Group Report of Final Project

Group 2

Ruiqi Li, Yixi Liang, He Huang

Department of Data Science, The George Washington University

DATS_6203_10: Machine Learning II

Amir Jafari

Dec 12, 2022

Introduction

According to Ledig et al. (2017), image super resolution refers to a task which turns low-resolution images into its high-resolution vision. Before deep neural networks, people use several traditional ways to do this task, including inter nearest method and other methods. After the development of computer vision, more and more deep neural network methods are created to implement image super resolution task. In this project, we first used traditional methods to check their performance, then built CNN network to test how deep neural networks perform on image super resolution. Then, we used autoencoder and GAN type model to help us implement image super resolution.

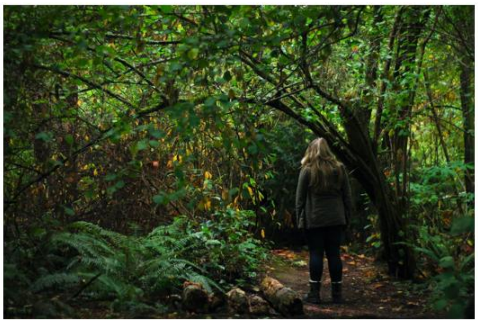
In this project, we used TensorFlow Keras to build and train our models, and finally check their performance.

Dataset

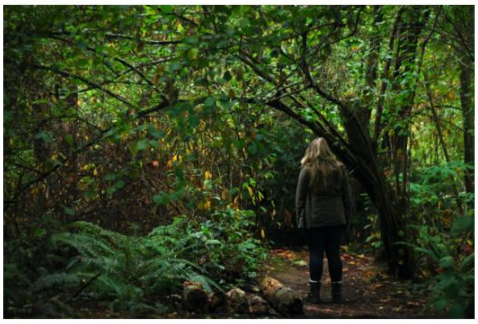
The dataset is from a Kaggle dataset which collected real lift images from *Unsplash.com*. This dataset contains 1254 high-resolution images with 3 different kinds of low-resolution vision of each image. One of the images with total 4 vision are shown below. Low-resolution images has 3 visions: good one has half pixels of the original high-resolution image; middle one has 1/4 pixels of the original; and bad one has 1/6 pixels of the original.

Figure 1

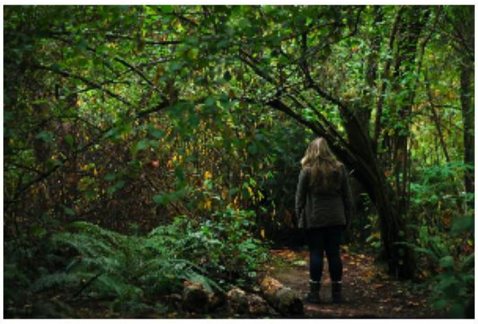
Four visions of same image



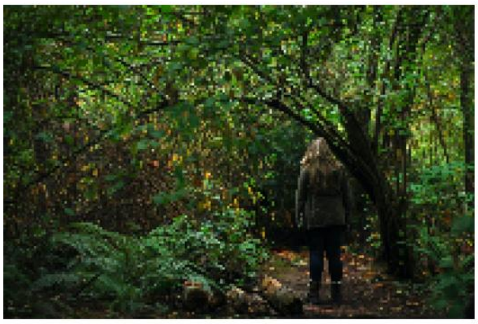
High resolution



Low Resolution
good



Low Resolution
middle



Low Resolution
bad

Traditional Methods

Except using deep neural network, we explored some tradition methods on image resolution. All the traditional methods are to set some rules to fill up the missing value of pixels when turn low-resolution image to high-resolution image. The first method is inter nearest method, which fill up the missing value with the same value of its nearest pixel. The second method is inter linear method, which uses the value of near 4 pixels to fill up this pixel. In figure 2, the original low-resolution image is the left image, inter nearest method image is shown in middle image, and inter linear method image is shown in right image. All these methods can be down with cv2 package in python. However, the performance of these traditional methods is so pool.

Figure 2

Traditional Method of Image Resolution

CNN model

To begin with, researchers from Cornell published a 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 (Figure 3). “Given a low-resolution image Y, the first convolutional layer of the SRCNN extracts a set of feature maps. The second layer maps these feature maps nonlinearly to high-resolution patch representations. The last layer combines the predictions within a spatial neighborhood to produce the final high-resolution image F(Y)” (Dong, Loy, He, Tang, page 4).

Figure 3

Structure of SRCNN

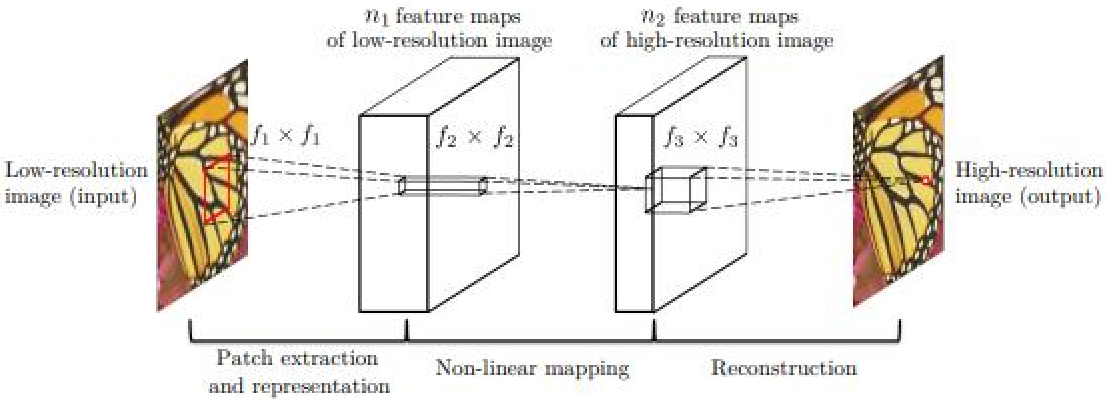


Figure 4 is the formula of the first layer. Y is the input image, W1 and B1 are filters and biases, and '*' is convolution operation. The first part does n1 times of convolution with a kernel of size of c*f1*f1, where c is number of channels and f1 is filter size, and ReLU for output.

Figure 4

Formula in first layer

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

Then in the second part of non-linear mapping, the general formula and operation are like the first part except transforming the n1-dimensional vectors to vectors with n2 dimensions (Figure 5).

The general formula and operation are like the first part except transforming the n1-dimensional vectors to vectors with n2 dimensions. B2 represents n2 dimensions and W2 represents n2*n1*f2*f2 (n2 filters of n1*f2*f2).

Figure 5

Formula in second layer

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

The third part reconstructs the image to high resolution with the operation described in Figure 4. B3 stands as c dimensions and W3 stands as c*n2*f3*f3.

Figure 6

Formula in last layer

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

For the loss function, mean square error is used in this case.

Figure 7

Mean Square Error

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

Experimental setup

To better feed the images to SRCNN, we save a dataframe that stores the paths of low-resolution images and high-resolution images and combine it with ImageDataGenerator. The rescale is set to 1/255 to scale the image's RGB coefficients so that data can be inputted. We set the validation split as 0.3 and resize the image to half of the original size (Figure 5). Figure 8 is the construction of

SRCNN. About tuning, we use 1 x 1 kernel in the second layer because “1 x 1 convolution is suggested to introduce more non-linearity to improve the accuracy” (Tsang, paragraph 12).

Figure 8

SRCNN Construction

```
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
```

Then we choose Adam as the optimizer for efficiency and set the learning rate to 0.0005. Figure 9 is the summary of the model. 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.

Figure 9

SRCNN Summary

```
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
_________________________________________________________________
```

Below are the visualization comparisons of low-resolution, high-resolution, and SRCNN-tuned version of an example image at three epoch statuses. Figure 10 is the comparison at epoch 1, Figure 11 is the one at epoch 5, and Figure 12 is the one at epoch 20. The left subplot of each figure represents the low resolution and high resolution. 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. Epoch 1's prediction is still blurry compared to the high-resolution version. 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 version of the image.

Figure 10

Comparison with SRCNN image at Epoch = 1

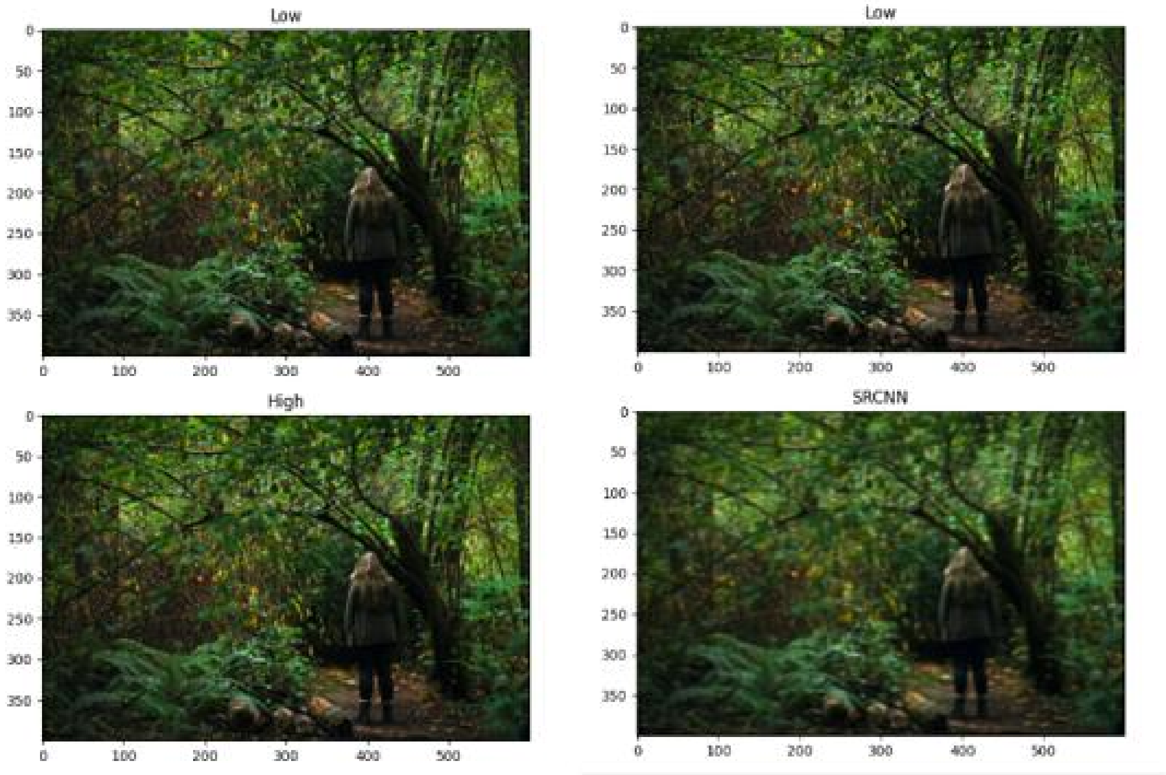


Figure 11

Comparison with SRCNN image at Epoch = 5

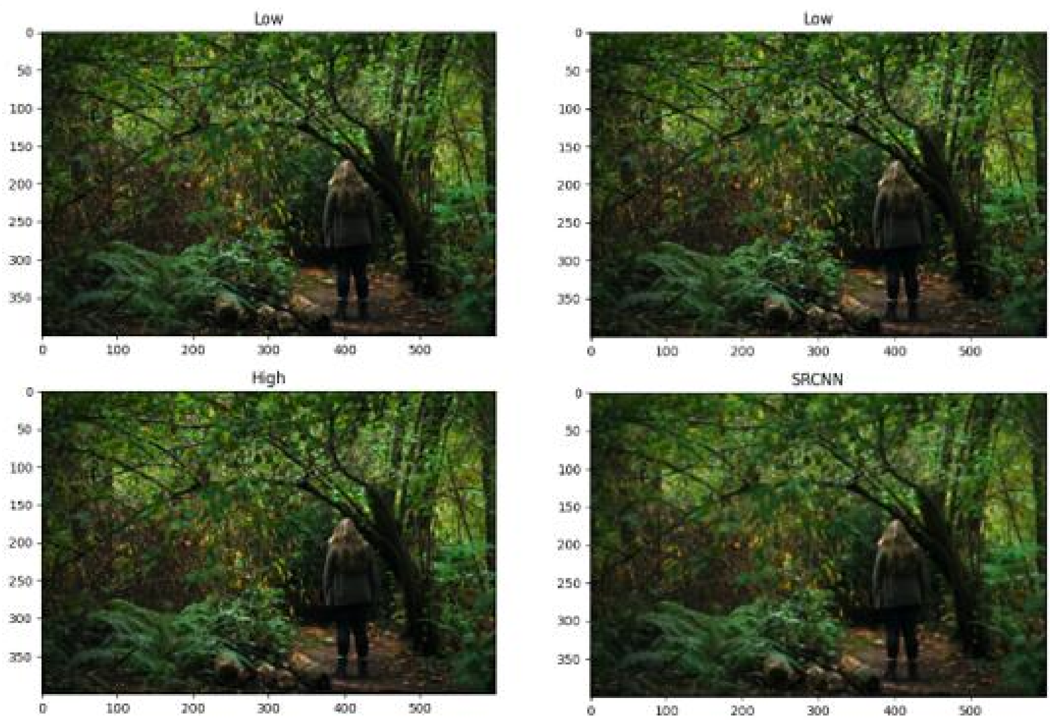
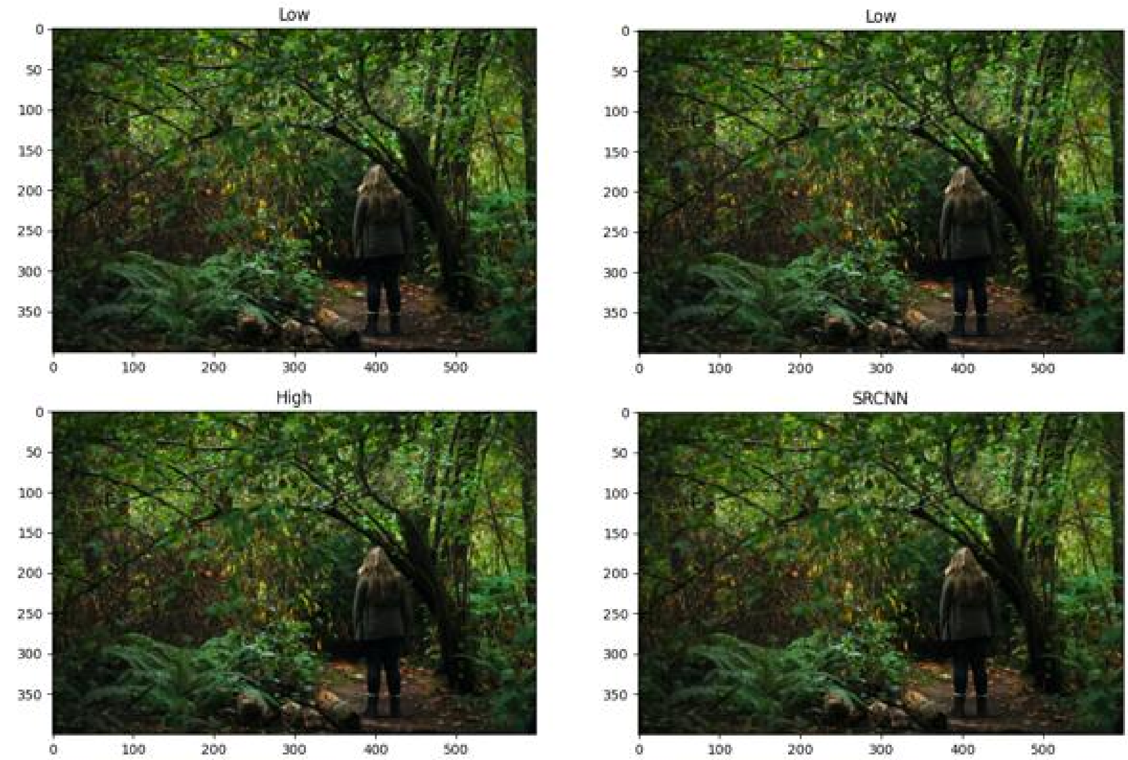


Figure 12

Comparison with SRCNN image at Epoch = 20



In addition, this is a part of the evaluation script when running with test data. The mean square error is averagely maintained around 0.002 (Figure 13).

Figure 13

Evaluation Script Feedback

```
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
```

SRCNN looks does not have many layers or too complex structure, but it is efficient. More layers and more manipulations do not necessarily mean better performance. Hyperparameter tuning is also important. When evaluating the model, we found batch size should not be big, and the image should be resized to a smaller scale because these will cause memory problems when training. To improve it, a better environment with more GPU resources could be a boost to reduce the memory issues. About the image data, even though the dataset has provided images with low resolution, we could do more data augmentation with more methods from Transform like flipping and changing color scale. There could be some other loss functions that are better than MSE in this case, too.

Autoencoder

In this section we try to use autoencoder to do the image super resolution. Firstly, we use autoencoder sample code from Keras official document to try autoencoder. And we found that one of the applications of autoencoder is Image denoising. We can use this feature to denoise the low-resolution image. And then we try the different structure of autoencoder, and finally we make our own design structure. The autoencoder designed by (Shaikh, 2022) is good. But there are still some problems remain. The image looks good but not as good as the high resolution one, so we try to change the decoder part from CNN to CNN2dtranspose.

Figure 14

Architecture of Autoencoder.

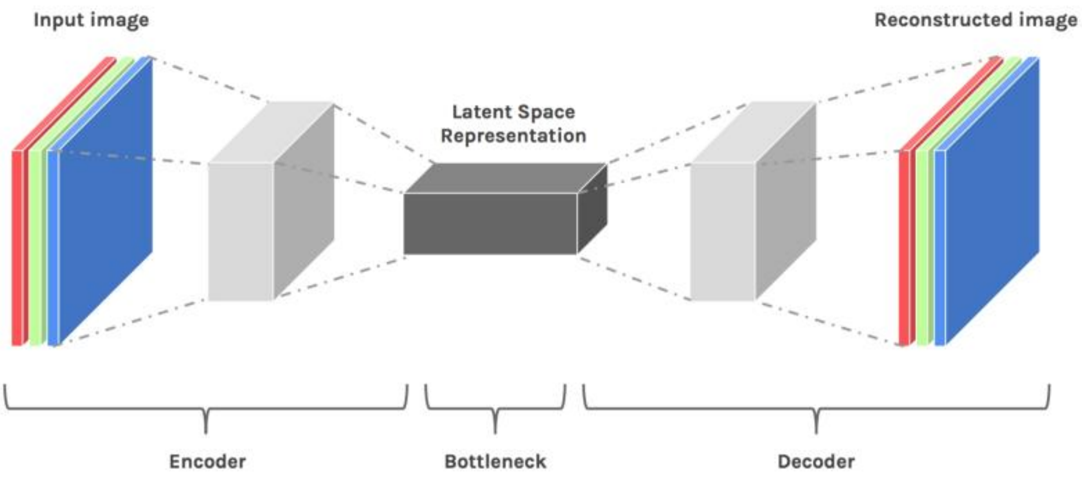
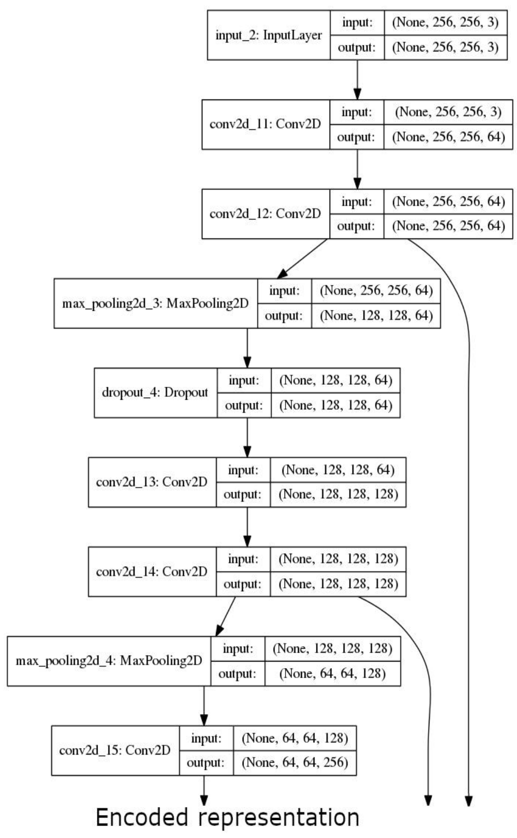


Figure 15

Autoencoder structure from (Shaikh, 2022)



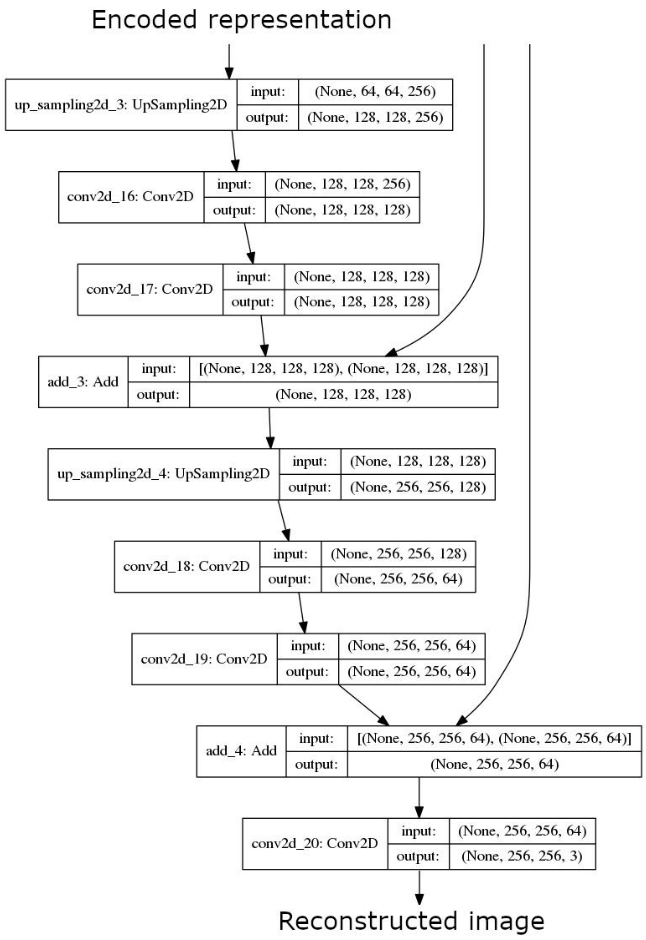


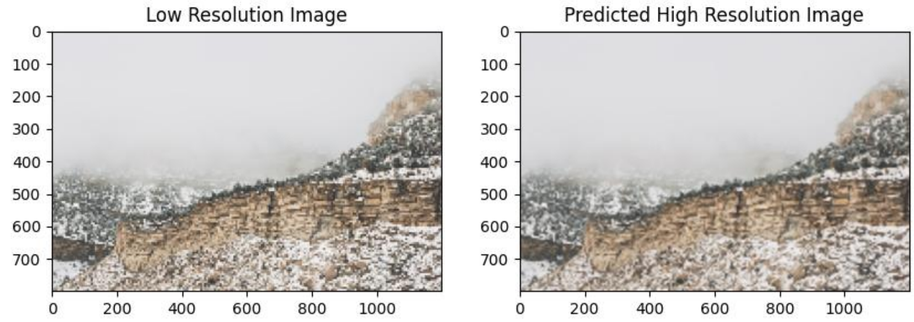
Figure 16

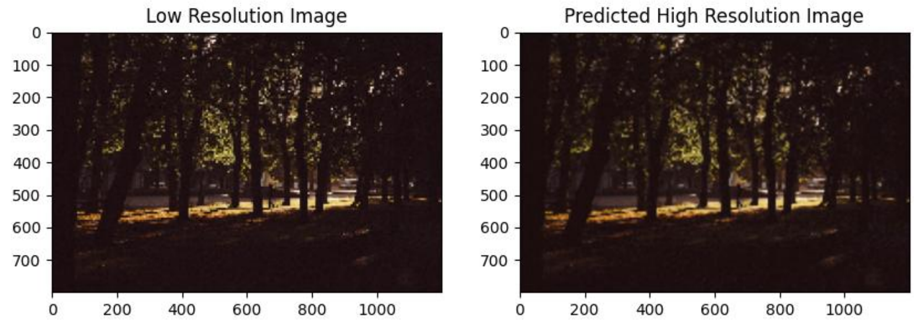
Script of autoencoder structure we change.

```
Layer (type)                Output Shape         Param #     Connected to
==================================================================================================
encoder_input (InputLayer)  [(None, 800, 1200,   0           []
                            3)]

conv2d (Conv2D)             (None, 800, 1200, 6  1792        ['encoder_input[0][0]']
                            4)

conv2d_1 (Conv2D)           (None, 800, 1200, 6  36928       ['conv2d[0][0]']
                            4)

conv2d_2 (Conv2D)           (None, 800, 1200, 6  36928       ['conv2d_1[0][0]']
                            4)

max_pooling2d (MaxPooling2D)  (None, 400, 600, 64  0         ['conv2d_2[0][0]']
                            )

dropout (Dropout)           (None, 400, 600, 64  0           ['max_pooling2d[0][0]']
                            )

conv2d_3 (Conv2D)           (None, 400, 600, 12  73856       ['dropout[0][0]']
                            8)

conv2d_4 (Conv2D)           (None, 400, 600, 12  147584      ['conv2d_3[0][0]']
                            8)

max_pooling2d_1 (MaxPooling2D)  (None, 200, 300, 12  0       ['conv2d_4[0][0]']
                            8)

conv2d_5 (Conv2D)           (None, 200, 300, 25  295168      ['max_pooling2d_1[0][0]']
                            6)

conv2d_transpose (Conv2DTransp  (None, 400, 600, 25  590080  ['conv2d_5[0][0]']
ose)                        6)

conv2d_transpose_1 (Conv2DTran  (None, 400, 600, 12  295040  ['conv2d_transpose[0][0]']
spose)                      8)

add (Add)                   (None, 400, 600, 12  0           ['conv2d_4[0][0]',
                            8)                                'conv2d_transpose_1[0][0]']

conv2d_transpose_2 (Conv2DTran  (None, 800, 1200, 1  147584  ['add[0][0]']
spose)                      28)

conv2d_transpose_3 (Conv2DTran  (None, 800, 1200, 6  73792   ['conv2d_transpose_2[0][0]']
spose)                      4)

conv2d_transpose_4 (Conv2DTran  (None, 800, 1200, 6  36928   ['conv2d_transpose_3[0][0]']
spose)                      4)

add_1 (Add)                 (None, 800, 1200, 6  0           ['conv2d_transpose_4[0][0]',
                            4)                                'conv2d_2[0][0]']

conv2d_transpose_5 (Conv2DTran  (None, 800, 1200, 3  1731    ['add_1[0][0]']
spose)                      )

==================================================================================================
Total params: 1,737,411
Trainable params: 1,737,411
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 17

Result of the autoencoder





The difference between these two structures is small. And we do not think the changes are working, and this is only randomly change we do not know the theory of this. And we also know that it might work to change the loss function, but we do not have enough to try them.

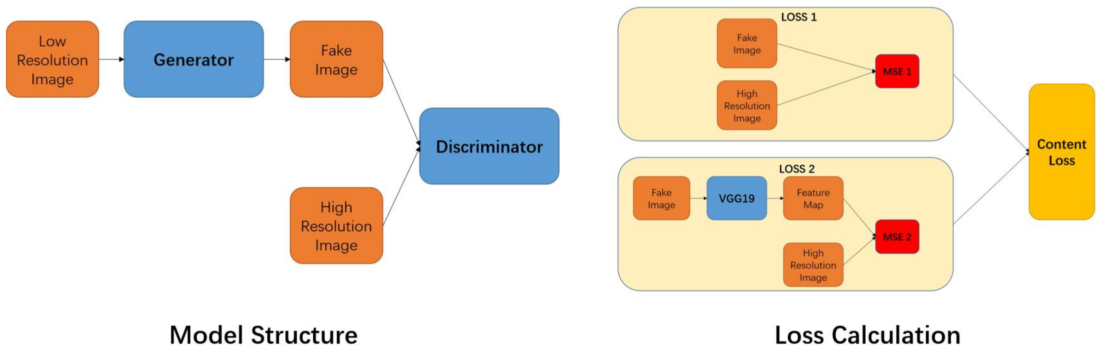
After that we also find Variational Autoencoder, hope that work for us. But we find that VAE is kind of generative model, and we do not think that will work for our project, since the output of VAE is stochastic images.

GAN

After using autoencoder, we tried to use GAN on image super resolution task. After some research, we found that the first GAN model for image super resolution task is Super Resolution GAN(SRGAN). According to Ledig et al. (2017), the basic model and the calculation of loss function for SRGAN is shown in figure below. The basic structure is just like the normal GAN. The generator generates fake image with low resolution images, then the model feeds both fake images and high-resolution images to the discriminator to detect which image is real. The biggest difference is that the loss calculation of SRGAN. Not like other GAN only calculating MSE between fake images and high-resolution images (Loss 1), the SRGAN also uses VGG19 to create feature map with the fake images and calculate the MSE between the feature map and the high-resolution images (Loss 2). The reason why SRGAN adds loss 2 to total loss is that for image super resolution, low-resolution images and high-resolution images are strictly equivalent in shape and position, and all we need to do is to add more texture on the low-resolution images, which can be exactly shown in feature maps in CNN models.

Figure 18

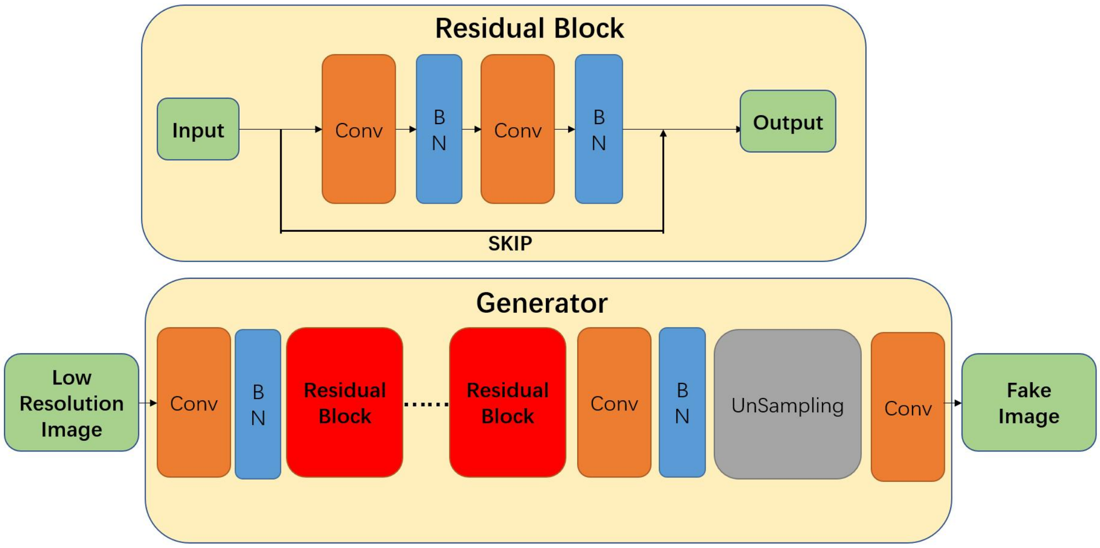
Model Structure and Loss Calculation for Generator



We built our own SRGAN model with the Generator shown below. All the activation function in this generator is PRelu function. In the SRGAN model we designed, there are total 8 residual blocks. For the discriminator, it is a simple CNN image binary classification model with LeakyReLU as activation function.

Figure 19

Structure of Generator



For training, we first freeze generator and train the discriminator, then unfreeze generator and freeze discriminator to train the generator. Every 5 epoch, we set the model output a prediction image (fake image) that the generator generates. The changes of the prediction images are shown below. From the figure, at the first several epochs, the prediction images generated by the generator only has the same shape as the original images, and the color is totally incorrect. However, after more training epochs, generator has learned how to get correct colors and add more details on the image, in other words, add more image texture that low resolution image doesn't have.

Figure 20

Changes of prediction image

Actually, except SRGAN, there are some other GAN models for image super resolution task. ESRGAN(Enhanced Super Resolution GAN) changes the structures and loss calculation of SRGAN to have a very good performance. However, building ESRGAN models and training them are beyond our capabilities. We didn't find how to build this model line by line. Also, new vision ESRGAN, real-ESRGAN, are constantly being updated. Now, the author is improving its performance on comic image.

For GAN models, we built and trained our own SRGAN model, and got very good performance on image super resolution. However, in the training, we found that GAN models always generate some random influences on image color or item position. Those random influences make the prediction image not like the high-resolution vision of the original low-resolution image, but like a brand-new image with high-resolution which is only influenced by the original image. Maybe GAN type models need to be adjusted to reduce this kind of random.

Result and Conclusion

To sum up, in this project, we tried tradition method with cv2 and deep neural network models on this image super resolution task. Tradition methods with cv2 were easy but had bad performance. Overall, deep neural network models work really good on image super resolution. The SRCNN has an overall good performance in leveraging the image solution, yet some improvements could be done. We could use loss functions other than MSE, run more epochs, and use other sizes of kernels. Autoencoder's performance is significant. However, when we use the lowest resolution as our input, the predicted image is just like blurring the boundary of the lowest resolution image, it is not actually improving the quality of the image. For GAN type models, we built and trained SRGAN model. With more and more train epoch, GAN type models performed better and better. However, when training is enough to generate high-resolution prediction image, there will be some random change to make the prediction image not exactly like the original one. Also, because all the data is real life pictures, the models performed so bad on paintings or comic images.

Several improvements can be made to this project. More efficient model structures can help us to reduce the training time and get better results. Second, we can add more types of images in training data. Different types of images like paintings and comic images can improve models' performance on these types of super resolution task. Lastly, maybe image augmentation is useful on these deep learning models.

References

Chollet, F. (n.d.). *Building Powerful Image Classification Models Using Very Little Data*. The Keras Blog ATOM. Retrieved December 11, 2022, from https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Dong, C., Loy, C. C., He, K., & Tang, X. (2015, July 31). *Image super-resolution using deep convolutional networks*. arXiv.org. Retrieved December 11, 2022, from https://arxiv.org/abs/1501.00092

Huang, Y. (n.d.). *Teaching/code_example.ipynb at master · Yuxiaohuang/teaching*. GitHub. Retrieved December 11, 2022, from 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

Kang, C. (n.d.). Super-resolution convolutional neural network. Retrieved December 12, 2022, from https://goodboychan.github.io/python/deep_learning/vision/tensorflow-keras/2020/10/13/01-Super-Resolution-CNN.html

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., ... & Shi, W. (2017). Photo-realistic single image super-resolution using a generative adversarial network. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4681-4690).

Shaikh, Q. (2022, October 14). Image super resolution (from unsplash). Kaggle. Retrieved December 11, 2022, from https://www.kaggle.com/datasets/quadeer15sh/image-super-resolution-from-unsplash

Tf.keras.preprocessing.image.imagedatagenerator : tensorflow V2.11.0. TensorFlow. (n.d.). Retrieved December 11, 2022, from https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

Tsang, S.-H. (2022, June 24). *Review: SRCNN (super resolution)*. Medium. Retrieved December 11, 2022, from https://medium.com/coinmonks/review-srcnn-super-resolution-3cb3a4f67a7c

Walraven, B. (2019, February 11). *Boost your CNN with the keras imagedatagenerator*. Retrieved December 11, 2022, from https://medium.com/@bcwalraven/boost-your-cnn-with-the-keras-imagedatagenerator-99b1ef262f47

Appendix

Part 1 CNN model

```
import cv2

import numpy as np

import pandas as pd

import os

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, UpSampling2D,
    Add, Dropout

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.model_selection import train_test_split


# take this amount for test, too large will result in our of memory and training will be
    too slow

img_data_df = pd.read_csv('image_data.csv')[:500]

# when testing, use this line

#img_data_df = pd.read_csv('image_data.csv')[:100]


def load_img(path, res):

    img = tf.keras.utils.load_img(path)

    img = tf.keras.utils.img_to_array(img)

    # original input (800, 1200, 3)
```

```
    img = tf.image.resize(img, (400,600))
    img = tf.convert_to_tensor(img)
    if res == 'hr':
        img = img / 255
        img = img.numpy()
    elif res == 'lr':
        img = img / 255
        img = img.numpy()
    return img

### get the lists of low resolution and high resolution pictures
low_list = []
high_list = []
low_list_image_path = []
high_list_image_path = []
for i in img_data_df['low_res']:
    pp = 'low res/' + i
    low_list_image_path.append(pp)
    temp_tensor_low = load_img(pp, 'lr')
    low_list.append(temp_tensor_low)
for i in img_data_df['high_res']:
    p = 'high res/' + i
    high_list_image_path.append(p)
    temp_tensor = load_img(p, 'hr')
```

```
        high_list.append(temp_tensor)


for i in img_data_df['low_res']:
        p = 'low res/' + i[:-6] + '_4.jpg'
        low_list_image_path.append(p)
        temp_tensor_low = load_img(p, 'lr')
        low_list.append(temp_tensor_low)
for i in img_data_df['high_res']:
        p = 'high res/' + i
        high_list_image_path.append(p)
        temp_tensor = load_img(p, 'hr')
        high_list.append(temp_tensor)


for i in img_data_df['low_res']:
        ppp = 'low res/' + i[:-6] + '_6.jpg'
        low_list_image_path.append(ppp)
        temp_tensor_low = load_img(ppp, 'lr')
        low_list.append(temp_tensor_low)
for i in img_data_df['high_res']:
        p = 'high res/' + i
        high_list_image_path.append(p)
        temp_tensor = load_img(p, 'hr')
        high_list.append(temp_tensor)
```

```python
### show some sample pictures
c = 1
for i in zip(low_list[:3], high_list[:3]):
    fig, axs = plt.subplots(2,1,figsize=(6, 8))
    axs[0].imshow(i[0])
    axs[0].title.set_text('Low')
    axs[1].imshow(i[1])
    axs[1].title.set_text('High')
    plt.tight_layout()
    # plt.savefig('initial_{}_2'.format(c))
    # plt.savefig('initial_{}_4'.format(c))
    # plt.savefig('initial_{}_6'.format(c))
    c+=1
    plt.show()

# p = {'low_res_paths': super_low_list_image_path,
#         'high_res_paths': high_list_image_path}
# p = {'low_res_paths': super_super_low_list_image_path,
#         'high_res_paths': high_list_image_path}
p = {'low_res_paths': low_list_image_path,
     'high_res_paths': high_list_image_path}

p = pd.DataFrame(data = p)
```

```
p, test = train_test_split(p, test_size=0.2)

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)
eval_datagen = ImageDataGenerator(rescale = 1/255)

# 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')

# training low resolution
train_lowresimage_generator = train_datagen.flow_from_dataframe(
        p,
        x_col = 'low_res_paths',
```

```
        target_size = original_shape,
        class_mode = None,
        batch_size = batch_size,
        interpolation='nearest',
        seed = 42,
        subset = 'training')

# val high resolution
val_hiresimage_generator = test_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 = 'validation')

# val low resolution
val_lowresimage_generator = test_datagen.flow_from_dataframe(
        p,
        x_col = 'low_res_paths',
        target_size = original_shape,
        class_mode = None,
```

```
        batch_size = batch_size,
        interpolation='nearest',
        seed = 42,
        subset='validation')

# test high resolution
test_hiresimage_generator = eval_datagen.flow_from_dataframe(
        test,
        x_col = 'high_res_paths',
        target_size = original_shape,
        class_mode = None,
        batch_size = batch_size,
        interpolation='nearest',
        seed = 42)

# test low resolution
test_lowresimage_generator = eval_datagen.flow_from_dataframe(
        test,
        x_col = 'low_res_paths',
        target_size = original_shape,
        class_mode = None,
        batch_size = batch_size,
        interpolation='nearest',
        seed = 42)
```

```
train_generator = zip(train_lowresimage_generator, train_hiresimage_generator)
val_generator = zip(val_lowresimage_generator, val_hiresimage_generator)
test_generator = zip(test_lowresimage_generator, test_hiresimage_generator)

def imageGenerator(generator):
    for (low_res, hi_res) in generator:
            yield (low_res, hi_res)

train_img_gen = imageGenerator(train_generator)
val_image_gen = imageGenerator(val_generator)

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


train_len = train_hiresimage_generator.samples
steps_per_epoch = train_len // batch_size
val_len = val_hiresimage_generator.samples
validation_steps = val_len // batch_size


path = 'srcnn_model.h5'
isExist = os.path.exists(path)


if not isExist:
    model = model()
    model.summary()
    checkpoint = tf.keras.callbacks.ModelCheckpoint('srcnn_model.h5',
                                                    verbose = 2,
                                                    save_best_only =
    True)
    early_stopping = tf.keras.callbacks.EarlyStopping(verbose = 2,
                                                      patience = 10)
    plateau = tf.keras.callbacks.ReduceLROnPlateau(verbose = 2,
                                                   factor = 0.1,
                                                   patience = 5)
    model.fit(
    train_img_gen,
```

```
        batch_size = batch_size,
        steps_per_epoch = steps_per_epoch,
        validation_data = val_image_gen,
        validation_steps = validation_steps,
        epochs = 50,
        verbose = 2,
        callbacks = [plateau, early_stopping, checkpoint])

        t = iter(test_generator)
        count = 0
        for i in range(len(test)):
            if count == 15:
                break
            x = next(t)
            x_train, x_test = x[0], x[1]
            model.evaluate(x=x[0], y=x[1], verbose=2)
            count += 1
else:
        model = tf.keras.models.load_model('srcnn_model.h5')
        model.summary()
        t = iter(test_generator)
        count = 0
        for i in range(len(test)):
            if count == 15:
```

```
                break
            x = next(t)
            x_train, x_test = x[0], x[1]
            model.evaluate(x=x[0], y=x[1], verbose=2)
            count += 1

sample_datagen = ImageDataGenerator(rescale=1/255, validation_split = 0.3)
s = {'low_res_paths': low_list_image_path[:12],
     'high_res_paths': high_list_image_path[:12]}
# s = {'low_res_paths': super_low_list_image_path[:12],
#      'high_res_paths': high_list_image_path[:12]}
# s = {'low_res_paths': super_super_low_list_image_path[:12],
#      'high_res_paths': high_list_image_path[:12]}

s = pd.DataFrame(data = s)
# sample high res
sample_hiresimage_generator = sample_datagen.flow_from_dataframe(
    s,
    x_col = 'high_res_paths',
    target_size = original_shape,
    class_mode = None,
    batch_size = batch_size,
    interpolation = 'nearest',
    seed = 42,
```

```
        subset = 'validation')

# sample low res

sample_lowresimage_generator = sample_datagen.flow_from_dataframe(

        s,

        x_col = 'low_res_paths',

        target_size = original_shape,

        class_mode = None,

        batch_size = batch_size,

        interpolation = 'nearest',

        seed = 42,

        subset = 'validation')

sample_generator = zip(sample_lowresimage_generator,
        sample_hiresimage_generator)



cc = 1

j = iter(sample_generator)

one = next(j)

img1 = one[0]

sr1 = model.predict(img1, batch_size = batch_size)

img1 = cv2.resize(img1[0], (600, 400))

sr1 = cv2.resize(sr1[0], (600, 400))

fig, axs = plt.subplots(2, 1, figsize = (6, 8))

axs[0].imshow(img1)

axs[0].title.set_text('Low')
```

```
axs[1].imshow(sr1)

axs[1].title.set_text('SRCNN')

plt.tight_layout()

#plt.savefig('low_high_pred_{}_2'.format(cc))

#plt.savefig('low_high_pred_{}_4'.format(cc))

#plt.savefig('low_high_pred_{}_6'.format(cc))

cc += 1

plt.show()


two = next(j)

img2 = two[0]

sr2 = model.predict(img2, batch_size = batch_size)

img2 = cv2.resize(img2[0], (600, 400))

sr2 = cv2.resize(sr2[0], (600, 400))

fig, axs = plt.subplots(2, 1, figsize = (6, 8))

axs[0].imshow(img2)

axs[0].title.set_text('Low')

axs[1].imshow(sr2)

axs[1].title.set_text('SRCNN')

plt.tight_layout()

#plt.savefig('low_high_pred_{}_2'.format(cc))

#plt.savefig('low_high_pred_{}_4'.format(cc))

#plt.savefig('low_high_pred_{}_6'.format(cc))

cc += 1
```

```
plt.show()


three = next(j)
img3 = three[0]
sr3 = model.predict(img3, batch_size = batch_size)
img3 = cv2.resize(img3[0], (600, 400))
sr3 = cv2.resize(sr3[0], (600, 400))
fig, axs = plt.subplots(2, 1, figsize = (6, 8))
axs[0].imshow(img3)
axs[0].title.set_text('Low')
axs[1].imshow(sr3)
axs[1].title.set_text('SRCNN')
plt.tight_layout()
#plt.savefig('low_high_pred_{}_2'.format(cc))
#plt.savefig('low_high_pred_{}_4'.format(cc))
#plt.savefig('low_high_pred_{}_6'.format(cc))
cc += 1
plt.show()
```

Part 2 Autoencoder

```
import numpy as np
import pandas as pd
import os
import re
```

```
import tensorflow as tf

from keras.layers import Input, Dense, Conv2D, MaxPooling2D, Dropout,
        Conv2DTranspose, UpSampling2D, add, LeakyReLU

from keras.models import Model

from keras import regularizers



# from keras.preprocessing.image import load_img, img_to_array



from keras.preprocessing.image import ImageDataGenerator

from keras.utils.image_utils import load_img

from tensorflow.python.keras.callbacks import ModelCheckpoint, EarlyStopping,
        ReduceLROnPlateau

import matplotlib.pyplot as plt

import cv2 as cv

#%% #
        -----------------------------------------------------------------------------------------------
        -----------------

IMAGE_WIDTH = 800

IMAGE_Length = 1200



#%% #
        -----------------------------------------------------------------------------------------------
        -----------------

cur_file = os.getcwd()

base_directory = cur_file + '/Image Super Resolution - Unsplash'

hires_folder = os.path.join(base_directory, 'high res')
```

```
lowres_folder = os.path.join(base_directory, 'low res')

data = pd.read_csv(base_directory + f"/image_data.csv", encoding='ISO-8859-1')
data['low_res'] = data['low_res'].apply(lambda x: os.path.join(lowres_folder,x))
data['high_res'] = data['high_res'].apply(lambda x: os.path.join(hires_folder,x))
print(data.head())

batch_size = 1

image_datagen = ImageDataGenerator(rescale=1./255,validation_split=0.15)
mask_datagen = ImageDataGenerator(rescale=1./255,validation_split=0.15)

train_hiresimage_generator = image_datagen.flow_from_dataframe(
        data,
        x_col='high_res',
        target_size=(IMAGE_WIDTH, IMAGE_Length),
        class_mode = None,
        batch_size = batch_size,
        seed=42,
        subset='training')

train_lowresimage_generator = image_datagen.flow_from_dataframe(
        data,
        x_col='low_res',
```

```
            target_size=(IMAGE_WIDTH, IMAGE_Length),
            class_mode = None,
            batch_size = batch_size,
            seed=42,
            subset='training')

val_hiresimage_generator = image_datagen.flow_from_dataframe(
            data,
            x_col='high_res',
            target_size=(IMAGE_WIDTH, IMAGE_Length),
            class_mode = None,
            batch_size = batch_size,
            seed=42,
            subset='validation')

val_lowresimage_generator = image_datagen.flow_from_dataframe(
            data,
            x_col='low_res',
            target_size=(IMAGE_WIDTH, IMAGE_Length),
            class_mode = None,
            batch_size = batch_size,
            seed=42,
            subset='validation')
```

```
train_generator = zip(train_lowresimage_generator, train_hiresimage_generator)

val_generator = zip(val_lowresimage_generator, val_hiresimage_generator)


def imageGenerator(train_generator):

    for (low_res, hi_res) in train_generator:

            yield (low_res, hi_res)


n = 0

for i,m in train_generator:

    img,out = i,m


    if n < 5:

        fig, axs = plt.subplots(1 , 2, figsize=(20,5))

        axs[0].imshow(img[0])

        axs[0].set_title('Low Resolution Image')

        axs[1].imshow(out[0])

        axs[1].set_title('High Resolution Image')

        plt.show()

        n+=1

    else:

        break
```

```
input_img = Input(shape=(IMAGE_WIDTH, IMAGE_Length, 3),
        name='encoder_input')


l1 = Conv2D(64, (3, 3), padding='same', activation='relu')(input_img)

l2 = Conv2D(64, (3, 3), padding='same', activation='relu')(l1)

l3 = Conv2D(64, (3, 3), padding='same', activation='relu')(l2)

l4 = MaxPooling2D(padding='same')(l3)

l4 = Dropout(0.3)(l4)

l5 = Conv2D(128, (3, 3),    padding='same', activation='relu')(l4)

l6 = Conv2D(128, (3, 3), padding='same', activation='relu')(l5)

l7 = MaxPooling2D(padding='same')(l6)

l8 = Conv2D(256, (3, 3), padding='same', activation='relu')(l7)


# l9 = UpSampling2D()(l8)

l10 = Conv2DTranspose(256, (3, 3), padding='same', activation='relu', strides=2,
        output_padding=1)(l8)

l11 = Conv2DTranspose(128, (3, 3), padding='same', activation='relu')(l10)



l12 = add([l6, l11])

# l13 = UpSampling2D()(l12)

l14 = Conv2DTranspose(128, (3, 3), padding='same', activation='relu', strides=2,
        output_padding=1)(l12)

l15 = Conv2DTranspose(64, (3, 3), padding='same', activation='relu')(l14)

l16 = Conv2DTranspose(64, (3, 3), padding='same', activation='relu')(l15)
```

```
l17 = add([l16, l3])

decoded = Conv2DTranspose(3, (3, 3), padding='same', activation='relu')(l17)

autoencoder = Model(input_img, decoded)
autoencoder =
    tf.keras.models.load_model('autoencoder_change_deconvolution_1200.h5')
autoencoder.compile(optimizer='adam', loss='mean_squared_error',
    metrics=['accuracy'])

autoencoder.summary()

train_samples = train_hiresimage_generator.samples
val_samples = val_hiresimage_generator.samples

train_img_gen = imageGenerator(train_generator)
val_image_gen = imageGenerator(val_generator)

model_path = "autoencoder_change_deconvolution_1200.h5"
checkpoint = ModelCheckpoint(model_path,
                             monitor="val_loss",
                             mode="min",
                             save_best_only = True,
                             verbose=1)
```

```
earlystop = EarlyStopping(monitor = 'val_loss',
                          min_delta = 0,
                          patience = 9,
                          verbose = 1,
                          restore_best_weights = True)

learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss',
                                            patience=5,
                                            verbose=1,
                                            factor=0.2,
                                            min_lr=0.00000001)

hist = autoencoder.fit(train_img_gen,
                       steps_per_epoch=train_samples//batch_size,
                       validation_data=val_image_gen,
                       validation_steps=val_samples//batch_size,
                       epochs=8, callbacks=[earlystop, checkpoint,
    learning_rate_reduction])

plt.figure(figsize=(20,8))
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
```

```
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()


n = 0
for i,m in val_generator:
    img,mask = i,m
    sr1 = autoencoder.predict(img)
    if n < 20:
        fig, axs = plt.subplots(1 , 3, figsize=(20,4))
        axs[0].imshow(img[0])
        axs[0].set_title('Low Resolution Image')
        axs[1].imshow(mask[0])
        axs[1].set_title('High Resolution Image')
        axs[2].imshow(sr1[0])
        axs[2].set_title('Predicted High Resolution Image')
        plt.show()
        n+=1
        # cv.imwrite(f'test_img/test_N_6_{n}.jpg', sr1[0]*255)
    else:
        break
```

Part 3 GAN

```
import tensorflow as tf

import numpy as np

import pandas as pd

from tensorflow.keras import layers

import os

import matplotlib.pyplot as plt

from tqdm import tqdm

from PIL import Image


#---------------------------------------------------------------------------------------------------------------
-----------

# hyperparameter

EPOCH = 100


LR_all = 0.001

LR_dis = 0.0001


low_shape = 128

high_shape = 512


batch_size = 1
```

```
#--------------------------------------------------------------------------------------------------------
-----------

# data preprocessing

base_directory = '../data'

hires_folder = os.path.join(base_directory, 'high res')

lowres_folder = os.path.join(base_directory, 'low res')


img_data_df = pd.read_csv('image_data.csv')


def load_img(path, is_hr=False, shape=(128, 192)):

    img = tf.keras.utils.load_img(path)

    img = tf.keras.utils.img_to_array(img)

    img = tf.image.resize(img, shape)

    if is_hr:

        img = img / 255.0

        # img = (img - 127.5) / 127.5

    else:

        img = img / 255.0

    return img.numpy()


lr_imgs = np.array([load_img(os.path.join(lowres_folder, i)) for i in
    tqdm(img_data_df['low_res'][:300])])
```

```
hr_imgs = np.array([load_img(os.path.join(hires_folder, i), True, (512, 768)) for i in
    tqdm(img_data_df['high_res'][:300])])


def plot_imgs(lr_img, hr_img):

    k = np.random.randint(0, len(lr_imgs), (3))

    lr = [lr_img[x] for x in k]

    hr = [hr_img[x] for x in k]

    for i in range(6):

        if i < 3:

            plt.subplot(2, 3, i+1)

            plt.imshow(lr[i % 3])

            plt.axis('off')

        else:

            plt.subplot(2, 3, i+1)

            plt.imshow(hr[i % 3])

            plt.axis('off')

    plt.show()


# plot_imgs(lr_imgs, hr_imgs)


#----------------------------------------------------------------------------------------------------
    -----------

# model structure

def residual_block(in_layer, filters, stride=1):

    # weight initializer
```

```
    ini = tf.keras.initializers.RandomNormal(stddev=0.02)

    rb = layers.Conv2D(filters, (3, 3), padding='same', strides=stride,
    kernel_initializer=ini)(in_layer)

    rb = layers.BatchNormalization()(rb)

    rb = layers.PReLU()(rb)

    rb = layers.Conv2D(filters, (3, 3), padding='same', strides=stride,
    kernel_initializer=ini)(rb)

    rb = layers.BatchNormalization()(rb)

    rb = layers.Add()([in_layer, rb])

    return rb


def upsample(in_layer):

    ini = tf.keras.initializers.RandomNormal(stddev=0.02)

    x = layers.Conv2D(256, (3, 3), strides=1, kernel_initializer=ini,
    padding='same')(in_layer)

    x = layers.UpSampling2D()(x)

    x = layers.PReLU(shared_axes=[1, 2])(x)

    return x




def build_generator(image_shape, n_res=16):

    # weight initializer

    ini = tf.keras.initializers.RandomNormal(stddev=0.02)



    input_img = layers.Input(shape=image_shape)
```

```
    g = layers.Conv2D(64, (9, 9), strides=1, padding='same',
    kernel_initializer=ini)(input_img)

    g = layers.PReLU(shared_axes=[1, 2])(g)

    g1 = g

    for _ in range(n_res):

        g = residual_block(g, 64)

    g = layers.Conv2D(64, (3, 3), padding='same', strides=1,
    kernel_initializer=ini)(g)

    g = layers.BatchNormalization()(g)

    g = layers.Add()([g1, g])

    g = upsample(g)

    g = upsample(g)

    g = layers.Conv2D(3, (9, 9), strides=1, padding='same', kernel_initializer=ini)(g)

    model = tf.keras.models.Model(input_img, g)

    return model


def convblock(in_layer, filters, stride=1):

    # weight initializer

    ini = tf.keras.initializers.RandomNormal(stddev=0.02)

    x = layers.Conv2D(filters, (3, 3), strides=stride, padding='same',
    kernel_initializer=ini)(in_layer)

    x = layers.BatchNormalization()(x)

    x = layers.LeakyReLU(alpha=0.2)(x)

    return x
```

```
def build_discriminator(image_shape):

    # weight initializer

    ini = tf.keras.initializers.RandomNormal(stddev=0.02)

    # input

    input_img = layers.Input(shape=image_shape)

    d = layers.Conv2D(64, (3, 3), strides=1, padding='same',
    kernel_initializer=ini)(input_img)

    d = layers.LeakyReLU(alpha=0.2)(d)

    d = convblock(d, 64, 2)

    d = convblock(d, 128)

    d = convblock(d, 128, 2)

    d = convblock(d, 256)

    d = convblock(d, 256, 2)

    d = convblock(d, 512)

    d = convblock(d, 512, 2)

    d = layers.Dense(1024)(d)

    d = layers.LeakyReLU(alpha=0.2)(d)

    d = layers.Dense(1)(d)

    d = layers.Activation('sigmoid')(d)

    model = tf.keras.models.Model(inputs=input_img, outputs=d)

    return model



def vgg_model(hr_shape):

    vgg = tf.keras.applications.vgg19.VGG19(include_top=False,
    weights='imagenet', input_shape=hr_shape)
```

```
    out = tf.keras.layers.Rescaling(1/12.75)(vgg.layers[15].output)

    return tf.keras.models.Model(vgg.inputs, out)




def composite_model(gen, disc, vgg, lr_shape, hr_shape):

    lr_in = layers.Input(shape=lr_shape)

    hr_in = layers.Input(shape=hr_shape)

    gen_img = gen(lr_in)

    vgg_feature = vgg(gen_img)    # content loss


    disc.trainable = False

    validity = disc(gen_img)    # adversarial loss

    return tf.keras.models.Model(inputs=[lr_in, hr_in], outputs=[validity,
    vgg_feature])


#-----------------------------------------------------------------------------------------------------
    -----------

# build model

lr_shape = lr_imgs[0].shape

hr_shape = hr_imgs[0].shape


strategy = tf.distribute.get_strategy()

with strategy.scope():

    # lr image -> hr image

    gen_model = build_generator(lr_shape, n_res=8)
```

```
    gen_model = tf.keras.models.load_model('save_epoch_gen.h5')


    # validate lr and hr

    disc = build_discriminator(hr_shape)

    # disc = tf.keras.models.load_model('save_epoch_disc.h5')

    disc.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=LR_dis,
    beta_1=0.9),

                 loss='mse', metrics=['accuracy'])


    # vgg for feature extraction

    vgg = vgg_model(hr_shape)

    vgg.trainable = False


    # composite model

    composite = composite_model(gen_model, disc, vgg, lr_shape, hr_shape)

    composite.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=LR_all,
    beta_1=0.9),

                      loss=['mse', 'mse'],

                      loss_weights=[1e-3, 1])


#-----------------------------------------------------------------------------------------------------
    -----------

# prepare for train

train_lr_batch = []

train_hr_batch = []
```

```
for i in tqdm(range(int(len(lr_imgs)/batch_size))):

    start = i * batch_size

    end = start + batch_size

    train_lr_batch.append(lr_imgs[start:end])

    train_hr_batch.append(hr_imgs[start:end])


#----------------------------------------------------------------------------------------------------
    -----------

# Training

print('Training start')

test = load_img(os.path.join('../data/low res/1_6.jpg'))


with strategy.scope():

    epochs = EPOCH

    for e in range(epochs):

        disc_out = disc.output_shape

        patch_shape = (disc_out[1], disc_out[2], disc_out[3])

        fake_label = np.zeros((batch_size, *patch_shape))

        real_label = np.ones((batch_size, *patch_shape))

        g_losses = []

        d_losses = []


        for b in tqdm(range(len(train_lr_batch))):

            lr_img_ap = np.array(train_lr_batch[b])
```

```
            hr_img_ap = np.array(train_hr_batch[b])


            # generate fake image using generator

            fake_img = gen_model.predict_on_batch(lr_img_ap)

            # train the discriminator to distinguish between real and fake

            disc.trainable = True

            d_gen_loss = disc.train_on_batch(fake_img, fake_label)

            d_real_loss = disc.train_on_batch(hr_img_ap, real_label)


            avg_d_loss = np.add(d_gen_loss, d_real_loss) * 0.5

            d_losses.append(avg_d_loss)

            disc.trainable = False

            # VGG image feature

            image_feature = vgg.predict(hr_img_ap)


            # train generator using the composite model

            g_loss, _, _ = composite.train_on_batch([lr_img_ap, hr_img_ap],
[real_label, image_feature])


            g_losses.append(g_loss)


        g_losses = np.array(g_losses)

        d_losses = np.array(d_losses)
```

```
        g_loss = np.sum(g_losses, axis=0) / len(g_losses)
        d_loss = np.sum(d_losses, axis=0) / len(d_losses)

        print(f'epochs :: {e} d_loss :: {d_loss} g_loss :: {g_loss}')
        if (e + 1) % 5 == 0:
            gen_model.save('save_epoch_gen.h5')
            # disc.save('save_epoch_disc.h5')
            print('Model Saved')
            pred = gen_model.predict(np.expand_dims(test, axis=0))
            pred = np.squeeze(pred)
            pred = pred * 255
            pred = tf.clip_by_value(pred, 0, 255)
            pred = Image.fromarray(tf.cast(pred, tf.uint8).numpy())
            N = (e + 1) + 145
            pred.save('gan_{}.jpg'.format(N), quality=95)



#-----------------------------------------------------------------------------------------------------------
test = load_img(os.path.join('../data/low res/1_6.jpg'))

pred = gen_model.predict(np.expand_dims(test, axis=0))
pred = np.squeeze(pred)
```

```
new_pred = pred


plt.subplot(1, 2, 1)

plt.imshow(new_pred)

plt.title('Prediction')

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(test)

plt.title('Original')

plt.axis('off')

plt.show()


print('all fine!')
```