Deep Scene Interpolation

John Clow
jclow@stanford.edu

Brendan Corcoran
bmc2016@stanford.edu

[Matthew Volk]
mvolk@stanford.edu

Abstract

In this paper, we aim to accurately predict intermediate, uncaptured frames given a discrete sequence of images of a scene, taken by a moving observer. We build on recent work utilizing convolutional neural networks (CNNs) for novel image generation, improving the appearance of texture and structure. Our network consists of an encoder-decoder CNN that predicts a dense correspondence and masks. Finally, we apply a blending network that synthesizes a final prediction from the input images, correspondences, and mask.

1. Introduction

Novel view synthesis, generating unseen new perspectives of an object or scene from existing views, has many desirable applications. Examples include interpolating 2D images of a virtual 3D tour (for applications like Google Street View), editing photos by manipulating 3D objects, and compressing videos and light-field data [2] [5].

1.1. Related Work

There is a wealth of previous work related to this topic, involving various degrees of geometric analysis and machine learning techniques.

Geometry-based methods

On one end of the spectrum, there are approaches that involve little or no machine learning, instead carefully modeling the geometry of the scene. This situation can be viewed as a structure-from-motion (SFM) problem, where the complete 3D reconstruction of the scene must be inferred as well as the camera poses for each image [1]. To generate the unseen middle image, one can simply interpolate from the adjacent camera poses and then render the intermediate perspective. This approach suffers from the usual difficulties of SFM, including textureless regions that require human intervention. This method also makes it difficult to predict occluded regions causing there to be holes in the output image. These two problems make it difficult for geometry-based methods to produce realistic-looking images.

Direct pixel generation methods

On the other end of the spectrum are purely machine learning-based approaches. These approaches address the problem of occluded regions, as learning can generalize to make a guess at under-constrained problems. One common such approach involves feeding an image and a transformation into a neural network, which directly computes and outputs the resulting image. Tatarchenko et al. implement such a network [6], with an encoder-decoder CNN. The underlying idea is that the encoder network learns a low-dimensional parametrization of the object or scene. Then, using this parameterization and a transformation vector, the decoder network learns how to rehydrate the parametrization and render the appropriate perspective, generating the pixels of the output image directly. The major problem with such encoder-decoder architectures is that they tend to produce quite blurry images.

Mixed methods

Another set of projects seeks to combine the advantages of both geometric and deep learning techniques. Instead of using a CNN to directly generate output pixels, this approach leverages networks to learn how to sample pixels from the original images. [7] [3]. Though these projects produce good results for synthetic objects on a white background, their performance on more complex real-world scenes is unsatisfactory. More recently, researchers at Google analyzed more advanced ways of modeling pixel flows, in which they achieve good results with multilevel flow models with more advanced neural networks and loss functions[4].

2. Technical Approach

We base our approach largely on [3] and [7]. Our pipeline has two stages: an encoder-decoder network and a view morphing network.

2.1. Problem Formalization

We define the scene interpolation problem as follows: given two sequentially-captured images of a scene, generate the unseen interpolated image. As we discuss later, our approach generalizes to generating a series of intermediate scenes, but we constrain the problem to reproducing the scene equidistant in time from the two inputs.

2.2. Dataset

For all of our experiments, we used the KITTI Vision Benchmark dataset, which contains street scenes captured by the camera of an autonomous car.

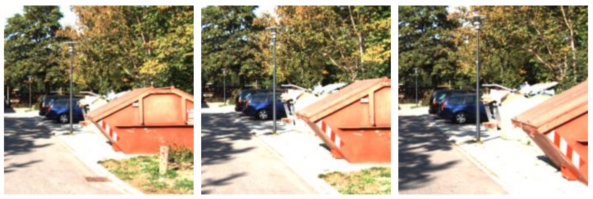
Figure 1: Sample KITTI Dataset Images

From the raw data, we split the videos into frames and cropped them into 224 x 224 images. We divided this into sets of 3 consecutive frames. Each of these triples served as one data point. Image 1 and Image 3 are the input images to the model and the middle image (Image 2) serves as the ground truth against which we compare our prediction. We varied the number of frames between Image 1 and Image 2 between 1 and 6 frames. The number of frames between the images was symmetric such that Image 2 always was centered between the other two images.

However, it is worth noting that the speed of the car and surrounding objects varied significantly, such that the number of frames between two images is not always predictive of the magnitude of the difference between them.

2.3. Architecture

As illustrated in Figure 2, the encoder network first encodes the two rectified images. The same encoder is applied to both images such that the two image parameterizations are comparable. The resulting features are concatenated and fed into two separate decoder networks, one that finds correspondences C and one that finds masks M_1 and M_2.

The view morphing network is an entirely fixed operation that requires no learned weights. As shown in Figure 3, the view morphing network uses the dense correspondence matrix C to create a “best guess” output image for each of the input images. Then, it takes the Haadamard product of these “guess” images and the learned masks M_1 and M_2, and sums the two resulting images to

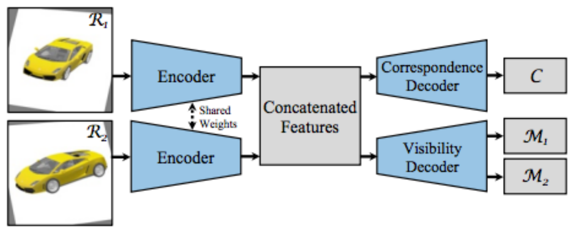

Figure 2: Encoder-Decoder Network [3]

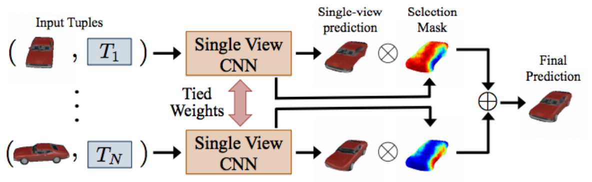

Figure 3: View Morphing Network [7]

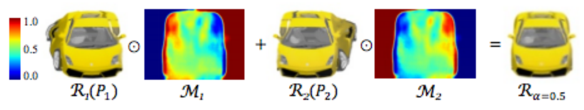

Figure 4: Explicit Final Output Calculation [3]

give the final output prediction. A more explicit mathematical formulation of what this network does can be seen in Figure 4. This whole pipeline is entirely differentiable, since we use bilinear interpolation when creating the guess images.

2.3.1 Evaluation

Our initial evaluation approach was to calculate the Euclidean distance of the predicted image from the ground truth. We believed that Euclidean distance would be appropriate for this task because we believed the resulting image would be very close to the ground truth, making harsh penalizations of small differences justifiable.

As such, we used Euclidean loss as our baseline to train the network. However, the resulting images lacked detail, as Euclidean loss encourages the algorithm to fit a uniform color. This effect can be seen in Figure 7, in which the generated image has clearly been desaturated. Further, fitting an exact texture is difficult because Euclidean loss significantly punishes even slight misfits of the texture due to its squared term. This also tends to produce a blur effect that is immediately noticeable to a human observer.

After inspecting the gradient flow, we hypothesized that we needed to more heavily weigh local gradients in our pipeline to improve texture and help with the bluriness problem. As such, we tried implementing what we call a Texture Loss algorithm. The main premise is that we penalize images for being smooth, so therefore, for every pixel in both the ground truth and the predicted output, we subtracted every pixel from the pixel 1 pixel to the right (a difference image) and took the L2 loss of the two difference images. We did the same thing for the 1 the one pixel above it (in the y direction) as well as for 2,4,8 pixels away in both the x and y direction.

We thought that because we were having problem with textures in our old images, that this would help substantially, but it turned out that our real problem was that the network with the L2 loss function was not being trained enough. When we actually trained both the L2 loss network as well as the texture loss network for enough time (12 epochs, 4 hours), both of them produced great results, and both performed similarly. In some respects, the L2 loss function did better, as the texture loss variant sometimes tried to force weird looking textures. For example, in figure 5, we can see that the textures in the tree in the top left of the texture loss image is weird while the one of the normal loss looks ok. After some calculations after the fact, we also realized that texture loss mainly served to punish elements being a few pixels offset instead of textures being smoothed out, as the latter generated even higher relative differences than L2 loss.

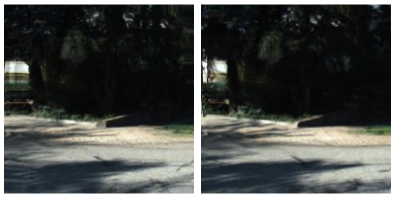
Figure 5: Validation Set Texture Loss (left) vs L2 Loss Images

3. Experiments

3.1. Baseline

Before performing experiments on our full model, we built a naive network against which we could compare our results. Our baseline implementation is a direct pixel generation method, involving an encoder-decoder CNN that takes in two images of dimension (224 x 224 x 3) and attempts to produce the intermediary morphed image. This baseline does not use rectification, and it does not use any sort of correspondence matching to align images.

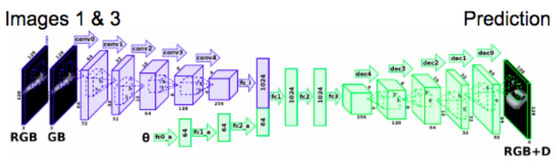

Figure 6: Baseline Model Architecture [6]

The encoder portion of the network consists of first concatenating the two input images into a tensor of shape (224 x 224 x 6), then feeding it through 6 convolution layers in the format [Conv - ReLU - MaxPool] x 5 - Conv, resulting in a 7 x 7 x 1024 volume. The decoder consists of 5 transpose convolution layers, resulting in a 224 x 224 x 3 output image. This network is visualized in Figure 6

To train this model, we used Euclidean Loss with an Adam optimizer (initial learning rate of 2e-3). We found that this baseline model, as

Figure 7: Baseline Model Performance

predicted, produced unsatisfactory results on this dataset. As is apparent in Fig. 7, the results lose a lot of color information and appear very blurry.

3.2. Overfitting a small training set

This and the following subsections discuss the performance of our full model.

To test the effectiveness of our full network, we first trained on a small set of image triples and intentionally overfit the data to ensure the network was at least capable of generating intermediate images. As demonstrated in Figure 9, the model was able to produce a reasonable result, though a far from perfect one. The generated image correctly models the movement of the car in the bottom right of the initial picture along with the relative movement of the tree in the upper left. However, there are noticeable flaws, including the shadow effect in the tree, making it blurry, and the loss of texture in the street.

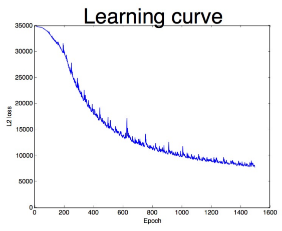

Figure 8: Absolute Proof of Concept Loss Curve

3.3. Results and Discussion

Finally, we run the full model on a large training (16,000 triples) and validation set (5,696 triples).

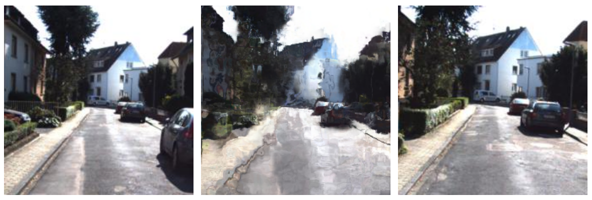
Figure 9: Overfit Proof of Concept

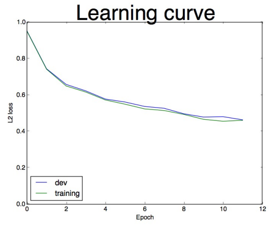

Figure 10: Normalized Loss Curve

First, we can examine the loss function of the training run (Figure 10). Here, we plot normalized L2 Loss, meaning the L2 loss of the output image relative to the L2 Loss of an image that is a 50-50 blend of Image 1 and Image 3 (that is, the most naive output image). In this plot, we see that the model is initially blending the input images uniformly (normalized loss is 1.0). However by the end of training, the normalized loss is below 0.5 indicating quite an improvement.

Next, we examine a few of the resulting images to see what the model does well and where we can improve. In Figure 11, note how the generated image was able to synthesis a shadow on the street that is different from (and interpolates between) the shadows in Image 1 and Image 3. A shortcoming that is observable in this example is the thick black line that delineates the sidewalk from the street appears twice in the generated image. That is, it was not able to merge the two lines from the input images into one line. See the appendix for more generated samples. The generated images tended to be worse the more the two images differed from each other, as can be seen in Figure 18, in which the two inputs have almost no overlap whatsoever.

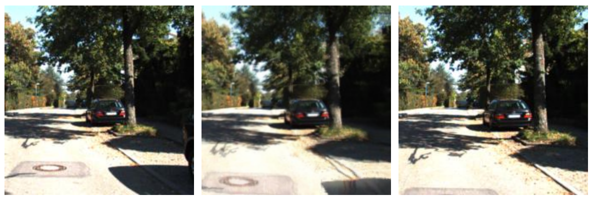
Figure 11: Example generated image (middle)

3.4. Additional Visualizations

To clarify what the view morphing network does, we present a visualization of what each step in the network does for a given input pair. In Figures 12 and 13, we see our two input images before and after the correspondence matrix C has been applied to them. Intuitively, this correspondence matrix approximates the flow of individual pixels between the first and second inputs, and it applies this shift. One can see that, as the camera moves forward, the silver car on the left of the image moves out of the frame (comparing the left picture in Figures 12 and 13). In Figure 12, it is clear the applying the correspondence matrix attempted to take this into account: the car is shifted to the left; the opposite is apparent in Figure 13. However, the two images have clear artifacts of the original left. A clear example of this is the stretched car in Figure 13. To rectify this, we learn and apply two masks.

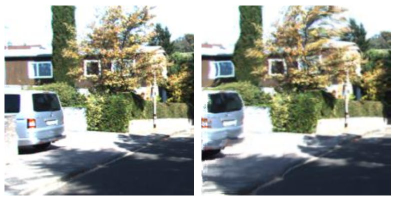
Figure 12: Image 1 Before and After C Applied

In 14, we show the generated masks M_1 and M_2 for two input images and show the resulting output image. These masks are (224x224x1)-dimensional matrices of weights in the range $[0, 1]$. In the visualization, small weights are blue while large weights are red. The thing to notice

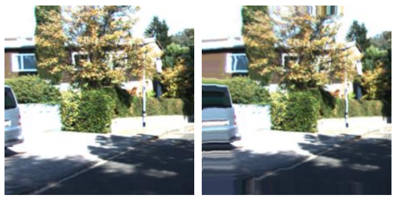

Figure 13: Image 3 Before and After C Applied

here is that from Image 1 to Image 3, the camera moves forward and to the right. This means that certain points and surfaces that need to be in the output image do not appear in Image 3 (specifically points at the left and bottom of Image 1). We can see that this is exactly what the network learned to do. The bottom and left sides of Image 3's mask are blue, meaning we should not use pixels sampled from that region in the output image. The inverse is true of Image 1. These masks are applied to the two inputs and the results are summed, yielding the predicted image to the right. Though far from perfect, it does a reasonable job of interpolating the car between its position in the two input images.

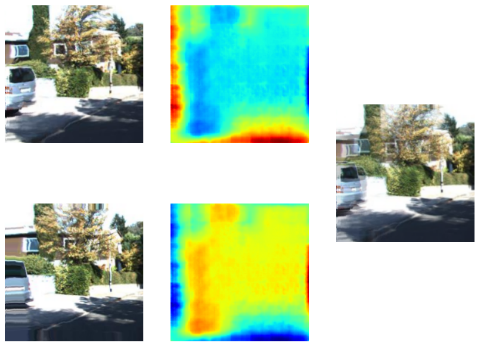

Figure 14: Blending Operation Visualization

4. Conclusion

We present an implementation of a network that takes advantage of both image geometry (generating correspondences and mask matrices) and deep learning. We show that this encoder-decoder network is able to generate significantly clearer and more accurate images than a simple, purely neural encoder-decoder network, as can be seen when comparing the results of Figure 7 against any of our final model outputs. We further show a quantitative improvement, as well, by normalizing our loss against the simplest possible loss in Figure 10. Though there remains significant room for improvement, in particular in (1) eliminating image blurriness and (2) improving output image texture, we hope that our results add to the corpus of work indicating the potential of hybrid methods for view synthesis.

5. Future Work

We believe one other promising application of deep scene interpolation is in video compression. If deep scene interpolation performance continues to improve, one could get away with throwing out entire frames in video and simply re-interpolating them to rehydrate the video. Performance on such a task could be quantified with peak signal-to-noise ratio (PSNR), the usual metric for compression, instead of L2 norm.

Along the same vein, the model generalizes to interpolating not just the center frame but any intermediate frame in between the two inputs. Formally, we can synthesize any image a fraction α past Image 1, where image $I_{\alpha=0}$ is Image 1 and $I_{\alpha=1}$ is Image 3. In our model, we restrict ourselves to only calculating $I_{0.5}$, but we could find any I_{α}, as formalized in [3].

We would also like to see the effects of applying a GAN to the final output image. Generative adversarial networks have shown promise in generating realistic images, and we believe a GAN trained on real images might be able to sharpen our output images and make them more realistic-looking. Ideally, to simplify the design and to maintain gradient flow, this GAN would be pretrained and merely applied as a postprocessing step to the images output by our proposed architecture.

Finally, we would like to see how our model generalizes to different datasets. KITTI, though a large dataset, consists entirely of outdoor images taken by a car. If trained on this dataset, it is not clear whether the network could interpolate images taken indoors as well or images where the camera rotated / moved in a less linear fashion than the car in question did.

Github Repository

`https://github.com/Gamrix/cs231n_proj`

References

[1] Y. Furukawa and J. Ponce. Accurate, dense, and robust multi-view stereopsis. 2010.

[2] B. Girod, C.-L. Chang, P. Ramanathan, and X. Zhu. Light field compression using disparity-compensated lifting. 2003.

[3] D. Ji, J. Kwon, M. McFarland, and S. Savarese. Deep view morphing. 2017.

[4] Z. Liu, R. Yeh, X. Tang, Y. Liu, and A. Agarwala. Video frame synthesis using deep voxel flow. *CoRR*, abs/1702.02463, 2017.

[5] M. Magnor and B. Girod. Data compression for light-field rendering. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(3):338–343, 2000.

[6] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Multi-view 3d models from single images with a convolutional network. 2016.

[7] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros. View synthesis by appearance flow. 2017.

Appendix: Sample Generated Images

The left and right images are the two inputs, and the third is the generated image.

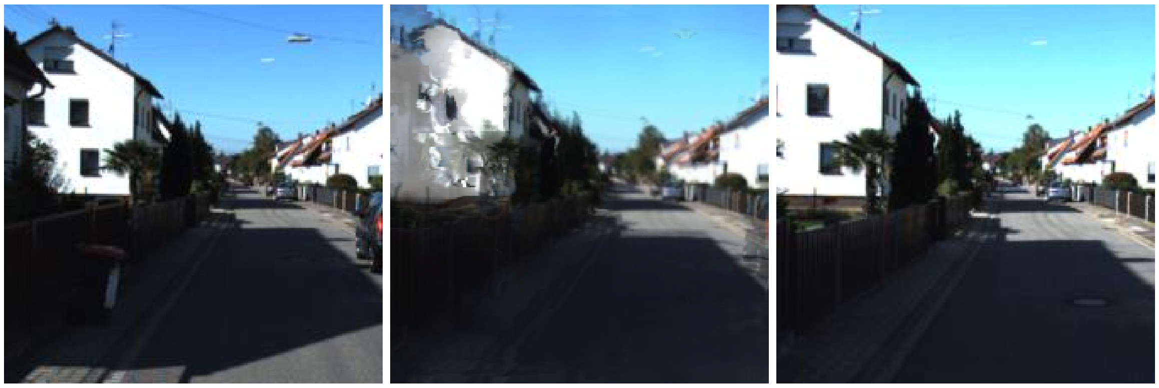

Figure 15: A Reasonable Generated Image

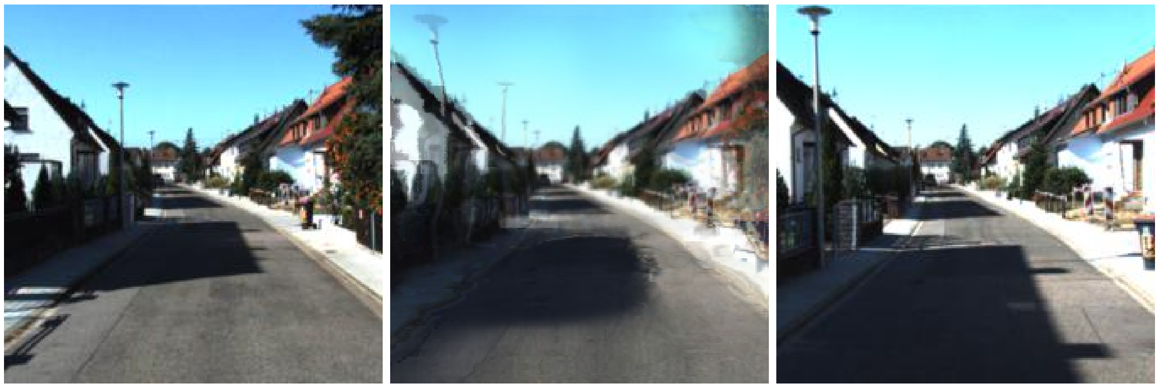

Figure 16: Another Reasonable Generated Image

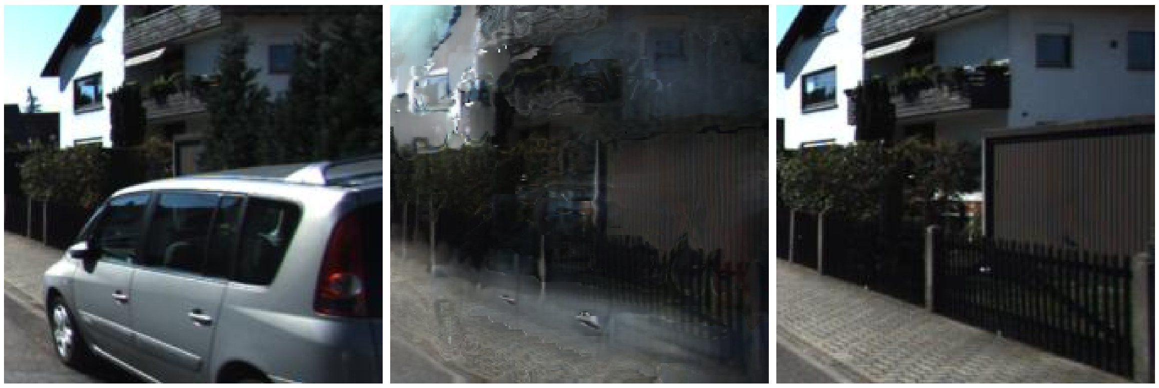
Figure 17: A Somewhat Bad Generated Image

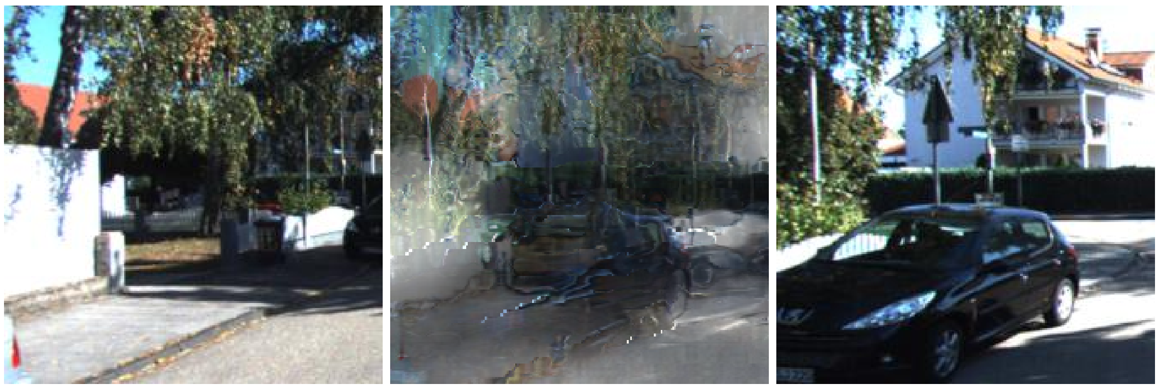
Figure 18: An (Unrepresentative) Particularly Bad Generated Image