma\big(\mathbf{y}_2\big)$$

Here $\sigma(\cdot)$ represents the sigmoid function. Finally, the output of the gated residual block is formed by adding this gated output back to the original input.

$$\mathbf{o} = \mathbf{x} + \mathbf{z}_{\text{gate}}$$

1.2.1 Conditional Gated Residual Block

To make our PixelCNN++ model conditional, we now include a class embedding provided as $\mathbf{h}$. If a class condition is provided to the gated residual block, it projects $\mathbf{h}$ via a linear layer into a vector of size $3 \times$ num_filters. This vector consists of 3 biases: the input bias, the gate bias and the output bias. These biases are applied additively, so we get the following modifications:

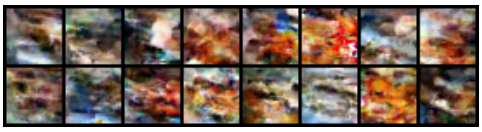
$$\begin{aligned} \mathbf{z}_1 &\leftarrow \mathbf{z}_1 + \text{input\_bias}(\mathbf{h}) \\ \mathbf{z}_{\text{gate}} &= \mathbf{y}_1 \odot \sigma\big(\mathbf{y}_2 + \text{gate\_bias}(\mathbf{h})\big). \\ \mathbf{o} &= \mathbf{x} + \mathbf{z}_{\text{gate}} + \text{output\_bias}(\mathbf{h}) \end{aligned}$$

The goal of these changes is to condition the entire residual block on the class information, thus modulating the processing based on the class label.

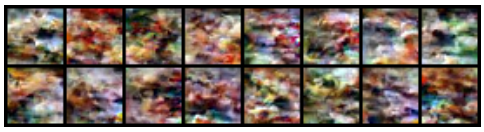
2 Experiments

2.1 Training Method

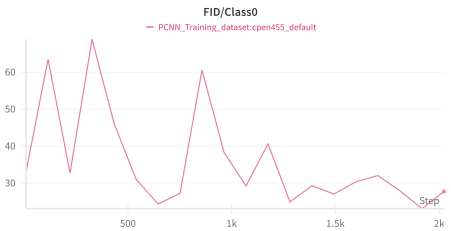
Throughout the development process, I experimented with different training strategies and models to balance sample generation quality and classification performance. Inspired by the TAs' Wandb dashboard, I enhanced my own dashboard by tracking not only the overall FID but also per-class FID values. The figures below show generated class samples along with the corresponding per-class FID values for one of the early fusion models.



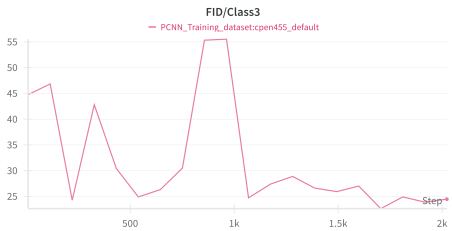
(a) Early-Fusion Class 0 Samples – step 1809



(b) Early-Fusion Class 3 Samples – step 2018



(c) Class 0 FID Score per sampling interval



(d) Class 3 FID Score per sampling interval

Figure 2: Early-Fusion model generated Class samples and corresponding FID scores

While the per-class FID metric provided useful insight into how well the model generated samples for each category, two major issues led to its eventual de-prioritization. First, the computational expense of frequent FID evaluation significantly increased training time. Second, FID does not always correlate with the perceived visual image quality, as it may look noisy even when the FID is low.

Arguably the most important addition to my training process was the tracking of training and validation accuracy scores. My methodology to produce and submit the highest benchmark scoring model was as follows: In each epoch, training and validation accuracy is calculated. Specifically, the calculation is determined using the log likelihood (a.k.a. log probability) that an image belongs to a certain class. The predicted class for an image is the class that produces the largest log likelihood. Then, the class predictions are compared with the ground truth labels. As a result, we are able to measure the training and validation accuracy.

While training the model, whenever it achieves a new high on its validation accuracy, the model is saved. This way, we attempt to continuously save the "best" model, which is not dictated by the latest model or epoch, but rather by its validation accuracy. Fortunately, the accuracy on the validation set tends to generalize well on the test set. My submitted model gets a **validation set accuracy of 75.9%**, and a **test set accuracy of 76.6%**, all while achieving a FID below 30. There is a downside though which is that the **generated samples are very noisy**. As a result, it must be noted that the submitted model is far from perfect, and still has a lot of room for improvement.

2.2 Analysis

I would like to acknowledge and point out the inconsistent behavior in the validation accuracy. While training my model, the training accuracy sees a gradual increase until it levels out and fluctuates between an accuracy score of 50 to 60. On the other hand, the validation set has an accuracy score ranging from the mid-40s to mid-70s. This shows that the model itself is not stable, which suggests there are some underlying issues in the gated residual block fusion implementation.

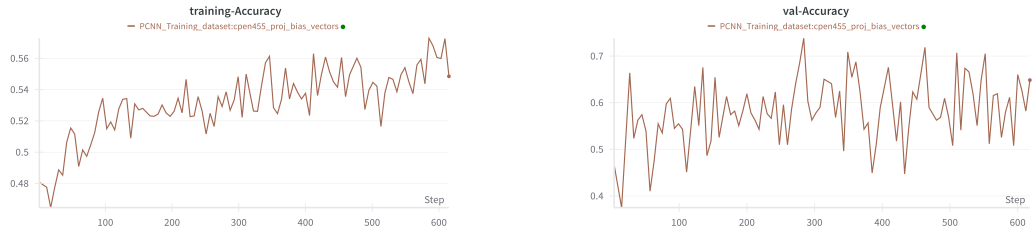


Figure 3: Training and validation accuracy for the gated residual block fusion model. You can see these graphs are very volatile and only indicate a minor uptrend in the training accuracy. Clearly this indicates there is some underlying issue with the conditional model implementation.

3 Conclusion

Through strategic training of the model – tracking training and validation accuracy – my model performed pretty well on the benchmarks, achieving a **FID score of 25.16**, and an **accuracy of 76.6%** on the test set (according to the Hugging Face leaderboard). Although I would like to attribute this score to my methodology of strategically embedding the class condition in the model, I believe it was more so that I got lucky at a certain point where my model achieved a validation accuracy score around 75%. While training my model, I continuosly save the model with the best validation accuracy, and luckily, it generalizes well with the test set.

For future improvements of my model, I would definitely like to improve the quality of the generated images. Although the generated samples achieve a low FID score, they are not visually appealing to the human eye, and appear to have a lot of noise. For future avenues to improve my model, one thing I would like to try out is a concatenation or FiLM fusion strategy throughout the layers of the network. Additionally, something I was considering and would like to look into is normalization techniques. My theory is that the high noise regions are a result of blow-up, perhaps through certain non-linear activations. As a result, we get very high-intensity pixels, which means it maxes out a subset of pixel values in the RGB channels. This might be an issue with the additive method I've been using to fuse class conditions directly into intermediate tensors through learned biases.

References

[1] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional Image Generation with PixelCNN Decoders," *arXiv:1606.05328 [cs]*, Jun. 2016, Available: `https://arxiv.org/abs/1606.05328`.

[2] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications," arXiv:1701.05517, Jan. 19, 2017, [Online]. Available: `https://arxiv.org/abs/1701.05517`.

[3] Wikipedia Contributors, "Fréchet inception distance," Wikipedia, Mar. 30, 2021, [Online]. Available: `https://en.wikipedia.org/wiki/Fr%C3%A9chet_inception_distance`.

[4] H. Gao, "An Explanation of Discretized Logistic Mixture Likelihood," *Medium*, Jul. 23, 2019, [Online]. Available: `https://medium.com/@smallfishbigsea/an-explanation-of-discretized-logistic-mixture-likelihood-bdfe531751f0` (accessed Apr. 16, 2025).

4 Appendix

A Bonus Questions

A.1 Why do masked convolution layers ensure the autoregressive property of PixelCNN++?

Masked convolutions are essential in PixelCNN++ because they guarantee that at every convolutional layer, each pixel's value is computed solely based on **past** (already generated) pixel values. This trait of the PixelCNN++ is what enforces the autoregressive property in the network.

Let's dive more deeply into this. First of all let's clearly define what it means to be autoregressive. In order for a model to be autoregressive, it needs to predict the next component in a sequence from the previous inputs in the sequence. We can see that the model does this since the PixelCNN++ architecture follows the rule:

$$P(x) = \prod_{i=1}^{N} P(x_i \mid x_1, \ldots, x_{i-1})$$

Or in other words, PixelCNN++ is an autoregressive model that factorizes the joint distribution of an image into a product of conditional distributions. We can see that it's autoregressive since the prediction of a pixel x_i is conditioned only on the pixels that come before it. But the network must be designed to support this property, which is where masked convolution layers come into play.

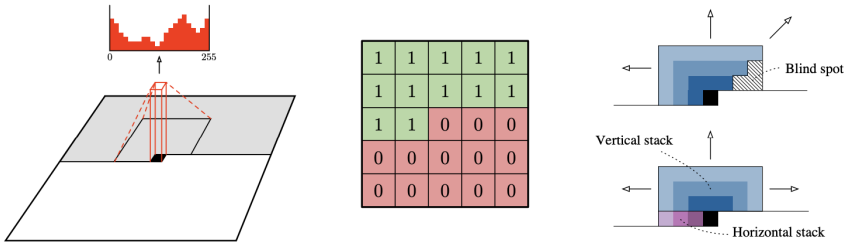


Figure 4: This figure is taken from [1]. On the left, is an illustration of how PixelCNN uses the neighborhood of already generated pixels to predict the next pixel. To generate a pixel x_i, the model must only rely on previously generated pixels $\{x_1, \ldots, x_{i-1}\}$. In the center, a masked filter matrix is shown, demonstrating how the model zeros out connections to future pixels, thus preventing access to data below or to the right of the current location during predictions. On the right, PixelCNN's design can result in a blind spot in the receptive field. This is an area the network cannot use for prediction. By combining the vertical and horizontal stack convolutional streams (shown in blue and purple at the bottom), the model ultimately manages to capture the entire receptive field.

In a masked convolution, a binary mask is applied to the convolutional filters so that certain weights (corresponding to future or not-yet-generated pixels) are zeroed out. This ensures that the convolution at location i does not have access to pixel values from any position j where $j > i$.

By zeroing out connections to future pixels, every convolutional operation respects the autoregressive property of generating pixels only based on the previous pixel values. When generating pixels sequentially during sampling, due to the masked convolution, the network only sees the pixels that have been generated. Thus, we can see why masked conolution layers ensure the autoregressive property of PixelCNN++.

A.2 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)

There are many advantages of using a mixture of logistics as can be seen by PixelCNN++. Following off of the original PixelCNN++ report [2] I will summarize these key advantages:

- **Memory and Computational Efficiency**: Instead of handling a bulky 256-way softmax per sub-pixel, the model uses a small number of logistic components (often around 5). This reduces the parameter overhead and results in denser gradients, speeding up training.
- **Smooth Representation of Pixel Intensity**: The continuous logistic mixture captures the fact that neighboring pixel values are similar (e.g., 127 is close to 128 and 129), unlike the discrete softmax which treats these values independently.

B Early Fusion Model

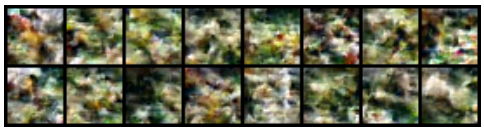
There were two trained early fusion models with slightly different implementations. They performed quite well in terms of the FID score measurement, but the accuracy on the test set was only 59%. In general, my early fusion models were implemented using a class embedding layer and directly adding it to the model input.

In one of the fusion models we directly add a class embedding to the input x. In another model, we add the class embedding to the beginning of the Up-Pass. Formally, we can define $e(c)$ as the embedding for class c and so, the fusion is expressed as:

$$u_0 = u_0 + e(c), \quad ul_0 = ul_0 + e(c)$$

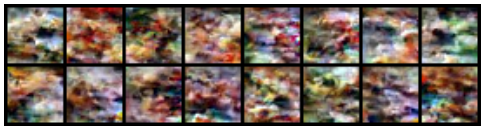
where u_0 and ul_0 are the initial activations from the first convolution layers.

(a) Early fusion epoch 250 Class 0 Samples



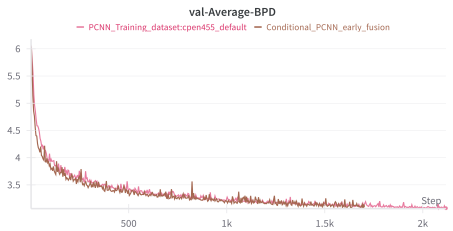
(b) Early fusion epoch 250 Class 1 Samples

(c) Early fusion epoch 250 Class 2 Samples



(d) Early fusion epoch 250 Class 3 Samples

Figure 5: Generated samples for each class after 250 epochs (Early Fusion)



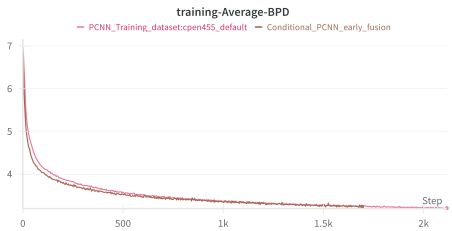


Figure 6: Training and Validation Averages (BPD) for both early fusion models

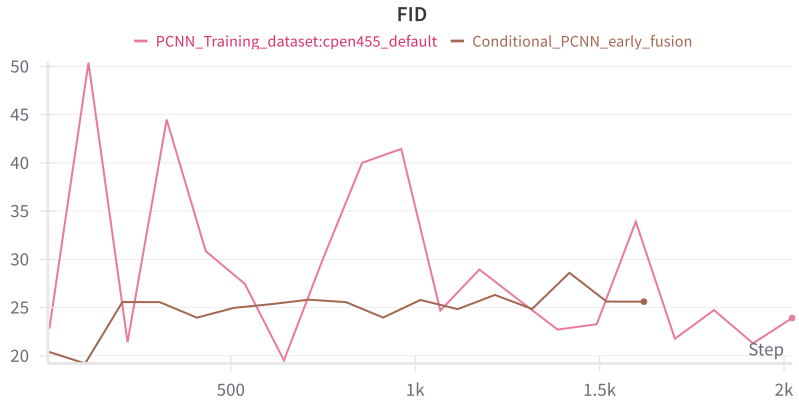
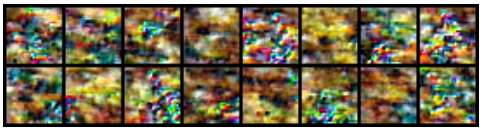


Figure 7: Overall FID score for both early fusion models

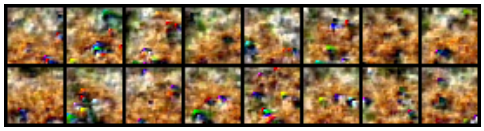
C Gated Residual Block Fusion Model

This appendix is to show the generated samples by the gated residual block fusion model. As you can see, the noise in the model seems to increase as the model continues to train. This is likely due to some improper implementation or logic which causes the generated images to look like such. That being said, I would like to acknowledge this fact, but also let the readers know that this is the model I submitted because it managed to achieve a FID below 30 and both a test and validation score above 75%. While there's no doubt in my mind that this model has some logical errors, I submitted it solely because it had the best performance. It should also be noted that I didn't submit the final model, but rather the trained model at epoch 104.

However, it should be noted that the FID measurement has been shown to be innacurate at times: my samples corroborate that fact, as the images in epoch 100 are not clear, but still have a low FID score.



(a) Epoch 25 Class 0 Samples

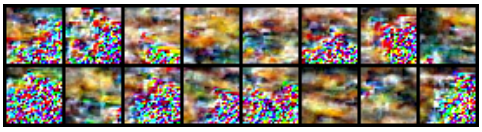


(b) Epoch 25 Class 1 Samples

(c) Epoch 25 Class 2 Samples

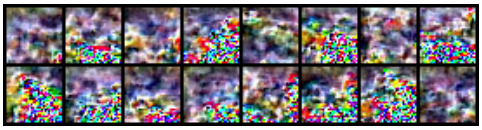
(d) Epoch 25 Class 3 Samples

Figure 8: Generated samples for each class after 25 epochs (Gated Residual Block Fusion)

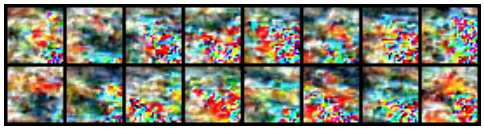


(a) Epoch 100 Class 0 Samples

(b) Epoch 100 Class 1 Samples



(c) Epoch 100 Class 2 Samples



(d) Epoch 100 Class 3 Samples

Figure 9: Generated samples for each class after 100 epochs (Gated Residual Block Fusion)

Model Name	Accuracy	F1 Score	Submission Date
Final_Model_Hopefully	0.772	0.766	2025-04-15T19:10:52Z
nova_v4_large_epoch180	0.772	0.764	2025-04-15T18:44:52Z
pixelCNN++_fixed	0.768	0.766	2025-04-08T22:51:38Z
nova_v1.1_earlyfusion	0.768	0.764	2025-04-14T23:15:32Z
test	0.768	0.764	2025-04-14T23:23:18Z
cooked_or_cooking	0.766	0.761	2025-04-15T19:18:15Z
fight-the-landlord	0.764	0.766	2025-04-15T07:44:44Z
pixelCNN++ Please Works	0.764	0.766	2025-04-14T21:37:16Z
pixelCNN++	0.764	0.756	2025-04-14T22:35:12Z
test2	0.763	0.759	2025-04-14T19:02:16Z
pixelCNN++_INSHALLAH	0.761	0.758	2025-04-15T10:40:40Z
pixelCNN++ middlajj	0.761	0.758	2025-04-15T10:36:05Z

Figure 10: The model's performance (cooked_or_cooking) on the Hugging Face leaderboard. A test set accuracy of 76.6% and an F1 Score of 76.1%. But the age old question remains: did I cook or did I get cooked – Sun Tzu probably. I am at the mercy of the grader I suppose.

D AI Acknowledgment

I would like to acknowledge my use of generative AI technologies to aid me in this final project. The aid mainly consisted of queries related to the codebase and fusion implementations. This includes but is not limited to:

- Help with debugging.
- Using Generative AI to understand aspects of the project and the code base.
- Generating custom scripts. For example, a script to create a csv of predicted labels on the test set given a model. This csv is then uploaded to the Hugging Face leaderboard.
- Strategizing possible fusion implementations.
- Slight revisions in the report documentations.

If the grader would like more details on specific queries and/or responses, please let me know. All the conversations are saved in the tools' history.