An intriguing failing of convolutional neural networks and the CoordConv solution

Rosanne Liu[1] **Joel Lehman**[1] **Piero Molino**[1] **Felipe Petroski Such**[1] **Eric Frank**[1]

Alex Sergeev[2] **Jason Yosinski**[1]

[1]Uber AI Labs, San Francisco, CA, USA [2]Uber Technologies, Seattle, WA, USA

{rosanne,joel.lehman,piero,felipe.such,mysterefrank,asergeev,yosinski}@uber.com

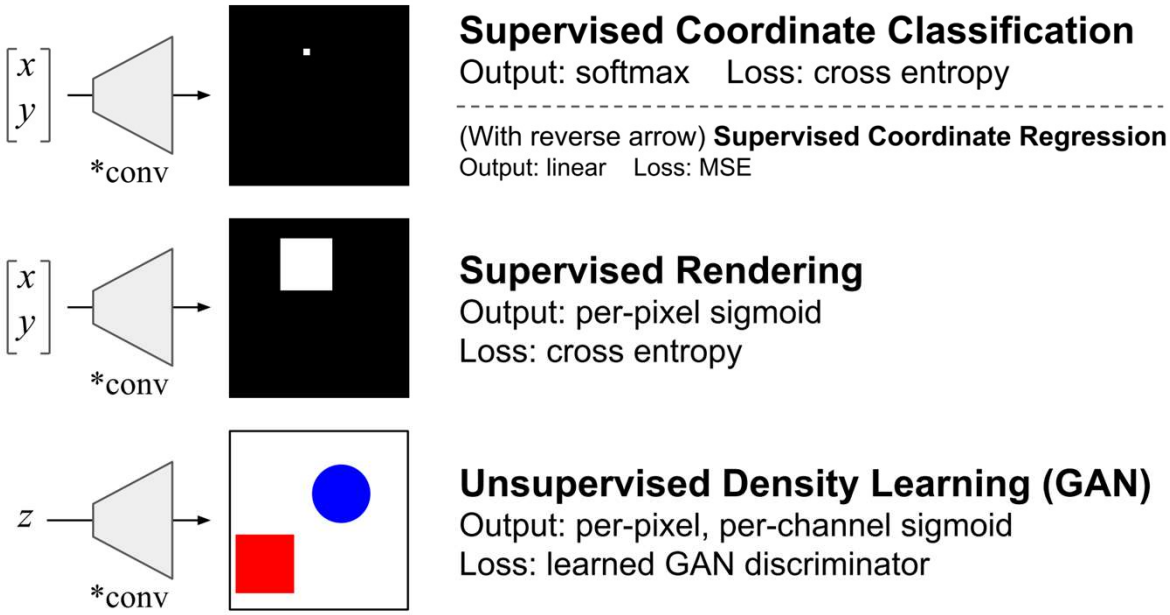


Figure 1: Toy tasks considered in this paper. The **conv* block represents a network comprised of one or more convolution, deconvolution (convolution transpose), or CoordConv layers. Experiments compare networks with no CoordConv layers to those with one or more.

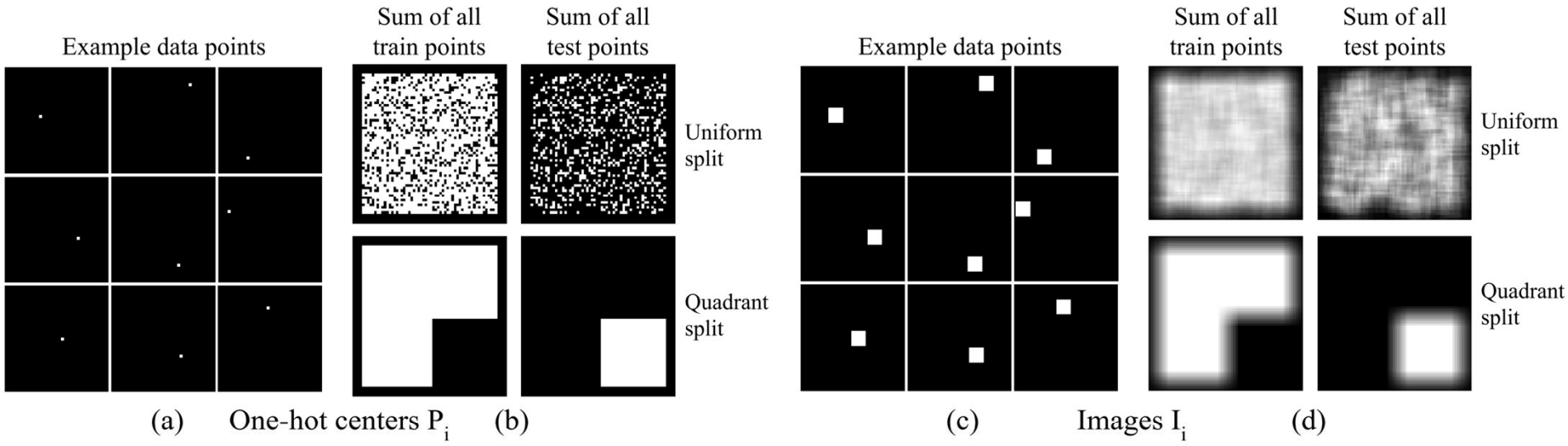
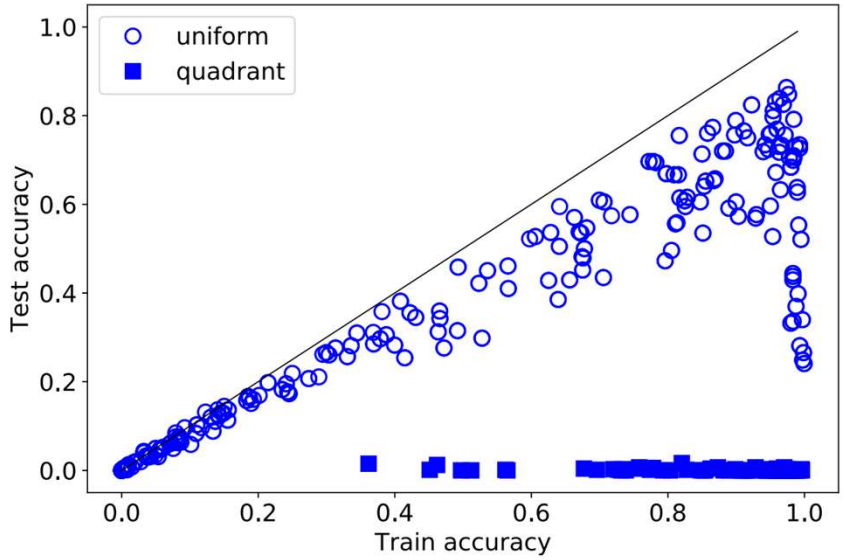
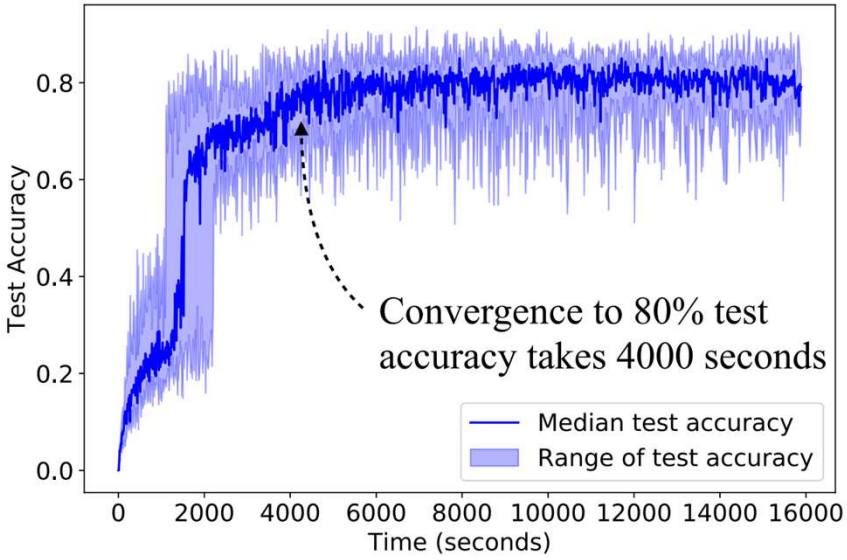


Figure 2: The Not-so-Clevr dataset. **(a)** Example one-hot center images P_i from the dataset. **(b)** The pixelwise sum of the entire train and test splits for uniform vs. quadrant splits. **(c)** and **(d)** Analagous depictions of the canvas images I_i from the dataset. Best viewed electronically with zoom.

Convolution





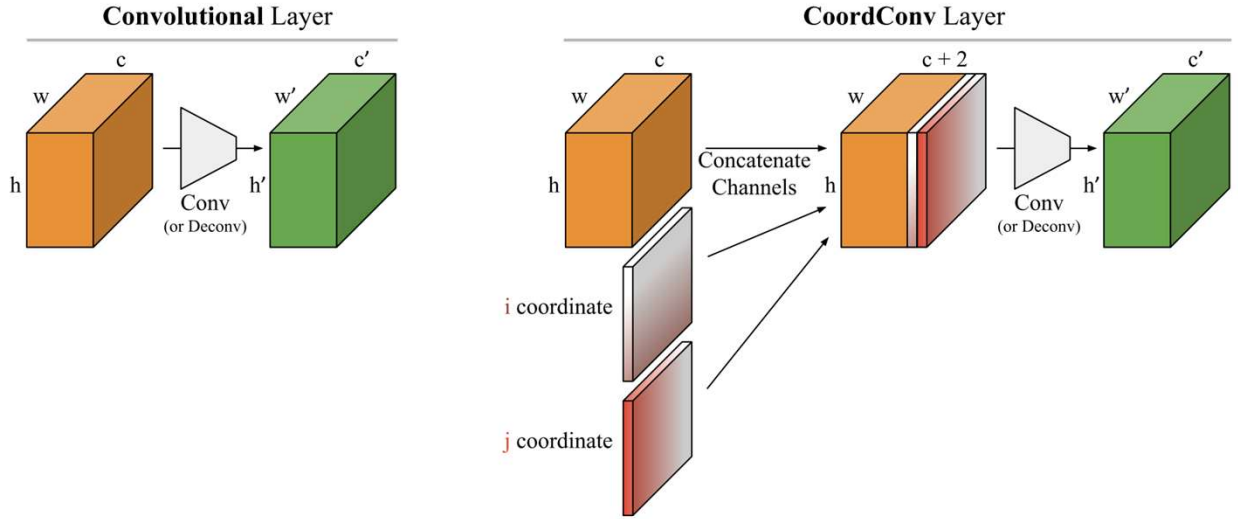


Figure 3: Comparison of 2D convolutional and CoordConv layers. **(left)** A standard convolutional layer maps from a representation block with shape $h \times w \times c$ to a new representation of shape $h' \times w' \times c'$. **(right)** A CoordConv layer has the same functional signature, but accomplishes the mapping by first concatenating extra channels to the incoming representation. These channels contain hard-coded coordinates, the most basic version of which is one channel for the i coordinate and one for the j coordinate, as shown above. Other derived coordinates may be input as well, like the radius coordinate used in ImageNet experiments (Section 5).

Concretely, the i coordinate channel is an h×w rank-1 matrix with its first row filled with 0's, its second row with 1's, its third with 2's, etc. The j coordinate channel is similar, but with columns filled in with constant values instead of rows.

Number of parameters. Ignoring bias parameters (which are not changed), a standard convolutional layer with square kernel size k and with c input channels and c' output channels will contain $cc'k^2$ weights, whereas the corresponding CoordConv layer will contain $(c+d)c'k^2$ weights, where d is the number of coordinate dimensions used (e.g. 2 or 3). The relative increase in parameters is small to moderate, depending on the original number of input channels. [2]

Translation invariance. CoordConv with weights connected to input coordinates set by initialization or learning to zero will be translation invariant and thus mathematically equivalent to ordinary convolution. If weights are nonzero, the function will contain some degree of translation dependence, the precise form of which will ideally depend on the task being solved. Similar to locally connected layers with unshared weights, CoordConv allows learned translation dependence, but by contrast

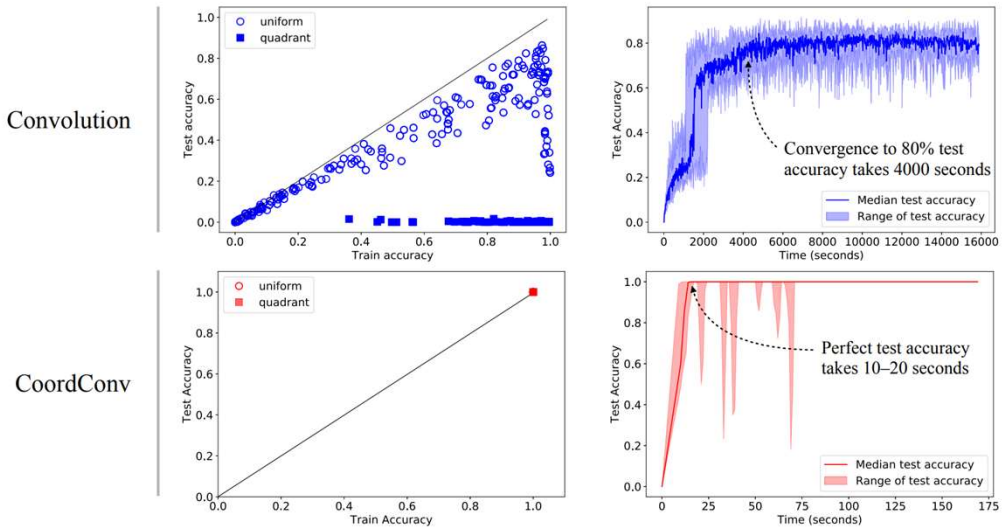
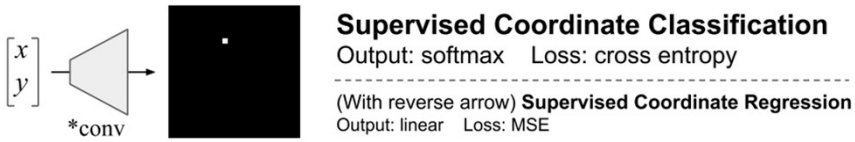


Figure 4: Performance of convolution and CoordConv on Supervised Coordinate Classification. **(left column)** Final test vs. train accuracy. On the easier uniform split, convolution never attains perfect test accuracy, though the largest models memorize the training set. On the quadrant split, generalization is almost zero. CoordConv attains perfect train and test accuracy on both splits. One of the main results of this paper is that the translation invariance in ordinary convolution does not lead to coordinate transform generalization even to neighboring pixels! **(right column)** Test accuracy vs. training time of the best uniform-split models from the left plot (any reaching final test accuracy ≥ 0.8). The convolution models never achieve more than about 86% accuracy, and training is slow: the fastest learning models still take over an hour to converge. CoordConv models learn several hundred times faster, attaining perfect accuracy in seconds.



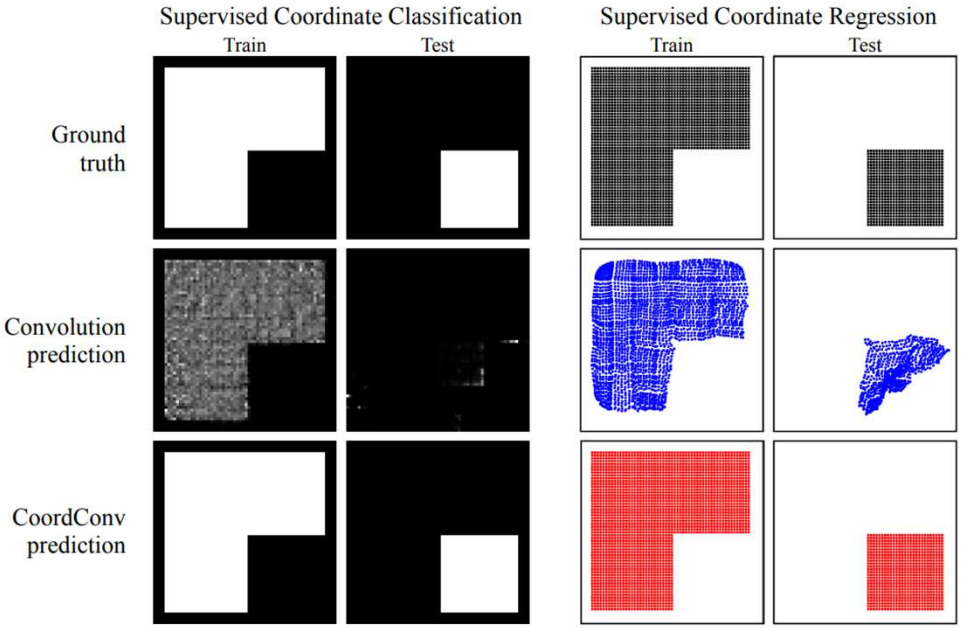


Figure 5: Comparison of convolutional and CoordConv models on the Supervised Coordinate Classification and Regression tasks, on a quadrant split. **(left column)** Results on the seemingly simple classification task where the network must highlight one pixel given its (x, y) coordinates as input. Images depict ground truth or predicted probabilities summed across the entire train or test set and then normalized to make use of the entire black to white image range. Thus, e.g., the top-left image shows the sum of all train set examples. The conv predictions on the train set cover it well, although the amount of noise in predictions hints at the difficulty with which this model eventually attained 99.6% train accuracy by memorization. The conv predictions on the test set are almost entirely incorrect, with two pixel locations capturing the bulk of the probability for all locations in the test set. By contrast, the CoordConv model attains 100% accuracy on both the train and test sets. Models used: conv–6 layers of deconv with strides 2; CoordConv–5 layers of 1×1 conv, first layer is CoordConv. Details in Section S2. **(right column)** The regression task poses the inverse problem: predict real-valued (x, y) coordinates from a one-hot pixel input. As before, the conv model memorizes poorly and largely fails to generalize, while the CoordConv model fits train and test set perfectly. Thus we observe the coordinate transform problem to be difficult in both directions. Models used: conv–9-layer fully-convolution with global pooling; CoordConv–5 layers of conv with global pooling, first layer is CoordConv. Details in Section S3.

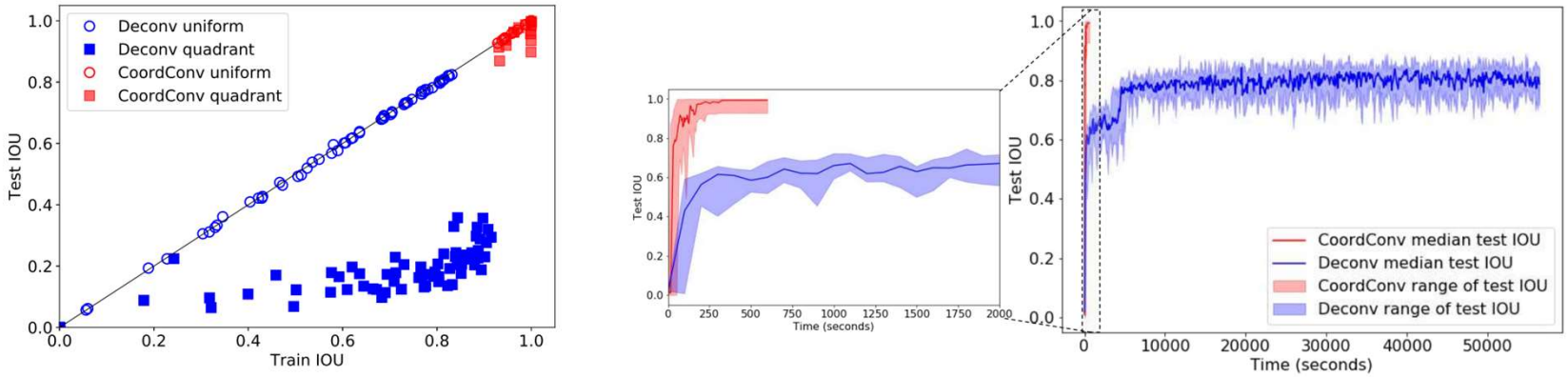
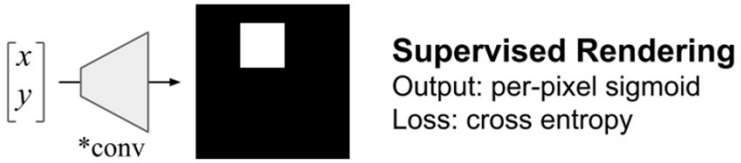


Figure 6: Results on the Supervised Rendering task. As with the Supervised Coordinate Classification and Regression tasks, we see the same vast separation in training time and generalization between convolution models and CoordConv models. **(left)** Test intersection over union (IOU) vs Train IOU. We show all attempted models on the uniform and quadrant splits, including some CoordConv models whose hyperparameter selections led to worse than perfect performance. **(right)** Test IOU vs. training time of the best uniform-split models from the left plot (any reaching final test IOU ≥ 0.8). Convolution models never achieve more than about IOU 0.83, and training is slow: the fastest learning models still take over two hours to converge vs. about a minute for CoordConv models.



ImageNet Classification As might be expected for tasks requiring straightforward translation *invariance*, CoordConv does not help significantly when tested with image classification. Adding a single extra 1×1 CoordConv layer with 8 output channels improves ResNet-50 [9] Top-5 accuracy by a meager 0.04% averaged over five runs for each treatment; however, this difference is not statistically significant. It is at least reassuring that CoordConv doesn't hurt the performance since it can always learn to ignore coordinates. This result was obtained using distributed training on 100 GPUs with Horovod [30]; see Section S5 in Supplementary Information for more details.

Object Detection In object detection, models look at pixel space and output bounding boxes in Cartesian space. This creates a natural coordinate transform problem which makes CoordConv seemingly a natural fit. On a simple problem of detecting MNIST digits scattered on a canvas, we found the test intersection-over-union (IOU) of a Faster R-CNN network improved by 24% when using CoordConv. See Section S6 in Supplementary Information for details.

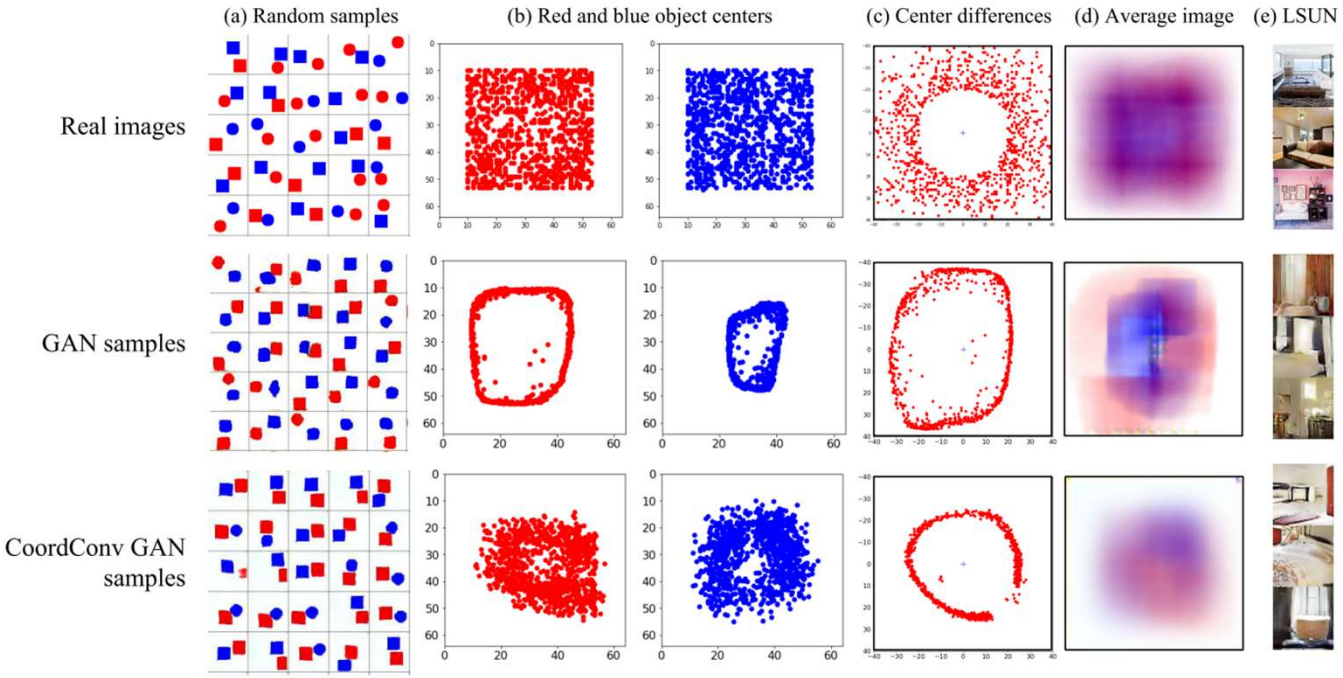
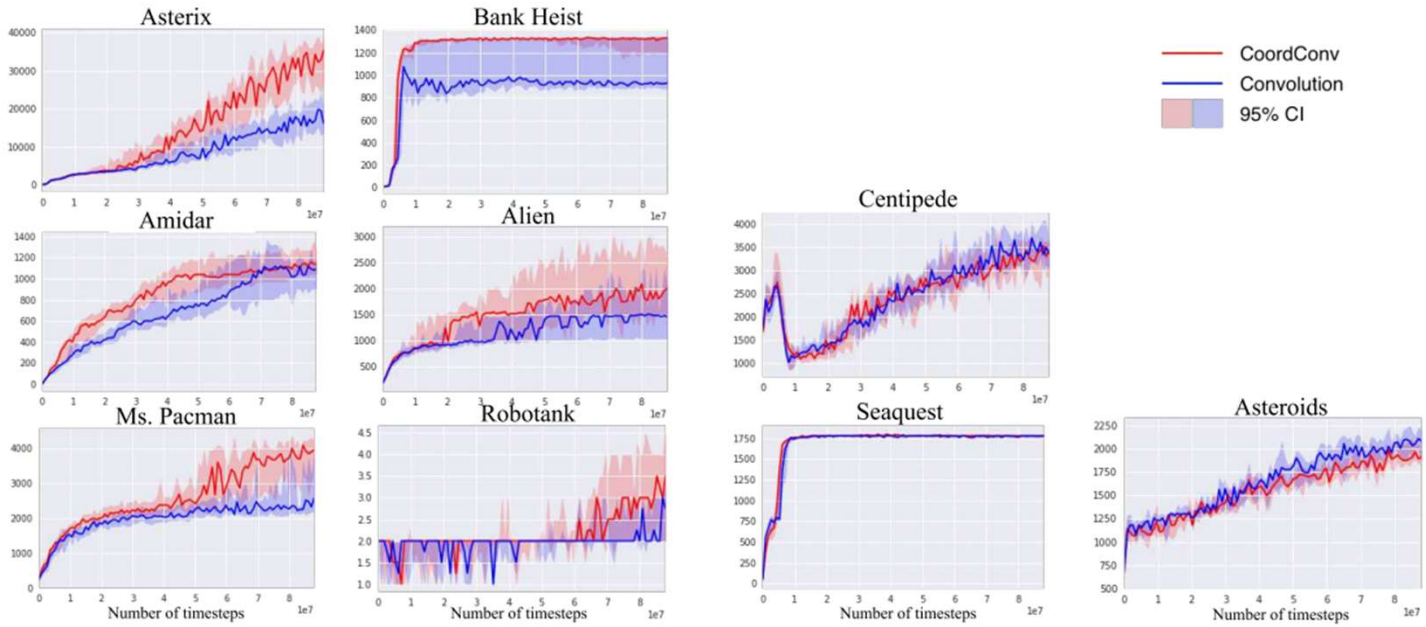


Figure 7: Real images and generated images by GAN and CoordConv GAN. Both models learn the basic concepts similarly well: two objects per image, one red and one blue, their size is fixed, and their positions can be random **(a)**. However, plotting the spread of object centers over 1000 samples, we see that CoordConv GAN samples cover the space significantly better (average entropy: *Data* red 4.0, blue 4.0, diff 3.5; *GAN* red 3.13, blue 2.69, diff 2.81; *CoordConv GAN* red 3.30, blue 2.93, diff 2.62), while GAN samples exhibit mode collapse on where objects can be **(b)**. In terms of relative locations between the two objects, both model exhibit a certain level of model collapse, CoordConv is worse **(c)**. The averaged image of CoordConv GAN is smoother and closer to that of data **(d)**. With LSUN, sampled images are shown **(e)**. All models used in generation are the best out of many runs.



(a) Better or faster (b) Similar (b) Worse

Figure 8: Results using A2C to train on Atari games. Out of 9 games, (a) in 6 CoordConv improves over convolution, (b) in 2 performs similarly, and (c) on 1 it is slightly worse.

Thanks