Supplementary Materials

Appendix A: Basic Model Architecture

PixelCNN differs from other generative models in that it uses an explicit density function. This means that it uses an explicit specification of the distribution of the random variable [18], most models in machine learning and statistics are in this form [44]. Generative adversarial networks (GAN) instead use an implicit density function, in which a generator implicitly defines a probability distribution based on a latent vector [45, 46]. PixelCNN can further be distinguished from other generative models in that it uses a tractable density function that optimizes the likelihood of the training data. Variational auto-encoders (VAE) on the other hand defines an intractable density, because it makes either variational approximations, Monte Carlo approximations, or both [47].

PixelCNNs also differ greatly from other generative models because they optimize the likelihood of training based on the individual pixels [18]. PixelCNN is an autoregressive model, and predicts pixels one by one and bases the prediction based on previous predictions [18]. In essence, a PixelCNN makes a prediction of what it thinks the intensity of the next pixel will be based on all the previous pixels. An example of this prediction method is given in Figure S1A. The PixelCNN function can be expressed as the product of the probabilities of a pixel based on all previous pixels [18]:

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

In contrast, PixelMiner is not limited to only making predictions based on previous pixels for the external slices, by using non-masked convolutions. An example of this prediction method is given in Figure S1B. The PixelMiner function can be expressed as the product of the probabilities of a pixel based on all previous pixels for the target slice, but all pixels for the external slices :

$$p(t, x, b) = \prod_{i=1}^{n} p_i(x_i|t; x_1, ..., x_{i-1}; b)$$

Where t, x, b are the top, middle, and bottom slices respectively.

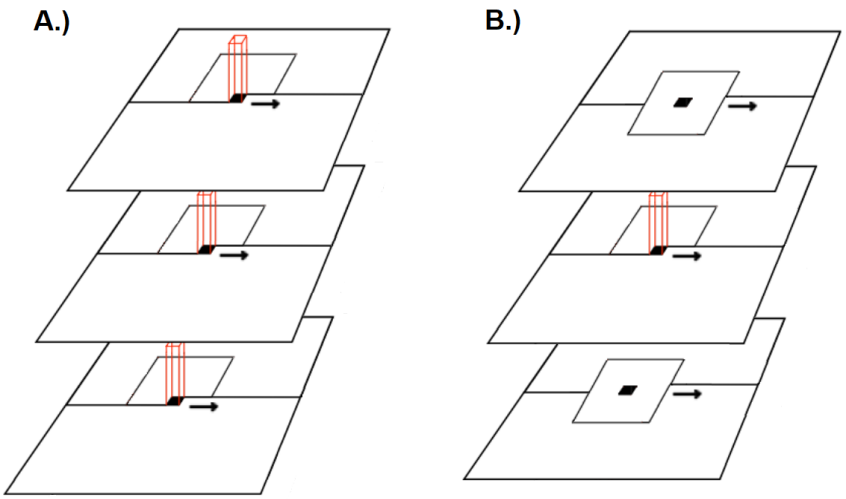

Figure S1: A demonstration of how PixelCNN based models and PixelMiner function. The red column indicates the probability distribution of a pixel. Diagram A is a standard causal PixelCNN. It has three RGB channels with masked convolutions and is trained to predict all three channels. Diagram B is PixelMiner which is a mixed causal noncausal model. It has two unmasked kernels on the top and bottom while only predicting the middle slice.

A major problem with PixelCNN is that there is a blind spot in the receptive field. This is a result of using a mask in a kernel which propagates through the convolution [18]. Where the original PixelCNN was created by zeroing out weights in the convolutions mask, the issue was later overcome by instead creating two separate convolutions named the vertical and horizontal stacks. The vertical stack is for the rows above the pixel being predicted, while the horizontal stack is for the pixels just before the predicted pixel [18].

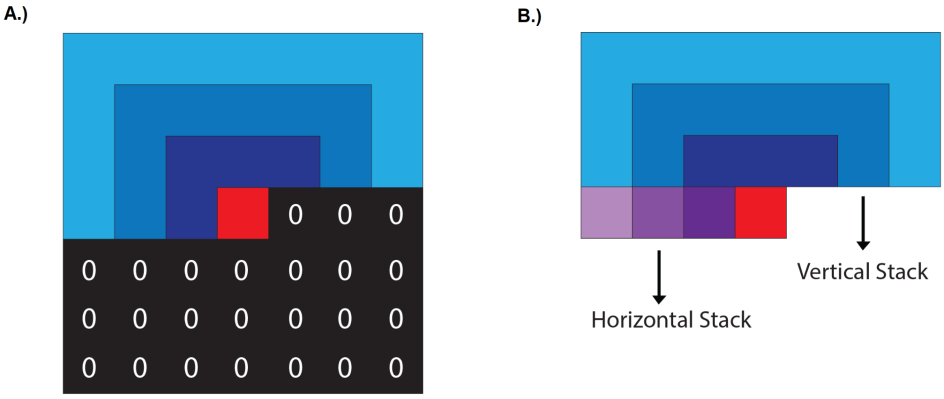

Figure S2: Diagram A represents a masked convolutional kernel with zeroed out weights and diagram B represents the convolutional kernel of PixelCNN, PixelCNN++, and PixelMiner in which the convolution is a combination of two convolutional kernels referred to as the horizontal and vertical stacks.

While this model could be trained using 256 outputs at the final layer output layer with a softmax activation and categorical cross entropy to predict the pixel intensity, there are several

disadvantages to using this method: it is very costly in terms of memory, and it also ignores distances of pixel intensities by treating the data as being categorical. The higher dimensionality also causes gradients to become sparse during training [19]. A solution to these issues is the use of the Discretized Logistic Mixture loss [19], and is performed by predicting a distribution in relation to pixel intensities. It could be done by using a simple distribution such as a normal distribution, but with Discretized Logistic Mixture loss predicts a mixture distribution to better represent the true distribution. The model's output predicts three parameters, the means, coefficients, and logscales times the number of distributions to be mixed for each channel. The maximum likelihood estimate can be used to select the predicted pixel value and then the negative log likelihood is used to determine the loss.

In addition, PixelMiner is a Gated PixelCNN [18], and does not use ReLU activations. It uses a series of operations to create gates similar to long short term memory (LSTM) or gated recurrent units (GRU). The gated activate unit can be represented as:

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

Where $*$ is the convolution operator, $\odot$ is the element-wise product, k is the number of layers, and σ is the sigmoid nonlinearity function.

Theoretically gated convolutional layers provide an advantage by allowing them to access an entire neighborhood of previous pixels using highway networks [18]. Additionally, another potential gain is that gated convolutional layers have multiplicative units which could potentially help Gated PixelCNNs to model complex interactions [18].

Appendix B: Standard Model Results

PixelCNN based models were designed specifically for 2D images with three RGB channels [18][19][41][48]. Because of this, training a PixelCNN model with only two dimensional convolutions causes problems when training the model. During training the model can lead down two different types of paths; it could attempt to predict pixels purely auto-regressively based on only a single slice, or it can focus on factoring in all of the slices to predict the next pixel. The effects of not factoring in all slices will result in images that appear smoother and less noisy, but will be warped with odd artifacts. Figure 1 provides two examples, 1A of an image with warped artifacts, and 1B of an image without. This causes model training to be extremely difficult, as the training needs constant human supervision to ensure that it trains down the correct path, as there is no guarantee that it can be corrected once it deviates.

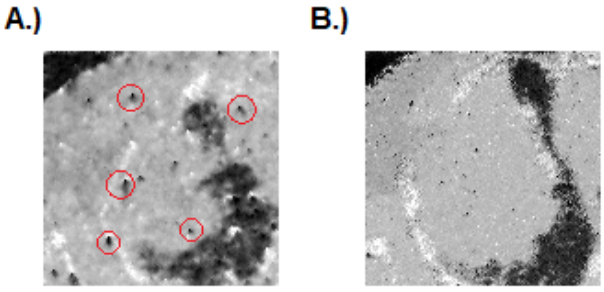

Figure S3: The sample image A is an example of an image generated from a model that should be stopped. The red circles indicate common artifacts that grow downward in a triangular shape off of mistakes. Sample image B is from a model that should continue training. Sample image B is noisier because the model does not try to build upon mistakes in pixel estimation.

Appendix C: Evaluation

Two common measures for evaluating information loss in images include the peak signal to noise ratio (PSNR), and structural similarity index (SSIM) [49]. Both quality measures perform their evaluations as a pixel to pixel comparison not unlike the mean squared error (MSE). PSNR uses MSE explicitly in its equation, whereas SSIM has been shown to be closely related to PSNR through their shared link to MSE [49]. These types of measures may be quite useful for information loss on compression algorithms, but because each individual pixel needs to match the referenced pixel exactly they can be a very poor measure for many generative models, including PixelMiner. This could be demonstrated by measuring an image that has been shifted over one pixel, which will give poor results even though the images are exactly the same, see figure S4.

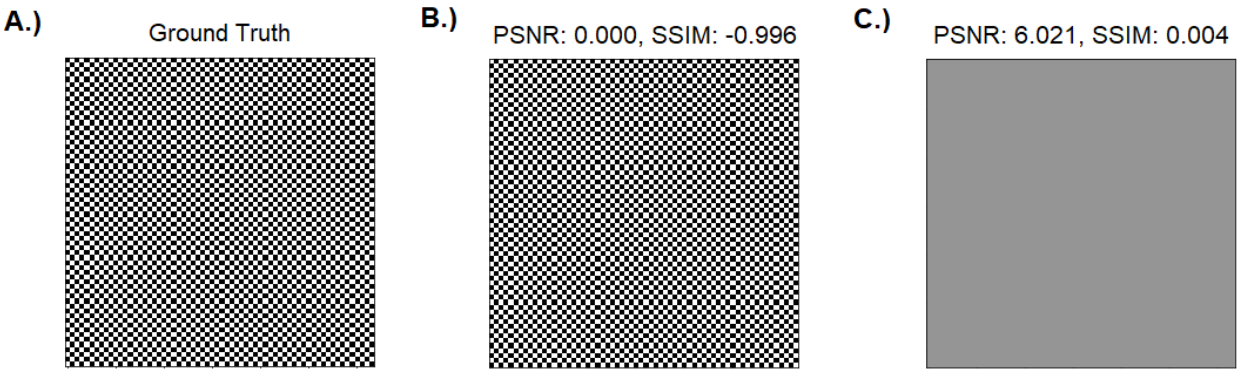

Figure S4: Examples A and B are both identical textures with pixels being unaligned. Example C is the same texture as A but with a very strong gaussian blur. The PSNR and SSIM measure show absolute zero similarity whereas the blurred image on the right shows some relationship to the ground truth.