Journal of Intelligent Learning Systems and Applications, 2023, 15, 89-107
https://www.scirp.org/journal/jilsa
ISSN Online: 2150-8410
ISSN Print: 2150-8402

Camera Independent Motion Deblurring in Videos Using Machine Learning

Tyler Welander, Ronald Marsh, Bryce Gruber

Department of Electrical Engineering and Computer Science, University of North Dakota, Grand Forks, North Dakota, USA
Email: tyler.welander@und.edu, ronald.marsh@und.edu, bryce.gruber@und.edu

How to cite this paper: Welander, T., Marsh, R. and Gruber, B. (2023) Camera Independent Motion Deblurring in Videos Using Machine Learning. *Journal of Intelligent Learning Systems and Applications*, **15**, 89-107.
https://doi.org/10.4236/jilsa.2023.154007

Received: August 2, 2023
Accepted: November 3, 2023
Published: November 6, 2023

Abstract

In this paper, we will be looking at our efforts to find a novel solution for motion deblurring in videos. In addition, our solution has the requirement of being camera-independent. This means that the solution is fully implemented in software and is not aware of any of the characteristics of the camera. We found a solution by implementing a Convolutional Neural Network-Long Short Term Memory (CNN-LSTM) hybrid model. Our CNN-LSTM is able to deblur video without any knowledge of the camera hardware. This allows it to be implemented on any system that allows the camera to be swapped out with any camera model with any physical characteristics.

Keywords

Motion Blur, Video, Convolutional Neural Network, Long Short-Term Memory, AirSim, OpenCV

1. Introduction

Optical cameras often play a crucial role in autonomous unmanned aerial vehicles. They are often used in navigation systems for object detection and avoidance, or they can be used for collecting visual information about the environment the vehicle is operating in. Unfortunately, various issues arise when using optical cameras in these situations. One such issue is motion blur. Motion blur can introduce unwanted artifacts or distort the image or video to the point where it no longer contains any useful information. To overcome motion blur, the image or video must be processed to try to reduce or eliminate the blur.

Various methods can be used to deblur an image or video. These methods can be in the form of statistical analysis such as heavy tail analysis proposed by Zhu [1]. There are also techniques that utilize machine learning such as the Variant Depth Network proposed by Guo, Wang, Hong-Ning, and Li [2]. There are also

more unconventional methods such as using super-resolution as proposed by Fang, Ning, Zhan, and Zongqian [3]. All these techniques have a common flaw, and that is that they are often tailored to a specific camera. These methods will consider the various characteristics of the camera to aid in the process of deblurring the image. To overcome the issue of camera specificity, we need a method that is completely agnostic to the camera. This will allow the method to be used with any image or video.

In this paper, we will be looking at a novel method for deblurring video. This method utilizes machine learning and is completely camera agonistic. The method was developed and tested in a virtual environment created using the Unreal Engine and AirSim such that camera parameters could be adjusted throughout testing and development to determine if it is truly agnostic. First, we will provide background knowledge and terminology of the various terms and technical ideas used throughout the paper. Then, a brief survey of related works and any definitions of important concepts will be provided. We will then discuss our experimental design, such as information on the environments and datasets. Details on our proposed model will then be provided along with the results of our experiment. We will then provide a conclusion that will discuss all the important takeaways from this experiment, lessons learned, and any future work would like to accomplish.

2. Background

The main issue with cameras is that they often introduce motion blur in the images. Motion blur is caused by the shutter speed on the camera being too slow for how fast the camera or the objects in the image or video are moving. If there is movement in the image and the shutter doesn't close fast enough, the camera will collect light information about the object as it is moving. This will cause motion blur to occur in the image or video and it will look like streaking or smearing in the image. **Figure 1** shows what motion blur looks like in an image.

To acquire a quality image, that both humans and machines can interpret, the settings for the aperture, sensitivity to light, and shutter speed all need to be balanced to create an image that is properly exposed and free of artifacts. Most cameras manufactured presently can auto-balance some of these key characteristics, but when trying to integrate small and cheap cameras into a system, there are often sacrifices that need to be made in the camera design. For instance, a manufacturer may try to reduce the cost of a camera by reducing the sensitivity to light or by slowing down the shutter speed. This does lower the cost of producing the camera but it also lowers the quality of the images the camera produces. This will in turn cause humans and machines to misinterpret the details in the images the camera produces.

Having a fast shutter on a camera comes at a cost of size and price, so lower priced cameras generally have slower/longer shutters. For this reason, there have been several techniques developed for removing motion blur from a single captured image.

Figure 1. Motion blur example.

The concept of motion blur also affects video since video is just a series of pictures, typically referred to as frames, played at a certain speed. The unique opportunity in trying to correct motion blur with video, as opposed to single images, is that video is typically captured such that sequences of frames are remarkably similar to other frames in the sequence. By utilizing the surrounding frames, additional information may be provided to aid in removing motion blur in a video sequence more effectively than a single image.

While there are solutions to motion blur that can be applied by hardware, not all cameras have the capability to modify their settings to the degree that is needed. For this reason, our goal is to provide a software solution that can be applied to a camera capable of capturing standard video. For this discussion, we are going analyze two primary approaches to this problem through software means, using traditional computational methods and applying machine learning methods.

2.1. Blur Kernel

A blurred image can be considered to be composed of a combination of a clear image and a modifying element in the form of a blurring kernel. The equation 1 describes this:

$$B = k \otimes L + n \tag{1}$$

In Equation (1), a clear image L has a blur kernel k applied and noise factor n added to generate the resultant blurred image B. To unblur an image, one must extract the blur kernel from the blurred image to inversely reconstruct the clear image.

Multiple techniques for deblurring using the blur kernel concept have been developed. A maximum a posteriori (MAP) estimation-based method specializes in trying to maximize the probability of the posterior [4] [5] [6]. Adjustments are made to the blur kernel and the clear image. Variational Bayesian (VB)-based methods seek the best blur kernel by utilizing a distribution of probable clear images [7] [8] [9]. Another method is the edge-based method [4] [5] [6].

This method will extract edges and use the generated clear image to estimate the value of the blur kernel. However, it must be done in conjunction with filtering or optimizing. As pointed out by Liu, Shaoguo, Wang, *et al.* [10], these methods are not always the most accurate as the aperture chosen for the blur kernel is often done manually. This is not ideal since choosing the wrong size kernel to use will significantly impact the generated results.

2.2. Point-Spread Function

The theory behind the Point-Spread Function (PSF) is similar to that of the blur kernel in the way that the blurred image is composed of the initial image and a blurring element. In this case, instead of a blurring kernel, a blurring function is applied to an image to generate the blurred image. For example, to blur an image with a blur kernel you would use Equation (1). To blur an image with a PSF, you would use Equation (2).

$$B = \mathrm{PSF} \otimes L + n \tag{2}$$

As can be seen, the only difference is the blur kernel is replaced with a PSF. The unique opportunity that this technique provides is the ability to incorporate frequency domain analysis for deblurring.

There have been several attempts to remove motion blur through the PSF approach. One such attempt involved taking several images of the same subject with the camera set to different shutter speeds. When the PSF is transformed to the frequency domain, the PSF is expected to contain null values filled with zeros. By changing the shutter speed used to capture the image, the range of the null values is shifted. This allows for null filling to take place by filling the null values of one image captured by another to form a joined PSF as shown by Agrawal [11]. While this technique appears promising, there are two notable issues. Agrawal assumes that the vehicle is traveling at a constant velocity; while this should be fine when using a small sample size of images captured quickly, this variable cannot be guaranteed in all situations. Additionally, to take videos of an object with different shutter speeds in the real world, specialized software may have to be integrated into the camera to switch shutter speeds after capturing an image.

2.3. Super Pixels

Another approach for deblurring is to separate regions of an image into superpixels as demonstrated by Qiao [12]. Many attempts at addressing motion blur through a mathematical approach try to find a single blur kernel for the entirety of an image. While this may simplify the approach to the problem, it may be difficult to address the varying levels of motion blur that could occur in the image. Superpixels allow for creating many smaller sections of the original image and to calculate many localized blur kernels that are more accurately defined by the small, localized area.

Generating accurate blur kernels from a blurred image can be difficult without

having a clear reference to verify. Qiao was able to use sections of superpixels to accurately define localized blur kernels within a video by using multiple frames of a video to synthesize a clear frame. This process is broken up into three main parts, segmentation, kernel estimation, and path matching, to generate a deblurred frame.

The initial segmentation of a video frame is driven by a few different factors. One of these factors includes color and may not generate accurate results if there are moving objects that contain multiple high-contrasting colors, therefore, the estimated motion of an object is considered when segmenting the video frame into superpixels.

After segmentation, blurry and clear superpixels are identified. While a frame should be expected to be composed of blurry pixels by a majority, some clear superpixels may be present. Blur kernels are then calculated for each superpixel. Each kernel can then be optimized from surrounding superpixels within the frame to form the global kernel. A comparison between the target blurred frame and a neighboring clear frame can then be used to estimate the motion model.

Clear superpixels within a series of frames are used to replace blurry superpixels through path matching. Variables from the previous two steps are used to optimize the search and eventually apply texture synthesis to cleanly blend superpixels after matching within the reconstructed frame.

The main caveat this technique brings, though having promising results, is the computational time it takes to compute individual frames. Ideally, we would like to have as close to real-time processing as possible, but with the outlined technique, the quickest a single frame took over 16 seconds to process.

2.4. Neural Networks

Machine learning methods involve utilizing any algorithm that falls in any of the various classes of algorithms that are supposed to simulate learning in a computer. In general, machine learning methods and algorithms work by analyzing data to approximate the desired solution or searching the search space looking for the desired solution. Russell and Norvig defined these classes as neural networks, evolutionary algorithms, and swarm intelligence algorithms [13].

Neural networks are a set of simulated neurons arranged in various architectures. These architectures each have strengths that make them ideal for certain situations. Convolutional neural networks (CNN) are quite useful for image processing tasks while long short-term neural networks are great for processing time-dependent data. What makes neural networks so powerful is they can classify or group data based on the patterns within the data. The developer of the neural network only needs to know the type of data he or she is getting, the structure of the data, and the goal of the neural network to develop a functioning neural network. There is no need to know what the data contains. Neural networks will gain the ability for classifying or grouping data through the process of training. There are three forms of training neural networks can go through. The

first form is supervised learning. Supervised learning is when the training set of data is labeled or there is a set of data that says what the data is. An example of this is when you are trying to train an image classifier. Before the user starts using the network to say what images are, a set of images will be passed in to train the network. The images in this training set will contain some sort of metadata that says what the image is. After this training set is used, the neural network will be able to identify non-labeled data to a certain degree of accuracy depending on the quality of the training data set. The second form of training is unsupervised learning. Unsupervised learning is the same as supervised learning except the training set is not labeled. This form of training is often used for anomaly detection or any tasks where the grouping of the data is not necessarily known beforehand. The last type of training is reinforcement learning. This type of training is continuously happening as the network is being used and usually doesn't end. Reinforcement learning is generally used in applications where training data is hard to get or is continuously changing. Robotics is an example of when reinforcement learning is employed.

Neural networks have been proven to show some promise in the field of motion deblurring. In [14], Chen *et al.* proposed a CNN-LSTM hybrid model. Their model works by first looking at each image or frame in the video and it attempts to deblur the image without looking at the other images. It will then predict an inverse blurring kernel and record that amount. Once the model has done this for all frames, it will then attempt to find an average inverse blurring kernel to be used across all the images. The idea behind this is that by just looking at a single image the inverse blurring kernel can be inaccurate compared to what the desired kernel would be, and to correct it, you average the kernel across all frames to try to improve the accuracy of the deblurring kernel. The results of this method can be seen in **Figure 2**. Now, this proposed model has two flaws for our use case. The first flaw is that it can not be performed in real-time. Since it is analyzing all the frames, the entire video needs to be completed before the deblurring process can begin. The second flaw is that the model must take into account the camera parameters to aid in the deblurring process which is something we are trying to avoid.

(a) Input Image (b) Conv-LSTM

Figure 2. Results of Conv-LSTM shown.

Yang *et al.* [15] have also attempted to utilize a CNN network to remove motion blur from video. Instead of utilizing a standard CNN, they developed a 3D CNN to analyze and extract spatial-temporal features to generate additional intermediate frames. By combining the 3D CNN with a Fourier accumulation module (FAM), generated intermediate frames can be further processed to remove the blurring effect. An example of the input image and the resulting clear image from their model can be seen in **Figure 3**. Depending on the type of alignment used in processing, computational times can vary between several seconds to a few minutes. To improve computational speed one may, the best choice is to not apply alignment; however, this is only recommended in situations when relative motion is small. While relative motion can be minimized by increasing the frame rate of a video, this is not always possible with all hardware used for capturing video.

There are a few issues with utilizing traditional CNN methods for reducing motion blur within images. Input images may have unwanted deblurring effects due to the loss of spatial information in some of the image preparation steps. Additionally, to increase the quality of the result, more convolutional layers are typically added to a network causing a larger model size and demanding more computational resources. Guo *et al.* [2] has demonstrated that a variant-depth network (VDN) provides a solution to these commonly found limitations. By using subnetworks that vary in depth and create different levels of deblurring, more information is preserved during the computational process. As shown by the group's research, not only does this method excel at the qualitative results when compared to other methods by producing deblurred images that are extremely similar to their truth images, but also outperformed the other methods computational time. In comparison, the developed VDN is over twice as fast as its fastest competitor as tested by Guo *et al.* This method is currently optimized for single images but could potentially be expanded for use with video deblurring in the future with some modifications.

2.5. Super-Resolution

Along with the previously stated techniques for motion deblurring, there are a few unconventional techniques that are quite effective. One of these techniques that we are interested in is super-resolution. Super-resolution is a class of techniques that are used to increase the resolution of images or frames in videos. From a high level, super-resolution works by performing some form of analysis on the pixels of the image. The analysis is then used to calculate the appropriate values for the pixels missing in the image and these generated pixels are then inserted throughout the image to increase the resolution. Even though these pixels were not in the original image and were generated after the fact, it can be argued that the generated pixels are accurate to a certain extent. For this paper, we are not directly concerned with how super-resolution works or its primary use case of it. In image deblurring, there are instances where pixels are missing so we can use the super-resolution to aid us in the deblurring process.

(a) Input Image (b) 3D CNN + FAM

Figure 3. Results of 3D CNN + FAM.

In [16], Matsushita *et al.* proposed a method for motion deblurring using super-resolution. The idea behind their proposed method is that they would first perform an analysis of the image to estimate a blur kernel. The inverse of the blur kernel would be performed on the image to reduce the motion blur. This method worked for reducing the motion blur, but the image was not an accurate duplicate of the original image that did not have any motion blur. This is due to the motion blur overwriting or corrupting the pixels in the image so the data for those pixels would be lost. In most cases, motion blur can be corrected enough so that the human eye cannot pick up differences at a glance. In severe cases where it cannot be corrected enough, super-resolution, as shown by the authors, can be used effectively to restore the missing or corrupted pixels. For our experiment, we are not concerned with the super-resolution used by the authors. Instead, we are more concerned with how they ran their experiment. The authors not only tested their proposed method on a single image, but they also tested it on a varying number of frames from a video. As can be seen in **Figure 4** is that the accuracy of their method improves with the number of frames the analysis is being performed on.

3. Experiment

For our work, we took a two-phased approach. In the first phase, we tested our model against a simulated dataset that contains video frames of a simple environment with very few objects. We then tested the Variant-Depth Network (VDN) model proposed by Guo, Wang, Dai, and Li [2] with this same dataset. A comparative analysis was performed on the results of these two models for the simple environment. The next step of this phase was to test both models in a complex environment that had many objects in it. This step followed the same procedure as our simple environment. For the second phase, we trained and tested our model using the Go Pro dataset [17]. This phase allowed us to compare our model against multiple other models that used this dataset. The purpose of this two-phase approach is due to there being a small number of suitable public datasets that could be used to test our model. Most datasets are created to suit a specific test and are never released publicly. Also, most models are not open sourced so they would have to be reimplemented using the details from the

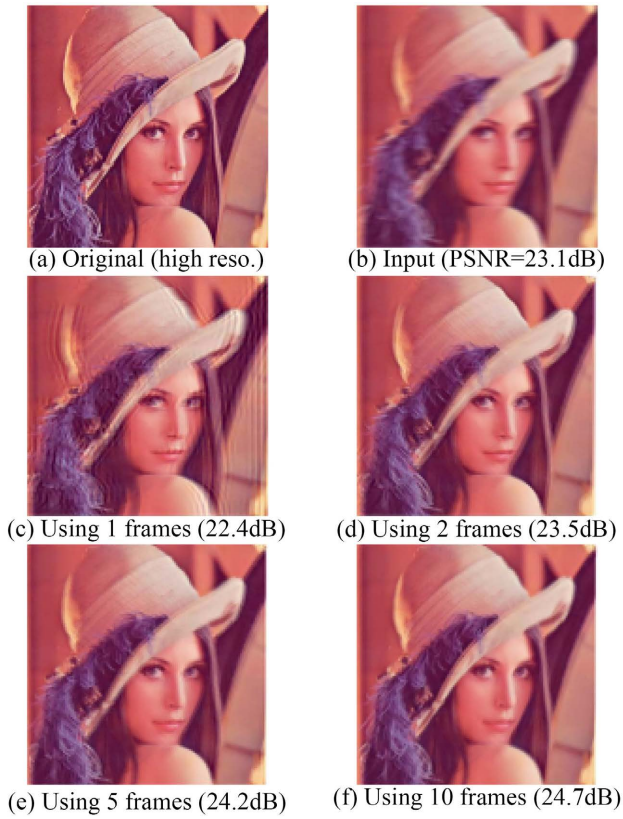


Figure 4. Results from the review of super resolution method by Matsushita *et al.* [16].

respective paper. This creates a difficult scenario to accurately test a model. To overcome this, we developed an open-source dataset of a simulated environment. This allowed us to test our model and the VDN against it. However, since this is a dataset of a simulated environment, we felt that it was a good starting point, but we should continue to use at least one dataset of real world images. We decided to use the Go Pro dataset for our real-world dataset as multiple models have already been tested against it and the models were all open source. Since the models were open source, we were able to train and test all models using the same computing platform. The computing platform used for these experiments include an Intel i7-9700k, NVidia RTX 2070 Super, 16 GB of 2800 MHz DDR4 memory, and a one TB m2 NVMe SSD running Windows 11.

3.1. Model

Our model that we developed for this experiment was a convolutional neural network-long short-term memory (CNN-LSTM) hybrid implemented using Python 3.10, Pytorch 1.13, and Cuda 11.6. The implemented model can be viewed as several separate models that all work together to accomplish motion deblurring. The first model is a CNN that is used to extract features from each frame. We will call this model the feature extractor. The purpose of the feature extractor is to generate the feature maps of each image. A feature map contains all the

information about the various details and objects in the image such as data on points, lines, or various shapes. For this particular model there are 32 feature maps. The feature extractor takes in each frame of the video as an input. Pytorch models require all inputs to be in the form of a tensor so the data of each frame must be preprocessed so it can be packaged as a tensor. In our case since each frame is 640 × 640 pixels, our input tensor is a three-dimensional tensor of size 640 × 640 × 3. The depth value of 3 is needed to represent the RGB values of each pixel. The input tensor is first passed to a two-dimensional convolutional layer. A 3 × 3 matrix is used for the convolution kernel in this layer. The output of the convolutional layer is then passed to an activation layer that uses a ReLu activation function. The output of the activation layer is then passed to a flattening layer to reduce the dimension of the tensor to one dimension. From here, the tensor is then passed to a buffer layer to be stored until the next four frames are processed by the CNN model. Once five frames are processed, those five frames are then sent to the LSTM component of our model.

As stated above, the LSTM processes five frames at a time. The goal of the LSTM is to use this series of frames to help determine where motion blur is occurring and the amount of motion blur. It accomplishes this by using the two frames before a specified frame and the two frames after to try to determine how the objects within the frame are moving while the shutter of the camera is open. This data/image flow is depicted in Figure 5. The frames are depicted as N1, N, or N2. The N frame is the current frame being processed. The 1 or 2 denote the position of the frame in the series relative to the current frame being processed. These five frames act as the input for the LSTM. Once the frames are processed by the LSTM, the oldest frame, or N + 2, is discarded and a new frame is requested from the CNN. The remaining frames shift over to accommodate the new frame. So, frame N + 1 becomes N + 2, frame N becomes N + 1, frame N − 1 becomes N, and N − 2 becomes N − 1. The new frame is depicted as N − 2. To accomplish these steps, the LSTM is architected with four hidden states and two LSTM layers stacked together. Each hidden state represents each frame in the series and two LSTM layers were chosen over one because it gave us the best results during our initial testing. After the frames are done being analyzed by the LSTM and removed from the series, they are sent to the final component of our model.

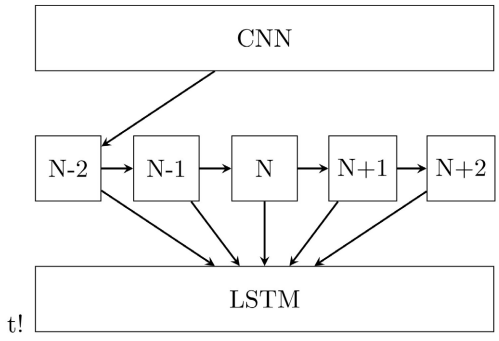


Figure 5. The flow of frames from CNN to LSTM.

The final component of our model is for post-processing the data and transforming the tensor that was output from the LSTM back into an image of the same size as the initial frames. This is accomplished by reshaping the output tensor of the LSTM to a tensor of shape $638 \times 638 \times 3$. The reasoning as to why we reshape the tensor to this shape instead of the initial $640 \times 640 \times 3$ is because the convolutional layer in the first component of the model causes us to lose data along the edges of the frame. To correct this, we add an upsampling layer to this component. The upsampling layer calculates the values for each field along the one pixel edge by calculating the average of the three nearest neighbors from the input tensor. After the tensor has been upsampled, we have the desired shape of the tensor of $640 \times 640 \times 3$. The tensor is then passed into a function to be used to generate the new frame.

3.2. Environments

Since the version of AirSim that we are utilizing is a plugin for the Unreal Engine, the possibilities for different environments are endless with the proper development. We used the Blocks and ZhangJiajie projects that were included in the Windows v1.8.1 release of AirSim to generate the synthetic data.

The Blocks environment [18], shown in **Figure 6** is rather simple. It is filled with simple block structures with some occasional points of interest, such as cubes, cylinders, or spheres. This allowed us to generate a simple image for the initial development stages.

ZhangJiajie [19] is noticeably more complex than Blocks. As shown in **Figure 7**, the inclusion of vegetation and other real-world features allows us to test our approach with a more realistic model.

3.3. Datasets

As stated above, our experiment used the Blocks and the ZhangJiajie environments to generate our simulated dataset. Our decision to utilize AirSim and these environments was driven by the fact that AirSim could generate various artifacts in the images, such as motion blur. However, it was discovered that the motion blur feature within AirSim was not functioning correctly, so an alternative method needed to be developed for generating the sets of images.

The approach used for generating sets of images with motion blur was to add a constant vertical or horizontal blur to various sets of images. Example kernels for this operation are shown in Equations (3) and (4) respectively. This allowed us to continue utilizing the environments developed within the Unreal Engine for generating synthetic data.

$$\frac{1}{5}\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{3}$$

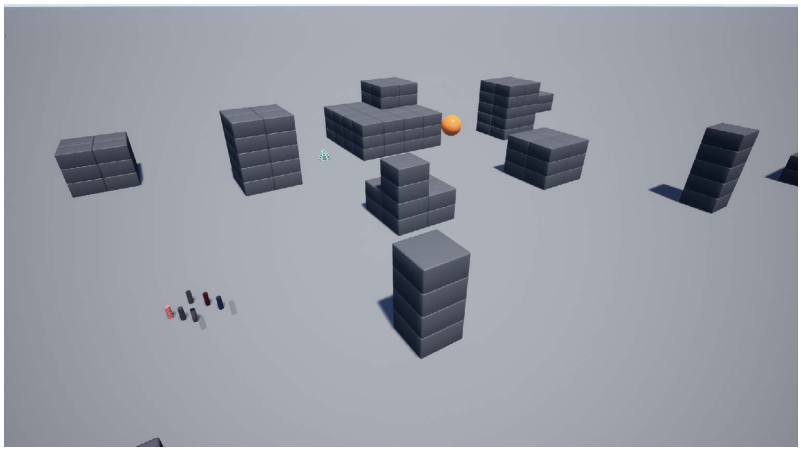


Figure 6. Sample image of blocks environment.

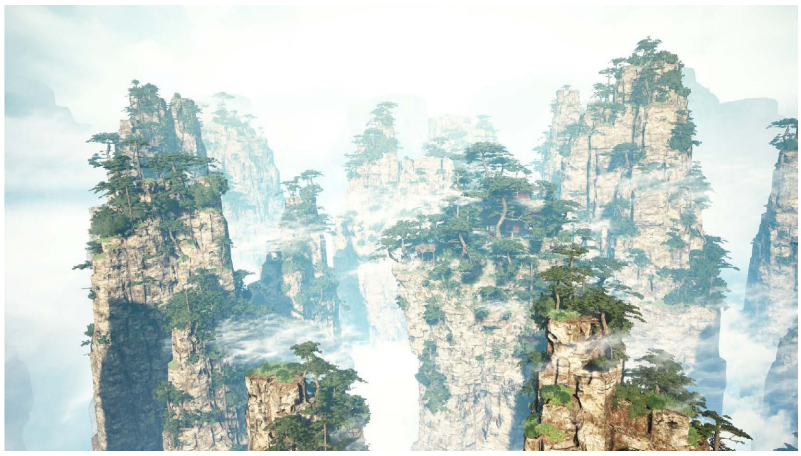


Figure 7. Sample image of ZhangJiajie environment.

$$\frac{1}{5}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

The images generated from the simulated environment will be used as clear images for training and testing the neural network. Additionally, the images will also provide image data for generating the images with motion blur. The other type of data used to generate sample source images with motion blur is the image depth map. The depth maps are generated using built in functions in AirSim and the Unreal Engine. Examples of both image types, normal vision and depth vision, captured from the synthetic environment are shown in **Figure 8**.

The reason behind the generation of the image depth map is related to how motion blur is different according to how close or far away a subject is from the camera. In this case, we are operating on the basis that all of the synthetic environments are static and the camera is moving. This would cause the subjects closer to the camera to have more motion blur. With using the depth map as a

(a) Image taken within environment (b) Depth map of image from environment

Figure 8. Generated depth map of an image.

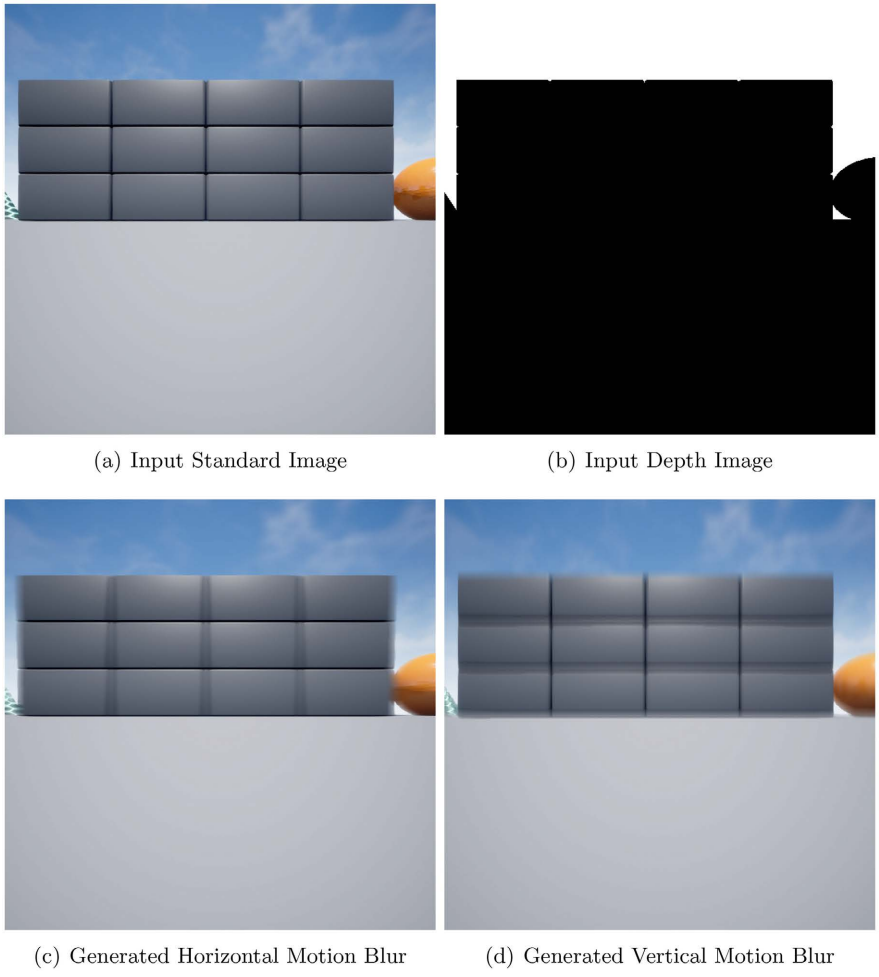
reference, different areas of the image will have blur kernels with varying apertures to them. The larger the value in the depth map, the larger the blur kernel that is applied.

As stated, there are two different datasets that we generated for this experiment. In both environments, five key locations were picked out in each of the environments. We then proceeded to fly the simulated drone around each of these key locations for 30 seconds. This resulted in five minutes of total video between the two environments. All videos were recorded at 24 frames per second. Before we applied the motion blur, we had a total of 7200 frames. After all the videos were collected, we selected 3 kernel sizes for horizontal blurring and 2 kernel sizes for vertical blurring. For horizontal motion blurring, we used kernel sizes of 10, 30, and 50 to produce frames with motion blur in them. The reason for multiple blur kernels is to simulate the varying speeds the simulated drone is flying at. The larger blur kernel corresponds to faster velocities the simulated drone is flying at. For vertical motion blurring, we selected kernel sizes 20 and 40. Each frame in our dataset was used to produce five motion-blurred images with the selected kernel sizes to give us a dataset size of 36,000 frames. **Figure 9** shows a generated depth map and examples of final images after generating motion blurring.

For our real-world dataset, we utilized the Go Pro dataset. The Go Pro dataset is divided into two individual datasets. One of the datasets is for training and the other one is for testing. The training dataset is comprised of 2103 images and the testing dataset is comprised of 1111 images. Each of these images in the datasets also have a corresponding sharp image included in the dataset. This sharp image is compared against the deblurred image our model produces to ensure our model can accurately deblur the images.

3.4. Results

As stated above, our model was evaluated and compared in two different experiments. First, our model was compared to the VDN model proposed by Guo, Wang, Dai, Hong-Ning and Li using Peak Signal-to-Noise Ratio (PSNR) and structural similarity index measure (SSIM) methods for comparative results. For both PSNR and SSIM, higher values correspond to better results and lower values correspond to worse results. The VDN model was chosen for comparison



(a) Input Standard Image (b) Input Depth Image

(c) Generated Horizontal Motion Blur (d) Generated Vertical Motion Blur

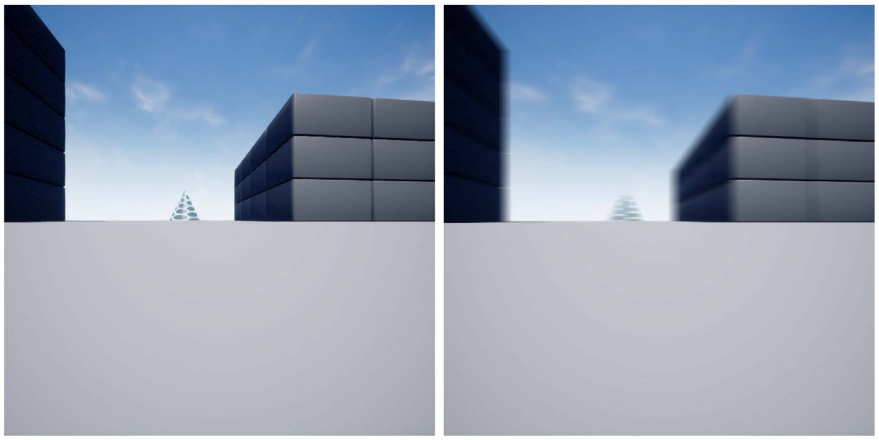
Figure 9. Sample blur results.

because the code was open source, so we were able to test their model using our dataset. Both models were trained for 10 epochs against our generated dataset. The number of epochs was chosen after initial testing revealed the rate of which the accuracy improved was starting to slow down. From **Table 1**, you can see that the VDN has a PSNR value of 31.02 and a SSIM of 0.9473, and our CNN-LSTM model produced values of 30.86 and 0.9398 for PSNR and SSIM respectively. In **Figure 10**, you can see some of the resulting images from this experiment.

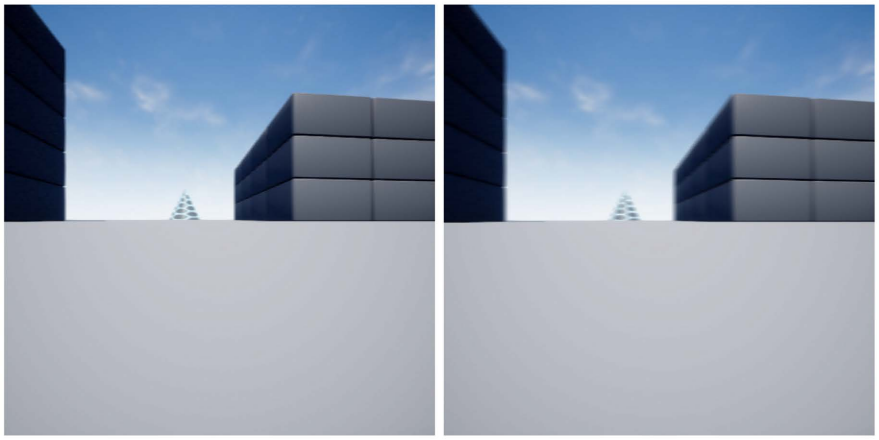
Our second experiment involved comparing our model against the deep multi-scale CNN model purposed by Nah, Kim, and Lee [17], the Non-Uniform CNN model purposed by Sun, Cao, Xu, and Ponce in [20], the Total Variation-Latent Image (TV-L1) model purposed by Kim and Lee [21], and the VDN purposed by Guo, Wang, Dai, Hong-Ning and Li [2]. For this second experiment, instead of using our simulated dataset, we used the Go-Pro dataset [17] and compared the models using PSNR and SSIM. All the models were trained for 10 epochs. From **Table 2**, you can see that our CNN-LSTM does not perform well compared to the other models. With a PSNR of 21.22 and a SSIM of 0.7988, our CNN-LSTM (in bold) did the worst of all the models.

Table 1. VDN and CNN-LSTM results.

Model	PSNR	SSIM
VDN	31.02	0.9473
CNN-LSTM	30.86	0.9398



(a) Target Image (b) Generated Blurred Image



(c) Deblurring Results of VDN (d) Deblurring Results of CNN-LSTM

Figure 10. Deblurring results.

Table 2. Results from Go-Pro dataset experiment.

Model	PSNR	SSIM
Deep Multi-Scale CNN	27.41	0.8873
Non-Uniform CNN	23.55	0.8322
TV-L1	22.74	0.8203
VDN	22.62	0.8198
CNN-LSTM	**21.22**	**0.7988**

During our analysis of the results from the first experiment, we discovered that our CNN-LSTM model had a severe drop in performance around the edges

of the image. With this observation, we decided to see if our CNN-LSTM produced better results if we ignored a 5-pixel wide strip along each of the edges when calculating the PSNR and SSIM using images from our simulated dataset and the Go Pro dataset. This led to a large increase in accuracy for our model. This improvement was so great, our model outperformed the VDN when the VDN also ignored the edges for the PSNR and SSIM calculations using both the simulated dataset and the Go Pro dataset. Table 3 shows the results in our simulated environment and Table 4 shows the results for the Go Pro dataset. As you can see from Table 3 and Table 4, our CNN-LSTM outperformed the VDN in both experiments. For the simulated environment, our CNN-LSTM had a PSNR of 37.52 and a SSIM of 0.9625 while the VDN had values of 34.36 and 0.9584 for the PSNR and SSIM respectively. For the experiment using the Go Pro dataset, our model had values of 23.41 and 0.8274 for the PSNR and SSIM while the VDN had 23.02 and 0.8256. With this new data, we can argue that our CNN-LSTM is superior to the VDN if we can improve its performance along the edges of the image. While our model is still less accurate than the other models when ignoring the edges, our CNN-LSTM is very close in accuracy to the TV-L1 and the Non-Uniform CNN models using the Go Pro dataset. With our model only being 0.11 PSNR and 0.0022 SSIM behind the TV-L1, we believe that with simple tuning our model will become more accurate than the TV-L1. For the Non-Uniform CNN model, there was a difference of 0.55 PSNR and 0.0194 SSIM between it and our CNN-LSTM. While the difference is larger than the difference between our model and the TV-L1, we still believe we can become more accurate than the Non-Uniform CNN model with some tuning. As for the data from our second experiment (using real-world images), we speculate that our model had a severe performance decrease due to the increased number of objects in the scenes and the more realistic motion blurring happening in the images.

Table 3. Results when ignoring the edges for simulated dataset.

Model	PSNR	SSIM
VDN	34.36	0.9584
CNN-LSTM	37.52	0.9625

Table 4. Results when ignoring the edges for Go Pro dataset.

Model	PSNR	SSIM	Runtime
Deep Multi-Scale CNN	28.11	0.9002	36.07s
Non-Uniform CNN	23.96	0.8468	58.22 s
TV-L1	23.52	0.8296	124.36 s
VDN	23.02	0.8256	31.74 s
CNN-LSTM	**23.41**	**0.8274**	**24.53 s**

After collecting the results from the previous experiment, we decided to run one more test. We wanted to see how our CNN-LSTM model compared to the other models in terms of runtime. We captured the runtime of each of these models by capturing how long each model took to deblur all the images in the test set of the Go Pro dataset. We then repeated each run five times and then we averaged the results. As you can see in **Table 4**, our model has superior runtimes with an average runtime of 24.53 seconds while the next fastest was the VDN model with 31.74 seconds as the average runtime. If we assume a framerate of 24 hertz, only 3 of the models will be capable of running in real time. These models are our CNN-LSTM, the Deep Multi-Scale CNN, and the VDN. For determining if a model can run in real time, we checked if the total runtime was less than 46.29 seconds. If it was less than 46.29 seconds, the model was deemed capable of running in real time. The 46.29 seconds is determined by dividing the total number of images in the test set, 1111 images, by 24. The number 24 represents the number of images the model needs to process in one second if we assume the camera runs at 24 hertz.

4. Conclusion

In this paper, we proposed a hardware agnostic solution to motion deblurring in videos. This enables our solution to be used in any application where camera hardware can be interchangeable, and the physical characteristics of the camera are not static. While experimental results showed that our model was not the most accurate model, they did show that our model had a significant improvement in accuracy when ignoring the edges of the images. Even though all models had an increase in accuracy when ignoring the edges, our model had a large enough improvement that it outperformed the VDN and came very close to the TV-L1 and the Non-Uniform CNN models. Our results also showed that our model was the fastest out of all the tested models. This enables our model to be the optimal solution in applications where speed is more desirable than accuracy, and when the center of the image is more important than the edges. For future work, we hope to optimize our solution so not only is it the fastest, but it is also the most accurate even when we do not ignore the edges.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Zhu, Y.-F. (2010) Blur Detection for Surveillance Video Based on Heavy-Tailed Distribution. 2010 *Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics*, Shanghai, 22-24 September 2010, 101-105. https://doi.org/10.1109/PRIMEASIA.2010.5604950

[2] Guo, C., Wang, Q., Dai, H.-N. and Li, P. (2022) VDN: Variant-Depth Network for Motion Deblurring. *Computer Animation and Virtual Worlds*, **33**, e2066. https://doi.org/10.1002/cav.2066

[3] Fang, N. and Zhan, Z.Q. (2022) High-Resolution Optical Flow and Frame-Recurrent Network for Video Super-Resolution and Deblurring. Neurocomputing, **489**, 128-138. https://doi.org/10.1016/j.neucom.2022.02.067

[4] Qi S., Jia, J.Y. and Aseem, A. (2008) High-Quality Motion Deblurring from a Single Image. ACM Transactions on Graphics, Vol. 27. https://doi.org/10.1145/1360612.1360672

[5] Krishnan, D., Tay, T. and Fergus, R. (2011) Blind Deconvolution Using a Normalized Sparsity Measure. *CVPR* 2011, Colorado Springs, 20-25 June 2011, 233-240.

[6] Li, X., Zheng, S.C. and Jia, J.Y. (2013) Unnatural L0 Sparse Representation for Natural Image Deblurring. 2013 *IEEE Conference on Computer Vision and Pattern Recognition*, Portland, 23-28 June 2013, 1107-1114. https://doi.org/10.1109/CVPR.2013.147

[7] Fergus, R., Singh, B., Hertzmann, A., Roweis, S.T. and Freeman, W.T. (2006) Removing Camera Shake from a Single Photograph. *ACM Transactions on Graphics*, **25**, 787-794. https://doi.org/10.1145/1141911.1141956

[8] Levin, A., Weiss, Y., Durand, F. and Freeman, W.T. (2009) Understanding and Evaluating Blind Deconvolution Algorithms. 2009 *IEEE Conference on Computer Vision and Pattern Recognition*, Miami, 20-25 June 2009, 1964-1971. https://doi.org/10.1109/CVPR.2009.5206815

[9] Levin, A., Weiss, Y., Durand, F. and Freeman, W.T. (2011) Efficient Marginal Likelihood Optimization in Blind Deconvolution. *CVPR* 2011, Colorado Springs, 20-25 June 2011, 2657-2664. https://doi.org/10.1109/CVPR.2011.5995308

[10] Liu, S.G., Wang, H.B., Wang, J. and Pan, C.H. (2016) Blur-Kernel Bound Estimation From Pyramid Statistics. *IEEE Transactions on Circuits and Systems for Video Technology*, **26**, 1012-1016. https://doi.org/10.1109/TCSVT.2015.2418585

[11] Agrawal, A., Xu, Y. and Raskar, R. (2009) Invertible Motion Blur in Video. *ACM Transactions on Graphics*, **28**, 1-8. https://doi.org/10.1145/1576246.1531401

[12] Qiao, C.B., Lau, R.W.H., Sheng, B., Zhang, B.X. and Wu, E.H. (2017) Temporal Coherence-Based Deblurring Using Non-Uniform Motion Optimization. *IEEE Transactions on Image Processing*, **26**, 4991-5004.

[13] Russell, S. and Norvig, P. (2021) Artificial Intelligence: A Modern Approach. Pearson, London.

[14] Chen, H.Y., Teng, M.G., Shi, B.X., Wang, Y.Z. and Huang, T.J. (2022) A Residual Learning Approach to Deblur and Generate High Frame Rate Video With an Event Camera. *IEEE Transactions on Multimedia*, 1-14. https://doi.org/10.1109/TMM.2022.3199556

[15] Yang, F., Xiao, L. and Yang, J.X. (2020) Video Deblurring Via 3D CNN and Fourier Accumulation Learning. 2020 *IEEE International Conference on Acoustics, Speech and Signal Processing*, Barcelona, 04-08 May 2020, 2443-2447. https://doi.org/10.1109/ICASSP40776.2020.9054514

[16] Matsushita, Y., Kawasaki, H., Ono, S. and Ikeuchi, K. (2014) Simultaneous Deblur and Super-Resolution Technique for Video Sequence Captured by Hand-Held Video Camera. 2014 *IEEE International Conference on Image Processing*, Paris, 27-30 October 2014, 4562-4566. https://doi.org/10.1109/ICIP.2014.7025925

[17] Nah, S., Kim, T.H. and Lee, K.M. (2018) Deep Multi-Scale Convolutional Neural Network for Dynamic Scene Deblurring. *Computer Vision and Pattern Recognition*, **1**, 1-21.

[18] Setup Blocks Environment for AirSim. https://microsoft.github.io/AirSim/unreal_blocks/

[19] Shouhuzhedelang (2017) Zhang Jia Jie Mountain. Unreal Engine Marketplace. https://www.unrealengine.com/marketplace/en-US/product/zhangjiajie-mountain

[20] Sun, J., Cao, W., Xu, Z. and Ponce, J. (2015) Learning a Convolutional Neural Network for Non-Uniform Motion Blur Removal. 2015 *IEEE Conference on Computer Vision and Pattern Recognition*, Boston, 07-12 June 2015, 769-777. https://doi.org/10.1109/CVPR.2015.7298677

[21] Kim T.H. and Lee, K.M. (2014) Segmentation-Free Dynamic Scene Deblurring, Computer Vision and Pattern Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, 23-28 June 2014, 2766-2773.