Image Classification with Conditional PixelCNN++

Shovon Biswas
Department of Physics and Astronomy
University of British Columbia, Canada

Abstract

We modify the unconditional autoregressive PixeCNN++ image generation model to generate images conditioned on class labels and use it to classify images using Bayes's theorem. Depending on how strongly the conditioning is implemented, we investigate two cases which we call *mild* and *strong* conditional models. We find that the mild model outperforms the strong model when used as a classifier. We can achieve an accuracy $\approx 85\%$ with moderate training when the conditional autoregressive model is used as a classifier.

1 Model

The PixelCNN++ model [1] is a modification of the PixelCNN model [2]. Both models are autoregressive models that generated the next pixel conditioned on the previous pixels:

$$p_\theta(x_i) = \prod_{i=0}^{n} p_\theta(x_i \mid x_{i-1}, x_{i-2}, ..., x_0) \equiv \prod_{i=0}^{n} p_\theta(x_i \mid x_{<i}) \tag{1}$$

In both models, pixels are generated sequentially in raster scan order (left-to-right, top-to-bottom). PixelCNN employs masked convolutional neural networks to ensure that the prediction of pixel x_i depends solely on previously generated pixels $x_{<i}$, modeling each conditional distribution $p(x_i \mid x_{<i})$ as a discrete probability over 256 possible values per channel for 8-bit RGB images, typically parameterized by a softmax. This approach, while effective, is computationally intensive and struggles to capture the continuous nature of pixel intensities.

PixelCNN++ addresses these limitations by modeling $p(x_i \mid x_{<i})$ using a mixture of logistic distributions, defined as $p(x_i \mid x_{<i}) = \sum_{k=1}^{K} \pi_k \cdot \text{logistic}(x_i \mid \mu_k, s_k)$, where K denotes the number of mixture components, and π_k, μ_k, and s_k represent the weight, mean, and scale of the k-th logistic component, respectively. The logistic distribution is given by $\text{logistic}(x \mid \mu, s) = \frac{1}{s}\sigma\left(\frac{x-\mu}{s}\right)\left(1-\sigma\left(\frac{x-\mu}{s}\right)\right)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function. To capture inter-channel dependencies in RGB images, PixelCNN++ constrains the means across channels, such as $\mu_k^r = f(x_{<i})$, $\mu_k^g = \mu_k^r + \Delta\mu_k^{rg}$, $\mu_k^b = \mu_k^g + \Delta\mu_k^{gb}$, where $\Delta\mu_k^{rg}$ and $\Delta\mu_k^{gb}$ are learned offsets. Additionally, PixelCNN++ incorporates dropout regularization to mitigate overfitting, simplifies the network architecture for computational efficiency. These enhancements enable PixelCNN++ to generate higher-quality images with improved training stability and efficiency compared to PixelCNN, marking a significant advancement in autoregressive image modeling.

1.1 Conditional PixelCNN++

PixelCNN++ employs a stack of masked convolutional layers with residual connections and gated activation units, inspired by WaveNet [3], to model complex spatial dependencies efficiently. It further utilizes a multi-scale architecture with parallel "slow" and "fast" stacks, where the fast stack conditions on downsampled features from the slow stack to integrate global context.

For conditional image generation, we use a training set where each image has a class label. There are several way of adding class labels to the unconditional model. For deeper label integration, both the original PixelCNN and PixelCNN++ added the label information in the resnet layer. This is shown in **Algorithm 1** which we dub as *Mild conditioning*. Note that, the class-labels are added as bias to preserve the translation invariance of the convolution. However, since we are interested in classification with our conditional model, we might try to impose a stronger conditioning. This is achieved by adding class-labels twice after two convolution layers as shown in **Algorithm 2**. We call this *Strong conditioning*. The other structures of the original PixelCNN++ model [1] remain unchanged in our implementation.

Algorithm 1: Mild conditioning

```
 1: Input: Input data
 2: Apply non-linearity
 3: Apply convolution
 4: if skip connection information exists then
 5:     Mix with skip connection information
 6: end if
 7: Apply non-linearity
 8: Apply dropout
 9: Apply convolution
10: Add class-labels as bias
11: Apply gating mechanism
12: Return: Processed output
```

Algorithm 2: Strong conditioning

```
 1: Input: Input data
 2: Apply non-linearity
 3: Apply convolution
 4: Add class-labels as bias
 5: if skip connection information exists then
 6:     Mix with skip connection information
 7: end if
 8: Apply non-linearity
 9: Apply dropout
10: Apply convolution
11: Add class-label as bias
12: Apply gating mechanism
13: Return: Processed output
```

2 Experiments

The model was trained on a custom dataset called ‘CPEN455‘ which contains 4160 labeled training images, each image being $H \times W = 32 \times 32$ divided into four categories. During training, a loss function calculates how well the PixelCNN++ model's predicted mixture of logistic distributions matches the actual pixel values in the input images. For each batch, the model outputs parameters (means, scales, and mixture weights) for a logistic mixture distribution. The loss function then: (1) computes the probability of each observed (discretized) pixel value under this predicted distribution using logistic CDF differences, (2) sums the negative log probabilities across all pixels and mixture components, and (3) The loss function is then normalized as :

$$\text{BPD} = \frac{\text{loss}}{\text{batch size} \times \text{channel} \times \text{H} \times W \times \log(2)} \tag{2}$$

The model is then trained to minimize the BPD and the results are validated using a validation set of 520 images. We run the initial benchmark of the mild model using a linear scheduler for a fixed number of epoch as shown in Fig 1.

The plots show that the model benefits from higher numbers of resnet laters and a higher number of CNN filters. The cost of increasing these two parameters is a larger models and slower inference due to more computations. Interestingly, increasing the number of mixture components for the logistic mixture distribution lowers BPD. However, constraining it too much might harm the quality of generated images as the number of logistic mixtures allows for flexibility but slower sampling rate.

Based on the above results, we fix the number of model parameters as follows for both mild and strong conditional models:

number of resnet layers=5, number of CNN filters= 120, number of logistic mixtures= 10.

For training our model with use *adam* [4] with pytorch's built-in *CosineAnnealingLR* learning rate scheduler. This scheduler leads to much less oscillation in the later stages of the training compared to the linear scheduler. We also modify the supplied code to be able to use 2 T4 GPUs for training.

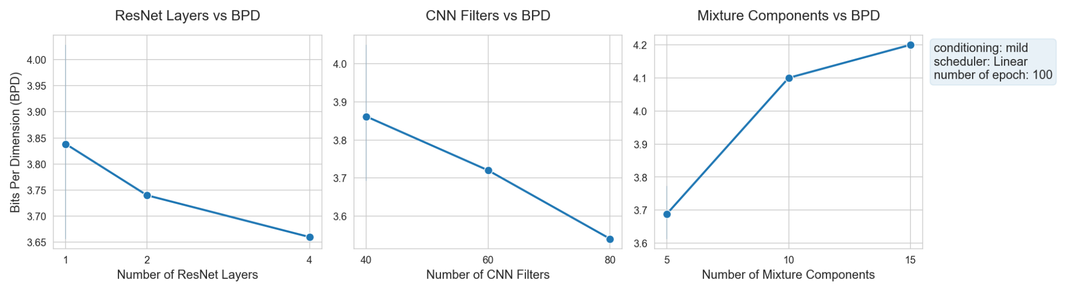

Figure 1: The model's BPDs with respect to various model parameters after a fixed number of epoch are plotted.

While training we made sure that the model does not overfit by monitoring the BPD on the training and validation sets.

2.1 Image generation

Although we train our model to minimize BPD, we measure the quality of the conditionally generated images by their Fréchet Inception Distance (FID), which measures the similarity of the generated images to the supplied real images. The reason behind this is that FID is a more effective metric than BPD for assessing human-perceived visual quality in generative models [5]. While lower BPD indicates better statistical modeling, it may not ensure perceptual realism, whereas lower FID aligns closely with human vision by measuring similarity in a feature space derived from real images. The results for this experiment on the validation set is summarized in Fig: 2

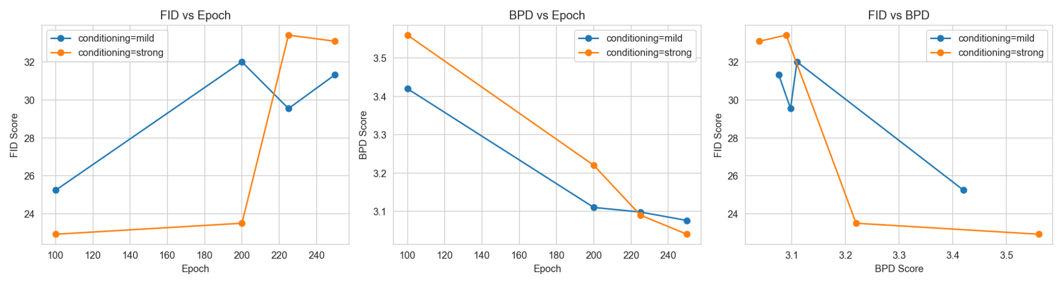

Figure 2: Training and Image generation metrics of the models are shown.

The results show that as the training progresses, both models' BPD decreases which is expected. However, the FID scores increases for both models. This is a bit surprising since we would expect the FID score to go lower as the training progresses. The strong conditioning model performs worse than the mild model. In particular, at the final stage of training, the strong model's BPD is lower than the mild model but its FID score is higher. This signals that the model is learning to memorize.

2.2 Classification

Once we have a conditional model, we can use it to classify image. Given the model outputs $p(x\,|c_i)$ where c_i is the class-label for a fixed number of class $i = 1, 2, ..., N$, using Bayes's theorem we get

$$p(x \mid c_i) \propto p(c_i \mid x)\; p(c_i) \tag{3}$$

In our training dataset $p(c_i) = \frac{1}{4}$ for all $i = 1, 2, 3, 4$.. Thus given an image x, we calculate the $p(x \mid c_i)$ for all $i = 1, ..., 4$ and take the class-label that has maximum probability as our predicted class. We accomplish this by simply modify our loss function that outputs *batch-averaged* negative log probabilities to output negative log probabilities *per sample*. The results for this experiment on the validation set is summarized in Fig: 3

Here, we find that the mild model consistently performs as the better classifier reaching approximately $\approx 85\%$ accuracy on the validation set. For the strong model, the accuracy on the validation set goes down at the later stages of the training. This also signals memorization.

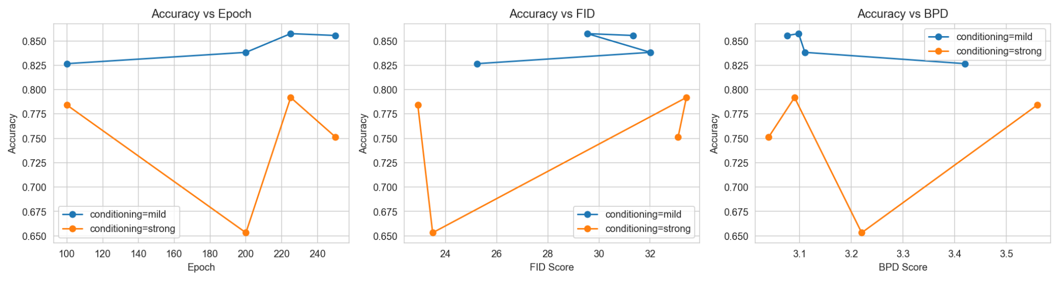

Figure 3: Results for the accuracy experiment in the Validation set

3 Conclusion

In this work, we modifed a unconditional PixelCNN++ model to an image classifier. We investigated to instances which we call mild and strong conditional models. We find that these two different fusion strategies lead to interesting result. The strong model, in which the image labels are fused twice in the resnet layer, performs poorly as a classifier than the mild model where the fusion was done only once.

During the experiments we found evidence of memorization in the strong model. It should be noted that we did not find any evidence of overfitting during the training for both models, which is usually signaled by poorer performance on the validation set compared to the training set. It will be interesting to investigate why this happens in details. Also, we could try other fusion strategies to investigate their performance as classifiers.

We used BPD to optimize the network. However, we were interested in images that look better for human eyes. It would be interesting to train the model to optimize on FID or some other scores.

Our experiments were performed until 250 epochs. However, the model still does not overfit, so it could be trained for more epochs. In particular, our model still has quite high FID (≈ 30) score on the validation data for this custom dataset, which is not great for the PixelCNN++ type models which can achieve FID score between $10 - 20$. A longer training would, perhaps, lead to a better FID score.

Finally, it would be interesting to train our model on a differ dataset. Our code already allows for it. Unfortunately this could not been done due to time and resource constraints.

References

[1] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

[2] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29, 2016.

[3] Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu, et al. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 12, 2016.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.