Quantitative Analysis of Blurring Effects in Autoencoder Architectures for Video Interpolation

Lucas McCabe
Johns Hopkins University
lmccabe2@jhu.edu

William Glad
Johns Hopkins University
wglad1@jhu.edu

Guillermo Pinczuk
Johns Hopkins University
gpinczu1@jhu.edu

Abstract

We present a novel framework for quantitatively evaluating the blurring effects of autoencoder models for video interpolation. We demonstrate the applicability of this framework by comparing four autoencoder topologies trained using three commonly-used loss functions. We find that some loss function-model architecture combinations are preferable to others when attempting to account for blurring effects in generated images. Finally, we recommend that future related work aim to incorporate our measurement of Relative Blur Factor into its loss functions to adequately accommodate for the blurring effects that interfere with the credibility of interpolated video frames.

1. Executive Summary

Artificial Neural Networks represent a bottom-up attempt to model macro-level cognitive processes and reasoning by mimicking micro-level processes at the neuronal level. In particular, artificial neurons attempt to model their neurophysiological counterparts through the creation of individual nodes or perceptrons that feature an assortment of input values, weights and activity and activation functions. The interactions of these features within a perceptron represent a computational effort to mimic individual neuronal activity, and when located in a layered interacting network, a collection of such perceptrons can be employed to model macro-level cognitive processes at the level of the brain.

Video interpolation is the process of generating an intermediate frame or set of frames to be placed in between original exiting frames, in order to improve the clarity of movement that occurs in the video. There are three primary motivations for this interest. First, video is often taken at lower-than-ideal frame rates, typically for aesthetic or technical reasons. For example, Hollywood film is primarily shot at 24 frames per second (FPS) and capsule video endoscopy is often taken at 5 FPS, whereas motion capture tends to look most fluid at 60 FPS [1] [2]. If substantial movement is displayed in 24 FPS film, it can appear blurry, so frame rate upscaling via video interpolation can make video filmed at 24 FPS appear at 48 FPS or higher. Second, slow-motion replay is frequently required for sports critique and law enforcement, and this reduces the perceived frame rate of the video [3]. For instance, a 60 FPS video replayed at 50% of its original speed will appear to be a 30 FPS video that is twice as long as its original. Finally, storing or transferring video is both technically and financially expensive.

The ability to save or stream only half the original frames and interpolate the ones in between as necessary could reduce costs while improving user experience. The primary difficulty in the task of video interpolation is one of fidelity. Simply averaging the pixel intensity of surrounding frames fails to produce convincing intermediate frames, and autoencoder implementations are often criticized for generating excessively blurry images [4]. This tendency to generate unclear images can be explained by loss functions that do not adequately penalize blurriness, but existing comparisons of loss functions toward this end tend to be subjective and opinion-based [5].

We propose a method for quantitatively evaluating the blurring effects of neural networks for video interpolation. We implement this method to compare the performances of four autoencoder architectures trained using five commonly-used loss functions, and we conclude with analysis and recommendations for future work.

2. Design, Architecture, and Data Processing

Autoencoder models aim to learn latent, lower-dimensional representations of data. For computer vision tasks, these models iteratively encode images or pairs of images into a latent space before retrieving them in the same or another output space. We compare several autoencoder network topologies and training approaches to evaluate the blurriness of interpolated images.

Our data processing pipeline consists of six stages:

1. Download a video from YouTube.
2. Extract each frame from the downloaded video.
3. Convert each frame to greyscale coloring.
4. Proportionally resize each frame to 200x100, cropping as necessary.
5. Normalize pixel intensity values to the range [0,1].
6. Pseudo-randomly assign data along a 70%/30% training/testing split.

For the final run of our models, we use a 12-minute scene from the movie *Bambi* [6]. The data extraction phase produces over 12,000 frames from this video. We narrow our focus to a single video, because learning to encode and decode multiple videos, particularly ones with different visual styles, would require more complex methods.

We implement four similarly-structured autoencoder models. Three of these models are multilayer autoencoder architectures, consisting of an input layer; three fully-connected encoder layers of sizes 1024, 512, and 256, respectively; a code layer of size 128; three fully-connected decoder layers of sizes 256, 512, and 1024, respectively; and an output layer.

The Forward, Backward, and Bi-Directional Autoencoder models share this common architecture. The Forward Autoencoder is trained to predict frames, given their immediately previous frames. The Backward Autoencoder is trained to predict frames, given their immediately subsequent frames. The Bi-Directional Autoencoder is trained to predict frames, given their immediately previous and immediately subsequent frames.

The fourth model we implement is a Convolutional Autoencoder, trained to predict frames their immediately previous frames. The key difference between the Convolutional Autoencoder and the Forward Autoencoder is that the Convolutional Autoencoder uses convolutional layers and pooling layers for the encoder and upsampling layers for the decoder.

The Convolutional Autoencoder has two convolutional layers with 16 filters apiece and two corresponding pooling and upsampling layers in the encoder and decoder modules, respectively. The encoder's convolutional layers aim to learn image patterns and lower-dimensional representations of images, while the decoder translates that lower-dimensional image representation into the higher dimensional space.

We include all associated code, which generates approximately 90 GB of output files and data, in the Git repository found here.

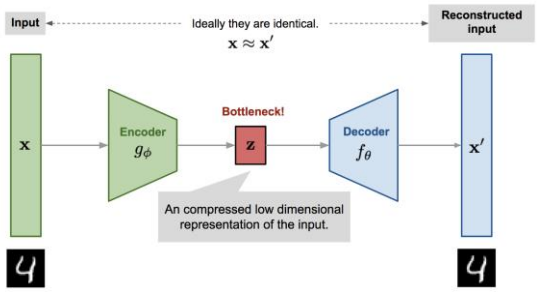


Figure 1: Visualization of an autoencoder model (Source: Lilian Weng) [7].

3. Computational Performance

Across all models implemented, perceptrons use rectified linear unit activation functions, with the exception of those nodes in the output layer, which use sigmoid activation functions. Due to time constraints, we limit training of each model to 10 epochs. Forward, Backward, and Bi-Directional Autoencoder models use a batch size of 64, while the Convolutional Autoencoder employs a batch size of 1.

We train our four autoencoder architectures using three different loss functions:

3.1 Mean Squared Error (MSE)

Mean Squared Error involves calculating the sum of squared residuals and then dividing by the sample size, as represented in the loss equation:

$$\mathrm{L} = \frac{1}{n}\sum_{1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

whereby $y^{(i)}$ corresponds to the target value, which in the current context represents the quantized value of pixelized images we wish to replicate, while $\hat{y}^{(i)}$ represents model output [8].

3.2 Mean Squared Logarithmic Error (MSLE)

The Mean Squared Logarithmic Error is a variant of MSE, as represented in the loss equation:

$$\mathrm{L} = \frac{1}{n}\sum_{i=1}^{n}(log(y^{(i)}+1) - log(\hat{y}^{(i)}+1))^2$$

In particular, both the target and model output values are logged prior to calculating the residual, with the value 1 added to both to prevent the possibility of taking the logarithm of 0. The effect of taking logs is to dampen the impact on the loss function of large differences between the target and predicted values. Additionally, by taking logs, the residual corresponds to an approximate percentage difference between target and predicted values [8].

3.3 Cosine Proximity (CP)

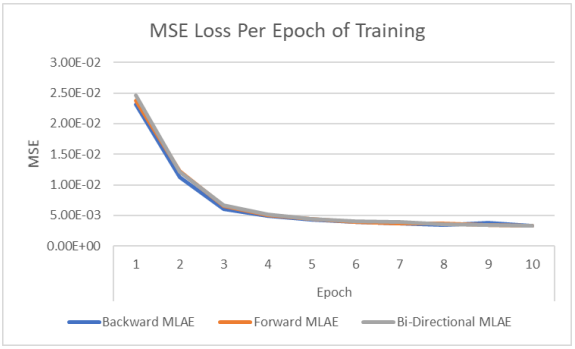
Cosine Proximity is an array-level nearness function represented in the loss equation:

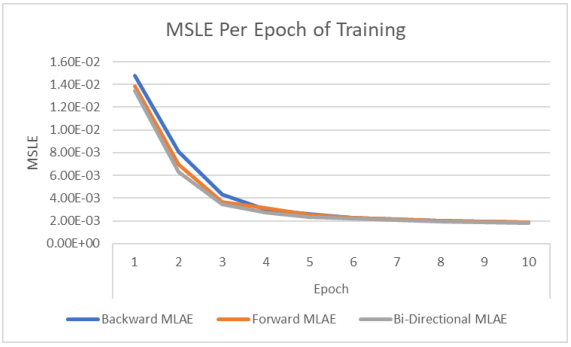
$$\mathrm{L} = -\frac{\mathbf{y}\cdot\hat{\mathbf{y}}}{\|\mathbf{y}\|_2\cdot\|\hat{\mathbf{y}}\|_2} = -\frac{\sum_1^n y^{(i)}\cdot\hat{y}^{(i)}}{\sqrt{\sum_1^n (y^{(i)})^2}\cdot\sqrt{\sum_1^n (\hat{y}^{(i)})^2}}$$

Note that this equation can be rewritten as $\mathrm{L} = -\cos\theta$, where θ is the angle between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ in R^n. Now suppose $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are located so that both originate from the origin. As $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ become more similar (through model training), $\cos\theta$ increases and L decreases, and in the limit approaches -1. Hence, minimization of L is achieved by training model weights to maximize the similarity between target and predicted (vector) values [8].

Unsurprisingly, all models arrived at low loss after just a few epochs of training (Figure 2). The non-convolutional models generated strikingly similar loss curves for each loss function, which we suggest is due to their structural similarity.

Training loss of the convolutional models is not shown in Figure 2, because they update weights markedly more frequently than the other models, resulting in distorted loss curves when plotted per-epoch. That said, they achieve lower loss than the other models in every epoch.





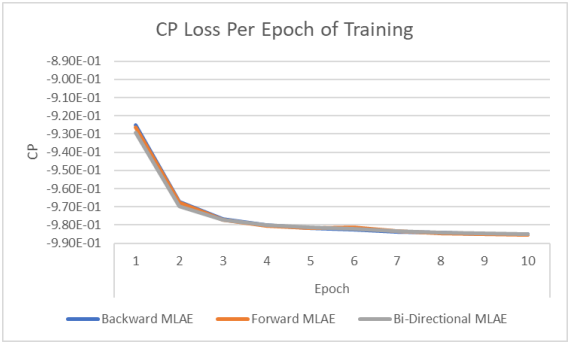


Figure 2: Mean Squared Error, Mean Squared Logarithmic Error, and Cosine Proximity of the Backward, Forward, and Bi-Directional Multi-layer Autoencoder models during training.

4. Analysis and Evaluation

The Laplacian operator is a spatial filter frequently used for edge detection. It is taken as the second derivative of an image's pixel intensities. In other words – for an image whose pixel intensities are given by I(x,y) – the Laplacian is represented by the function:

$$Lap(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Taking the Laplacian of an image returns a same-dimensional image whose edges are exaggerated several-fold and whose non-edge regions are markedly diminished. We observe that, by taking the pixel intensity variance of this resultant image, we arrive at a proxy for image clarity, since sharply-defined edges have high local variances in intensity. This result has previously been reported by Pech-Pacheco et al. and Bansal et al. [9] [10]. Expanding upon their findings, we introduce two relevant variables: Blur Factor and Relative Blur Factor.

4.1 Blur Factor

We define the blurriness of an image P by its blur factor (BF), represented by the function:

$$BF(P) = \frac{1}{var(Lap(P))}$$

whereby var(·) is the variance of a matrix and Lap(·) is the Laplacian of an image. The Blur Factor of an image is high when the image is blurry and low when it is sharp.

We compute the Blur Factor of every generated image from every model in its post-training testing (Figure 3). We make three observations from this analysis:

1. Those non-convolutional autoencoder models trained using the Cosine Proximity loss function generated exceptionally sharp images during testing.

2. The convolution-based models tended to generate particularly blurry images, despite achieving low loss during training.

3. In contrast with Observation 1, the convolutional autoencoder generated blurrier images when trained using Cosine Proximity loss.

Observation 2 demonstrates the risks associated with loss function selection for video interpolation. The most frequently-used loss functions do not directly penalize excessive blurring, and thus models that achieve low loss should not be necessarily expected to generate sharp images.

4.2 Relative Blur Factor

It is worth noting that the goal of video interpolation is indiscernibility, not necessarily clarity. Blurriness may contribute to interpolated frames being irritatingly detectable, but a blurry frame may be appropriate for the moment (for example, in instances of particularly rapid movement).

For this reason, we calculate Relative Blur Factor (RBF), which compares the Blur Factor of a predicted frame to that of its ground truth:

$$RBF(P_{predicted}, P_{true}) = \frac{BF(P_{predicted})}{BF(P_{true})} - 1$$

We subtract 1 from the ratio of Blur Factors so that Relative Blur Factor can provide a notion the percentage to which the predicted image is more or less blurry than its target.

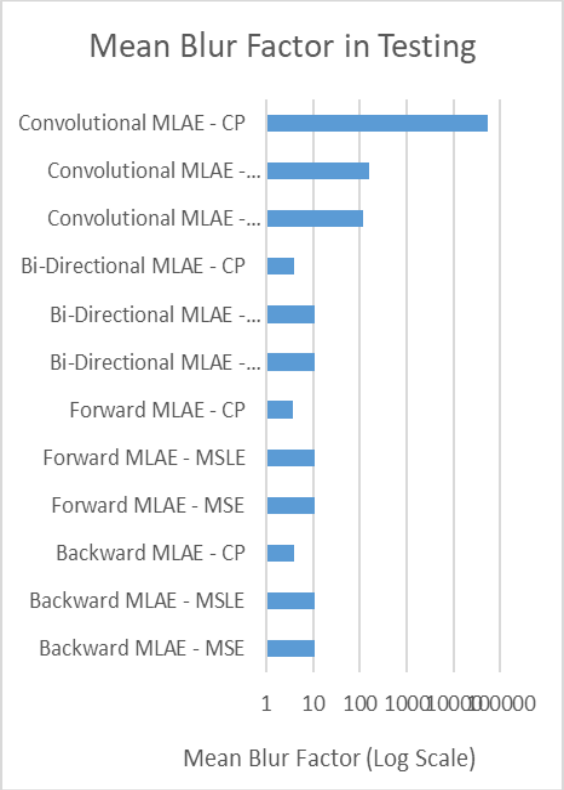


Figure 3: Mean Blur Factor in Testing. A log scale is used to accommodate for the excess blurriness of the convolution-based models.

Blur Factor (Predictions)		
	Mean	STD
Backward MLAE - MSE	10.9949	17.3854
Backward MLAE - MSLE	10.8436	15.2972
Backward MLAE - CP	4.00839	2.95358
Forward MLAE - MSE	10.7538	17.594
Forward MLAE - MSLE	10.9283	18.8961
Forward MLAE - CP	3.77434	2.25595
Bi-Directional MLAE - MSE	10.7391	15.2558
Bi-Directional MLAE - MSLE	10.7269	22.6795
Bi-Directional MLAE - CP	3.94646	2.46125
Convolutional MLAE - MSE	117.685	56.5844
Convolutional MLAE - MSLE	154.445	71.3409
Convolutional MLAE - CP	52638.3	22244.4

Table 1: Blur Factor in Testing. Numerical values are shown for detail, to accompany Figure 3.

From the definition of Blur Factor, we see that Relative Blur Factor is equivalent to

$$RBF(P_{predicted}, P_{true}) = \frac{var(Lap(P_{true}))}{var(Lap(P_{predicted}))} - 1$$

We compute the Relative Blur Factor of every generated image from every model in its post-training testing (Figure 4). We make three observations from this analysis:

1. As noted from their associated Blur Factors, the convolution-based models produced extraordinarily blurry images.
2. Non-convolutional models trained using Cosine Proximity averaged negative Relative Blur Factor, indicating that these models generated predicted images that were actually sharper than their ground truth counterparts.

3. Although both negative and positive Relative Blur Factors can be problematic for indiscernibility, the Relative Blur Factors of the non-convolutional Cosine Proximity models generated predicted images that were the closest to being equally blurry to their targets.

5. Summary and Conclusion

Autoencoders are frequently used for the purpose of video interpolation, which aims to recreate intermediate frames in video. Unfortunately, autoencoders tend to generate blurry images, and this effect interferes with the believability of interpolated frames. This tendency is, in part, due to the selection of loss functions for evaluating atuoencoder performance [11]. Extensive work has been completed analyzing loss functions for their tendency to reduce generated image blurring, but few do so without relying on subjective human-level perceptive input.

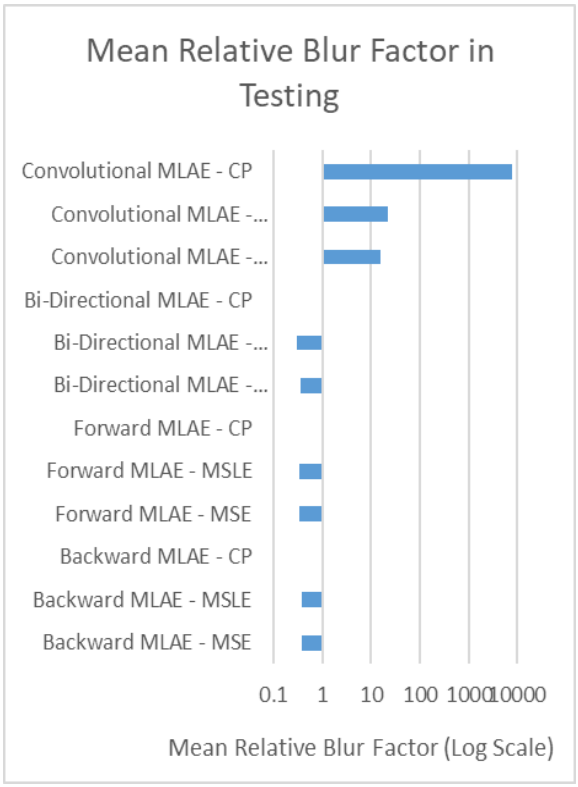


Figure 4: Mean Relative Blur Factor in Testing. A log scale is used to accommodate for the excess blurriness of the convolution-based models. The RBFs of the non-convolutional Cosine Proximity models do not appear because their negative values cannot be displayed with a log scale.

Relative Blur Factor		
	Mean	STD
Backward MLAE - MSE	0.39392	0.69326
Backward MLAE - MSLE	0.37855	0.63957
Backward MLAE - CP	-0.43245	0.25595
Forward MLAE - MSE	0.33912	0.72271
Forward MLAE - MSLE	0.34999	0.72292
Forward MLAE - CP	-0.45711	0.1971
Bi-Directional MLAE - MSE	0.36767	0.67801
Bi-Directional MLAE - MSLE	0.30246	0.85977
Bi-Directional MLAE - CP	-0.42955	0.26931
Convolutional MLAE - MSE	16.1837	3.05879
Convolutional MLAE - MSLE	21.7373	4.26764
Convolutional MLAE - CP	7820.67	1421.55

Table 2: Relative Blur Factor in Testing. Numerical values are shown for detail, to accompany Figure 4.

It is clear that many common loss functions used for video interpolation inadequately account for blurring effects in generated outputs. Minimizing blurring in generated images, however, is not sufficient, because the images a model aims to successfully interpolate may be deliberately blurry.

Toward this end, we present a quantitative framework for evaluating the blurriness of images and the degree to which the blurriness of an interpolated image is accounted for by the blurriness of its associated ground truth image. We then leverage this framework comparatively. We implement twelve machine learning models – four different autoencoder architectures, each trained using three different commonly-used loss functions. We then lean on our developed notions of Blur Factor and Relative Blur Factor to draw insights from each model's performance post-training.

We observe that our convolutional models, despite having achieved the lowest loss values during training, generated the most excessively blurry images during testing. We point to this as a significant motivator for selecting loss functions that appropriately penalize undesirable outcomes.

We conclude with recommendations for future work. First, we recommend that array-level nearness loss functions such as Cosine Proximity be used when developing simple multilayer autoencoders for video interpolation, as thisloss function may reduce the unexplained bluriness in interpolated frames. Next, we advize that future work encorporate blur adjustment, such as Relative Blur Factor into loss functions themselves, in order to more appropriately accommodate for generated blurring effects. Finally, we acknowledge that Relative Blur Factor does encounter situations where it is insufficient.

For instance, it is conceivably possible for two images to have identical Blur Factors, despite being blurry or sharp in completely different locations. In this context, the Relative Blur Factor would suggest that the images have nearly the same amount of blurring. As a result, Relative Blur Factor could not be used to correctly identify that the differences in blurring would appear jarring or unbelievable.

The easy computability of Relative Blur Factor motivaties its appeal (not unlike similar useful but potentially imprecise metrics, such as R^2). That said, more robust analysis of blur effects could be accomplished by computing blur features repeatedly at a local level, rather than evaluating the image at once, as a whole.

6. Citations

[1] "Hacking Film: Why 24 Frames Per Second?" Film Independent. April 10, 2018. https://www.filmindependent.org/blog/hacking-film-24-frames-per-second.

[2] Mohammed, Ahmed, Ivar Farup, Sule Yildirim, Marius Pedersen, and Øistein Hovde. "Variational Approach for Capsule Video Frame Interpolation." *EURASIP Journal on Image and Video Processing* 2018, no. 1 (2018). doi:10.1186/s13640-018-0267-9.

[3] Slow Motion Replays. https://www.mediacollege.com/video/production/vt/slow-motion-replay.html.

[4] Zhao, Shengjia, Jiaming Song, and Stefano Ermon. "Towards a Deeper Understanding of Variational Autoencoding Models." *ArXiv*, February 28, 2017. https://arxiv.org/pdf/1702.08658.pdf.

[5] Zhao, Hang, Orazio Gallo, Iuri Frosio, and Jan Kautz. "Loss Functions for Image Restoration with Neural Networks." *ArXiv*. Accessed April 20, 2018. https://arxiv.org/pdf/1511.08861.pdf.

[6] Lewis, Scott. "Bambi Thumper Teaches Bambi to Walk and Speak HD." YouTube. November 24, 2017. https://www.youtube.com/watch?v=MaWnvvrngyQ

[7] Weng, Lilian. "From Autoencoder to Beta-VAE." Lilianweng.github.io. August 12, 2018. https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html.

[8] Keras-Team. "Keras-team/keras." GitHub. https://github.com/keras-team/keras/blob/master/keras/losses.py.

[9] Pech-Pacheco, J.l., G. Cristobal, J. Chamorro-Martinez, and J. Fernandez-Valdivia. "Diatom Autofocusing in Brightfield Microscopy: A Comparative Study." *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*. doi:10.1109/icpr.2000.903548.

[10] Pech-Pacheco, J.l., G. Cristobal, J. Chamorro-Martinez, and J. Fernandez-Valdivia. "Diatom Autofocusing in Brightfield Microscopy: A Comparative Study." *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*. doi:10.1109/icpr.2000.903548.

[11] Mathieu, Michael, Camille Couprie, and Yann LeCun. "Deep Multi-scale Video Prediction beyond Mean Square Error." November 17, 2015.