Deep Motion: A Convolutional Neural Network for Frame Interpolation

Neil Joshi, Duncan Woodbury

January 25, 2017

Abstract

Frame interpolation refers to a video processing technique where mathematical interpolation methods are applied to generate intermediate frames between given frames for applications like framerate conversion and slow-motion effects. To effectively perform frame interpolation, often much contextual information about the video is necessary. We propose a convolutional neural network (CNN) to eliminate the need for such contextual information, where the implementation of a network architecture that identifies relationships between objects in a set of frames enables the generation of approximate intermediary frames in an end-to-end fashion.

1 Introduction

The concept of interpolation is a foundational counterpart to the field of mathematics from the time of Euclid, used throughout scientific research fields as a basis for estimation in the absence of perfect information. The specific application we focus on in this paper is video frame interpolation, where by creating a function mapping between subsequent frames f_1 and f_2 sampled from a video, it becomes possible to accurately estimate the values of the image features in intermediary frames.

The naive approach to this is a linear interpolation between frames f_1 and f_2, corresponding to the raw pixel values of each image. If times t_1 and t_2 correspond to when each frame occurred, we have an intermediary frame f_i occurring at $t_i = \frac{t-t_1}{t_2-t_1}$ given by

$$f_i = (1 - t_i)f_1 + t_i f_2. \tag{1}$$

This most simplistic approach considers the raw pixel values as features for interpolation, and in most cases will not give a satisfactory result.

This leads to the consideration of objects and contextual semantics within each frame for the refined approach of motion interpolation. By performing object recognition, we can find the optical flow of the scene, given as a vector field that maps the displacement of each object between frames f_1 and f_2. The optical flow then gives us a set of mapping functions by which to estimate the intermediary behavior of each object. This can be thought of as a discretization of linear interpolation over the image, where we generate relative mappings corresponding to the motion of each object within the pair of frames, rather than a singular mapping corresponding to the linear change in pixel values between the frames. Using the information gained from optical flow, the image field is gradually deformed to account for perspective change that might occur at interpolated frames. Clearly this is a superior method to naive linear interpolation, however it entails the challenge of precisely identifying the relevant image features and creating a set of functions that accurately maps the motion pattern of each between the frames. This can easily require high precision and lengthy computations, requiring an undesirably high degree of estimation when done manually.

This naturally leads to the application of machine learning, whereby manual computations can be overlooked by implementation of a learning architecture capable of identifying contextual relationships within an image. The specific approach utilizes a convolutional neural network (CNN) architecture, a multi-layered feed-forward network which at each layer identifies and groups objects by applying image convolutions to identify specific features. For video frame interpolation, the CNN is given tuples of successive frames f_i, f_j, f_k, and is trained to approximate the intermediary frame f_j based off f_i and f_k.

To apply this technique in our study, a network architecture was created based off the `Keras` deep learning libraries for python and trained over categorized video data from the `YouTube-8m` and `KITTI` datasets. Training was enabled largely by use of the python `numpy` libraries, and a model was produced that generates low-resolution motion-interpolated frames with a significantly lower error as compared to the naive linear approach.

The outline of the rest of this paper is as follows: in Section 2 we give an overview of previous and related work in the field of machine learning; in Section 3 we give a detailed description of the network architecture and training procedures designed to enable frame interpolation; in Section 4 we describe the results, strengths and limitations of the model; in Section 5 we summarize our results and postulate extensions and future continuations of the work described herein.

1.1 System Description

The hardware used to enable the computation done in this study consists of a single NVidia 980 Ti GPU card.

2 Related Work

2.1 CNN for Scenery Prediction

Google produced a study coined "DeepStereo", which introduces a novel approach to scene interpolation using CNNs that approximates a projection of an image via a function mapping in terms of depth and color to the predicted scene [1]. The research team was able to attain remarkable accuracy for an arbitrary scene and an arbitrary desired projection, when using categorically dissimilar data for training and for testing.

2.2 U-Net

Ronnegerber, et. al from Cornell University proposed U-Net in 2015 [2], a recursive CNN (RCNN) based network architecture consisting of a multi-layer contracting network that successively implements upsampling layers, rather than pooling layers, to increase the resolution and resultant feature space of the output data. This enables a "fully convolutional" approach, where localization and convergence of the objective function (such as equation (3)) can occur much quicker and with many fewer training samples [2]. The work described in Section 3 uses these ideas to obtain more complete and accurate contextual information about input data to enable a more time-efficient training process.

2.3 Learning Image Matching by Simply Watching Video

Long, et. al [4] proposed an unsupervised learning based approach to the problem of image matching, which necessitated the need to train a CNN for frame interpolation. It is from this work that we derive many tweaks to our methodology, including the use of the Charbonnier loss function, and the use of the KITTI dataset for initial training. We go several steps forward, however by focusing attention solely on the frame interpolation CNN, and utilizing a large dataset of real-world videos to fine tune our network.

3 Methodology

The sections below detail our python-based implementation of a CNN for video frame interpolation enabled by the Keras and Tensorflow deep learning libraries. The objective function we seek to minimize is the Charbonnier error loss, given by

$$\rho(x) = \sqrt{\alpha^2 + \epsilon^2}. \tag{2}$$

In this case, α represents the difference between the true frame f and the approximated frame f_i, so we can more accurately write the objective function as

$$\min\left(\sqrt{\|f - f_i\|^2 + \epsilon^2}\right). \tag{3}$$

This function is advantageous over standard functions like the mean squared error loss in that the additional parameter ϵ lets us control the behavior of the function as the loss approaches zero. It is possible to choose ϵ in such a way that when the loss falls below a certain threshold, the function quickly grows, giving a pronounced signal that the model has converged within the acceptable given error range. To enable gradient descent-based optimization of (3), the Keras native Adam optimizer was used with default parameter settings in all training cases, with a learning rate $\alpha = 0.001$ for stochastic gradient descent (SGD) [3]. To help avoid getting stuck at local optima, the learning rate is decreased by a factor of ten when the training loss becomes stagnant.

3.1 Data Collection/Processing

Recent advancements in certain sub fields of machine learning have led to the creation of data libraries like `KITTI` and `YouTube-8m` (`8m`).

The `KITTI RAW` dataset contains many image sequences of driving footage, which is used to train self-driving car software. This dataset is beneficial to us in that it contains sequences of uniform motion, which allows the network to better understand the task of frame interpolation, rather than frame averaging, when it receives difficult input. The dataset contains over 80,000 samples, that is, 80,000 frame triplets, to complete one forward-backward pass on the network. We also set aside separate image sequences for validation, in order to properly track the performance of our model.

`YouTube-8m`, which provide millions of videos categorically labeled in an optimal format for current applied learning techniques, was used for the fine tuning process, where the model is now unconstrained in problem difficulty, and thus must become more robust to a variety of input frames. We downloaded only 20,000 videos from this dataset, with the requirement that the videos be 1280-by-720 resolution, and of type .mp4. In reality, however, the actual number of samples available in just 20,000 videos is incredibly high, that, combined with data augmentation techniques, this dataset is virtually infinite, and so no validation set was necessary, since the likelihood of an exact repeated sample was unlikely.

Both datasets were retrieved via live batch generator functions, which work well with the Keras framework. This live retrieval of data allows us to perform data augmentation and preprocessing on-the-fly. However, with regards to `YouTube-8m`, who's frames are stored in .mp4 video files, the use of the efficient, FFMPEG library was necessary to quickly retrieve frame data, while training the network at the same time, and not become too much of an I/O bottleneck.

For the `YouTube-8m` dataset specifically, we encountered the common issue of the frame triplets either showing no difference, or showing too much difference. Either case is not useful for training our network, and may in fact throw the network off. Therefore it is imperative to limit these samples, since they are so common in YouTube videos, which can range from static scenes to wildly variable scenes. To combat this, we employ a function to compute the average per-pixel difference between frames, which we denote as δ:

$$\delta = mean|f_k - f_i|$$

We then add the constraint that each sample triplet must satisfy, that $0.02 > \delta > 0.2$. These values were determined through manual experimentation, in order to filter out triplets with too little or too much variance.

3.2 Network Architecture

The Convolutional Neural Network architecture consists of 2 key blocks. The first block type contains two 3x3 Convolutional layers, followed by a 2x2 Max Pooling layer. The second block type contains a 2x2 Upsampling layer, followed by two 3x3 Convolutional layers. The network is structured with 5 blocks of the first type, followed by 5 blocks of the second type, where the number of filters for each Convolutional layer increase as the layers reach the center of the network, up to size 512. The network input has shape 6x128x384, in which we concatenate the first and last frames, f_i and f_j, on the channel axis. The last layer is then just a 1x1 Convolutional layer with 3 filters, thus outputting a shape of 3x128x384, because of the equal number of Max Pooling and Upsampling layers.

This inherent symmetry of the network allows yet another trick to increase performance. Before each of block of the second type, we have a Merge layer, that concatenates the output of the opposite block in the first half of the network. This allows us to maintain feature information from

earlier parts of the network, which had a narrower field of view. In fact, because we use a 128x384 resolution image as our network input, and we have 5 Max Pooling layers, the height and width of the layer outputs reaches a 4x12 resolution, which allows us to see a broader view of the input space as well.

This general CNN architecture is largely inspired by the work done in [2], where Ronnegerber, et. al demonstrated that proper ordering and choice of layers for the maximization of contextual information gain can greatly reduce space complexity and time until convergence. In addition to his overall structure, we add a few additional Convolution layers to increase the complexity of the model, and we also add Batch Normalization layers after every Convolution layer. This additional Batch Normalization layer, however, came with an unfortunate, and unforeseen side-effect, in that by using it, the model weights are now tied to the input image shape, so while the model is still Fully Convolutional (i.e. no fully-connected layers), we are limited to input images of size 128x384.

In addition, we utilize the ReLU non-linearity activation function after each Convolutional layer, and also perform padding on the input in order to maintain the same input and output relative size. This is trivially done by the use of `border_mode='same'`, in Keras.

3.3 Training the CNN

The initial training was done using the entire `KITTI` dataset, where over $150,000$ images were processed over the course of ten hours. Note that this means nearly two passes over the dataset were performed. Consistent with the objective given by (3), an increase in the accuracy of the model is given by an increase in the loss incurred during validation, hence the weights defining the network topology are only updated when such an increase in validation loss occurs. This initial training process created an unrefined model, which then was further improved by training over $80,000$ samples from `8m` over the next twenty-four hours. The data processing stage for `8m` samples takes significantly longer than for `KITTI` samples due to the need to sample MPEG-4 frames during runtime, whereas `KITTI` is provided already as sets of sequential disjoint images. The same technique was used in training with the `8m` dataset, where model weights are updated only given an acceptable increase in validation loss.

4 Results

Figure 1: Loss of unrefined model with respect to iteration count

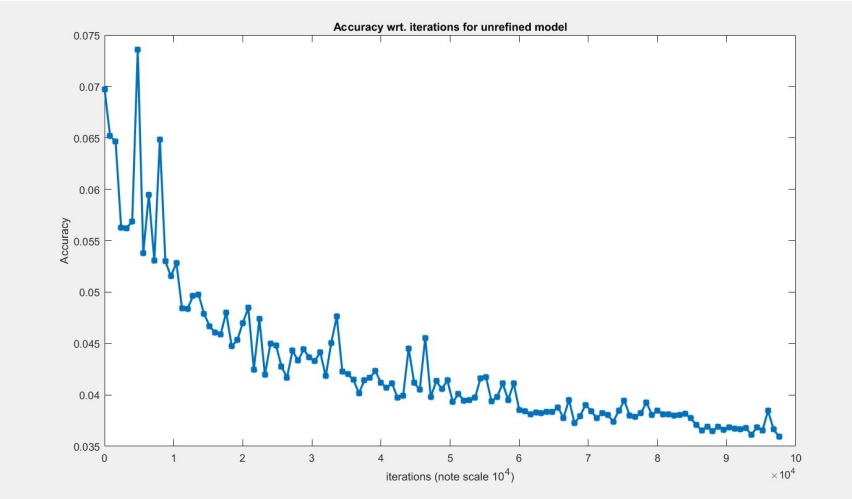


Figure 2: Loss of refined model with respect to iteration count

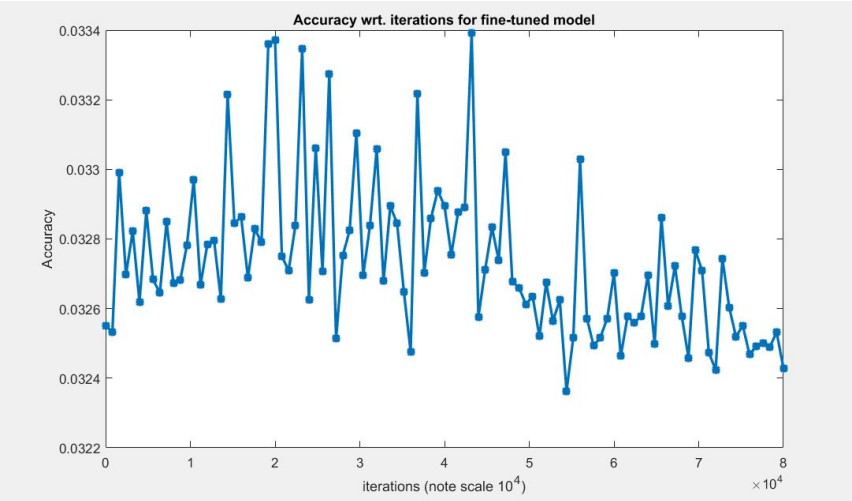


Figure 3: Four sets of image tuples from the KITTI validation set, where "Blended" gives a naive linear interpolation, "Middle GT" gives the ground truth true middle frame, and "Prediction" gives the approximation generated by our model

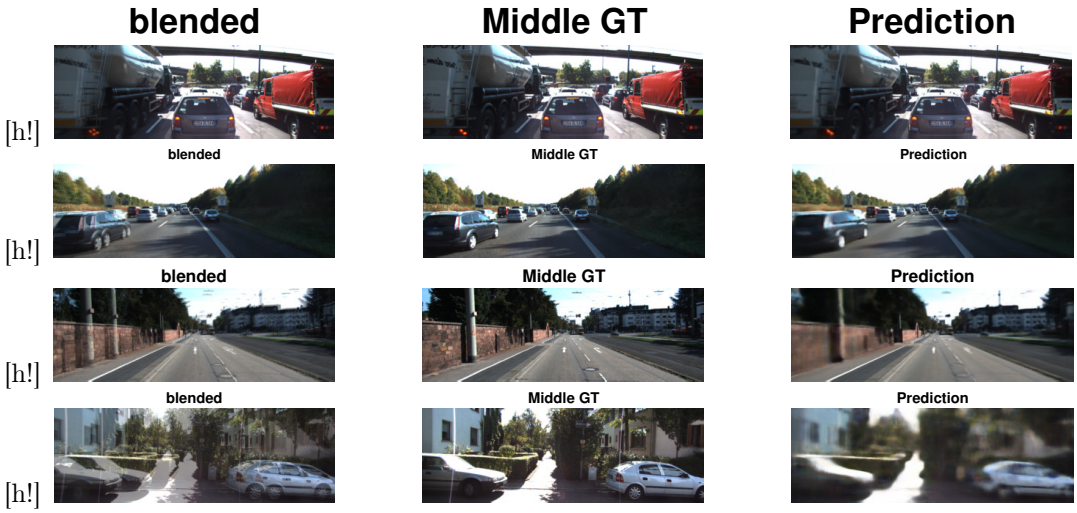


Figure 4: Comparison of model performance before and after fine-tuning using the YouTube-8m dataset. It shows that with exposure to a wider variety of images, the model becomes more rubust, and is able to handle the unusual image/motion below.

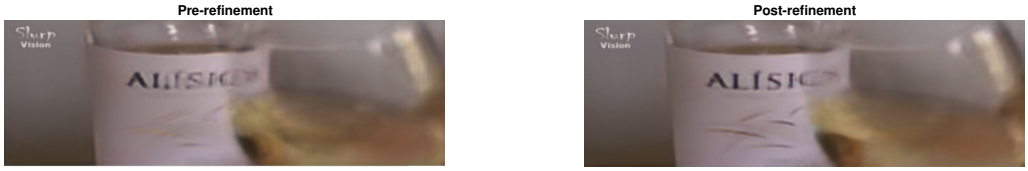
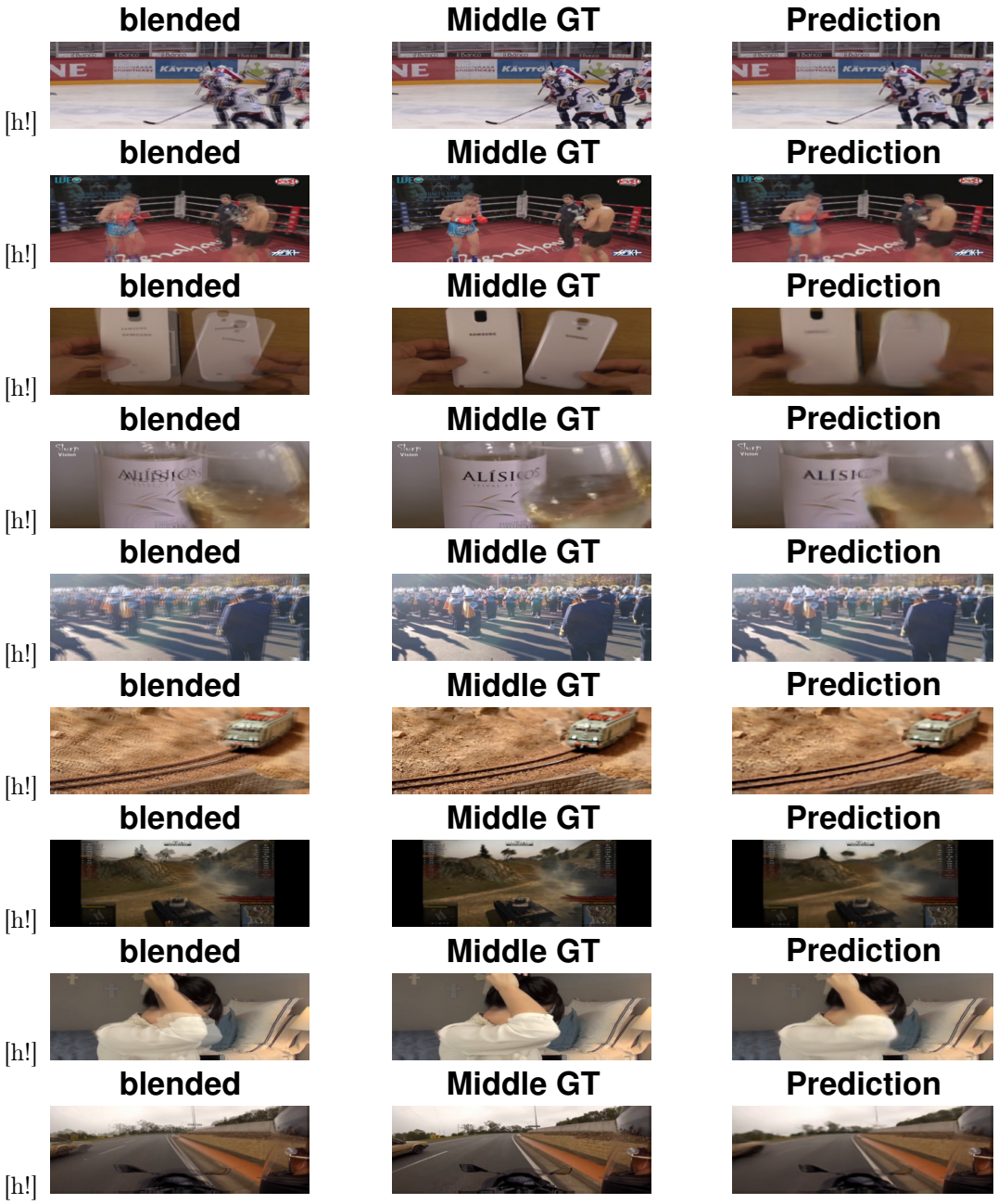


Figure 5: Nine sets of image tuples from YouTube-8m, where "Blended" gives a naive linear interpolation, "Middle GT" gives the ground truth true middle frame, and "Prediction" gives the approximation generated by our model



For more examples, including some interesting videos with multiplied frame rates and slow motion, visit our project on GitHub: https://github.com/neil454/deep-motion

5 Conclusions

We have designed and implemented an end-to-end model to perform frame interpolation, using a simple and intuitive CNN. Our results show promising results, and it is clear that the network excels at predicting the interpolated location of objects in the image scenes, even if the end result is of lower quality than desired. There are several possible extensions one could try for improving our solution. Since we did not observe any over-fitting during training, it is possible that a more complex model could improve performance during training. There is also the idea of utilizing weights from a model pre-trained on ImageNet, for example. Lastly, while Convolutional Neural Networks are a leading tool in many applications, the use of LSTMs, Recurrent Neural Networks, or even Generative models, could find some use in this problem, this is especially interesting, since the input of our network is only two frames, one wonders how the performance may fair with more input frames, and a more temporal input in general.

References

[1] John Flynn and Ivan Neulander and James Philbin and Noah Snavely, *DeepStereo: Learning to Predict New Views from the World's Imagery*, CoRR **abs/1506.06825** (2015).

[2] Olaf Ronneberger and Philipp Fischer and Thomas Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, CoRR **abs/1505.04597** (2015), available at http://arxiv.org/abs/1505.04597.

[3] Diederik P. Kingma and Jimmy Ba, *Adam: A Method for Stochastic Optimization*, CoRR **abs/1412.6980** (2014), available at http://arxiv.org/abs/1412.6980.

[4] Gucan Long and Laurent Kneip and Jose M. Alvarez and Hongdong Li, *Learning Image Matching by Simply Watching Video*, CoRR **1603.06041v2** (2016), available at https://arxiv.org/pdf/1603.06041v2.pdf.