Subtitle Removal with Ulyanov's Deep Image Prior

Anirudh Singh Shekhawat
University of California San Diego
La Jolla, CA 92093
anirudhshekh@gmail.com

Aishma Raghu
University of California San Diego
La Jolla, CA 92093
airaghu@eng.ucsd.edu

Samer Roy Sabri
University of California San Diego
La Jolla, CA 92093
samrsabri@gmail.com

Michelle Sit
University of California San Diego
La Jolla, CA 92093
mcsit@eng.ucsd.edu

Abstract

In this paper, we attempt to apply Dmitry Ulyanov et al's Deep Image Prior architecture for image enhancement. The network achieves impressive results by only training on one data point: the image it is attempting to enhance. Using Deep Prior's random initialization of a convolutional neural network as a "handcrafted prior", we tackle the problems of super-resolution and text removal from video files. We show that instead of training the network from scratch for every frame in a video, we can remove text from images faster by using a network that is pre-trained on another image from the video. We attempt to pre-train on multiple images with unsatisfactory results.

1 Introduction

Technological advances in digital storage space, embedded systems, and cameras have enabled us to capture and consume large numbers of pictures. Image data – in the form of still frames as well as video – is rapidly becoming the primary medium of communication and knowledge transfer in a high speed world where fewer people are willing to spend time reading pages of text that can be easily communicated through a few seconds of moving images. However, while human beings can rapidly find meaning and structure in pictures by just looking at them – thus considering image data to be the faster mode of knowledge transfer – the much higher dimensionality and volume of pixel data over plain text demands processing and editing of this data which is a time consuming and difficult process. As such, there is an increasing interest in using artificial intelligence to improve image quality as well as to extract the structures and patterns in the data that are implicitly extrapolated by the human mind. Current research in the field has successfully shown the applications of neural networks to image enhancement problems such as denoising, reconstruction, super-resolution, inpainting, pre-image extraction, and flash no-flash reconstruction [4].

A lot of these applications have been limited to still images. Video data involves a large number of still frames that are rapidly traversed through to replicate motion. Applying these image enhancements to a single 100 second video file at 36fps (frames per second) requires the enhancement of 3600 still frames. Given the high-speed nature of this data consumption, there is a need for rapid data processing with minimal human intervention.

In this paper, we focus on two image enhancement tasks: superresolution, where we attempt to increase an image's resolution, and inpainting, where corrupted parts of the image are "guessed" by the network based on the rest of the image.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

2 Related Work

Past work has been done to use deep learning to generate and restore images. In general, image enhancement methods fall into two broad categories: deep learning methods, usually based on CNNs (Convolutional Neural Networks) and methods without learning where hand-crafted image priors are used. This project is interesting in part because it cuts through that distinction.

The authors of Deep Image Prior claim that their network is the first to directly investigate the prior captured by deep convolutional generative networks independently of learning the network parameters from large-scale datasets. An important ability of the network is that the network does not learn from data, but instead uses weights that are always randomly initialized, so that the only prior information is in the structure of the network itself. Its only input is the image it is trying to enhance, a mesh or random input image, and some random initializations. Instead of searching for the solution in the space of images, it looks for it in the space of a CNN's hyperparameters. Deep Image Prior uses a CNN Based Encoder-Decoder (as explained in [3]) to generate an enhanced version of the image depending on the task.

From a more theoretical perspective, what this paper shows is that a lot of information about the image and its statistics can be captured by the structure of a convolutional net that could generate this image.

We looked at one example of learning-based approaches and one example of approaches that use handcrafted priors for inpainting to understand how Deep Image Prior fits in:

- "Context Encoders: Feature Learning by Inpainting" by Pathak et al. uses "an unsupervised visual feature learning algorithm driven by context-based pixel prediction" ([7]). They predicted corrupted regions of an image from non-corrupt regions using a simple encoder-decoder pipeline with feature-wise fully connected layers between the encoder and the decoder. Their loss is the sum of a reconstruction loss (L2) and an adversarial loss where the generator and discriminator network are optimized jointly using alternating Stochastic Gradient Descent. This method does not require labels, but it does need to train on a large image dataset where parts of the input images are dropped and the network attempts to predict them. The feature representation produced by the encoder network can be used relatively successfully for other tasks like classification. Because it only trains on one image, Deep Image Prior is not as good at inpainting large portions of an image that requires some domain knowledge (e.g., inpainting an eye in a face image).
- "An Improved Image Inpainting Algorithm based on Image Segmentation," which uses a combination of watershed image segmentation and explores the image segmentation region for similar pixels to inpaint in the corrupted regions ([6]). Even though Deep Image Prior is learning-based, you will notice that this method is very similar to Deep Image Prior in the sense that both rely on self-similarity to fill corrupted regions in an image.

3 Implementation

3.1 Architecture

In this section, we describe the general architecture as well as how it is adapted to each of superresolution and inpainting.

The architecture proposed by the original authors is a purely convolutional neural network that uses LeakyReLU as a non-linearity and the ADAM optimizer. The network is initialized with an input z and random weights. The input z is a random image (each pixel is uniformly distributed between 0 and 0.1) with 32 channels and width and height same as the desired output.

The network conforms to an "hourglass" or "decoder-encoder" architecture that looks like 2 CNNs stacked on top of each other: one with 5 down blocks which does downsampling into an intermediary output that is then upsampled by the "second" CNN which has 5 up blocks and outputs a new image. As you can see in figure 1, each down block and each up block applies 2 convolutional layers each followed by a batch normalization layer and a leaky ReLU. Each down block downsamples by 2x by using a stride of 2 in its first convolutional layer. Each up block is followed by a 2x upsampling operation (bilinear for super-resolution and nearest for inpainting).

The network sometimes uses skip blocks to feed the input from early layers directly to late layers, in addition to their "natural" input from the network. Each skip block is made up of one convolutional layer, one batch normalization layer, and a Leaky ReLU. When skip connections are used, the output of a skip connection is concatenated with the output of up layer "i+1" (i.e., previous layer) to constitute the input to up layer i. This kind of skip-connection is a common technique for CNNs.

During training, based on the task, additional random noise is added to the original input z which is generated using random uniformally distributed noise.

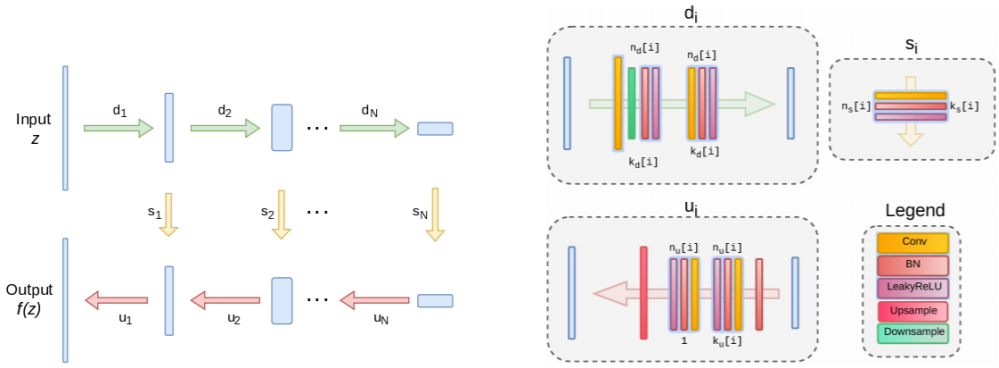


Figure 1: **The architecture used in the experiments.** We use "hourglass" (also known as "decoder-encoder") architecture. We sometimes add skip connections (yellow arrows). $n_u[i]$, $n_d[i]$, $n_s[i]$ correspond to the number of filters at depth i for the upsampling, downsampling and skip-connections respectively. The values $k_u[i]$, $k_d[i]$, $k_s[i]$ correspond to the respective kernel sizes.

Figure 1: Figure from Ulyanov et Al explaining the "hourglass" architecture

3.2 Super-Resolution

We implemented 4x resolution, which means our network's output was an image with 3 channels and 4x height and width as the original image.

We used Deep Image Prior's architecture for this task:

- For the 5 downsampling and 5 upsampling layers:
  - kernel size 3 for each convolutional layer
  - 128 features for each convolutional layer
- 5 "Skip" connections between the i-th layer of the upsampling network and the 5-i th layer of the downsampling network
  - 5 skip connections
  - kernel size 1 for each connection
  - 4 features for each connection
- Other parameters
  - input noise of 1/30
  - 2000 iterations before interruption
  - Learning rate 0.01
- To compute the loss, we downsample the output of the network in a differentiable fashion and use MSE loss between that downsampled image and the original image we are trying to enhance.

The authors of Deep Image Prior also found that they could obtain a higher output resolution by increasing input noise and the number of iterations. Thus there is a trade-off between low processing time and high resolution outputs.

3.3 Inpainting

We use Deep Image Prior's network for text inpainting. This method requires producing a mask separately (see "Creating Masks" subsection below), which "covers" the area that needs to be inpainted.

- For the 5 downsampling and 5 upsampling layers:
  - kernel size 7 for each layer
  - increasing number of features for each layer: 16, 32, 64, 128, 128
- Only **one** "Skip" connection between the 5th layer of the upsampling network and the 1st layer (from the "bottom") of the downsampling network
  - 1 skip connection
  - kernel size 1 for that last connection
  - 4 features for that connection
- Other parameters
  - input noise of 0
  - 3000 iterations before interruption
  - Learning rate 0.01
- The loss has two inputs: the original ("noisy") image with the mask applied to it (i.e., in our case, subtitle pixels are zeroed), and the network's output with the mask applied to it. In other words, nothing backpropagates back from the mask area, the inpainting area.

3.4 Creating masks for inpainting

We need to create a matrix with the same dimension as the images (but with 1 channel). This matrix should be 0 for pixels that are not part of the subtitles, and 1 for pixels that are "subtitle pixels." We developed an algorithm that works in two steps. Each step iterates over each pixel in the image.

In our image, subtitles are of a distinct shade of yellow (we simply call them yellow) that is not reproduced anywhere in the video outside of the subtitles. We mark the pixels we can detect, then mark a radius around them, and set all the pixels within the radius as 1s in the mask (everything else becomes 0).

More formally, we start with a Mask of all 0s, and then:

- Step 1: Iterate through every pixel, and mark every yellow pixel detected as a 1.
- Step 2: Iterate through every pixel, and if it's within the radius of a yellow pixel, include it in the mask.

As always, we realized later that we could have done it all in one loop. But the principle is the same, so the two hyper-parameters are:

- The pixel values (across 3 channels RGB) that should mark the pixel as yellow in step 1. We use a specific shade of yellow, but it could work for any color.
- The radius for step 2 (we use 3 and 5)

See Results section for details.

3.5 Pre-training Deep Image Prior

After getting satisfactory results for the subtitle removal task, we explored whether pre-training can be beneficial with Deep Image Prior. Rather than re-initializing weights for every frame of the video clip, we were interested in whether we could save time and achieve better results by pre-training the network on similar images instead of initializing the network's weights randomly each time. Processing video frames is particularly suited for this task because there is only a gradual change in the scene from one frame to the next.

3.5.1 Pre-training with 1 image

Initially, we looked at whether the network can learn from inpainting one frame and use this learning to inpaint a similar frame faster. The architecture used is the same with the speed up obtained on the second image as a litmus test to see whether pre-training improves the performance of the network.

We first trained the network on two different images with different subtitles independently to obtain the baseline for evaluation, and then used the model obtained after training for 3000 epochs on the first image to then train with the second image as the target. We found that a pre-trained network (i.e., trained by inpainting a different but similar frame in a video) can perform text inpainting faster than a randomly initialized network (see results in next section).

In a video where frames follow a sequential order, each frame shares similar components with the preceding and succeeding image. This follows for as long as the scene in the frame is roughly about the same and there are no cuts from one shot to another. Thus you'd expect sequential training to speed up fairly well up to a cut, and then reset and start relearning all over. In the next section, we explore alternative methods of processing and learning through multiple images as an alternative to sequential training.

3.5.2 Pre-training with multiple images

We tried to extend our pre-training technique by incorporating more than 1 image to the training step. The idea here is a little bit more ambitious: could we use Deep Image Prior more like an autoencoder, in that the networks' weights incorporates prior knowledge from multiple images.

Instead of a single image, we used four images with similar components in our pre-training step and developed four different networks to determine and compare the best results:

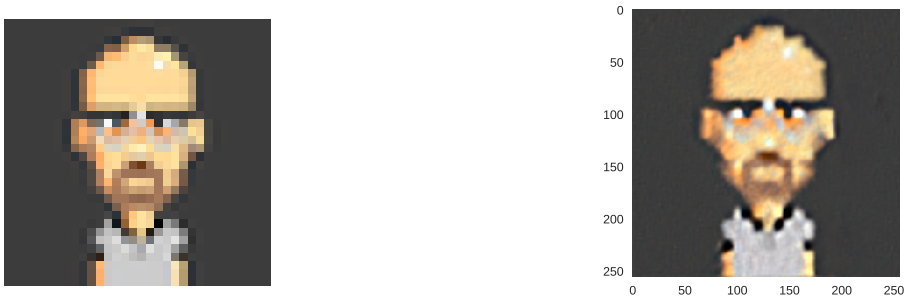
1. **M-Average:** We utilized the same setup used to pre-train the network on 1 image. In the previous setup, during pre-training, we "teach" the network to go from its random input z to one label image over 3000 iterations. Here, during pre-training, we "teach" to go from z to each image in a batch of 4 images at each iteration. The same input z is used in testing. We call it M-average because it is likely to learn to produce an average of the 4 images in the batch.
2. **M1:** We generated a random input z with 32 channels. After that, we added 4 additional channels to the input z. z_1 has the first additional channel as all ones, and the other additional channels as zeros. z_2 has the second additional channel as all ones, and the other additional channels as zeros. During pre-training, we "teach" the network to map z_1 to the first label, .z_2 to the second label, etc. During testing, we use input z_0, where all the additional channels are set to 0.
3. **M2:** We generated different uniform noisy inputs, each slightly different from each other due to the random nature of the input generator. We used these random inputs $z_1 \ldots z_4$ to train the network using mini-batch approach to create the corresponding images. The input z_5 used for testing was also randomly generated.
4. **M2-Average:** We used the same network structure as the M2 network. However, instead of generating a fifth input randomly, we averaged the random inputs $z_1 \ldots z_4$ to obtain z_5.

Each method uses the same network architecture for both training and testing. The only differences were (1) in the way pre-training inputs were generated and combined with target images and (2) how the fifth input used in testing is generated.

4 Data

4.1 Initial data used to verify our implementation

We initially tested the performance of our super-resolution network on a noisy image we pulled from Google Image Search (e.g., Figure 2a). The original image taken was of the dimensions 32 x 32 pixels and resolved up to 256 x 256 pixels. We also confirmed our networks' soundness by replicating some of the original results from the paper (see next section).



(a) Original Image

(b) Output Image at Epoch 940

Figure 2: Comparing the original image and the super-resolution image.

4.2 Data for Subtitle Inpainting and Pre-training

For subtitle removal, we selected a clipping of the 1977 Woody Allen movie Annie Hall, obtained from YouTube. [5]. The video is a 2 minute 31 seconds clip of the main protagonists having a conversation on the balcony. The clipping contains English subtitles in yellow text transcribing their thoughts during the conversation. The video is made up of 1280 by 720 pixel still frames served at a speed of 25 fps. Thus our dataset for the subtitle removal problem consists of 3,775 still frames. For the purpose of this task, we only consider the video portion with the subtitles - from the 0:48 mark to the 1:39 mark. As the only images required during the training step are the target images containing the subtitles, the reduced datasets helps ensure we are not training the model on frivolous data. Figure 9a shows a still from the video with the subtitles.

5 Results

5.1 Reproducing a Super Resolution result

We were able to build the network architecture described in the previous sections using PyTorch.

We tested our network on small images in order to reach convergence faster. We got encouraging preliminary results for increasing the resolution, the results for one image are included in figure 2.

We then reproduced the results for the images used in the paper and the supplementary material. We were able to produce results similar to those of the paper, thus confirming the soundness of our implementation. We show here the result on doing super-resolution on a zebra image, which is originally 584×388(Figure 3) and we super-resolve it to 2304×1536. Shown in Figure 4 is a zoomed-in patch of from the zebra image for the original and superresolved image. Comparing Figure 4a with Figure 4b, we observe that the model fills in the texture of the fur of the zebra very well.

Figure 3: Low Resolution Image

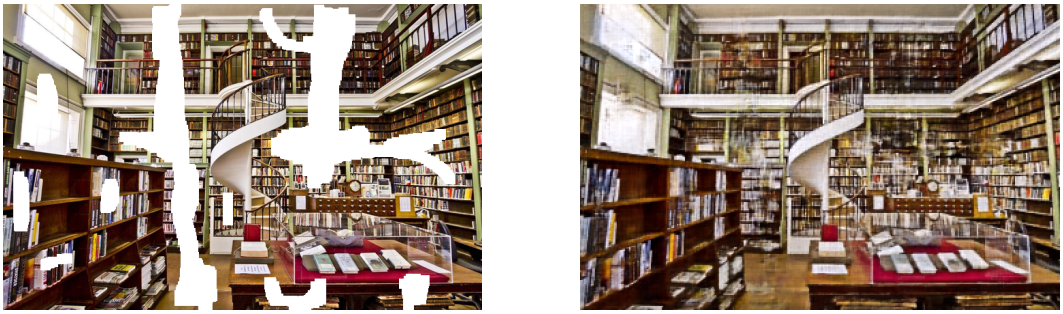
(a) Original Image – Zoomed In

(b) Superresolution – Zoomed In

Figure 4: Comparing the original image and the super-resolution image.

5.2 Reproducing an inpainting result

We were able to reproduce Deep Image Prior's results on text inpainting. In Figure 5a, we see the image of a room with a mask applied which removes large patches from the original image. We then use this masked image to train our model on this masked image and allow the network to fill in the removed patches. As we can observe, the network does a good job in filling the missing patches with relevant pixels.



(a) Image with mask applied (b) Inpainted image

Figure 5: Comparing the masked image and the inpainted image

5.3 New results with images from video

Next, we show results on one frame we extracted from the movie Annie Hall (see Data section for details). In figure 7, you see the original image as well as the image the network produces at the end of training. In the next figure (6), you see the output of the network every 300 iterations until 3000. Zooming in will allow you to convince yourself that the inpainting (in the subtitle region) is getting better as the network iterates.

(a) Output Image at Epoch 0 (b) Output Image at Epoch 200 (c) Output Image at Epoch 300

(d) Output Image at Epoch 500 (e) Output Image at Epoch 800 (f) Output Image at Epoch 1100

(g) Output Image at Epoch 1400 (h) Output Image at Epoch 1700 (i) Output Image at Epoch 2000

(j) Output Image at Epoch 2300 (k) Output Image at Epoch 2600 (l) Output Image at Epoch 2900

Figure 6: Images every 100 epochs from our network.

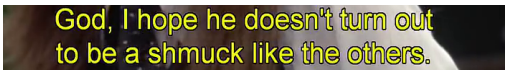
(a) Original Image with Subtitles

(b) Output Image at Epoch 2900

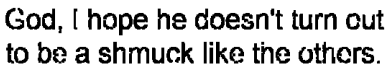
Figure 7: Comparing the original frame and the text-inpainted frame.

5.4 Creating filters for inpainting

Figure 8 shows a region of the original image with subtitles, as well as different filters produced with different radius values (1, 3, and 5). As shown in the figure, the filter size of 5 is the least conservative (marks the most pixels as subtitle pixels). It is, however, the one we use for our training because it's the one that captures all the pixels on all of the images we extracted from our video.



(a) Region of original image with subtitles



(b) Same region for filter with radius 1

(c) Same region for filter with radius 3

(d) Same region for filter with radius 5

Figure 8: Original output and masks obtained for different values of radius (1, 3, and 5)

5.5 Pre-training with one image

Our empirical results show that the the network indeed can adapt faster to the new frame when using weights learned while processing the previous frame. Figure 9 shows the two subtitled images used to test whether pre-training helps. The image on the left is the one used during pre-trained, and the one on the right is the test image.

Figure 11 shows the results of the pre-trained model vs. the untrained model during the same iterations of testing. The network that pre-trained on a similar image converged in around 700 iterations instead of 3000 for the network that was untrained (i.e., intialized randomly).

(a) Pre-Training Image

(b) Image for Transfer Learning

Figure 9: Images used for Transfer Learning

(a) Output Image at Epoch 200
Learning from scratch

(b) Output Image at Epoch 200
Transfer Learning

(c) Output Image at Epoch 500
Learning from scratch

(d) Output Image at Epoch 500
Transfer Learning

(e) Output Image at Epoch 1000
Learning from scratch

(f) Output Image at Epoch 1000
Transfer Learning

(g) Output Image at Epoch 1500
Learning from scratch

(h) Output Image at Epoch 1500
Transfer Learning

(i) Output Image at Epoch 2000
Learning from scratch

(j) Output Image at Epoch 2000
Transfer Learning

Figure 10: Network learning from scratch vs. transfer learning by using a pre-trained network

To better understand this speed up, you have to look closely at Figure 10. In particular, zoom in and analyze the woman's tie. You will see that the detail reached by our pre-trained network on the 500th iteration is only reached by the network learning from scratch at 2000 iterations.

(a) Without Pre-training at Epoch 2900

(b) With Pre-training at Epoch 700

Figure 11: Images used for Pre-training

5.6 Pre-training with Multi-images

As you can see in figure 12, of the four networks, the M1 network appears to have learned the underlying patterns the best. The network learned that the tie had dots so it attempted to place a few in the area that the subtitles had obscured. One of the dots is too large and does not fit well with the rest of the tie. However, the fact that it learned this pattern is impressive compared with the rest of the networks. In addition, the texture of the hair appears to be natural enough compared to the rest of the networks. M-Average has the image with the most natural appearing features. The tie is not distorted compared to the other images and the hair is straight instead of slightly crooked where the subtitles were. However, the network is conservative in placing details such as the dots or may not have learned them at all.

The M2 networks did not perform very well in the end. The M2 network clearly shows where the subtitles had been filled in. The M2-Average network simply takes the average pixels in the surrounding area where the subtitles gap had been and attempts to fill it in. This resulted in a block around the knot of the tie and a clear separation in the hair where the subtitles were.

(a) Best Image from M1 Network

(b) Best Image from M-Average Network

(c) Best Image from M2 Network

(d) Best Image from M2-Average Network

Figure 12: Best Images from Multi-Image Transfer Learning

6 Discussion

We were able to reproduce some of Deep Image Prior's results at a satisfactory level. In this section, we focus on our attempts to extend Deep Image Prior's work and do pre-training.

6.1 Pre-training with 1 image

Our results show that the network is able to converge faster by fine-tuning the architecture of a previously trained network. This network only needs to be trained on one other image from the same frame. This enables the network to continue to reuse information from similar frames and spend less time repetitively training on the same weights and allocate resources to efficiently processing new information. We hypothesize that the network converges faster with transfer learning because it reuses some of the features it learned from past frames.

This speed-up is crucial since training the network for the task takes time and even short video clips would contain thousands of video frames. Thus, exploiting the structure in video frames leads to faster processing of the clip.

It is important to note that there was no indication that pre-training on a similar image allows Deep Image Prior to produce qualitatively better results.

It would be interesting to find out how speed gains correlate with the similarity in images. We suspect that pre-training on any image would produce faster convergence simply because it makes the "CNN Prior" from the randomly initialized network slightly "smarter."

6.2 Pre-training with Multi-Images

We found that networks that are pre-trained on several images generally converge faster when it comes to reproducing the non-corrupt portion of the image. However, their convergence for the corrupt portion is more unstable: it can be reached faster than the 3K iterations needed when training from scratch, but the optimum reached is not stable. For example, for the M1 network, the Mavg network achieves the best result at 2200 iterations and gets worse after that. This is possibly due to the fact that pre-training on several images yields a more unstable network.

So far, it seems like pre-training on one image is superior to pre-training on 4 images. Indeed, pre-training is 4x slower for multiple images because the batch size for each iteration is 4x greater. In addition, a network pre-trained on one image converges much faster (300 vs. 2400 iterations) and its optimum is much more stable.

Among all pre-training methods for multiple images, it seems M1 is the best method. This is probably due to the fact that the network has to learn 4 separate representations for each image in the pre-training. More testing would be required to come up with more conclusive answers.

7 Conclusion and Future Work

In this paper, we successfully replicated and extended Ulyanov et al's Deep Image Prior. One appeal of this network is that it is able to learn from a single image instead of a large dataset and can be generalized for other image processing tasks through small variations in the network weights. We used this to our advantage to in-paint the backgrounds of frames taken from the film Annie Hall and remove the subtitles on the screen. We then further extended their network by successfully implementing pre-training using the model through sequential chaining of images as well as training on multiple images at once.

Pretraining on one image yielded much faster convergence, but we were not able to achieve a lot of success with pretraining on more than one image.

To further our work, we would like to explore the implementation of clustering algorithms to pick out homogeneous images to eliminate human involvement in the pre-processing image selection step. This could be used to pick out the appropriate frames from the videos to be used for pre-training, obtain the results and then restitch the frames in the correct order to generate a full video instead of just the individual frames. Deep image prior could also have applications in predicting successive frames based on previous frames that could be used to generate panoramas, one-take shots or videos, and if successfully implemented, possibly animation of objects.

We hope that with the right architecture changes, a deep image prior can behave like an auto-encoder without requiring as much training data. Achieving this would require a better understanding of how an image is represented by the weights in a generative CNN, and how information from several images can be recorded and used by such a network.

References

[1] Anaya, J., & Barbu, A. (2014). RENOIR-A Dataset for Real Low-Light Image Noise Reduction. arXiv preprint arXiv:1409.8230.

[2] T. Plotz and S. Roth, “Benchmarking denoising algorithms with real photographs”, CoRR,vol. abs1707.01313, 2017

[3] Li Y., He S., & Deng L. CNN-based Encoder-Decoder for Frame Interpolation. Available: http://cs231n.stanford.edu/reports/2017/pdfs/701.pdf

[4] Ulyanov D., Vedaldi A., & Lempitsky V, "Deep Image Prior", CoRR,vol. abs1711.10925, 2017.

[5] Allen W. Annie Hall (1977) - The Balcony Scene. Available: https://www.youtube.com/watch?v=3tcH2zSqVCc&t=1s

[6] Ying H., Kai L., & Ming Y. "An Improved Image Inpainting Algorithm Based on Segmentation", Procedia Computer Science vol. 107, 2017, 796-801

[7] Pathak D., Krahenbuhl P., Donahue J., Darrell T., & Effros A. "Context Encoders: Feature Learning by Inpainting", v2 arXiv:1604.07379

8 Appendix

8.1 Individual Contributions

Samer - Implementing Deep Image Prior architecture, Filtering Subtitles, pre-training with one image, Multi-image pre-training (M1), Report Consolidation

Anirudh - Literature Study, Implementing Deep Image Prior architecture, Reproducing Super-resolution Results, Multi-image pre-training (Mavg), Report Consolidation

Michelle - Dataset Selection, Literature Study, Reproducing inpainting Results, Multi-image pre-training (M2avg), Report Consolidation

Aishma - Input Processing, Vanilla DIP, Multi-image pre-training (M2), Model Evaluation, Report Consolidation