Project Proposal - ECE 176

Brett Kinsella
Electrical and Computer Engineering
A18033152

Rohan Gujral
Computer Science and Engineering
A18498935

Abstract

Image inpainting has long been a problem that researchers have tried to solve through a variety of methods over the years. It is the task of filling missing or corrupted pieces of an image with a computed version of that missing piece. In a paper entitled *Context Encoders: Feature Learning by Inpainting*[8] a new methodology for image inpainting was proposed through a modified convolutional neural network architecture the authors call *Context Encoders* [8]. The main idea is to establish context around the missing section of an image by analyzing the surrounding components to generate the appropriate content for that missing region. In this project, we hope to reimplement the neural network architecture described in this paper using alternative training datasets while achieving similar results.

1 Problem Definition

Now a staple in many modern smartphones, image inpainting is synonymous with editing and cropping out unsightly or undesirable aspects of an image. Drawn in by its abilities, we decided to implement an elementary version of an image inpainting neural network. At a higher level, we are training a neural network to guess what should be filled in the missing piece using context, or surrounding pixels in this case. The solution mentioned in the paper describes a method that attempts to balance realism and pixel accuracy in the output.

2 Tentative Method

The convolutional neural network, CNN, described in the paper uses an encoder-decoder-style architecture with a channel-wise fully connected layer to connect the two and utilizes two separate loss functions in tandem to optimize the network. The first of the loss functions is a reconstruction loss that consists of a normalized masked L2 distance, while the other is an adversarial loss derived from a generative adversarial network. The original network was implemented using the Caffe [3] framework and Torch, but we intend to reimplement the network using a more modern framework such as PyTorch [7]. PyTorch offers significant advantages over Caffe as PyTorch allows the model architecture to be developed in Python instead of PROTOTXT files [3] and offers easy-to-use modern utilities for experimentation. Furthermore, while the original paper used the stochastic gradient descent, SDG, solver, ADAM [4], for optimization purposes, we intend to use a more modern approach to SGD such as RADAM [5] or combine ADAMW [6] with a learning rate scheduler such as the 1cyle policy [9] to improve performance. In brief, we plan to reimplement the exact neural network architecture described in Pathak et al. [8], but utilize modern tools for ease of implementation and to enhance performance where possible.

3 Experiments

For the method outlined above, a relevant dataset composed of 12000 images from the online artificial intelligence and machine learning community Kaggle [2] will be used to train the first iteration of our model. The data set [1] is composed of 12,000 images that are divided into five distinct categories of landscapes: coast, desert, forest, glaciers, and mountains. For each image, a duplicate will be created for which a certain amount of pixels will be blocked out, image masking, and each pair will be fed as training data to the network. The dataset is already split into training, validation, and testing groups. We also intend to experiment with various sizes and locations of image masking, to see what effect, if any, it has on the final weights and biases. Not all of the images are the same size, so we would have to manually resize the images to fixed dimensions before training and testing.

References

[1] DeepNets. *Landscape Recognition | Image Dataset | 12k Images*. URL: https://www.kaggle.com/datasets/utkarshsaxenadn/landscape-recognition-image-dataset-12k-images/data.

[2] Google. *Kaggle: Your Machine Learning and Data Science Community*. URL: https://www.kaggle.com/.

[3] Yangqing Jia et al. *Caffe: Convolutional Architecture for Fast Feature Embedding*. 2014. arXiv: 1408.5093 [cs.CV]. URL: https://arxiv.org/abs/1408.5093.

[4] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[5] Liyuan Liu et al. *On the Variance of the Adaptive Learning Rate and Beyond*. 2021. arXiv: 1908.03265 [cs.LG]. URL: https://arxiv.org/abs/1908.03265.

[6] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG]. URL: https://arxiv.org/abs/1711.05101.

[7] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG]. URL: https://arxiv.org/abs/1912.01703.

[8] Deepak Pathak et al. *Context Encoders: Feature Learning by Inpainting*. 2016. arXiv: 1604.07379 [cs.CV].

[9] Leslie N. Smith and Nicholay Topin. *Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates*. 2018. arXiv: 1708.07120 [cs.LG]. URL: https://arxiv.org/abs/1708.07120.