CPEN 455 Final Project, Conditional PixelCNN++

Aaditya Suri
UBC, CPEN 455
asuri02@student.ubc.ca
41935511

Abstract

The report describes a conditional PixelCNN model which is used for generating and classifying images. The model was able to achieve approximately 2.8 BPD and a 73% classification accuracy on the validation set. The following sections describe the Model architecture, experiments and a conclusion.

1 Model

To convert the unconditional PixelCNN base model to a conditional one, we need to be able to pass in the class labels and fuse them with the pixel information from the images. Figure 1 describes the model architecure for fusing class label information with the images.

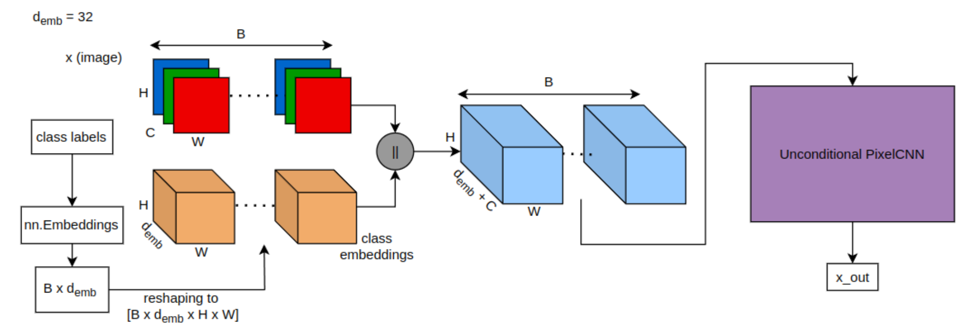

Figure 1: Model architecture

1.1 Generating class embeddings

I first encoded the class labels (`class0, class1, class2, class3`) to integers using the `my_bidict` data structure. Class embeddings were generated using the nn.Embeddings module from Pytorch, for a batch size of B, and setting the embedding dimension size, d_{emb} to 32, we get class embeddings of shape $[B \times d_{emb}]$. these were then expanded and reshaped to the shape $[B \times d_{emb} \times H \times W]$ to match the image dimensions.

1.2 Fusing class label information with pixel data

After generating class label embeddings of the shape $[B \times d_{emb} \times H \times W]$, the class embeddings are concatenated with the pixel data along the channel (C) dimension. The final concatenated output has the shape $[B \times (d_{emb} + C) \times H \times W]$. This output is then passed to the unconditional PixelCNN model that was provided to us.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

With this method, we can fuse the class label information with the pixel data. Thus we can go from this unconditional model

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i|x_1, x_2, \ldots, x_{i-1}) = \prod_{i=1}^{n} p_\theta(x_i|x_{<i}) \tag{1}$$

To the conditional model where the pixel generation depends on the previous pixel values as well as the class label information

$$p_\theta(x|c) = \prod_{i=1}^{n} p_\theta(x_i|x_{<i}, c) \tag{2}$$

1.3 Image generation

The images are generated using this `sample()` function in `utils.py`, one pixel at a time

1.4 Image classification

For zero shot image classification, we can use the `discretized_mix_logistic_loss()` function in `utils.py`. the original function sums over the entire `log_sum_exp` value of shape $[B \times H \times W]$ to produce the negative log likelihood of the entire batch, this value is not particularly useful for image classification, we need the negative log likelihood of each image in the batch. To do this, we can simply sum over the H and W dimensions using the following

```
-torch.sum(log_sum_exp(log_probs), dim=(1, 2))
```

Now to classify an image x, we can calculate this value for that image given each class label, and we classify the image as the label with the minimum log likelihood. thus modeling the following equation:

$$p(c = i|x) = \frac{p_\theta(x|c = i) \times p(c = i)}{\sum_{k=1}^{4} p_\theta(x|c = k) \times p(c = k)} \tag{3}$$

2 Experiments

The current model for fusing class information with the pixel data was chosen after trying several different model configurations and looking at the sampling results, the current model produced the best results. Some of the sampling results from the previous and current model architectures are shown below, all sampled at 90 epochs:

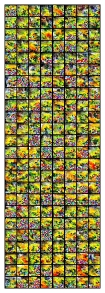
Figure 2: First model

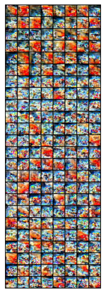
Figure 3: Intermediate model

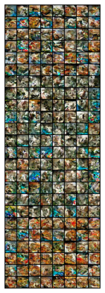
Figure 4: Current model

clearly, the current model architecture has much better samples in Figure 4 at the same level of training. It was also observed that increasing the values of `nr_resnet`, `nr_logistic_mix` and `nr_filters` greatly improved the sample quality. The current model was trained for 500 epochs, however the model weights at the 500th epoch did not generate the best samples, and the model at the 420th epoch was chosen as the best model. The samples generated at this epoch are shown below.

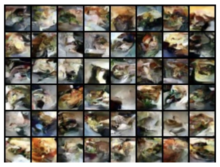
Figure 5: class0

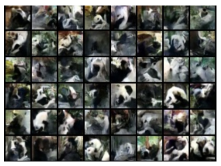
Figure 6: class1

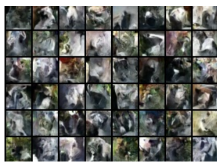
Figure 7: class2

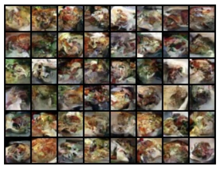
Figure 8: class3

during the training process, to monitor how well the model is trained, the 500 epochs were trained incrementally in batches of 100 epochs where the training for the next 100 epochs is started by loading the last saved .pth file from the previous 100 epochs. During this, It was also observed that on increasing the batch size from 8 to 16, there was a relatively big decline first in the training BPD, as shown below:

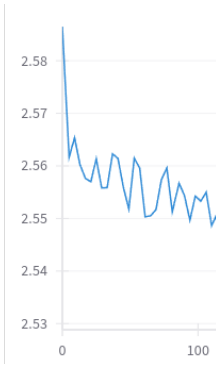

Figure 9

The entire training and validation BPD curves are shown in in Figure 10:

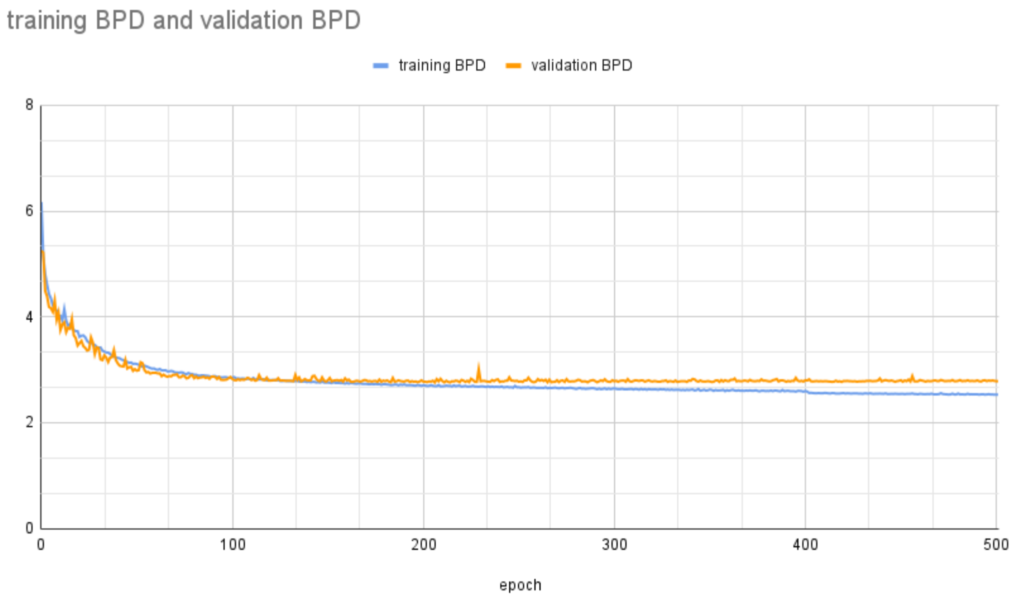

Figure 10

3 Conclusion

For this model, the classification accuracy results on the training, validation and test sets are shown in Table 1:

Table 1: Image classification accuracy

Metric	Train	Validation	Test
Accuracy	0.98	0.73	0.74

the image generation also resulted in an FID score of around 41.

3.1 Possible Improvements

One possible improvement over the current way to fuse class labels is that since d_{emb} is set to 32, and the images only have 3 channels, after the concatenation, not a lot of information from the pixel data is retained. This can be improved by adding intermediate layers to capture more features from the pixel data before fusing with the class label information.

Currently, the class labels are only added to the input at the very beginning of the forward pass. Adding this fusion at different network depths can also potentially improve the model performance.