Korea Advanced Institute of Science and Technology

School of Electrical Engineering

EE838B Special Topics in Image Engineering

Advanced Image Restoration and Quality Enhancement

Student's Name: Dinh Vu

Student's ID: 20184187

Homework 4

1. Dataset

Because the given dataset consists 5 videos in the MPEG-4 format, these videos must be split into frames. After that, not all of frames are used, the frames, which are transition between two adjacent scenes or containing many artefacts, are discarded. Then, each video is divided into scenes and each scene includes frames having reasonable quality with noticeable motion.

All above process is operated in Matlab with 2 files: `ExtractFrame.m` and `SceneSeparation.m`. The frames at the beginning and ending of each scene in each video is written in 5 files: `video1.csv`, `video2.csv`, `video3.csv`, `video4.csv`, `video5.csv` in the folder `data`.

2. Network Architecture

The structure of network is shown in Figure 2.1. It can be seen that the backbone of the network is based on U-net with 4 skip connection between encoder and decoder.

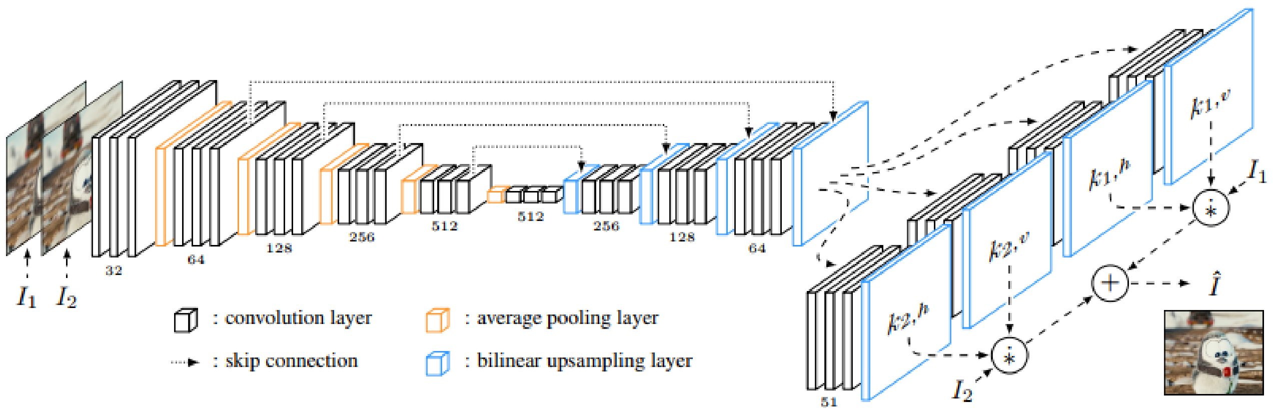


Figure 2.1. The network architecture [1]

The most unique and importance component in the architecture is local convolution ($*$), presented in Figure 2.2. After the last bilinear upsampling layer, there are 4 kernels having size of H×W×51, where H×W is the size of input network I_1 and I_2. Each pixel is corresponding with a vertical kernel and a horizontal kernel, which are taken from k1v, k1h if the pixel is in frame I_1 and from k2v, k2h if the pixel is in frame I_2. The local convolution, actually not proper name, is summation of element-wise product between the kernel and the corresponding pixels in the frame. When calculating the next pixel, the new horizontal and vertical kernel are applied, there is not any sliding window, so that is why this separable convolution not usual convolution.

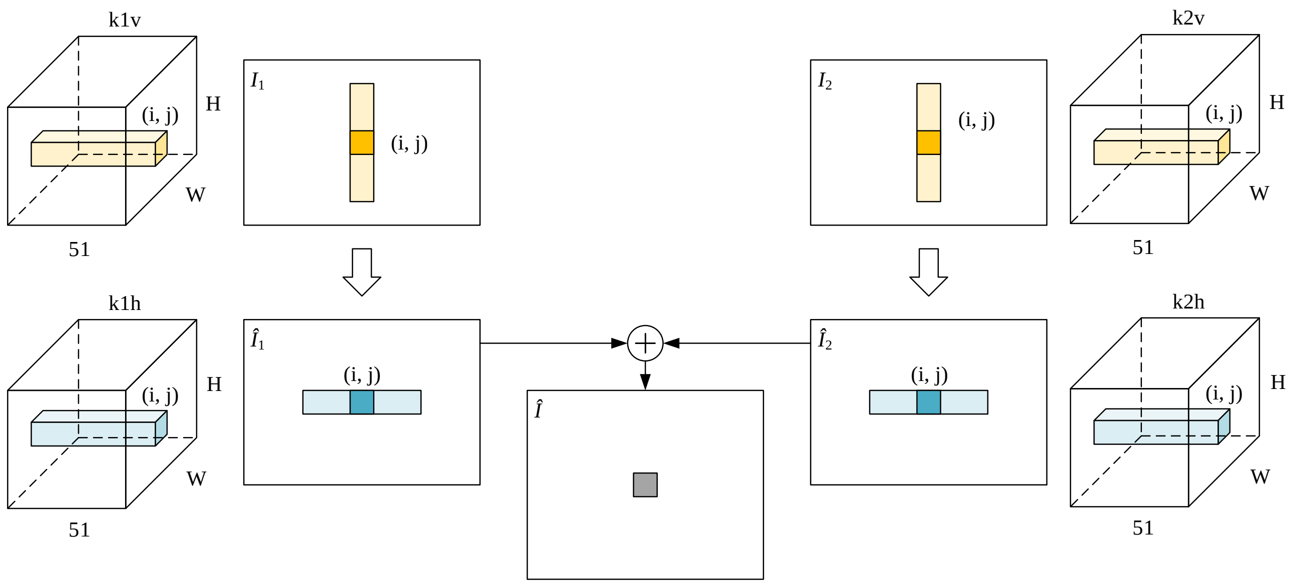


Figure 2.2. The local convolution

When implementing in Python, the zero-padding and parallel programming must be considered because this separable convolution consumes most of computation time due to high resolution of the input frames. For the sake of convenience explanation, let assume that the kernel length is 7 instead of 51, and 7 continuous pixels in one line must be determined horizontally.

1	2	3	4	5	6	7

0	0	0	1	2	3	4	5	6	7

First, 3 zero-pixels are padded in the left of pixel 1. To calculate the pixel from 1 to 7, the pixels can be re-arranged from space to depth, displayed in Figure 2.3. It is consequently that the image need to be shifted 7 times with stride 1, then concatenate together to get the shape in Figure 2.3b. This process is repeated with the vertical direction.

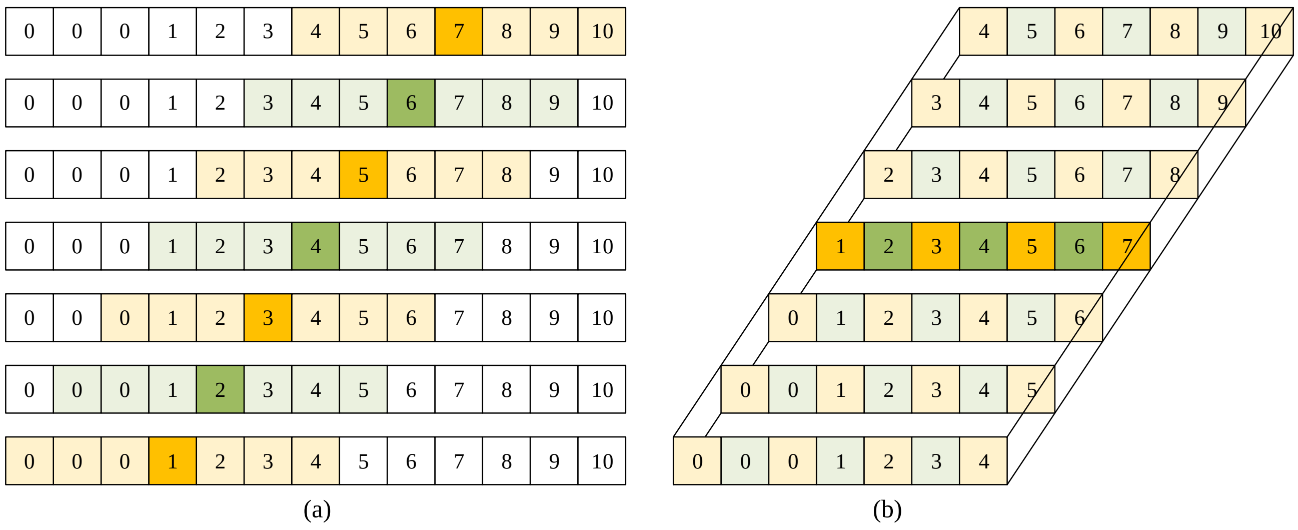


Figure 2.3. The example of parallel programming for local convolution

In order to expand for kernel length 51, and 2-D images, the frame must be zero-padding with 25 pixels in the top for vertical and 25 pixels in the left for horizontal by using `tf.pad()` function. The shifted frames are generated by `tf.manip.roll()` function. Figure 2.4 presents the shifted images for local convolution.

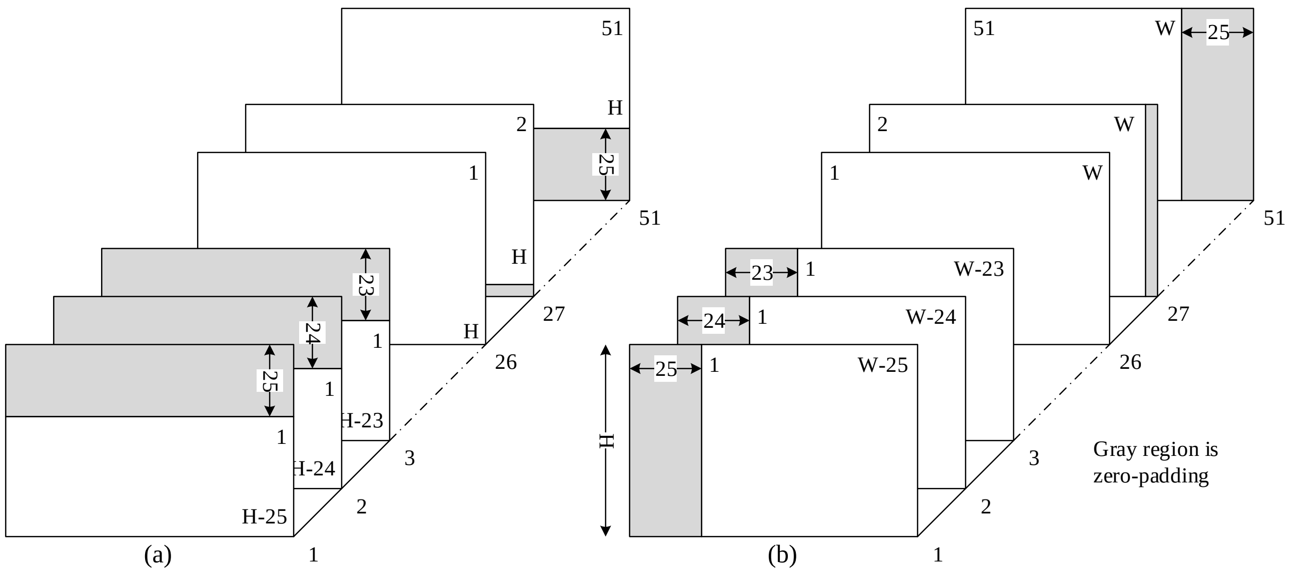
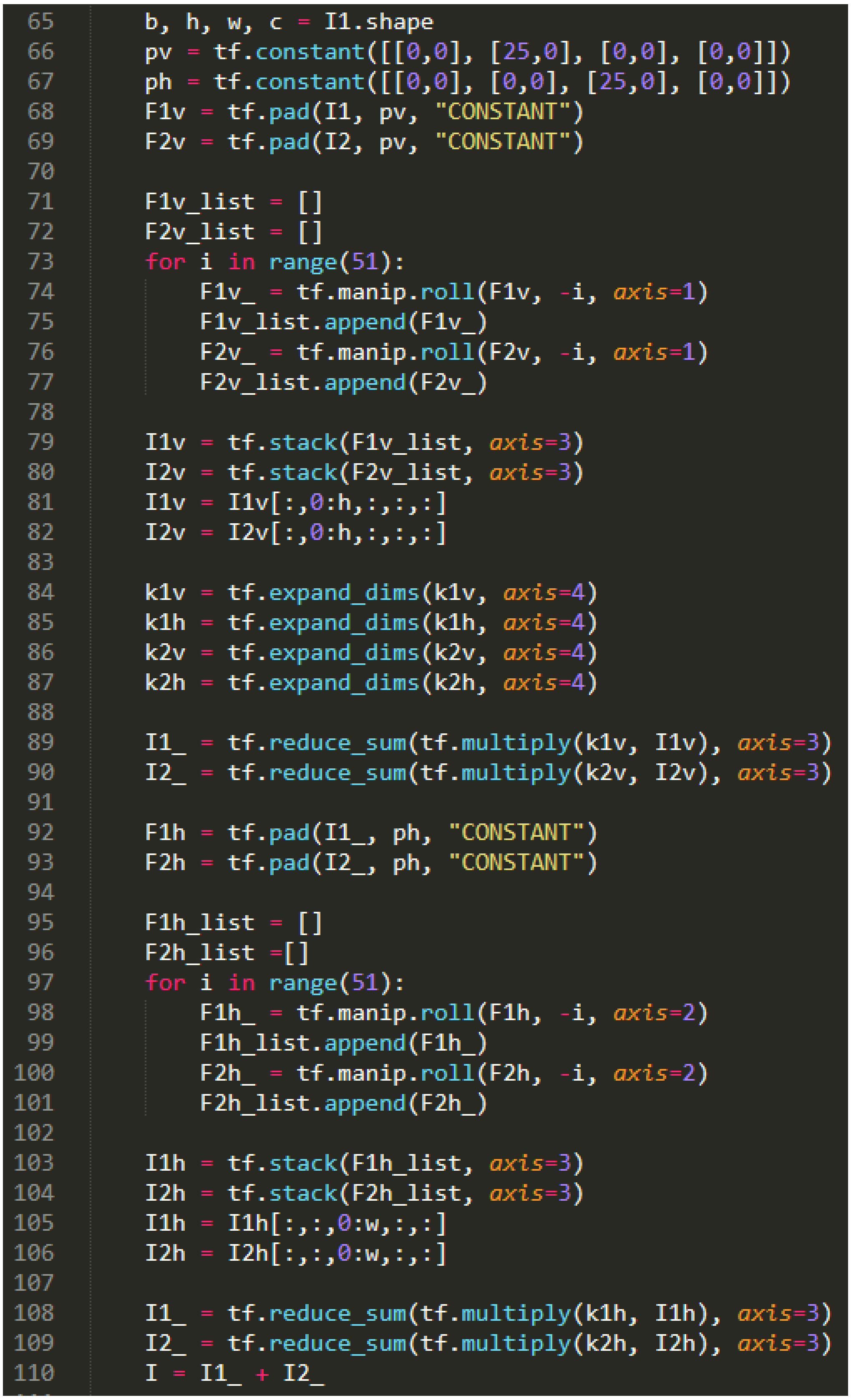


Figure 2.4. The example of creating shift images for horizontal direction

The implementation of the local separable convolution is written from line 65-110 in `model.py` file which also presented in Figure 2.5 below.

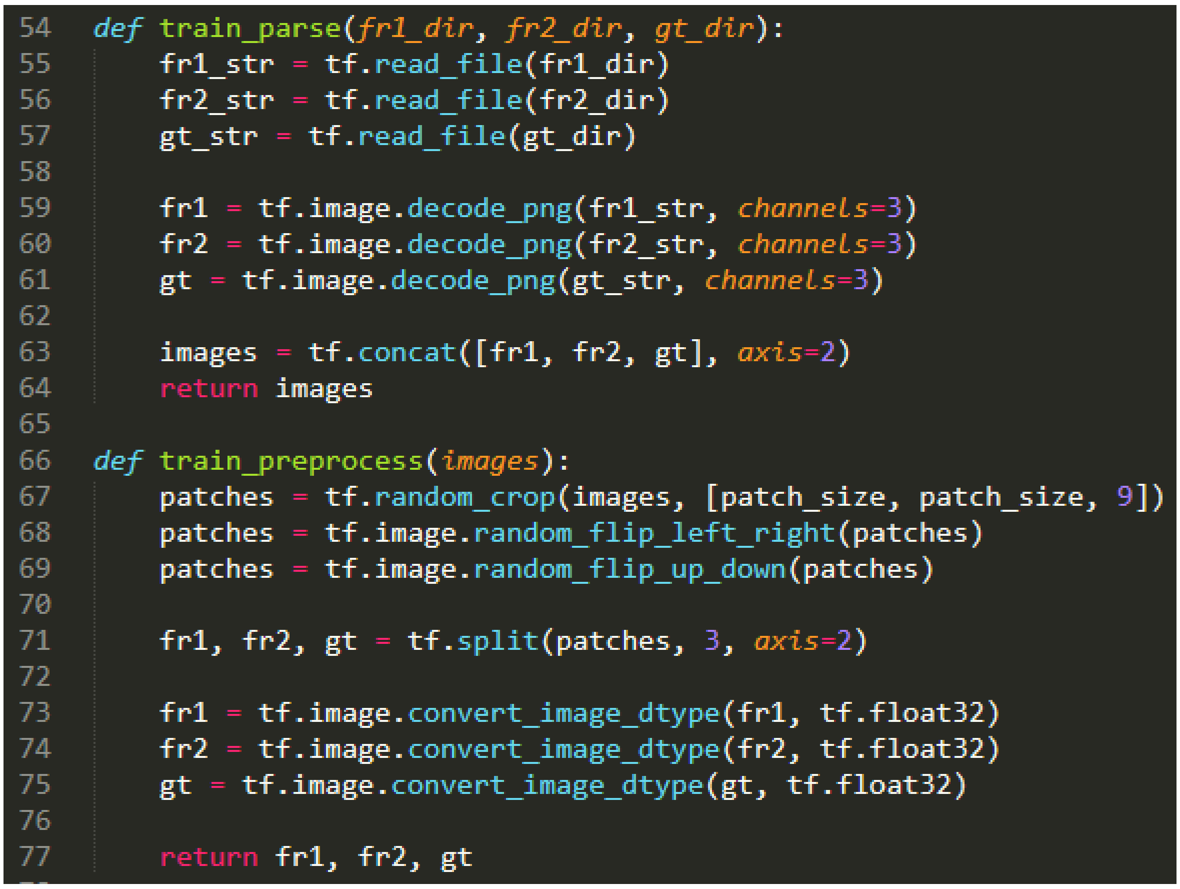


```python
    b, h, w, c = I1.shape
    pv = tf.constant([[0,0], [25,0], [0,0], [0,0]])
    ph = tf.constant([[0,0], [0,0], [25,0], [0,0]])
    F1v = tf.pad(I1, pv, "CONSTANT")
    F2v = tf.pad(I2, pv, "CONSTANT")

    F1v_list = []
    F2v_list = []
    for i in range(51):
        F1v_ = tf.manip.roll(F1v, -i, axis=1)
        F1v_list.append(F1v_)
        F2v_ = tf.manip.roll(F2v, -i, axis=1)
        F2v_list.append(F2v_)

    I1v = tf.stack(F1v_list, axis=3)
    I2v = tf.stack(F2v_list, axis=3)
    I1v = I1v[:,0:h,:,:,:]
    I2v = I2v[:,0:h,:,:,:]

    k1v = tf.expand_dims(k1v, axis=4)
    k1h = tf.expand_dims(k1h, axis=4)
    k2v = tf.expand_dims(k2v, axis=4)
    k2h = tf.expand_dims(k2h, axis=4)

    I1_ = tf.reduce_sum(tf.multiply(k1v, I1v), axis=3)
    I2_ = tf.reduce_sum(tf.multiply(k2v, I2v), axis=3)

    F1h = tf.pad(I1_, ph, "CONSTANT")
    F2h = tf.pad(I2_, ph, "CONSTANT")

    F1h_list = []
    F2h_list =[]
    for i in range(51):
        F1h_ = tf.manip.roll(F1h, -i, axis=2)
        F1h_list.append(F1h_)
        F2h_ = tf.manip.roll(F2h, -i, axis=2)
        F2h_list.append(F2h_)

    I1h = tf.stack(F1h_list, axis=3)
    I2h = tf.stack(F2h_list, axis=3)
    I1h = I1h[:,:,0:w,:,:]
    I2h = I2h[:,:,0:w,:,:]

    I1_ = tf.reduce_sum(tf.multiply(k1h, I1h), axis=3)
    I2_ = tf.reduce_sum(tf.multiply(k2h, I2h), axis=3)
    I = I1_ + I2_
```

Figure 2.5. The implementation of the local separable convolution in tensorflow

3. Training

Before training, two consecutive frames are read from disk then directly random crop to take 2 patches 128×128. After that, the patches are random flip horizontally and vertically for data augmentation. The pixel values are re-scale from 0 to 1 while the pixel data type is converted from `uint8` to `float32`. This training preprocess is implemented in line 54-77 in `train.py` file, which also is displayed in Figure 3.1.



```
54  def train_parse(fr1_dir, fr2_dir, gt_dir):
55      fr1_str = tf.read_file(fr1_dir)
56      fr2_str = tf.read_file(fr2_dir)
57      gt_str = tf.read_file(gt_dir)
58  
59      fr1 = tf.image.decode_png(fr1_str, channels=3)
60      fr2 = tf.image.decode_png(fr2_str, channels=3)
61      gt = tf.image.decode_png(gt_str, channels=3)
62  
63      images = tf.concat([fr1, fr2, gt], axis=2)
64      return images
65  
66  def train_preprocess(images):
67      patches = tf.random_crop(images, [patch_size, patch_size, 9])
68      patches = tf.image.random_flip_left_right(patches)
69      patches = tf.image.random_flip_up_down(patches)
70  
71      fr1, fr2, gt = tf.split(patches, 3, axis=2)
72  
73      fr1 = tf.image.convert_image_dtype(fr1, tf.float32)
74      fr2 = tf.image.convert_image_dtype(fr2, tf.float32)
75      gt = tf.image.convert_image_dtype(gt, tf.float32)
76  
77      return fr1, fr2, gt
```

Figure 3.1. The training preprocess

The training process was operated in a computer with an Intel(R) Core(TM) i7 CPU @ 4.20 GHz and a GPU NVIDIA GTX GeForce 960Ti. The total number of epochs is 50 and the batch size equals to 16. The initial learning rate is 0.001 during the first 20 epochs, then the learning rate is decreased by half after every 10 epochs. In order to see the impact of perceptual loss, from scratch, the network was trained with only L1 loss and L1 loss combining with perceptual loss separately with the same above configuration.

3.1. *The L1 Loss*

The training with only L1 loss consumed more than 32 hours to converge. The L1 loss is defined in the following equation.

$$\mathcal{L}_1 = \left\|\hat{I} - I_{gt}\right\|_1$$

Where, $\hat{I}$ and I_{gt} are the interpolation frame and the ground truth respectively. Figure 3.2 below show the L1 loss value during the training process.

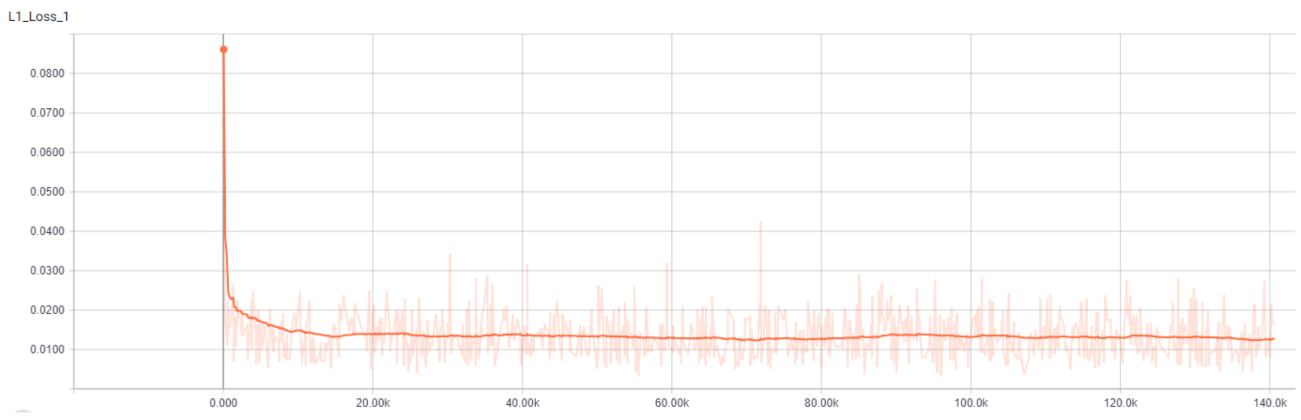


Figure 3.2. The L1 loss during training with only L1

3.2. The Combined Loss

The training process with L1 loss and perceptual loss consumed more than 38 hours to converge longer than just using L1 loss. The perceptual loss and combined loss are defined in the following equation.

$$\mathcal{L}_F = \left\|\phi(\hat{I}) - \phi(I_{gt})\right\|_2^2$$

$$\mathcal{L}_C = \mathcal{L}_1 + \gamma\mathcal{L}_F$$

Where:

- $\phi()$ is the extracted features of the image from `relu4_4` layer in VGG-19 network
- $\mathcal{L}_C$ is the combined loss with $\gamma = 10^{-4}$

Figure 3.3, 3.4 and 3.5 shows the value of 3 different loss function during the training process.

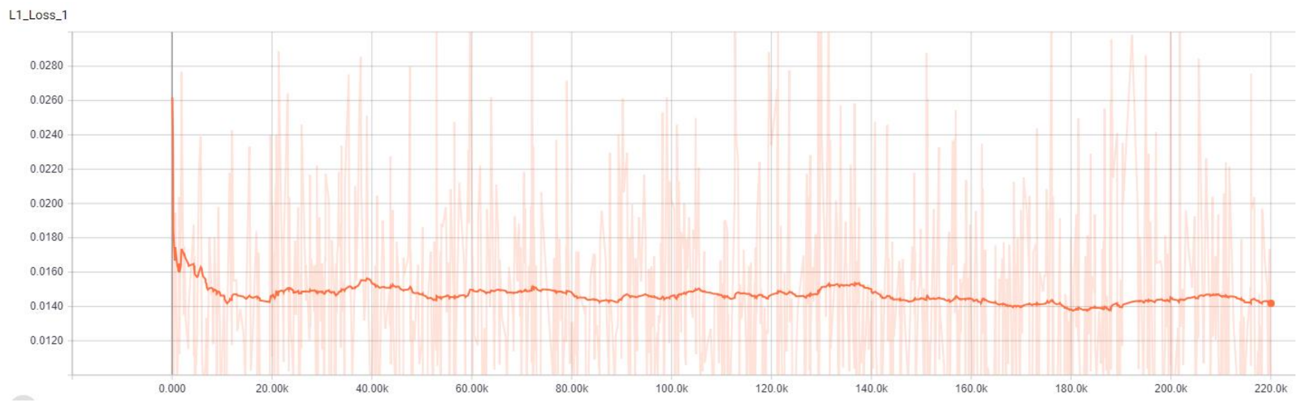


Figure 3.3. The L1 loss during training with the combined loss

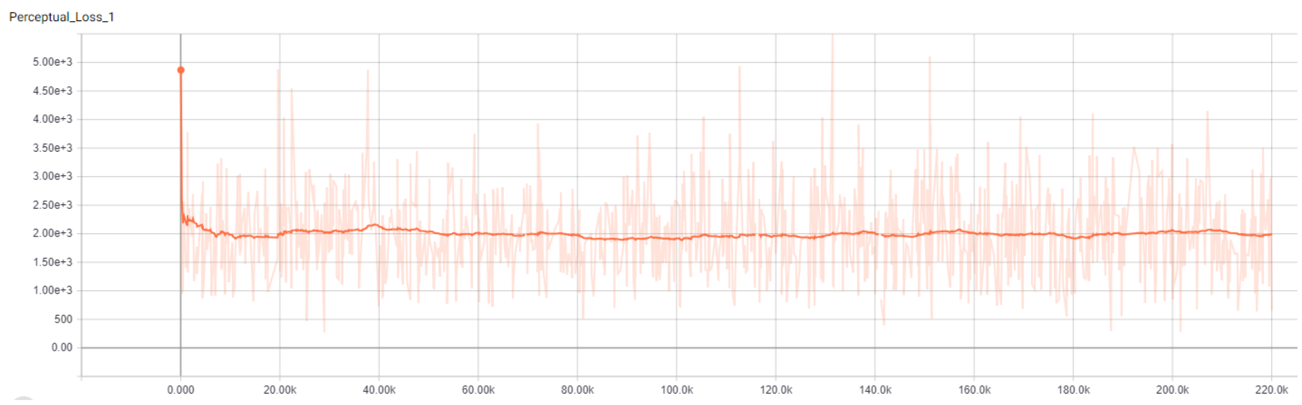


Figure 3.4. The perceptual loss during training with the combined loss

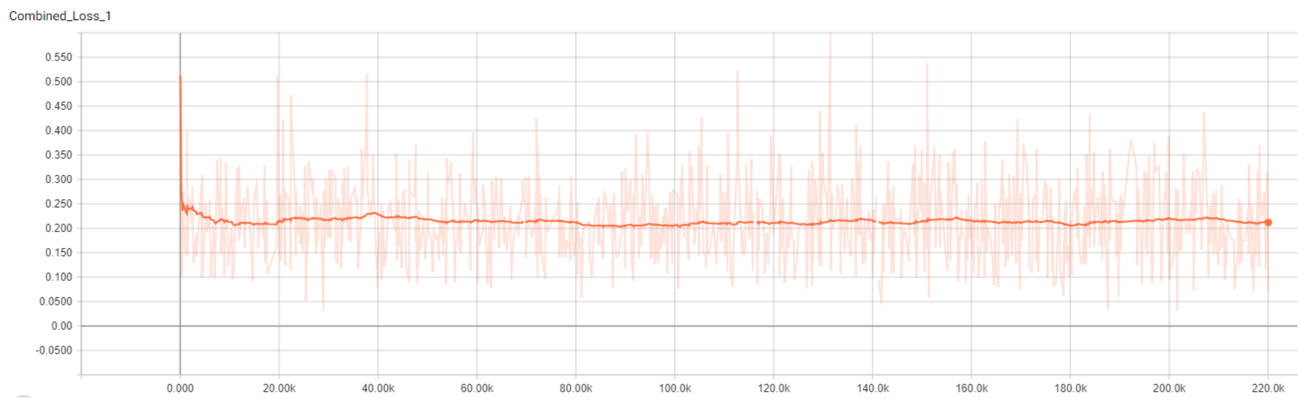
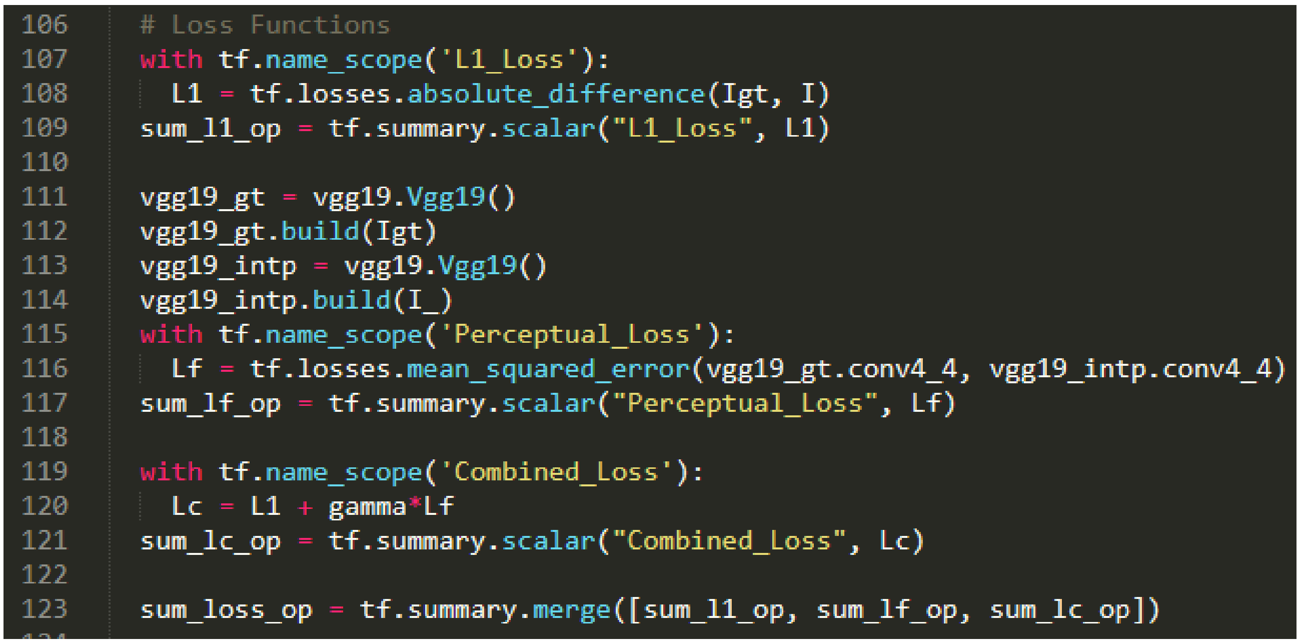


Figure 3.5. The combined loss during training

The implementation of 3 loss functions above is presented from line 106 to 123 in `train.py` file and shown in Figure 3.6 below.



```
106  # Loss Functions
107  with tf.name_scope('L1_Loss'):
108    L1 = tf.losses.absolute_difference(Igt, I)
109  sum_l1_op = tf.summary.scalar("L1_Loss", L1)
110
111  vgg19_gt = vgg19.Vgg19()
112  vgg19_gt.build(Igt)
113  vgg19_intp = vgg19.Vgg19()
114  vgg19_intp.build(I_)
115  with tf.name_scope('Perceptual_Loss'):
116    Lf = tf.losses.mean_squared_error(vgg19_gt.conv4_4, vgg19_intp.conv4_4)
117  sum_lf_op = tf.summary.scalar("Perceptual_Loss", Lf)
118
119  with tf.name_scope('Combined_Loss'):
120    Lc = L1 + gamma*Lf
121  sum_lc_op = tf.summary.scalar("Combined_Loss", Lc)
122
123  sum_loss_op = tf.summary.merge([sum_l1_op, sum_lf_op, sum_lc_op])
```

Figure 3.6. The implementation of the loss functions

4. Validation

The validation frames are read from disk then converted from `uint8` to `float32` while pixel values are scaled from 0 to 1. These frames are push to the model to generate the intermediate frames. All validation process is written in `valid.py` file. Table 4.1 and 4.2 shows the validation results of two models.

Table 4.1. The experimental results of the model trained with only L1 loss

Images	L1 Loss	PSNR (dB)	SSIM	MS-SSIM
val_00739	0.0207	26.98	0.7975	0.9131
val_01639	0.0022	38.07	0.9859	0.9934
val_03382	0.0120	30.32	0.8994	0.9327
Average	0.0115	31.87	0.8944	0.9460

Table 4.2. The experimental results of the model trained with the combined loss

Images	L1 Loss	Perceptual Loss	Combined Loss	PSNR (dB)	SSIM	MS-SSIM
val_00739	0.0235	1346.32	0.1581	26.33	0.7864	0.9040
val_01639	0.0024	379.48	0.0404	37.03	0.9851	0.9923
val_03382	0.0139	1633.18	0.1772	29.45	0.8868	0.9192
Average	0.0132	1122.97	0.1255	31.02	0.8867	0.9374

It is noticeable to see that four qualified criterions L1 loss, PSNR, SSIM and MS-SSIM of the combined loss model is lower than L1 loss model. However, the difference of these criteria between two models are not significant and obviously, it does not mean that the combined loss function is not better than L1 loss function. Looking at Figure 4.1 and 4.2 below, the combined loss model generated the intermediate frames less blurred than the L1 loss model.

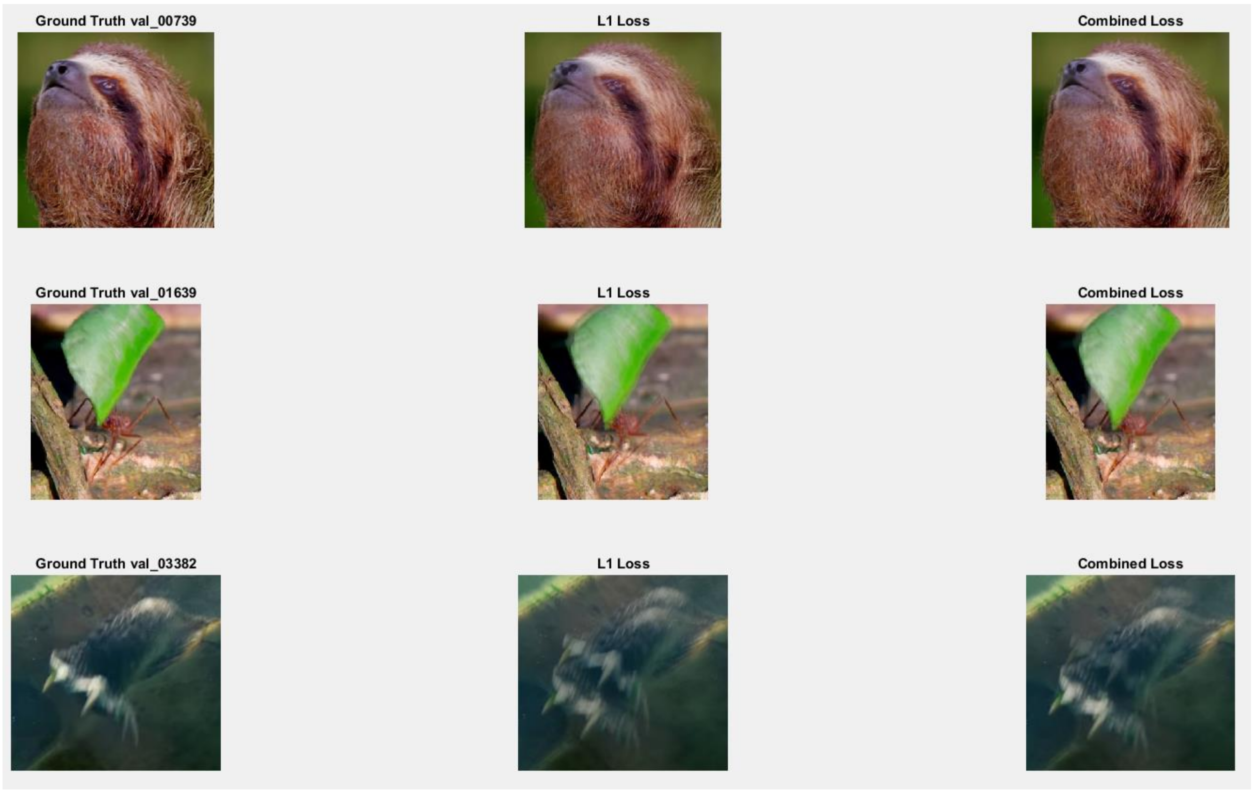


Figure 4.1. The comparison perceptual quality of three required examples

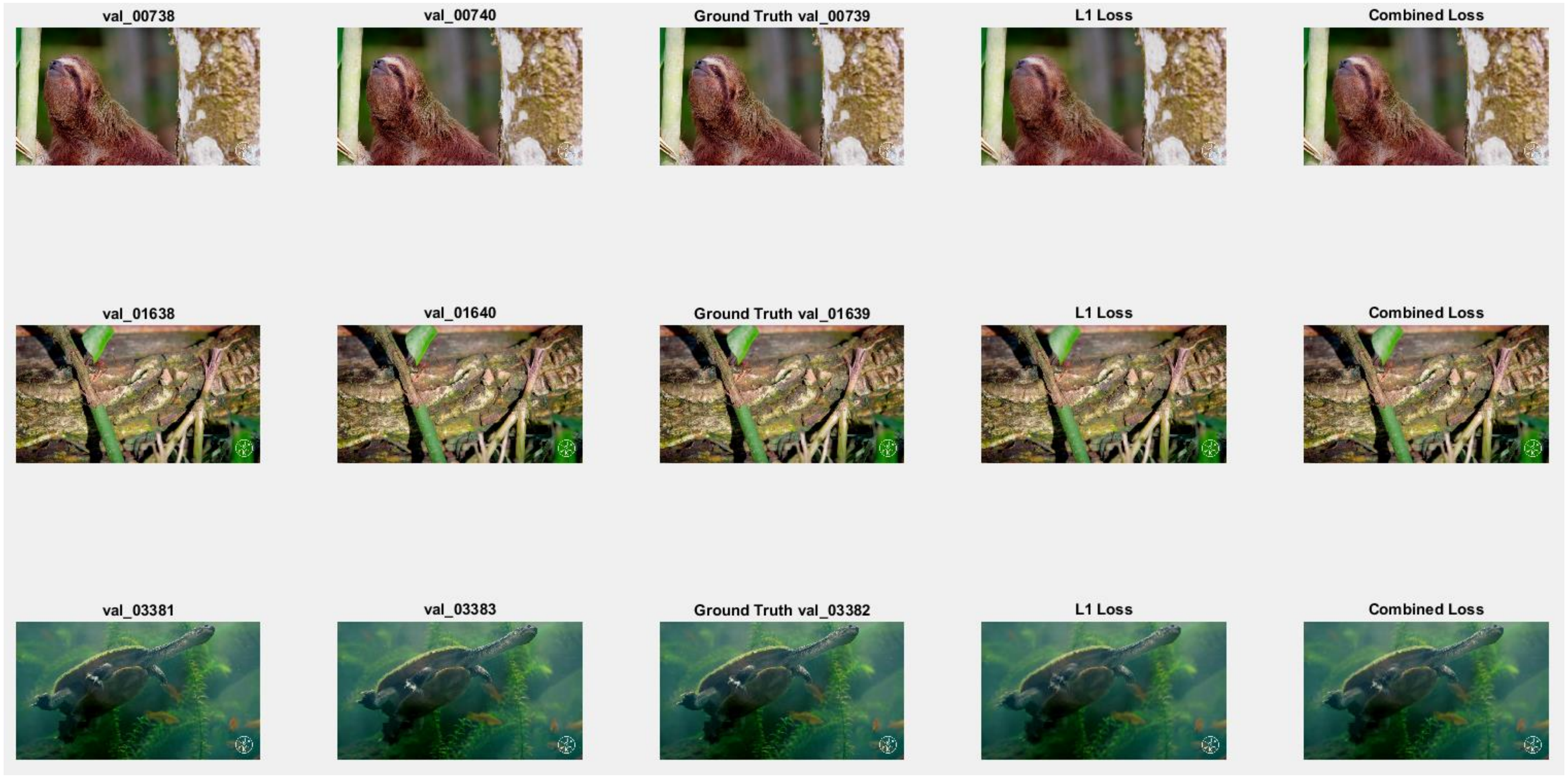


Figure 4.2. Three required validation examples

5. Test

In the real application, there is not any ground truth so PSNR, SSIM, MS-SSIM cannot be calculated. Therefore, in `test.py` file, only generation code is implemented.

Reference

[1] Simon, Niklaus, Long Mai and Feng Liu, "Video Frame Interpolation via Adaptive Separable Convolution", *IEEE International Conference on Computer Vision (ICCV),* October 2017.