Project Report - ECE 176

Alden Yue
ECE
A15902882

Abstract

For this project, I will be re-implementing the work done by Pathak et al. for Context Inpainting missing portions of an image. I will reimplement their Context Encoder, a convolution neural network trained to generate content on missing portions of the image based on the surrounding context of the rest of the image. In the paper, they use unsupervised learning to train their CNN on the ImageNet and Paris Google StreetView datasets. However, for my re-implementation, I will be using the Places365 dataset, which is a scene recognition dataset (Places).

1 Introduction

Even if part of the information is missing or distorted, human beings can use the underlying structure of the rest of the information to predict the rest of the information. For example, while someone is speaking to you, part of their speech may be inaudible due to a loud noise because you're having a conversation on a noisy speech. Even though you're missing information, you can infer what they meant to say using the context of the rest of their speech. The same sentiment is applied to missing portion of images. If we were to take an arbitrary portion out of an image, humans can use the underlying context of the rest of the image to infer the missing portions. Using CNNs, we want to learn how to predict the structure of the missing portions.

This brings us to our motivation. Given an image with the center portion missing, people are able to make a mental representation of what should be within the missing portion. In Figure 1 in the second to last row for example. there is a picture of a red brick wall that is missing the center portion. Even without knowing the true image, a person should be able infer from the top and bottom of the door frame that there should be a door entrance in the missing portion. If we gave this to a an artist with the same objective, they should be able to paint in the missing information that looks similar to the original. If a human being can use context to produce a result, then a CNN should be able to do the same as well.

Figure 1: Example images from Places365 compared to the same images with a missing center block

Pathak's Context Encoder is a model that consists of an encoder and decoder. The encoder captures the context of the image and compresses it into a lower dimensional feature representation. The Decoder then uses said feature representation to create a hypothesis for the missing portion of the image. This encoder-decoder structure is similar to that of autoencoders, where an image is compressed into a lower-dimensional feature space 'bottleneck' before reconstructing the image. Both are also trained using unsupervised learning. However, they differ in that Pathak's Context Encoder has a more difficult task as it requires a higher semantic understanding of the image. The context encoder does not have access to nearby pixels due to large block of missing information. Therefore, the context encoder needs to understand the high-level features to paint in the large missing portion.

The Context Encoder needs to both understand the context as well as provide a realistic guess at what the missing portion is. For our Context Encoder, we will use L2 loss between the original portion and the generated portion. This is a good start, but in order to generate more realistic results, we also want to include adversarial loss. We train an adversarial discriminator in tandem with our Context Encoder to classify each infill square as either real or fake. Then we include an adversarial loss term to our Context Encoder that rewards the network the better it can fool the adversarial network. This extra adversarial loss will help create image patches that are more clear and specific rather than blurry.

2 Related Works

The success of image classification and generation has paved the way for and explore solutions to more difficult problems such as image inpainting. For the Context Inpainter specifically, our adversarial loss is computing using a Generatively Adversarial Network (GAN) which we use help our context encoder create increasingly realistic images.

2.1 Image Generation

For the Context Inpainter Specifically, our adversarial loss is computed using a Generatively Adversarial Network (GAN) which we use help our context encoder create more realistic images. Previous work on GANs such as Dosovitskiy et al. and Rifai et al use GANs to generate images of chairs and faces (Dosovitskiy, Rifai). While these works rely on large and labeled datasets for these categories, the Context Encoder can be trained in an unsupervised manner

2.2 Texture Synthesis and Infilling

Due to large size of the center block (one fourth of the image area), texture synthesis would not work well in this case. Methods that don't utilize the context of the image would not work well. Filling larger holes are usually done through scene completion, which finds the nearest neighbor match for the hole from a data set of millions of images. However, scene completion is done for whole objects and would struggle for an arbitrary removal. The scene completer would perform poorly on multiple objects that are partially removed. Moreover, the k-nearest neighbors metric of the scene completer performs rather poorly in comparison to the l2 distance metric our context encoder uses.

3 Context Encoder

3.1 Pipeline

At a high-level, the context encoder itself is composed of 2 parts: the encoder and the decoder. The encoder takes an image with the center portion removed and compresses the image into a lower dimensional feature representation or a "bottleneck." The encoder then receives the features from the bottleneck and up-samples them to generate the and estimate of the missing content. To compute the loss, we compute the reconstruction loss as the mean squared error (MSE) between the real and the generate image.

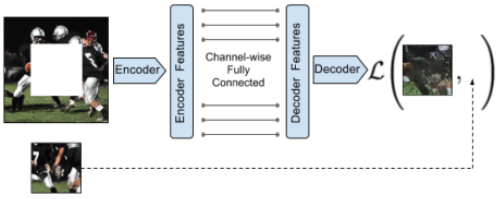
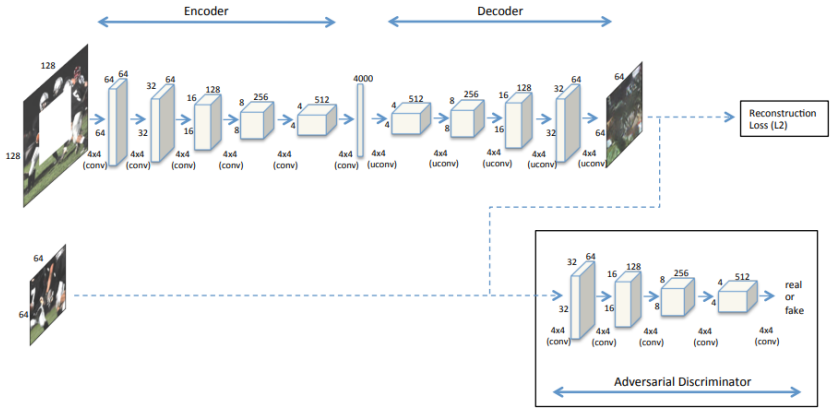


Figure 2: High-level design of the context encoder

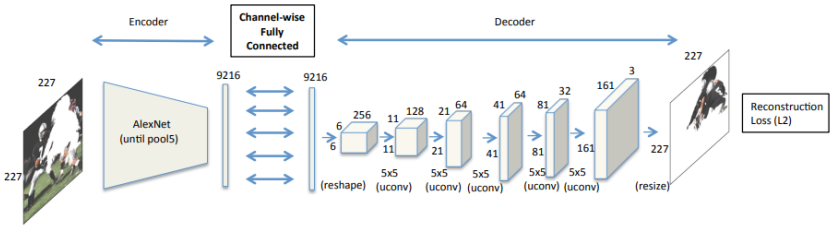
3.2 Architectures

Pathak describes two different architectures that they used in their paper. The first architecture they used took in a 227 by 227 pixel image with the encoder using AlexNet architecture until pool5. The decoder consists of 5 up-sampling layers using transpose convolution to produce an image of the

same size. The reconstruction loss is then computed only on the masked out portion. In contrast, the second architecture takes in a 128 by 128 image with the encoder consisting of 5 convolutional layers. The input of the encoder has the center portion taken out and the output is the same size of the center portion. The reconstruction loss is computed as the MSE between the real and generated image. The adversarial loss increases based on how many times we fail to fool the adversarial discriminator. Simultaneously, the real and generated image patches are used to train the adversarial discriminator. In the paper, they failed to make the AlexNet architecture converge with adversarial loss and found that the Convolution encoder generate finer results (Pathak). Due to the limitation of my hardware and the time frame, I will just be reimpelmented the convolution encoder with adversarial discriminator architecture.



(a) Context encoder trained with joint reconstruction and adversarial loss for semantic inpainting. This illustration is shown for *center region dropout*. Similar architecture holds for arbitrary region dropout as well. See Section 3.2.



(b) Context encoder trained with reconstruction loss for feature learning by filling in *arbitrary region dropouts* in the input.

Figure 3: Pool-free design with adversarial Discriminator and AlexNet encoder architectures

3.3 Loss Function

$$L_{rec}(x) = ||\hat{M} \odot (x - F((1 - \hat{M}) \odot x)||_2^2 \tag{1}$$

$$L_{adv}(x) = \max_D \mathbb{E}_{x \in \chi}[\log(D(x)) + \log(1 - D(F((1 - \hat{M}) \odot x))] \tag{2}$$

$$L_{dis}(x) = \max_D \mathbb{E}_{x \in \chi}[\log(D(x))] + \mathbb{E}_{z \in Z}[\log(1 - D(G(z))] \tag{3}$$

$$L_{total} = \lambda_{rec} L_{rec} + \lambda_{adv} L_{adv} \tag{4}$$

The context encoder has two parts to its loss function: reconstruction and adversarial loss. Reconstruction loss (1) is the L2 loss between the true and generated patch. We define $\hat{M}$ is the center mask where it is 1 in the center 64 by 64 pixel and is 0 everywhere else, $\odot$ as an element-wise multiplication operation, F as the function representing our context encoder, and x as the images. L2 loss captures the overall structure and contributes to coherence, but tends to average the prediction, resulting in blurry images.

On the contrary, adversarial loss is used to make realistic images and is defined as the binary cross entropy error between the real/fake label and the guess of our adversarial discriminator. We define D as the decision of our adversarial discriminator. We train our adversarial discriminator simultaneously using equation (3) and use equation (2) for our loss function for our context encoder. With equation (2), we punish the context encoder for each patch it isn't able to fool the AD with. Finally, our total loss function for our context encoder (4) is a linear combination of (1) and (2). By using both reconstruction and adversarial loss, we gain the benefit of reconstruction loss (structure and coherence) while mitigating its down sides with adversarial loss (sharpness and realness).

3.4 Implementation Details

To make and train Pathak's context encoder, I re-implemented the architecture and wrote the training function in PyTorch. As in the paper, each block in the architecture consists of a convolutional layer, batch normalization, and ReLU non-linear activation function. Each encoder and discriminator block uses a 4 by 4 convolution with stride 2 and padding 1 to compress the feature dimensions by a factor of 2 and a leaky ReLU function. Each decoder block uses a transpose convolution of size 4, stride 2, and padding 1 and a ReLU function. The size of the bottleneck is set to 4000.

As for training the context encoder, I set $\lambda_{rec} = 0.999, \lambda_{adv} = 0.001$ as in the paper. For both encoder and discriminators, I used Adam optimizers with parameters $lr = 2e-4, \beta_1 = 0.5, \beta_2 = 0.999$. Each image from the dataset is then transformed to size 128 by 128 so that it can work for our designed architecture and center mask. As in the paper, we randomly initialize the weights for both encoder and discriminator. The context encoder and adversarial discriminator are trained with the loss functions as described above with small change. For reconstruction loss, we use a weighted L2 loss, weighing the pixels on the edges 10 times more than the others. In the paper they use 7 pixels from the edge for 227 by 227 pixel images, so for our image size we use 4 overlapping pixels.

As for our dataset, we will use a train-test split of 90 percent training and 10 percent testing data. We will train for a few epochs on the training before visually evaluating our patches on the testing set to see how well the context encoder generalizes to new data. In the paper, it took 14 hours to converge in 100k iterations in 110 epochs with the AlexNet architecture with their Nvidia Titan X GPU. However, since I am limited computationally, I ran the training on my laptop GPU overnight for only 4 epochs (each epoch taking roughly 2.5 hours to run) so the inpainting results may not be as clear as shown in the paper.

4 Experiments

4.1 Dataset

In the original paper, they train the context encoder on ImageNet and Paris Google Streetview pictures, giving the context encoder millions of data to choose from with large variety of scenes with semantic detail. For this project, I decided to train the context encoder on the Places365 data set. Places365 is similar to ImageNet in that contains millions of images for image recognition tasks. For Places365, it is primarily used for scene recognition tasks. By choosing similar set to the paper, I hope to be able to learn semantic details just as well as with ImageNet.

4.2 Evaluation

To evaluate the context encoder's performance, I will examine the side-by-side comparison of the original image versus the filled in image output by the context encoder. After training the model on the training set for 4 epochs, I obtained 32 examples shown below in Figure 4.

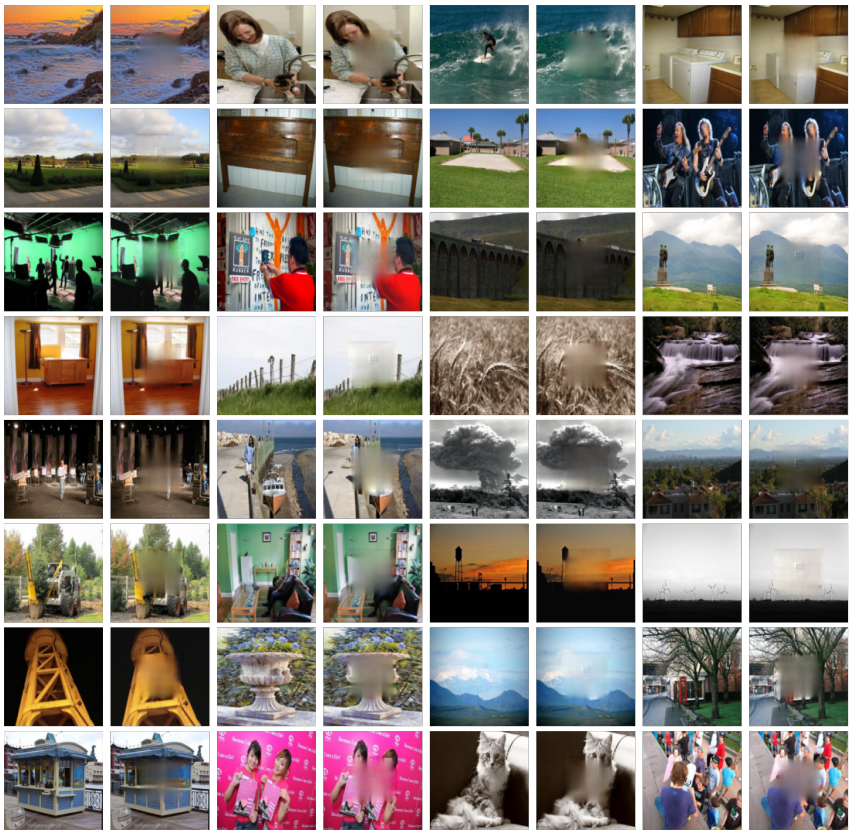


Figure 4: Infilled Places365 images after training for 4 epochs

Looking at each of the comparisons, the context encoder seems to have a good grasp of the colors of the context. From a distance, it would hard tell if something was off. It is also able to capture most of the obvious details of the context. For instance, for images containing the line between the sky and land, it is able to capture that semantic detail quite well and be able to fill in a very convincing patch

with colors and contours that match with the scene. However, for images with finer details, such as images with people, animals, or more intricate buildings and structures, the encoder does not seem to understand and ends up making a very blurry patch.

Each of these filled in images are quite blurry and are not as sharp as the images trained with l2 and adversarial loss in the paper. This is most likely due to their context encoder being trained on for 110 epochs in 14 hours compared to my 4 epochs in 10. While the context encoder has a good grasp of the general context, it probably takes a bit longer for the adversarial loss to start creating more realistic patches. As it stands right now, I see this as a successful implementation. Unfortunately, I do not have the hardware and time to train the context encoder, but with the resources I have, I am satisfied with the results.

5 Conclusion

In conclusion, the current direction of this project in re-implementing the context encoder seems promising. The context encoder is able to grasp the basic colors and semantics of the surrounding context. Although a bit blurry, the patches makes sense given the context. While it understands the basic features of the context, such as square geometries like the windows, boxes, and the horizon. Given more time and resources, I would also implement random crop masks as well to help prevent the context encoder from overfitting to the center block. I would also train the context encoder for more epochs to confirm the progress of the images. Overall, I see this project as a success.

References

Pathak, Deepak, et al. 'Context Encoders: Feature Learning by Inpainting'. CVPR, 2016.

Y. Bengio. Learning deep architectures for ai. Foundations and trends in Machine Learning, 2009.

A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. CVPR, 2015.

S. Rifai, Y. Bengio, A. Courville, P. Vincent, and M. Mirza. Disentangling factors of variation for facial expression recognition. In ECCV, 2012.

Places: A 10 million Image Database for Scene Recognition B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017