# Conditional PixelCNN++ for Image Classification

**Sean Ghaeli**
UBC
78590676
seanghaeli@student.ubc.ca

## Abstract

PixelCNN++ is leveraged as a powerful generative model, offering tractable likelihood computation by modeling pixel dependencies through conditional distributions. Extending this framework to conditional generative tasks via class embeddings enables targeted image synthesis. In the context of zero-shot image classification, a trained conditional PixelCNN++ provides a promising solution by synthesizing images conditioned on potential classes and evaluating their likelihood. Following this approach, we pre-trained a Conditional PixelCNN++ to achieve an F1 score of $79.6\%$, accuracy of $79.6\%$ and Fréchet Inception Distance (FID) of $21.3$ across the CPEN 455 Hugging Face Final Project validation dataset.

## 1 Model

PixelCNNs model the joint distribution of pixel values in an image $x$ as the product of conditional distributions, $x_i$ being a single pixel:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1}).$$

The pixel dependencies are ordered in raster scan order, meaning each pixel depends on all pixels above and to the left of it within the image. This dependency structure ensures that each pixel's value is influenced only by preceding pixels, not future ones. For a more complete description of PixelCNN please see Van den Oord et al., 2016 [1].

Given a high-level image description encoded as a latent vector $h$, the goal is to model the conditional distribution $p(x|h)$ of images corresponding to this description. The conditional PixelCNN extends the unconditional model by incorporating terms dependent on $h$ into the activations prior to nonlinearities. This modified model, represented by the equation:

$$p(x|h) = \prod_{i=1}^{n^2} p(x_i | x_1, \ldots, x_{i-1}, h),$$

allows for the generation of images conditioned on specific descriptions, enriching the model's capability for conditional image generation.

This also provides a direct pathway to conduct zero-shot classification using Bayes' rule:

$$p(c|x) = \frac{p(x|c)p(c)}{p(x)}.$$

In this equation, $p(c|x)$ represents the probability of class $c$ given image $x$, $p(x|c)$ denotes the likelihood of observing image $x$ given class $c$, $p(c)$ represents the prior probability of class $c$, and

$p(x)$ is the marginal likelihood of image $x$. By leveraging the conditional PixelCNN model, $p(x|c)$ can be estimated directly, enabling the classification of images into unseen classes based on their generated likelihoods under each potential class.

We condition the generation of images by injecting a one-hot encoding of the class at various stages throughout the model's forward pass. Throughout the forward pass (Depicted in Figure 1), feature maps of the input are split which we will refer to separately as $u$ and $ul$, and traverse two stages: an up pass and a down pass. Class embeddings are incorporated at four key stages: (1) at the beginning of the forward pass, (2) just before the up pass, (3) again just before the down pass, and (4) right before activation. The one-hot encoding of our class, denoted as $h_{j,k,c}$ ($c$ representing the class, $j$ representing which of the four above stages it is being applied, and $k$ specifying the feature map u or ul.), serves as latent embeddings of a specific class. Essentially, we provide the model with information about the target class through this encoding, enabling it to generate images that align with the characteristics of that class.

Throughout our experiments, we combinatorically modify which embeddings are used in pursuit of the best-performing model. During our training phase, we generate images conditioned on each class and evaluate the loss by comparing the reconstructed images. Subsequently, we predict the class associated with the lowest loss. For loss computation, we employ the Discretized Mix Logistic Loss, inspired by the work of Salimans et al. (2017) [2].

## 2 Experiments

Following the work of Van den Oord et al., 2016 [1], our initial experiment uses $tanh$ as the activation function and all class embeddings in Figure 1 on. Later experiments used $ReLU$ as the activation, in which results were seen to improve. Maximum epochs were set to 50, as most experiments converged to a stable image classification accuracy within this amount of training. The various training experiments are summarized in Table 1, with hyperparameters in 3. Training on a T4 GPU, these hyperparameters allow for training at reasonably high speeds, at around 1 minute per epoch. Ultimately, our best classification model was found to have an F1 score of $79.6\%$ on the Hugging Face test set (Table 2). During training, a notable trend emerged wherein the training loss exhibited a rapid increase, contrasting with the continuous decrease observed in the validation loss, as depicted in Figure 2. This phenomenon serves as a significant indicator of overfitting, suggesting that the model has begun to memorize the training data rather than generalizing to unseen data.

One notable constraint observed during training until 50 epochs was the consistently low quality of generated images, as evidenced in Figure 3, despite consistently achieving FID scores below 30 throughout the training process. Notably, while image quality remained subpar, the underlying classification task remained viable. This is because the efficacy of the classification task hinges on the ability of class embeddings to effectively cluster images within their respective classes in the latent space, rather than their alignment with human perception of image quality. Consequently, our model demonstrated proficiency in classification well before achieving satisfactory image generation quality. Extending our best model's training to 350 epochs revealed discernible enhancements in image generation, with distinct shapes as well as color palettes emerging for each class, as seen in 4. It is reasonable to hypothesize that further computational resources would lead to further improvements in image generation quality and could be a subject of future study. However, our primary constraint in this regard was computational capacity.

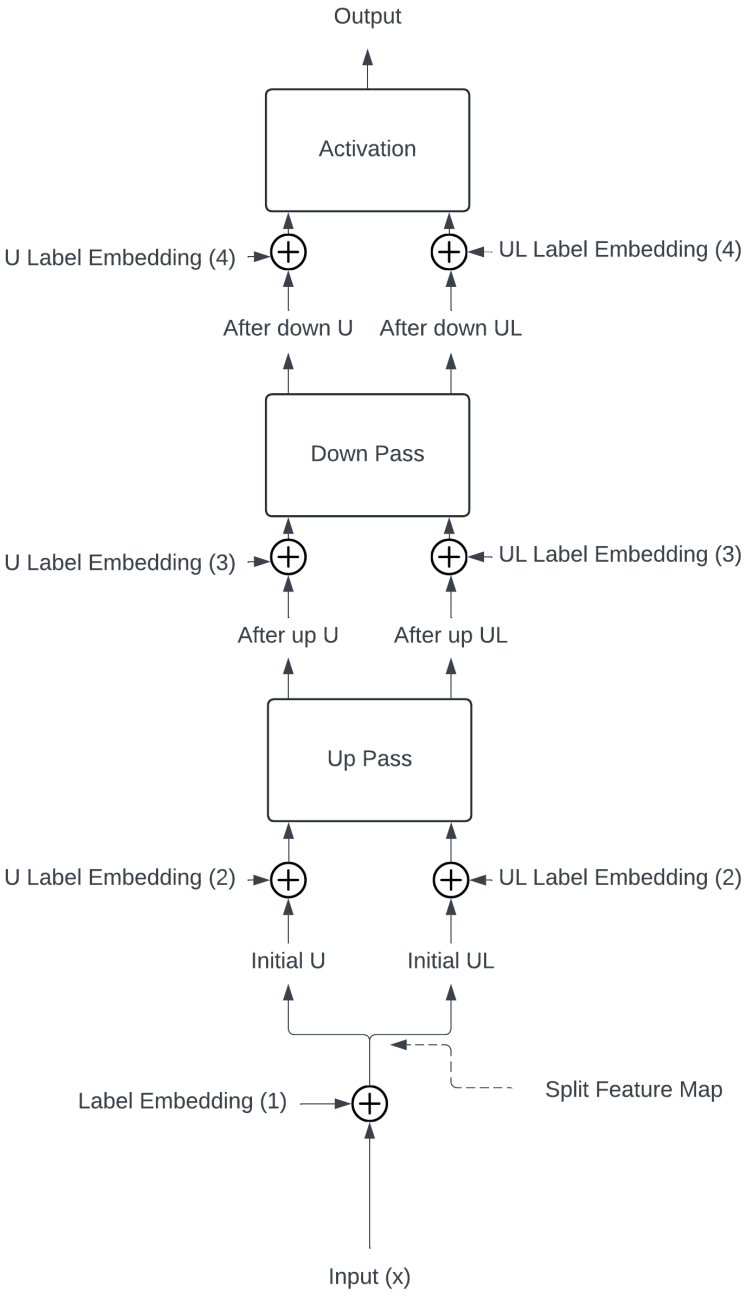



Figure 1: Diagram of how embeddings are infused into the model at various points throughout the PixelCNN++ forward pass. For example, "U Label Embedding (2)" corresponds to $h_{2,c,u}$.

# 3 Conclusion

In conclusion, our exploration into the Conditional PixelCNN++ has yielded promising results in the domain of zero-shot image classification. Leveraging the power of PixelCNN++ architecture, we successfully extended the model to facilitate targeted image synthesis by conditioning on class embeddings. Through a series of experiments, we demonstrated the model's capability to generate images aligned with specific class descriptions, achieving competitive performance metrics on the CPEN 455 Hugging Face Final Project validation dataset.

Our key findings underscore the effectiveness of Conditional PixelCNN++ in zero-shot image classification tasks, with our best model achieving an F1 score of $79.6\%$ and an accuracy of $79.6\%$ (Table 2).

Furthermore, after significant training, image generation began yielding recognizable structural features of classes in Figure 4, suggesting that the model had learned meaningful representations of class characteristics. This observation highlights the model's ability to capture intricate class-specific features.

Our workflow revolved significantly around computational resources, primarily relying on the T4 GPU provided by Kaggle, which was available at no cost. This constraint limited our ability to experiment with larger, more complex models that could potentially capture class-specific information more effectively. As a result, we chose to work with small to medium-sized models, enabling quick training iterations and facilitating rapid experimentation and refinement of our approach. Despite these limitations, our streamlined approach allowed for efficient exploration of model architectures and parameter configurations. Looking ahead, there is a clear avenue for future research to explore the use of more complex models capable of comprehensive representation of class label information, leveraging advanced computational resources to further enhance our understanding and capabilities in conditional generative modeling for zero-shot learning and targeted image synthesis applications.

# References

[1] Van den Oord, A. et al. (2016) Conditional Image Generation with PixelCNN Decoders. *Arxiv*.

[2] Salimans, T. et al. (2017) PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications. *Arxiv*.

# Appendix

In the attached code with this submission, the checkpoint model is the only model inside the models folder, located at 'models/conditional_pixelcnn.pth'. The images to do the FID calculations with are in 'samples/'. The logits for the test set are stored in 'test_logits.npy'. The csv results of the test set output are stored in 'submission.csv'.

The classification script can be run by typing (from the project's root folder) 'python classification_evaluation.py' and likewise for the image generation script using 'python generation_evaluation.py'.

Table 1: Model Performance Summary

| Model | Epochs of Training | Accuracy on Validation (%) | FID |
|---|---|---|---|
| 1 | 50 | 77.4 | 27.0 |
| 2 | 50 | 79.9 | 23.8 |
| **3** | **50** | **81.9** | **21.3** |
| 4 | 50 | 78.3 | 25.4 |

Notes: Model 1 trained with all embeddings on, tanh activation. Model 2 trained with U Label Embedding (4) off, ReLU activation. **Model 3 trained with UL Label Embedding (4) off.** Model 4 trained with UL Label Embedding (4) and Label Embeddings (3) off.
Please note that validation accuracies are represented here because despite 24 hours of waiting, only Model 3's results were listed on Hugging Face, which are listed in Table 2.

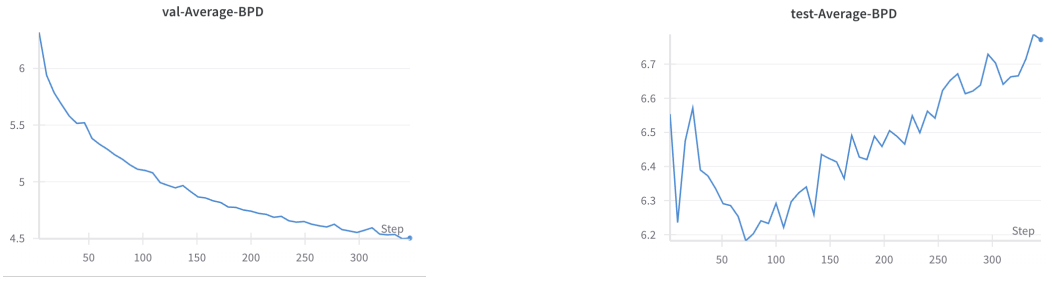
Table 2: Best Model Test Set Performance Summary

| F1 | FID | Accuracy on Test Set (%) |
|---|---|---|
| 79.6 | 21.3 | 79.6 |

Our best model's (Model 3 from Table 1) performance on the Hugging Face test dataset.

Table 3: Hyperparameters of the Model In Table 1

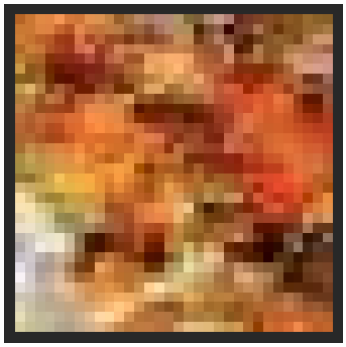
| Hyperparameter | Value |
|---|---|
| Learning Rate | 0.002 |
| Learning Rate Decay | 0.999995 |
| Batch Size | 16 |
| Number of Filters | 100 |
| ResNets | 2 |
| Number of Logistic Mixes | 5 |




(a) Validation Loss

(b) Test Set Loss

Figure 2: Validation Loss and Test Set Loss. The x-axis represents 7-steps per epoch.



(a) Class A, Burgers

(b) Class B, Pandas

(c) Class C, Koalas

(d) Class C, Pizza

Figure 3: Image generation samples at Epoch 50, from best model in Table 2

(a) Class A, Burgers

(b) Class B, Pandas

(c) Class C, Koalas

(d) Class C, Pizza

Figure 4: Image generation samples at Epoch 350, from best model in Table 2