CAP4613 Final Project
Image Inpainting

Jiayu Huang, Heng Sun

Abstract

Like any other technology based on machine learning and deep learning, image inpainting has become an important and convenient tool in daily life use especially in recovering pictures. There are many pre-made models in the market that we can use to eliminate unexpected spots or missing parts and recover them to a normal picture. However, due to our group's experiences, they are not always performing well under certain circumstances. Therefore, our group decides to build our own model to predict and recover the flawed pictures. This report will discuss how we make an image inpainting application from the beginning to the end.

Introduction

When working with the data transfer of images or the detailing of photographs in general, users may find their images altered and destroyed as a result of loss of pixel information. Our project seeks to fill in missing pixels of an image similar to how image editing software utilizes a "content-aware fill" feature.

Our group aims to take this to the next level by providing more accurate pixel restoration using GAN which modern-day software does not use. In this specific case, our dataset focuses on images with overlying blocks covering certain aspects. In this project, we aim to remove these blocks revealing a generated image that should match the natural image itself.

For our project, we input a set of images missing pixel information. After running through our software, each image will be digitally reconstructed in its entirety, having new images that accurately match their originals before pixel information was removed.

This is accomplished through Generative Adversarial Networks (GAN's). Generative modeling is an unsupervised learning task in machine learning. It involves automatically discovering and learning the regularities or patterns in input data. Generative modeling can be used to generate or output new examples that plausibly could have been drawn from the original dataset. GANs are an exciting and rapidly changing field, delivering on the promise of generative models.

Previous Work (Background)

Context Encoders: Feature Learning by Inpainting

Deepak Pathak Philipp Krähenbühl Jeff Donahue Trevor Darrell Alexei A. Efros
University of California, Berkeley
{pathak,philkr,jdonahue,trevor,efros}@cs.berkeley.edu

Before we start our project, we find many related materials including others' essays, projects and real projects. Among them, an article called *Context Encoders: Feature learning by inpainting* written by Deepak Pathak is similar to our project's topic. Their main goal is using an algorithm to eliminate the missing parts in an image, which is exactly what we want to do in the project. They use an unsupervised visual feature learning algorithm driven by context based pixel prediction. They have a clear and explicit plan to achieve the expected goal. The first thing they do in the algorithm is propose a context encoder to a convolutional neural network. This will generate contexts of the missing parts in an image conditioned on its surrounding environments and colors. In order to get a better result, their group uses two

different loss functions: standard pixel wise reconstruction loss function and adversarial loss function.

(a) Input context (b) Human artist

(c) Context Encoder ($L2$ loss) (d) Context Encoder ($L2$ + Adversarial loss)

At the end of their paper, they discovered that using adversarial loss function has a better result than using standard pixel wise reconstruction loss function because adversarial loss function is able to handle multiple modes in the output and produce a much sharper result. This paper gives us a hint on which loss function we should use and it helps us to reduce the time it would take to test different loss functions.

Overview

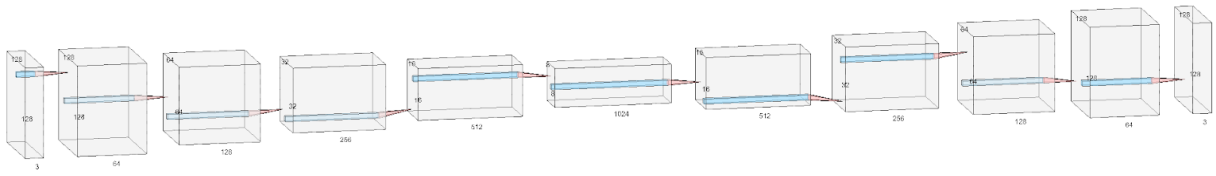
To properly train a program to achieve these goals, a dataset containing thousands of original animal pictures was utilized. This dataset was found on kaggle. The primary issue with this dataset was their lack of masks, which resulted in our group performing data

manipulation techniques to properly receive the data desired. Although our final dataset was derived from the saved dataset, our images largely differ in many respects, including size, resolution, and even the data type. In order to make sure these pictures will work correctly in the algorithms, modifications were made.

The first thing to note about the dataset was their difference in size: some provided images contained thousands of pixels for both length and width, while others barely contained a hundred. This resulted in an established fixed width and length for each image in an attempt to reduce the workload in the algorithm part. The fixed width and length was created by using an average of every image. After this, each image in the dataset was cropped to match the new width and length. To ensure the images were still recognizable after being cropped, the center part of the original picture was selected. The resulting dataset became our newly generated pictures.

After acquiring a dataset of images of similar sizes, the next step was to create masks, blocking out pixel information. This was required for training and testing purposes. To accomplish this, each mask was generated manually through a program built in python. This program would iterate through the dataset and add a white box atop the image to “mask it.” Furthermore, this masking process was done randomly, so no two images have exactly the same lack of pixel information. The figures to the right show the dataset before and after being modified.

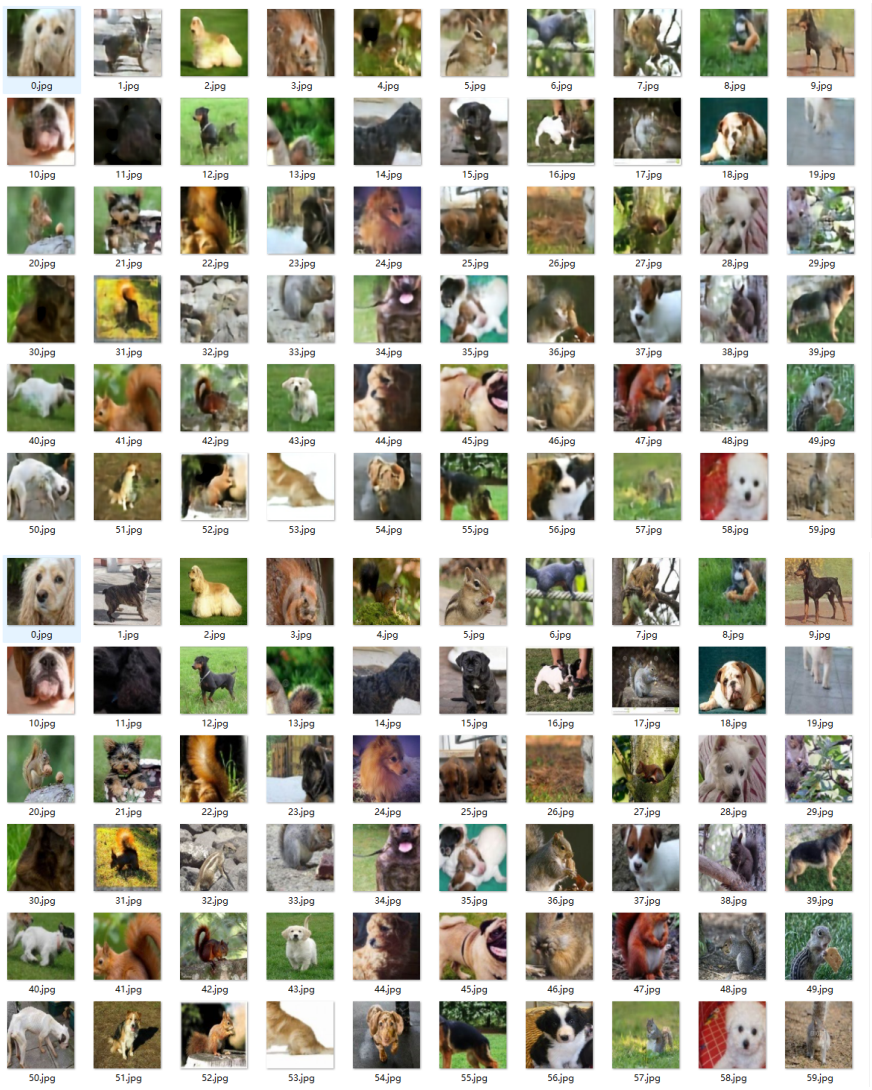
After being provided with the required dataset, the data could be fed through the program to fill in missing pixel information. The figure below shows the architecture of the fully convolutional layer.



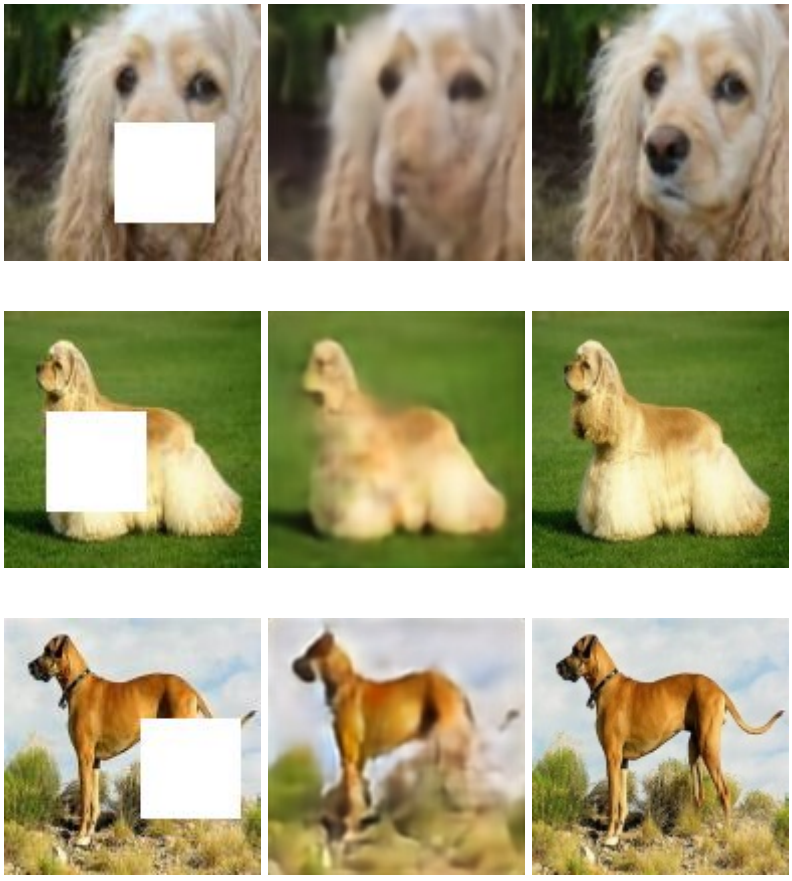
The input dimension is 128 by 128 by 3. A convolutional layer with the same padding, which ensures that the output layer has the same size as the input layer, is followed by a max-pooling layer to reduce the dimension. The Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training the model. Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction.

For each following convolutional layer, we doubled the feature maps in order to increase the ability of the CNN to extract features. After getting the 8 by 8 by 1024 latent space, we implement an upsampling layer with the nearest interpolation to upscale the dimension followed by a convolutional layer with the same padding. Each convolutional layer is followed by a batch normalization layer, which has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks. Also, we use ReLu as the activation function because it is fast and can avoid the vanishing gradient. The model is compiled using the mean squared error loss function and the evaluating metrics as the mean absolute error.

Results and Analysis



The resolution of each image is reduced because our model predicts the whole image but not just the mask. As you can see here these are some good results as well as some bad ones. We noticed that the poor performance of the current loss function is partially related to its convergence properties. So, to improve the results, we have to discover other l2-based approaches.



left mask middle predicted right real image

Conclusion

In conclusion, we used CNN and ReLU method to create an image inpainting application and it is able to recover the destroyed pictures back to what they look like normally by building our own model to achieve.

Reference

Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, Alexei A. Efros; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2536-2544