EECE 571F: Deep Learning with Structures

Lecture 6: Autoregressive Models I (Images)

Renjie Liao

University of British Columbia

Winter, Term 1, 2023

Course Scope

- Brief Intro to Deep Learning

- Geometric Deep Learning

    - Deep Learning Models for Sets and Sequences: Deep Sets & Transformers
    - Deep Learning Models for Graphs: Message Passing & Graph Convolution GNNs
    - Group Equivariant Deep Learning

- Probabilistic Deep Learning

    - Auto-regressive models, Large Language Models (LLMs)
    - Variational Auto-Encoders (VAEs) and Generative Adversarial Networks (GANs)
    - Energy based models (EBMs)
    - Diffusion/Score based models

Course Scope

- Brief Intro to Deep Learning

- Geometric Deep Learning

  - Deep Learning Models for Sets and Sequences: Deep Sets & Transformers
  - Deep Learning Models for Graphs: Message Passing & Graph Convolution GNNs
  - Group Equivariant Deep Learning

- Probabilistic Deep Learning

  - **Auto-regressive models**, Large Language Models (LLMs)
  - Variational Auto-Encoders (VAEs) and Generative Adversarial Networks (GANs)
  - Energy based models (EBMs)
  - Diffusion/Score based models

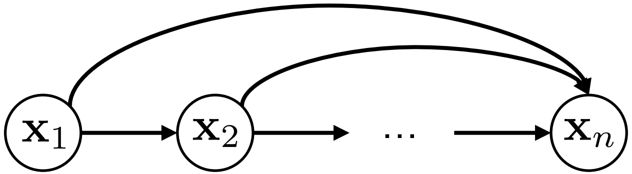
Autoregressive Models

Long history in statistics, econometrics, and signal processing.

We are a given n-dimensional data x

$$p_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_\theta(\mathbf{x}_i|\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}) = \prod_{i=1}^{n} p_\theta(\mathbf{x}_i|\mathbf{x}_{<i})$$

Graphical model:



PixelCNNs

Autoregressive model for images.

$$p_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_\theta(\mathbf{x}_i | \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}) = \prod_{i=1}^{n} p_\theta(\mathbf{x}_i | \mathbf{x}_{<i})$$

$\mathbf{x}_i$ is pixel value, e.g., {0, 1, …, 255}

n = height × width

Every term $p_\theta(\mathbf{x}_i | \mathbf{x}_{<i})$ is modeled by the same CNN (softmax readout)

PixelCNNs

$$p_\theta(\mathbf{x}_i | \mathbf{x}_{<i})$$

Conditioned on all pixels that are top-left!

One can also vectorize an image as a sequence and use RNNs to build the autoregressive model, e.g., PixelRNNs [2].

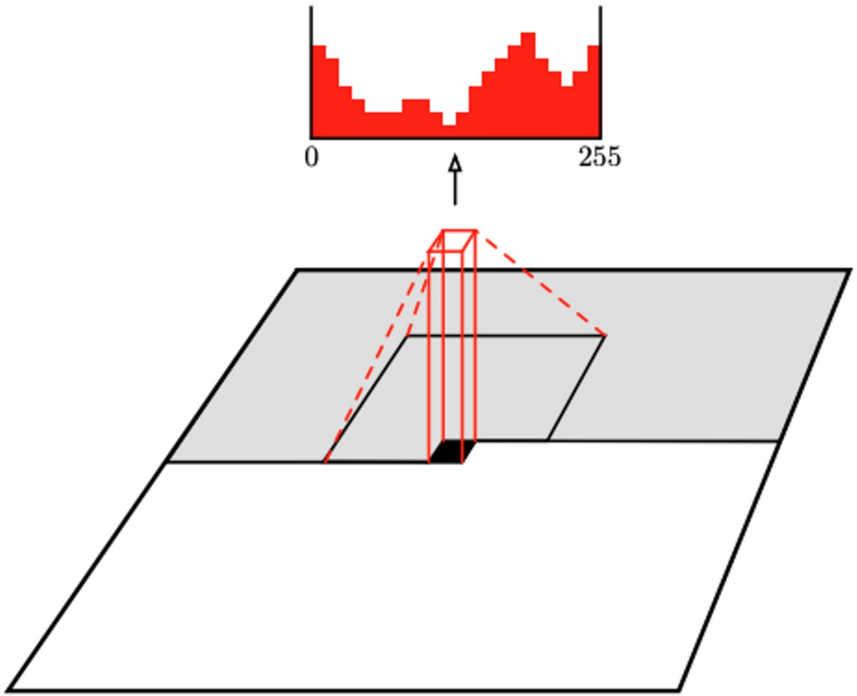


Image Credit: [1]

What About Color Images?

Autoregressive conditioning again along channels:

$$p_\theta(\mathbf{x}_R, \mathbf{x}_G, \mathbf{x}_B) = \prod_i^n p_\theta(x_{R,i} | \mathbf{x}_{R,<i}, \mathbf{x}_{G,<i}, \mathbf{x}_{B,<i}) \times$$

$$p_\theta(x_{G,i} | x_{R,i}, \mathbf{x}_{R,<i}, \mathbf{x}_{G,<i}, \mathbf{x}_{B,<i}) \times$$

$$p_\theta(x_{B,i} | x_{G,i}, x_{R,i}, \mathbf{x}_{R,<i}, \mathbf{x}_{G,<i}, \mathbf{x}_{B,<i})$$

How to Implement?

1. Mask Input
2. Convolution

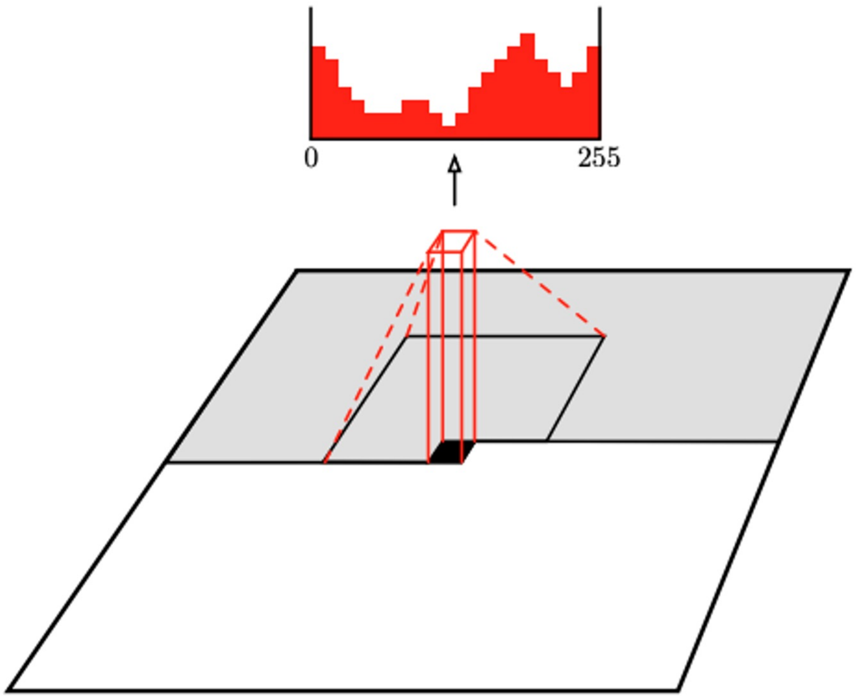


Image Credit: [1]

How to Implement?

For each image, we need $H \times W$ masks and convolutions to compute the likelihood!

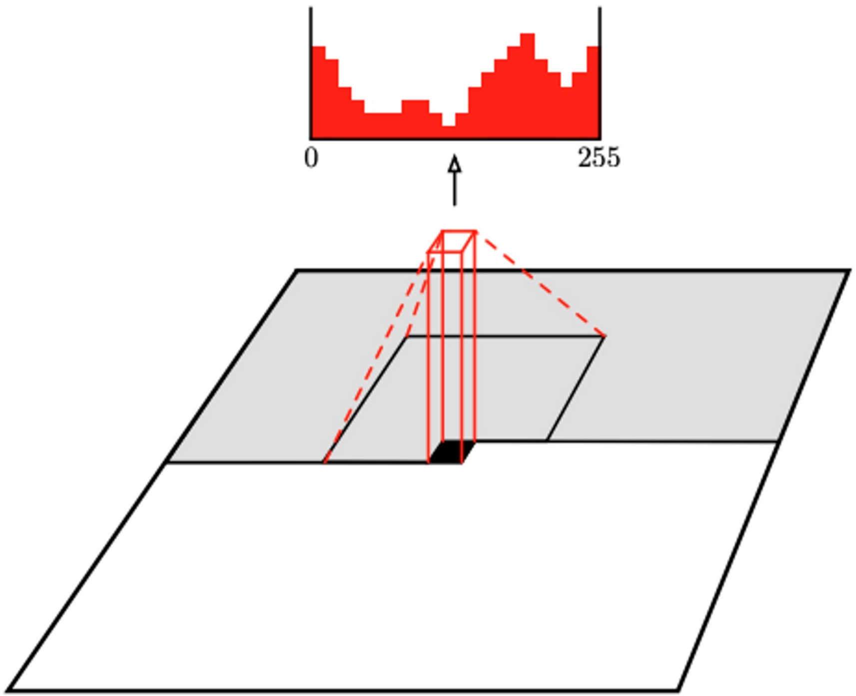
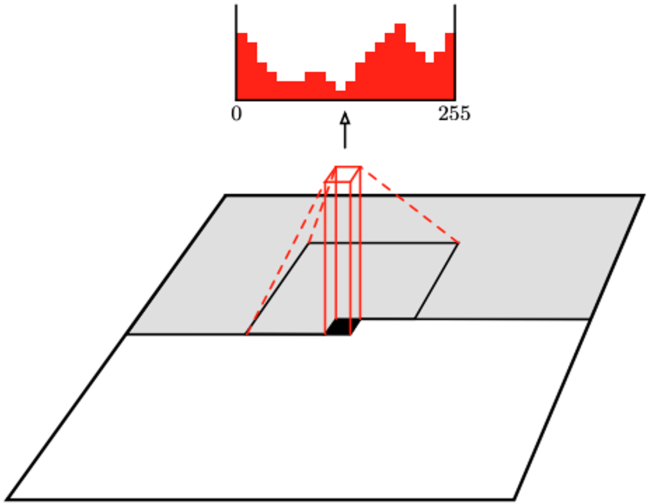


Image Credit: [1]

Solutions in PixelCNNs

Masked Filter + Smart Stack of Regular Convolutions!

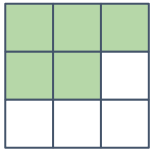


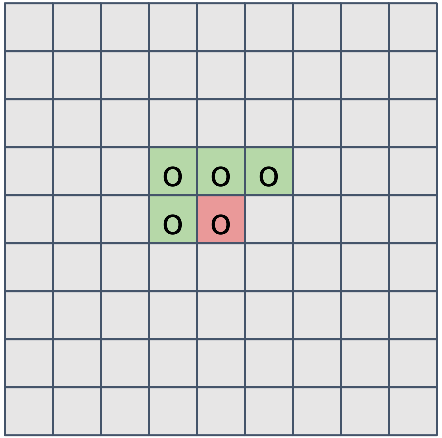
1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0

Image Credit: [1]

Masked Filter

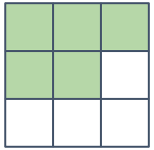
Masked 3×3 filter

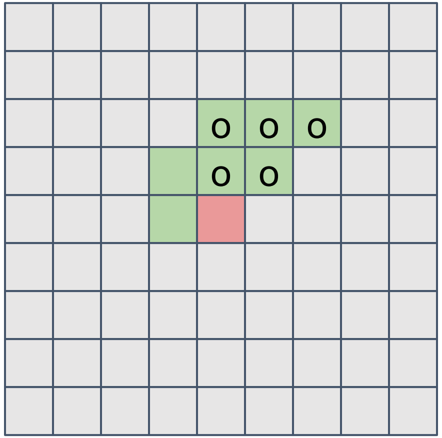




Masked Filter

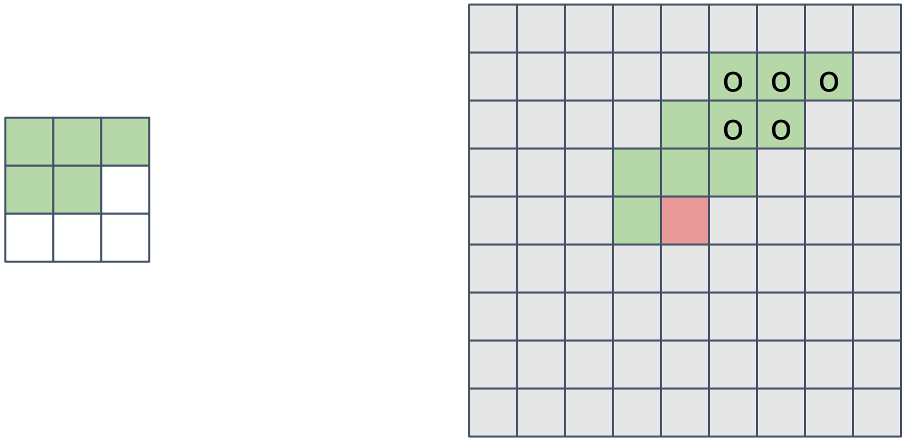
Masked 3×3 filter





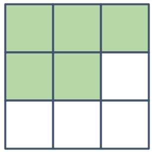
Masked Filter

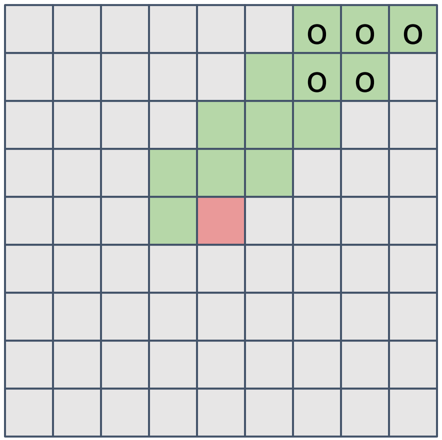
Masked 3×3 filter



Masked Filter

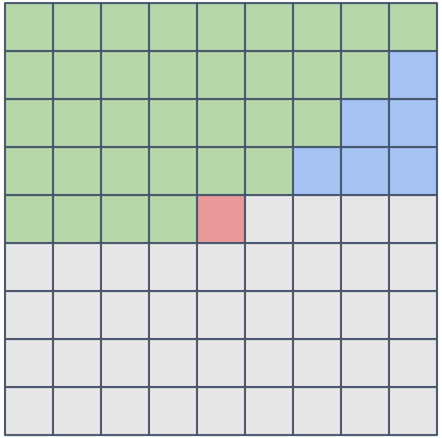
Masked 3×3 filter





Masked Filter

Masked 3×3 filter



Naively applying masked filter causes blind spots (blue area)!

How to Resolve Blind Spots?

Applying two stacks of masked convolutions!

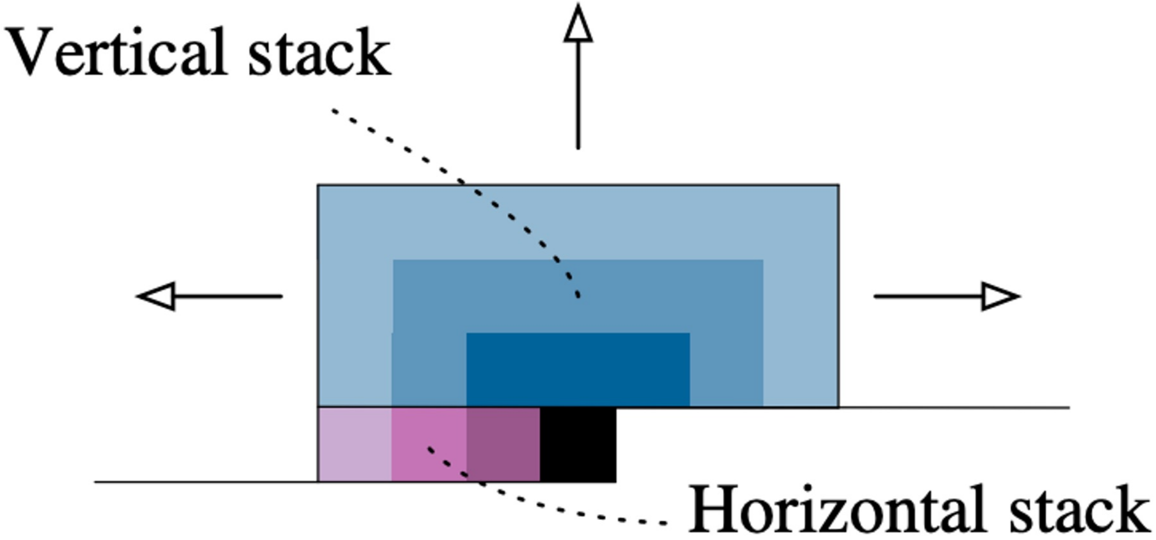
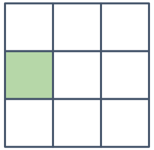


Image Credit: [1]

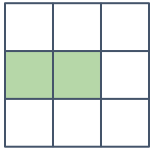
How to Resolve Blind Spots?

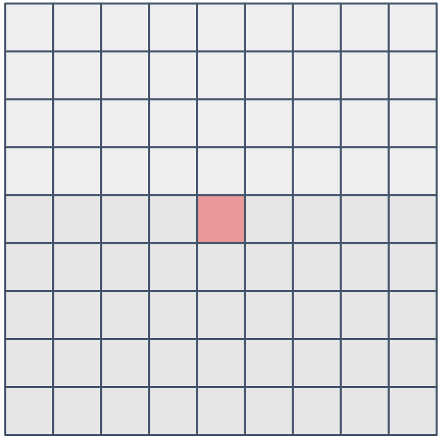
Horizontal Stack (Implemented by masked 2D convolution)

Horizontal Mask 1



Horizontal Mask 2

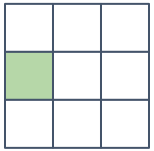


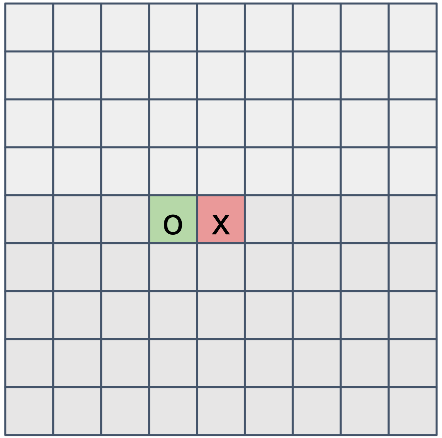


How to Resolve Blind Spots?

Horizontal Stack (Implemented by masked 2D convolution)

Horizontal Mask 1



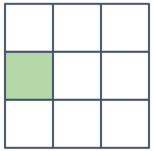


Mask 1 → Mask 2 → ... → Mask 2

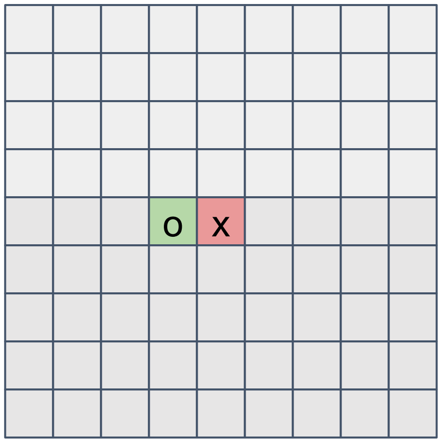
How to Resolve Blind Spots?

Horizontal Stack (Implemented by masked 2D convolution)

Horizontal Mask 1



Avoid using information at current location!



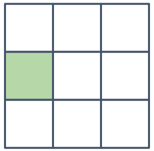
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

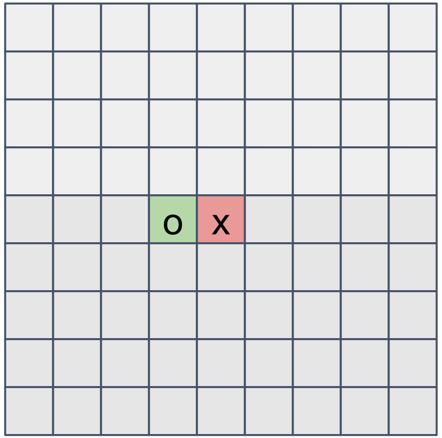
Horizontal Stack (Implemented by masked 2D convolution)

Note that the same masked filter is convolved everywhere!

Horizontal Mask 1



Avoid using information at current location!

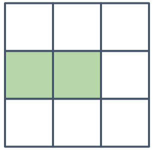


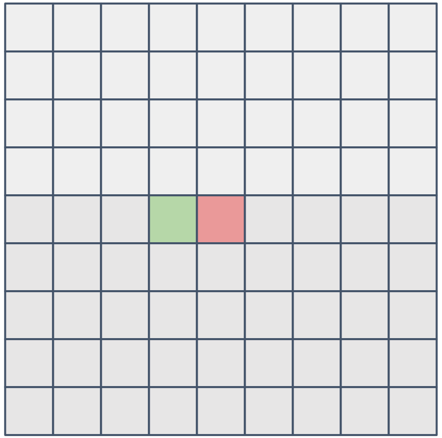
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Horizontal Stack (Implemented by masked 2D convolution)

Horizontal Mask 2



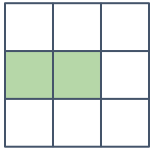


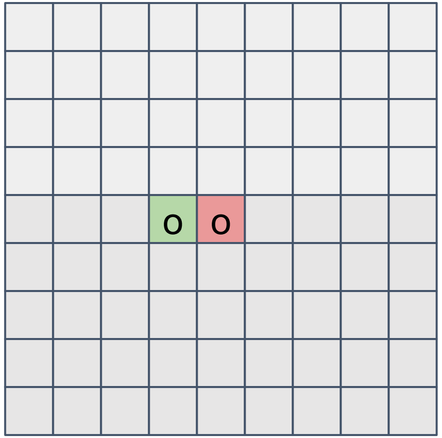
Mask 1 → **Mask 2** → ... → Mask 2

How to Resolve Blind Spots?

Horizontal Stack (Implemented by masked 2D convolution)

Horizontal Mask 2



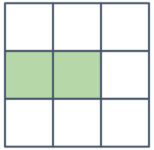


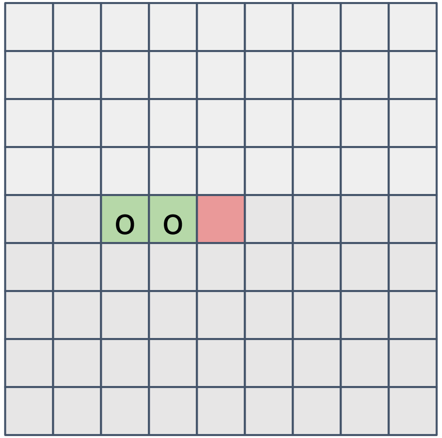
Mask 1 → **Mask 2** → ... → Mask 2

How to Resolve Blind Spots?

Horizontal Stack (Implemented by masked 2D convolution)

Horizontal Mask 2



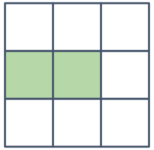


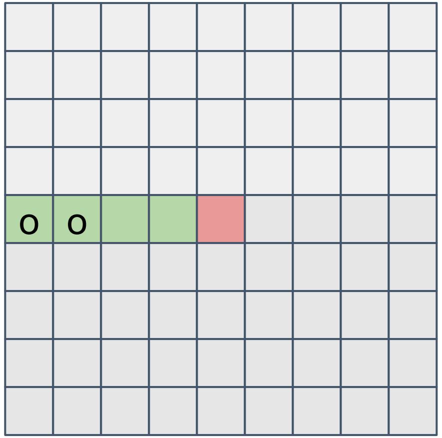
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Horizontal Stack (Implemented by masked 2D convolution)

Horizontal Mask 2



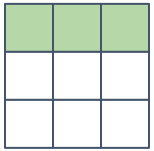


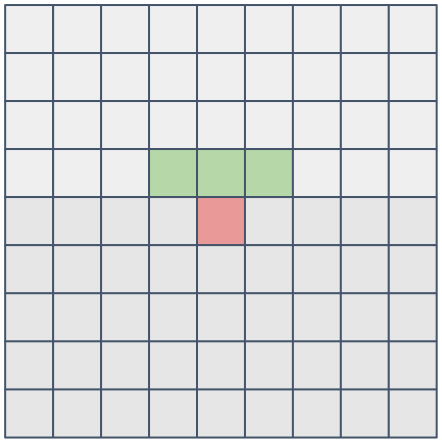
Mask 1 → Mask 2 → ... → **Mask 2**

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1

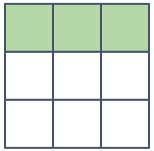




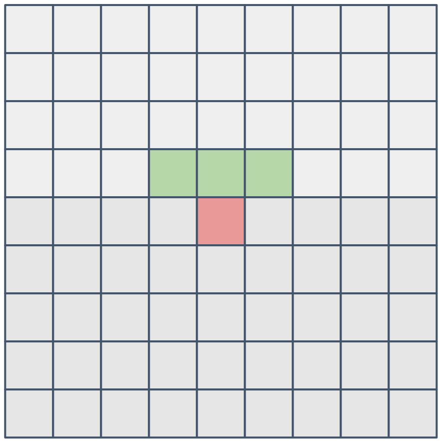
How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



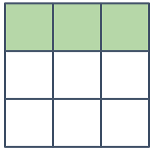
Avoid using information at current location!



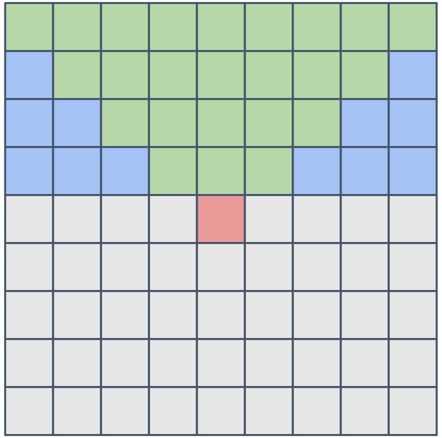
How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



Applying vertical masked filter causes blind spots (blue area) too!

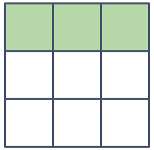


How to Resolve Blind Spots?

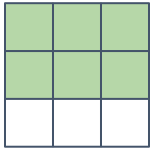
Vertical Stack (Implemented by masked 2D convolution)

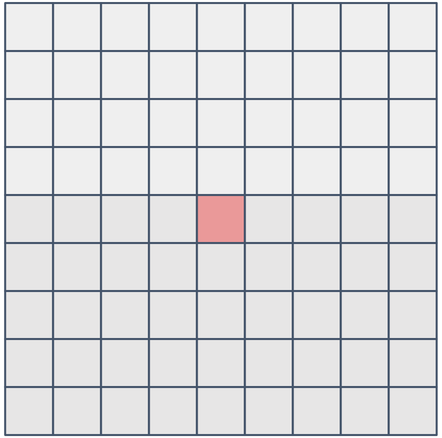
We again use two masked filters to remove blind spots!

Vertical Mask 1



Vertical Mask 2

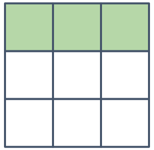


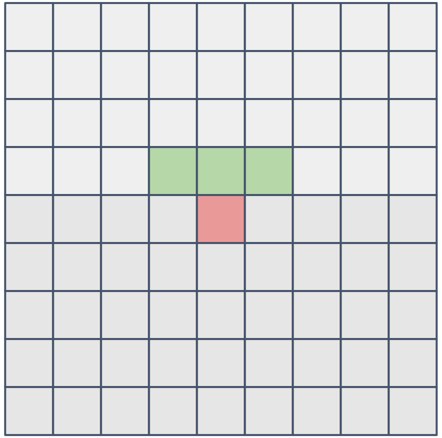


How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1





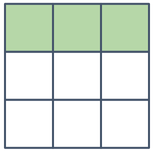
Mask 1 → Mask 2 → ... → Mask 2

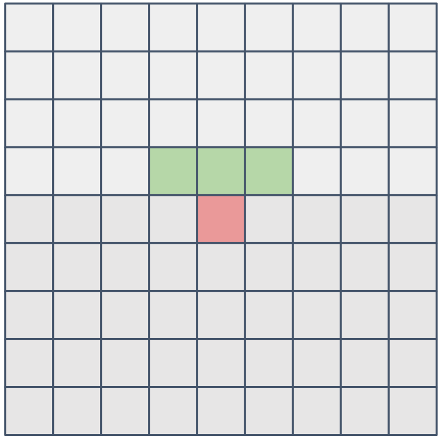
How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Note that the same masked filter is convolved everywhere!

Vertical Mask 1



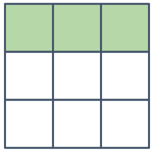


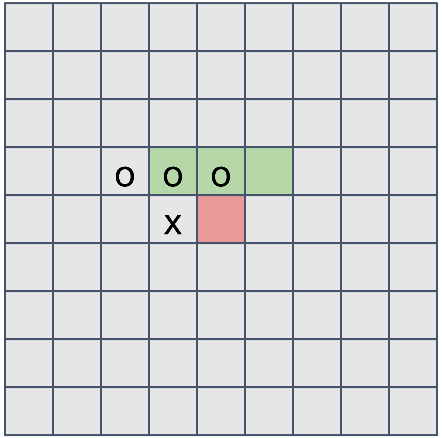
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



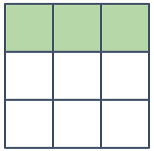


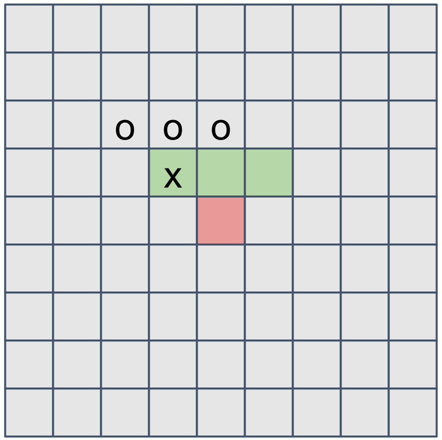
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



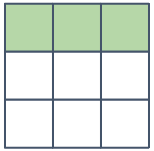


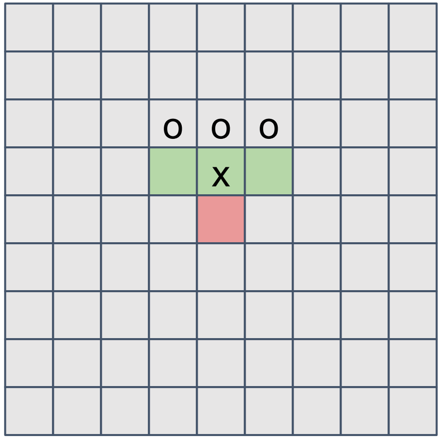
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



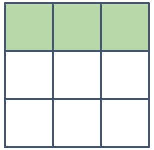


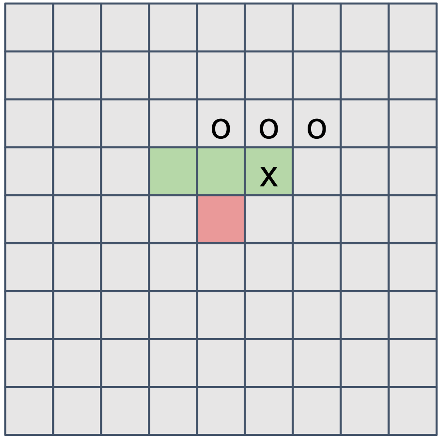
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



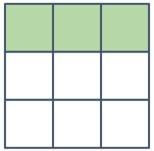


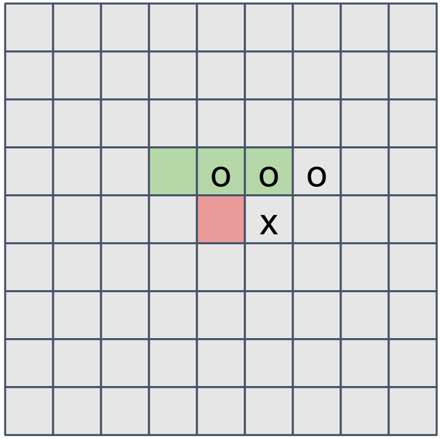
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



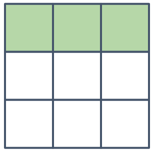


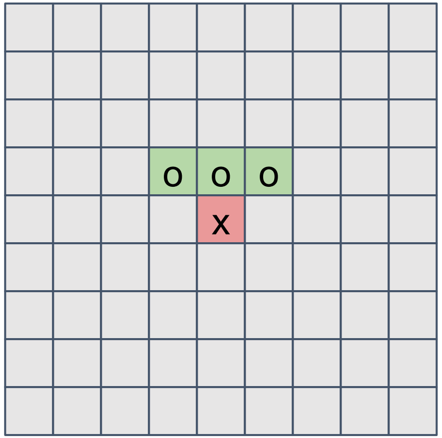
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 1



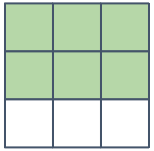


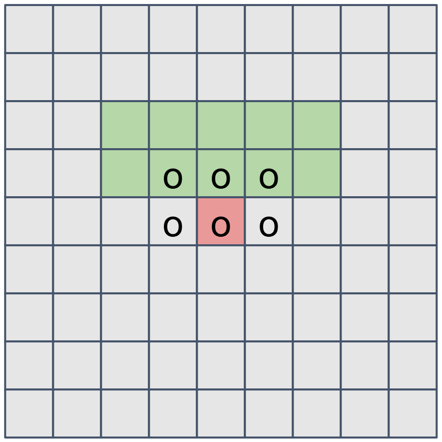
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 2



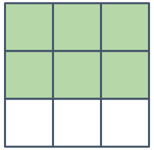


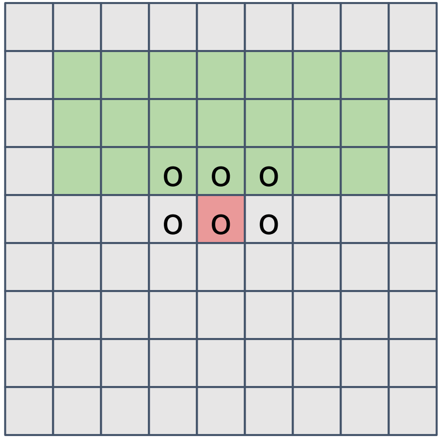
Mask 1 → **Mask 2** → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 2



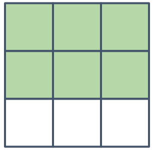


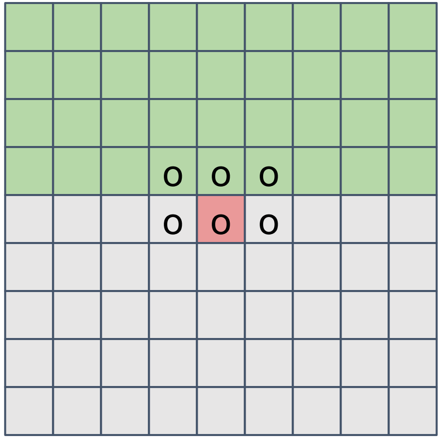
Mask 1 → Mask 2 → ... → Mask 2

How to Resolve Blind Spots?

Vertical Stack (Implemented by masked 2D convolution)

Vertical Mask 2



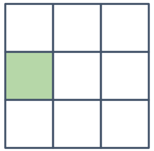


Mask 1 → Mask 2 → ... → **Mask 2**

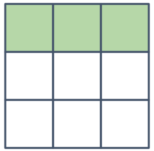
How to Resolve Blind Spots?

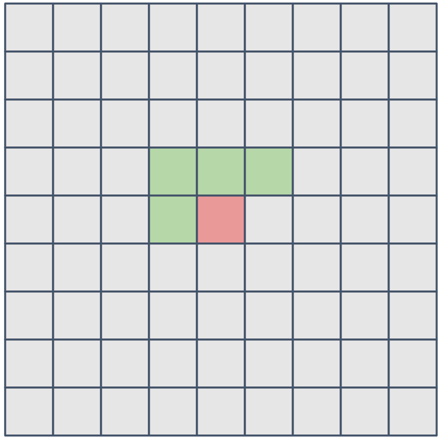
Combine Horizontal and Vertical Stacks

Horizontal Mask 1



Vertical Mask 1



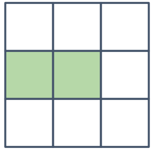


Layer 1 → Layer 2 → ... → Layer L

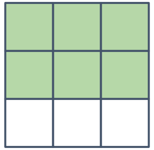
How to Resolve Blind Spots?

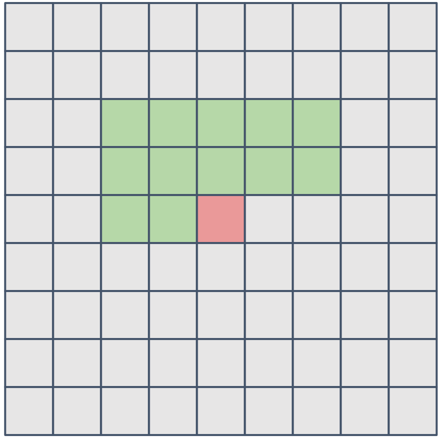
Combine Horizontal and Vertical Stacks

Horizontal Mask 2



Vertical Mask 2



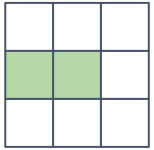


Layer 1 → **Layer 2** → ... → Layer L

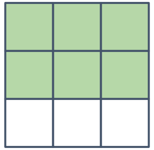
How to Resolve Blind Spots?

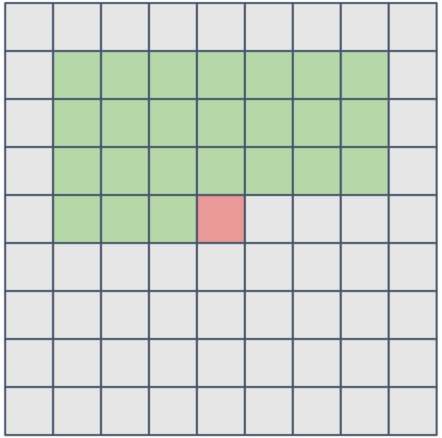
Combine Horizontal and Vertical Stacks

Horizontal Mask 2



Vertical Mask 2



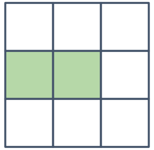


Layer 1 → Layer 2 → ... → Layer L

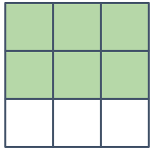
How to Resolve Blind Spots?

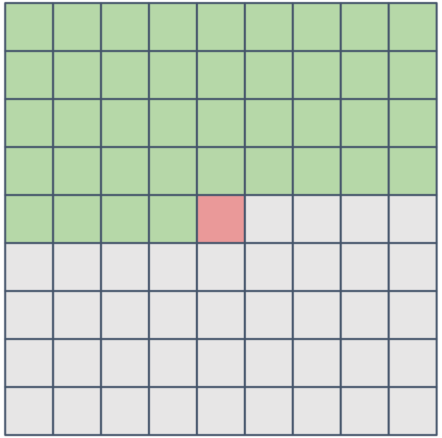
Combine Horizontal and Vertical Stacks

Horizontal Mask 2



Vertical Mask 2



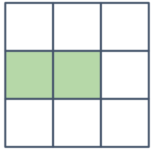


Layer 1 → Layer 2 → ... → **Layer L**

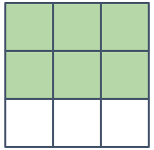
How to Resolve Blind Spots?

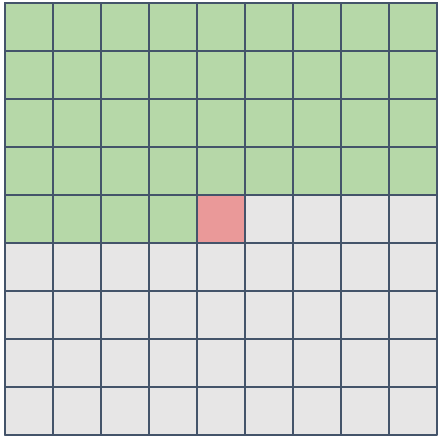
Combine Horizontal and Vertical Stacks

Horizontal Mask 2



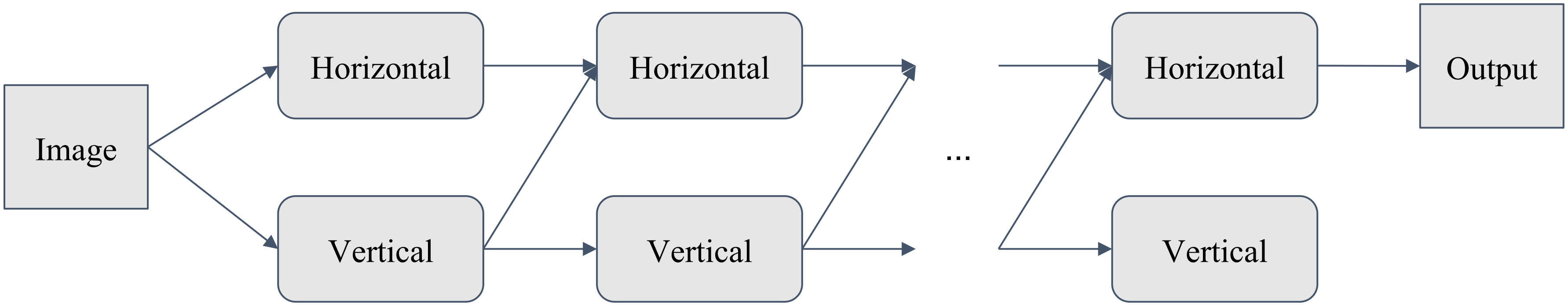
Vertical Mask 2





Layer 1 → Layer 2 → ... → **Layer L**

PixelCNN Architecture



PixelCNN Architecture

Gated Convolutions $\mathbf{y} = \tanh\left(\mathbf{W}_f \mathbf{x}\right) \odot \sigma\left(\mathbf{W}_g \mathbf{x}\right)$

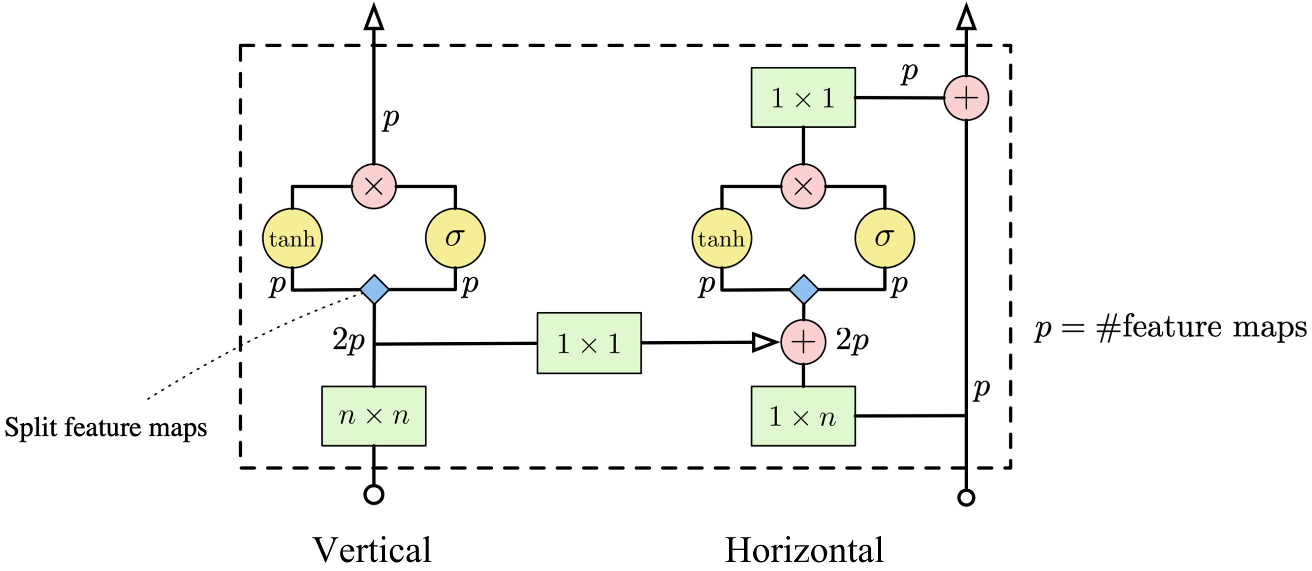


Image Credit: [1]

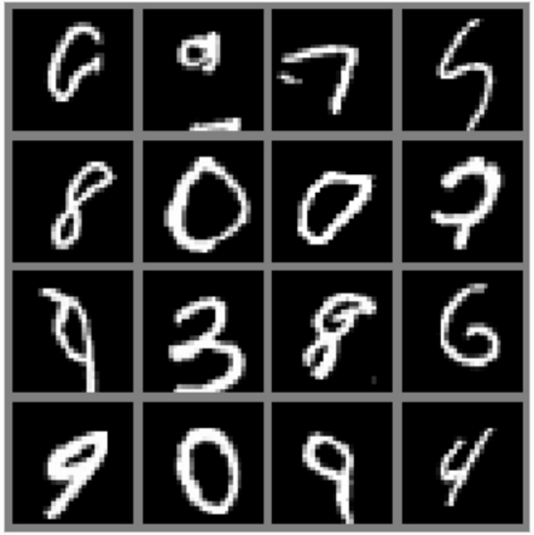
PixelCNN Performances

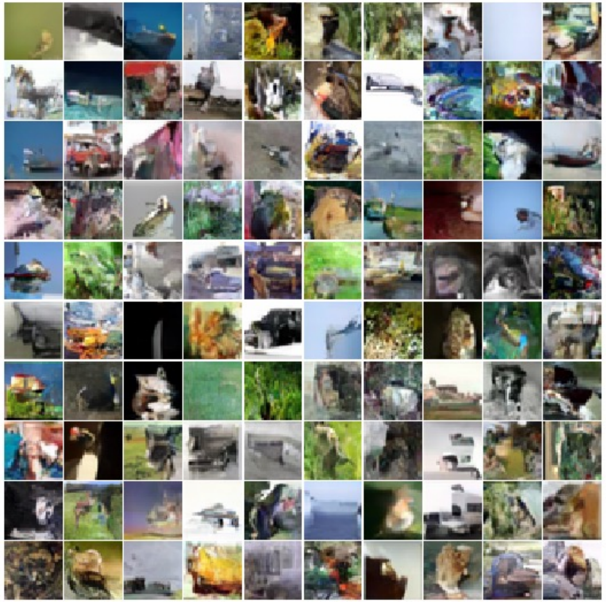
CIFAR 10:

Model	**NLL Test (Train)**
Uniform Distribution: [30]	8.00
Multivariate Gaussian: [30]	4.70
NICE: [4]	4.48
Deep Diffusion: [24]	4.20
DRAW: [9]	4.13
Deep GMMs: [31, 29]	4.00
Conv DRAW: [8]	3.58 (3.57)
RIDE: [26, 30]	3.47
PixelCNN: [30]	3.14 (3.08)
PixelRNN: [30]	3.00 (2.93)
Gated PixelCNN:	3.03 (2.90)

PixelCNN Performances

Unconditional Generation:





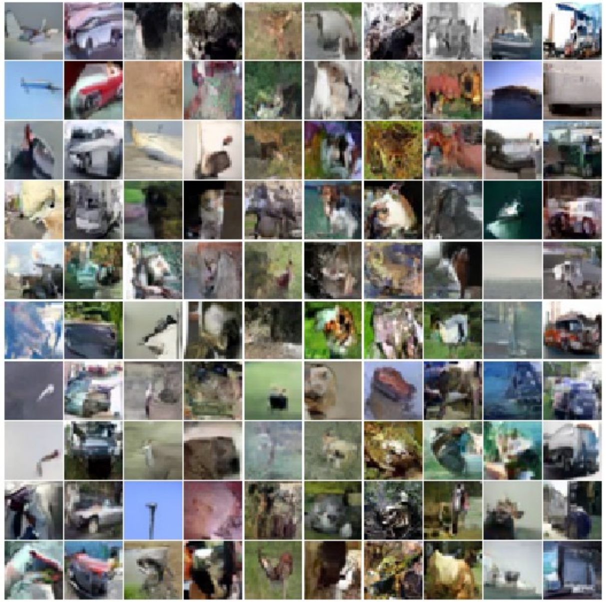


Image Credit: [1,3,4]

PixelCNN Performances

Conditional Generation (Image Completion):

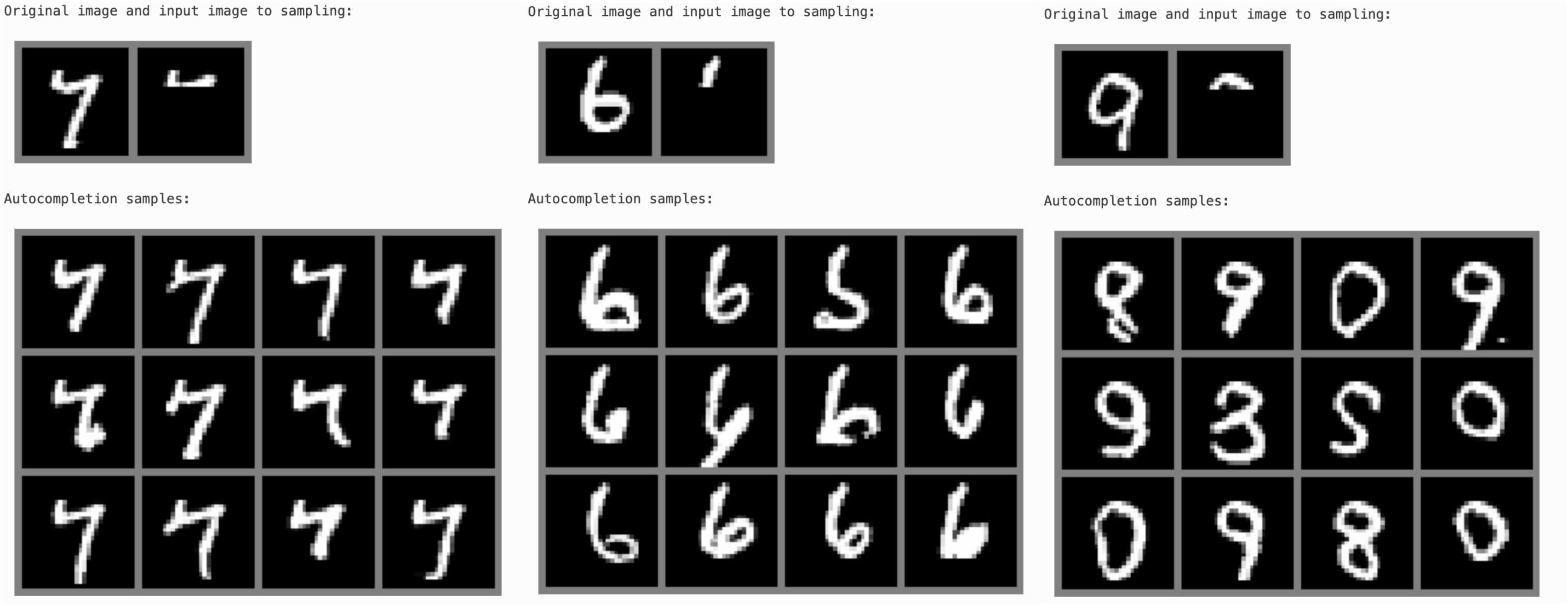


Image Credit: [4]

PixelCNN Performances

Conditional Generation (Class Label):

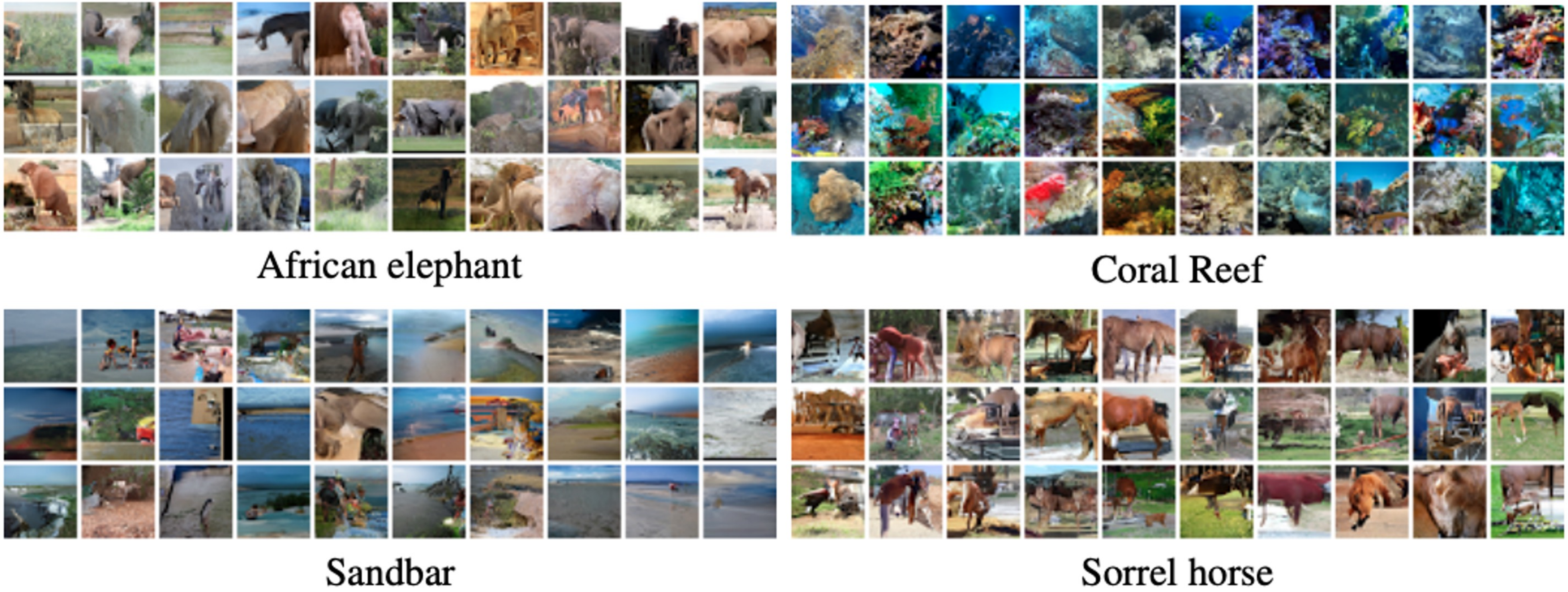


Image Credit: [1]

Pros vs Cons

+ **Parallel Training**

One forward pass to compute losses at all locations (i.e., all conditional probabilities)!

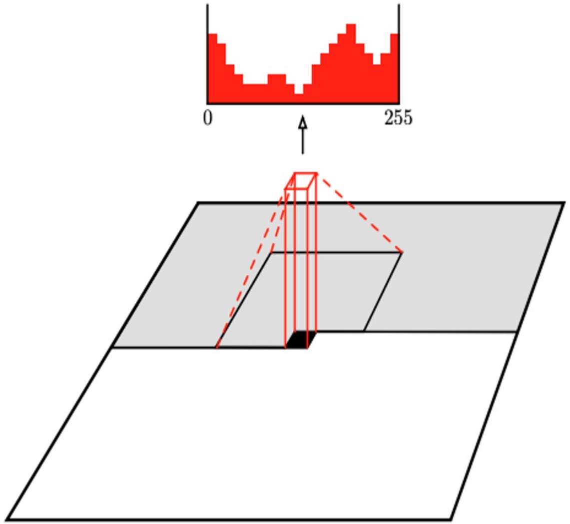


Image Credit: [1]

Pros vs Cons

+ **Parallel Training**

One forward pass to compute losses at all locations (i.e., all conditional probabilities)!

+ **Strong Performances**

PixelCNN++ [3] further improves performances by:

1. Softmax → discretized mixture of logistic distributions
2. Downsample & upsample, dropout, skip connections, etc.

$$\nu \sim \sum_{i=1}^{K} \pi_i \mathrm{logistic}(\mu_i, s_i)$$

$$P(x|\pi,\mu,s) = \sum_{i=1}^{K} \pi_i \left[\sigma((x+0.5-\mu_i)/s_i) - \sigma((x-0.5-\mu_i)/s_i)\right],$$

Pixel intensity distribution

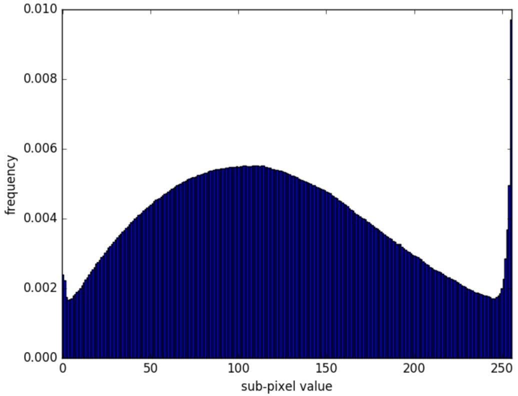


Image Credit: [3]

Pros vs Cons

+ **Parallel Training**

One forward pass to compute losses at all locations (i.e., all conditional probabilities)!

+ **Strong Performances**

PixelCNN++ [3] further improves performances by:

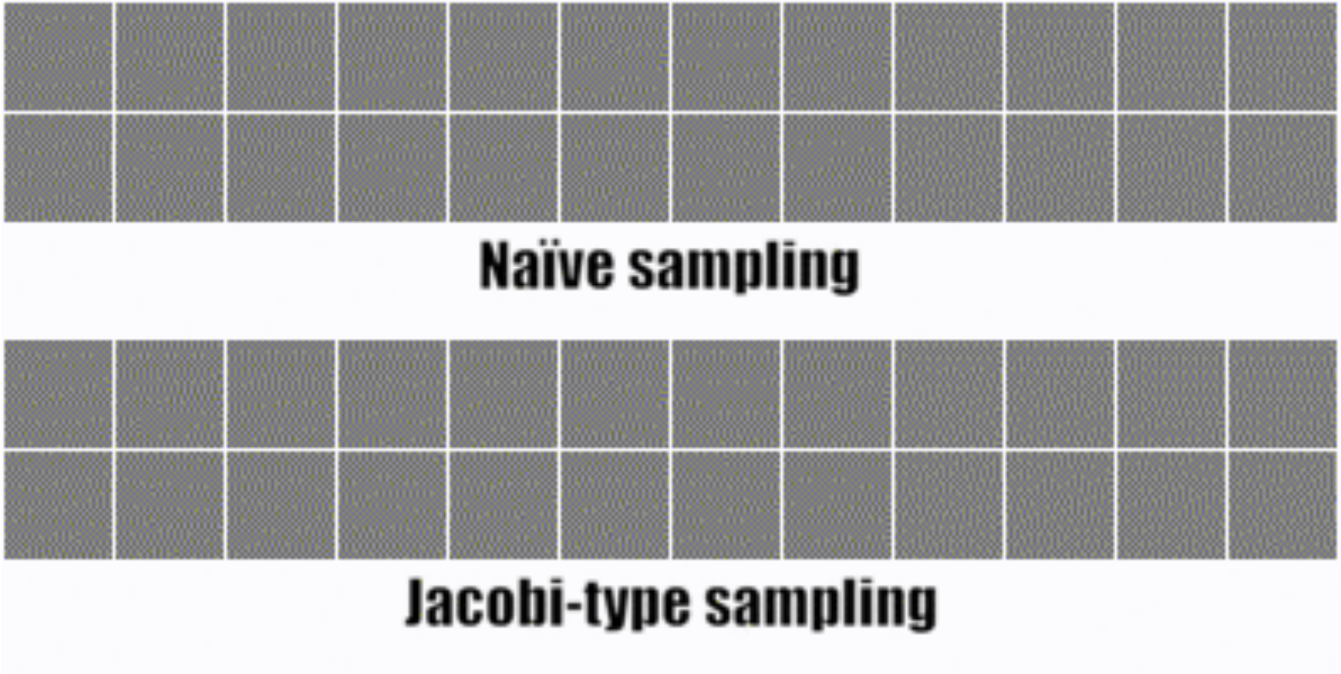
1. Softmax → discretized mixture of logistic distributions
2. Downsample & upsample, dropout, skip connections, etc.

- Slow Sampling

This is due to the sequential nature of autoregressive sampling.
It could be further improved by methods, e.g., [5].

Pros vs Cons

Parallel Nonlinear Equation Solving [5] speeds up auto-regressive sampling:



References

[1] van den Oord, A., et al. “Conditional Image Generation with PixelCNN Decoders.” In Advances in Neural Information Processing Systems 29, pp. 4790–4798 (2016).

[2] van den Oord, A., et al. “Pixel Recurrent Neural Networks.” arXiv preprint arXiv:1601.06759 (2016).

[3] Salimans, Tim, et al. “PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications.” arXiv preprint arXiv:1701.05517 (2017).

[4] https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial12/Autoregressive_Image_Modeling.html

[5] Song, Y., Meng, C., Liao, R. and Ermon, S., 2021, July. Accelerating feedforward computation via parallel nonlinear equation solving. In International Conference on Machine Learning (pp. 9791-9800). PMLR.

Questions?