Ruiqi Li

Professor Amir Jafari

DATS 6203

12 Dec 2022

ML2 Project Individual Report

My group's project focuses on super image resolution, which is to enhance the resolution of images. The original dataset is available on Kaggle. Some of the techniques we used include computer vision, autoencoder, and general adversarial network. The dataset contains at least 1000 images, in low resolution and high resolution separately in two folders. For a general outline of shared work, we handle implementation of SR-CNN, Autoencoder, and General Adversarial Network.

To begin with, here is the context information about SRCNN. Researchers from Cornell published their paper about the design of SRCNN, so the main information below comes from the paper. "SR" means single image super resolution. This type of CNN consists of patch extraction and representation, non-linear mapping, and reconstruction (Fig 1). "Given a low-resolution image Y, the first convolutional layer of the SRCNN extracts a set of feature maps.

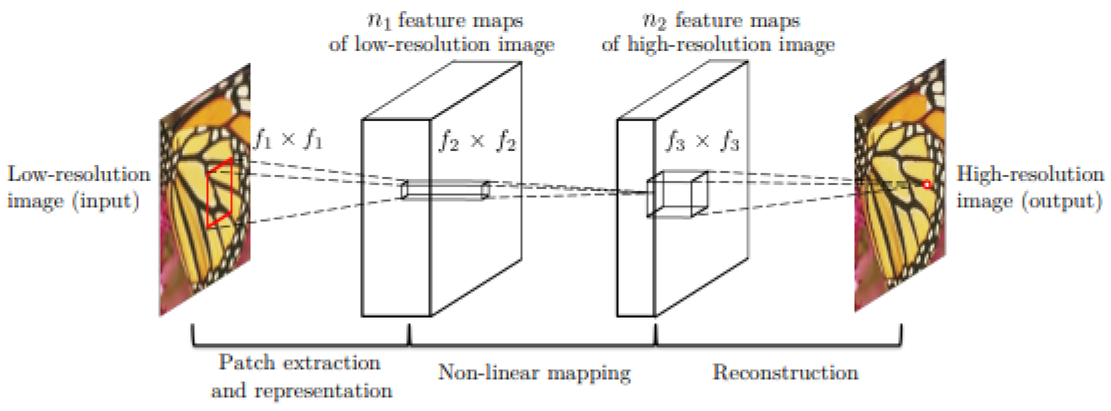

n_1 feature maps of low-resolution image
n_2 feature maps of high-resolution image
$f_1 \times f_1$
$f_2 \times f_2$
$f_3 \times f_3$
Low-resolution image (input)
High-resolution image (output)
Patch extraction and representation
Non-linear mapping
Reconstruction

Fig 1

The second layer maps these feature maps nonlinearly to high-resolution patch representations. The last layer combines the predictions within a spatial neighbourhood to produce the final high-resolution image F(Y)" (Dong, Loy, He, Tang, page 4). Fig 2 is the formula of the first layer. Y is the input image, W1 and B1 are filters and biases, and '*' is convolution operation. The first

$$F_1(\mathbf{Y}) = \max(0, W_1 * \mathbf{Y} + B_1)$$

Fig 2

part does n1 times of convolution with a kernel of size of c*f1*f1, where c is number of channels and f1 is filter size, and ReLU for output. Then in the second part of non-linear mapping, the

$$F_2(\mathbf{Y}) = \max(0, W_2 * F_1(\mathbf{Y}) + B_2)$$

Fig 3

general formula and operation are like the first part except transforming the n1-dimensional vectors to vectors with n2 dimensions (Fig 3). B2 represents n2 dimensions and W2 represents n2*n1*f2*f2 (n2 filters of n1*f2*f2). The third part reconstructs the image to high resolution with the operation described in Fig 4. B3 stands as c dimensions and W3 stands as c*n2*f3*f3.

$$F(\mathbf{Y}) = W_3 * F_2(\mathbf{Y}) + B_3$$

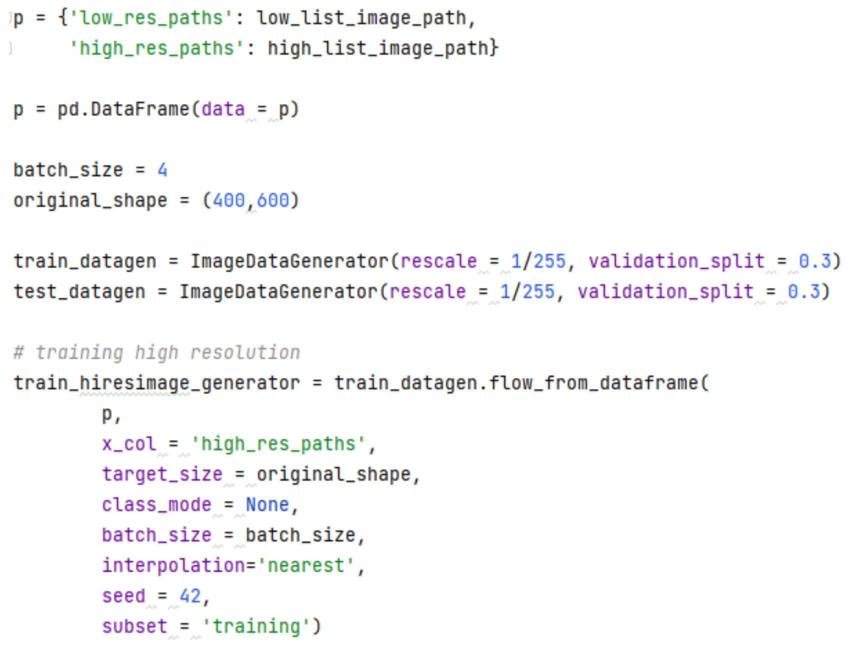
Fig 4

For the loss function, mean square error is used in this case (Fig 5).

$$L(\Theta) = \frac{1}{n}\sum_{i=1}^{n} ||F(\mathbf{Y}_i; \Theta) - \mathbf{X}_i||^2$$

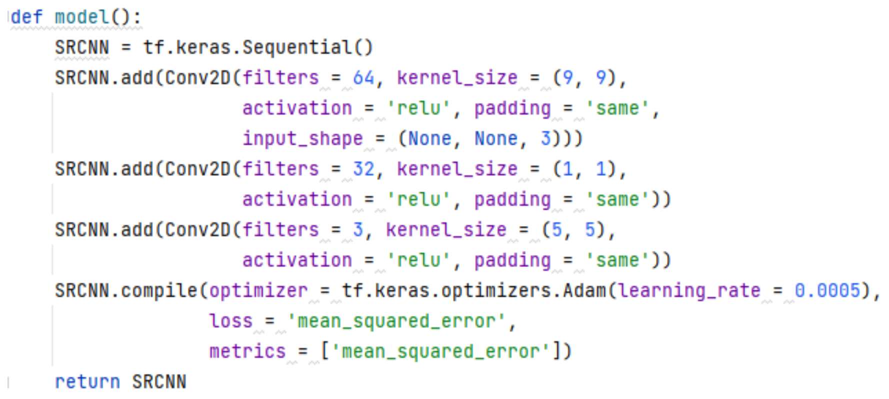
Fig 5

This portion of work includes generating the predicted high-resolution images by SRCNN. To better feed the images to the model, I save the dataframe that stores the paths of low-resolution images and high-resolution images and combine it with ImageDataGenerator. For my ImageDataGenerator, I set the rescale value as 1/255 to scale the image's RGB coefficients so that I can input the data. I also set the validation split as 0.3 and resize the image to half of the original size. Example code is in Fig 5. Next, Fig 6 is the construction of SRCNN, which is the best trial among my attempts. It is noticeable that I use 1 x 1 kernel in the second layer because “1 x 1 convolution is suggested to introduce more non-linearlity to improve the accuracy” (Tsang, paragraph 12). I use Adam as the optimizer for efficiency and set the learning rate to 0.0005. For the other things, I follow the procedure above and keep tuning the hyperparameters.



```
p = {'low_res_paths': low_list_image_path,
     'high_res_paths': high_list_image_path}

p = pd.DataFrame(data = p)

batch_size = 4
original_shape = (400,600)

train_datagen = ImageDataGenerator(rescale = 1/255, validation_split = 0.3)
test_datagen = ImageDataGenerator(rescale = 1/255, validation_split = 0.3)

# training high resolution
train_hiresimage_generator = train_datagen.flow_from_dataframe(
        p,
        x_col = 'high_res_paths',
        target_size = original_shape,
        class_mode = None,
        batch_size = batch_size,
        interpolation='nearest',
        seed = 42,
        subset = 'training')
```

Fig 5



```
def model():
    SRCNN = tf.keras.Sequential()
    SRCNN.add(Conv2D(filters = 64, kernel_size = (9, 9),
                     activation = 'relu', padding = 'same',
                     input_shape = (None, None, 3)))
    SRCNN.add(Conv2D(filters = 32, kernel_size = (1, 1),
                     activation = 'relu', padding = 'same'))
    SRCNN.add(Conv2D(filters = 3, kernel_size = (5, 5),
                     activation = 'relu', padding = 'same'))
    SRCNN.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.0005),
                  loss = 'mean_squared_error',
                  metrics = ['mean_squared_error'])
    return SRCNN
```

Fig 6

The left subplot represents the low resolution and high resolution. The right represents the low resolution and SRCNN-tuned resolution. We can see the improvement of SRCNN training from epoch 1 to epoch 20. Epoch 1’s prediction is still blurry. In epoch 5, the result has been improved. At epoch 20, we can see more textures and more brightness in the SRCNN result. It is close to the high-resolution of the image.

Below are the visualization comparisons of low-resolution, high-resolution, and SRCNN-tuned version of the three epoch stages of 1, 5, 20 (Fig 7, Fig 8, Fig 9). The left subplot represents the low resolution and high resolution. The difference between these two can be easily found since the high-resolution version has more textures and is much brighter than the low resolution version. The right subplot of each figure represents the low resolution and SRCNN-tuned resolution. We can see the improvement of SRCNN training from epoch 1 to epoch 20. At epoch 20, we can see more textures and more brightness in the SRCNN result. It is close to the high-resolution of the image.

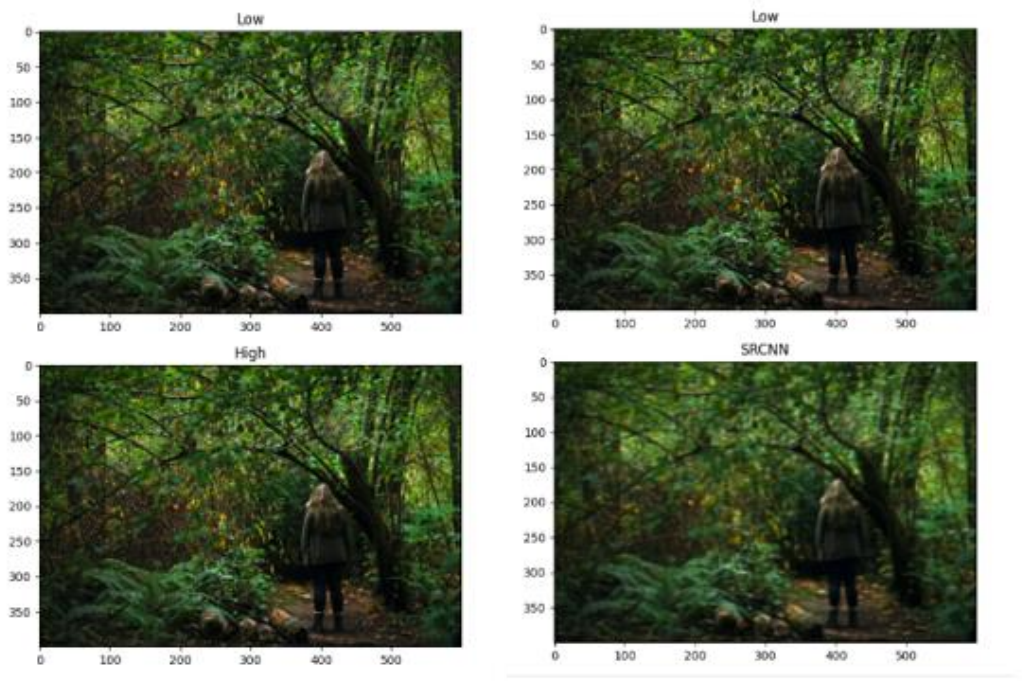


Fig 7

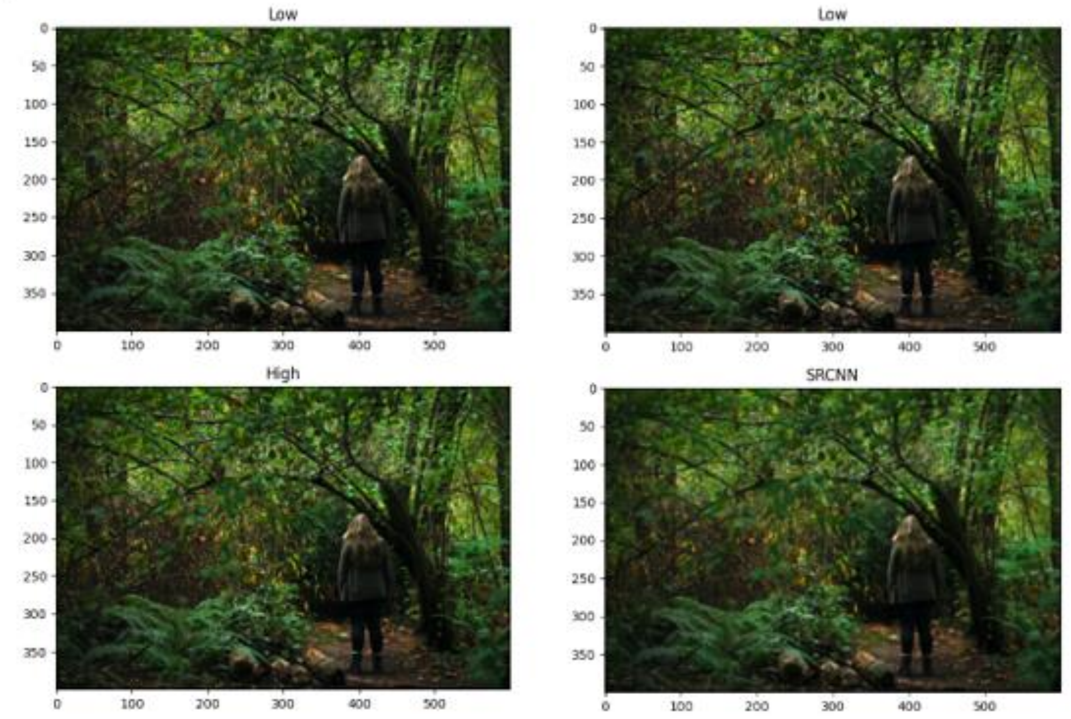


Fig 8

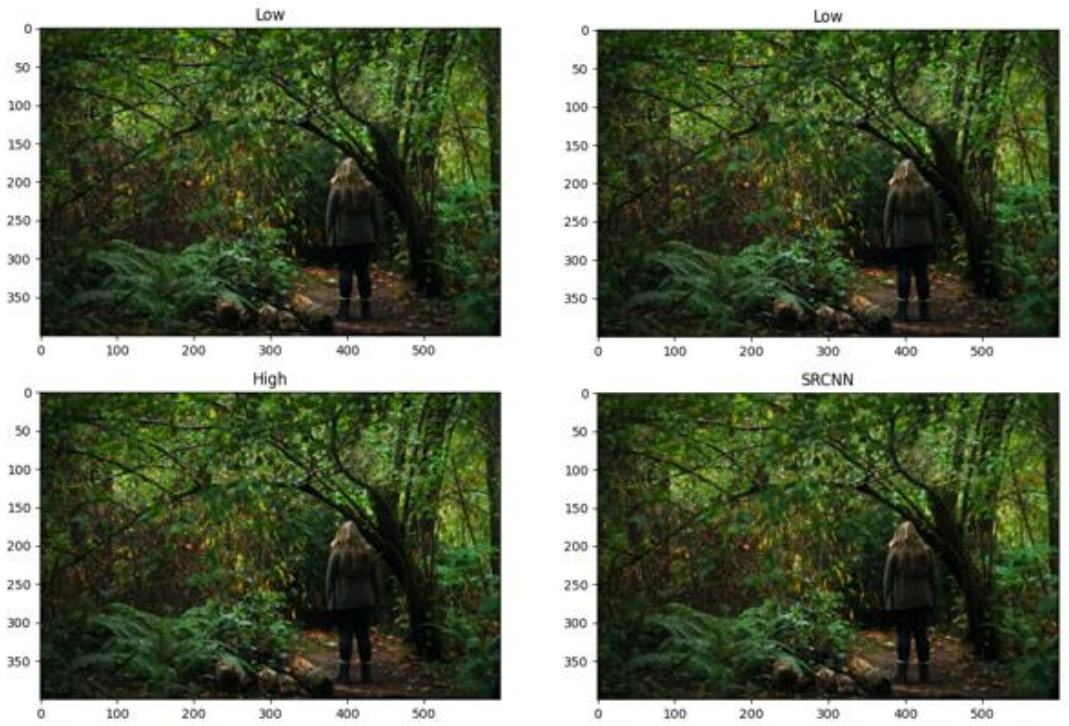
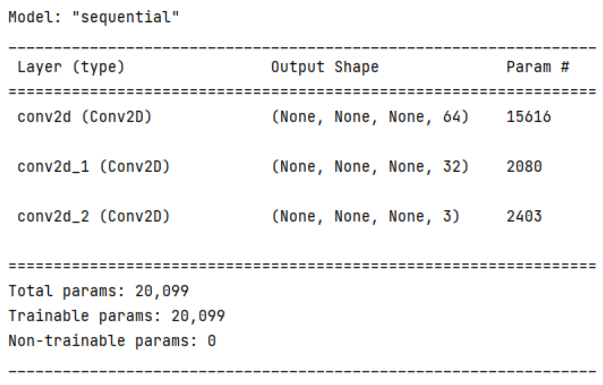


Fig 9

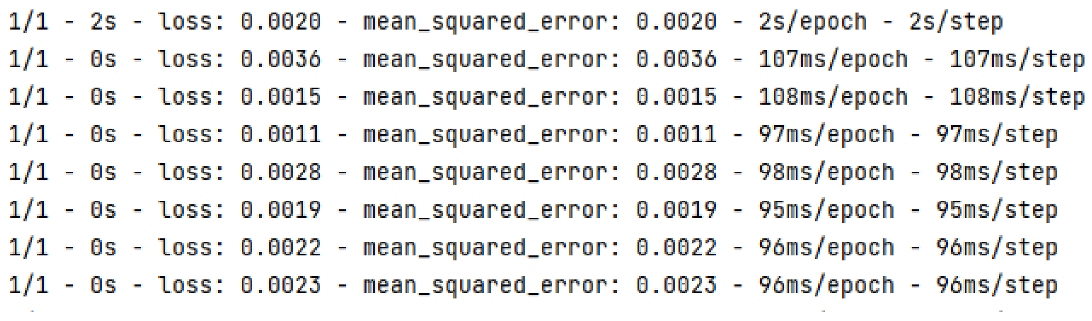
In addition, Fig 10 shows the summary of the SRCNN. For example, the input size of (None, None, 3), the first conv2d outputs the shape of (None, None, None, 64) since the number of filters is set to be 64 and the first 'None' represents the batch size in model. Each layer's output shape will be determined by the initial input size and transform based on the parameter settings of every layer. While remaining the padding as same and activation as ReLU, each operation should be similar except for the setting of kernel.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, None, None, 64)    15616

 conv2d_1 (Conv2D)           (None, None, None, 32)    2080

 conv2d_2 (Conv2D)           (None, None, None, 3)     2403

=================================================================
Total params: 20,099
Trainable params: 20,099
Non-trainable params: 0
_________________________________________________________________
```

Fig 10

MSE loss script is in Figure 11. This is a part of the evaluation script when running with test data. The mean square error is averagely maintained around 0.002.



```
1/1 - 2s - loss: 0.0020 - mean_squared_error: 0.0020 - 2s/epoch - 2s/step
1/1 - 0s - loss: 0.0036 - mean_squared_error: 0.0036 - 107ms/epoch - 107ms/step
1/1 - 0s - loss: 0.0015 - mean_squared_error: 0.0015 - 108ms/epoch - 108ms/step
1/1 - 0s - loss: 0.0011 - mean_squared_error: 0.0011 - 97ms/epoch - 97ms/step
1/1 - 0s - loss: 0.0028 - mean_squared_error: 0.0028 - 98ms/epoch - 98ms/step
1/1 - 0s - loss: 0.0019 - mean_squared_error: 0.0019 - 95ms/epoch - 95ms/step
1/1 - 0s - loss: 0.0022 - mean_squared_error: 0.0022 - 96ms/epoch - 96ms/step
1/1 - 0s - loss: 0.0023 - mean_squared_error: 0.0023 - 96ms/epoch - 96ms/step
```

Fig 11

In conclusion, the SRCNN is a simple but efficient computer vision model for image resolution improvement. I used to add multiple layers and various manipulations when writing the neural network, but this SRCNN does not have a too complex structure and it solves the problem. More layers and more manipulations may not mean better neural network models. The hyperparameter tuning is also important. When evaluating the model, I found batch size should not be big, and the image must be resized to a smaller scale because these will cause memory problems when training. For the improvements, I think running in a better environment with more GPU resources could be a boost to reduce the memory issues. About the image data, data augmentation could be a way. Even though the dataset has provided images with different levels of low resolution, I could use some more methods from Transform like flipping and changing color scale, but I think memory will be a problem. There could be some other loss functions that are better than MSE in this case, too.

For my percentage of the code, it should be around 3%, since there are empty lines and lines of assigning variables, I take the calculation as (40 – 35) / (40+150). This is the best I can evaluate.

References

1. Shaikh, Quadeer. "Image Super Resolution (from Unsplash)." *Kaggle*, QUADEER SHAIKH, 14 Oct. 2022, https://www.kaggle.com/datasets/quadeer15sh/image-super-resolution-from-unsplash.

2. Dong, Chao, et al. "Image Super-Resolution Using Deep Convolutional Networks." *ArXiv.org*, 31 July 2015, https://arxiv.org/abs/1501.00092.

3. Huang, Yuxiao. "Teaching/code_example.Ipynb at Master · Yuxiaohuang/Teaching." *GitHub*, 10 Jan. 2022, https://github.com/yuxiaohuang/teaching/blob/master/gwu/machine_learning_I/spring_2022/code/p3_deep_learning/p3_c2_supervised_learning/p3_c2_s3_convolutional_neural_networks/code_example/code_example.ipynb.

4. Shaikh, Quadeer. "Image Super Resolution Using Autoencoders." *Kaggle*, Kaggle, 16 Oct. 2022, https://www.kaggle.com/code/quadeer15sh/image-super-resolution-using-autoencoders.

5. Kang, Chanseok. "Super-Resolution Convolutional Neural Network | Chan`s Jupyter." *Super-Resolution Convolutional Neural Network*,

https://goodboychan.github.io/python/deep_learning/vision/tensorflow-keras/2020/10/13/01-Super-Resolution-CNN.html.

6. Chollet, Francois. “Building Powerful Image Classification Models Using Very Little Data.” *The Keras Blog ATOM*, https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html.

7. Walraven, Brandon. “Boost Your CNN with the Keras ImageDataGenerator.” *Medium*, Medium, 11 Feb. 2019, https://medium.com/@bcwalraven/boost-your-cnn-with-the-keras-imagedatagenerator-99b1ef262f47.

8. Tsang, Sik-Ho. “Review: SRCNN (Super Resolution).” *Medium*, Coinmonks, 24 June 2022, https://medium.com/coinmonks/review-srcnn-super-resolution-3cb3a4f67a7c.

9. “Tf.keras.preprocessing.image.imagedatagenerator.” *TensorFlow*, c.