Reconstruction of High Resolution Movies using Transfer Learning

Member:
Tianxiang GAO, City University of Hong Kong
Zhipeng ZHU, Chinese University of Hong Kong

Mentor:
Dr. Rick Archibald, Oak Ridge National Laboratory
Dr. Kwai Wong, Joint Institute for Computational Sciences

ABSTRACT

Most existing deep-learning based super-resolution algorithm use 2D CNN to do super-resolution on single image. Different from 2D CNN, 3D CNN could accept a sequence of images as input, therefore is able to extract the correlations within the image sequence. Hence, 3D CNN is a good choice for performing super-resolution on videos, in which consecutive frames share significant similarities that cannot be ignored.

We propose a video super-resolution algorithm consisting of a 2D model and a 3D model. We do super-resolution twice. First, we use the 2D model to do single-image super-resolution on every frame and save the results. Then, we pack the processed frames as sequence of images and put them into the 3D model to do super-resolution again.

We perform transfer learning with respect to the 2D model. The structure of the 2D model is based on existing super-resolution models that have achieved good performance in single image super-resolution. We use parameters trained by others as the initializer of some layers of our model, and then we fine tune the model on our own dataset.

Objective

Contemporary physical observation and simulation produces a large amount of images and videos. Due to limit of current technology, not all of them are high-resolution ones. Our objective is to design a method to do video super-resolution on these kinds of data. We mainly focus on climate data; later we can move to data in other fields of physics.

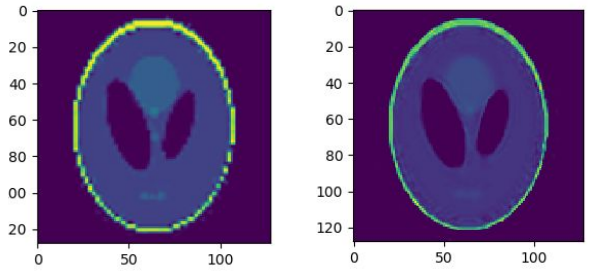
Method

Our main idea is to append a 2D super-resolution model with a 3D one. The 2D model is for basic reconstruction, processing frames independently. And the 3D model uses information from pre-frames and post-frames, adding more details to the reconstructed sequential of images. By combining a 2D super-resolution model with a 3D one, we hope to achieve a better result than that of traditional models.

1. Dataset Selection and Pre-processing

a. Shepp-logan Phantom

Shepp-logan phantom is a model of human head, using ellipsoids to represent different tissues. To generate a sequential set of images suitable for our neural network to process, we make the ellipsoids slightly change in shape and location over time. Consecutive frames will have high similarity, and easy for frames to learn information from each other.



b. Independent Images Taken from Reality

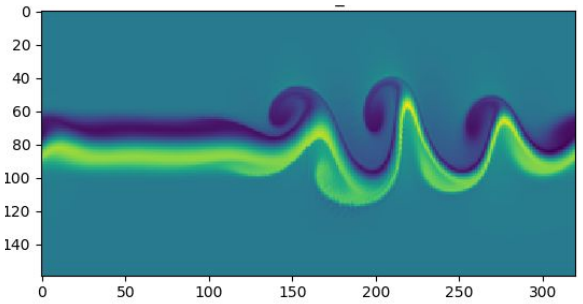
To test the performance of 2D neural networks, there are no restrictions on consecutiveness of input images. The basic 2D model we used are trained over images taken from daily life. We input similar real-life images to see the performance of the basic model, and how much improvement we can get by doing transfer learning.

c. Movies from Real-life Scenes

To test 3D neural network, input should be sequential images with some similarities within time domain. There is a website mentioned by the paper about 3D SRnet, providing movies with main features slightly moving over time. Doing 3D convolution takes longer time than 2D convolutions. The size of each frames from different movies are fixed and small, and thus are more suitable to be test cases of 3D neural networks.

d. Climate Data Generated by Shallow-Water-Equation

Shallow water equations are usually used for describing the flow under a pressure surface. The function generates different floating numbers when time changes, plotting slightly fluctuating waves among time.



Since the purpose of our project is to reconstruct computer-generated movies, we use the climate data for the final testing of all the models.

However, computer-generated video data is easier for neural network to learn. Considering the difference between these data and the videos token from reality, performance of real-life scene dataset may not be as good as the current results.

2. 2D super-resolution model

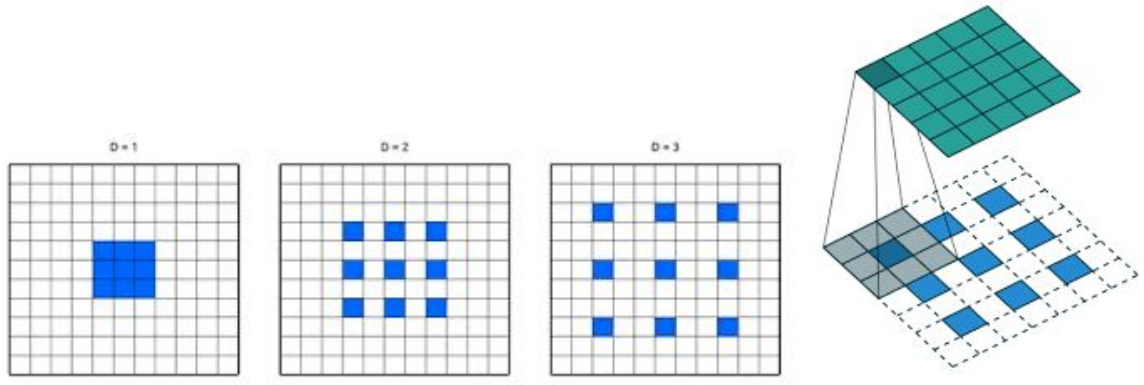
In the past few years, researchers have proposed many deeping-based super resolution models. We design 2 models for our 2D super-resolution part, based on two of existing single image super-resolution models, the Super-resolution Convolutional Neural Network model (SRCNN) and the Fast Super-resolution Neural Network model (FSRCNN).

The SRCNN model is the first super-resolution model using deep learning. It is well-known for its simple structure and efficient performance. SRCNN only have three layers, all of which are convolutional layers. The first layer is designed for feature extraction, which has 64 filters, with size 9*9. The second layer is a non-linear mapping layer, which has 32 filters, with size 64*1*1. The third layer is designed for reconstruction, which has 1 filter, with size 32*5*5. All of the three layers use 'ReLU' activation function. Further, It is worth noting that the input of SRCNN should be of the same size of the output, which requires us to resize the low-resolution image to the size of the high resolution one using bicubic interpolation before putting it into the neural network.

In our own model, we preserve the structure of SRCNN. Also, we use the filters of the first layer , which is trained by the author of SRCNN, as the initializers of our first layer. The initializer of the second and third layers are just the default setting of Keras. We change the activation function to 'pReLU' to solve the problem of vanishing gradient. Then, we fine-tune the first layer, and self-train the second and third layer on our own dataset. Further, we tries to add more layers to SRCNN model, but it is not satisfactory because performance is not improved much, and the training and processing time is increased.

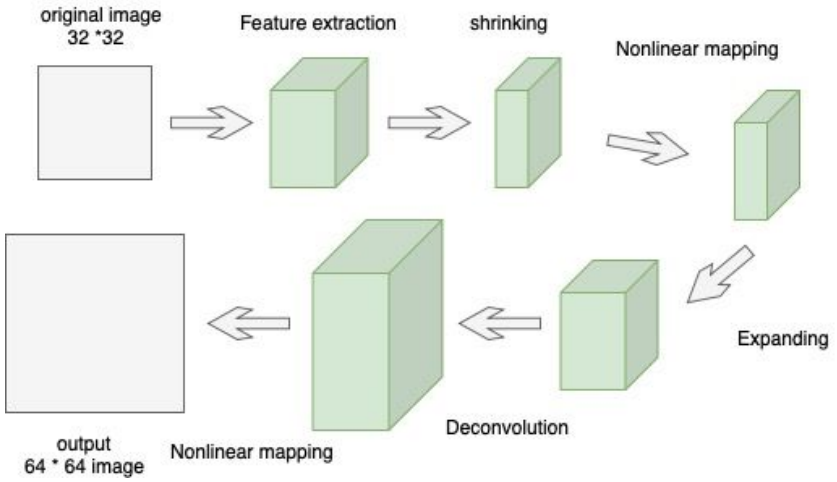
The FSRCNN model is the improved version of SRCNN. Different from SRCNN, FSRCNN do not require the low-resolution images to be resized before processing. Instead, FSRCNN use a deconvolution layer as the last layer to upsample images to a bigger size. The deconvolution layer has another name 'transpose convolution layer'. It can be considered as the inverse transform of classic convolution. The explicit process of deconvolution can be

depicted by the following picture. We add zeros between entries of a matrix to get a new matrix, and do convolution on the new matrix with respect to some filter.



The explicit structure of FSRCNN is as follows. FSRCNN has 4 convolutional layers and one deconvolution layer. The first layer is designed for feature extraction, which has 32 filters, with size 5*5. The second layer shrinks the size of the result of the first layer, which has 5 filters, with size 32*1*1. The third layer do non-linear mapping, which has 5 filters, with size 5*3*3. The fourth layer expands the size of the result of the third layer, which has 32 filters, with size 5*1*1. The last layer is a deconvolution layer with stride equal to the upscaling factor, which has 1 filter, with size 32*9*9. All of the 5 layers use 'pReLU' activation function with alpha initialized to be zero.

We derive our own model based on the idea behind the FSRCNN model. In our model, the first and the fourth layers are preserved. The number of filters of the second and the third layer is changed to 8 to extract more information from the low-resolution image. Also, we find that it is difficult for the parameter to converge to a good local minimum if the size of the filter is set to be too large in the deconvolution layer. Therefore, we increase the number of filters of the deconvolution layer to 32, and reduce the size of it to 32*2*2. Also, we add a non-linear layer after the deconvolution layer to add more non-linearity to our model after upsampling images. This layer has 1 filter, with size 32*5*5. Same as other layers, this layer uses 'pReLU' activation function. We set the parameters of the first four layers, which are trained by the authors of FSRCNN, as the initializers of first 4 layers. Then, we fine-tune these 4 layers, and self-train the last two layers using our own dataset.



3. 3D super resolution model

For current videos with fps equal to 60, consecutive frames are almost the same because the temporal interval between them is only 1/60 second. Hence, when performing super-resolution on one single frame, we can also leverage the information from frames nearby. With more information, we intend to improve the quality of the super-resolution. The idea illustrated above can be implemented using 3D convolution.

Our idea of the 3D model is based on the 3D SRNet model proposed by Kim et al. The 3D model is used for video super-resolution. Different from 2D model, the input of the 3D model is a 3D matrix, rather than a 2D matrix. The filter of a 3D convolution is also a 3D matrix. Therefore, when performing 3D convolutions, we can extract information from more than one matrix, which is suitable for finding the correlations between consecutive frames. For each frame in a video, we pack it with 2 frames before it and 2 frames after it to obtain a 3D matrix with size height*width*5. By doing this, we can train the network on both the spatial and the temporal domain. Our goal is to leverage the correlations between one frame and 4 frames nearby to add more details to the high-resolution image.

The structure of our 3D model is as follows. Similar to 2D models, the first layer is designed for feature extraction, which has 32 filters, with size 5*5*3. The second layer is for shrinking, which has 8 filters, with size 32*1*1*3. The third and fourth layer are designed for nonlinear mapping, which both have 8 filters, with size 8*3*3*3. The fifth layer is for expanding the interim result for later processing, which has 32 filters, with size 8*3*3*3. Until now, all the layers are used for extracting features from input images, and choose

important features from them. Also, we set the padding option to be ‘same’ in all these 5 layers, which means that the size of the result right after the fifth layer is 32*height*width*5. After the fifth layer, we start to derive our final result from the information we collect. The sixth layer has 32 filters, with size 32*3*3*3. We set the padding option to be ‘valid’, which means that the interim result of the sixth layer is 32*(height-2)*(width-2)*3. The seventh layer also has 32 filter, with size 32*3*3*3, padding to be ‘valid’. After this, the interim result has size 32*(height-4)*(width-4)*1, which means that we have obtained one single frame from 5 consecutive frames. We add one non-linear layer as the last layer to obtain one single image. The last layer has 1 filter, with size 32*5*5. For all the layers mentioned above, we use ‘prelu’ activation function with alpha initialized to be 0.

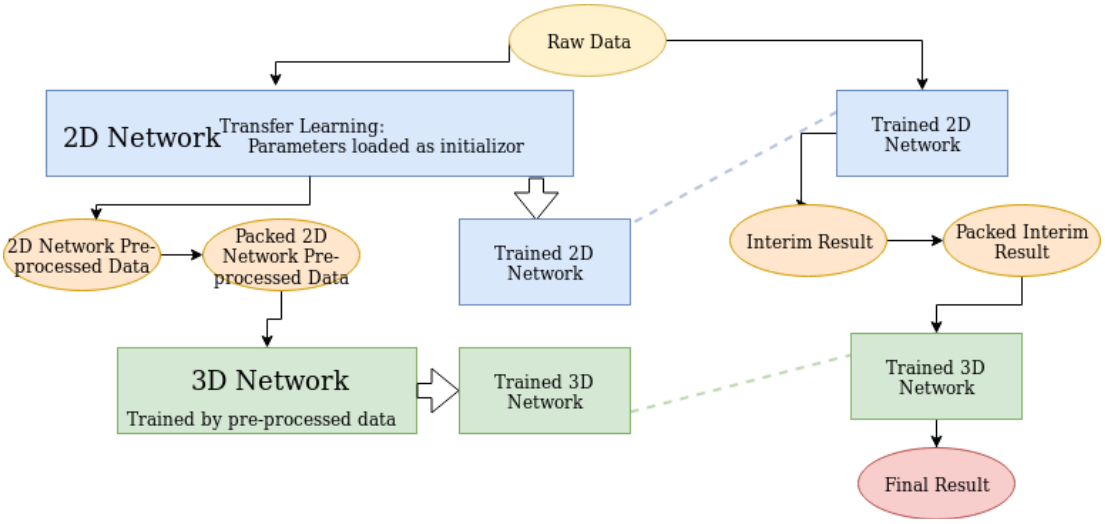
4. Combine 2D Network and 3D Network

As mentioned before, our main idea is to combine 2D model with 3D model. We propose 2 methods to implement our idea.

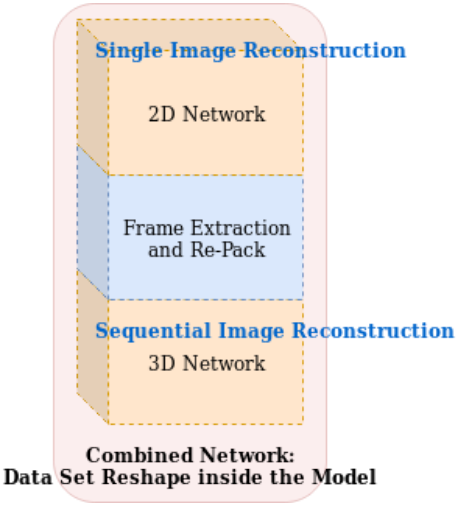
The first method is to train 2D model and 3D model separately. First, we train the 2D model using climate data. Then, we process the raw data using the saved 2D model. After that, we pack the processed data as the input of the 3D model for training. By doing so, we hope we could increase the quality of super-resolution twice.

This method has several advantages. First, it is easy to train and implement because we are actually training two separate models. Second, we can set different hyperparameters for each models to ensure convergence of both the 2D and 3D model. But this method also has one disadvantage. That is the lack of interaction between 2D and 3D models because we train 2D and 3D models separately.

The structure of the training process overall is illustrated by the picture below.



Another method is to concatenate the 2D model and 3D model in one single neural network. We add a layer between 2D model and 3D model to process the data from single frames into packages consisting of 5 consecutive frames. For example, if the output of the 2D network have size M*N*AMOUNT, where M is the height of the images, N is the width of the images, and AMOUNT is the total amount of frames, then after processed by the middle layer, the size will become M*N*5*AMOUNT.



Concatenating 2D and 3D models mean that when can train the 2D model and 3D model at the same time. Since 2D and 3D models could interact each other during training, we probably could obtain a better result. But this method also have some disadvantages. First, a deeper neural network is less likely to converge to a good local minimum, which require us to tune hyperparameters more carefully to ensure convergence. Second, we have to set same hyperparameters for 2D and 3D models, which may decline the performance of 2D or 3D models slightly compared to training them separately.

Experiment and Results

1. Main Idea

To test how good a reconstruction algorithm is, when processing data we leave one set of uncompressed images as ground truth.

We compare the similarity and clarity of compressed or shrinked images with the ground truth twice: before neural network processing and after neural network processing. If images after processing is more similar than the one before, means the algorithm is more suitable for performance improvement.

Considering the limitation of computing resources, we are training on 120 images and testing on 50 images.

2. Indexes Information

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

- MAE: Mean Absolute Error

Mean absolute error is used to measure the difference between two variables. For our cases, we are using MAE to see the difference between two images and see their similarity. The lower MAE is, the similar two images should be. To achieve better performance, we want MAE to be the smaller the better.

- MSE: Mean Squared Error

Mean squared error is used to measure squared difference between two variables. This is also an index to measure the loss of information among images, but the difference will be amplified by the operation of "square". We determine MSE to be our loss function when training neural network. During training, we want to decrease the MSE.

$$MSE = \frac{1}{m\,n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

- PSNR: Peak Signal-to-Noise Ratio

The index is a logarithmic-scaled representation, measuring the ratio between the maximum possible power of a signal and the power of corrupting noise.

$$\begin{aligned} PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\ &= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned}$$

For the formula, MAX_I is the maximum possible pixel value of the image.

According to the formula, the less MSE is, the larger PSNR should be. Our goal is to increase PSNR by processing.

- SSIM: Structural Similarity Index

Structural similarity index is widely used in digital-image processing to predict the quality. Our purpose is to improve similarity, thus SSIM should be the larger the better.

μ_x the average of x;
μ_y the average of y;
σ_x^2 the variance of x;
σ_y^2 the variance of y;
σ_{xy} the covariance of x and y;
$c_1=(k_1 L)^2$, $c_2=(k_2 L)^2$ two variables to stabilize the division with weak denominator;
L the dynamic range of the pixel-values (typically this is $2^{\#bits\ per\ pixel}-1$);
$k_1=0.01$ and $k_2=0.03$ by default.

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

3. Training and Results

In our training process, we use high resolution data collected as the ground truth. We downscale the ground truth by scale factor 2 using bicubic interpolation to get low-resolution images. Our goal is to obtain high-resolution images from low-resolution images through the network. We compare the reconstructed high-resolution images with the ground truth to measure the performance of our models.

1. 2D Model

Since the image size of input size and output size are different before neural network processing, we cannot know how similar the raw data (compressed images) and the ground truth (uncompressed images) are.

- Epoch = 100

Totally 120 images in data set	Train Set (80% of all train pictures)			Validation Set (20% of all train pictures)		
	MSE	SSIM	PSNR	MSE	SSIM	PSNR
Epoch = 0	0.1787	0.0206	7.6772	0.0853	0.0334	10.6893
Epoch = 100	1.3292e-4	0.9764	38.9665	1.4266e-4	0.9744	38.4576

Table 1

- Epoch = 150

Totally 120 images in data set	Train Set (80% of all train pictures)			Validation Set (20% of all train pictures)		
	MSE	SSIM	PSNR	MSE	SSIM	PSNR
Epoch = 0	0.1787	0.0206	7.6772	0.0853	0.0334	10.6893
Epoch = 150	6.4832e-5	0.9871	41.8839	7.0370e-5	0.9859	41.5278

Table 2

2. 3D Model

a. 3D SRnet Model trained with Raw Data

The neural network is complicated to learn and requires a lot of computing resources. The training result, however, is not better than to simply use 2D network.

(a-0). 3D SRnet Model trained with Climate Data

- Raw data quality: PSNR = 13.8458 dB; SSIM = 0.5304; MSE = 0.0413

Totally 120 images in data set	Train Set (80% of all train pictures)			Validation Set (20% of all train pictures)		
	MSE	SSIM	PSNR	MSE	SSIM	PSNR
Epoch = 0	0.0342	0.6628	14.7373	0.0280	0.6572	15.5334
Epoch = 100	0.0226	0.7377	16.4686	0.0231	0.7262	16.3682

Table 3

(a-1). 3D SRnet Model trained with Movies from Real-life Scenes

At the first stage, we used movies from real-life scenes rather than climate data. This is because of the smaller size of these movies' frames, which shortens the time to finish one epoch in 3D cases. Real-life scene videos are also in worse qualities than the videos constructed by computer. This helps us to clearly see how much improvement 3D SRnet model could achieve.

- Images are in shape 64 x 64

Totally 236	Train Set (80% of all train pictures)	Validation Set (20% of all train pictures)

images in data set	MSE	SSIM	PSNR	MSE	SSIM	PSNR
Epoch = 0	2504.1546	0.4550	16.4601	518.8638	0.5568	20.9809
Epoch = 100	99.3784	0.8788	28.2013	128.0339	0.8552	27.1830

Table 4

b. 3D SRnet Model trained with 2D network pre-processed Images

- Trained with pre-processed to PSNR = 42.08dB data set, patience for early stop = 5

Totally 120 images in data set	Train Set (80% of all train pictures)			Validation Set (20% of all train pictures)		
	MSE	SSIM	PSNR	MSE	SSIM	PSNR
Epoch = 0	0.0984	0.6668	11.7868	0.0051	0.9022	22.9122
Epoch = 67 (Early Stop)	8.4290e-4	0.9837	40.7561	1.0119e-4	0.9812	39.9499

Table 5

- Trained with pre-processed to PSNR = 42dB data set, patience for early stop = 5

Totally 120 images in data set	Train Set (80% of all train pictures)			Validation Set (20% of all train pictures)		
	MSE	SSIM	PSNR	MSE	SSIM	PSNR
Epoch = 0	0.0750	0.7401	15.3667	0.0019	0.9306	27.2128
Epoch = 80	4.5721e-05	0.9907	43.4010	4.9283e-05	0.9896	43.0746

Table 6

The more clear the interim-result (the data to be trained) is, the more useful it helps in improving the first few epochs' performance. The final improvement is not severe by 3D network processing. But since the training progress was not early-stopped, more epochs might help to even improve the performance.

3. Combined 2D Network and 3D Network

a. 2D Network and 3D Network (3D network trained with raw data)

Totally 120 images in data set	MSE	SSIM	PSNR
2D network (*Table 2*)	7.593760673e-05	0.9847759507246662	41.201901955249504
After data re-packing*	8.124760623839217e-05	0.9846107211562674	40.9071674094057
3D SRnet network	0.0096011151711	0.803450561713647	20.17703292426752

Table 7

b. 2D Network and 3D Network (3D network trained with 2D network pre-processed result)

Totally 120 images in data set	MSE	SSIM	PSNR
2D network (*Table 2*)	5.807663600363179e-05	0.987824159428385	42.36632787676039
After data re-packing*	6.178588381302932e-05	0.9877120117274287	42.09522054489414
3D SRnet network	4.632699437652877e-05	0.9910728454576089	43.34586764073154

Table 8

*** Data Repacking**: To process images using the 3D network, the size of final result images should smaller by 4 in width and height respectively. This is caused by the downgrading process inside 3D network. When test the similarity of packed images and ground truth, we cut off the 4-pixel wide boundary of to-be-processed data. Thus the performance comparison will be more consistent with the final result.

Analysis and Future Steps

1. Dataset Selection

There are two different choice when choosing train set and test set of neural network:

One is to pick both the train set and the test set from climate data. Neural network training generally requires train set and test set has high level similarity. By applying this design, train set and test set will have many common scenes, better for the performance.

However, if train set and test set are too much similar, it may cause overfitting problem: what neural network has learned is specific for the data we provided, and not suitable for other computer-generated videos.

Another is to regard climate data only as test set. As for train set, we need equations like shallow-water-equation to generate similar computer-generated videos. This method can help to avoid overfitting problem, but may compromise the performance of neural network on testset.

Currently, we have tried to take both the train set and test set from climate data only. For future improvement, it's a good idea to try the second method and use different data to generalize the trained neural network.

2. Reconstruction Tools

Besides neural network processing, there are many ways to reconstruct downscaled images with mathematical based methods. Using different python packages for resizing (enlarging) will show different results. For example, openCV, Pillow and scipy.misc have embedded functions for interpolation based resizing. User can choose tools as "bicubic", "bilinear" or "nearest" by themselves. Neural network is not the simplest or the only way to do reconstruction.

3. Compression and Contraction

Information lost in images could be due to compression or contraction. Compression, including lossless compression and lossy compression, is to minimize the storage of redundant information inside images. Contraction, however, is using the natural of information lost in zoom-out process to throw out information.

The basic models we referred to are doing super-resolution, which generally enhances resolution by enlarging the total amount of pixels. These method should be more suitable for cases that lost information because of contraction, but not for all cases of general compressions.

Besides super-resolution, there are some other methods to enhance image resolution. For example, compression artifacts removal can sense distorted parts and reconstruct based on these entries.

4. Interim Result

The maximum of PSNR is only 48dB, and it's hard to achieve PSNR larger than 40dB. When training on "combined 2D and 3D Network", interim result is too good that the result of re-trained 3D network even makes a setback in performance. If we want to see if the combination network can help to even improve performance, interim results' quality should not be too good, and leave some improving space for the 3D model to learn.

During experiment we found that the quality of 2D network processed data would have impact on the quality of 3D network output (*Table 7 and Table 8*). The key point is to find a trade-off value between the "significant improvement" and "starting quality" for the 2d network pre-processed data.

One possible method is to limit the epoch trained over the first network. Before the 70th epoch, PSNR is lower than 36dB. To see if any improvements are made by the 3D network, the overall epoch numbers in 2D network training process could be set as 70.

5. "3D + 2D" Network Combination

To do both independent image reconstruction and sequential image reconstruction in one model, we hope we can use not only the information hidden inside the image itself but also information stored in consecutive frames.

However, for each model and dataset, there should be a peak value that the result could be improved to by neural network processing. The inter-compensation idea of combination does not mean the peak values are also combined. Thus it's reasonable that the combination network's result is not better than processed only by one network.

Based on the explanation above, there's another idea other than our current "2D+3D" combined network: Process images with 3D network at first, and then the 2D network. The reason behind is that, since 3D network has more complicated layers and difficult to learn,

the peak value should be lower than 2D network. By appending 2D network after 3D network, we can at least avoid the "cask effect" caused by 3D network's peak value.

We haven't tried the "3D+2D" combination yet. 3D network trained with images preprocessed by 2D model could learn, but the one trained with raw data made no improvement and was early-stopped at early stage. The problem could be caused by hyper-parameter set are not suitable for the dataset, but we didn't have time to test and see whether the assumption is the correct reason. After fixing the problem of the 3D model, it will be a good trial to try the "3D+2D" combination: train 3D network first; input the 3D network pre-processed images into 2D network and get 2D network-trained final result.

6. Two-in-one 2D Network and 3D Network

To apply back-propagation in the combination network structure, we want to build a two-in-one network. The interim results are not output after processed by the 2D network, but directly feeded to the later 3D network. By doing this, we tried to self-define a lambda layer, packing 5 images together as a package, and then directly feed the output tensor of this layer into the 3D network.

The challenge we are facing is that, we need to use iterations inside the lambda layer, and the index for iteration is the batch size of each input tensor. This means we need an exact integer to indicate how many iterations the loop needs to take. But we cannot get the exact number of batch size since the number is dynamically stored. And this makes a confliction.

Currently we haven't successfully implemented this model, but there are two suggested methods to solve the problem:

The first is to do data pre-processing before the beginning of two-in-one network. Accordingly, 2D model's input shape is changed. The structure of 2D network need to be modified respectively, but the concept of 2D network is still the same. This idea is more simple to implement. But 2D network needs to process a lot of duplicated frames and causes redundancy. If computing resource permits, this is a good way to try.

Another way is to avoid the usage of iterations in the lambda layer. Using vectorise operations may be a good substitution of for-loop. Keras and tensorflow are matruelly developed, it's good for efficiently build network model, but difficult to do self-modification.

Hopefully with the help of MagmaDNN, more function could be provided to implement the design.

As mentioned before, it's also a good idea to implement 3D network before the 2D network ("3D+2D" combined network). It will be easier to do implementation since no data repacking process is needed, and no lambda layer is required.

Reference

Dumoulin V, Visin F. A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285. 2016 Mar 23.

Appendix

Data:

Movies from real-life scene:

https://media.xiph.org/video/derf/
We only used sets called "akiyo" and "container" on the website to train our 3D models.

Climate data:

Each entries are stored as float32, values between (-10, 10). Before feeding the data into neural network, need to renormalize to (0, 1).
https://bitbucket.org/EDKLW/image-reconstruction-2019/src/master/Video_Stream/H_Field_Nl5.mat
The processed data stored in an h5 file.
https://www.dropbox.com/sh/8809q13y36nct2t/AADKbI4XtWDQGO-h_031RGkLa?dl=0

Fine-tuned parameters for 2D model:

To implement the concept of transfer learning, we loaded some fine-tuned parameters provided to our network, whether as initializer or to be frozen inside our model.
https://bitbucket.org/EDKLW/image-reconstruction-2019/src/master/papers%20and%20codes/Accelerating%20SRCNN/
https://bitbucket.org/EDKLW/image-reconstruction-2019/src/master/papers%20and%20codes/SRCNN/SRCNN_test/model/

Code:

main.py

The basic idea of the code is to make a comparison over different models mentioned above. Train set and test set loaded for each model are the same, only the progress to build and train the models are different.
https://bitbucket.org/EDKLW/image-reconstruction-2019-a/src/master/final/main.py

two_d.py

The basic model that we referred to as a prototype. It was raised by existing paper. We repeated the structure of the model and applied transfer learning to it.
https://bitbucket.org/EDKLW/image-reconstruction-2019-a/src/master/final/two_d.py

fsrcnn.py

The model we used for 2D independent image reconstruction. The model used deconvolutional layer to zoom in compressed and shrinked images. The model also applied the concept of transfer learning.
https://bitbucket.org/EDKLW/image-reconstruction-2019-a/src/master/final/fsrcnn.py

code_3d.py

The model is based on the concept of 3D SRnet. Data processing is needed before input into the 3D convolutional layers to run the model. Detailed processing progress is in ***load_data.py***.
https://bitbucket.org/EDKLW/image-reconstruction-2019-a/src/master/final/code_3d.py

code_3d_processed.py

Same model used in ***code_3d.py***. The difference is on the dataset for training the network. To be specific on climate data, the model is trained on ***fsrcnn.py*** pre-processed images.
https://bitbucket.org/EDKLW/image-reconstruction-2019-a/src/master/final/code_3d_processed.py

two_d_three_d.py

No new networks are trained. The code loaded fsrcnn network and 3D SRnet trained before, and made concatenated reconstruction.
https://bitbucket.org/EDKLW/image-reconstruction-2019-a/src/master/final/two_d_three_d.py

two_d_processed_three_d.py

No new networks are trained. The code loaded fsrcnn network and 3D SRnet trained before. The difference from the last one is this 3D model are trained over fsrcnn-network pre-processed results.
https://bitbucket.org/EDKLW/image-reconstruction-2019-a/src/master/final/two_d_three_d.py

Result:

Table 5: ***2d_processed_3d_quality_matched.out***
https://bitbucket.org/EDKLW/image-reconstruction-2019-a/src/master/results/2d_processed_3d_quality_matched.out