CS 166 Final Project: Double-DIP

Brady Bhalla

1 Introduction

For my final project, I implemented the Double-DIP ("Double Deep Image Prior") framework (introduced in [1]) and used it to perform image segmentation, transparency separation, and watermark removal. I was able to achieve very good results in scenarios where ambiguity could be resolved and less optimal results otherwise. I also tested the denoising and inpainting abilities of a single Deep Image Prior (DIP), which helps show that the Double-DIP model works so well because of the strength of the DIP at modeling images.

Before introducing how Double-DIP works, we need to look at the algorithms behind it. Convolutional Neural Networks (CNNs) are the basic building block of the model. A single convolutional layer performs convolutions with learned kernels to transform an input multi-channel image into an output multi-channel image. A CNN is constructed from multiple convolutional layers and can be trained using the backpropagation algorithm in order to make predictions from data. The specific CNN architecture used in this project is the U-Net [2], which first applies a series of downsampling convolutional layers and then follows with a series of upsampling convolutional layers, outputting an image with the same size as the input image. In addition, convolutions applied to outputs from the downsampling layers are concatenated with inputs to the upsampling layers which allows for more paths by which information can flow through the network.

The Deep Image Prior is based on the observation that the structure of CNN generators such as U-Nets are very good at modeling low-level image statistics even without being trained on any data [3]. Because of this, images generated by these networks are likely to be similar to real-world images, forming a prior that can be used to solve inverse problems such as denoising and inpainting. This is done by simply choosing a random fixed input to the U-Net and training the model to produce the output of the corrupted image. Because low-level image statistics are built into the structure of the model, it will likely do a good job of recovering a realistic uncorrupted image by its training process.

Finally, Double-DIP uses multiple DIPs in order to decompose images in a variety of ways [1]. The basic framework is to use two DIPs to generate image components and a third DIP to generate a mask used to combine the components. By defining the loss as follows, the Double-DIP will generate a useful reconstruction of the original image from its components. Let the component DIP outputs be y_1 and y_2, and let the mask DIP output be m. The loss of the entire system has three weighted parts:

1. **Reconstruction loss:** combining the components using the mask should return the original image I, so $L_{reconst} = \|(m \odot y_1 + (1-m) \odot y_2) - I\|$ where $\odot$ is elementwise multiplication. The reconstruction $m \odot y_1 + (1-m) \odot y_2$ will be close to y_1 where $m \approx 1$ and close to y_2 where $m \approx 0$.

2. **Exclusion loss:** the two component DIPs should actually produce distinct components of the image, which is enforced by penalizing them for having correlated gradients. This is defined to be $L_{excl} = \sum_{n=1}^{N} \|\Phi(y_1^{\downarrow n}, y_2^{\downarrow n})\|$ where $\Phi(T, R) = \tanh\left(\sqrt{\frac{\|\nabla R\|}{\|\nabla T\|}}|\nabla T|\right) \odot \tanh\left(\sqrt{\frac{\|\nabla T\|}{\|\nabla R\|}}|\nabla R|\right)$ and $T^{\downarrow n}$ is a bilinear downsampling of T with factor 2^{n-1} [4].
3. **Mask loss:** depending on the application, a different loss can be applied to the mask in order to produce the desired results. For example, in image segmentation the mask should be roughly binary, so we use $L_{mask} = (\sum |m - 0.5|)^{-1}$ to encourage the values to be close to the extremes.

Adding up each of these loss components gives a measure of the model's performance with regards to the particular task, so backpropagation can be used to minimize the loss and increase the Double-DIP's performance. After training, useful information can be taken from the mask and/or components depending on what task the framework was applied to.

2 Methods

The first step of implementing Double-DIP was to create a U-Net which could be used as a Deep Image Prior. I decided to do this in PyTorch because it is the machine learning library I am most comfortable with and it has a lot of auto-differentiable built-in functions. The Double-DIP paper does not discuss specifics of the model architecture, so I based the U-Net model on Figure 21 in the Deep Image Prior paper [3]. This paper also contains hyperparameters that were successfully used for different Deep Image Prior tasks, so I used these as a starting point for my experimentation.

After creating a functioning U-Net, I tested it on inpainting and denoising, two tasks performed in [3] that only require a single Deep Image Prior. Denoising is the simplest task, where the DIP is trained with L2 loss between the output and the target image. Inpainting involves filling in the pixels removed by a fixed mask. Training the DIP for the task is almost identical to denoising, except that only pixels which are not removed are considered in the loss. The DIP performed very well on these tasks and there were not any issues getting them to work as expected.

Next, I implemented image segmentation, transparency separation, and watermark removal using the Double-DIP framework. For image segmentation, the setup and losses are exactly as described in Section 1, with two DIPs generating components of the image and an almost-binary mask to combine them. The model was able to separate the image into two components, but it struggled to get the expected results. In an ideal segmentation, the foreground is separated from the background and the mask is completely binary. In addition, Figure 1 of [1] shows that the component outputs should extend past the mask boundary (the image is a zebra standing on grass, and the components are a full zebra pattern and a full grass pattern), but this effect was very difficult to reproduce. Using a simple image with distinct foreground and background, I spent many hours tweaking the model hyperparameters and coefficients on different terms of the loss until I was able to get an effect similar to the one in the paper. However, it only happened on around half of the training runs and did not work well for more complicated images.

Transparency separation has a similar setup to image segmentation except the mask is a constant between 0 and 1 rather than a binary mask (so we get rid of the L_{mask} loss term). The task is to take an image formed by two overlayed images and recover the originals. Double-DIP

had a big problem with ambiguity on this task. Because there is only a single image to start with, there are likely many ways that it could feasibly be divided into components with low loss. I found that for all combinations of parameters I tried across a few different images, none of them were separated correctly. The Double-DIP paper also observed that the ambiguity of separation is much higher in this task than in image segmentation, and notes that if the model is adapted to take multiple images with different transparency mixtures then the ambiguity will be resolved [1]. This is done by creating a (constant) mask for each input image and modifying $L_{reconst}$ such that it adds the reconstruction loss for each input image. I added this to my transparency separation model, and also added random noise to each of the input images so the transparency separation couldn't be solved just by a linear system. With the removal of ambiguity, Double-DIP was able to correctly separate the transparent layers and even removed the added noise from the inputs.

Watermark removal is a special case of transparency separation because watermarks are just transparent content placed on top of an image. The watermark color can be assumed to be solid white but the mask can no longer be constant because only certain parts of an image have a watermark. With a single image, there would be ambiguity over what part is the watermark and what part is the image, but luckily identical watermarks are often placed on multiple different images. This can be put into the Double-DIP framework in order to simultaneously remove watermarks from a group of images. A DIP is created for each watermarked input image and a single DIP is created for the shared mask. Since the watermark color is assumed to be white, the reconstructed image from output y and mask m is $m \odot y + 1 - m$. The losses are computed almost the same way as described in Section 1, but the exclusion loss is computed between every pair of DIP outputs. The mask loss is not required here but can have benefits especially when the watermark resembles a binary mask. This task required a good amount of parameter tuning, but working surprisingly well overall. A major challenge was finding the right balance of parameters to drive the gradients of images apart while not causing an image to lose detail, but this could be done successfully in most cases with enough time spent on fine-tuning hyperparameters.

3 Results

My results were pretty good overall for the methods I tested, but it required a lot of fine-tuning of hyperparameters and may not generalize as well to other input images for certain applications. To begin, the single DIP tasks of denoising and inpainting yielded very strong results. Figure 1 shows the results of denoising using a single DIP, while Figure 2 shows the results of inpainting using a single DIP. Recall that the inputs to the model are just the input image for denoising and the image along with the mask of removed pixels for inpainting.

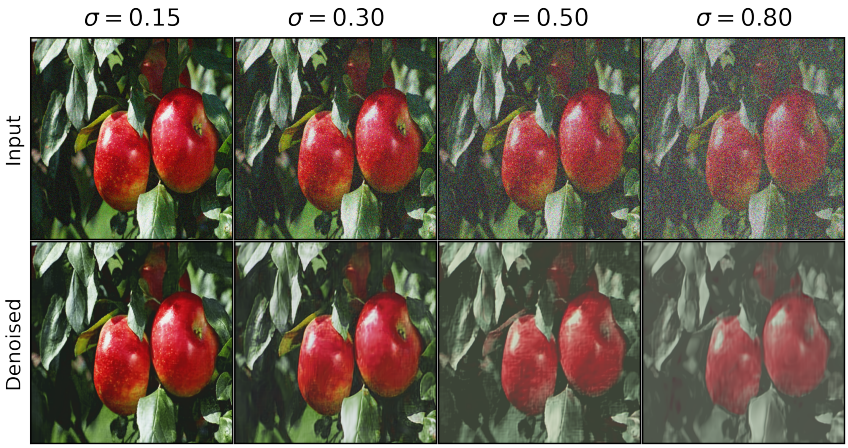

Figure 1: The top row shows noisy inputs using additive Gaussian noise that has σ defined by the corresponding column. The bottom row is the DIP output which is relatively noise-free in comparison.

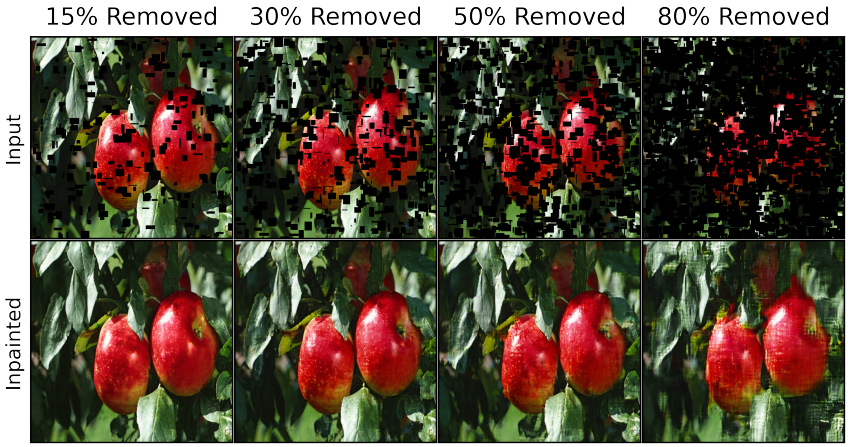

Figure 2: The top row shows the input to the model with different amounts of pixels removed. The bottom row show the DIP output. Note that the DIP has access to both the top-row image and the mask used to remove pixels.

Both of these methods work very well for low amounts of image corruption, but they also work surprisingly well when most of the image is corrupted. The DIP is able to recover the rough shape and colors of the image even when 80% of it is removed. This strength of the DIP at modeling images is what allows it to be used as such a strong prior for all the generation performed with Double-DIP. Figure 3 shows the results of Double-DIP segmentation. The model is able to find a mask which roughly aligns to the subject of each image and the colors/patterns from each component of the image extend outward past the mask boundary. The Double-DIP model is able to segment simple images despite never being programmed or trained explicitly to do so.

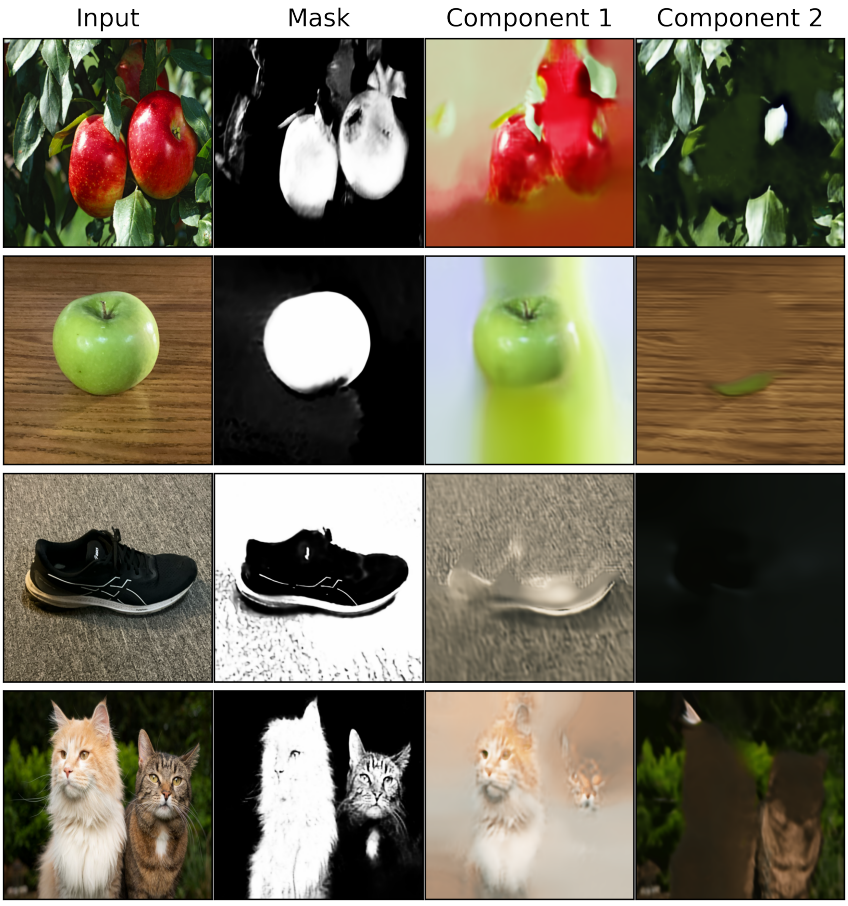

Figure 3: Each row shows Double-DIP segmentation applied to a different image. The first column is the input to the model and the other three columns are the generated decomposition. The masks are generally binary and correspond with the subjects of the images.

Figure 4 shows the results from the transparency separation Double-DIP. When two inputs are provided with different transparency mixtures, the model is able to recover the original images even though noise has been added. The last row has only one input and highlights the problem of ambiguous images from above. The model separates this image into two components, but they are not the ones making up the input image.

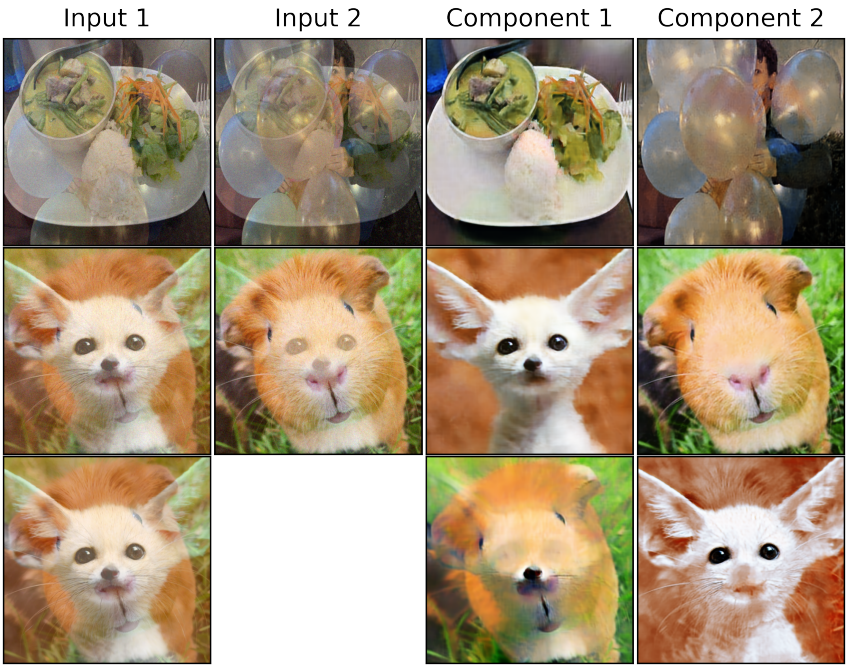

Figure 4: The first two columns of each row are inputs to a Double-DIP transparency separator and the last two columns are the recovered components. The last row shows how the model fails to separate the correct components with only a single input image.

Finally, Figures 5-7 show the results of the watermark removal model. These are some of the best results from this project in my opinion because the Double-DIP is able to identify what the watermarks are from the input images and then remove them. In particular, Figure 6 shows an example where the watermark is completely solid. The model is able to remove this and fill in the blocked part of the image without making it look too unnatural. Figure 7 highlights a weakness of this method, where the watermark mask includes extra things that are not actually part of a watermark. This happens when there is overlap between the content of the images, and is hard to prevent without additional inputs because the model does not have any extra information on what watermarks should look like.

Figure 5: A watermark removal example. The first row shows three input images with an identical watermark and the second row contains the recovered unmarked images as well as the watermark.

Figure 6: Another watermark-removal example where the watermark is completely opaque. The recovered images have the pixels under the watermark filled in and it is hard to find a trace of the original watermark in the output.

Figure 7: A watermark-removal example using real watermarked images found on the internet. This works pretty well, but the Double-DIP also falsely identifies part of the sky as included in the watermark, reducing the quality of the final images in that region.

4 Conclusion

Both the single Deep Image Prior and Double-DIP were able to succeed at all of the tasks I tested it on. I was able to get working examples for everything, but this required a lot of time spent on changing hyperparameters. Double-DIP seems to be pretty sensitive to changes in image size and complexity, so this is something to take into account if this model is being used in practice. In addition, the challenge of ambiguity in image decomposition seemed to be a big challenge for the model. For the best results in transparency separation and watermark removal I needed to use multiple input images. This worked well for my project since I could always create more images, but may not be the case in all scenarios.

References

[1] Gandelsman, Y., Shocher, A. & Irani, M. "Double-DIP": Unsupervised Image Decomposition via Coupled Deep-Image-Priors (2018). URL http://arxiv.org/abs/1812.00467. ArXiv:1812.00467 [cs].

[2] Ronneberger, O., Fischer, P. & Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation (2015). URL http://arxiv.org/abs/1505.04597. ArXiv:1505.04597 [cs].

[3] Ulyanov, D., Vedaldi, A. & Lempitsky, V. Deep Image Prior. *International Journal of Computer Vision* **128**, 1867–1888 (2020). URL http://arxiv.org/abs/1711.10925. ArXiv:1711.10925 [cs, stat].

[4] Zhang, X., Ng, R. & Chen, Q. Single Image Reflection Separation with Perceptual Losses (2018). URL http://arxiv.org/abs/1806.05376. ArXiv:1806.05376 [cs].