Pixel Recurrent Neural Networks

Sanchit Vohra (sv4)

Outline

- Problem Statement and Motivation
- Related Ideas in Generative Modelling
- Main Ideas
  - PixelRNN model
  - RNN, LSTM, and convolutional LSTM
  - Row LSTM, Diagonal BiLSTM
  - Residual connections, masked convolutions
  - Multiscale PixelRNN
  - Receptive field blind-spot and horizontal and vertical stack
  - Gated Convolutional Layers
  - Conditioning on latent space
- Empirical Results

Problem Statement and Motivation

- Model distribution of high-dimensional data
- Expressive, tractable and scalable
- Papers focus on images: high-dimensional and highly structured

(a) Original face (b) Masked input (c) Our result

Related Ideas in Generative Modelling

- Stochastic latent variable models (VAE) doesn't compute proper probability distributions + reconstruction loss leads to blurry generations
- Adversarial learning models (GAN) produce sharper outputs but are unstable during training
- Techniques that model joint distributions as product of conditionals (NADE) lack highly expressive sequence models (RNN)
- Modifying VAE to output conditional probabilities (MADE) parralizes inference but doesn't improve model performance (scalability)

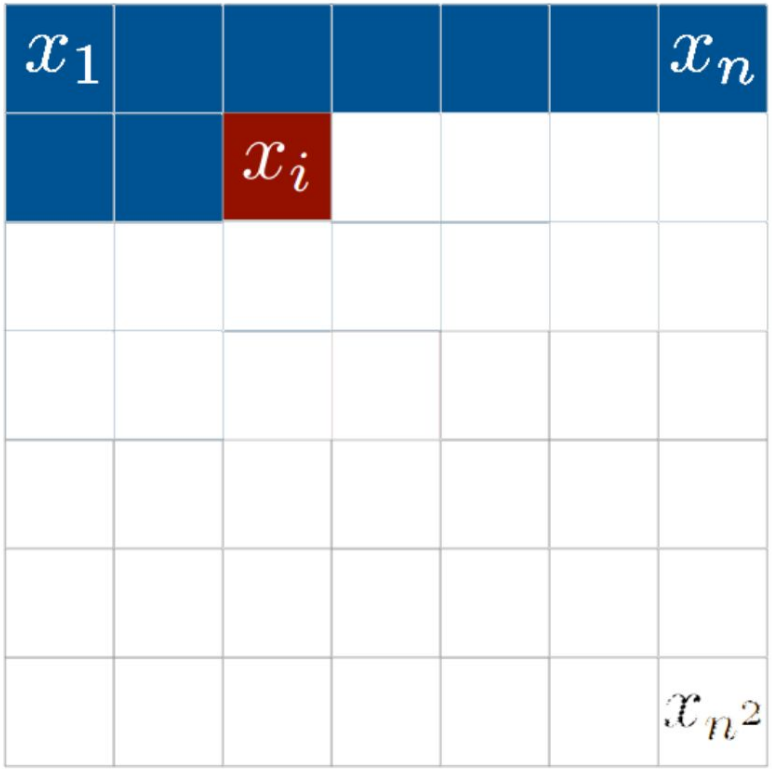
PixelRNN Model

- Model joint probability distribution as a product of conditionals
- In PixelRNN model, each pixel is conditionally dependent on previous pixels from top to bottom and left to right
- Additionally, each channel RGB is conditionally dependent on previous channel

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

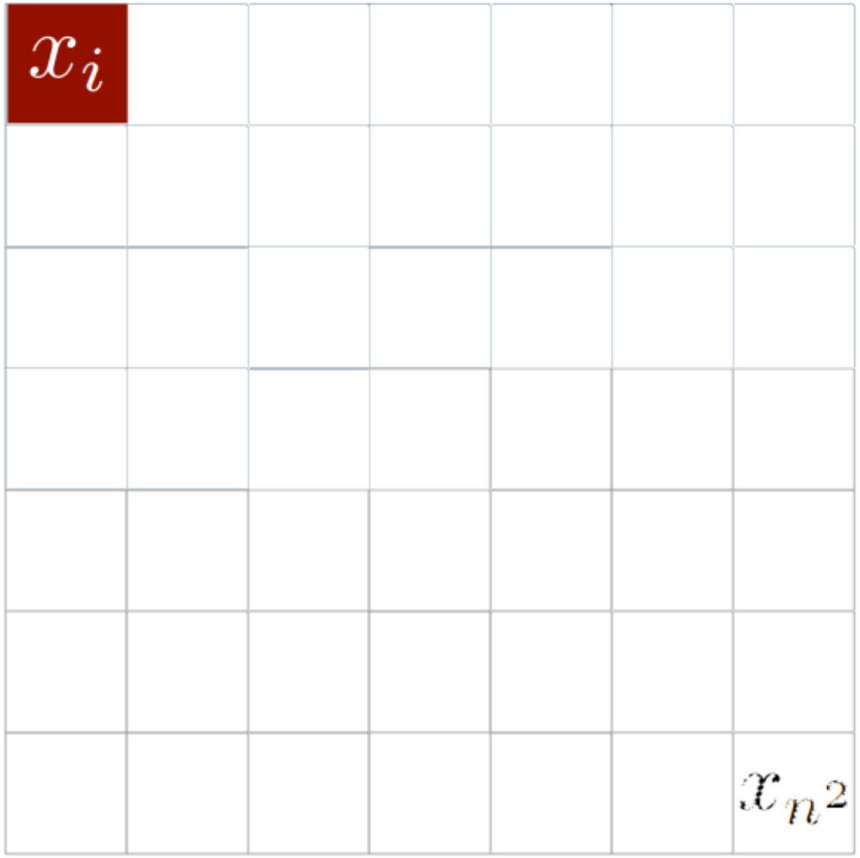
$$p(x_{i,R}|\mathbf{x}_{<i})p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R})p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G})$$

PixelRNN Model

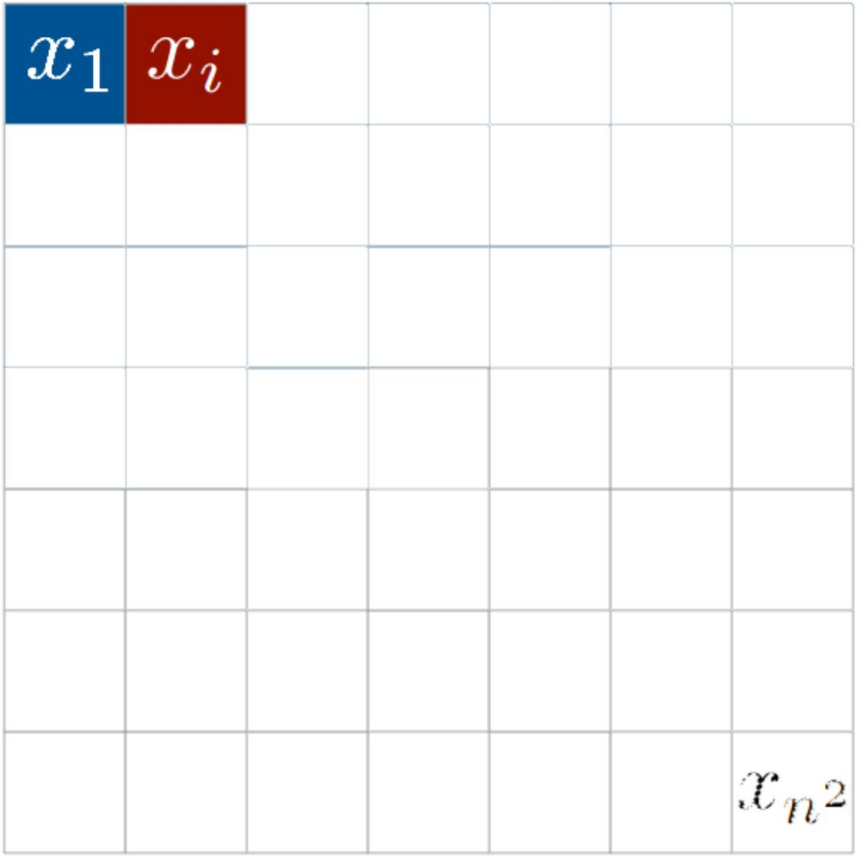


$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, \ldots, x_{i-1})$$

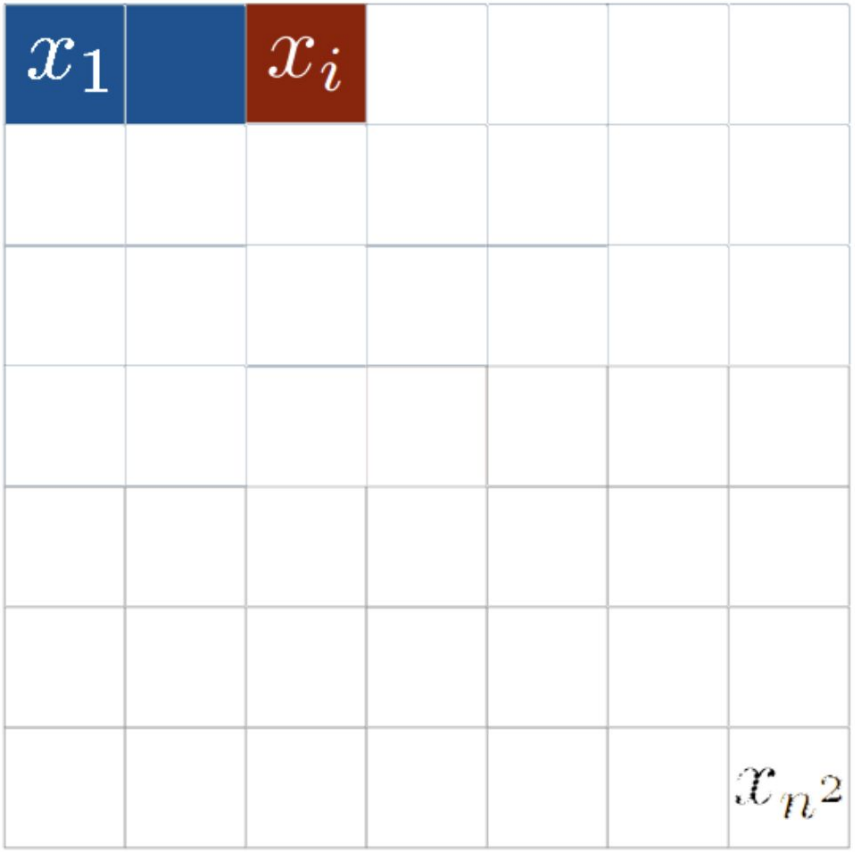
$$p(x_{i,R}|\mathbf{x}_{<i})p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R})p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G})$$



PixelRNN Generation Process



PixelRNN Generation Process

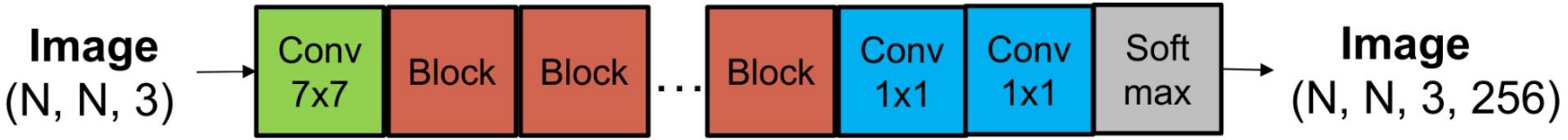


PixelRNN Generation Process

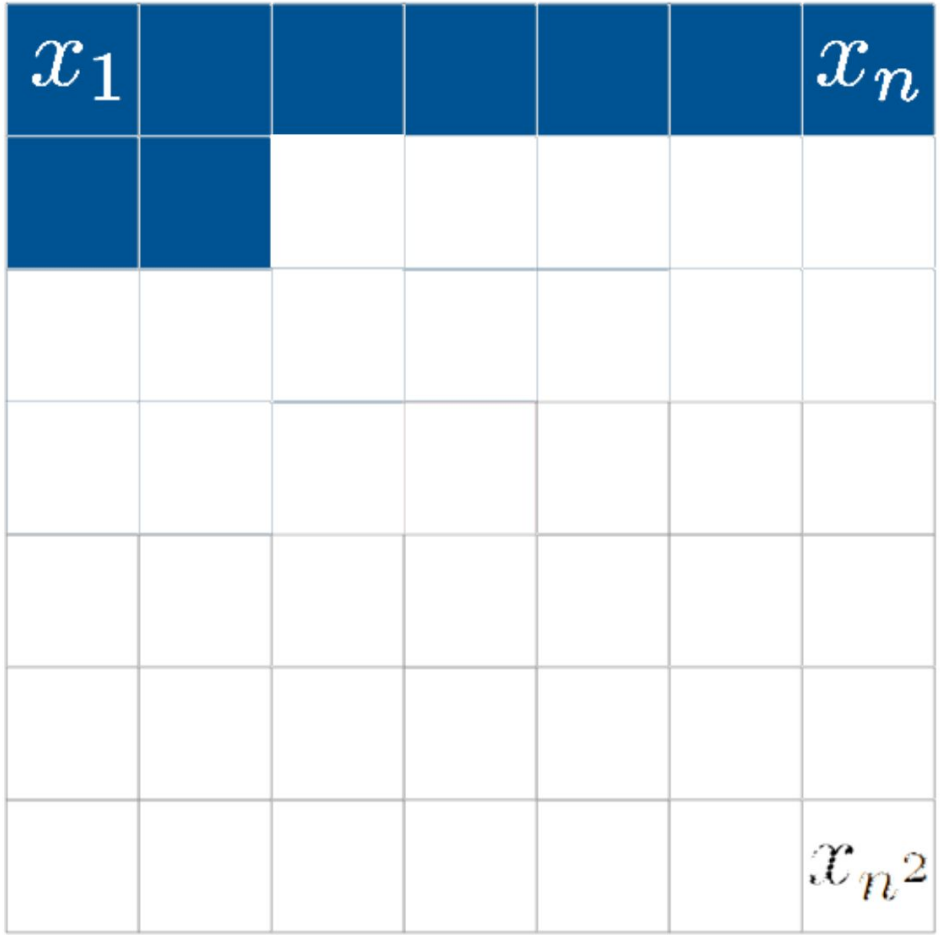
Training vs generation

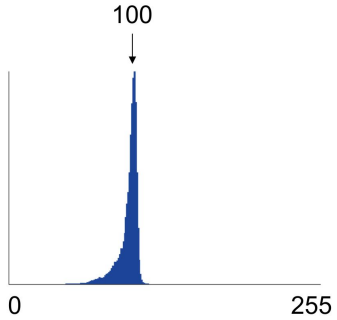
- Generation is extremely slow because of generating pixel-by-pixel and channel-by-channel sequentially
- However, training can be done quickly in parallel since all the conditional inputs are already present

PixelRNN Model



- Always start with 7x7 masked convolution
- The “Block” in the diagram can be convolutions, RowLSTM, Diagonal BiLSTM
- All the above blocks contain residual connections
- Final output is 256 value softmax giving probability for each channel

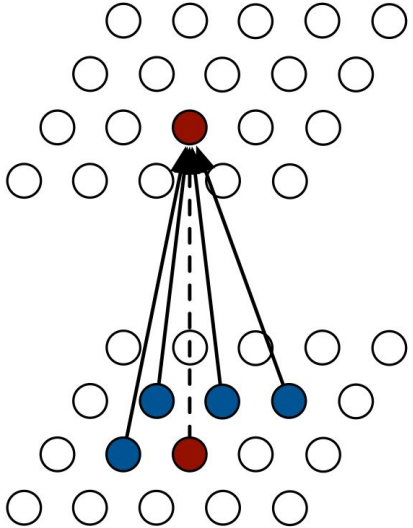




- Output layer has n x n x 3 x 256 shape
- Each 256 channel is normalized via softmax and represents multinomial probability distribution for pixel values
- Experimentally, this produced better results than continuous distribution

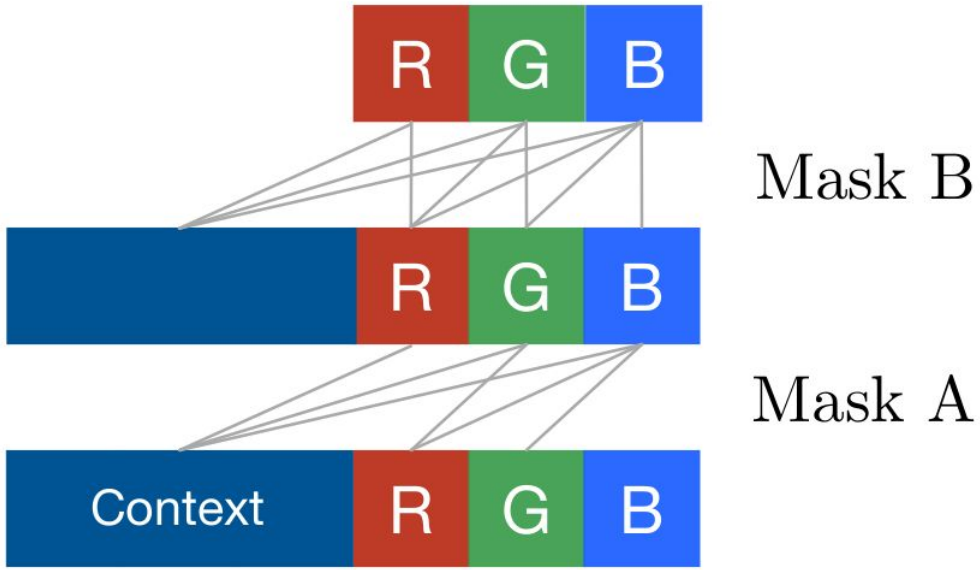
PixelCNN

- In PixelCNN each block is Masked 3x3 convolution
- Mask exists to protect (RGB) inter-channel dependencies
- Advantage: Parallel computation
- Disadvantage: Small Receptive Field



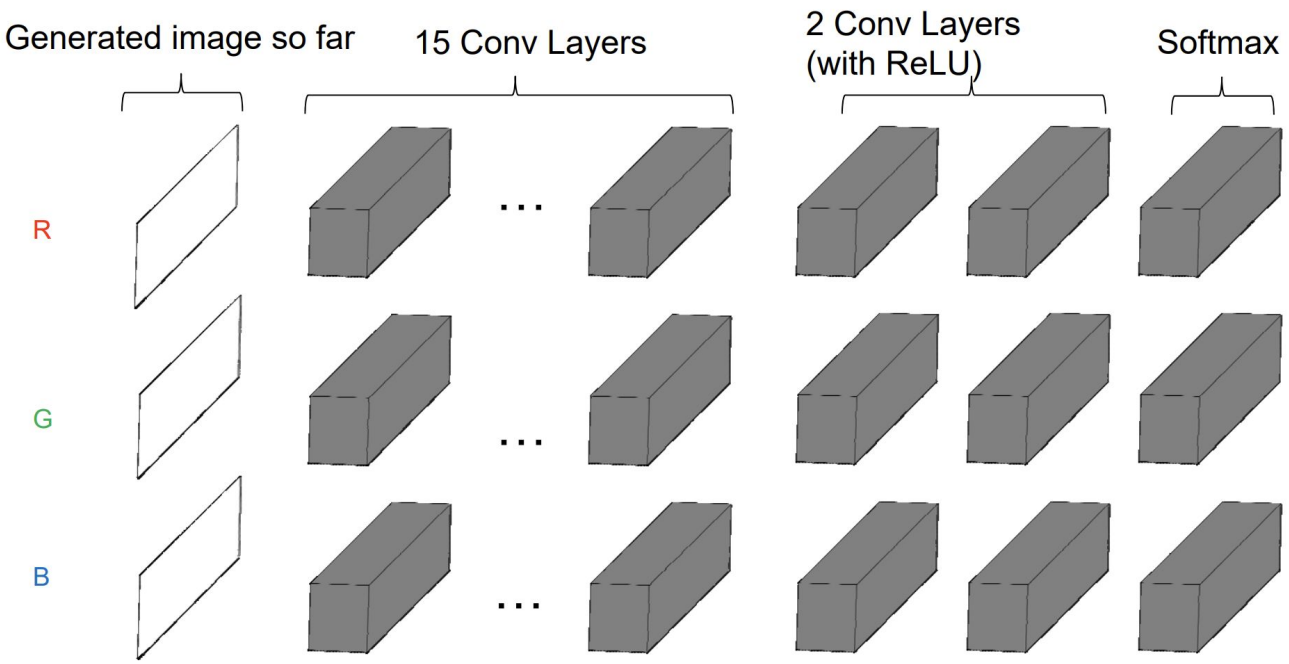
PixelCNN

PixelCNN Masking

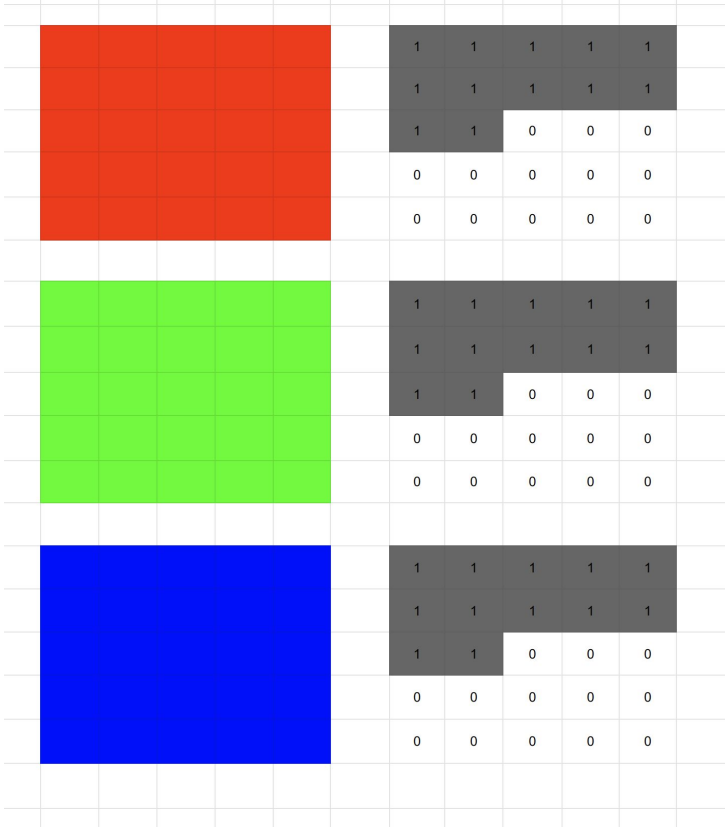


- Mask A only used for first 7x7 convolutional layer
- Mask B used for subsequent layers
- h features for each input position in subsequent layers is 3-dimensional (RGB)
- Mask B relaxes Mask A and allows channel (RGB) hidden state to be used as input

PixelRNN Model

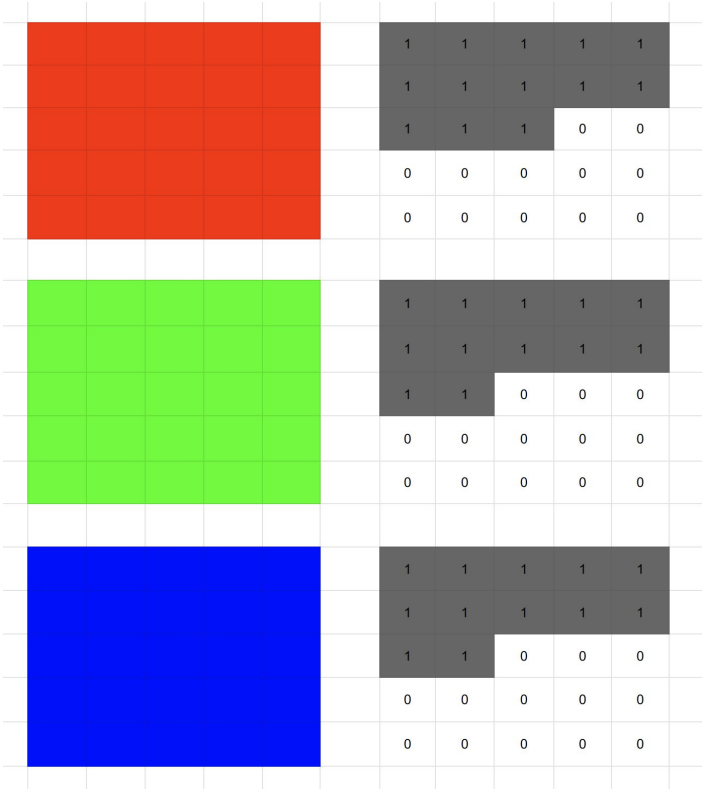


PixelRNN Model



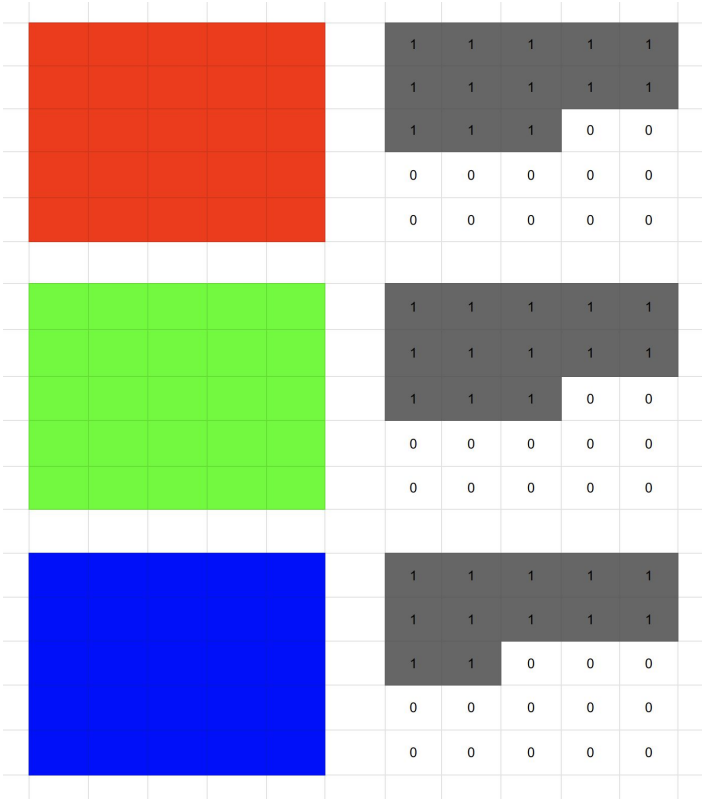
Mask A for R channel

PixelRNN Model



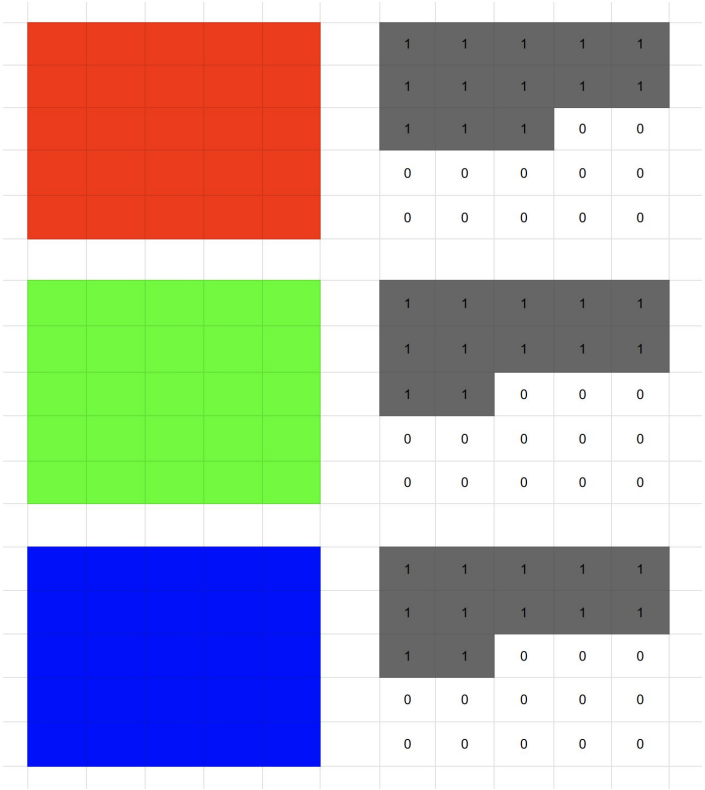
Mask A for G channel

PixelRNN Model



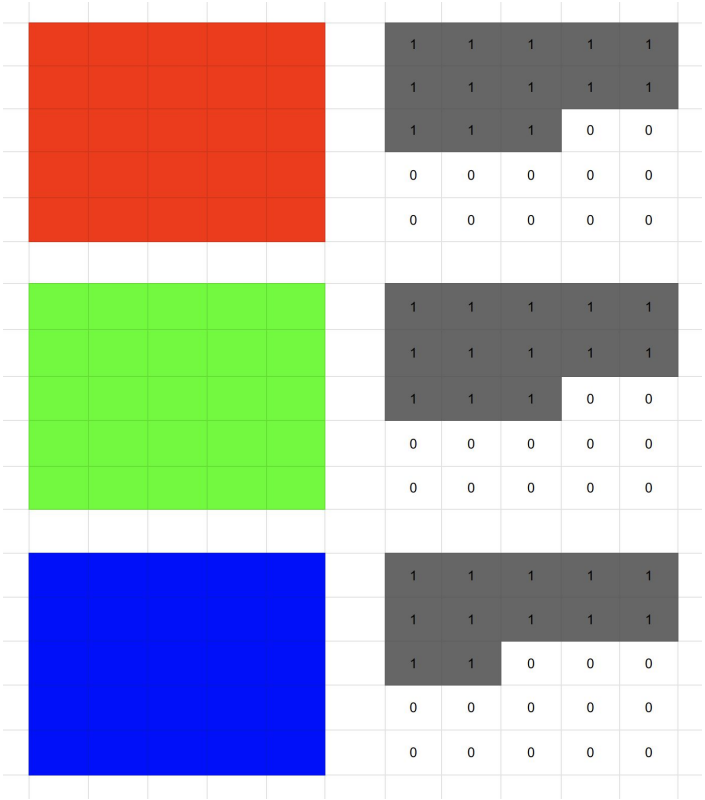
Mask A for B channel

PixelRNN Model



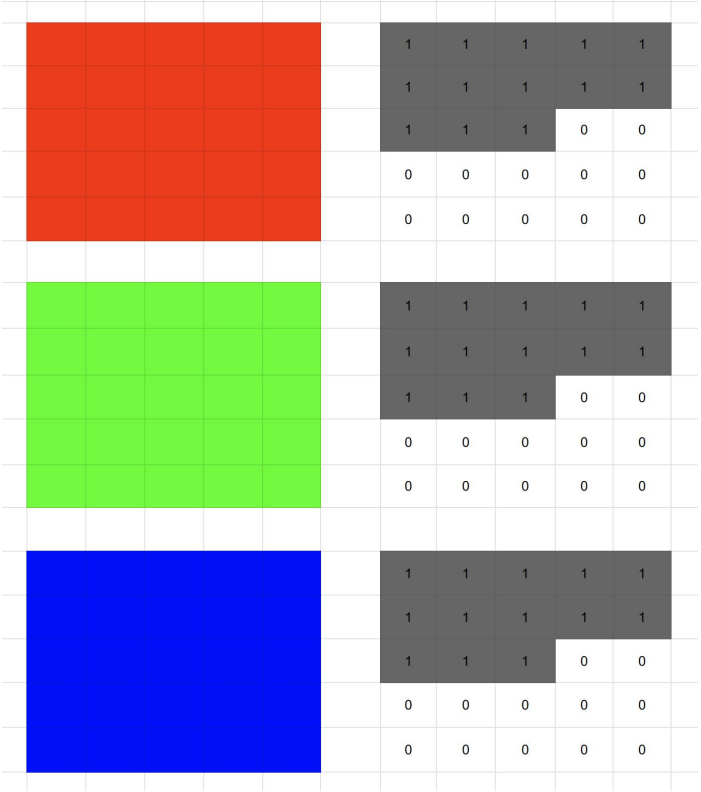
Mask B for R channel

PixelRNN Model



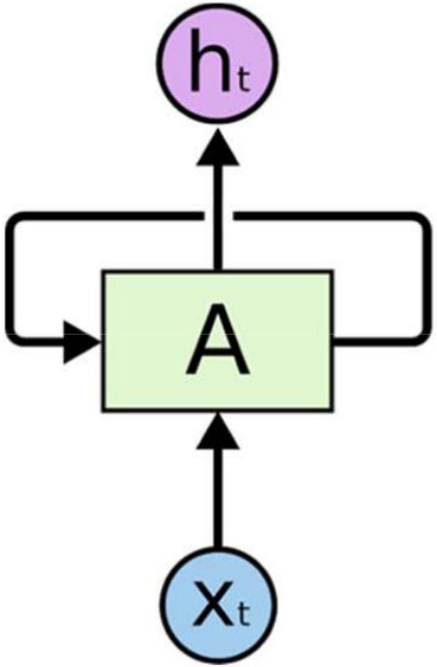
Mask B for G channel

PixelRNN Model



Mask B for B channel

RNN Recap



- Recurrent Neural Network (RNN) unit takes current input and previous hidden state as input
- In practice, vanilla RNN's have trouble learning long-term dependencies in sequences
- This is where LSTM come into play

LSTM Recap

- Forget-gate(f), input-gate(i), output-gate(o), cell-state(c), hidden-state(h)
- Lots of literature on the intuition behind different gates and their purpose
- Better at modelling long-term dependencies than RNNs

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Convolutional LSTM

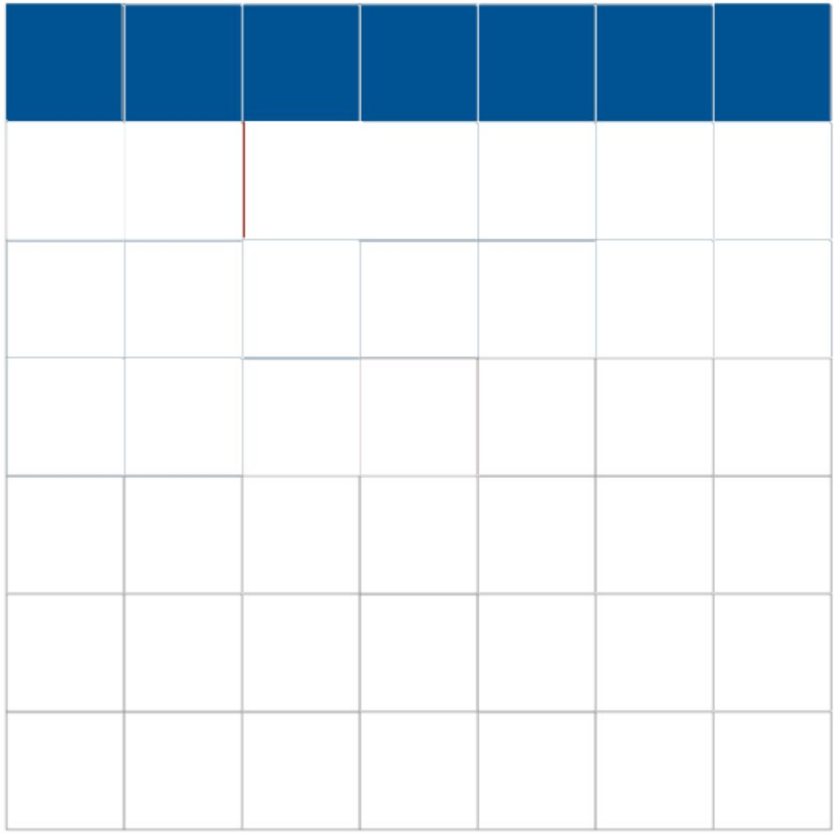
- Replace fully-connected weights in LSTM with convolutional layer

$$[\mathbf{o}_i, \mathbf{f}_i, \mathbf{i}_i, \mathbf{g}_i] = \sigma(\mathbf{K}^{ss} \circledast \mathbf{h}_{i-1} + \mathbf{K}^{is} \circledast \mathbf{x}_i)$$
$$\mathbf{c}_i = \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \mathbf{g}_i$$
$$\mathbf{h}_i = \mathbf{o}_i \odot \tanh(\mathbf{c}_i)$$

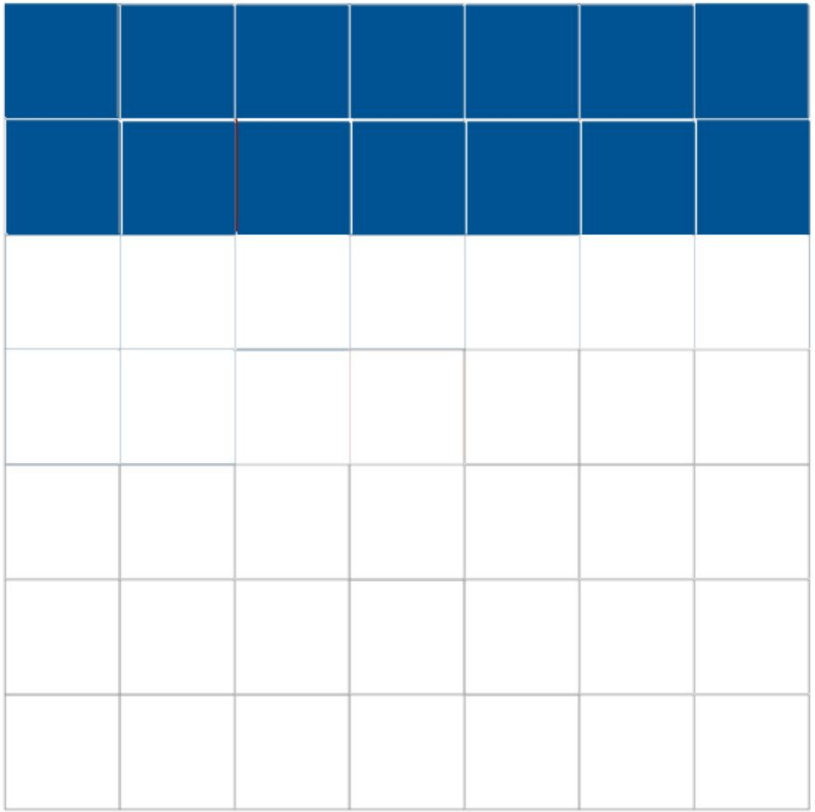
Row LSTM

- Row LSTM processes features row by row from top to bottom computing features for whole row at once
- convolution with k x 1 kernel size where k ≥ 3
- Pre-compute entire input-to-state K^{is} component of LSTM
- Compute state-to-state K^{ss} sequentially using previous hidden states
- K^{is} convolution uses Mask B to protect RGB inter-dependencies

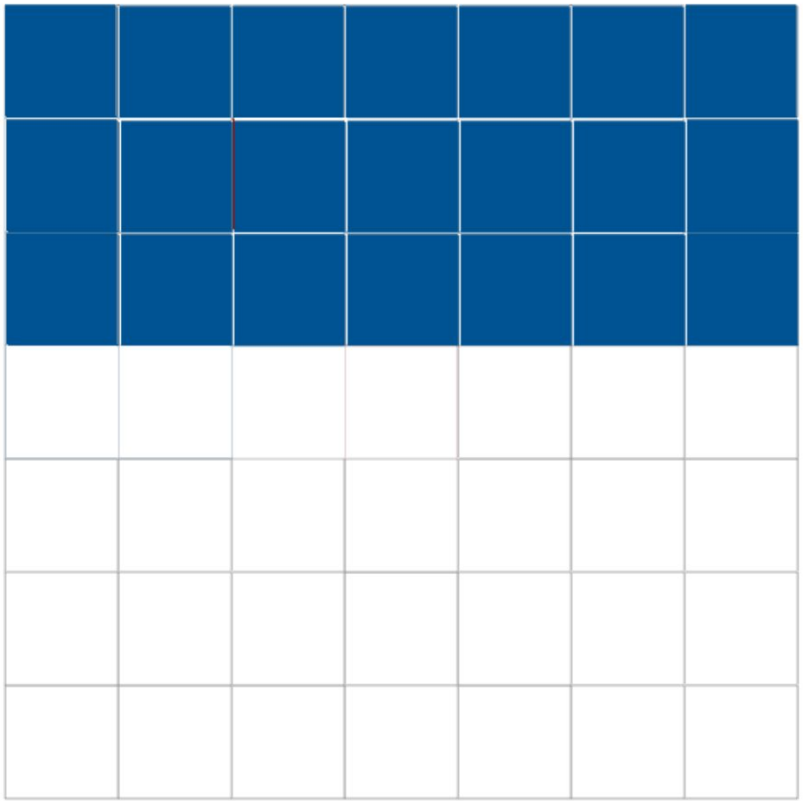
Row LSTM



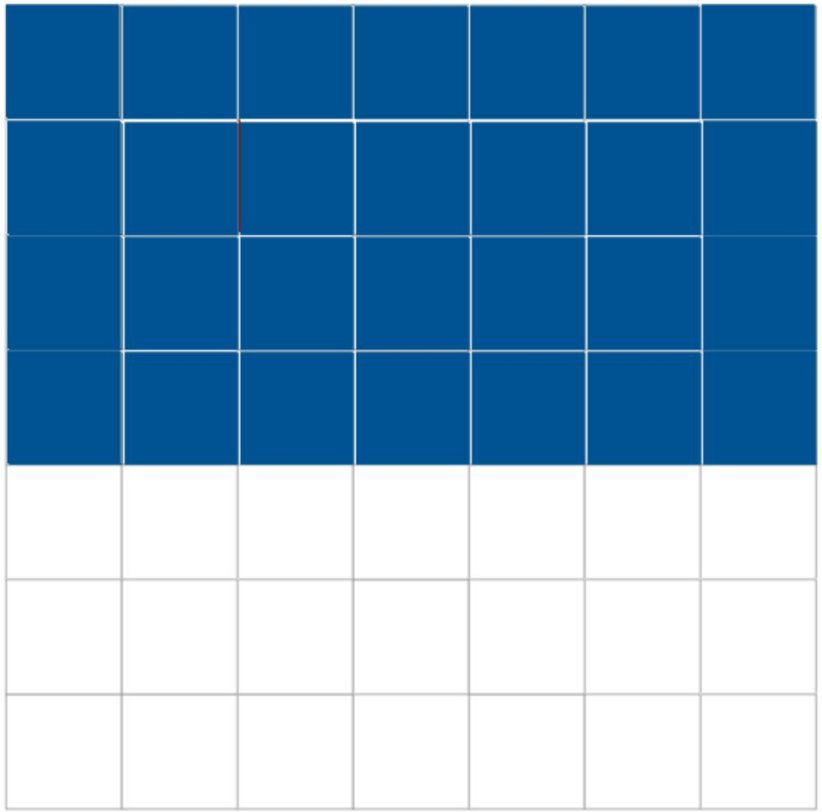
Row LSTM



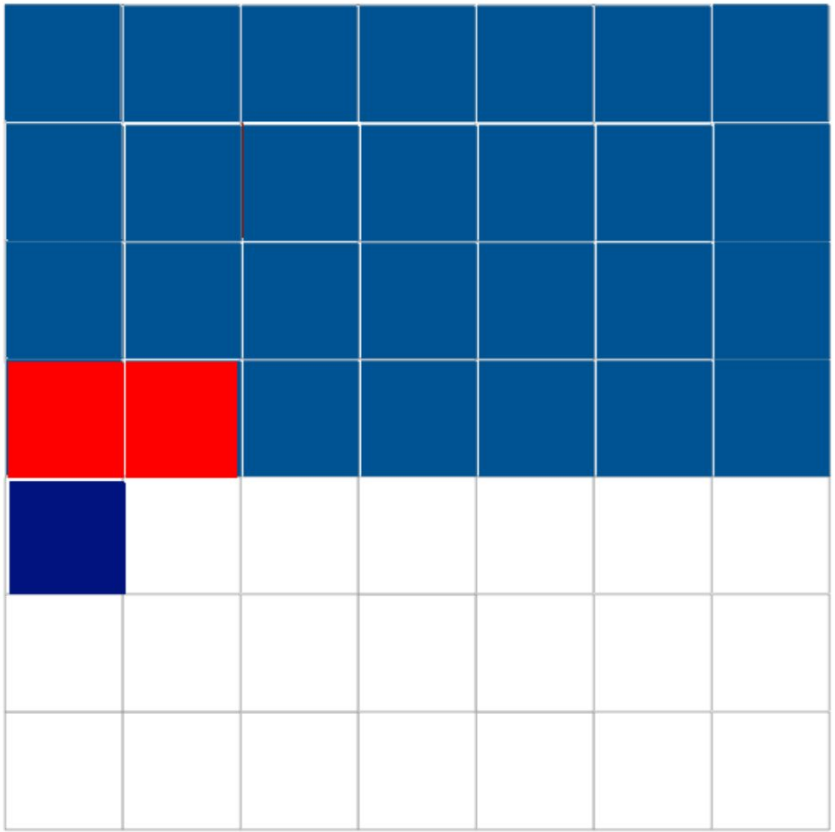
Row LSTM



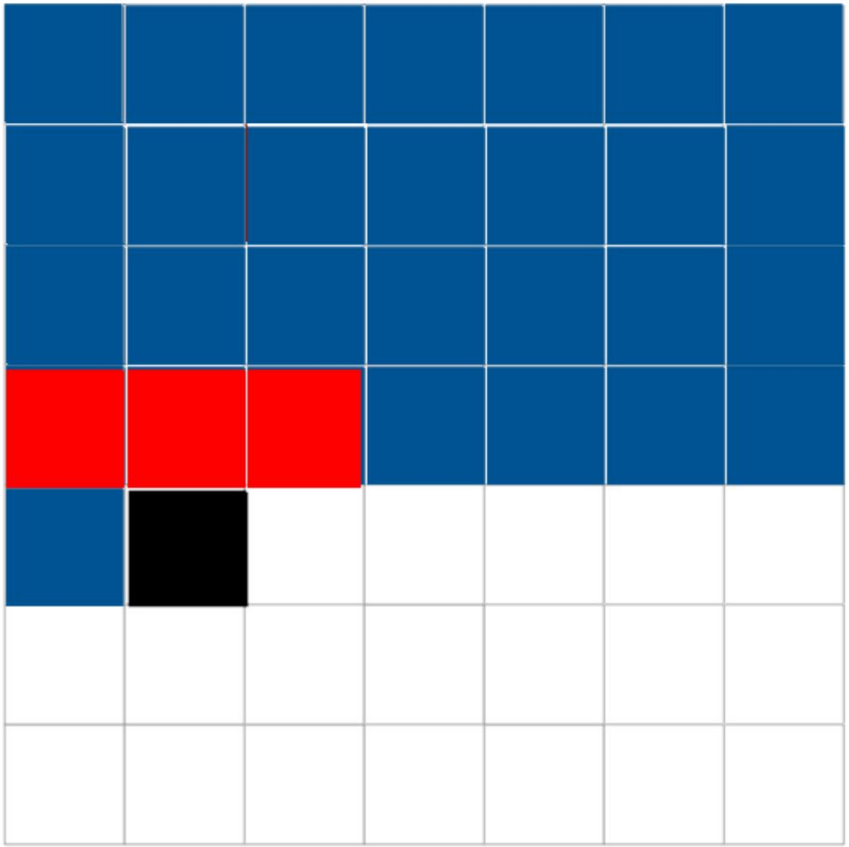
Row LSTM



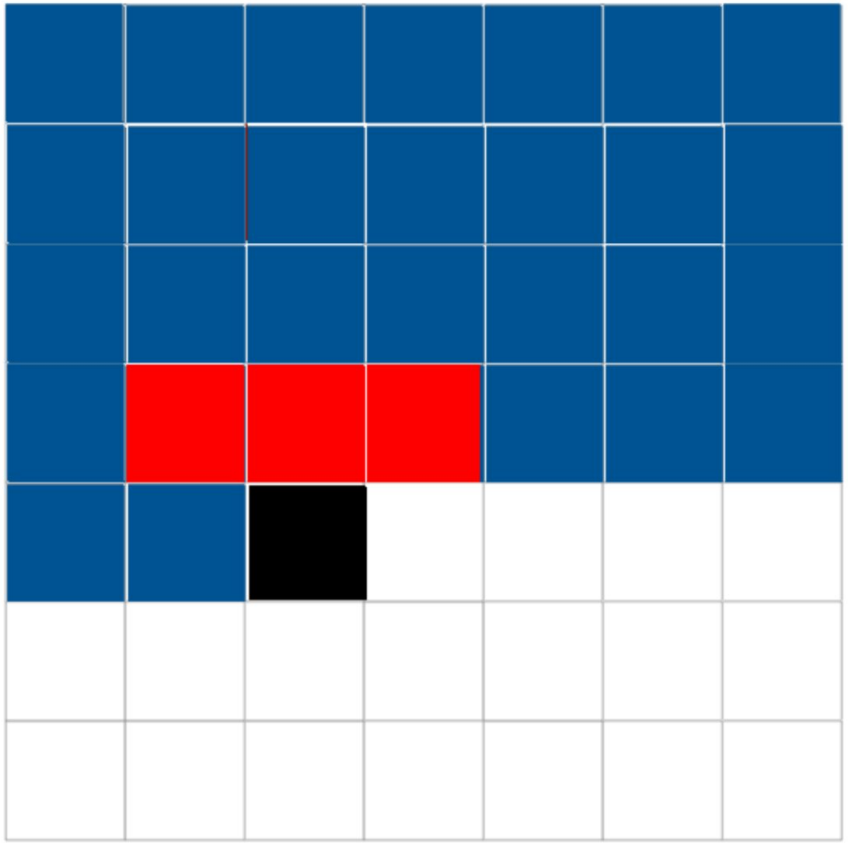
Row LSTM



Row LSTM

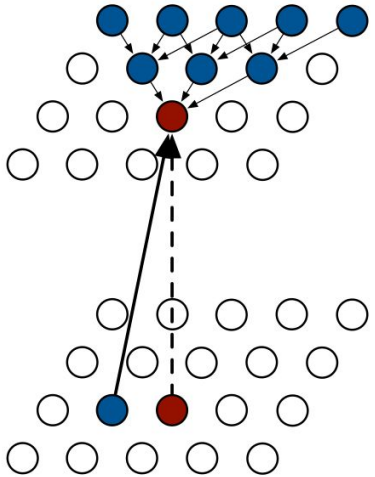


Row LSTM



Row LSTM

- Advantage: Compute state for entire row at once
- Disadvantage: Triangular receptive field

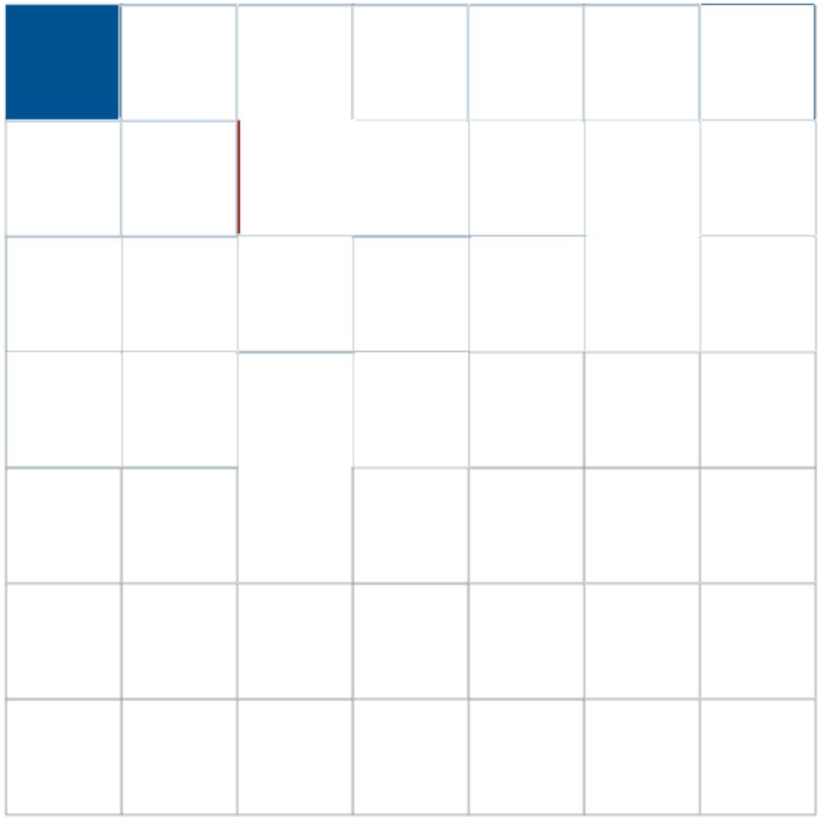


Row LSTM

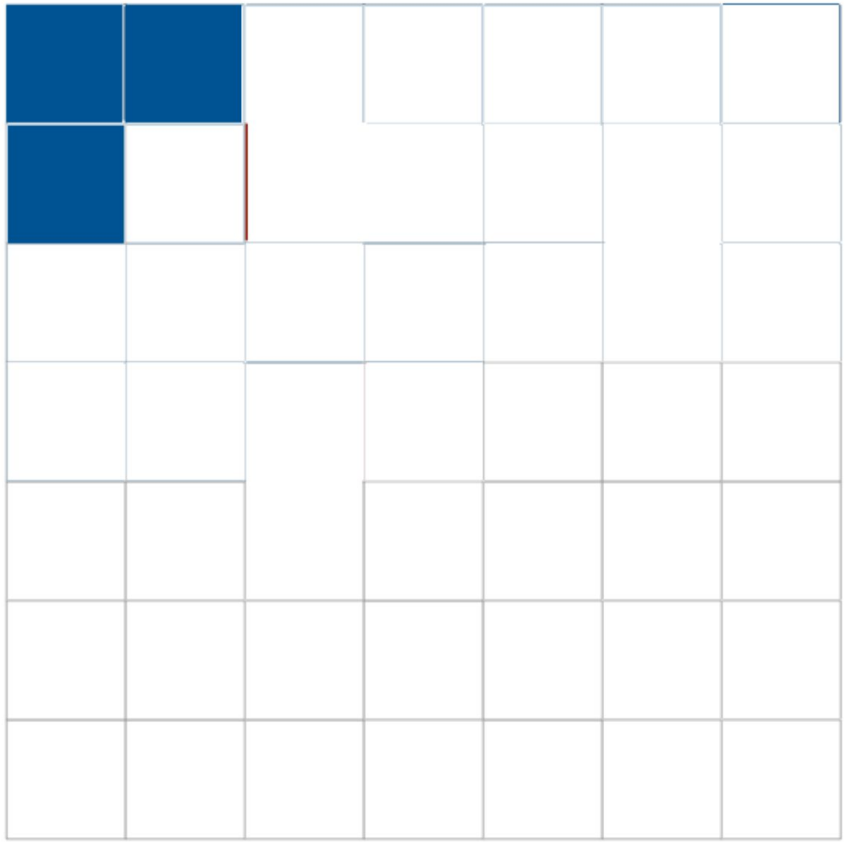
Diagonal BiLSTM

- Goal: capture entire available context
- Convolution scans diagonals of image from two directions
- Input-to-state is simple 1x1 convolution
- State-to-state is 2x1 convolution that operates on skewed image
- Convolutional outputs from two directions added together for final output
- K^{is} convolution uses Mask B to protect RGB inter-dependencies

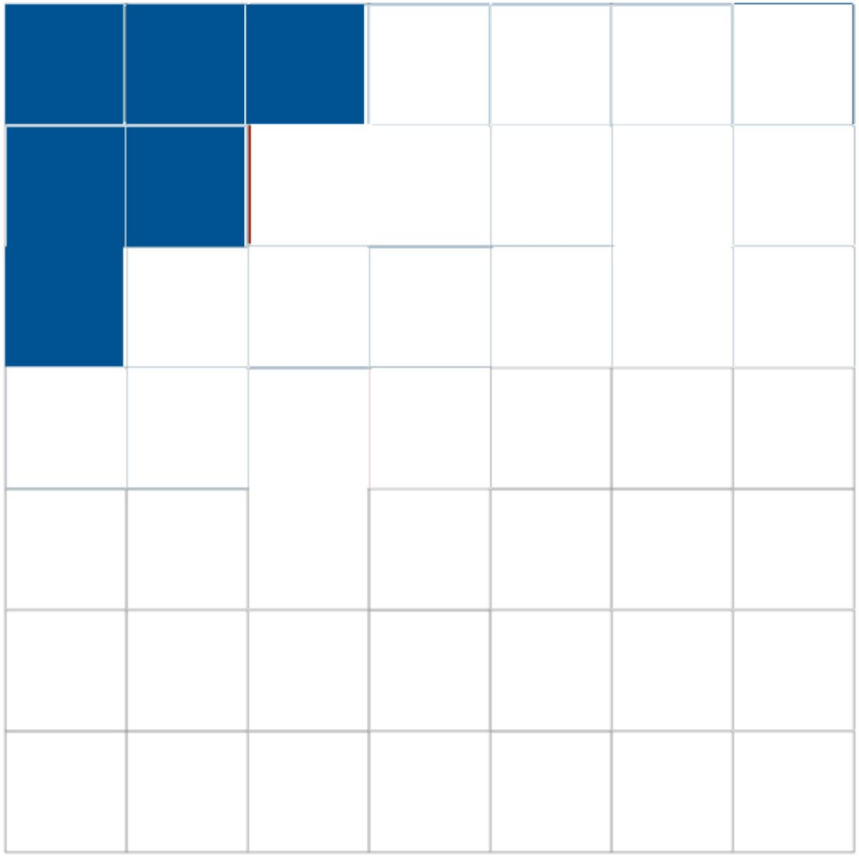
Diagonal BiLSTM



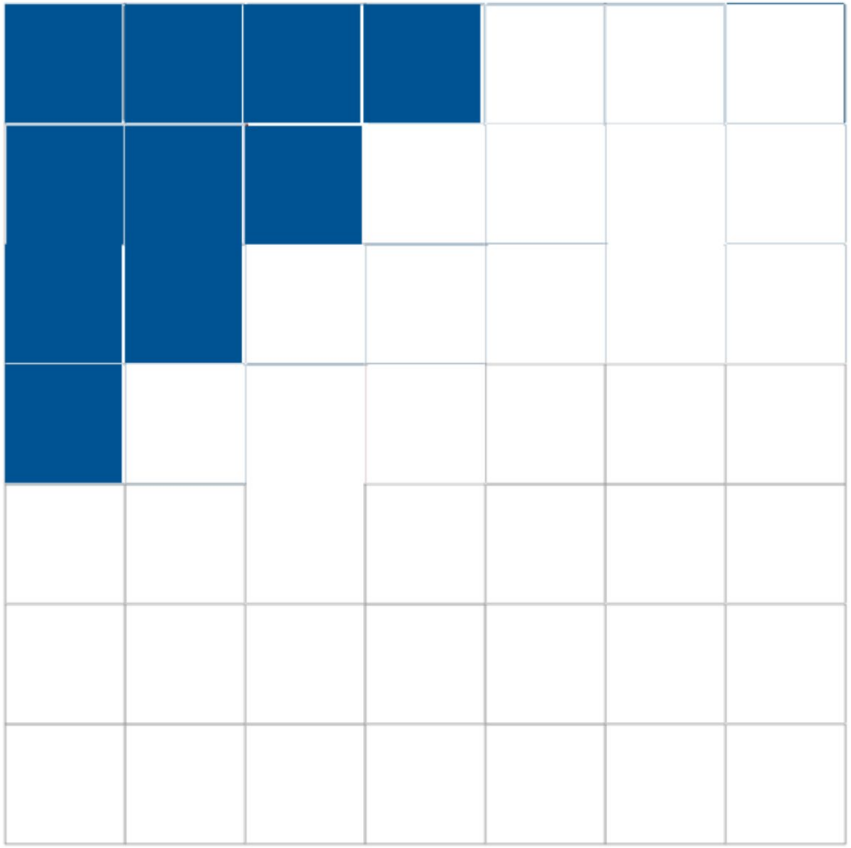
Diagonal BiLSTM



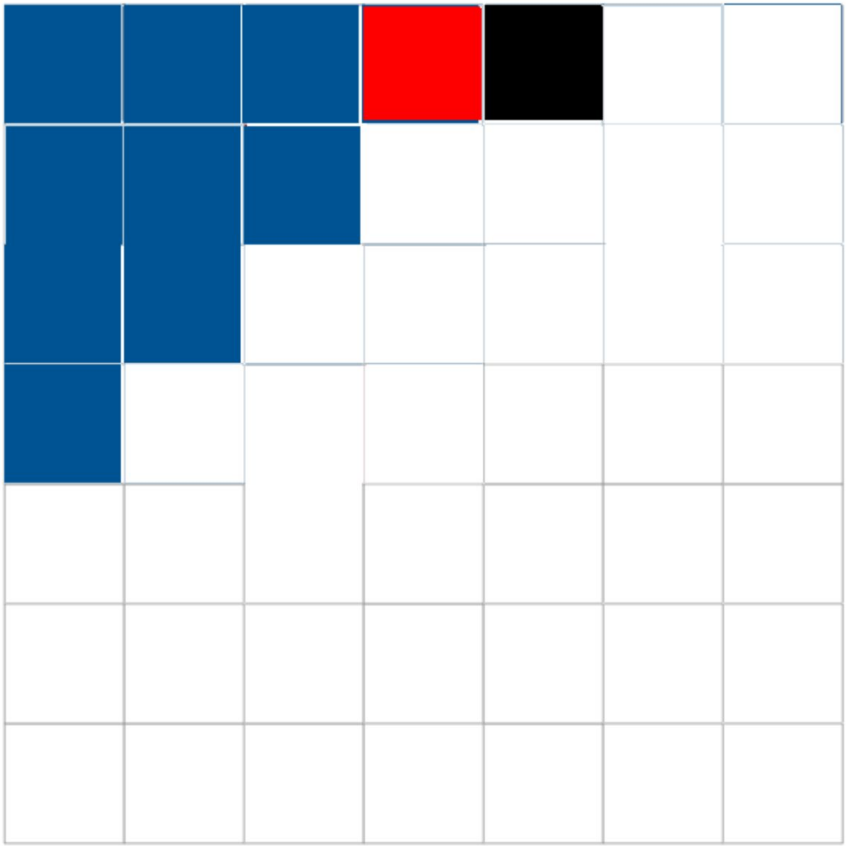
Diagonal BiLSTM



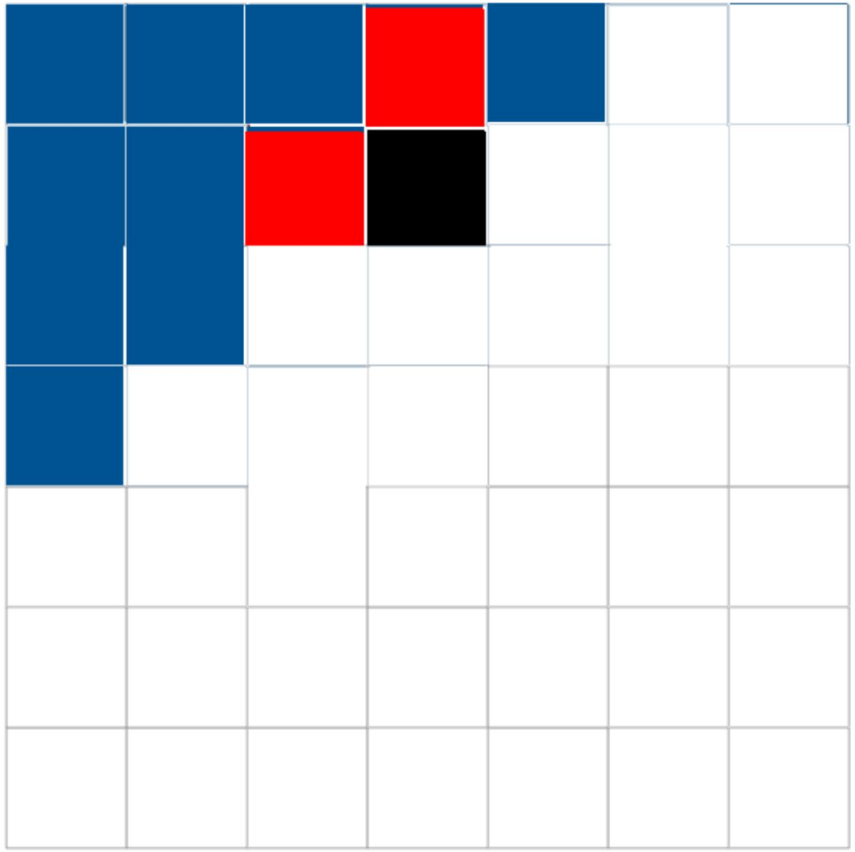
Diagonal BiLSTM



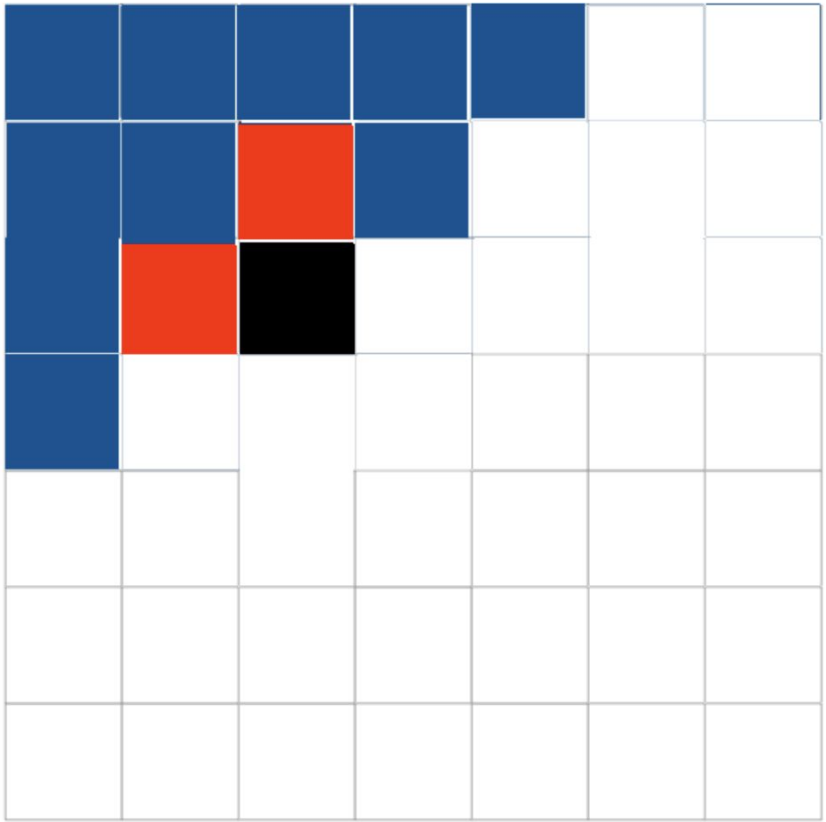
Diagonal BiLSTM



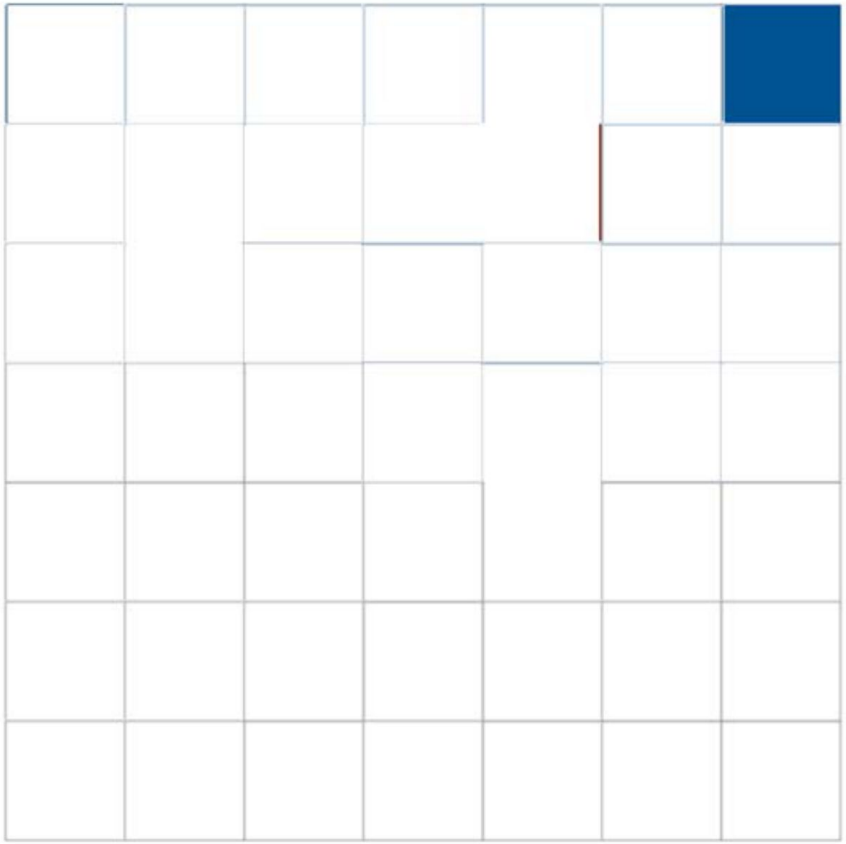
Diagonal BiLSTM



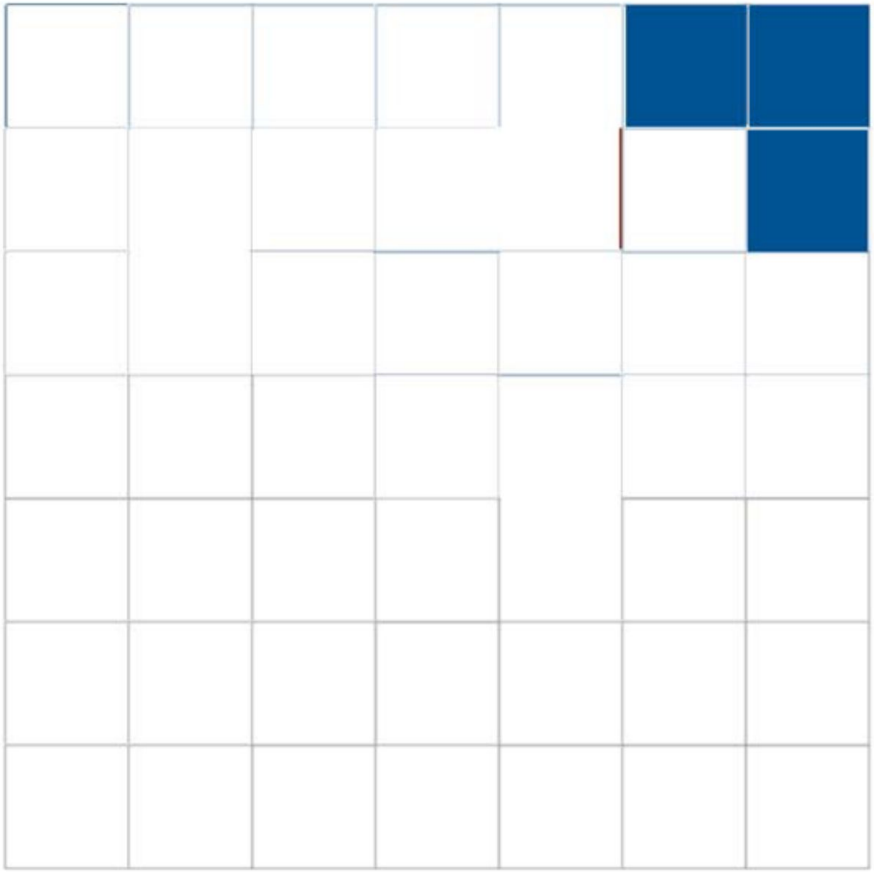
Diagonal BiLSTM



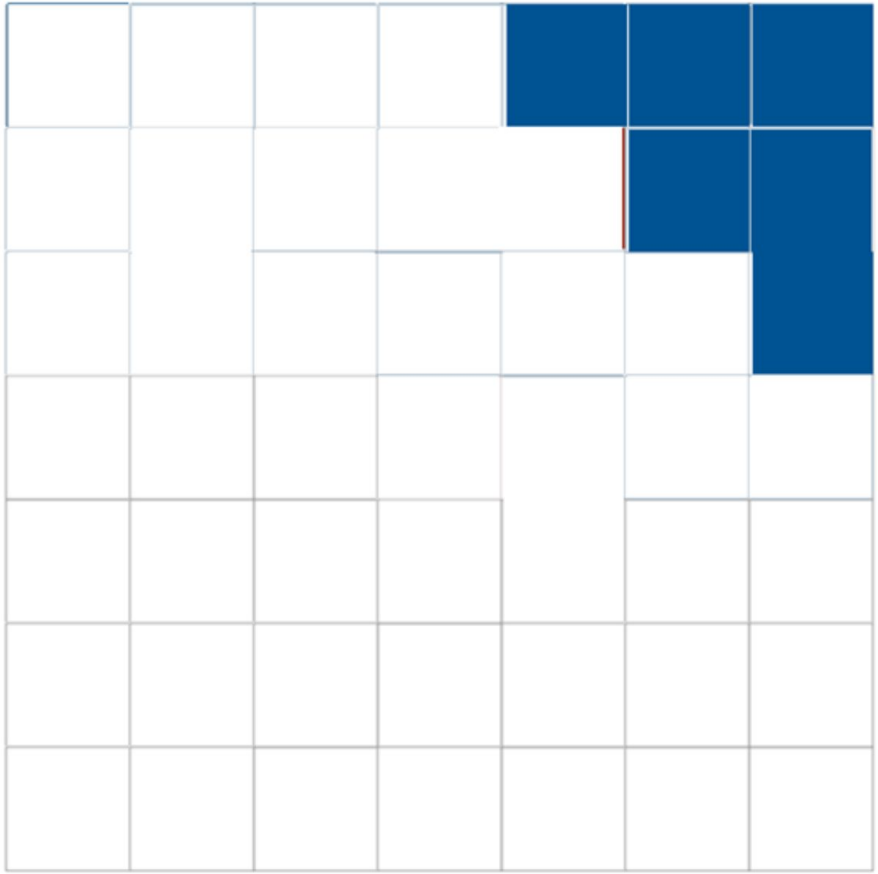
Diagonal BiLSTM



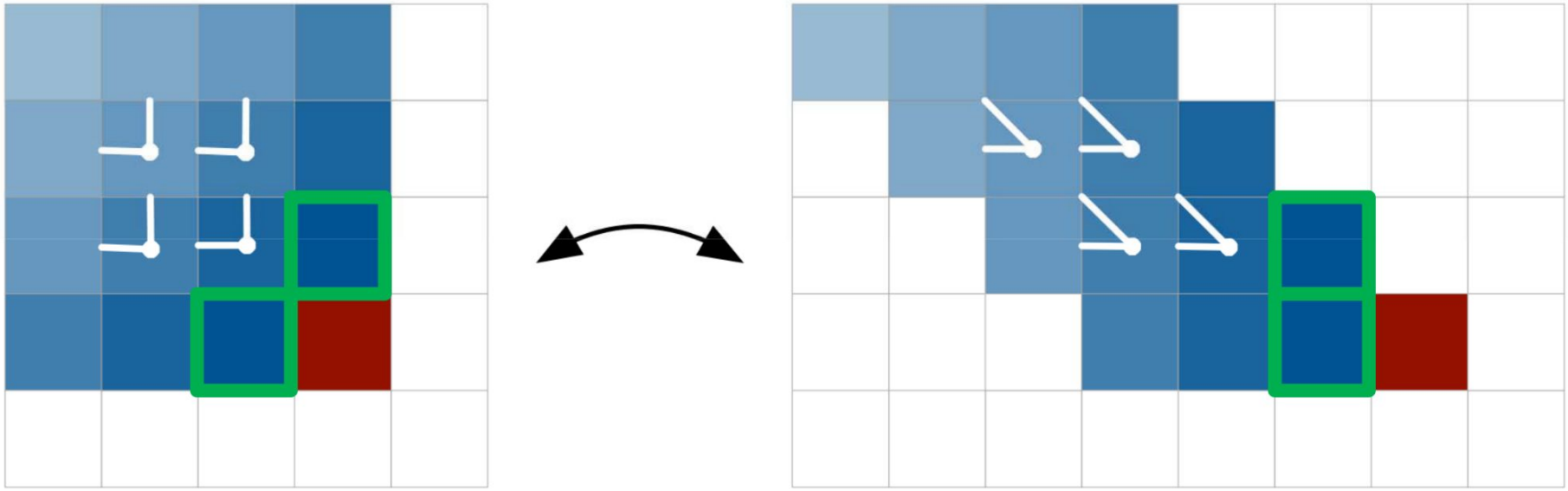
Diagonal BiLSTM



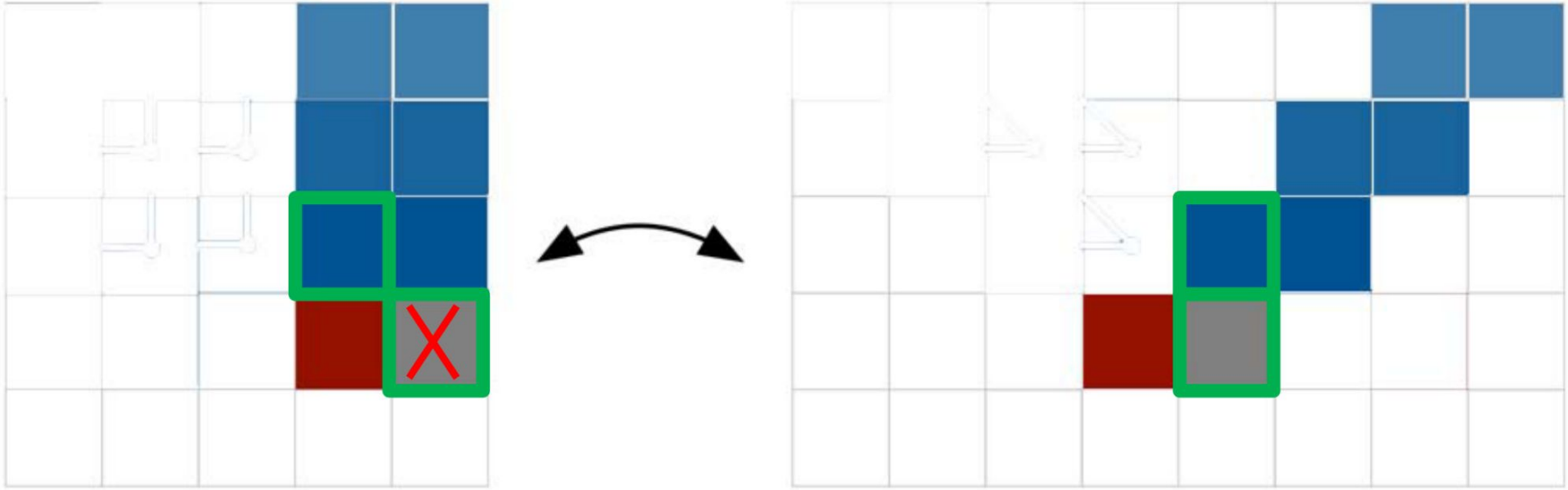
Diagonal BiLSTM



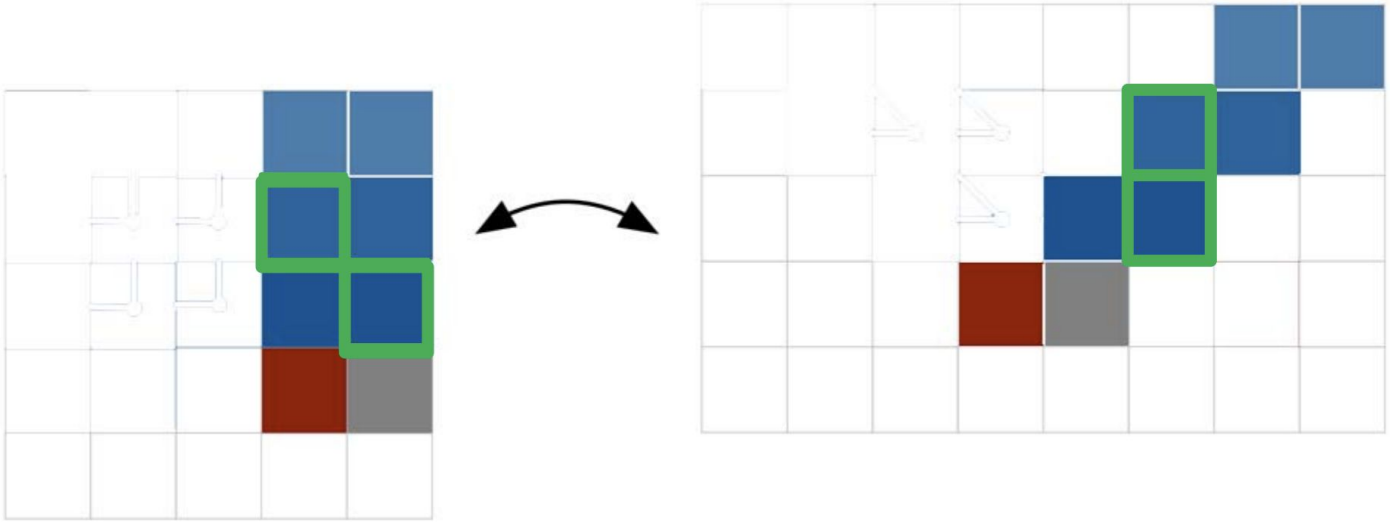
Diagonal BiLSTM



Diagonal BiLSTM

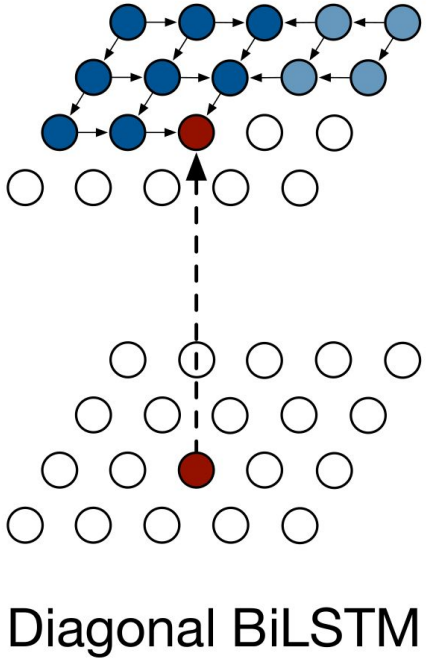


Diagonal BiLSTM



Diagonal BiLSTM

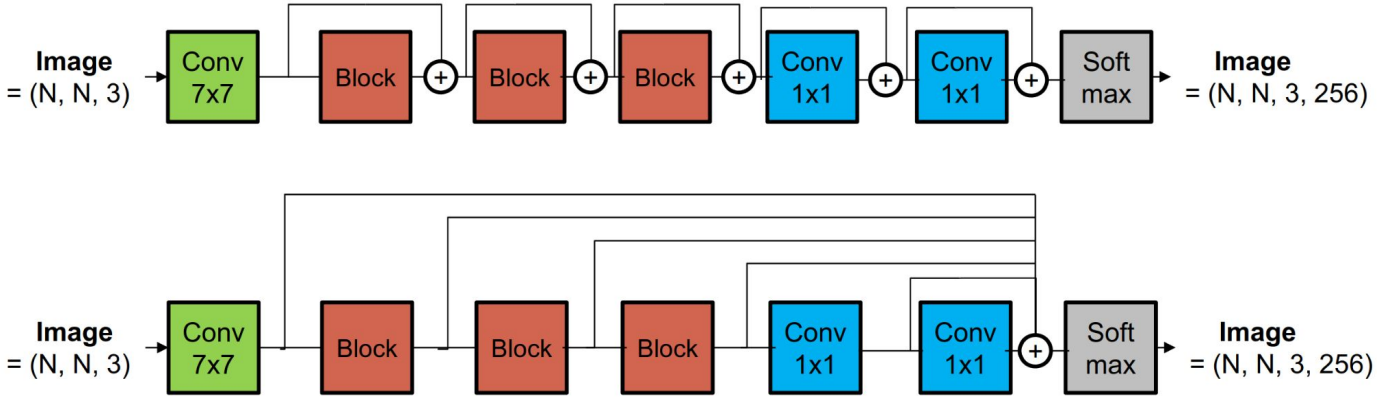
- Advantage: Receptive field captures all available context
- Disadvantage: Slower than Row LSTM



Diagonal BiLSTM

Residual Connections

- Enables training PixelRNN up to twelve layers of depth
- Increases both convergence speed and propagates signals more directly through the network



Residual Connections

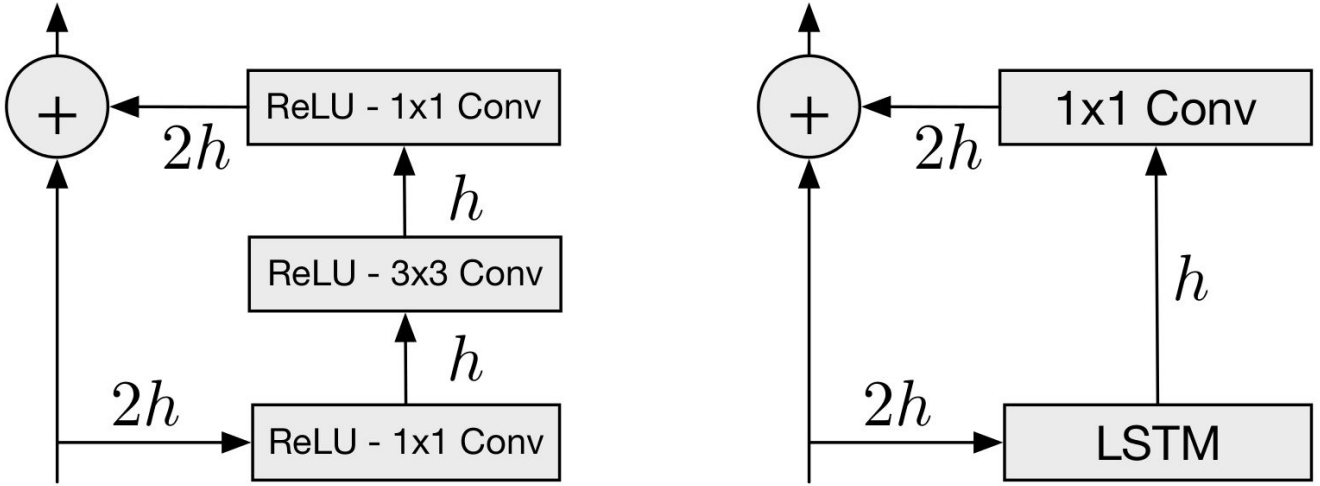
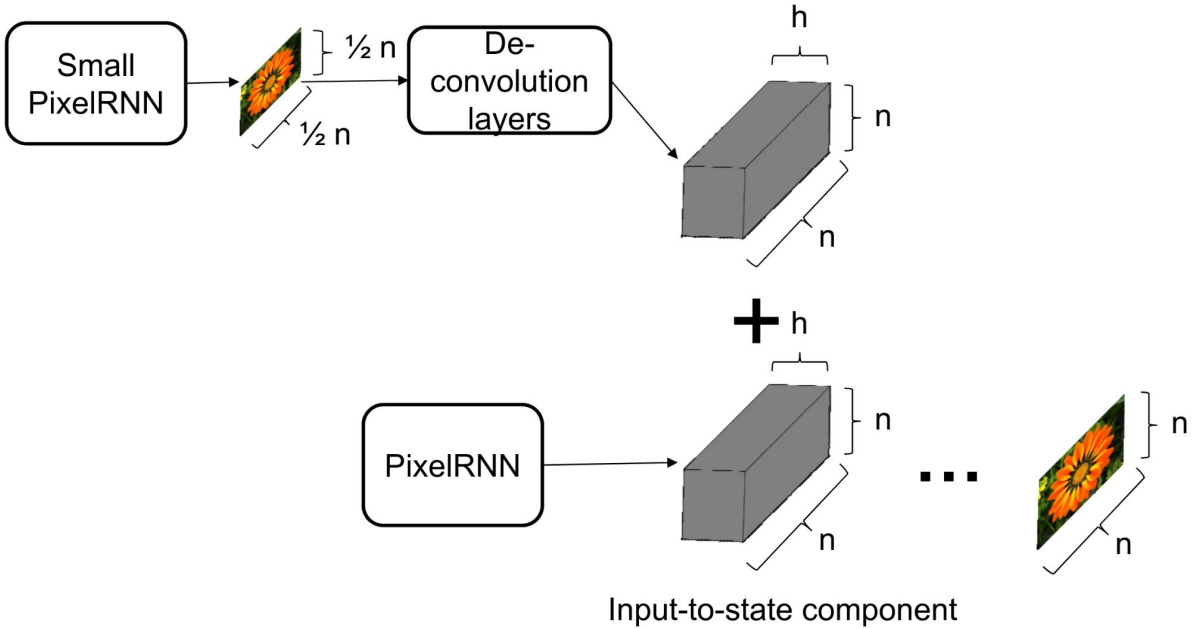


Figure 5. Residual blocks for a PixelCNN (left) and PixelRNNs.

Multi-Scale PixelRNN

- Unconditional network generates smaller s x s scale image
- Subsequent conditional network(s) use smaller s x s image as an additional input and generate larger n x n image
- Smaller image is upsampled using deconvolutional layers and added to input-to-state map of corresponding conditional network

Multi-Scale PixelRNN



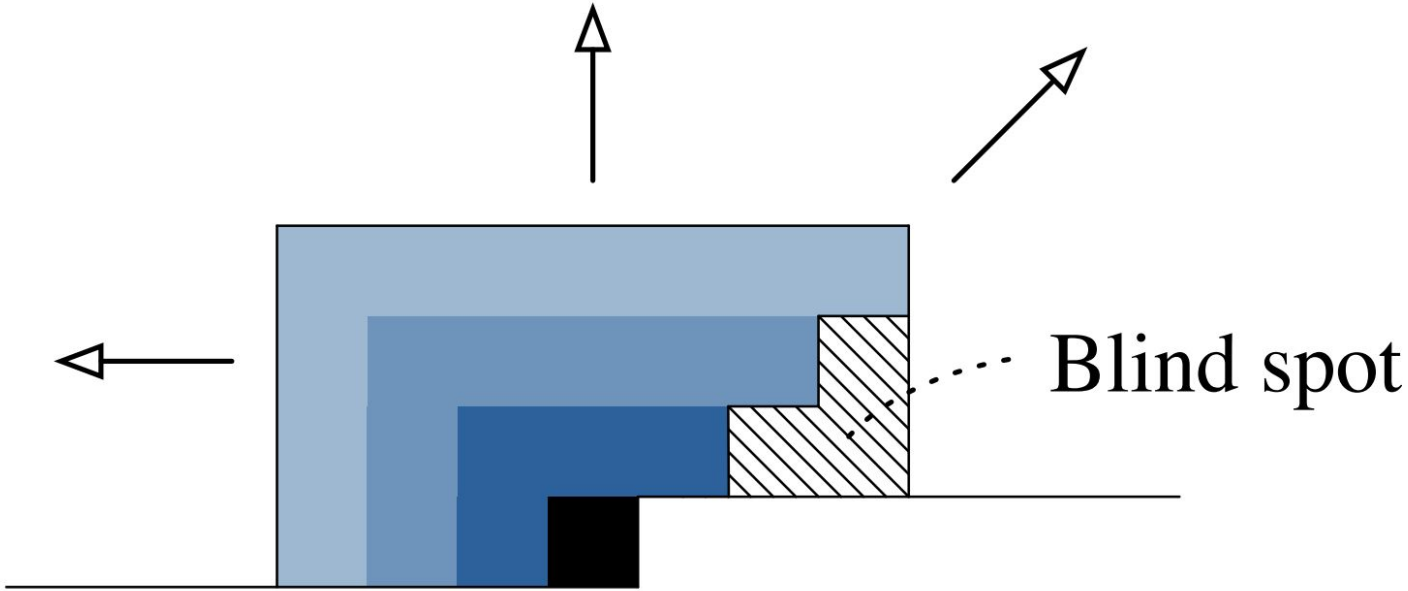
Gated PixelCNN

- PixelRNN are outperforming PixelCNN because:
  - PixelRNN models capture larger receptive fields
  - LSTM cells contain multiplicative units that model more complex interactions
- Fix receptive field using horizontal and vertical stacking convolutions
- Replace ReLU activation with gated activation unit to add more sophistication
- Now PixelCNN match PixelRNN performance while requiring half the training time

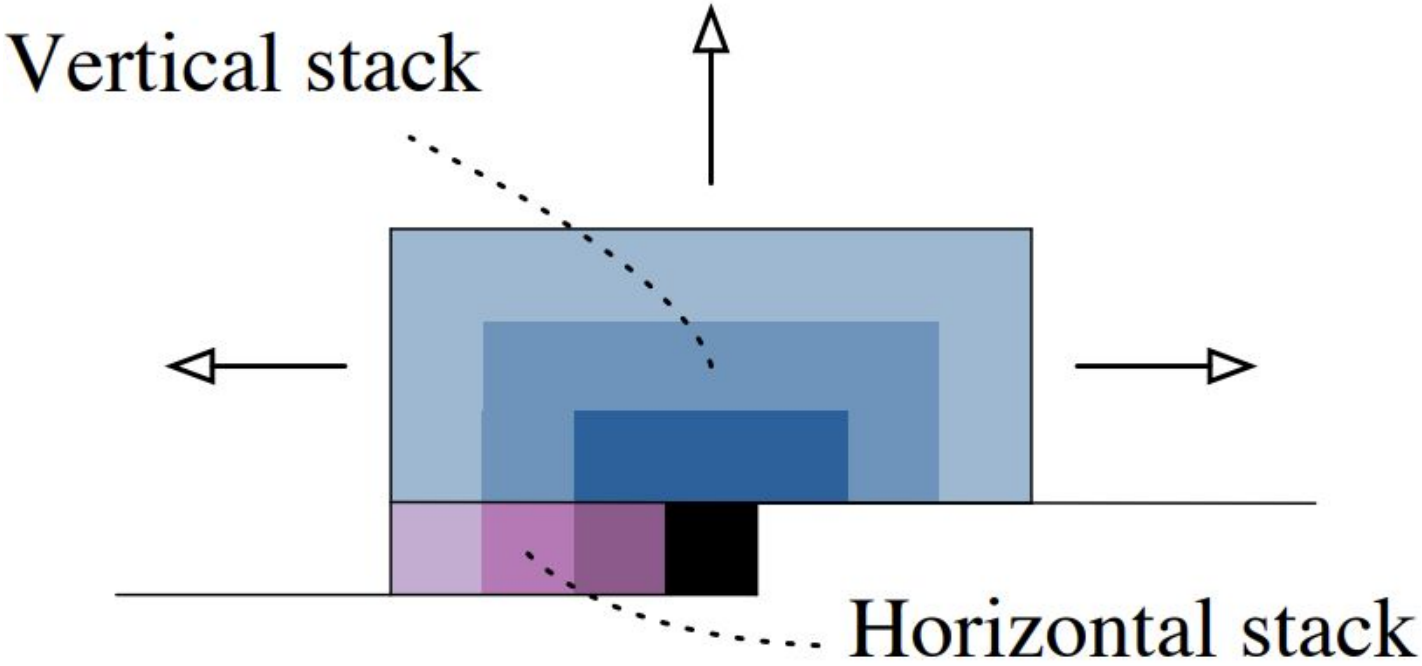
PixelCNN Receptive Field Blind Spot

- Because of kernel masking, the receptive field of PixelCNN contains a growing blind spot
- To fix this blind spot, single convolution operation is broken down into horizontal and vertical stack convolution
- Vertical stack in unmasked and captures entire receptive field in the above rows
- Horizontal stack is masked and uses pixel to the left + vertical stack as inputs
- This fixes the receptive field problem

PixelCNN Receptive Field Blind Spot



PixelCNN Receptive Field Blind Spot



Gated Convolutional Layers

- Replace ReLU activation with gated activation
- Two separate convolutions with half the feature maps
- Each convolution is followed by two non-linear activations
- Both outputs multiplied element-wise for final output

$$\mathbf{y} = \tanh(W_{k,f} * \mathbf{x}) \odot \sigma(W_{k,g} * \mathbf{x})$$

Gated Convolutional Layers

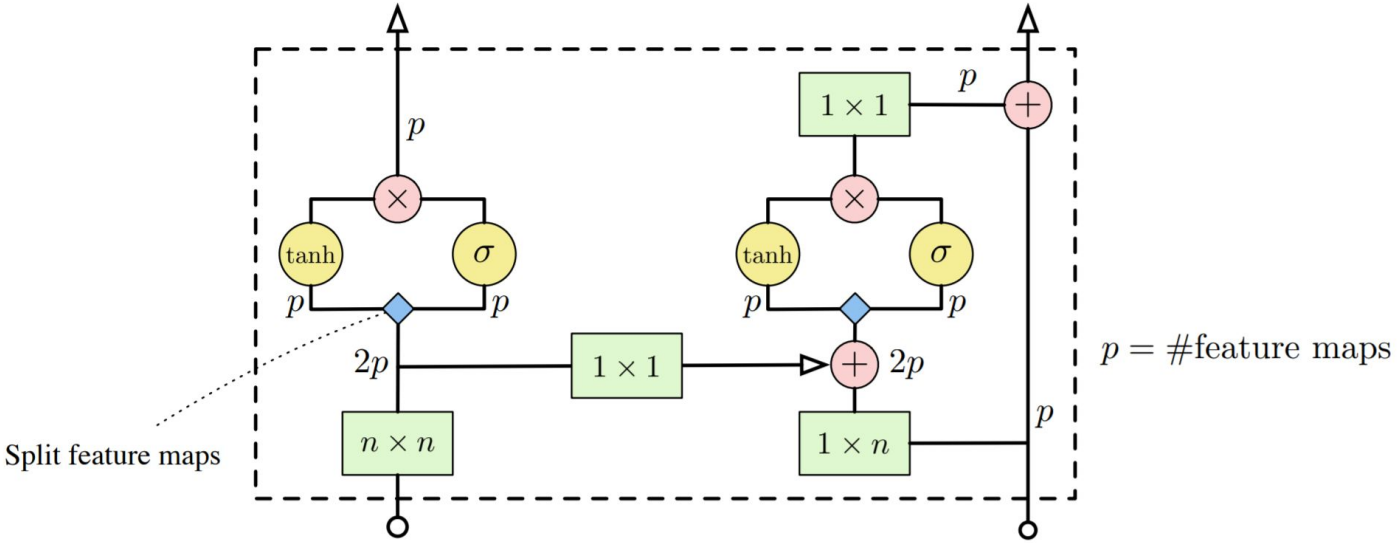


Figure 2: A single layer in the Gated PixelCNN architecture. Convolution operations are shown in green, element-wise multiplications and additions are shown in red. The convolutions with W_f and W_g from Equation 2 are combined into a single operation shown in blue, which splits the $2p$ features maps into two groups of p.

Conditional PixelCNN

- Given image description in high-dimensional latent vector h
- Goal: model conditional distribution of image given h
- h can contain information about objects in image e.g. one-hot encoding of objects in ImageNet sample
- h can only specify what is in the image but cannot control where in the image the object will appear
- Also developed a location dependent variant using deconvolution network m() map spatial representation of object as s = m(h) which has same height and width as image but arbitrary feature maps

Conditional PixelCNN

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

$$\mathbf{y} = \tanh(W_{k,f} * \mathbf{x} + V_{k,f}^T \mathbf{h}) \odot \sigma(W_{k,g} * \mathbf{x} + V_{k,g}^T \mathbf{h})$$

$$\mathbf{y} = \tanh(W_{k,f} * \mathbf{x} + V_{k,f} * \mathbf{s}) \odot \sigma(W_{k,g} * \mathbf{x} + V_{k,g} * \mathbf{s})$$

where $V_{k,g} * \mathbf{s}$ is an unmasked 1×1 convolution.

Empirical Results

Model	NLL Test
DBM 2hl [1]:	$\approx$ 84.62
DBN 2hl [2]:	$\approx$ 84.55
NADE [3]:	88.33
EoNADE 2hl (128 orderings) [3]:	85.10
EoNADE-5 2hl (128 orderings) [4]:	84.68
DLGM [5]:	$\approx$ 86.60
DLGM 8 leapfrog steps [6]:	$\approx$ 85.51
DARN 1hl [7]:	$\approx$ 84.13
MADE 2hl (32 masks) [8]:	86.64
DRAW [9]:	$\leq$ 80.97
PixelCNN:	81.30
Row LSTM:	80.54
Diagonal BiLSTM (1 layer, $h = 32$):	**80.75**
Diagonal BiLSTM (7 layers, $h = 16$):	**79.20**

Table 4. Test set performance of different models on MNIST in *nats* (negative log-likelihood). Prior results taken from [1] (Salakhutdinov & Hinton, 2009), [2] (Murray & Salakhutdinov, 2009), [3] (Uria et al., 2014), [4] (Raiko et al., 2014), [5] (Rezende et al., 2014), [6] (Salimans et al., 2015), [7] (Gregor et al., 2014), [8] (Germain et al., 2015), [9] (Gregor et al., 2015).

Model	NLL Test (Train)
Uniform Distribution:	8.00
Multivariate Gaussian:	4.70
NICE [1]:	4.48
Deep Diffusion [2]:	4.20
Deep GMMs [3]:	4.00
RIDE [4]:	3.47
PixelCNN:	3.14 (3.08)
Row LSTM:	3.07 (3.00)
Diagonal BiLSTM:	**3.00** (2.93)

Table 5. Test set performance of different models on CIFAR-10 in *bits/dim*. For our models we give training performance in brackets. [1] (Dinh et al., 2014), [2] (Sohl-Dickstein et al., 2015), [3] (van den Oord & Schrauwen, 2014a), [4] personal communication (Theis & Bethge, 2015).

Image size	NLL Validation (Train)
32x32:	3.86 (3.83)
64x64:	3.63 (3.57)

Table 6. Negative log-likelihood performance on 32×32 and 64×64 ImageNet in *bits/dim*.

Empirical Results

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)

Table 1: Test set performance of different models on CIFAR-10 in *bits/dim* (lower is better), training performance in brackets.

Empirical Results

32x32	Model	NLL Test (Train)
	Conv Draw: [8]	4.40 (4.35)
	PixelRNN: [30]	3.86 (3.83)
	Gated PixelCNN:	3.83 (3.77)
64x64	**Model**	**NLL Test (Train)**
	Conv Draw: [8]	4.10 (4.04)
	PixelRNN: [30]	3.63 (3.57)
	Gated PixelCNN:	3.57 (3.48)

Table 2: Performance of different models on ImageNet in *bits/dim* (lower is better), training performance in brackets.

Conditional Generation Examples

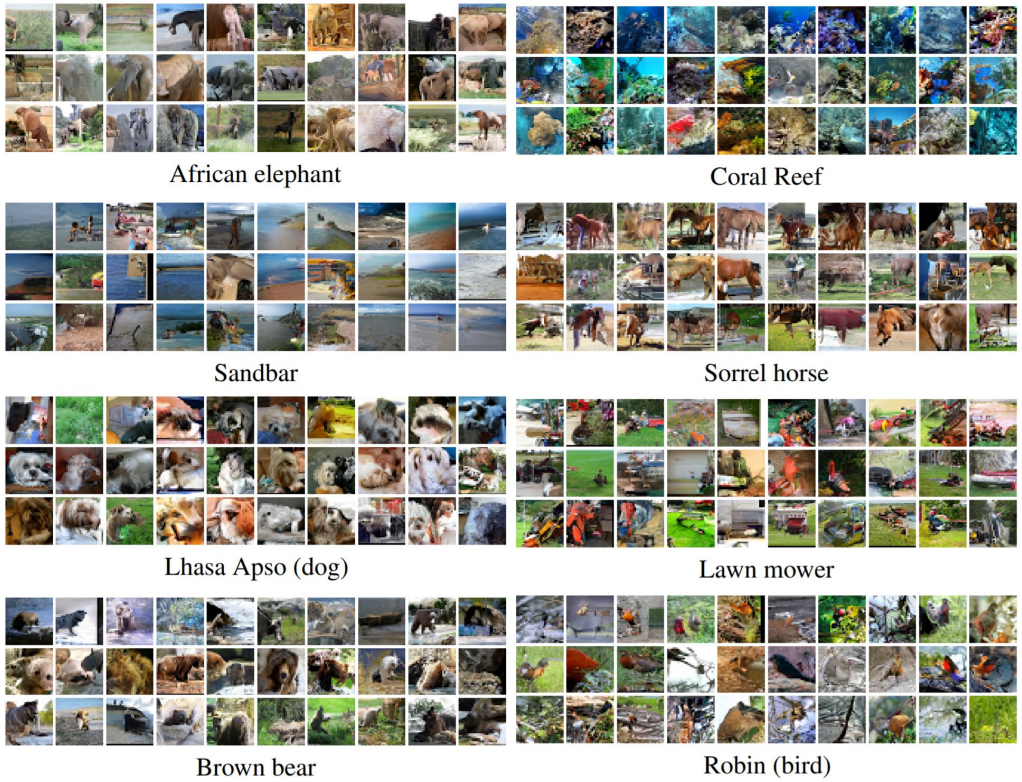


Figure 3: Class-Conditional samples from the Conditional PixelCNN.

References

- Oord, Aaron van den, et al. “Pixel Recurrent Neural Networks.” ArXiv:1601.06759 [Cs], Aug. 2016. arXiv.org, http://arxiv.org/abs/1601.06759
- Oord, Aaron van den, et al. “Conditional Image Generation with PixelCNN Decoders.” ArXiv:1606.05328 [Cs], June 2016. arXiv.org, http://arxiv.org/abs/1606.05328
- Slides from UCF PixelRNN presentation by Logan Lebanoff 2/22/17 https://www.crcv.ucf.edu/wp-content/uploads/2019/03/CAP6412_Spring2018_Pixel-Recurrent-Neural-Networks.pdf
- Video Presentation by Logan Lebanoff on PixelRNN https://www.youtube.com/watch?v=-FFveGrG46w