Video Super Resolution with Integrated 2D and 3D Convolution Neural Network

Member :
CheukHin HO, The Chinese University of Hong Kong
JiaXin LI, The Chinese University of Hong Kong

Mentor :
Dr. Kwai Wong, Joint Institute of Computational Sciences

Abstract

In most deep-learning based video super-resolution algorithms, a 2D CNN model is applied to each frame for image super-resolution and then a sequence of the super-resolved frames is taken as the input of 3D CNN.

However, in each frame, the temporal information which is useful to super-resolve the middle frame is different. As a result, using a single 2D model to pre-process all the frames in the sequence may not preserve the diverse information in each frame. Therefore, the idea of pre-processing each frame by different 2D CNN occurred to us.

In our video super-resolution model, each frame goes through a different 2D CNN model for single-image super-resolution and temporal information extraction. Then, the processed frames are packed as a sequence of images and put into the 3D model.

Further, we have proposed two candidates for the 2D CNN algorithms. We compare the limits and advantages of each candidate and try to find the better structure as well as gain some insight into the exact function of 2D CNN.

codes are available at https://bitbucket.org/cheukhinho/3dcnn-super-2021-pytorch/src/master/

Objective

Contemporary physical observation and simulation produces a large amount of images and videos. Due to technology limitations, not all of them are in high resolution. Our objective is to design a method to do video super-resolution on these kinds of data. First, we adopt climate data to compare our models with previous ones; then diamond video data is used to adjust our model to adapt more complex cases.

Definitions

In this section, the definition of some tools for measuring the performance of the model is introduced. Given a ground truth $m * n$ image $I(x, y)$, suppose $K(x, y)$ is the approximation of I, then we have the following definitions:

i. Mean squared error (MSE)

$$MSE = \frac{1}{mn}\sum_{i=0}^{m}\sum_{j=0}^{n} I(i,j) - K(i,j)$$

which measures how close the pixel-wise difference is. The lower the MSE, the better the approximation.

ii. Peak Signal to Noise Ratio (PSNR)

$$PSNR = 10log_{10}(\frac{MAX_I^2}{MSE})$$

where MAX_I is the maximum possible pixel value. PSNR is a measure ratio in how close does pixel values of $K(x, y)$is close to $I(x, y)$. The higher the PSR, the better the approximation.

iii. Structure Similarity Index(SSIM)

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(2\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

where $0 \leq SSIM(x, y) \leq 1$; and

μ_x and μ_y are average of x and y

σ_x^2 and σ_y^2 are varience of x and y

σ_{xy} is the covariance of x and y

c_1 and c_2 are constants to prevent division with small denominator

The higher the SSIM, the better the approximation.

Model Idea/structure/explanation

5-parallel

The model 5-parallel has two parts: the first part consists of 5 2D convolution layers in parallel and the second part consists of a stack of 3D convolution layers.

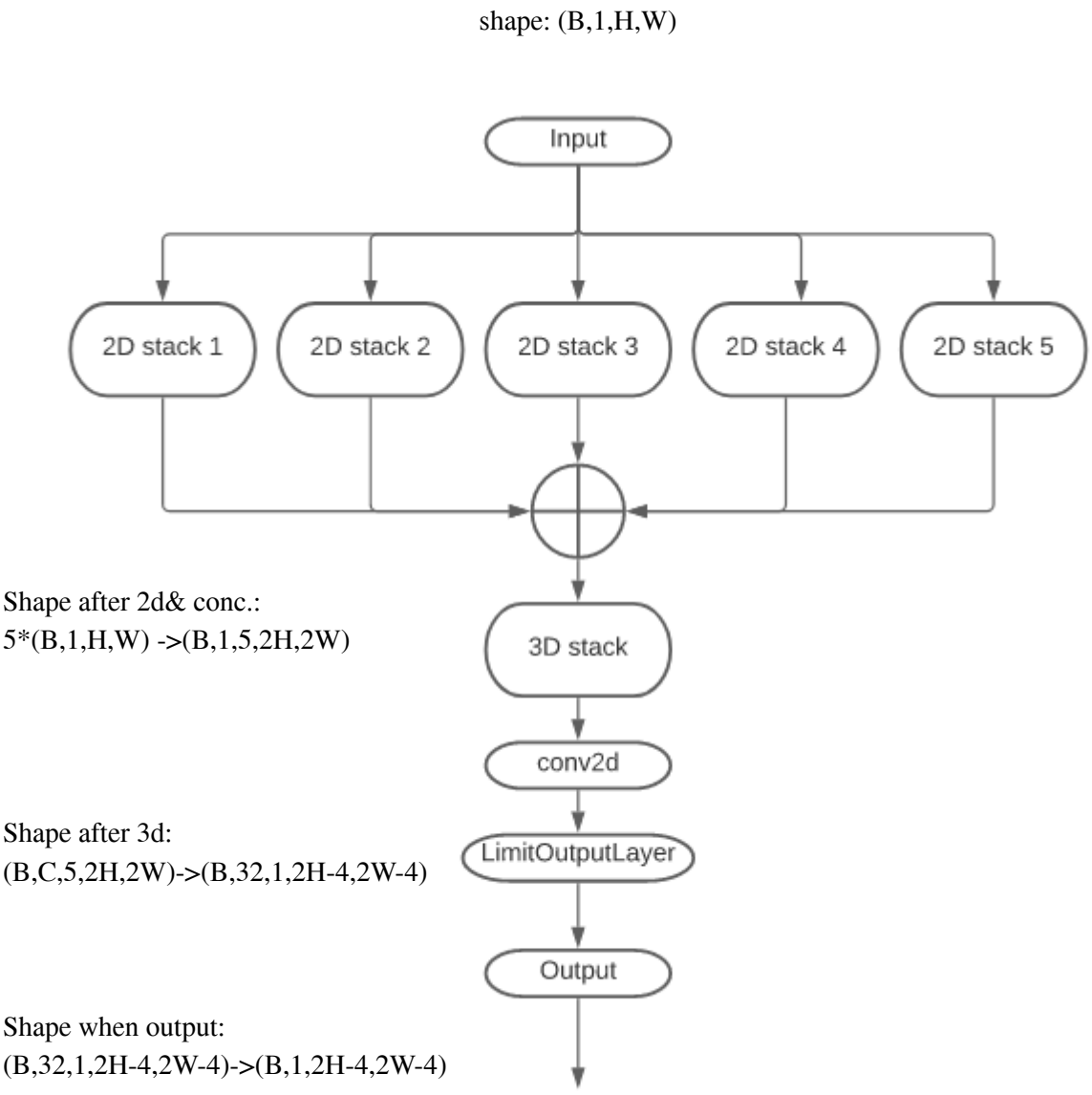

Figure : Model Structure
(B: bach size, C: channel, H: height, W: width)

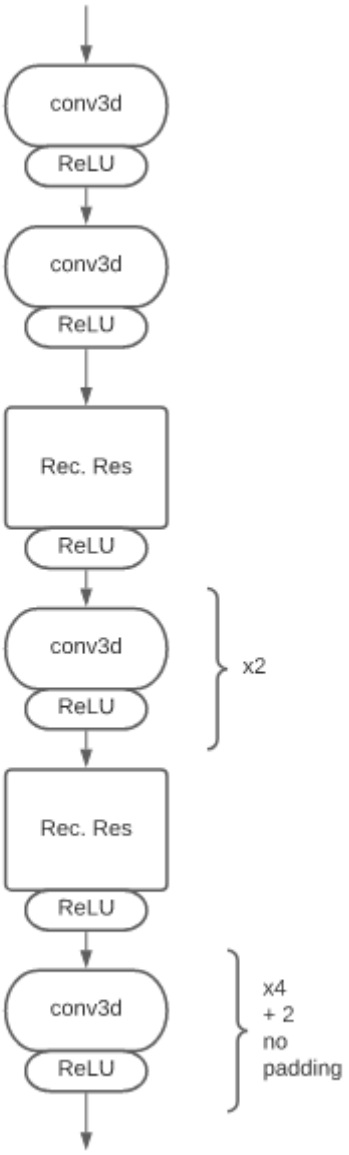

Figure : Structure of 3D stack

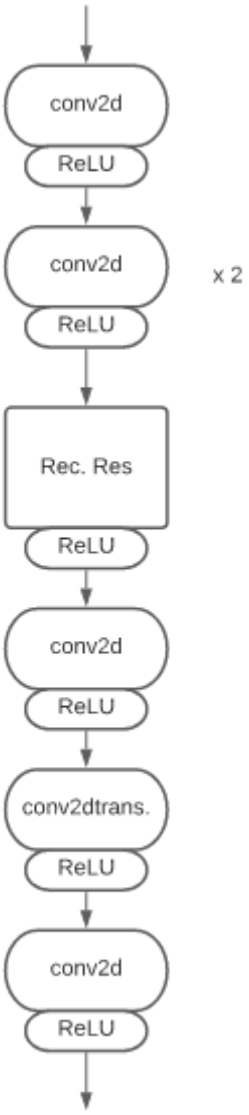

Figure : Structure of 2D stacks

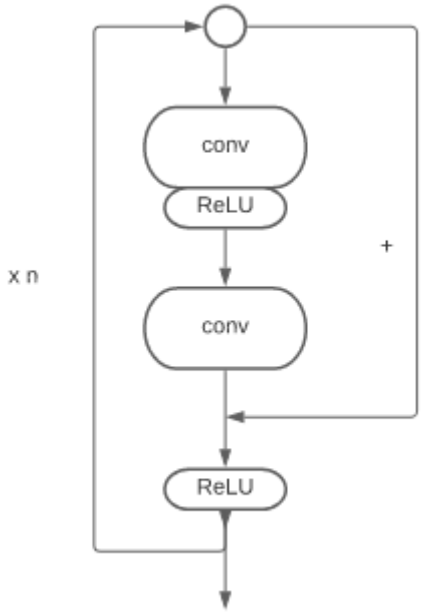

Figure: Structure of Recurrent Res Block for 2d and 3d
(n : number of recurrent)

For every input stack of 5 frames, it splits and each frame goes through different 2D stacks for independent 2D feature extraction of each frame. In particular, there is a small recursive block in the middle of each 2D stack. The 5 outputs are then concatenated together and fed into the 3D part. The 3D part gathers the information obtained from 2D parts (of each frame) to perform superresolution. Similarly, there are also two small 3D recursive blocks in the middle of the 3D stack. Throughout 2D and 3D parts, ReLU layers are used to add non-linearity to the model after each convolution layer. The output from the 3D part passes through a layer of conv2d and a output layer with the modified sigmoid function :

$$S(x) = (MAX - MIN) * sigmoid(x/100) + MIN$$

where MAX and MIN are the maximum and minimum of each entry of the image data.

Through training and back propagation while updating the weight of the network, we expect the 3D part to interact with the 2D part and help each other to learn to super resolve the image. Upon testing, we set $n_{2d} = n_{3d} = 5$.

Delta

Similar to 5-parallel, the Delta model is also composed of 2D convolution layers and 3D convolution stack.

The biggest difference between 5-parallel and Delta lies in the design of the 2D convolution part. Instead of using 5 parallel 2D convolution networks, the Delta model achieves the target of processing each frame differently in another manner to reduce the number of parameters. The idea is illustrated by the picture below.

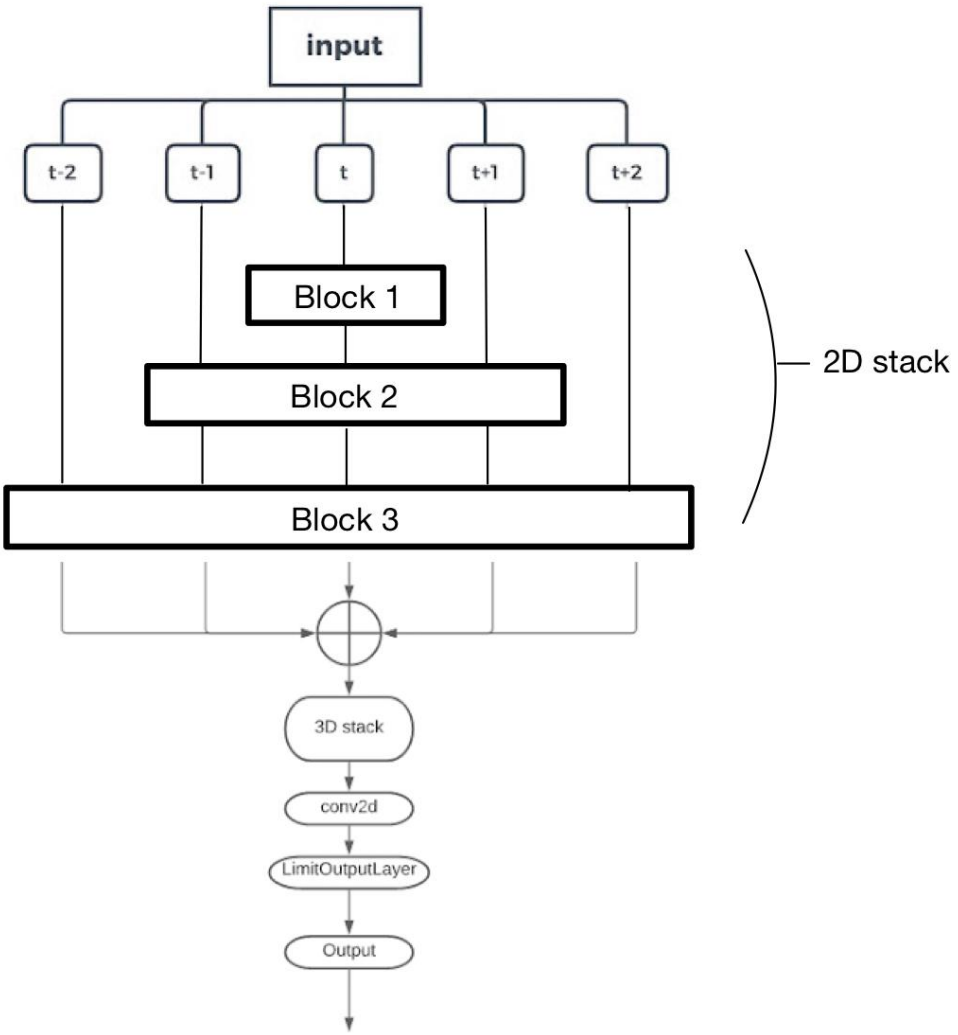

Figure : Structure of Delta
（note t-2,t-1,..,t+2 refer to the frame at time t-2,t-1,..,t+2）

With the idea of symmetry, the Delta model lets each frame pass through different numbers of 2D layers in the 2D convolution network. In this way, the 2D stack extracts the information of each frame respectively without introducing extra parameters.

In addition, Delta also adopts recursive learning to add complexity and depth. In Block 1 and Block 2, deep recursive layers are used for information extraction, as shown in the following picture.

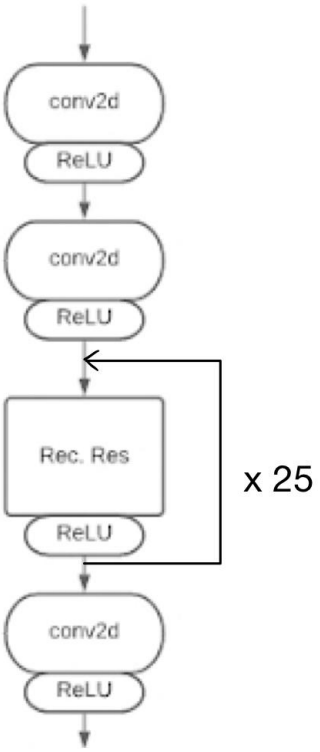

Figure : Structure of Block 1 and Block 2

Block 3 is composed of some 2D convolution layers and a 2D transpose layer, which is responsible for enlarging the image size. The structure of Block 3 is explained in the following picture.

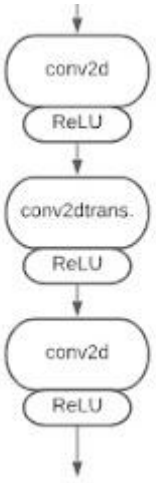

Figure : Structure of Block 3

Previous Work comparison (on climate dataset)

The major difference of 3 models can be summarized as follows:

	Previous	Delta	5-parallel
Is it a single CNN?	No	Yes	Yes
Does it contain multiple 2D layer stacks?	No	Yes	Yes
Is the recurrent 2D part involved?	No	Yes	Yes
Is the recurrent 3D part involved?	No	Yes	Yes
Is the range of output controlled?	No	No	Yes
Implementation	Tensorflow	Pytorch	Pytorch
Number of parameters	10,577(2D) 63,129(3D)	83,702	259,938

Table : Comparison of three models

Transfering from Tensorflow to Pytorch

The previous work was implemented in Tensorflow. There are few reasons for transferring from Tensorflow to Pytorch. First, many source code available are written in pytorch. If our code is also written in pytorch, it will be easier for us to combine or test our work with others' models. Another reason is that some features (e.g. obtaining 4d array from 5d array for future concatenation during training) are not directly supported in Tensorflow directly, whereas Pytorch has no such problem. It allows development of structure of complex networks easily. Also, parallel computation is emphasized in Pytorch and it has good compatibility with cuda packages. Lastly, memory-wise, Pytorch has better memory allocation than Tensorflow, which facilitates testing and training on local computers before moving on to XSEDE.

On *Climate* dataset

From the project last year, Tianxiang GAO (CityU) and Zhipeng ZHU (CUHK) test their final model, written in Tensorflow, mainly on the *Climate* dataset:

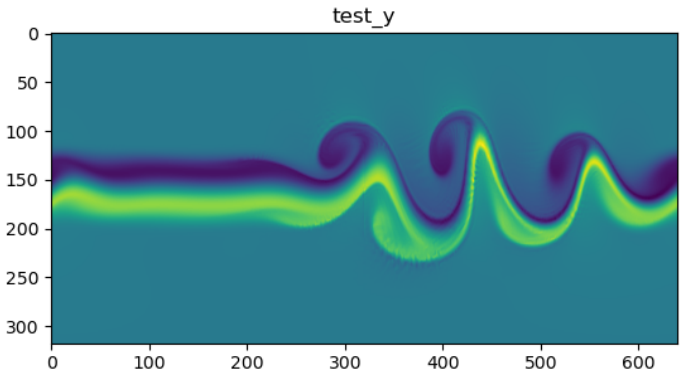

Figure : *Climate* Data Set

The results of their model in comparison with our preliminary model (before fine tune) on the testing set are listed in Figure j2.

	MSE	SSIM	PSNR
Raw	0.03296	0.6517	14.8202
Previous	4.9283e-05	0.9896	43.0746
Delta[1]	1.9801e-05	0.9943	46.0013
5-parallel[2]	2.8866e-05	0.9953	45.6618

Table : Comparison with Previous Model

However, as we can see that *Climate* is a computer-generated image, which is relatively smooth and continuous. The performance of each model is satisfactory. Though training and testing on *Climate* shows that our model is better than the previous one, it may not be good enough for general images.

Therefore, we proceed to work on a different target data set. In order to combine our model with another project on a classification problem, the target data set should have the following features:

[1] Model here is the preliminary version of Delta

[2] Model here is the preliminary version of 5-parallel

- the video is filming a single diamond
- the camera is rotating with constant angular velocity (with the diamond as center)

Data preparation and target dataset

For preparing frames from video, we used the extractor.py, while we used ffmpeg for combining the output as one video:

ffmpeg -framerate 30 -i %03d.png output.mp4

Data preparation

Generally, given a set of N HR images:

$$\Gamma = \{F_t \in M_{3\times H\times W}(\mathbb{R}) \mid 0 \leq t \leq N-1\}$$

with $c = 3$ being the number of channels, we first blur F_t with a Gaussian filter with sigma = 3. Then use INTER_CUBIC interpolation to resize the image to half of the original to become F'_t. Then, for each F_t with $2 \leq t \leq N-3$,for $|t-j| < 3$, pack F'_t with F'_j in order and form a 4-dimensional tensor $C \times 5 \times H/2 \times W/2$. Now, let $\Gamma_{tr}$and Γ_{te} be the set used in training and testing respectively, with $\Gamma = \Gamma_{tr} \cup \Gamma_{te}$ and $\Gamma_{tr} \cap \Gamma_{te} = \phi$. To be precise, each element in Γ_{tr} is an ordered pair of the 4-dimensional tensor input and the corresponding ground truth image of the target middle frame. Note that Γ_{tr} contains also the validation set. The training:validation:testing ratio is up to choice.

Target dataset

With respect to the above requirements, we obtain the dataset *Diamond* in the following video: https://www.youtube.com/watch?v=RSmSWGdjBmY

Figure : *Diamond* dataset

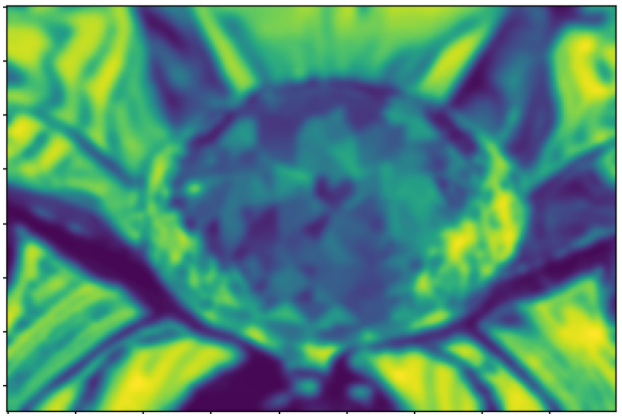

Figure : Blurred Image of *Diamond* Y-channel

300 frames are extracted from the video. 240 images are used for training and validation (with ratio and the remaining are used for testing sets. Here we only use $c = 1$and use the Y-channel of YUV decomposition of images for training and testing. In order words, we have

$$\Gamma = \{F_t \in M_{1\times H\times W}(\mathbb{R}) \mid 0 \le t \le N-1\}$$

with $N = 300$, $H = 300$, $W = 450$, $|\Gamma_{tr}| = 240$ and $|\Gamma_{te}| = 60$.

Statistics

The performance of each model (on testing set) are as follows:

	MSE	SSIM	PSNR
Raw	9193.6647	0.0392	8.5000
Delta	125.1786	0.9155	27.1675
5-parallel	127.5626	0.9184	27.0820

Table : Performance of Each Model on *Diamond* Dataset

For $|\Gamma_{tr}| = 240$, we can see that the performance of Delta and 5-parallel are similar to each other. 5-parallel performs slightly better on SSIM while Delta performs better in PSNR. Indeed, the number of parameters in 5-parallel is far more than that in Delta, which costs much more memory while training and testing. However, 5-parallel can be implemented parallel, which means the computational time cost of 5-parallel should be similar to that of Delta using similar layer depth.

A main problem that may cause bias to this result is that there are not enough samples for training and testing. Currently, there is no universal benchmark for video super resolution problems. Therefore, is it difficult to compare our result with codes available online. For the majority of papers, mostly they are using a huge data set which contains real videos and training the network for at least more than 3 days on GPU. It is also not possible for us to do so on XSEDE.

In addition, we have tried to use more data for training. For instance,$|\Gamma_{tr}| = 300$gives a satisfactory increase in improvement of PSNR and SSIM. For $|\Gamma_{tr}| = 300$ on the 5-parallel, the result on testing set (which differs from that in $|\Gamma_{tr}| = 240$) gives a PSNR of 27.6 and SSIM of 0.972. Therefore, it may be the case that both of the models converge to the best approximation that is possible to attain, with solely the information from the given dataset.

Pytorch Code for Better Efficiency

In order to enhance efficiency for training, some features are added to ensure the training and validation time cost and accuracy strikes a good balance.

1. use with torch.no_grad()
   for some of the steps, for example calculating SSIM and PSNR or validation during training, requires no grad information on variables. For each tensor in Pytorch, the defaulf of tensors has attribute requires_grad = True, and that when performing forward propagation on the model, the gradients of the operators are stored in the computational graph, which is a directed acyclic graph. This uses up GPU memories and needs more time. Therefore, we use with torch.no_grad to turn off such Pytorch features.

2. Automatic Mixed Precision (AMP) and grad scale
   The Automatic Mixed Precision autocast operations to datatype float16 or float32 depending on the operations. It can reduce the memory usage and runtime of the model.

   However, the problem of AMP is that while the gradient is small and the operation datatype is converted to float16, gradient “underflow” occurs. Therefore, grad scaling is used to prevent this problem by scaling the gradient during training.

3. Modify the learning rate
   Initially, we have set the learning rate of the Adam optimizer with 0.00001. The model converges but with relatively slow manners. Therefore, we change the learning rate of the Adam optimizer to 0.0005, which is two times the original. The final result of the model converges with similar MSE, PSNR and SSIM with the one of learning rate = 0.00001, while the training epoch needed for the validation data to reach PSNR = 26 has been greatly reduced to ~100 epochs to 20.

Discussion

In this report, we have illustrated two models: 5-parallel and Delta. There are some common features used in the model which found to be improving the result:

- Recurrent feature
  Recurrent network is currently popular in many of the computer vision problems or NLP problems. Here we used a similar idea and allowed the input image to pass through the same block multiple times. However, using recurrent features will cause stacking up in gradients, which uses up an immense amount of memory. As far as we know, there are currently no analytical methods to determine the optimal value of the number of recursions. By testing, we found that the number of recurrent that gives the best performance is 5 (for both 2D and 3D).

- Res feature
  Res block is a common feature used in neural network models. It allows flow of information from the previous layer, which is the starting layer of the Res block directly to the end of the block. The main advantage is to prevent vanishing gradient, which is said to be the degradation problem in deep neural networks, and in other words, Res block allows very deep neural networks to learn effectively.

- 2D & 3D interaction
  In previous project, Tianxiang GAO (CityU) and Zhipeng ZHU (CUHK) built a 2D + 3D model. However, they did not combine both of them as a single network. The 2D network works as a super resolution network, the output from the 2D CNN is being fed to the 3D model, which helps the 3D model to converge. However, according to their statistics, most of the reduction of error is done by the 2D model, whereas the 3D model has insignificant effect in improving the PSNR and SSIM. Therefore, we tried to combine both of the networks together, and hope that 3D network and 2D network can interact with each other to learn better.

  Indeed, we have printed out the auxiliary loss after the images has passed through the 2D stacks, however, we discovered that the 2D part is not doing super resolution, since the PSNR, MSE and SSIM of the input image did not improved as much as expected, and even worse than solely the 2D network. However, the 3D network is responsible for the remaining and the result PSNR is better than that of separating 2D and 3D models. We believe that the 2D parts are extracting 2D features, which helps the model in 3D part.

- 5-parallel performs better in SSIM while Delta performs better in PSNR
  For the *climate* and *diamond* dataset, we can observe the above result. 5-parallel obtains more 2D structural information from each frame since each frame passes through 2D CNN of same length, while 2 of them is much deeper than that of delta. However, a huge number of parameters might overfit the training and validation set, and the color of each pixel is not accurately predicted. Therefore, Delta performs better in PSNR.

Future Works

There are mainly few ways that possible improve the result of the model, which illustrates below:

Optical Flow

Optical flow is commonly used in obtaining temporal information between frames. For each pixel, an optical flow with respect to $n\ th$and $n\ +\ 1\ th$ frames record has two values, Δx and Δy, which denotes the change in x and y respectively from the $n\ th$to the $n\ +\ 1\ th$ frame. In video, each frame is of dimension (channel, H, W) , therefore optical flow is a (2, channel, H, W) matrix.

Figure: Example of Optical Flow

Data-preprocessing

Intuitively, the clearer the image, the easier and more information can be obtained from the image. Therefore, if we perform super resolution on the input dataset before inputting to the network the performance of the whole model might be able to improve.

In view of the above two directions, we propose the following scheme:

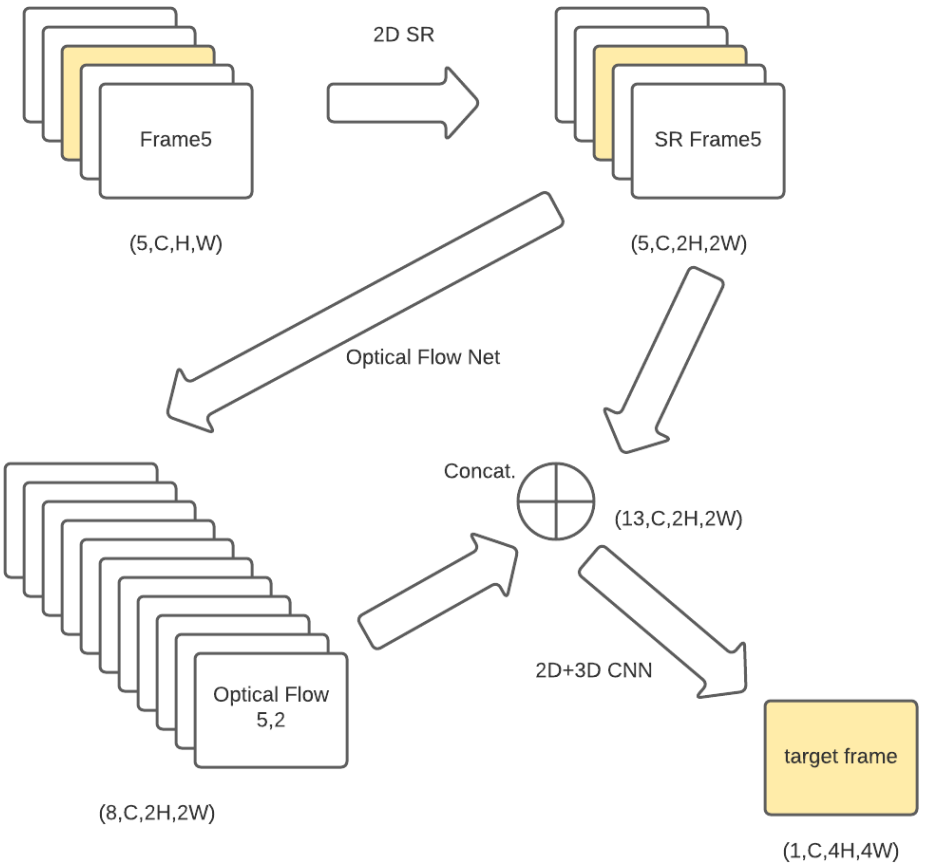

Figure: Proposed Future Model

The proposed scheme takes the same input as in our current model, which is a package of 5 frames. Instead of 2x super resolution, this proposed model gives a 4x super resolution.

FIrstly, given the 5 frames package, for every frame in the package, it passes through a 2D super resolution process. It can be done by CNN or any other methods. Then, the 5 frames package passes through an Optical Flow Net to obtain optical flow information between frames. The reason for doing super resolution before the Optical Flow Net is that we want a preliminary result for the super resolution to help us obtain clearer optical flow information between frames. Also, it can transform our problem to CNN from 4x super resolution to 2x super resolution. Next, concatenate the optical flow stack with the SR frames passed through the 2D SR and feed it to our 2D+3D CNN. The 2D+3D CNN will gather all the information and output a single super resolution target frame.

References

1. Gao, T., & Zhu, Z. (2019). *Reconstruction of High Resolution Movies using Transfer Learning*.