

PixelCNN

Hoang Pham

DSLAb

1. Introduction

Autoregressive (AR) generative models achieve the best results in density estimation tasks involving high dimension data, such as images or audio. They pose density estimation as a sequence modeling task. This report will elaborate PixelCNN model - one of the most important model in AR models.

1.1. Autoregressive model

AR is a class of generative models with a tractable likelihood. The model fully factorizes the probability density function on a data point \mathbf{x} over all its elements as:

$$p(\mathbf{x}) = \prod_{i=1}^D p(x_i | \mathbf{x}_{<i})$$

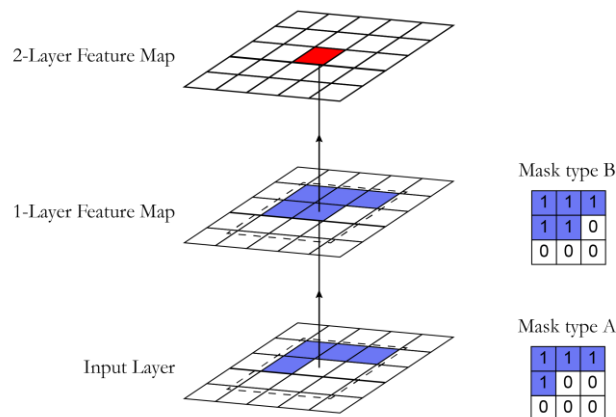
In the context of deep generative models, we will drop the condition of linear dependence and formulate our image problem as a random process. In particular, we will:

- Use a deep generative models (obviously non-linear), and
- Assume the pixels of an image are random variables with a specific ordering (top-down, bottom-up, left-right), which formulates it as a random process.

1.2. PixelCNN

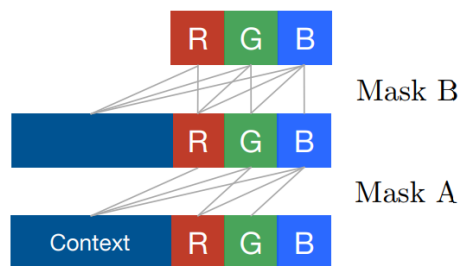
PixelCNNs are the convolutional version of PixelRNNs, which treat the pixels in an image as a sequence and predict each pixel after seeing the preceding pixels (defined as above and to the left, though this is arbitrary).

PixelRNNs are slow to train since the recurrence can't be parallelized — even small images have hundreds or thousands of pixels, which is a relatively long sequence for RNNs. Replacing the recurrence with masked convolutions, such that the convolution filter only sees pixels above and to the left, allows for faster training.

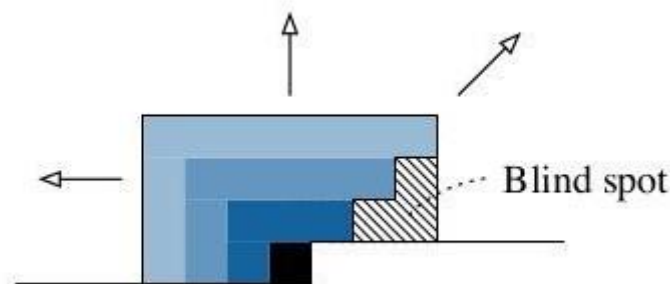


There are two types of mask. Mask type A does not consider the value of itself and are used in the first convolution layer. While mask type B consider the center element's value and are used in the following convolution layers.

When we experiment on color images, we need to consider the color conditioning on each pixel. We choose the sequent order is R -> G -> B. We also have two mask types. In mask type A, channels are not connected to themselves and only used in first convolution layer. And in mask type B, channels are connected to themselves and used in all subsequent convolution layers.



However, it's worth noting that the original PixelCNN produced worse results than the PixelRNN. One possible reason is that stacking masked convolutional filters results in blind spots, failing to capture all the pixels above the one being predicted



In order to increase the performance of pixelCNN, Gated PixelCNN was proposed to remove the blind spot by combining two convolution stack: one is horizontal stack and another is vertical stack.

Generating samples

Generating images is something that is a bit more complicated but follows the same idea from my post on Autoregressive Autoencoders:

1. Set $i=0$ to represent current iteration and pixel.
2. Start off with a tensor for your image $x(0)$ with any initialization.
3. Feed $x(i)$ into the PixelCNN network to generate distributional outputs $y(i+1)$.
4. Randomly sample sub-pixel $u(i+1)$ defined by $y(i+1)$.
5. Set $x(i+1)$ as $x(i)$ but replacing the current sub-pixel with $u(i+1)$.
6. Repeat step 3-5 until entire image is generated.

2. Experiments

2.1 Training details

- n_filters: 64
- Batch size 128
- Learning rate 0.001

Resnet block:

- ReLU
- 1x1 mask type B with in_channels = n_filters and out_channels = n_filters // 2
- ReLU
- 7x7 mask type B with in_channels = n_filters//2 and out_channels = n_filters//2, padding 3
- ReLU
- 1x1 mask type B with in_channels = n_filters//2 and out_channels = n_filters

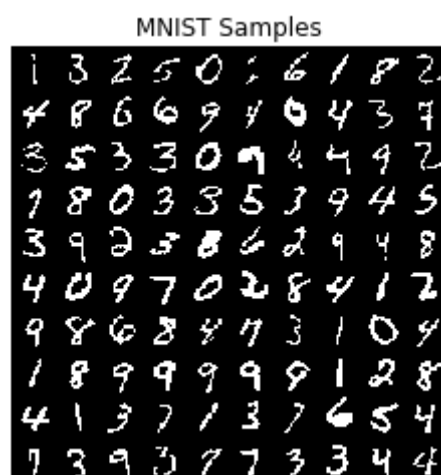
Model:

- 7x7 mask type A with in_channels = input_channels, out_channels = n_filters, padding 3
- (Layer normalizing + Resnet block) * num_resblock
- ReLU
- 1x1 mask type B with in_channels = n_filters and out_channels = n_filters
- ReLU
- 1x1 mask type B with in_channels = n_filters and out_channels = n_filters*n_colors

2.2 MNIST dataset

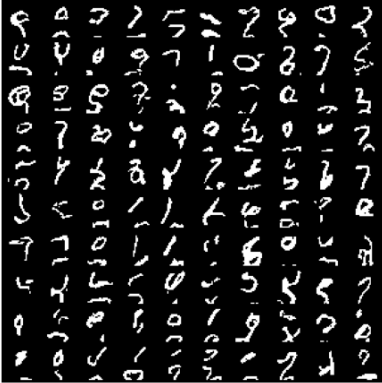


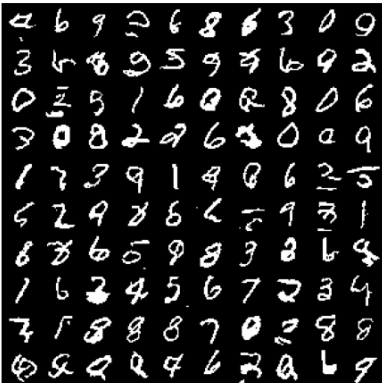


This dataset includes 60000 digit hand writing images from 0 to 9 in training set.







Each class has 6000 images with size 28x28



2.2.1. The effect of layer normalization

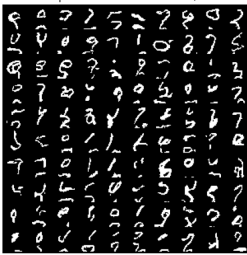


We training the above pixelCNN model with 100 epochs, skip connection and 5 resnet block










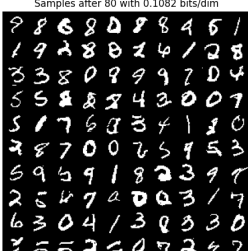


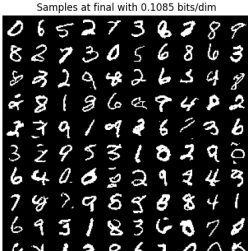


Layer normalization	No layer normalization
<p>Samples after 0 with 0.1241 bits/dim</p> 	<p>Samples after 0 with 0.1291 bits/dim</p> 
<p>Samples after 20 with 0.1086 bits/dim</p> 	<p>Samples after 20 with 0.1103 bits/dim</p> 
<p>Samples after 40 with 0.1083 bits/dim</p> 	<p>Samples after 40 with 0.1085 bits/dim</p> 

<p>Samples after 60 with 0.1078 bits/dim</p> 	<p>Samples after 60 with 0.1081 bits/dim</p> 
<p>Samples after 80 with 0.1082 bits/dim</p> 	<p>Samples after 80 with 0.1090 bits/dim</p> 
<p>Samples at final with 0.1085 bits/dim</p> 	<p>Samples at final with 0.1085 bits/dim</p> 

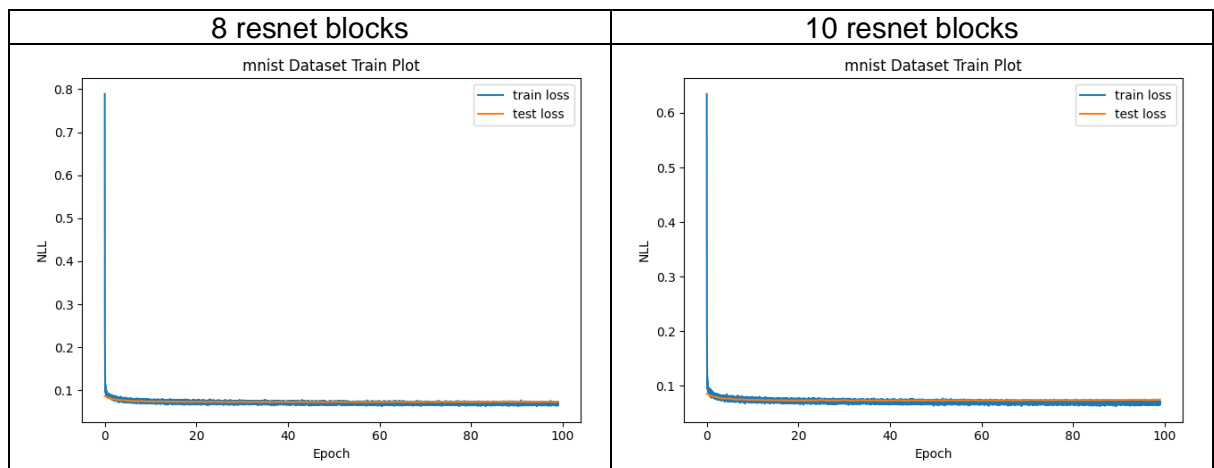
2.2.2. The effect of the number of resnet block

We training pixelCNN model with layer normalizing, 100 epochs and skip connection

5 resnet blocks	8 resnet blocks	10 resnet blocks
<p>Samples after 0 with 0.1241 bits/dim</p> 	<p>Samples after 0 with 0.1249 bits/dim</p> 	<p>Samples after 0 with 0.1228 bits/dim</p> 

<p>Samples after 20 with 0.1086 bits/dim</p> 	<p>Samples after 20 with 0.1054 bits/dim</p> 	<p>Samples after 20 with 0.1051 bits/dim</p> 
<p>Samples after 40 with 0.1083 bits/dim</p> 	<p>Samples after 40 with 0.1054 bits/dim</p> 	<p>Samples after 40 with 0.1050 bits/dim</p> 
<p>Samples after 60 with 0.1078 bits/dim</p> 	<p>Samples after 60 with 0.1054 bits/dim</p> 	<p>Samples after 60 with 0.1052 bits/dim</p> 
<p>Samples after 80 with 0.1082 bits/dim</p> 	<p>Samples after 80 with 0.1055 bits/dim</p> 	<p>Samples after 80 with 0.1061 bits/dim</p> 
<p>Samples at final with 0.1085 bits/dim</p> 	<p>Samples at final with 0.1061 bits/dim</p> 	<p>Samples at final with 0.1073 bits/dim</p> 

The training curves



2.3. CELEB dataset

CelebFaces Attributes Dataset (CelebA) is a large-scale face attributes dataset with more than **200K** celebrity images.



2.3.1. The effect of color conditioning

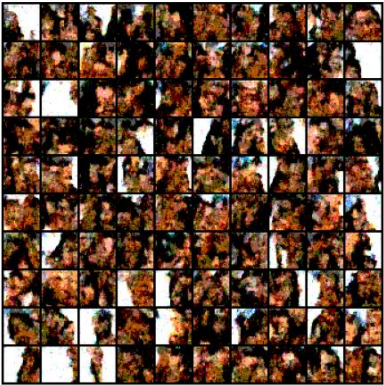

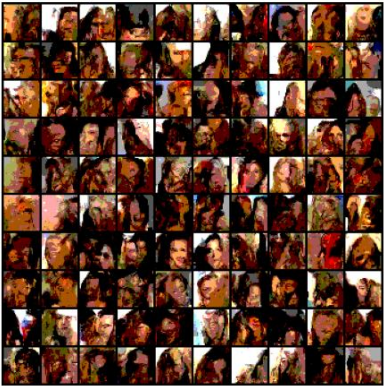

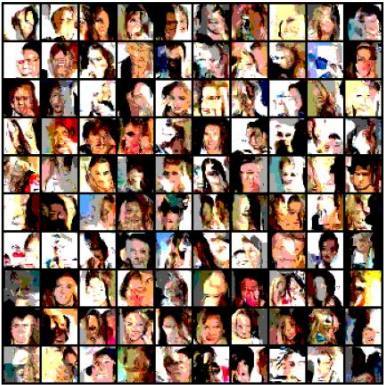



We training our models with 50 epochs and using layer normalizing with skip connectiong

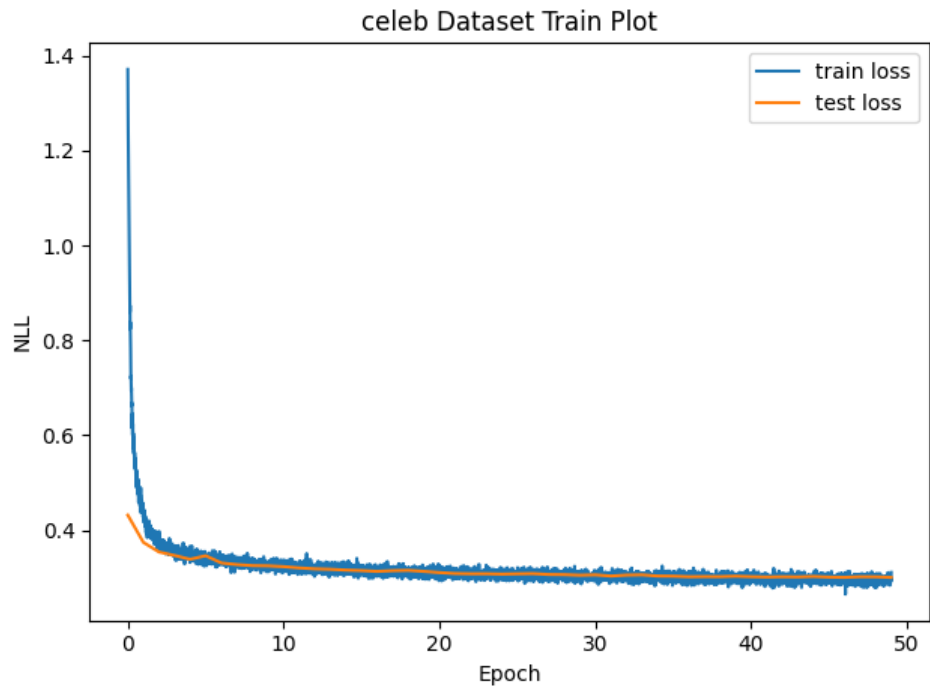
With color conditioning	Without color conditioning
<p>Samples after 0 with 0.6033 bits/dim</p> 	<p>Samples after 0 with 0.7019 bits/dim</p> 
<p>Samples after 10 with 0.4673 bits/dim</p> 	<p>Samples after 10 with 0.5786 bits/dim</p> 
<p>Samples after 20 with 0.4492 bits/dim</p> 	<p>Samples after 20 with 0.5636 bits/dim</p> 



2.3.2. The effect of the number of resnet block

Using layer normalizing, 50 epochs and color conditioning with skip connection.

8 resnet block	10 resnet block
<p data-bbox="355 237 635 257">Samples after 0 with 0.6033 bits/dim</p> 	<p data-bbox="962 237 1241 257">Samples after 0 with 0.6231 bits/dim</p> 
<p data-bbox="352 663 636 683">Samples after 10 with 0.4673 bits/dim</p> 	<p data-bbox="959 663 1244 683">Samples after 10 with 0.4665 bits/dim</p> 
<p data-bbox="352 1088 636 1108">Samples after 20 with 0.4492 bits/dim</p> 	<p data-bbox="959 1088 1244 1108">Samples after 20 with 0.4484 bits/dim</p> 
<p data-bbox="352 1514 636 1534">Samples after 30 with 0.4413 bits/dim</p> 	<p data-bbox="959 1514 1244 1534">Samples after 30 with 0.4415 bits/dim</p> 



Training curve of training with 10 resnet blocks

2.3.3. The effect of skip connection

Using layer normalizing, 50 epochs and color conditioning with 8 resnet block.

Skipped	No skipped
<p>Samples after 0 with 0.6033 bits/dim</p> 	<p>Samples after 0 with 1.6137 bits/dim</p> 
<p>Samples after 10 with 0.4673 bits/dim</p> 	<p>Samples after 10 with 0.7486 bits/dim</p> 
<p>Samples after 20 with 0.4492 bits/dim</p> 	<p>Samples after 20 with 0.6843 bits/dim</p> 

Samples after 30 with 0.4413 bits/dim



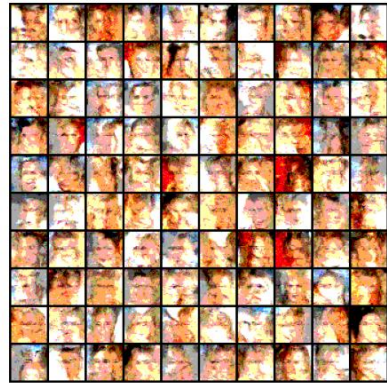
Samples after 30 with 0.6744 bits/dim



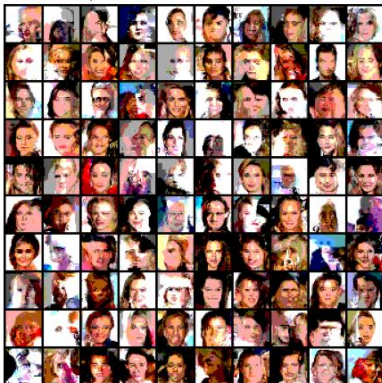
Samples after 40 with 0.4368 bits/dim



Samples after 40 with 0.6527 bits/dim

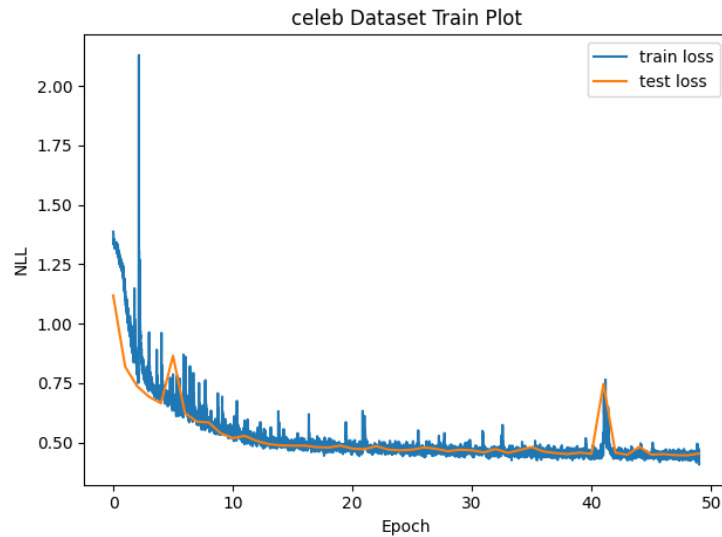


Samples at final with 0.4332 bits/dim



Samples at final with 0.6544 bits/dim





Training curve of training without skip connection

3. Conclusion

In this report, we reviewed the concept of Autoregressive models. Besides, we elaborated the pixelCNN model to know how it works and how it generates samples. Throughout experiments, we can see that pixelCNN can generate MNIST images slightly realistic. But with Celeb dataset, the generated images are still bad and not clear. We will try some advanced methods such as Gated PixelCNN, PixelCNN++ and PixelSnail to improve the quality of generated images as soon as possible.