Learning video processing by example

Antonio Haro, Irfan Essa
GVU Center / College of Computing
Georgia Institute of Technology

Abstract

We present an algorithm that approximates the output of an arbitrary video processing algorithm based on a pair of input and output exemplars. Our algorithm relies on learning the mapping between the input and output exemplars to model the processing that has taken place. We approximate the processing by observing that pixel neighborhoods similar in appearance and motion to those in the exemplar input should result in neighborhoods similar to the exemplar output. Since there are not many pixel neighborhoods in the exemplars, we use techniques from texture synthesis to generalize the output of neighborhoods not observed in the exemplars. The same algorithm is used to learn such processing as motion blur, color correction, and painting.

1. Introduction and related work

In this paper, we present a framework to represent arbitrary video processing so that it can be learned by example. We model video processing as a piece-wise constant high-dimensional function and utilize techniques from texture synthesis to address the curse of dimensionality. We examine a single example of video before and after it is processed and predict how an entirely different video sequence would look if processed with the same algorithm with no a priori knowledge of the algorithm. This is beneficial if the video processing is something that would be very time consuming (*e.g.* applying motion blur to an already rendered computer animation) or if the output is non-obvious (*e.g.* applying a painter's style to a video sequence).

We address several problems in this paper: (1) creation of feature vectors that encode the visual and dynamic properties of video regions, (2) use of a distance metric that ensures spatial and temporal continuity, and (3) use of this distance metric in an algorithm to learn video processing by example.

There is much related work to ours, particularly in synthesizing images and video. Image-based texture synthesis methods primarily model textures as either probability distributions ([3], [6], [8]) that are sampled to create new texture, or as Markov random fields (MRF). In the MRF-based approaches ([15], [1]), sets of feature vectors composed of pixel neighborhoods are matched against to synthesize new texture.

Related work from video synthesis includes ([14], [13]) where a cost function is optimized and Q-learning is used to create looped and controllable video sequences. Hidden markov models have also been used to segment an audio track from a video into small sequences that are then morphed to match a new vocal track [4].

The works most related to ours are ([10], [7]). There, given a pair of example images, the mapping between the two example images is learned and applied to a new image. We are interested in the temporal version of this problem: learning the mapping between two example sequences and applying the mapping to a new sequence.

Our algorithm is motivated by thinking of all video processing algorithms as high-dimensional functions that take as input arbitrary video sequences and output the processed versions of those sequences. We model changes to pixel neighborhoods as being localized in space and time. We use the color and motion information of pixel neighborhoods to match similar changes to the neighborhoods in modelling the processing function.

Of course, since we do not have all possible patches and do not have infinite streams of video, we are extremely burdened by the curse of dimensionality since we have an extremely sparse sampling of the high-dimensional feature space. We can get around it by ensuring that our feature vectors are close to each other in this high-dimensional space and by using texture synthesis techniques to make educated guesses about the high-dimensional function in areas where we have no information.

2. Learning from example

Our learning algorithm is based on building and matching feature vectors from video sequences (Figure 1). The main idea is that pixels with similar neighborhoods in the input sequence are mapped to pixels with a similar neigh-

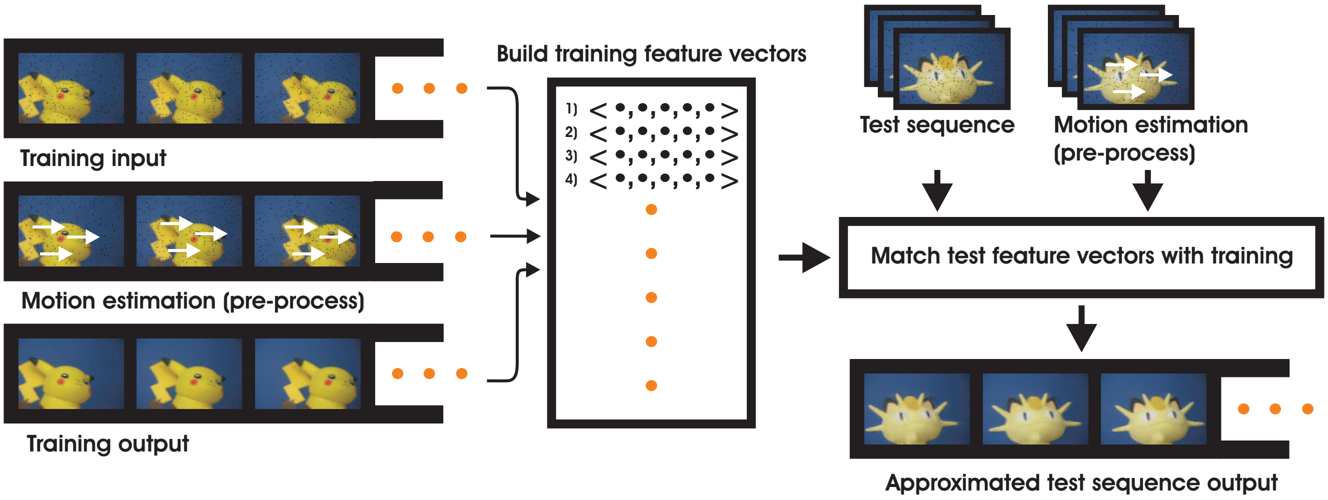

Figure 1. *Overview of our algorithm*

borhood in the output. However, we have an extremely sparse representation of the space since we have very few vectors of high dimensionality (*i.e.* we only have as many vectors as we do pixels in all frames of the input training sequence).

2.1. Building feature vectors

We construct feature vectors the same way for our training sequences and test sequences. Since our pixel feature vectors also encode the effects of processing on the pixel, we approximate a test sequence's output as being the same as its input initially (since we do not know the output) and update the output as we match (Section 2.2). No special handling is needed to build feature vectors for the training data since the output is known. We use the YIQ color space and match color histograms as is done in [10]. In general, we process all color channels, though for the motion blur and painting results, we only processed the Y channel, since that saved processing time.

Once color conversion has been done, we compute optical flow [2] of the entire sequence and construct Gaussian pyramids of the flow [5] . Next, we construct Gaussian pyramids of each frame in the sequence. Then, for a pixel at (i, j) in level l of frame n, we construct a feature vector comprised of:

1. the 3x3 neighborhood of pixel $(i/2, j/2)$ in level $l+1$, frame n of the input pyramid
2. half of the 5x5 neighborhood of pixel (i, j) in level l, frame n of the input pyramid
3. the 3x3 neighborhood of pixel $(i/2, j/2)$ in level $l+1$, frame n of the output pyramid
4. half of the 5x5 neighborhood of pixel (i, j) in level l, frame n of the output pyramid
5. the 3x3 neighborhood of pixel $(i'/2, j'/2)$ in level $l+1$, frame $n+1$ of the input pyramid
6. the 5x5 neighborhood of pixel (i', j') in level l, frame $n+1$ of the input pyramid
7. the 3x3 neighborhood of pixel $(i/2, j/2)$ in level $l+1$, frame n of the flow pyramid
8. half of the 5x5 neighborhood of pixel (i, j) in level l, frame n of the flow pyramid

where $i' = i + \Delta i$ and $j' = j + \Delta j$, and $(\Delta i, \Delta j)$ are the x, y components of the optical flow for (i, j).

This feature vector is similar to the one used in [10], but differs in that it also encodes the temporal properties of the pixel. Components 5 and 6 of the vector act as a "look ahead" to the next frame to encode how the pixel's neighborhood changes in appearance. Using the optical flow for these components effectively stabilizes the pixel's neighborhood between frames n and $n+1$. Components 7 and 8 are the flow itself, which aid in learning effects that are more motion rather than texture dependent, such as motion blur.

Gaussian pyramids are used to improve the quality of matches. At the coarser levels of the pyramid, large features in the video affect matching more than small features. Matching at all levels, from coarse to fine, ensures that small and large scale features affect the matching equally. We use 3x3 neighborhoods for features from the previous pyramid level and 5x5 neighborhoods for features from the current pyramid level because we found that those were the smallest sizes that would still yield good results. This agrees with the neighborhood sizes for the feature vectors in [10] for images.

2.2. Matching feature vectors

Once the training feature vectors have been constructed, we can approximate the test sequence's output. We march through the test sequence one pixel at a time in scanline order for each frame in the sequence in temporal order. For each pixel in the test sequence, we build its feature vector, match it with the "best" training feature vector (described below), and then copy the intensity of the pixel that best matches from the training output sequence to the approximated output sequence at the location of the pixel we are currently processing for the current color channel we are processing.

This matching algorithm is the same as the one used in [10]. Since our feature vectors encode changes due to dynamics very succinctly, the same matching algorithm results in temporally coherent results. The crucial insight in their work is that the "best" match is *not* always the vector with the smallest L2 distance, but could be one that matches the previously matched pixels better (according to a threshold, κ), thus ensuring spatial coherence. For processing video, however, we usually set κ extremely low (often to 0) since values that are too high yield results that are coherent spatially, but extremely uncoherent temporally.

3. Results

We tried our algorithm on several data sets of different video processing to see how it would perform. Our video processing ranged from being strictly spatial (*e.g.* color correction) to being both spatial and temporal (*e.g.* motion blur). Our algorithm was able to produce videos that looked like plausible results of the observed processing. Videos of our results can be found on: *http://www.cc.gatech.edu/cpl/projects/learningvideoproc/*.

Color correction We took a 22 frame sequence and changed its colors in Adobe Premiere. The original and color modified sequence were then provided as training to our algorithm before giving it a new sequence to process. Figure 2 shows the results. Color correction is a toy problem since it is purely a pixel based operation. The color tone is slightly off, but it is faithful to the tone of the training processed footage, which is what is expected.

Noise removal We added random salt and pepper noise to a 15 frame sequence of a toy being slid left and right. We used the noisy sequence as the training input and the original sequence as the training output. Then, as a test sequence, we used a different sequence that had noise added to it as well. Our algorithm removed the salt and pepper noise and created a temporally coherent video (Figure 3).

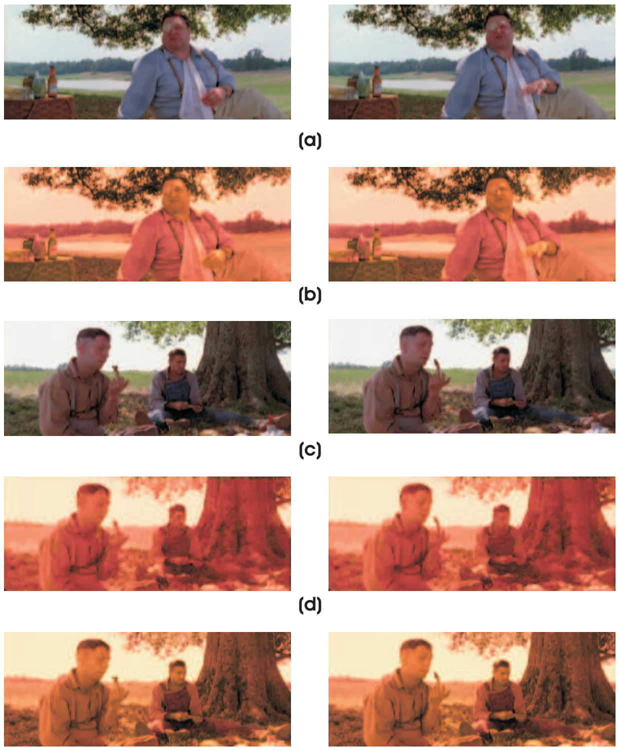

Figure 2. *Color correction: a) & b) training input/output, c) test sequence, d) approximated output, e) ground truth*

Motion blur from CG We took a 3D animation of a person kicking. We applied 3D motion blur in Maya, a 3D modelling and animation program, to create a motion blurred version of the animation. We then took the same animation data and applied it to a model with different geometry and texture and created a video sequence that we used as our test sequence (Figure 4). Our algorithm produces some motion blur (evident on the legs), but does not blur as much as Maya does. This is because our 3x3 and 5x5 neighborhoods are too small to capture such large amounts of blur. However, our algorithm was still able to learn to blur despite the fact that the effect relied on a priori 3D knowledge and was too large for its neighborhoods.

Non-photorealistic rendering of video Rendering video non-photorealistically is an interesting problem because the mapping between a painting's *style* to video is non-obvious. Prior work has been based on "pushing" strokes around based on optical flow calculated for the sequence. Temporal coherence is achieved by using optical flow and heuristics to constrain brush strokes ([12],[11]) or by "relaxing" the brush strokes through a minimization [9]. The main problem with these techniques is that the style that can be used is dependent on the implementation. We are able to render video based directly on existing art. Figure 5 shows a Van Gogh painting's brush strokes used on a video sequence, resulting in temporally coherent brush strokes that come from the painting (treated as a video sequence).

4. Conclusions and future work

We have presented a novel algorithm that can learn various types of video processing automatically with no a priori knowledge. Our algorithm is based on approximating complicated high-dimensional functions as collections of feature vectors. Our feature vectors encode the spatial and temporal properties of pixels and are then matched in a sparse feature space.

There are several improvements that can be made to the algorithm. The dynamics are currently estimated by using optical flow, but we would like to try better motion compensation/stabilization techniques. Our feature vectors are very high dimensional, so we are interested in seeing how the dimensionality can be reduced while retaining visual fidelity. The algorithm currently creates frames a pixel at a time, which is slow. We are interested in image patch-copying techniques so that we can increase its speed and possibly parallelize it.

References

[1] M. Ashikhmin. Synthesizing natural textures. *2001 ACM Symposium on Interactive 3D Graphics*, pages 217–226, March 2001.
[2] M. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow-fields. *CVIU*, 63(1):75–104, January 1996.
[3] J. S. D. Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. *Proceedings of SIGGRAPH 97*, pages 361–368, August 1997.
[4] C. Bregler, M. Covell, and M. Slaney. Video rewrite: Driving visual speech with audio. *Proceedings of SIGGRAPH 97*, pages 353–360, August 1997.
[5] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. In *IEEE Transactions on Communications*, 1983.
[6] A. Efros and T. Leung. Texture synthesis by non-parametric sampling. In *ICCV99*, pages 1033–1038, 1999.
[7] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001*, pages 341–346, August 2001.
[8] D. J. Heeger and J. R. Bergen. Pyramid-based texture analysis/synthesis. *Proceedings of SIGGRAPH 95*, pages 229–238, August 1995.
[9] A. Hertzmann. Paint by relaxation. *Proceedings of Computer Graphics International*, pages 47–54, July 2001.
[10] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. *Proceedings of SIGGRAPH 2001*, pages 327–340, August 2001.
[11] A. Hertzmann and K. Perlin. Painterly rendering for video and interaction. *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*, pages 7–12, June 2000.
[12] P. Litwinowicz. Processing images and video for an impressionist effect. *Proceedings of SIGGRAPH 97*, pages 407–414, August 1997.
[13] A. Schödl and I. Essa. Machine learning for video-based rendering. In *Advances in Neural Information Processing Systems*, volume 13, 2001.
[14] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa. Video textures. *Proceedings of SIGGRAPH 2000*, pages 489–498, July 2000.
[15] L. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. *Proceedings of SIGGRAPH 2000*, pages 479–488, July 2000.

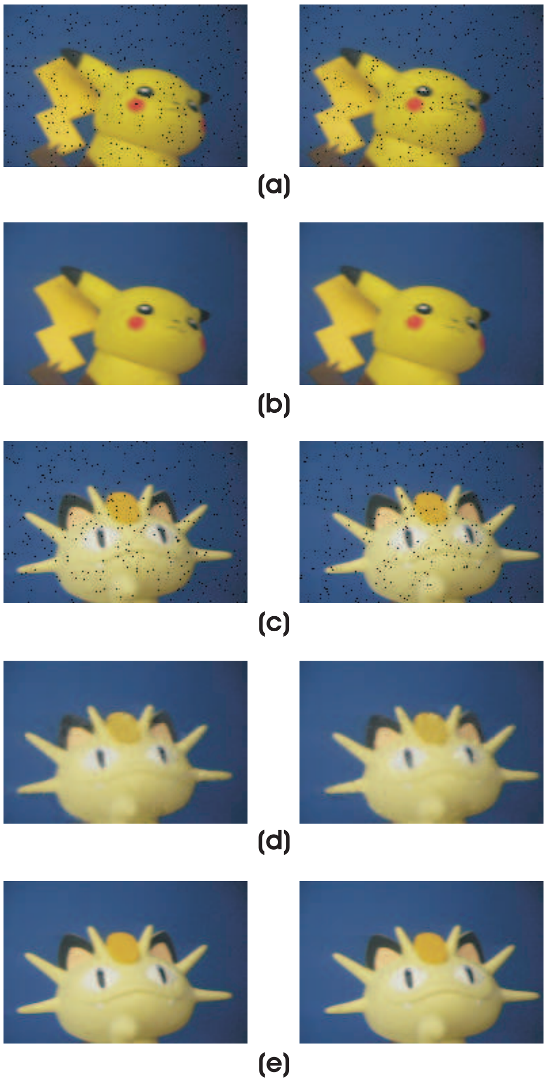

Figure 3. *Noise removal: a) & b) training input/output, c) test sequence, d) approximated output, e) ground truth*

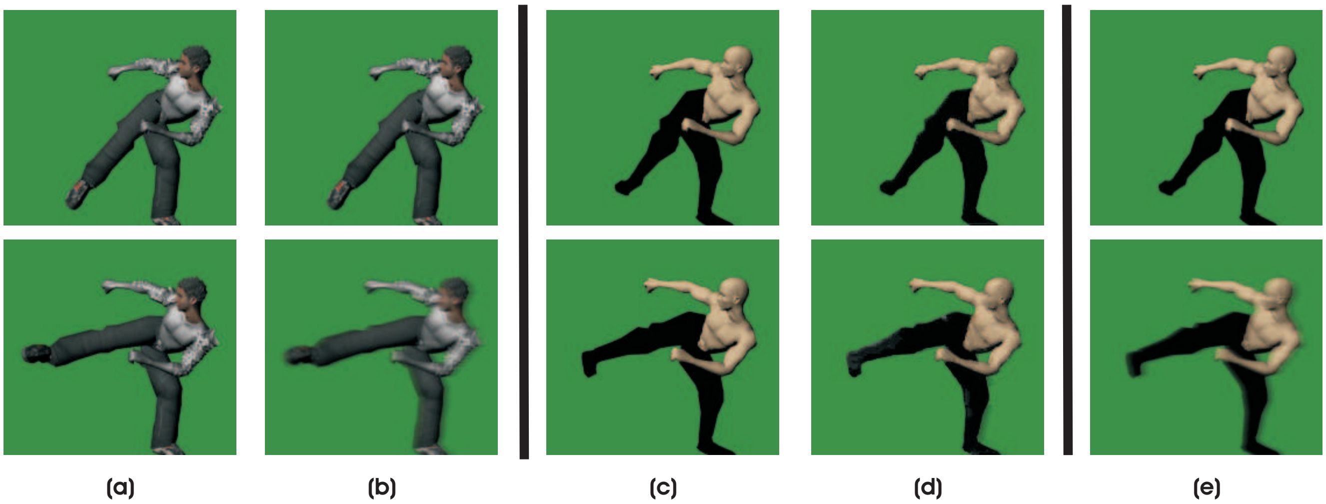

Figure 4. *Motion blur: a) & b) training input/output, c) test sequence, d) approximated output, e) ground truth*

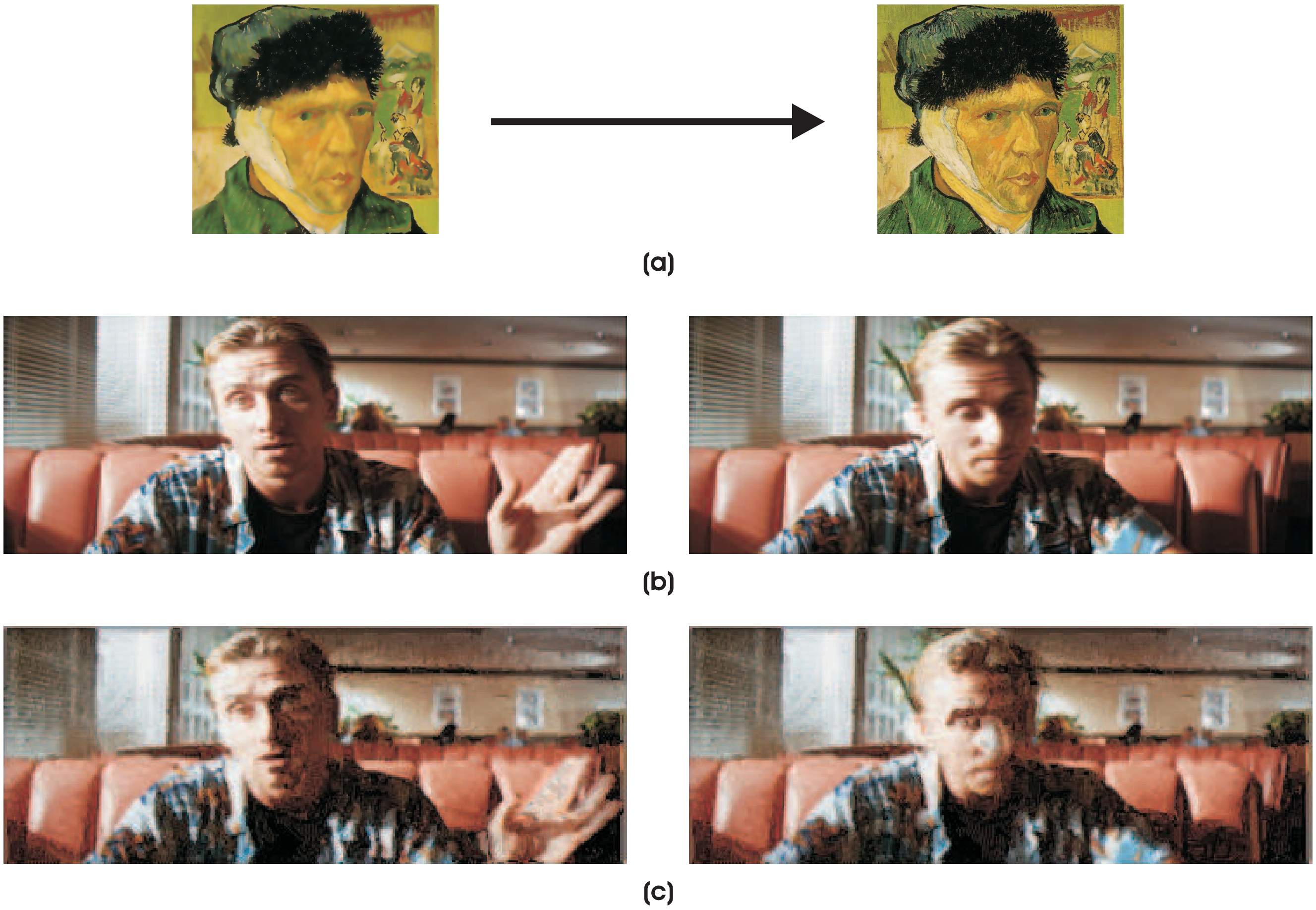

Figure 5. *Painting style: a) training input/output (video), b) test sequence, c) approximated output*