PixelCNN

&

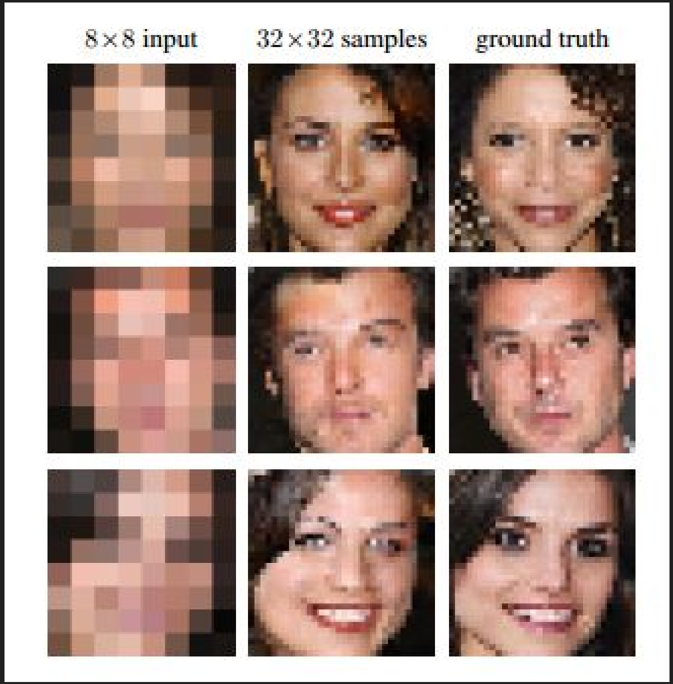
Pixel Recursive Super Resolution

Super Resolution

- Super Resolution is the problem of artificially enlarging a low resolution photograph or image
  - The higher the magnification ratio, the harder it is
- Applications?

Their Data

- Took 32x32 images of faces and blurred them to an 8x8 image
- Then try to recreate original 32x32 image



PixelCNN

- They applied the PixelCNN architecture to super resolution
- The PixelCNN architecture was initially used to fill holes in images
    - Their hole was always the bottom half of the image
- What types of networks would be necessary to accomplish this?

Figure 1. Image completions sampled from a PixelRNN.

They offered three ideas for how to do it

- Row LSTM runs the three pixels above the current one recursively
- Diagonal Bi-LSTM passes bidirectionally from all previous pixels
- PixelCNN simply convolves the pixels around the current one, and masks those from *~~~the future~~~*

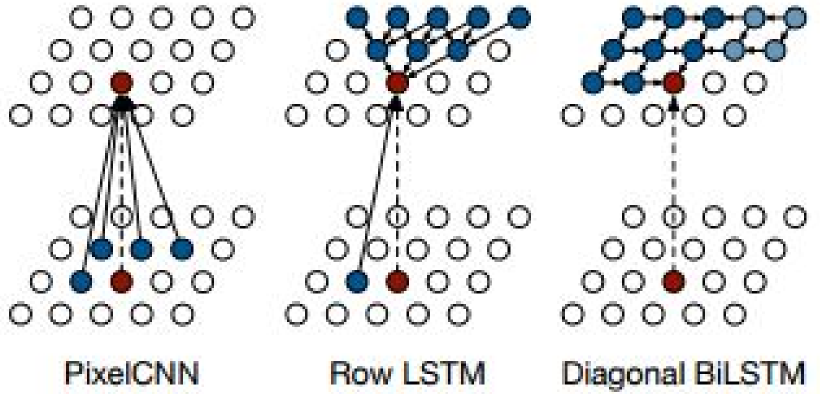


Figure 4. Visualization of the input-to-state and state-to-state mappings for the three proposed architectures.

Let’s talk about those three architectures

- How do the LSTMs process the sequential data?
- What advantages does each architecture have?
- Is symmetry necessary?
- Why are they all diagonal down and to the left?
- What do we do if the PixelCNN is on the left border?

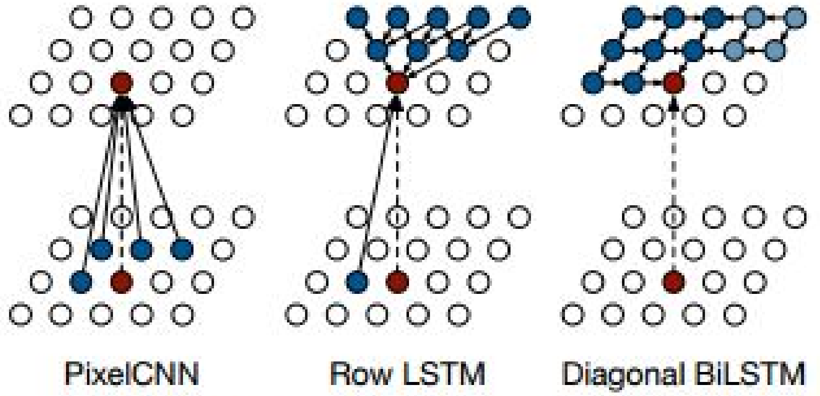


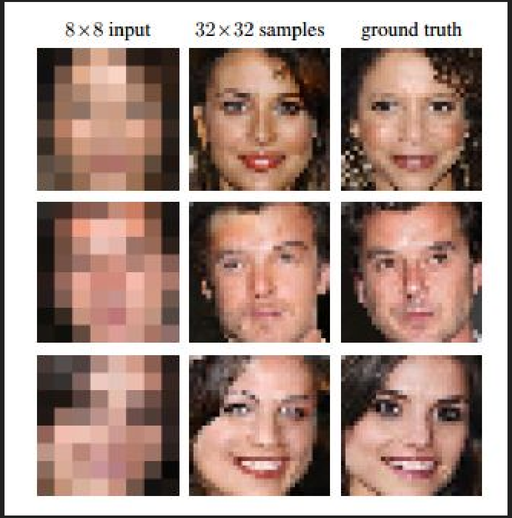
Figure 4. Visualization of the input-to-state and state-to-state mappings for the three proposed architectures.

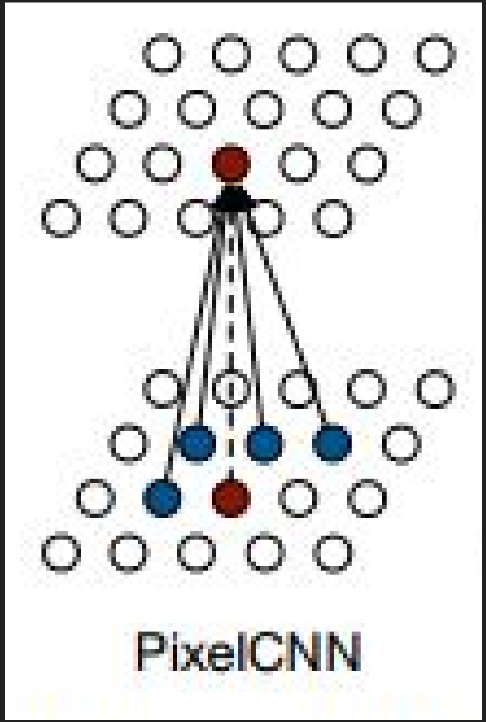
Results on the three architectures

- Diagonal Bi-LSTM outperformed on complicated datasets
  - PixelCNN did the worst on complicated datasets
  - What does this tell us about the role of *receptive fields*?

Back to Pixel Recursive Super Resolution

- They chose the PixelCNN architecture
- Why does this one make the most sense for our problem (as opposed to row and diagonal LSTMs)?





Why a recursive convolutional network?

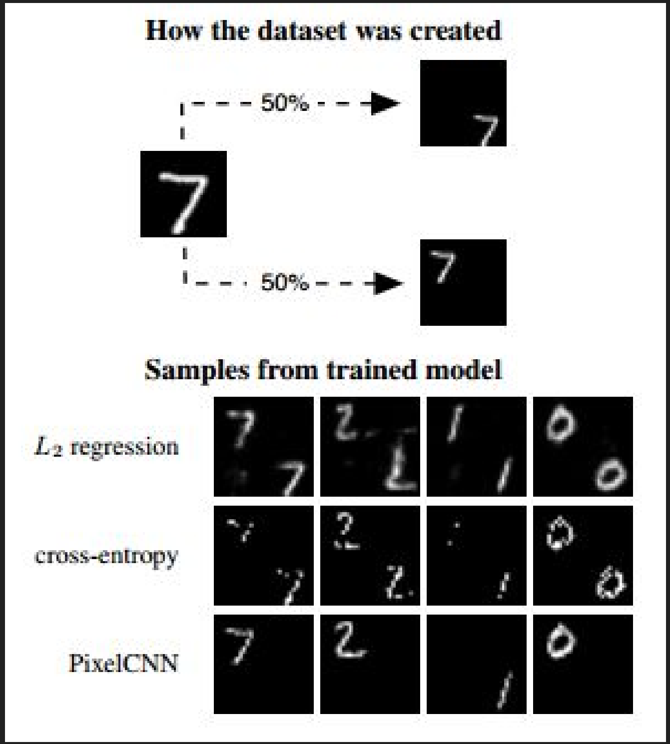
- Why didn’t the researchers simply put the blurry input into a convnet and teach it to predict the true image?

Why a recursive convolutional network?

- Why didn’t the researchers simply put the blurry input into a convnet and teach it to predict the true image?
- Because the pixels are not conditionally independent, and a regular convnet has trouble learning that

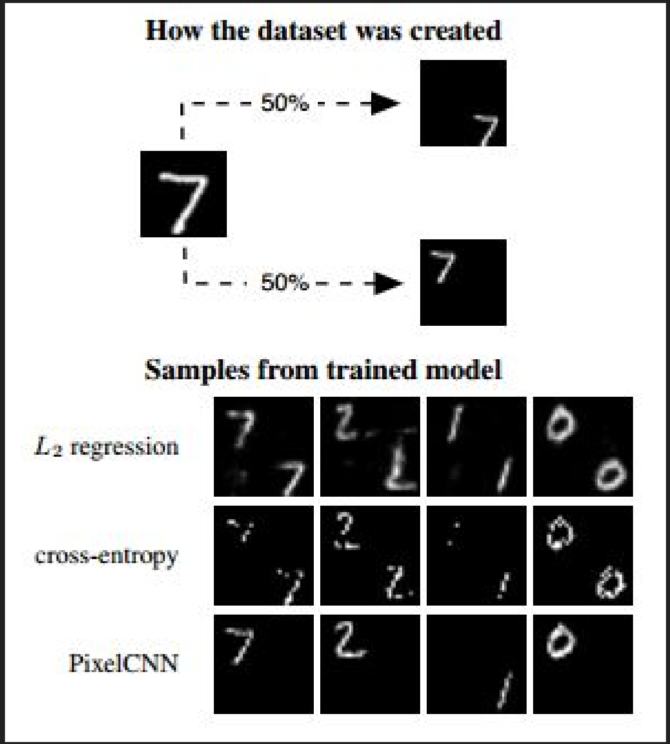
Problems with a regular convnet

- Basically, image generation needs a model that chooses one mode out of several options, rather than taking their average
  - If drawing an apple, we want green or red, not a brown-ish mix of the two
- To prove that PixelCNNs are better for such tasks, the researchers created a dataset of digits in one corner, then trained networks to generate more samples



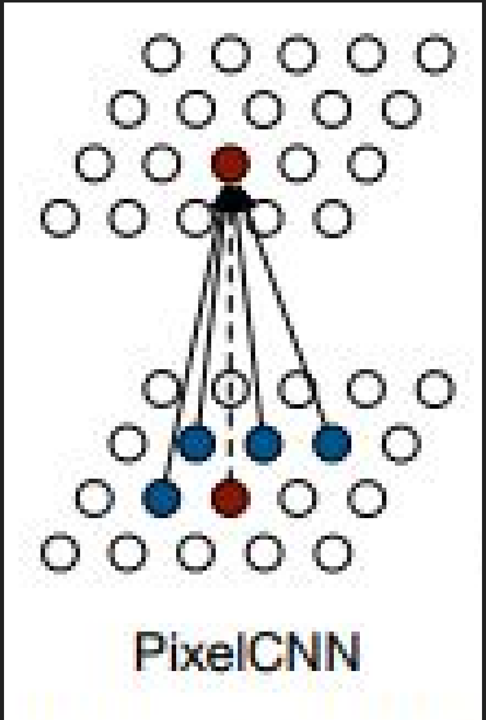
Problems with a regular convnet

- The regular convolutional network *never* generated a sample with just one digit
- The PixelCNN *only* generated samples with one digit



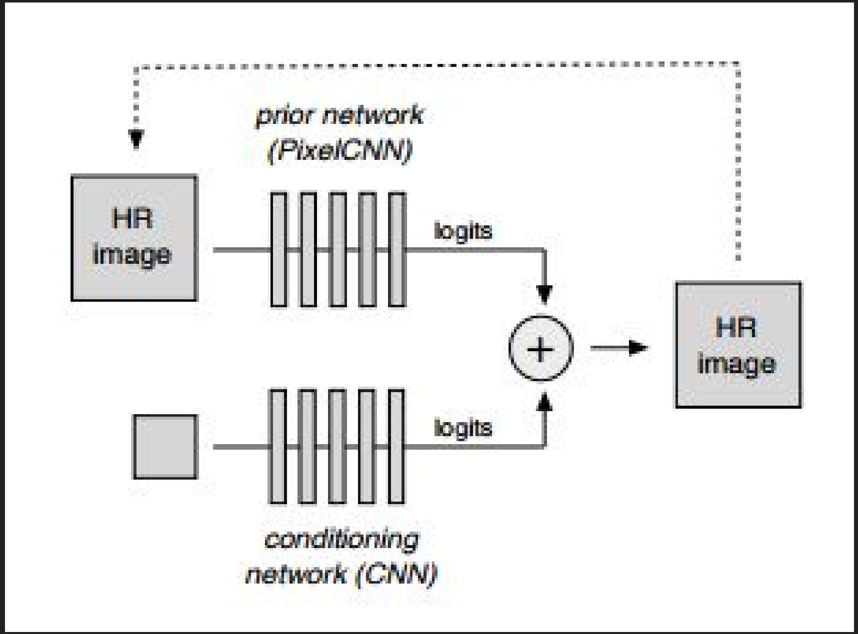
Problems with the PixelCNN

- The PixelCNN, however, doesn't look at the whole low-res image, which we actually have access to (as opposed to the paper where they filled in holes)
- So we want a system that will have access to the whole image but also make distinct decisions while generating
  - What would that look like?



We just add the two together

- How do we take error on the sum?
- How do we separate the partial derivatives?



Error Function

- A_i and B_i are 256 dimensional vectors
  - They contain probability of that color intensity on a scale from 0 to 256
  - Do this for each of the three colors at each pixel
- Softmax is a smooth way of taking the maximum
  - Why does it need to be smooth?
- Do Cross Entropy loss on this p

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

$$p(y_i \mid \mathbf{x}, \mathbf{y}_{<i}) = \mathrm{softmax}(A_i(\mathbf{x}) + B_i(\mathbf{y}_{<i}))\,.$$