Intelligent Visual Computing [Spring 2023]

Assignment 5: Autoregressive image completion

Overview

In this assignment you will learn to implement a basic autoregressive model for completing an image. The model is based on the PixelCNN described in the Pixel Recurrent Neural Networks, van den Oord et al. paper. The basic idea of the method is to synthesize pixels one-by-one starting from the top left corner in an image region and proceeding in a row major order.

This assignment counts for **20 points** towards your final grade if you are taking the 574 section. **If you are taking the 674 section, multiply your total points by 0.75** (i.e., the assignment counts for 15 points).

Instructions

Download the starter_code_dataset.zip. The zip file contains (i) the starter code, (ii) a training dataset of 2K 100x100 chair images, (iii) a small validation split of 96 images, (iv) a test split of 40 images. Unzip the package and start working on the "*model.py*", "*trainARImage.py*", and "*testARImage.py*" scripts based on the specifications described below.

The convnet you will implement is small, the dataset is small, and can be trained on the CPU (i.e., training takes ~10-15 minutes, and testing takes a few seconds per image in the instuctor's 5-year old laptop with a i7-5950HQ). If you want to use a GPU (this is not required by this assignment), you need to have access to a GPU graphics card with at least 2GB memory.

Since the dataset is very small, different executions of training may lead to different local minima with somewhat different results. You may repeat the training a few times to select the best model (i.e., the one that gives the best visual results at test time). A few examples of desired results are shown below. The odd columns show test images corrupted with salt-and-pepper noise in a small window near the center of the image. The even columns show expected restorations (the average test reconstruction error in terms of L1 error should be below 0.1). Ideally, one would want to use much larger training datasets for training image synthesis/completion methods, and also use more modern and complex autoregressive variants (e.g., see VQGAN, ASSET, ImageBART etc) based on transformers or diffusion models, which are all beyond the scope of this assignment.

What You Need To Do

Task A [30%]: Change the starter code in "*model.py*" so that you define the following convnet with three blocks, each consisting of:

1) a masked convolution layer with 16 filters applying 3x3 filters on the input image. Set stride to 1, dilation to 3 to enlarge the receptive field of each filter, padding to 'reflect'. The masked convolution zero-es out weights for the bottom right part of the pixel, as described in the class and the above paper.
2) a batch normalization layer
3) a leaky ReLU with 0.001 negative slope

The three blocks share the same structure as above (still with their own filter weights at each convolution layer). The masked convolution in the first layer should use a mask which zero-es out the weight at the center of the filters. The rest of the masked convolution layers should not mask the center. Use the MaskedCNN class appropriately while defining your model in PixelCNN.

After the above 3 blocks, a 1x1 convolution layer processes the 16 channels from the last block, and outputs a single channel (i.e., the grayscale intensity for the output image). Set the padding size in the convolution layers such that the resulting channel has the same height and width with the input image. The channel intensities should then be squeezed between 0 and 1 through a sigmoid function. Think about whether you should use the bias in each of the convolutional layers.

For all layer parameters, use the default PyTorch initialization scheme (no need to write any particular code for initializing the model parameters).

Task B [30%]: Change the starter code in "*trainARimage.py*" to train your network such that it predicts an output image that matches the input image. Training should be done using the L1 loss function (already implemented as nn.L1Loss in PyTorch). Complete the code such that (a) each batch is loaded correctly along with the desired targets, (b) backpropagation is executed to update the model parameters, (c) the L1 loss is computed for each batch, (d) the training and validation loss are computed for each epoch. The starter code will plot the training and validation losses for each epoch.

Task C [30%]: Change the starter code in "*testARimage.py*" to correct each pixel in the distorted region in an autoregressive manner. This involves implementing a loop that predicts each pixel starting at the top left of the distorted region and proceeding in a row major order. You should not change pixels outside the distorted region. The starter code already computes the L1 error for each test image.

Task C [10%]: You should submit a zip file that contains (a) your code, (b) a README.txt reporting the average reconstruction error you achieved for the best model you trained, and (c) your 40 output test images in a folder called 'results'. Do not include the dataset. Do not include the trained model.

Submission:

Please follow the **Submission** instructions in the course policy to upload your zip file to Gradescope. **Please do not include the dataset and models in your submission!**

starter_code_dataset.zip May 4 2023, 4:56 PM

Submission status

Submission status	This assignment does not require you to submit anything online
Grading status	Not graded
Due date	Friday, May 19, 2023, 11:59 PM
Time remaining	2 days 2 hours
Last modified	-
Submission comments	Comments (0)

◄ Thu Class Meeting

Jump to...

Bonus assignment: Reaction Report ►

Get Help
Help for Students
Help for Instructors