# Qumran Letter Restoration by Rotation and Reflection Modified PixelCNN

Lior Uzan Tel Aviv University Tel Aviv, Israel lioruzan@mail.tau.ac.il

Nachum Dershowitz Blavatnik School of Computer Science Blavatnik School of Computer Science Blavatnik School of Computer Science Tel Aviv University Tel Aviv, Israel nachum@cs.tau.ac.il

Lior Wolf Tel Aviv University Tel Aviv, Israel wolf@cs.tau.ac.il

Abstract—The task of restoring fragmentary letters is fundamental to the reading of ancient manuscripts. We present a method to complete broken letters in the Dead Sea Scrolls, which is based on PixelCNN++. Since the generation of the broken letters is conditioned on the extant scroll, we modify the original method to allow reconstructions in multiple directions. Results on both simulated data and real scrolls demonstrate the advantage of our method over the baseline. The implementation may be found at github.com/ghostcow/pixel-cnn-qumran.

## I. INTRODUCTION

The Dead Sea Scrolls date from the third century BCE to the first century CE and are written on parchment or papyrus, mainly in Hebrew (in various scripts), but also in Aramaic and Greek. They are in the process of undergoing high-quality digitization.1

Reconstructing the original texts from extant fragmentary ones is a major aspect of Qumran studies. Broken letters continue to pose a major challenge for scroll researchers. In many cases, research papers are dedicated to the implications of reading a broken or missing letter one way or another. In this work, we train a neural-network model to predict the pixel values of incomplete (broken) letters that are found at the edges of extant fragments.

Our model is based on the PixelCNN++ [1] method. This system, as well as the PixelCNN [2] and PixelRNN [3] methods, are all designed to generate an entire image pixelby-pixel. Therefore, they are typically applied in a raster order from top to bottom and from left to right. In our application, a random part of the letter is missing, and the bulk of the completion takes place in any direction relative to the existing information. We therefore modify the method in order to support arbitrary completion directions.

Our method is of immediate applicability. Due to the deterioration of the scrolls over time, they contain many broken letters, as can be seen in Fig. 1. These letters are usually interpreted by human scholars, deciphered using domain

<sup>1</sup>http://www.deadseascrolls.org.il.

knowledge and context. Although there exists an agreement about some completions, many are debated. Thus, a tool to complete these letters automatically and provide estimated probabilities of different possible reconstructions would be of great value to Oumran researchers.

Accordingly, we present a dataset of whole letters sampled from undamaged parts of the scrolls, and a probabilistic generative model trained on these samples to complete broken letters.

Our contributions are the following:

- a) Collection and annotation of a dataset of whole letters from fragments belonging to manuscript 11Q5, the Great Psalms Scroll.
- b) Collection of edge patches to evaluate the algorithm's ability to complete broken letters.
- c) Baseline results using PixelCNN++ on said dataset.
- d) The development of PixelCNN++ variants that are able to complete equally well in all directions.

The paper is organized as follows: Section II covers data collection; Sections III-A and III-B explain autoregressive modeling and the PixelCNN model, respectively; Section IV is our method for letter completion; and Section V reports on our experiments. It is followed by a brief discussion.

## A. Previous Work

PixelCNN [3] is a powerful class of autoregressive models for modeling natural images. They display diversity [4], are easy to train [1] and are becoming more and more competitive with GANs [5], [6], [7], [8] for generating natural images, while offering inherent advantages such as explicit density modeling. While some experiments on inpainting have been performed using these types of models, usually the inpainted area is a static rectangle [2], [9]. In our data, the missing sections of broken letters changes from letter to letter. This is both a challenge and a blessing. While the variability requires a more comprehensive solution, it also allows us to exploit some regularities from the underlying scroll when generating images.

Pixel order in autoregressive models for image density modeling has been studied in the context of completing MNIST digits [9], yet there too the inpainted area is a regular square. Inpainting has been performed on patches with random

Research supported in part by Grant #01019841 from the Deutsch-Israelische Projektkooperation (DIP), Grant #1330/14 of the Israel Science Foundation (ISF), Grant #I-145-101.3-2013 from the German-Israeli Foundation for Scientific Research and Development (GIF), and a grant from the Blavatnik Family Fund. It forms part of L.U.'s M.Sc. thesis at Tel Aviv University.

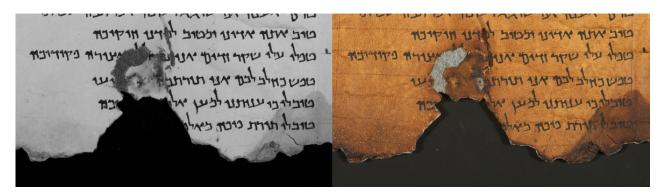


Fig. 1. Deteriorated area in a Qumran scroll. Left: infrared photograph. Right: full-color image.

contours with GANs [10], [11], yet these systems do not provide precise likelihood scores for the generation, which are important for our application.

# II. DATA COLLECTION

Before we discuss our data collecting methods, we present an informal explanation about the digital format of the scrolls. All scrolls are photographed in high resolution at several frequencies, both in visible light and in infrared (IR). Infrared shows ink better, since it negates some discolorations and other imperfections, so we proceed to use the (black and white) IR images exclusively.

The scroll photographs are annotated with a number of parameters. Some, such as plate number, fragment number, and recto/verso, serve to identify the original fragment. Others, such as LED light position, wavelength, date, and time, pertain to the conditions under which the photograph was taken. We seek to extract letters automatically from the images and simulate the broken letters on scroll edges using semisynthetic masks. The data collection process is described in the following subsections.

## A. Whole Letters

The Qumran scrolls are made up of many fragments, surviving in varying degres of quality. To get the most complete and clean letters, we use fragments from PAM plates 974–979, all of which contain excerpts from the *Great Psalms Scroll* (siglum 11Q5). We selected these fragments because they are large and relatively well preserved, one of the best preserved Biblical manuscripts discovered in the Qumran caves.

To extract letters from the scrolls, we use a simple word spotting algorithm [12]. The algorithm receives a binarized image as input and then uses connected components in order to detect, group, and subsequently crop letters from the photographs. Configuration parameters include minimal/maximal component height/width/size.

We note that there is a great variation between images with regard to color intensity levels and size. Even within two photographs of the same fragment there may be great variability. For example, if one photograph is slightly darker than the other, the same binarization threshold may produce clear letters in one image, yet noisy or incoherent letters in

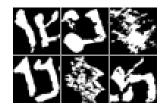


Fig. 2. Noise, broken letters, and frames containing more than one letter were discarded.



Fig. 3. Some samples of the letters in the dataset. First row: whole letters. Second row: artificially broken letters.

the other. This problem hinders efficient data collection, as manually modifying the threshold for each photograph is both time consuming and introduces inconsistencies in letter shape and thickness. We leave this problem for future work. In the version used in this paper, all thresholds were set manually.

The output of the connected component filtering step is a set of rectangles cropped closely to individual letters. Size range is 140–340 over 110–230. We discarded images of non-letters (such as holes in the scroll), broken letters, and images that clearly have more than one whole letter in them. Examples of discarded letters can be seen in Fig. 2.

We then resize the letter images using bicubic interpolation to  $32 \times 32$ , while maintaining the aspect ratio. Samples can be viewed in Fig. 3 (top). Using this process, we extracted a total of 4284 valid complete letter samples. We also attempted to annotate the samples, but the varying style of writing between fragments lead to some incorrectly labeled labels. We did not use these annotations in our experiments; letter statistics for are collected for posterity's sake in Fig. 4.

#### B. Semi-Synthetic Masks

With the whole letter dataset in place, we need a way to evaluate our model on the task of broken letter completion.

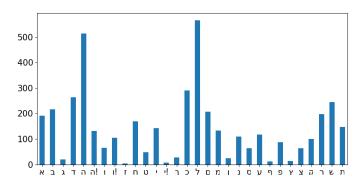


Fig. 4. Histogram of the annotated letters in the dataset. Letters marked with an exclamation mark are Aramaic.

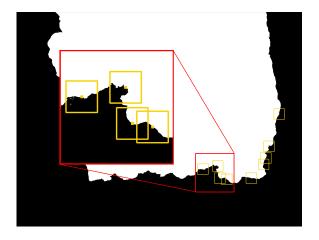


Fig. 5. Mask extraction process.

Unfortunately, average log-likelihood score of the images in some validation set is not a good metric for this task [13], [1], [6]. In completing broken letters we have some information on the letter: we have some existing ink and we know where the edge of the scroll is. The scroll area (with and without the ink) can be viewed as a mask on the image.

To address these points, we generate masks synthetically, and use peak signal-to-noise ratio (PSNR) to evaluate letter completion in a validation set. See Fig. 3 for examples of the end result. To obtain mask patches we first select a fragment with highly deteriorated edges, yet large enough to produce a multitude of samples. One such fragment is fragment 004 of plate 976. We clear the contents of the fragment, then crop randomly selected patches of size  $256 \times 256$  from the edge. The masks are resized to  $32 \times 32$  and assigned, one mask per letter, to the letters in the validation set. The method is described in Fig. 5, and final samples can be viewed in Fig. 3.

### C. Training and Validation Sets

To create a validation set, we must sample uniformly from our data. Yet, some fragments are so large that they are photographed in parts, and these parts overlap, so we cannot share letters from the same fragment between training and validation sets without risking contamination. We take 147 letters from fragment 4 of plate 976 for validation, and leave the rest (4137) for the training set. Further implementation details can be found in the released code.

### D. Real Broken Letters

To verify that the system works, we compiled a list of 19 broken letters for which there is a scholarly agreement on their correct readings. We located them and cropped their images from the corresponding fragments. Patches were extracted from the infrared photographs of the fragments like those in the dataset. We cleaned the patches by deleting all nonscroll textures, leaving only the ink and some parchment. We then proceeded to binarize and color-invert the patch to conform with the data format of the letters in the dataset. The binary mask was created by whitening the leftover parchment area. Finally, the letters and masks were resized to  $32 \times 32$ while preserving the aspect ratio of the images. Due to the preservation of the aspect ratio, resizing created margins. We extended the mask to these margins because nothing is supposed to be generated there. The evaluation protocol will be described in detail in Section V.

#### III. MODEL

# A. Autoregressive Generative Models

To explain PixelCNN, we must first explain what an autoregressive generative model is. An autoregressive generative model is a generative model that models the probability of multivariate data by modeling the conditional probabilities of each dimension on all previous dimensions (given some ordering). See [9] for an in-depth review of such models. In our case, the data is the images of whole letters from the Qumran scrolls, and the individual dimensions correspond to the pixels in the image. To be precise, let **x** be an image of size  $n \times n$ . Define  $x_1, \ldots, x_{n^2}$  to be the pixels taken from the image row by row in raster scan order. We assign a probability  $p(\mathbf{x}) = p(x_1, x_2, \ldots, x_{n^2})$  to the image, which by repeated application of Bayes' theorem can be written as:

$$p(\mathbf{x}) = \prod_{n=1}^{n^2} p(x_i | x_1, x_2, \dots, x_{i-1}).$$
 (1)

The value  $p(x_i|x_1, x_2, \ldots, x_{i-1})$  is the probability of the *i*th pixel  $x_i$  given all the previous pixels  $x_1, \ldots, x_{i-1}$ . These conditional probabilities over all possible pixel values  $x_i \in \{0, \ldots, 255\}$  are modeled by a neural network and are sampled according to it when generating an image. When sampling whole images from such a model, one begins by feeding an empty image (i.e. an array of size  $n \times n$  zeros), then sampling from the probabilities over pixel values generated for the top left corner pixel. Then, this pixel value is copied back to the input image, and the process is repeated to sample the second pixel, and so on. This process is repeated  $n^2$  times until all pixels have been sampled.

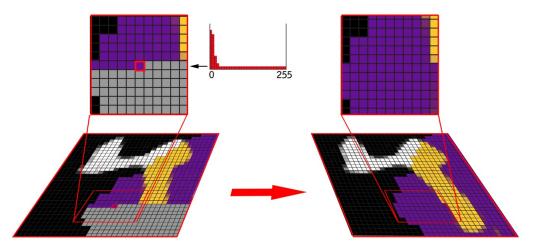


Fig. 6. PixelCNN algorithm. In this example, PixelCNN completes a broken *dalet*. Yellow and purple represent the unknown areas; white and black are known parts of the image; gray denotes areas to be completed. Each completion is a grayscale value between 0 and 255. The process is finished when all missing pixels have been completed.

The process for completing a *broken* letter is the same, except known pixels are not overridden. See Fig. 6 for a description of the process.

# B. PixelCNN and PixelCNN++

PixelCNN, first described in [2] an autoregressive generative model, designed for image data. In essence, it is a convolutional neural network [14] with "masked" convolutions, used to model the conditional probabilities of each pixel  $p(x_i|x_1, x_2, \ldots, x_{i-1})$ . The model we use in this work is the improved version called PixelCNN++ [1]. We now explain the model components in greater detail.

1) Masked Convolutions and the Dual Stream: In our scenario, each pixel is considered a random variable that takes on values from  $\{0, \ldots, 255\}$ . Each output must receive information only from pixels before it in order. Thus, each convolution must be masked such that the output at each step does not see "forward" pixels (Fig. 7). The size of the filter in each layer is small, however, the receptive field increases with depth, and after a sufficient number of layers, each output receives information from all pixels before it. Some problems arise when using masked convolutions regarding receptive field growth. The shape of the masked convolution creates "dead zones" in the receptive field, from which no information is received. The solution given in [2] is to create two "streams" of convolutions, one with a filter that expands the RF to pixels above and to the right of current pixel, and one that expands up and to the left. For brevity we omit details; please refer to [2] for further explanation.

2) *Gated Convolutions:* The rectified linear units between convolutions are replaced with multiplicative units, or gated activation units:

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

Where k is the layer number,  $\sigma$  the logistic sigmoid function,  $\odot$  the element-wise product, and \* is the convolutional

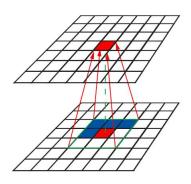


Fig. 7. Masked convolution. The center input pixel is used in all but the first layer.

operator. At the output, in our case, each pixel has one value: intensity. This is modeled as a multiclass classification problem where the classes are  $\{0, \ldots, 255\}$ . The network outputs are set to size  $n \times n \times 256$ . Let the output at location  $i \in \{1, \ldots, n^2\}$  be  $o \in \mathbb{R}^{256}$ . Let  $o = (o_0, \ldots, o_{255})$ , then the probability pixel *i* receives value *k* is defined by the Softmax function:

$$P(x_i = k) = \frac{e^{o_k}}{\sum_{j=0}^{255} e^{o_j}},$$

and is optimized using the conventional cross entropy loss.

#### C. PixelCNN++ Architectural Improvements

PixelCNN++ [2] improves upon the original PixelCNN architecture, without changing the underlying method.

1) Subsampling for Expanding Receptive Field: The authors of [1] use subsampling within the network to further enlarge the receptive field of network layers. Subsampling has been shown to expand the receptive field of convolutions both theoretically and effectively [15]. In contrast with PixelCNN, where all convolutions retained spatial resolution of the input, PixelCNN++ downsamples twice by a factor of 2, then upsamples twice to regain the original resolution. 2) Discretized Logistic Mixture Likelihood: The Softmax distribution, while extremely flexible, is costly in terms of memory and inhibits efficient optimization in various ways [1]. PixelCNN++ instead uses discretized logistic mixture to model conditional pixel probabilities, with the loss function being simply the likelihood of the pixel values in the image.

# IV. MODIFYING PIXELCNN++ FOR BROKEN LETTERS

Our goal is to improve broken letter completion over a PixelCNN++ baseline, using existing information about scroll orientation. One possible approach is to increase the amount of known scroll pixels the model is conditioned upon at the start of the generation sequence. This can be done by reordering the pixels, such that the known pixels will be at the start of the sequence. However, by changing the pixel order arbitrarily, we lose the ability to exploit 2D topology information. This is crucial for PixelCNN, because it's convolutional layers rely on local pixel correlations to learn [14].

Our approach is instead to adaptively rotate and reflect the broken letters in a way that advances known pixels from the scroll up the order of completion. We train a baseline of PixelCNN++ and three variants to this end:

- Single model adaptive orientation.
- Single model adaptive orientation; conditional.
- Multiple model adaptive orientation.

# A. Adaptive Orientation

We notice that when completing letters on the edges of scrolls, the mask will consistently be attached to one of the sides of the letter (see Fig. 5). Thus, transforming the image using reflections and rotations of 90 degrees, we can have the mask be in the top part of the image. This order places most of the known pixels first. For example, in Fig. 8 the orientation that maximizes the number of known pixels in the top rows, is the orientation on the second row, second column. This guarantees an increased amount of known pixels be conditioned on, without deforming or breaking 2D topology structure. In addition, these orientations comprise an extremely small number of transformations, relative to the possible space of pixel permutations.

Let the center of mass of some mask be  $c_x, c_y$ , and  $\tilde{c_x}, \tilde{c_y}$  be the the center of mass after some combination of rotations and reflections. We define this sequence of rotations/reflections to be correct if  $0 \le \tilde{c_x} \le n/2$ ,  $\tilde{c_y} \le \tilde{c_x}$  in image coordinates. Since the image can be rotated and reflected so that any single point will be in the aforementioned area, one can find a rotation-reflection sequence simply by trying all 8 options. When completing a broken letter, we rotate it first to the correct orientation, then complete the missing pixels as explained in Section III-A.

# B. The PixelCNN++ Variants in Our Experiments

1) Unmodified PixelCNN++ Baseline: To use the regular PixelCNN++ on our data, we make the changes described earlier in Section IV, and simply feed our data to the model. Precise hyper-parameters are specified in Section V-A.

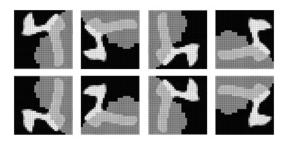


Fig. 8. Different orientations. Our method selects the sixth orientation (second row, second column) to complete the gray area.

2) Single Model Adaptive Orientation: In this variant, we rotate each minibatch of images randomly every training iteration. The idea is that once shown a broken letter to complete in some orientation, the model would deduce which one it is and complete it accordingly. This can also be an issue, because we do not control the orientation "guessed" by the model. We attempt to rectify this in the next variant.

3) Single Model Adaptive Orientation, Conditional: Let h be a latent vector that describes some high-level property in an image. We seek to model p(x|h), to be able to generate samples influenced by this property. The previous model was trained on all orientations, yet the orientation cannot be explicitly stated during completion. We attempt to solve this problem by setting h to be a label describing the correct orientation of the input. We train an adaptive orientation model conditioned on the current sample orientation. The network architecture is described in Section V-A.

4) Multi-model Adaptive Orientation: In this variant, we train seven more models in addition to the baseline, each one with a fixed orientation, totaling eight models – one per orientation. At test time, we select which model to use on each sample based on its correct orientation. This can be seen as an alternative method to controlling in what direction a letter should be completed.

### V. EXPERIMENTS

## A. Model Specifications

We first perform model selection for all methods by selecting the top performing models in bits/dim on the validation set, as in [3], [2], [1]. Bits/dim is the total discrete loglikelihood, normalized by the image dimensions (in our case  $32 \times 32 = 1024$ ). This represents the number of bits that a compression scheme based on this model would need to compress every pixel color value [13].

All models share the same basic modified PixelCNN++ architecture described in Section IV. Namely, there are six blocks of five ResNet style, two-stream, gated masked convolutional layers. Two downsamples and two upsamples are performed between blocks (1, 2), (2, 3), (4, 5), (5, 6) via strided convolutions and transposed strided convolutions, respectively. In addition to this, skip connections are used to connect blocks of same spatial resolution. Logistic mixtures of five components are used for the probability modeling. Dropout is used in each ResNet style layer between convolutions, with dropping probability p = 0.5. For the set orientation models, we use 30 filters per convolution; for the multi-orientation single models we use 80 filters. In the conditional model, conditioning is done as described in [1], via dynamic biases in each layer. To each layer is added a bias dependent on the current orientation label.

# B. Evaluation Methods

We perform two sets of evaluations. First, an evaluation on broken letters from our validation set, where we know the ground truth. Second, we evaluate real broken letters, where we use human evaluation to measure performance, as we don't have the whole letter as ground truth.

In the first set of experiments, we employ the synthetic masks on the test set in order to simulate broken letters. For this benchmark, we complete the letters of the entire validation set using the various methods, compare the completions with the corresponding ground truth, which contains the whole letters, and measure PSNR. Since we sample from our model probabilistically, a single sample might not yield the correct or best completion, so each result reported is the mean average PSNR over the test set, over 10 runs.

In the second experiment, the baseline and the best performing model is used to complete a held out test set of real broken letters. With each model, every letter in the test set is completed 20 times. In total,  $2 \times (20 \times 19) = 760$  samples are generated. We only have information on the test letters based on expert consensus, because they are broken in the Qumran scrolls. We therefore employ human reading of the completion in order to measure our methods' success. The human expert reviews the letters from the methods tested in random order, proceeds to label them according to what letter s/he sees. Unidentifiable letters are not labeled. We then compare these labels to the ground truth. The samples are then sorted by likelihood, as reported by the model.

#### C. Training Details

For each method tested, early stopping was used. Batch size was a constant 48; each model was trained on a single GPU. Polyak averaging was applied on the parameters; geometric learning rate decay with a factor of 0.9999995 was used for the set orientation models and a 0.9999995 decay rate for the single-model methods. The fixed-orientation models reached the low point in validation loss after about 200 epochs, the single-model adaptive orientation at 556, and the conditional version after 697 epochs.

#### D. Results on the Validation of Broken Letters

Results are given in Table I. All models trained on a single orientation reach approximately the same results on training and validation sets. The small gap between the two losses indicates that there was no overfitting in either. Clearly, the single model method with adaptive orientation is superior to any fixed orientation model, both in terms of bits/dim and in terms of PSNR. Also, we see that the multi-model method is comparable to the single model version. Finally, the conditional experiment failed, producing poor PSNR results. The model still achieves good bits/dim results, perhaps indicating some form of overfitting not captured by this metric. Operatively, we notice that the conditioning mechanism does not work well. When we attempt to set the orientation, we frequently get a letter in some other orientation altogether.

# E. Results on Real Broken Letters

As we have seen (Table I), the adaptive single- and multimodels achieve the best PSNR results. After observing several completions of validation broken letters, we noticed that the adaptive single model may complete a letter fixed to one orientation as a completely different orientation. This behavior was observed also in the conditional model. Since our work is application driven, we drop these models from the final evaluation. (Researchers should not be confused by unexpected flips or rotations.) Therefore, when we tested and evaluated on real broken letters, we included only the "no flip or rotation" baseline and the adaptive orientation multi-model.

Fig. 9 graphs the average correct prediction rate per "top k" most-likely samples. The correct prediction rate is taken per letter amongst the top k scoring samples, then an average is taken over all test letters. As seen, our model significantly outperforms the baseline in accuracy, with 20% correct completions on average. Looking at lower-ranked completions does not help.

We tried also to measure sample diversity by reporting correct generations per "top k" completions (Fig. 11). For each of the 19 test letters, we judged a group of generations to be successful if any of the top k completions were identified as correct by a human evaluator. When selecting the most likely examples, the adaptive orientation method outperforms the baseline at most values of k. Interestingly, when selecting the *most likely* sample in the baseline model, we get only 1 correct completion out of the 19, versus 4 correctly completed letters by the adaptive multi-model. The latter plateaus with the correct answer among the top 11 completions in 7 out of the 19 cases.

# VI. CONCLUSIONS

We presented a dataset and a novel approach to modeling broken letters from the edges of the ancient Qumran scrolls. Our method exploits scroll structure to improve letter completion. Results were reported for a benchmark of artificiallycreated broken letters, as well as for a benchmark of real broken letters on which there is scholarly agreement. For both benchmarks, our method clearly outperforms a strong PixelCNN++ baseline. Future work may leverage recent advancements in the field to generate samples faster [8] and in higher resolution [5].

#### REFERENCES

 T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," *arXiv Preprint arXiv:1701.05517*, 2017.

RESULTS OF VARIOUS METHODS. TRAINING AND VALIDATIONS LOSS VALUES ARE REPORTED IN BITS/DIM. PSNR IS EVALUATED OVER 10 RUNS; THE STANDARD DEVIATION OF EACH IS IN PARENTHESES.

| Model                              | Adaptive | Train Loss | Val. Loss | Mean avg PSNR | Mean PSNR SD |
|------------------------------------|----------|------------|-----------|---------------|--------------|
| No flip or rotation                | No       | 1.22       | 1.38      | 13.33 (0.20)  | 4.86 (0.22)  |
| 90° rotation                       | No       | 1.21       | 1.39      | 11.88 (0.11)  | 4.56 (0.19)  |
| 180° rotation                      | No       | 1.21       | 1.38      | 10.10 (0.14)  | 4.07 (0.30)  |
| $270^{\circ}$ rotation             | No       | 1.21       | 1.39      | 12.26 (0.07)  | 4.55 (0.29)  |
| Hor. Flip                          | No       | 1.22       | 1.38      | 13.37 (0.17)  | 5.12 (0.22)  |
| Hor. Flip + $90^{\circ}$ rotation  | No       | 1.22       | 1.39      | 12.71 (0.18)  | 4.77 (0.36)  |
| Hor. Flip + $180^{\circ}$ rotation | No       | 1.22       | 1.38      | 10.19 (0.13)  | 4.10 (0.39)  |
| Hor. Flip + $270^{\circ}$ rotation | No       | 1.22       | 1.39      | 11.41 (0.12)  | 4.76 (0.16)  |
| Multi-model                        | Yes      | -          | -         | 15.54 (0.14)  | 4.70 (0.29)  |
| Single Model                       | Yes      | 1.19       | 1.33      | 15.55 (0.14)  | 4.61 (0.32)  |
| Single Model, Conditional          | Yes      | 1.20       | 1.33      | 14.72 (0.09)  | 4.68 (0.18)  |

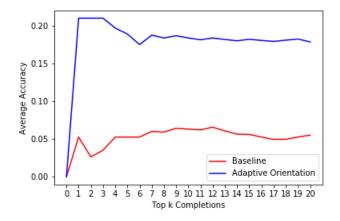


Fig. 9. Average accuracy per top k likely completions. The average is over the letters in the real letters test set.

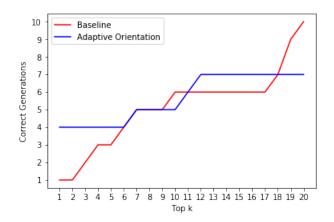


Fig. 10. Comparison of successful generations as a function of the most probable completions in each model.

- [2] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," arXiv Preprint arXiv:1601.06759, 2016.
- [3] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional image generation with PixelCNN decoders," June 2016. [Online]. Available: http: //arxiv.org/abs/1606.05328

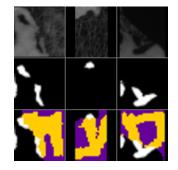


Fig. 11. Examples of completions from the test set, using Adaptive Orientation. First row: original broken letters. Second row: processed versions. Third row: the completions, colored to show the completed area.

- [4] S. Reed, A. van den Oord, N. Kalchbrenner, V. Bapst, M. Botvinick, and N. de Freitas, "Generating interpretable images with controllable structure," 2016, draft.
- [5] S. Reed, A. v. d. Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, D. Belov, and N. de Freitas, "Parallel multiscale autoregressive density estimation," arXiv Preprint arXiv:1703.03664, 2017.
- [6] R. Dahl, M. Norouzi, and J. Shlens, "Pixel recursive super resolution," arXiv Preprint arXiv:1702.00783, 2017.
- [7] A. Kolesnikov and C. H. Lampert, "Latent variable PixelCNNs for natural image modeling," 2016.
  [8] P. Ramachandran, T. L. Paine, P. Khorrami, M. Babaeizadeh,
- [8] P. Ramachandran, T. L. Paine, P. Khorrami, M. Babaeizadeh, S. Chang, Y. Zhang, M. A. Hasegawa-Johnson, R. H. Campbell, and T. S. Huang, "Fast generation for convolutional autoregressive models," *CoRR*, vol. abs/1704.06001, 2017. [Online]. Available: http://arxiv.org/abs/1704.06001
- [9] B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle, "Neural autoregressive distribution estimation," *Journal of Machine Learning Research*, vol. 17, no. 205, pp. 1–37, 2016.
- [10] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2536–2544.
- [11] C. Yang, X. Lu, Z. Lin, E. Shechtman, O. Wang, and H. Li, "Highresolution image inpainting using multi-scale neural patch synthesis," arXiv Preprint arXiv:1611.09969, 2016.
- [12] A. Kovalchuk, L. Wolf, and N. Dershowitz, "A simple and fast word spotting method," in *Frontiers in Handwriting Recognition (ICFHR)*, 2014.
- [13] L. Theis, A. v. d. Oord, and M. Bethge, "A note on the evaluation of generative models," arXiv Preprint arXiv:1511.01844, 2015.
- [14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [15] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 4898–4906.

TABLE I