Video Matting with Convolutional LSTM

Jiahao Zhang
U6921098

Peng Zhang
U6921163

Hang Zhang
U6921112

Abstract

TODO

1. Introduction

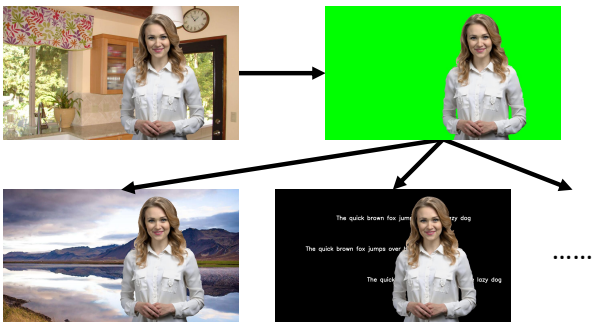

Figure 1. caption TODO.

background TODO + DEMO
motivation TODO
related works TODO

Semantic Segmentation TODO

Video Matting TODO

Convolutional LSTM TODO

Here summaries our main contributions,

- To our best knowledge, it is the first time to combinate Convolutional LSTM with Deeplab architect for video matting.
- We persent a possibility to apply video matting without background as input.

2. Problem Statement

Given an image i including one or more human figures (Foreground) f, we want to decompose it as

$$i = \alpha f + (1 - \alpha)b \tag{1}$$

where $alpha$ is the alpha matte, denoted by a greyscale image. We then define the video as a sequence of images (frames) $V = [i_1, i_2, \ldots, i_n]^T$, therefore the alpha matte A is also a sequence of α, so is the foreground F and background B.

$$V = AF + (1 - A)B \tag{2}$$

The alpha matte α works like a mask here, which means that during training, what we are actually doing is finding $i = \alpha i + (1 - \alpha)i$ with constrains $f = \alpha i$ and $b = (1 - \alpha)i$.

Our network G takes V as input and predicts alpha matte A', foreground F' and error map E' and hidden features H'. The model is fully convolutional therefore it accepts variant resolutions and aspect ratio.

3. Methods

In this section, we will introduce both model structure and loss functions for this project.

3.1. Model Structure

The model is inspired from the base part of [1], which can be divided into three parts as shown in Figure 2. One of the major differences is the input size. Instead of images with background ($B \times 6 \times H \times W$), we use video frames without background ($B \times T \times 3 \times H \times W$) where T refers to time. Another is we add ConvLSTM after each Decoder Block to obtain temporal features.

3.1.1 Encoder

The Encoder consists of four encoder blocks (EB), each contains a convolutional layer, a batch normalization layer and a ReLU activation layer. This architecture is taken from ResNet50 pretrained on ImageNet. It is by design that after each EB, the size of feature maps halves, and the number of channels increases in the order $[3, 64, 256, 512, 2048]$.

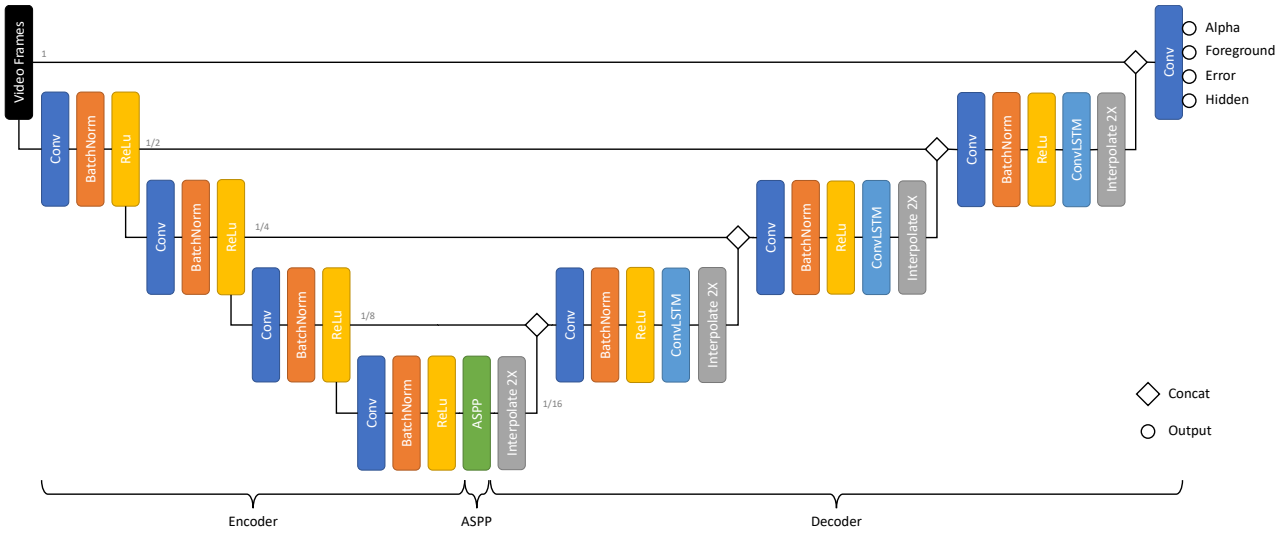

Figure 2. The architecture of our model.

3.1.2 ASPP

Atrous Spatial Pyramid Pooling (ASPP) utilizes the fusion of multiple convolutions with different dilation rates to increase receptive field, shrink the size of feature maps, just like pooling. But, meanwhile, it can keep more informative features.

3.1.3 Decoder

The decoder can be divided into three decoder blocks (DB) and an output block (OB), each of them starts with upsampling, implemented by 2x bilinear interpolation. Together with output of each EB, the feature map then pass a convolutional layer, a batch normalization layer, a ReLU activation layer and ConvLSTM layer. The output size of each DB doubles and the number of channels decreases in the order $[768, 384, 128, 51, 37]$. The OB generates the output directly, which contains 37 channels. The first channel is Alpha, which is the matte in greyscale. The next 3 channels are Foreground, it is designed to make the model focus more on the foreground. The next channel is Error, it is the predicted error, which will then be compared with the error between the predicted alpha and the ground truth alpha. The rest channels are set for the refine part of [1], which is not used in this project. But we still remain these channels for easier transfer learning.

3.1.4 ConvLSTM

$$
\begin{aligned}
\mathbf{i_t} &= \text{Sigmoid}\left(\text{Conv}\left(\mathbf{x_t}; \mathbf{w_{xi}}\right) + \text{Conv}\left(\mathbf{h_{t-1}}; \mathbf{w_{hi}}\right) + \mathbf{b_i}\right) \\
\mathbf{f_t} &= \text{Sigmoid}\left(\text{Conv}\left(\mathbf{x_t}; \mathbf{w_{xf}}\right) + \text{Conv}\left(\mathbf{h_{t-1}}; \mathbf{w_{hf}}\right) + \mathbf{b_f}\right) \\
\mathbf{o_t} &= \text{Sigmoid}\left(\text{Conv}\left(\mathbf{x_t}; \mathbf{w_{xo}}\right) + \text{Conv}\left(\mathbf{h_{t-1}}; \mathbf{w_{ho}}\right) + \mathbf{b_o}\right) \\
\mathbf{g_t} &= \text{Tanh} \quad \left(\text{Conv}\left(\mathbf{x_t}; \mathbf{w_{xg}}\right) + \text{Conv}\left(\mathbf{h_{t-1}}; \mathbf{w_{hg}}\right) + \mathbf{b_g}\right) \\
\mathbf{c_t} &= \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \mathbf{g_t} \\
\mathbf{h_t} &= \mathbf{o_t} \odot \text{Tanh}\left(\mathbf{c_t}\right)
\end{aligned}
\tag{3}
$$

TODO

3.2. Loss Function

The loss function is a combianation of three losses.

$$\mathcal{L}_\alpha = \|\alpha - \alpha^\star\|_1 + \|\nabla\alpha - \nabla\alpha^\star\|_1 \tag{4}$$

The first loss is a L1 loss between ground truth alpha $\alpha^\star$ and predicted alpha α, and their gradients (Sobel) defined as $\nabla\alpha$ and $\nabla\alpha^\star$ respectively. The gradients loss is added to learn about the contour of the matte.

Then we measure the difference between foreground by L1 loss as

$$\mathcal{L}_f = \|\alpha^* f' - \alpha^* f^*\|_1 \tag{5}$$

Finally, the difference of errors is computed with L2 loss to guide the model concentrate more on the error or alpha matte, e.g. the hair of the human figure.

$$\mathcal{L}_e = \|e' - e^*\|_1 \tag{6}$$

where e' is predicted by the model and $e^* = \alpha' - \alpha^*$.

The model $G(V) = (A', F', E', H')$ is then trained in average of frames on following loss,

$$\mathcal{L}_G = \frac{1}{n}\sum_{i=1}^{n}(\mathcal{L}_{\alpha_i} + \mathcal{L}_{f_i} + \mathcal{L}_{e_i}) \tag{7}$$

4. Experiments

This section describes the dataset, metrics we use, how we train the model, shows the results of experiment and discussion.

4.1. Experiment Setup

4.1.1 Datasets

The VideoMatte240K proposed by [1] takes too long to train (more than 24 hours for an epoch). So we use the subdataset of it and named it VideoMatte8K. It has 20 videos (7032 frames) as the training dataset and one video (928 frames) as the validation dataset. The Background dataset is the same one in [1]. Because the dataset is much smaller than the original one, inspired by [2], we did a lot of data augmentations on foreground, background and alpha matte at the same time to improve the robustness. Including static augmentation e.g. random affine, resize, horizontal flip, noise,color jitter, grayscale, sharpeness and blur, etc. and motion augmentation, mainly by easing functions list in Table 2. To be able to run more epochs with limited computational resources, we decided to random crop and resize all frames to 224×224.

4.1.2 Metrics

To measure the quantitative performance between the predicted alpha matte α' and ground truth α^*, we use the following four metrics proposed by [3].

MAD (Mean Absolute Difference)

$$\text{MAD}(\alpha', \alpha^*) = \|\alpha' - \alpha^*\|_1 \tag{8}$$

MSE (Mean Squared Error)

$$\text{MSE}(\alpha', \alpha^*) = \|\alpha' - \alpha^*\|_2^2 \tag{9}$$

GRAD (Spatial-Gradient Metric)

$$\text{GRAD}(\alpha', \alpha^*) = \frac{1}{1000}\sum_i (\nabla\alpha'_i - \nabla\alpha^*_i)^q \tag{10}$$

where we compute the sum of gradient difference on normalized alpha mattes. The α'_i and α^*_i are normalized from α' and α^* respectively. And the ∇ denotes the first-order Gaussian derivative filters with variance $\sigma = 1.4$. $q = 2$ following [1]. Then scale down 1000 times for better readability.

CONN (Connectivity)

$$\text{CONN}(\alpha', \alpha^*) = \frac{1}{1000}\sum_i \|\varphi(\alpha'_i, \Omega) - \varphi(\alpha^*_i, \Omega)\|_1 \tag{11}$$

The fewer differences does not always mean better performance, connectivity of predicted alpha matte is also essential. As shown in Equation 11, $\varphi(\alpha_i, \Omega)$ means the connectivity for pixel i in alpha matte α to the connected region source Ω. E.g. if $\varphi = 1$, the pixel it fully connected, $\varphi = 0$, the pixel is not connected at all. Instead of using power of p, L1 norm is used here. Then scale down 1000 times for better readability.

4.1.3 Implementation

Most code are adapted from the repositories on GitHub listed in Table 3. The whole training process ran on a laptop with 8-core CPU at 3.80GHz with 32GB memory and an external RTX 2070 GPU. Due to the limitation of computational power, we ran 10 epochs for each experiment and set batch size to 4 and sequence length to 8. In this setting, the average load of GPU memory is 7.6GB out of 8GB. The original FPS (Frame Per Second) is 30, we modified it to be 10 fps by jumping pass two frames for every frame during data augmentation. Each sequence shares a same background and each batch contains frames from different videos, as shown in Figure 4.

The optimizer we used is AdamW, which is a fixed version of Adam. The learning rate is $[1e^{-4}, 5e^{-4}, 5e^{-4}]$ for Encoder, ASPP and Decoder respectively.

Following the best practice and trade-off between speed and accuracy, we use native PyTorch automatic mixed precision.

Because in the implementation of PyTorch code, Conv2D and ConvLSTM need different input size. We managed to solve it by flatten the input first to $BT \times C \times H \times W$, feed through Conv2D layers, e.g. EB, then un-flatten back to $B \times T \times C \times H \times W$ for training ConvLSTM module, and flatten again for the output.

Training from scratch is difficult. We experienced a very unstable result when training from random weights (alpha matte is all black). So, we decided to apply transfer learning from fine-tuned pre-trained model provided by [1]. It is trained by both G_{base} and G_{refine} on the multiple datasets including completed VideoMatte240K. For transfering training, 372 out of 380 modules are transferred, the rest modules are the first convolutional layer (weight + bias) and three ConvLSTM layer (3 x (weight + bias)).

Each epoch takes around 45 minutes, plus two validation which takes extra 40 minutes, hence an experiment takes around 8 hours. The training loss is shown in Figure 5.

4.2. Experiment Results

To make comparsion, we did three experiments with same settings. With ConvLSTM is what we have proposed, Without ConvLSTM shares the same architecture but removes all ConvLSTM modules as shown in Figure 6. Original denotes the pretrained model by [1], which needs both image and background as input.

Table 1. caption TODO

	MAD	MSE	GRAD	CONN
Original	**2.04**	**0.94**	**0.11**	**102.40**
w/ ConvLSTM	5.18	3.73	7.65	259.67
w/o ConvLSTM	8.09	5.44	6.73	406.09

4.2.1 Discussion

TODO

4.2.2 Ablation Study

TODO + DEMO

5. Conclusion

Conclusion TODO

6. References

[1] Shanchuan Lin, Andrey Ryabtsev, Soumyadip Sengupta, Brian Curless, Steve Seitz, and Ira Kemelmacher-Shlizerman. Real-Time High-Resolution Background Matting. *arXiv:2012.07810 [cs]*, Dec. 2020. arXiv: 2012.07810 version: 1. 1, 2, 3, 4, 5

[2] Shanchuan Lin, Linjie Yang, Imran Saleemi, and Soumyadip Sengupta. Robust High-Resolution Video Matting with Temporal Guidance. *arXiv:2108.11515 [cs]*, Aug. 2021. arXiv: 2108.11515. 3, 5

[3] Christoph Rhemann, Carsten Rother, Jue Wang, Margrit Gelautz, Pushmeet Kohli, and Pamela Rott. A perceptually motivated online benchmark for image matting. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1826–1833, 2009. 3

[4] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. *arXiv:1506.04214 [cs]*, Sept. 2015. arXiv: 1506.04214. 5

7. Review

7.1. Self Reflection

TODO

7.2. Confidential Peer Review

TODO

In doing this project, to the best of my judgement, I confirm that Jiahao Zhang mainly contributed to TODO, and his/her overall contribution is about 34%, Peng Zhang mainly worked on TODO, and his/her contribution is about 33%, and Hang Zhang was responsible for TODO, and his/her contribution counts about 33% of the total project workload.

A. Appendix

Table 2. Easing functions used for motion data augmentation.

LinearInOut	BackEaseIn
BackEaseOut	BackEaseInOut
BounceEaseIn	BounceEaseOut
BounceEaseInOut	CircularEaseIn
CircularEaseOut	CircularEaseInOut
CubicEaseIn	CubicEaseOut
CubicEaseInOut	ExponentialEaseIn
ExponentialEaseOut	ExponentialEaseInOut
ElasticEaseIn	ElasticEaseOut
ElasticEaseInOut	QuadEaseIn
QuadEaseOut	QuadEaseInOut
QuarticEaseIn	QuarticEaseOut
QuarticEaseInOut	QuinticEaseIn
QuinticEaseOut	QuinticEaseInOut
SineEaseIn	SineEaseOut
SineEaseInOut	

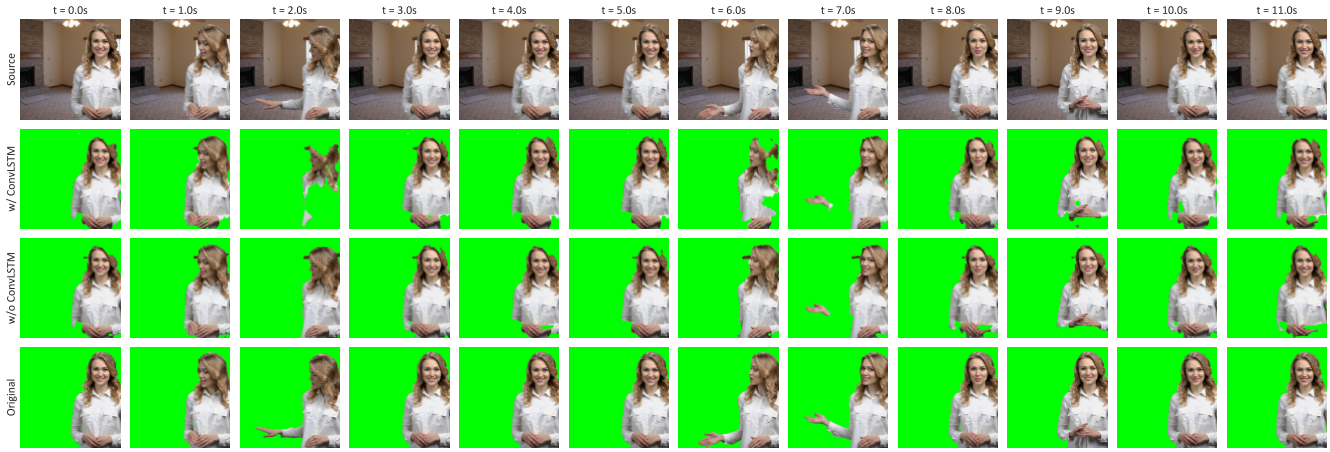

Figure 3. Visualization of matting result at different frame across three models.

Table 3. GitHub repositories used for this project.

Repository	Description
Real-Time High-Resolution Background Matting [1]	Basic model and project structure
Robust High-Resolution Video Matting with Temporal Guidance [2]	Data augmentation, loader and metrics
Convolutional LSTM Network [4]	ConvLSTM module

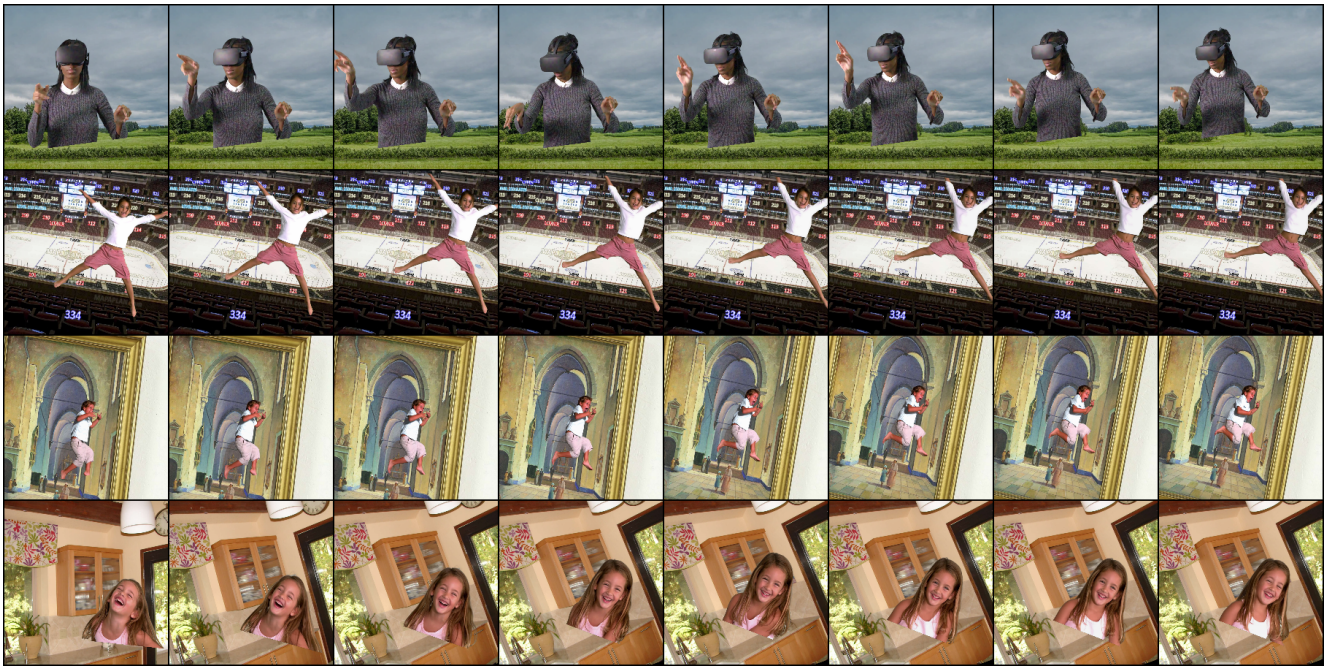

Figure 4. Demonstration of a batch.

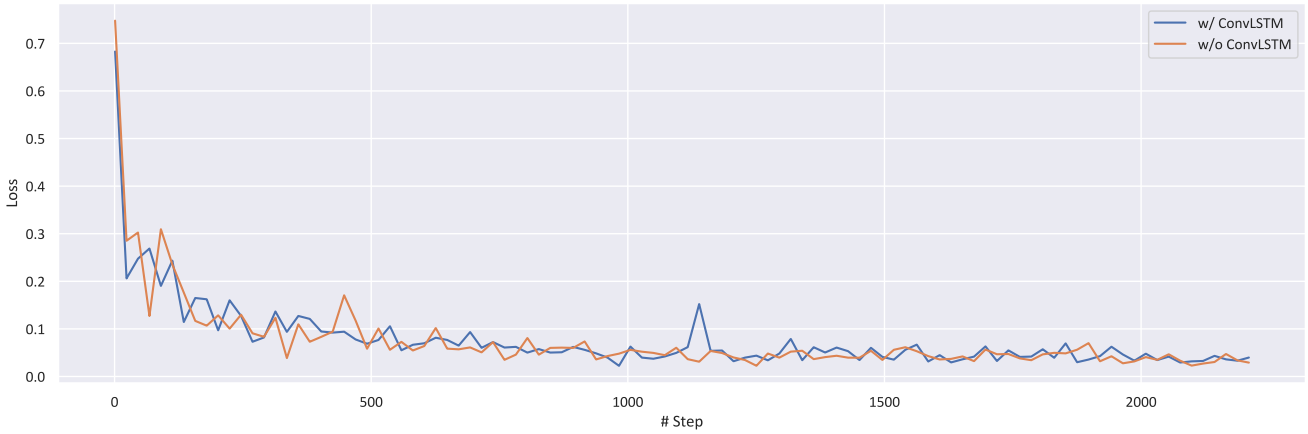

Figure 5. Training loss.

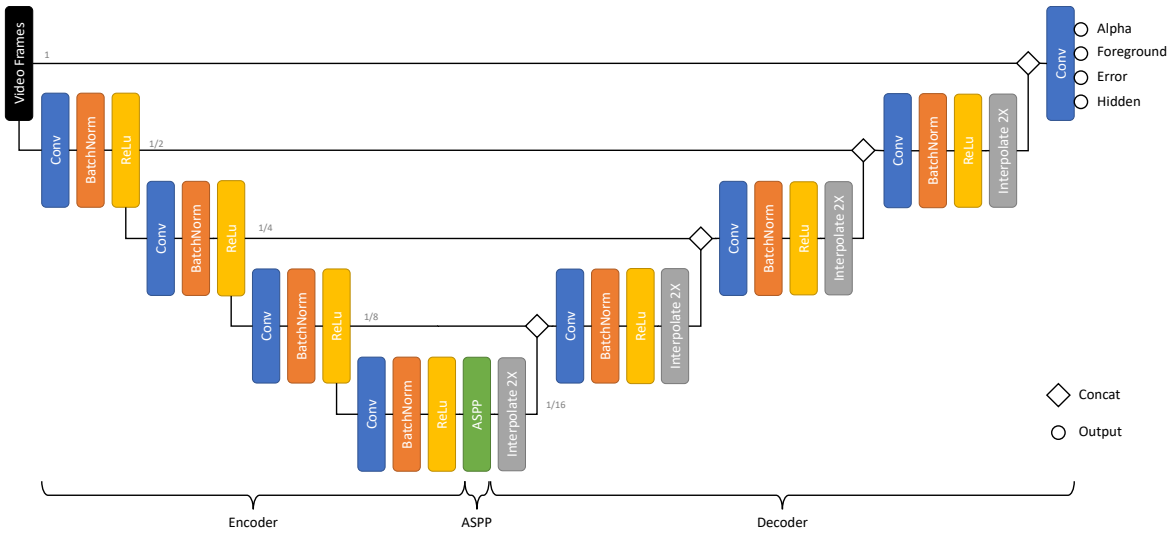

Figure 6. Model architcure of Without ConvLSTM.