Pix2Seq

This paper advocates a new approach, based on the intuition that if a neural net knows about where and what the objects are, we just need to teach it to read them out. And by learning to “describe” objects the model can learn to ground the “language” on pixel observations, leading to useful object representations.

Given an image, our model produces a sequence of discrete tokens that correspond to object descriptions (e.g., object bounding boxes and class labels), reminiscent of an image captioning system.

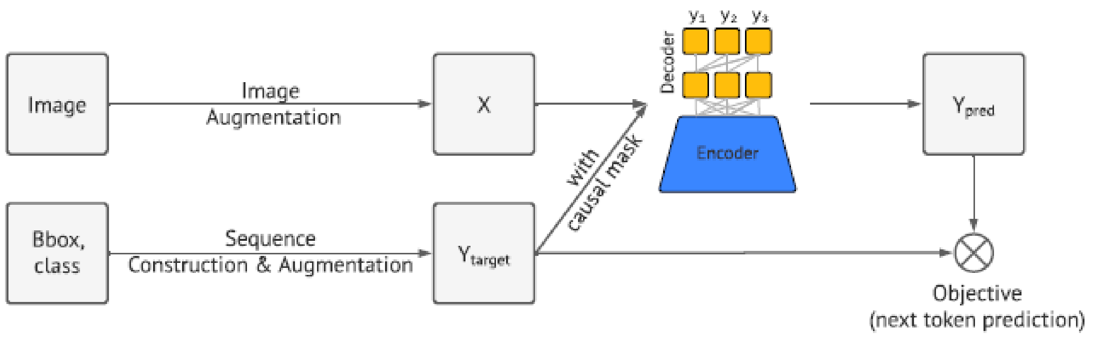
In essence, we cast object detection as a language modeling task conditioned on pixel inputs, for which the model architecture and loss function are generic and relatively simple, without being engineered specifically for the detection task.

For detection task with Pix2Seq:

- quantization and serialization scheme that converts bounding boxes and class labels into sequences of discrete tokens
- leverage an encoder-decoder architecture for perceiving pixel inputs and generating the target sequence

The objective function is simply the maximum likelihood of tokens conditioned on pixel inputs and the preceding tokens.

Pix2Seq framework



- Image Augmentation: As is common in training computer vision models, we use image augmentations to enrich a fixed set of training examples (e.g., with random scaling and crops).
- Sequence construction & augmentation: As object annotations for an image are usually represented as a set of bounding boxes and class labels, we convert them into a sequence of discrete tokens.
- Architecture: We use an encoder-decoder model, where the encoder perceives pixel inputs, and the decoder generates the target sequence (one token at a time).
- Objective/loss function: The model is trained to maximize the log likelihood of tokens conditioned on the image and the preceding tokens (with a softmax cross-entropy loss).

ARCHITECTURE, OBJECTIVE AND INFERENCE

Architecture:

Use an encoder-decoder architecture. The encoder can be a general image encoder that perceives pixels and encodes them into hidden representations, such as a ConvNet, Transformer, or their combination. For generation, a Transformer decoder will be used.

Objective:

Pix2Seq is trained to predict tokens, given an image and preceding tokens, with a maximum likelihood loss.

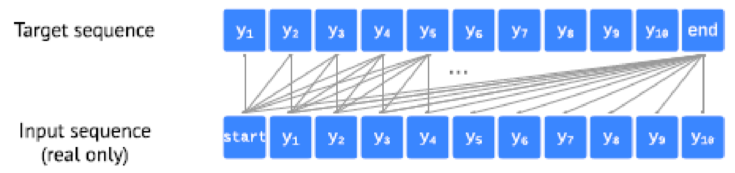
Inference:

At inference time, we sample tokens from model likelihood. This can be done by either taking the token with the largest likelihood (arg max sampling), or using other stochastic sampling techniques. We find that using nucleus sampling leads to higher recall than arg max sampling. The sequence ends when the EOS token is generated. Once the sequence is generated, it is straight-forward to extract and de-quantize the object descriptions.

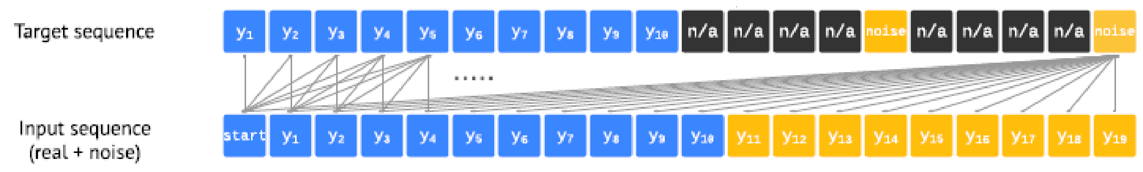
SEQUENCE AUGMENTATION TO INTEGRATE TASK PRIORS

The EOS token allows the model to decide when to terminate generation, but in practice, the model tends to finish without predicting all objects. This is likely due to 1) annotation noise, and 2) uncertainty in recognizing or localizing some objects.

To mitigate the problem we simply introduce a sequence augmentation technique, thereby incorporating prior knowledge about the task. The target sequence ~y in conventional autoregressive language modeling is the same as the input sequence y. And all tokens in a sequence are real. With sequence augmentation, we instead augment input sequences during training to include both real and synthetic noise tokens. We also modify target sequences so that the model can learn to identify the noise tokens rather than mimic them. This improves the robustness of the model against noisy and duplicated predictions.


Target sequence
y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 end
...
Input sequence (real only)
start y1 y2 y3 y4 y5 y6 y7 y8 y9 y10

(a) Conventional autoregressive langage modeling


Target sequence
y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 n/a n/a n/a n/a noise n/a n/a n/a n/a noise
.....
Input sequence (real + noise)
start y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 y11 y12 y13 y14 y15 y16 y17 y18 y19

Dataset: MS-COCO 2017 detection dataset

Environment: Jupyter Notebook

Results:

Training from scratch:

For training from scratch, we follow using a ResNet backbone, followed by 6 layers of transformer encoder and 6 layers of (causal) transformer decoder. We resize images (with a fixed aspect ratio) so the longer side is 1333 pixels. For sequence construction, we use 2000 quantization bins, and we randomize the order of objects every time an image is shown. We append noise objects to real objects such that each image contains 100 objects in total, and hence a sequence length of 500. The model is trained for 300 epochs with a batch size of 128.

We mainly compare with two widely recognized baselines: DETR and Faster R-CNN.

DETR and our model have comparable architectures, but our Transformer decoder does not require learned "object queries" or separated heads for box regression and classification, since our model generates different types of tokens (e.g., coordinate and class tokens) with a single softmax.

Faster R CNN is a well established method, with optimized architectures such as feature-pyramid networks (FPN). Faster R CNN is typically trained in fewer epochs than DETR or our model, likely because it explicitly incorporates prior knowledge of the task in the architecture itself. Thus we also include an improved Faster R CNN baseline, denoted as Faster R-CNN+, from, where Faster R-CNN models are trained with the GIoU loss, train-time random crop augmentations, and the long 9x training schedule.

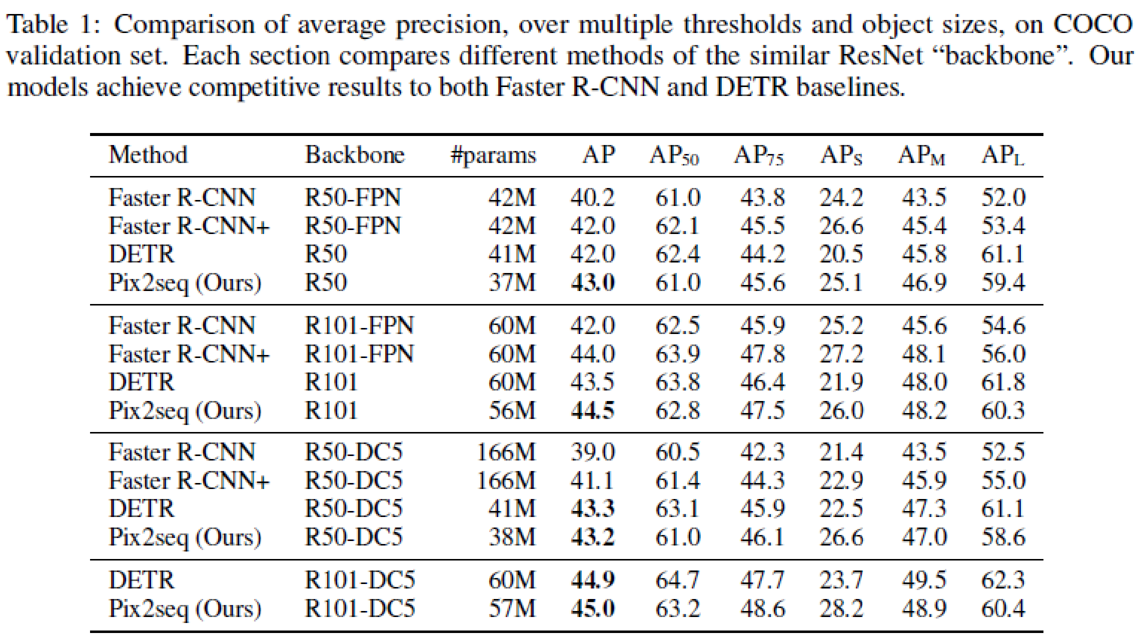


Table 1: Comparison of average precision, over multiple thresholds and object sizes, on COCO validation set. Each section compares different methods of the similar ResNet “backbone”. Our models achieve competitive results to both Faster R-CNN and DETR baselines.

Method	Backbone	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster R-CNN	R50-FPN	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster R-CNN+	R50-FPN	42M	42.0	62.1	45.5	26.6	45.4	53.4
DETR	R50	41M	42.0	62.4	44.2	20.5	45.8	61.1
Pix2seq (Ours)	R50	37M	**43.0**	61.0	45.6	25.1	46.9	59.4
Faster R-CNN	R101-FPN	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster R-CNN+	R101-FPN	60M	44.0	63.9	47.8	27.2	48.1	56.0
DETR	R101	60M	43.5	63.8	46.4	21.9	48.0	61.8
Pix2seq (Ours)	R101	56M	**44.5**	62.8	47.5	26.0	48.2	60.3
Faster R-CNN	R50-DC5	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster R-CNN+	R50-DC5	166M	41.1	61.4	44.3	22.9	45.9	55.0
DETR	R50-DC5	41M	**43.3**	63.1	45.9	22.5	47.3	61.1
Pix2seq (Ours)	R50-DC5	38M	**43.2**	61.0	46.1	26.6	47.0	58.6
DETR	R101-DC5	60M	**44.9**	64.7	47.7	23.7	49.5	62.3
Pix2seq (Ours)	R101-DC5	57M	**45.0**	63.2	48.6	28.2	48.9	60.4

Pretraining:

For pretraining on Objects365 dataset, we use similar settings as above with a few differences. Notably, instead of using the large 1333 1333 image size, we use a smaller image size of 640 640, and pretrain the models for 400K steps with batch size of 256. It is worth noting that this pretraining process is even faster than training from scratch due to the use of smaller image size. During the finetuning on COCO dataset, only a small number of epochs (e.g., 20 to 60 epochs) are needed to achieve good results. And we could use larger image size during fine-tuning as well.

the performances of Objects365 pretrained Pix2Seq models are strong across various model sizes and image sizes. The best performance (with 1333 image size) is 50 AP which is 5% higher than the best model trained from scratch, and the performance holds up very well even with 640 image size. Notably, with a smaller image size used for pretraining, the pretrain+finetune process is faster than training from scratch, and also generalizes better. Both factors are crucial for training larger and better models.

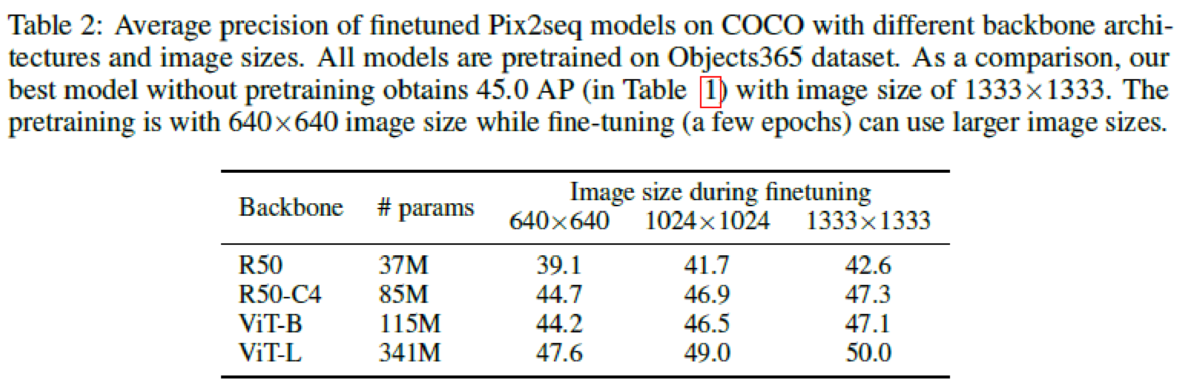


Table 2: Average precision of finetuned Pix2seq models on COCO with different backbone architectures and image sizes. All models are pretrained on Objects365 dataset. As a comparison, our best model without pretraining obtains 45.0 AP (in Table 1) with image size of 1333×1333. The pretraining is with 640×640 image size while fine-tuning (a few epochs) can use larger image sizes.

Backbone	# params	Image size during finetuning		
		640×640	1024×1024	1333×1333
R50	37M	39.1	41.7	42.6
R50-C4	85M	44.7	46.9	47.3
ViT-B	115M	44.2	46.5	47.1
ViT-L	341M	47.6	49.0	50.0

Github links:

Tensorflow: https://github.com/google-research/pix2seq

Pytorch: https://github.com/gaopengcuhk/Stable-Pix2Seq

Paperswithcode link:

https://paperswithcode.com/paper/pix2seq-a-language-modeling-framework-for