Parallel Multiscale Autoregressive Density Estimation

Scott Reed [1] **Aäron van den Oord** [1] **Nal Kalchbrenner** [1] **Sergio Gómez Colmenarejo** [1] **Ziyu Wang** [1]
Yutian Chen [1] **Dan Belov** [1] **Nando de Freitas** [1]

Abstract

PixelCNN achieves state-of-the-art results in density estimation for natural images. Although training is fast, inference is costly, requiring one network evaluation per pixel; O(N) for N pixels. This can be sped up by caching activations, but still involves generating each pixel sequentially. In this work, we propose a parallelized PixelCNN that allows more efficient inference by modeling certain pixel groups as conditionally independent. Our new PixelCNN model achieves competitive density estimation and orders of magnitude speedup - $O(\log N)$ sampling instead of $O(N)$ - enabling the practical generation of 512×512 images. We evaluate the model on class-conditional image generation, text-to-image synthesis, and action-conditional video generation, showing that our model achieves the best results among non-pixel-autoregressive density models that allow efficient sampling.

1. Introduction

Many autoregressive image models factorize the joint distribution of images into per-pixel factors:

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t | x_{1:t-1}) \quad (1)$$

For example PixelCNN (van den Oord et al., 2016b) uses a deep convolutional network with carefully designed filter masking to preserve causal structure, so that all factors in equation 1 can be learned in parallel for a given image. However, a remaining difficulty is that due to the learned causal structure, inference proceeds sequentially pixel-by-pixel in raster order.

[1]DeepMind. Correspondence to: Scott Reed <reedscot@google.com>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

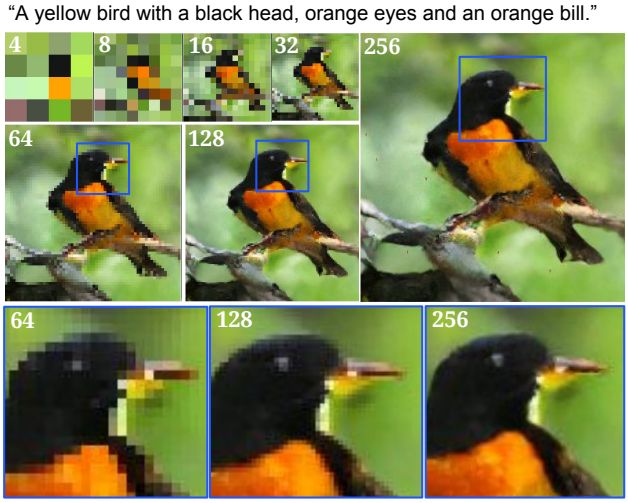

Figure 1. Samples from our model at resolutions from 4×4 to 256×256, conditioned on text and bird part locations in the CUB data set. See Fig. 4 and the supplement for more examples.

In the naive case, this requires a full network evaluation per pixel. Caching hidden unit activations can be used to reduce the amount of computation per pixel, as in the 1D case for WaveNet (Oord et al., 2016; Ramachandran et al., 2017). However, even with this optimization, generation is still in serial order by pixel.

Ideally we would generate multiple pixels in parallel, which could greatly accelerate sampling. In the autoregressive framework this only works if the pixels are modeled as independent. Thus we need a way to judiciously break weak dependencies among pixels; for example immediately neighboring pixels should not be modeled as independent since they tend to be highly correlated.

Multiscale image generation provides one such way to break weak dependencies. In particular, we can model certain groups of pixels as conditionally independent given a lower resolution image and various types of context information, such as preceding frames in a video. The basic idea is obvious, but nontrivial design problems stand between the idea and a workable implementation.

First, what is the right way to transmit global information from a low-resolution image to each generated pixel of the high-resolution image? Second, which pixels can we gen-

erate in parallel? And given that choice, how can we avoid border artifacts when merging sets of pixels that were generated in parallel, blind to one another?

In this work we show how a very substantial portion of the spatial dependencies in PixelCNN can be cut, with only modest degradation in performance. Our formulation allows sampling in $O(\log N)$ time for N pixels, instead of $O(N)$ as in the original PixelCNN, resulting in orders of magnitude speedup in practice. In the case of video, in which we have access to high-resolution previous frames, we can even sample in $O(1)$ time, with much better performance than comparably-fast baselines.

At a high level, the proposed approach can be viewed as a way to merge per-pixel factors in equation 1. If we merge the factors for, e.g. x_i and x_j, then that dependency is "cut", so the model becomes slightly less expressive. However, we get the benefit of now being able to sample x_i and x_j in parallel. If we divide the N pixels into G groups of T pixels each, the joint distribution can be written as a product of the corresponding G factors:

$$p(x_{1:T}^{1:G}) = \prod_{g=1}^{G} p(x_{1:T}^{(g)} | x_{1:T}^{(1:g-1)}) \tag{2}$$

Above we assumed that each of the G groups contains exactly T pixels, but in practice the number can vary. In this work, we form pixel groups from successively higher-resolution views of an image, arranged into a sub-sampling pyramid, such that $G \in O(\log N)$.

In section 3 we describe this group structure implemented as a deep convolutional network. In section 4 we show that the model excels in density estimation and can produce quality high-resolution samples at high speed.

2. Related work

Deep neural autoregressive models have been applied to image generation for many years, showing promise as a tractable yet expressive density model (Larochelle & Murray, 2011; Uria et al., 2013). Autoregressive LSTMs have been shown to produce state-of-the-art performance in density estimation on large-scale datasets such as ImageNet (Theis & Bethge, 2015; van den Oord et al., 2016a).

Causally-structured convolutional networks such as PixelCNN (van den Oord et al., 2016b) and WaveNet (Oord et al., 2016) improved the speed and scalability of training. These led to improved autoregressive models for video generation (Kalchbrenner et al., 2016b) and machine translation (Kalchbrenner et al., 2016a).

Non-autoregressive convolutional generator networks have been successful and widely adopted for image generation as well. Instead of maximizing likelihood, Generative Adversarial Networks (GANs) train a generator network to fool a discriminator network adversary (Goodfellow et al., 2014). These networks have been used in a wide variety of conditional image generation schemes such as text and spatial structure to image (Mansimov et al., 2015; Reed et al., 2016b;a; Wang & Gupta, 2016).

The addition of multiscale structure has also been shown to be useful in adversarial networks. Denton et al. (2015) used a Laplacian pyramid to generate images in a coarse-to-fine manner. Zhang et al. (2016) composed a low-resolution and high-resolution text-conditional GAN, yielding higher quality 256×256 bird and flower images.

Generator networks can be combined with a trained model, such as an image classifier or captioning network, to generate high-resolution images via optimization and sampling procedures (Nguyen et al., 2016). Wu et al. (2017) state that it is difficult to quantify GAN performance, and propose Monte Carlo methods to approximate the log-likelihood of GANs on MNIST images.

Both auto-regressive and non auto-regressive deep networks have recently been applied successfully to image super-resolution. Shi et al. (2016) developed a sub-pixel convolutional network well-suited to this problem. Dahl et al. (2017) use a PixelCNN as a prior for image super-resolution with a convolutional neural network. Johnson et al. (2016) developed a perceptual loss function useful for both style transfer and super-resolution. GAN variants have also been successful in this domain (Ledig et al., 2016; Sønderby et al., 2017).

Several other deep, tractable density models have recently been developed. Real NVP (Dinh et al., 2016) learns a mapping from images to a simple noise distribution, which is by construction trivially invertible. It is built from smaller invertible blocks called coupling layers whose Jacobian is lower-triangular, and also has a multiscale structure. Inverse Autoregressive Flows (Kingma & Salimans, 2016) use autoregressive structures in the latent space to learn more flexible posteriors for variational auto-encoders. Autoregressive models have also been combined with VAEs as decoder models (Gulrajani et al., 2016).

The original PixelRNN paper (van den Oord et al., 2016a) actually included a multiscale autoregressive version, in which PixelRNNs or PixelCNNs were trained at multiple resolutions. The network producing a given resolution image was conditioned on the image at the next lower resolution. This work is similarly motivated by the usefulness of multiscale image structure (and the very long history of coarse-to-fine modeling).

Our novel contributions in this work are (1) asymptotically and empirically faster inference by modeling conditional independence structure, (2) scaling to much higher reso-

lution, (3) evaluating the model on a diverse set of challenging benchmarks including class-, text- and structure-conditional image generation and video generation.

3. Model

The main design principle that we follow in building the model is a coarse-to-fine ordering of pixels. Successively higher-resolution frames are generated conditioned on the previous resolution (See for example Figure 1). Pixels are grouped so as to exploit spatial locality at each resolution, which we describe in detail below.

The training objective is to maximize $\log P(x;\theta)$. Since the joint distribution factorizes over pixel groups and scales, the training can be trivially parallelized.

3.1. Network architecture

Figure 2 shows how we divide an image into disjoint groups of pixels, with autoregressive structure among the groups. The key property to notice is that no two adjacent pixels of the high-resolution image are in the same group. Also, pixels can depend on other pixels below and to the right, which would have been inaccessible in the standard PixelCNN. Each group of pixels corresponds to a factor in the joint distribution of equation 2.

Concretely, to create groups we tile the image with 2×2 blocks. The corners of these 2×2 blocks form the four pixel groups at a given scale; i.e. upper-left, upper-right, lower-left, lower-right. Note that some pairs of pixels both within each block and also across blocks can still be dependent. These additional dependencies are important for capturing local textures and avoiding border artifacts.

Figure 3 shows an instantiation of one of these factors as a neural network. Similar to the case of PixelCNN, at training time losses and gradients for all of the pixels within a group can be computed in parallel. At test time, inference proceeds sequentially over pixel groups, in parallel within each group. Also as in PixelCNN, we model the color channel dependencies - i.e. green sees red, blue sees red and green - using channel masking.

In the case of type-A upscaling networks (See Figure 3A), sampling each pixel group thus requires 3 network evaluations [1]. In the case of type-B upscaling, the spatial feature map for predicting a group of pixels is divided into contiguous $M \times M$ patches for input to a shallow PixelCNN (See figure 3B). This entails M^2 very small network evaluations, for each color channel. We used $M = 4$, and the shallow PixelCNN weights are shared across patches.

[1] However, one could also use a discretized mixture of logistics as output instead of a softmax as in Salimans et al. (2017), in which case only one network evaluation is needed.

The division into non-overlapping patches may appear to risk border artifacts when merging. However, this does not occur for several reasons. First, each predicted pixel is directly adjacent to several context pixels fed into the upscaling network. Second, the generated patches are not directly adjacent in the $2K\times2K$ output image; there is always a row or column of pixels on the border of any pair.

Note that the only learnable portions of the upscaling module are (1) the ResNet encoder of context pixels, and (2) the shallow PixelCNN weights in the case of type-B upscaling. The "merge" and "split" operations shown in figure 3 only marshal data and are not associated with parameters.

Given the first group of pixels, the rest of the groups at a given scale can be generated autoregressively. The first group of pixels can be modeled using the same approach as detailed above, recursively, down to a base resolution at which we use a standard PixelCNN. At each scale, the number of evaluations is $O(1)$, and the resolution doubles after each upscaling, so the overall complexity is $O(\log N)$ to produce images with N pixels.

3.2. Conditional image modeling

Given some context information c, such as a text description, a segmentation, or previous video frames, we maximize the conditional likelihood $\log P(x|c;\theta)$. Each factor in equation 2 simply adds c as an additional conditioning variable. The upscaling neural network corresponding to each factor takes c as an additional input.

For encoding text we used a character-CNN-GRU as in (Reed et al., 2016a). For spatially structured data such as segmentation masks we used a standard convolutional network. For encoding previous frames in a video we used a ConvLSTM as in (Kalchbrenner et al., 2016b).

4. Experiments

4.1. Datasets

We evaluate our model on ImageNet, Caltech-UCSD Birds (CUB), the MPII Human Pose dataset (MPII), the Microsoft Common Objects in Context dataset (MS-COCO), and the Google Robot Pushing dataset.

- For ImageNet (Deng et al., 2009), we trained a class-conditional model using the 1000 leaf node classes.
- CUB (Wah et al., 2011) contains $11,788$ images across 200 bird species, with 10 captions per image. As conditioning information we used a 32×32 spatial encoding of the 15 annotated bird part locations.
- MPII (Andriluka et al., 2014) has around $25K$ images of 410 human activities, with 3 captions per image.

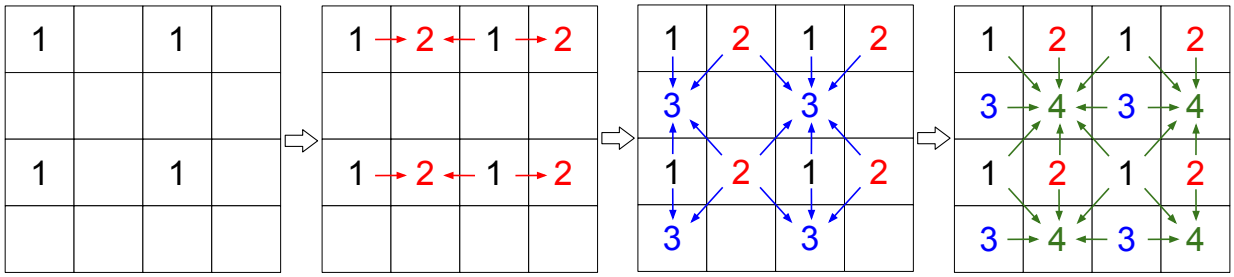

Figure 2. Example pixel grouping and ordering for a 4×4 image. The upper-left corners form group 1, the upper-right group 2, and so on. For clarity we only use arrows to indicate immediately-neighboring dependencies, but note that all pixels in preceding groups can be used to predict all pixels in a given group. For example all pixels in group 2 can be used to predict pixels in group 4. For images, the pixels in group 1 are copied from a lower-resolution image. For video, they can also be generated given the previous frames.

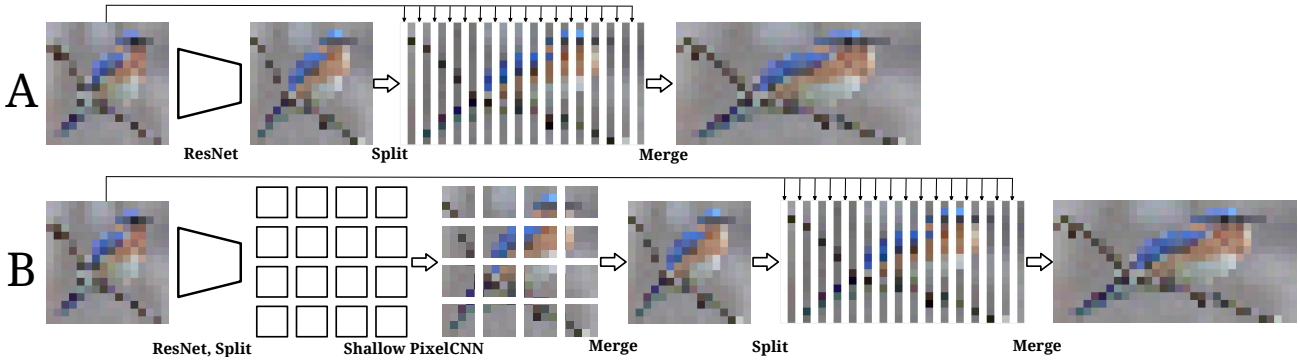

Figure 3. A simple form of causal upscaling network, mapping from a $K \times K$ image to $K \times 2K$. The same procedure can be applied in the vertical direction to produce a $2K \times 2K$ image. In reference to figure 2, we use this type of network for the group $1 \rightarrow 2$ factor. For the $1, 2 \rightarrow 3$ factor, we apply the same technique but in the vertical direction, and subsample the even columns (starting from zero) to get the group 3 pixels. For the $1, 2, 3 \rightarrow 4$ factor, we first build a $2K \times 2K$ input image with group 4 pixels zeroed. This is fed through a spatial dimension-preserving ResNet, from which we subsample the output group 4 pixels. **(A)** In the simplest version, a deep convolutional network (in our case ResNet) directly produces the right image from the left image, and merges column-wise. **(B)** A more sophisticated version extracts features from a convolutional net, splits the feature map into spatially contiguous blocks, and feeds these in parallel through a shallow PixelCNN. The result is then merged as in (A). Note that the entire pathway is trained end-to-end in both cases.

We kept only the images depicting a single person, and cropped the image centered around the person, leaving us about $14K$ images. We used a 32×32 encoding of the 17 annotated human part locations.

- MS-COCO (Lin et al., 2014) has $80K$ training images with 5 captions per image. As conditioning we used the 80-class segmentation scaled to 32×32.
- Robot Pushing (Finn et al., 2016) contains sequences of 20 frames of size 64×64 showing a robotic arm pushing objects in a basket. There are $50,000$ training sequences and a validation set with the same objects but different arm trajectories. One test set involves a subset of the objects seen during training and another involving novel objects, both captured on an arm and camera viewpoint not seen during training.

All models for ImageNet, CUB, MPII and MS-COCO were trained using RMSprop with hyperparameter $\epsilon = 1e-8$, with batch size 128 for $200K$ steps. The learning rate was set initially to $1e-4$ and decayed to $1e-5$.

For all of the samples we show, the queries are drawn from the validation split of the corresponding data set. That is, the captions, key points, segmentation masks, and low-resolution images for super-resolution have not been seen by the model during training.

When we evaluate negative log-likelihood, we only quantize pixel values to $[0, ..., 255]$ at the target resolution, not separately at each scale. The lower resolution images are then created by sub-sampling this quantized image.

4.2. Text and location-conditional generation

In this section we show results for CUB, MPII and MS-COCO. For each dataset we trained type-B upscaling networks with 12 ResNet layers and 4 PixelCNN layers, with 128 hidden units per layer. The base resolution at which we train a standard PixelCNN was set to 4×4.

To encode the captions we padded to 201 characters, then

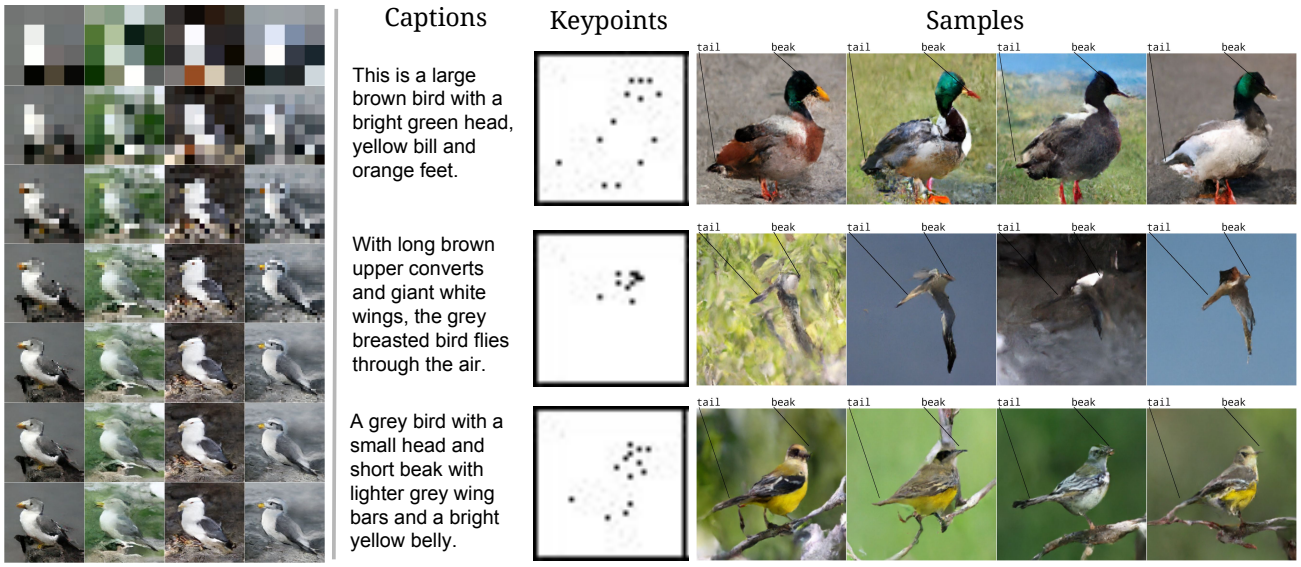

Figure 4. Text-to-image bird synthesis. The leftmost column shows the entire sampling process starting by generating 4×4 images, followed by six upscaling steps, to produce a 256×256 image. The right column shows the final sampled images for several other queries. For each query the associated part keypoints and caption are shown to the left of the samples.

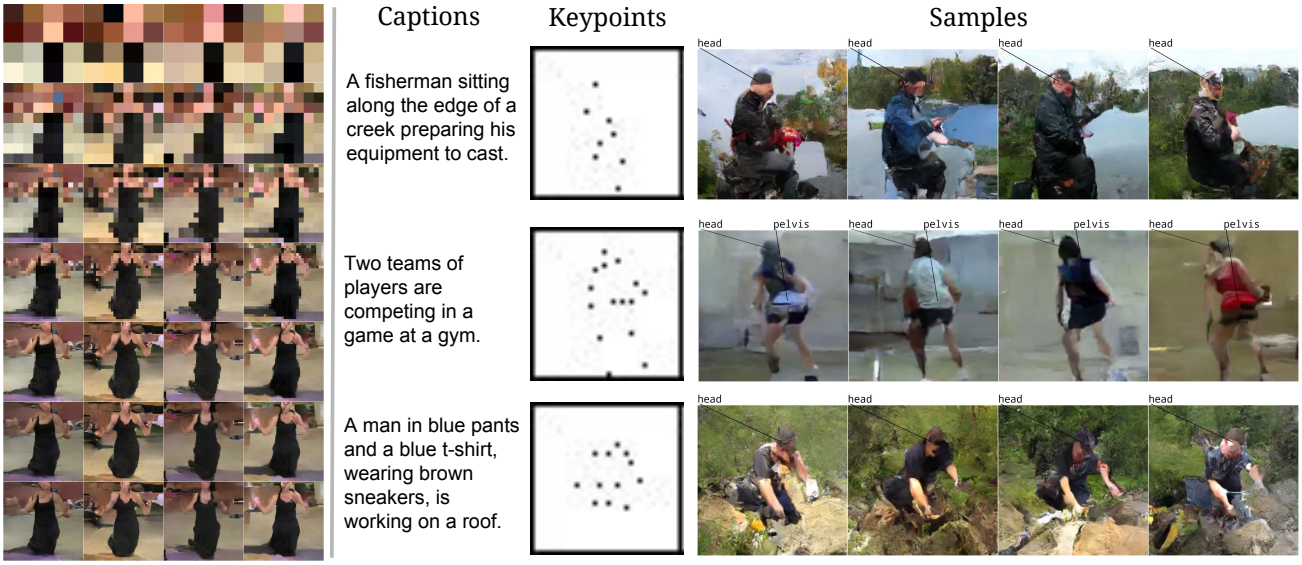

Figure 5. Text-to-image human synthesis.The leftmost column again shows the sampling process, and the right column shows the final frame for several more examples. We find that the samples are diverse and usually match the color and position constraints.

fed into a character-level CNN with three convolutional layers, followed by a GRU and average pooling over time. Upscaling networks to 8×8, 16×16 and 32×32 shared a single text encoder. For higher-resolution upscaling networks we trained separate text encoders. In principle all upscalers could share an encoder, but we trained separably to save memory and time.

For CUB and MPII, we have body part keypoints for birds and humans, respectively. We encode these into a $32 \times 32 \times P$ binary feature map, where P is the number of parts; 17 for MPII and 15 for CUB. A 1 indicates the part is visible, and 0 indicates the part is not visible. For MS-COCO, we resize the class segmentation mask to $32 \times 32 \times 80$.

For comparing and replicating quantitative results, an important detail is the type of image resizing used. For CUB, MPII and MS-COCO we used the default TensorFlow bilinear interpolation resizing to the final image size, by calling `tf.image.resize_images`. Using other resizing methods such as AREA resizing will still result in quality samples, but different likelihoods.

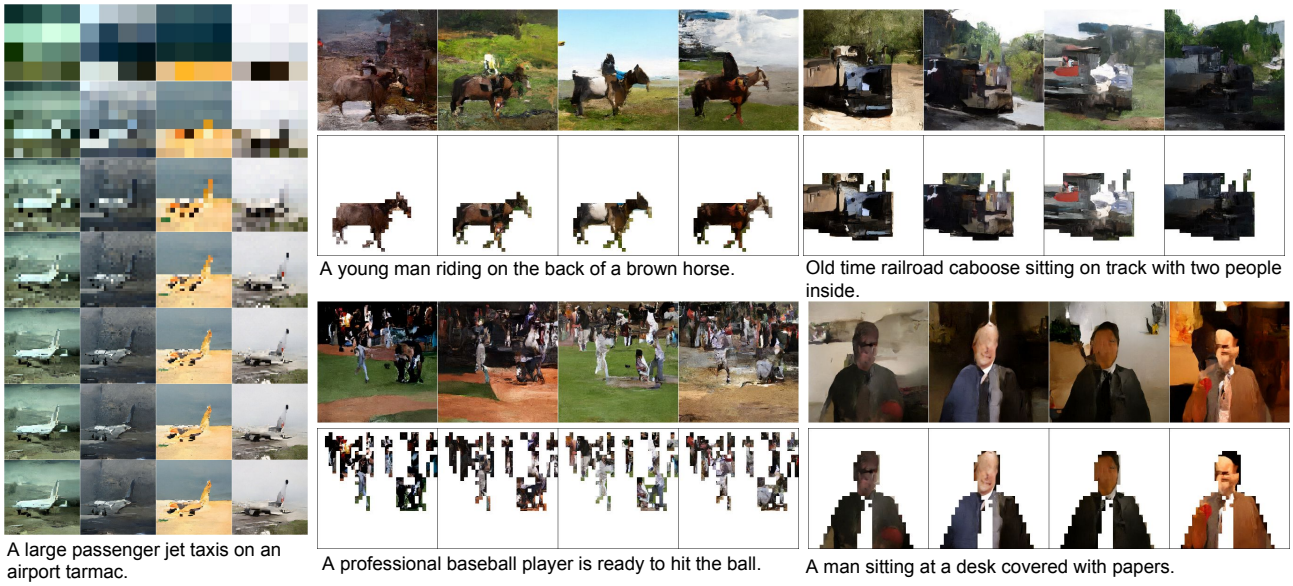

Figure 6. Text and segmentation-to-image synthesis. The left column shows the full sampling trajectory from 4×4 to 256×256. The caption queries are shown beneath the samples. Beneath each image we show the image masked with the largest object in each scene; i.e. only the foreground pixels in the sample are shown. More samples with all categories masked are included in the supplement.

For all datasets, we then encode these spatial features using a 12-layer ResNet. These features are then depth-concatenated with the text encoding and resized with bilinear interpolation to the spatial size of the image. If the target resolution for an upscaler network is higher than 32×32, these conditioning features are randomly cropped along with the target image to a 32×32 patch. Because the network is fully convolutional, the network can still generate the full resolution at test time, but we can massively save on memory and computation during training.

Figure 4 shows examples of text- and keypoint-to-bird image synthesis. Figure 5 shows examples of text- and keypoint-to-human image synthesis. Figure 6 shows examples of text- and segmentation-to-image synthesis.

CUB	**Train**	**Val**	**Test**
PixelCNN	2.91	2.93	2.92
Multiscale PixelCNN	2.98	2.99	2.98
MPII	**Train**	**Val**	**Test**
PixelCNN	2.90	2.92	2.92
Multiscale PixelCNN	2.91	3.03	3.03
MS-COCO	**Train**	**Val**	**Test**
PixelCNN	3.07	3.08	-
Multiscale PixelCNN	3.14	3.16	-

Table 1. Text and structure-to image negative conditional log-likelihood in nats per sub-pixel.

Quantitatively, the Multiscale PixelCNN results are not far from those obtained using the original PixelCNN (Reed et al., 2016c), as shown in Table 1. In addition, we increased the sample resolution by 8×. Qualitatively, the sample quality appears to be on par, but with much greater realism due to the higher resolution.

Based on reviewer feedback, we performed an additional experiment to study how much sample diversity arises from the upscaling networks, as opposed to the base PixelCNN. In the final appendix figure, we show for several 4×4 base images of birds, the resulting upscaled samples. Although the keypoints are fixed so the pose cannot change substantially, we observe a significant amount of variation in the textures, background, and support structure (e.g. tree branches and rocks that the bird stands on). The low-res sample produced by the base PixelCNN seems to strongly affect the overall colour palette in the hi-res samples.

4.3. Action-conditional video generation

In this section we present results on Robot Pushing videos. All models were trained to perform future frame prediction conditioned on 2 starting frames and also on the robot arm actions and state, which are each 5-dimensional vectors.

We trained two versions of the model, both versions using type-A upscaling networks (See Fig. 3). The first is designed to sample in $O(T)$ time, for T video frames. That is, the number of network evaluations per frame is constant with respect to the number of pixels.

The motivation for training the $O(T)$ model is that previous frames in a video provide very detailed cues for predicting the next frame, so that our pixel groups could be conditionally independent even without access to a low-resolution

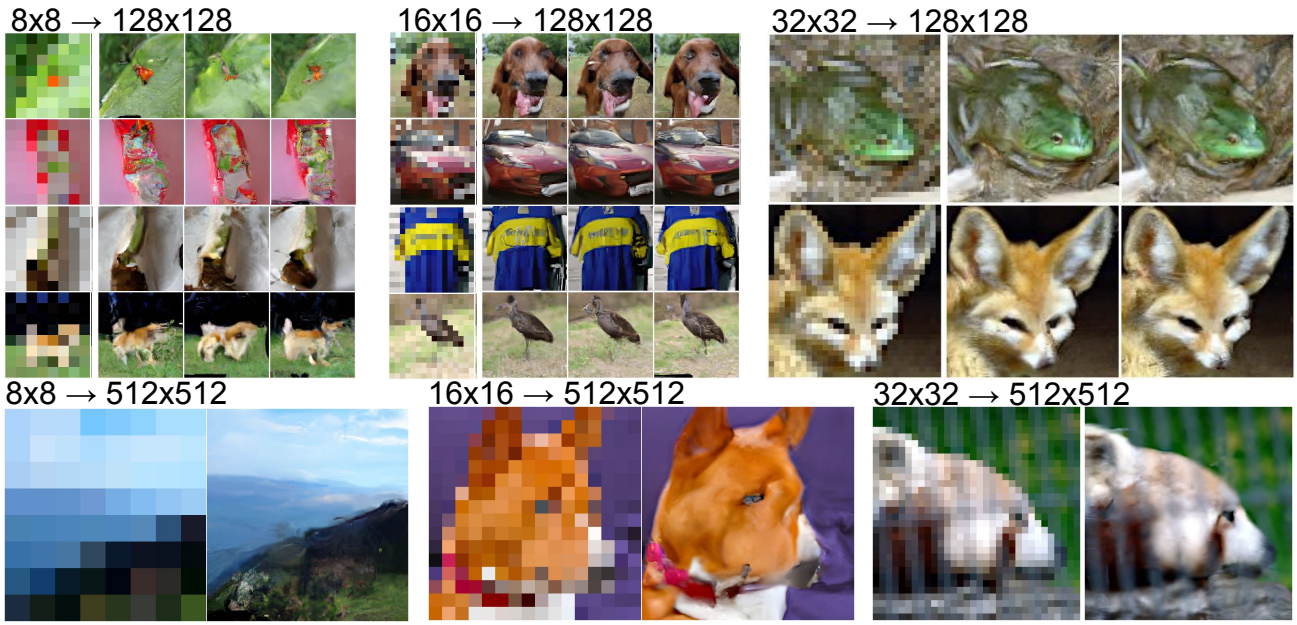

Figure 7. Upscaling low-resolution images to 128×128 and 512×512. In each group of images, the left column is made of real images, and the right columns of samples from the model.

image. Without the need to upscale from a low-resolution image, we can produce "group 1" pixels - i.e. the upper-left corner group - directly by conditioning on previous frames. Then a constant number of network evaluations are needed to sample the next three pixel groups at the final scale.

The second version is our multi-step upscaler used in previous experiments, conditioned on both previous frames and robot arm state and actions. The complexity of sampling from this model is $O(T \log N)$, because at every time step the upscaling procedure must be run, taking $O(\log N)$ time.

The models were trained for $200K$ steps with batch size 64, using the RMSprop optimizer with centering and $\epsilon = 1e{-}8$. The learning rate was initialized to $1e-4$ and decayed by factor 0.3 after $83K$ steps and after $113K$ steps. For the $O(T)$ model we used a mixture of discretized logistic outputs (Salimans et al., 2017) and for the $O(T \log N)$ model we used a softmax ouptut.

Table 2 compares two variants of our model with the original VPN. Compared to the $O(T)$ baseline - a convolutional LSTM model without spatial dependencies - our $O(T)$ model performs dramatically better. On the validation set, in which the model needs to generalize to novel combinations of objects and arm trajectories, the $O(T \log N)$ model does much better than our $O(T)$ model, although not as well as the original $O(TN)$ model.

On the testing sets, we observed that the $O(T)$ model performed as well as on the validation set, but the $O(T \log N)$ model showed a drop in performance. However, this drop does not occur due to the presence of novel objects (in fact this setting actually yields better results), but due to the novel arm and camera configuration used during testing [2]. It appears that the $O(T \log N)$ model may have overfit to the background details and camera position of the 10 training arms, but not necessarily to the actual arm and object motions. It should be possible to overcome this effect with better regularization and perhaps data augmentation such as mirroring and jittering frames, or simply training on data with more diverse camera positions.

The supplement contains example videos generated on the validation set arm trajectories from our $O(T \log N)$ model. We also trained $64 \rightarrow 128$ and $128 \rightarrow 256$ upscalers conditioned on low-resolution and a previous high-resolution frame, so that we can produce 256×256 videos.

4.4. Class-conditional generation

To compare against other image density models, we trained our Multiscale PixelCNN on ImageNet. We used type-B upscaling networks (Seee figure 3) with 12 ResNet (He et al., 2016) layers and 4 PixelCNN layers, with 256 hidden units per layer. For all PixelCNNs in the model, we used the same architecture as in (van den Oord et al., 2016b). We generated images with a base resolution of 8×8 and trained four upscaling networks to produce up to 128×128 samples.At scales 64×64 and above, during training we randomly cropped the image to 32×32. This accelerates training but does not pose a problem at test time because

[2]From communication with the Robot Pushing dataset author.

Model	Tr	Val	Ts-seen	Ts-novel
O(T) baseline	-	2.06	2.08	2.07
O(TN) VPN	-	0.62	0.64	0.64
O(T) VPN	1.03	1.04	1.04	1.04
O(T log N) VPN	0.74	0.74	1.06	0.97

Table 2. Robot videos neg. log-likelihood in nats per sub-pixel. "Tr" is the training set, "Ts-seen" is the test set with novel arm and camera configuration and previously seen objects, and "Ts-novel" is the same as "Ts-seen" but with novel objects.

all of the networks are fully convolutional.

Table 3 shows the results. On both 32×32 and 64×64 ImageNet it achieves significantly better likelihood scores than have been reported for any non-pixel-autoregressive density models, such as ConvDRAW and Real NVP, that also allow efficient sampling.

Of course, performance of these approaches varies considerably depending on the implementation details, especially in the design and capacity of deep neural networks used. But it is notable that the very simple and direct approach developed here can surpass the state-of-the-art among fast-sampling density models.

Model	32	64	128
PixelRNN	3.86 (3.83)	3.64(3.57)	-
PixelCNN	3.83 (3.77)	3.57(3.48)	-
Real NVP	4.28(4.26)	3.98(3.75)	-
Conv. DRAW	4.40(4.35)	4.10(4.04)	-
Ours	3.95(3.92)	3.70(3.67)	3.55(3.42)

Table 3. ImageNet negative log-likelihood in bits per sub-pixel at 32×32, 64×64 and 128×128 resolution.

Interestingly, the model often produced quite realistic bird images from scratch when trained on CUB, and these samples looked more realistic than any animal image generated by our ImageNet models. One plausible explanation for this difference is a lack of model capacity; a single network modeling the 1000 very diverse ImageNet categories can devote only very limited capacity to each one, compared to a network that only needs to model birds. This suggests that finding ways to increase capacity without slowing down training or sampling could be a promising direction.

Figure 7 shows upscaling starting from ground-truth images of size 8×8, 16×16 and 32×32. We observe the largest diversity of samples in terms of global structure when starting from 8×8, but less realistic results due to the more challenging nature of the problem. Upscaling starting from 32×32 results in much more realistic images. Here the diversity is apparent in the samples (as in the data, conditioned on low-resolution) in the local details such as the dog's fur patterns or the frog's eye contours.

Model	scale	time	speedup
$O(N)$ PixelCNN	32	120.0	1.0×
$O(\log N)$ PixelCNN	32	1.17	102×
$O(\log N)$ PixelCNN, in-graph	32	1.14	105×
$O(TN)$ VPN	64	1929.8	1.0×
$O(T)$ VPN	64	0.38	5078×
$O(T)$ VPN, in-graph	64	0.37	5215×
$O(T \log N)$ VPN	64	3.82	505×
$O(T \log N)$ VPN, in-graph	64	3.07	628×

Table 4. Sampling speed of several models in seconds per frame on an Nvidia Quadro M4000 GPU. The top three rows were measured on 32×32 ImageNet, with batch size of 30. The bottom five rows were measured on generating 64×64 videos of 18 frames each, averaged over 5 videos.

4.5. Sampling time comparison

As expected, we observe a very large speedup of our model compared to sampling from a standard PixelCNN at the same resolution (see Table 4). Even at 32×32 we observe two orders of magnitude speedup, and the speedup is greater for higher resolution.

Since our model only requires $O(\log N)$ network evaluations to sample, we can fit the entire computation graph for sampling into memory, for reasonable batch sizes. In-graph computation in TensorFlow can further improve the speed of both image and video generation, due to reduced overhead by avoiding repeated calls to `sess.run`.

Since our model has a PixelCNN at the lowest resolution, it can also be accelerated by caching PixelCNN hidden unit activations, recently implemented b by Ramachandran et al. (2017). This could allow one to use higher-resolution base PixelCNNs without sacrificing speed.

5. Conclusions

In this paper, we developed a parallelized, multiscale version of PixelCNN. It achieves competitive density estimation results on CUB, MPII, MS-COCO, ImageNet, and Robot Pushing videos, surpassing all other density models that admit fast sampling. Qualitatively, it can achieve compelling results in text-to-image synthesis and video generation, as well as diverse super-resolution from very small images all the way to 512×512. Many more samples from all of our models can be found in the appendix and supplementary material.

References

Andriluka, Mykhaylo, Pishchulin, Leonid, Gehler, Peter, and Schiele, Bernt. 2d human pose estimation: New benchmark and state of the art analysis. In *CVPR*, pp. 3686–3693, 2014.

Dahl, Ryan, Norouzi, Mohammad, and Shlens, Jonathon. Pixel recursive super resolution. *arXiv preprint arXiv:1702.00783*, 2017.

Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

Denton, Emily L, Chintala, Soumith, Szlam, Arthur, and Fergus, Rob. Deep generative image models using a Laplacian pyramid of adversarial networks. In *NIPS*, pp. 1486–1494, 2015.

Dinh, Laurent, Sohl-Dickstein, Jascha, and Bengio, Samy. Density estimation using Real NVP. In *NIPS*, 2016.

Finn, Chelsea, Goodfellow, Ian, and Levine, Sergey. Unsupervised learning for physical interaction through video prediction. In *NIPS*, 2016.

Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron C., and Bengio, Yoshua. Generative adversarial nets. In *NIPS*, 2014.

Gulrajani, Ishaan, Kumar, Kundan, Ahmed, Faruk, Taiga, Adrien Ali, Visin, Francesco, Vazquez, David, and Courville, Aaron. PixelVAE: A latent variable model for natural images. *arXiv preprint arXiv:1611.05013*, 2016.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Identity mappings in deep residual networks. In *ECCV*, pp. 630–645, 2016.

Johnson, Justin, Alahi, Alexandre, and Fei-Fei, Li. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016.

Kalchbrenner, Nal, Espeholt, Lasse, Simonyan, Karen, Oord, Aaron van den, Graves, Alex, and Kavukcuoglu, Koray. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016a.

Kalchbrenner, Nal, Oord, Aaron van den, Simonyan, Karen, Danihelka, Ivo, Vinyals, Oriol, Graves, Alex, and Kavukcuoglu, Koray. Video pixel networks. *Preprint arXiv:1610.00527*, 2016b.

Kingma, Diederik P and Salimans, Tim. Improving variational inference with inverse autoregressive flow. In *NIPS*, 2016.

Larochelle, Hugo and Murray, Iain. The neural autoregressive distribution estimator. In *AISTATS*, 2011.

Ledig, Christian, Theis, Lucas, Huszar, Ferenc, Caballero, Jose, Cunningham, Andrew, Acosta, Alejandro, Aitken, Andrew, Tejani, Alykhan, Totz, Johannes, Wang, Zehan, and Shi, Wenzhe. Photo-realistic single image super-resolution using a generative adversarial network. 2016.

Lin, Tsung-Yi, Maire, Michael, Belongie, Serge, Hays, James, Perona, Pietro, Ramanan, Deva, Dollár, Piotr, and Zitnick, C Lawrence. Microsoft COCO: Common objects in context. In *ECCV*, pp. 740–755, 2014.

Mansimov, Elman, Parisotto, Emilio, Ba, Jimmy Lei, and Salakhutdinov, Ruslan. Generating images from captions with attention. In *ICLR*, 2015.

Nguyen, Anh, Yosinski, Jason, Bengio, Yoshua, Dosovitskiy, Alexey, and Clune, Jeff. Plug & play generative networks: Conditional iterative generation of images in latent space. *arXiv preprint arXiv:1612.00005*, 2016.

Oord, Aaron van den, Dieleman, Sander, Zen, Heiga, Simonyan, Karen, Vinyals, Oriol, Graves, Alex, Kalchbrenner, Nal, Senior, Andrew, and Kavukcuoglu, Koray. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Ramachandran, Prajit, Paine, Tom Le, Khorrami, Pooya, Babaeizadeh, Mohammad, Chang, Shiyu, Zhang, Yang, Hasegawa-Johnson, Mark, Campbell, Roy, and Huang, Thomas. Fast generation for convolutional autoregressive models. 2017.

Reed, Scott, Akata, Zeynep, Mohan, Santosh, Tenka, Samuel, Schiele, Bernt, and Lee, Honglak. Learning what and where to draw. In *NIPS*, 2016a.

Reed, Scott, Akata, Zeynep, Yan, Xinchen, Logeswaran, Lajanugen, Schiele, Bernt, and Lee, Honglak. Generative adversarial text-to-image synthesis. In *ICML*, 2016b.

Reed, Scott, van den Oord, Aäron, Kalchbrenner, Nal, Bapst, Victor, Botvinick, Matt, and de Freitas, Nando. Generating interpretable images with controllable structure. Technical report, 2016c.

Salimans, Tim, Karpathy, Andrej, Chen, Xi, and Kingma, Diederik P. PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

Shi, Wenzhe, Caballero, Jose, Huszár, Ferenc, Totz, Johannes, Aitken, Andrew P, Bishop, Rob, Rueckert, Daniel, and Wang, Zehan. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *CVPR*, 2016.

Sønderby, Casper Kaae, Caballero, Jose, Theis, Lucas, Shi, Wenzhe, and Huszár, Ferenc. Amortised MAP inference for image super-resolution. 2017.

Theis, L. and Bethge, M. Generative image modeling using spatial LSTMs. In *NIPS*, 2015.

Uria, Benigno, Murray, Iain, and Larochelle, Hugo. RNADE: The real-valued neural autoregressive density-estimator. In *NIPS*, 2013.

van den Oord, Aäron, Kalchbrenner, Nal, and Kavukcuoglu, Koray. Pixel recurrent neural networks. In *ICML*, pp. 1747–1756, 2016a.

van den Oord, Aäron, Kalchbrenner, Nal, Vinyals, Oriol, Espeholt, Lasse, Graves, Alex, and Kavukcuoglu, Koray. Conditional image generation with PixelCNN decoders. In *NIPS*, 2016b.

Wah, Catherine, Branson, Steve, Welinder, Peter, Perona, Pietro, and Belongie, Serge. The Caltech-UCSD birds-200-2011 dataset. 2011.

Wang, Xiaolong and Gupta, Abhinav. Generative image modeling using style and structure adversarial networks. In *ECCV*, pp. 318–335, 2016.

Wu, Yuhuai, Burda, Yuri, Salakhutdinov, Ruslan, and Grosse, Roger. On the quantitative analysis of decoder-based generative models. 2017.

Zhang, Han, Xu, Tao, Li, Hongsheng, Zhang, Shaoting, Huang, Xiaolei, Wang, Xiaogang, and Metaxas, Dimitris. StackGAN: Text to photo-realistic image synthesis with stacked generative adversarial networks. *arXiv preprint arXiv:1612.03242*, 2016.