Out[289]:

Click here to toggle on/off the raw code.

1. Import image ¶

First we import our image, we totally have 1500 low resolution images and corresponding high resolution image

```
Time: 26.052173852920532
```

Metric

We are going to use PSNR as our final measurement metric.

Define basic parameters

We have 3 channels (RBG), and set seed and define the number of pixels taken from each picture to be our training data.

RMK

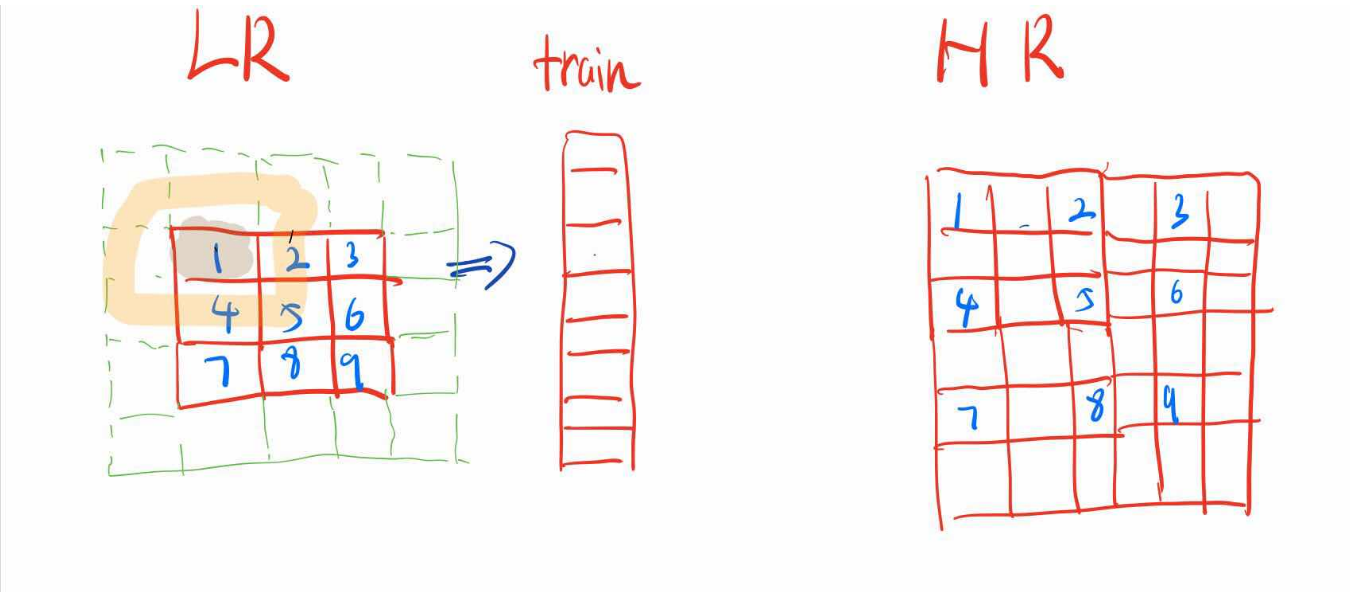
- We choose sample_size = 100 rather than 1000 because in the further research, we found that the result of 1000 pixels per image has similar PSNR with 100 pixels per image, but have significant improvement of speed. We will discuss it later.

1.1 Try OpenCV Function

We implement the *INTER_NEAREST* function in OpenCV to see the performance.

```
psnr: 25.783482291479018
Time: 298.13437128067017
```

2. Preparation



2.1. Get Features X and respond y

We construct our features as follow:

1. Given a low resolution image, we padding the image first.
2. Pick n pixels (100) from each picture, for each point, we select surrouding 8 pixels as our features, and vectorize them.
3. Find the corresponding pixels(y) in high resolution image to each pixel that we randomly picked in low resolution image. (Remark: the index of corresponding y must be even, once we find the corresponding one in high resolution image (i.e the ith row and jth column), we pick HR[i,j],HR[i+1,j],HR[i,j+1],HR[i+1,j+1] as responds, where HR is hight resolution image.)
4. Centralize both X's and y's using the original picked pixel LR[i,j].
5. repeat 1-4 for 3 channels

2.2. Prediction (12 models)

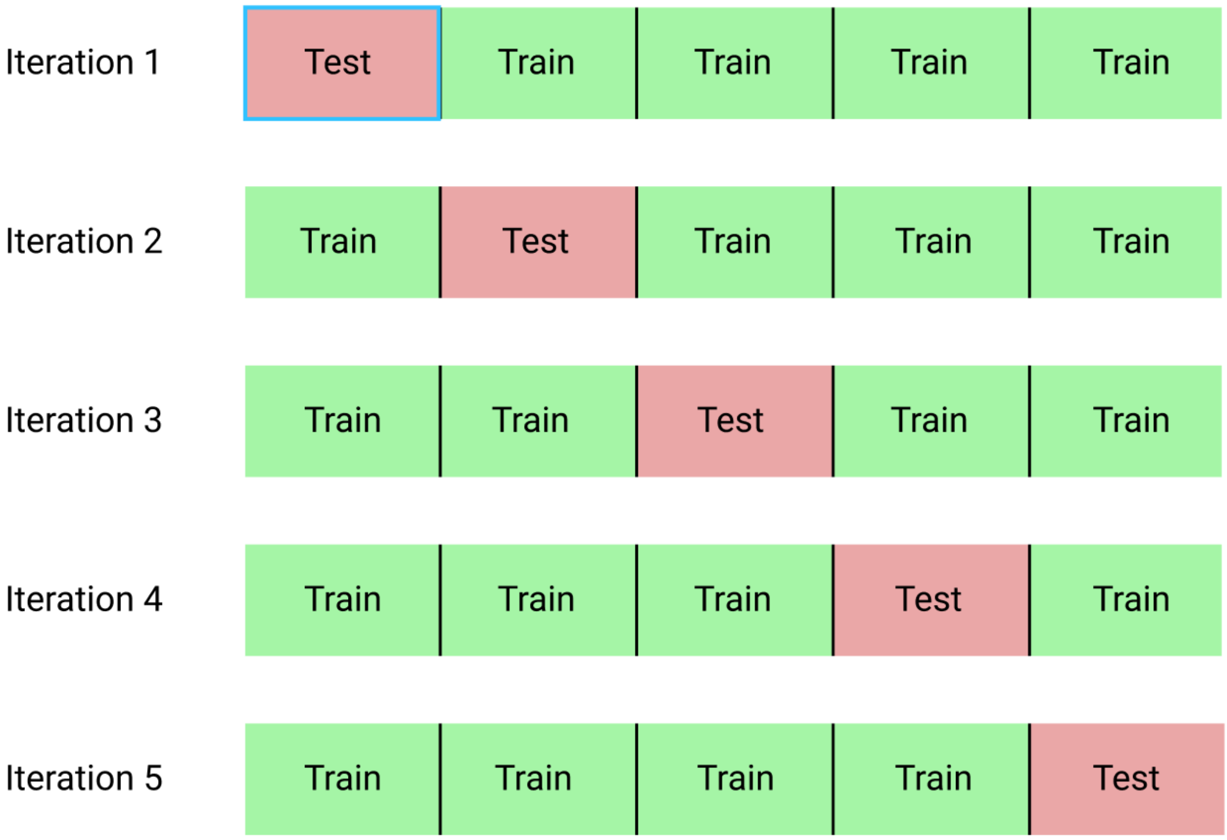
We have 4 pixels to be predicted, as requested in the description. In our implementation, we provide two predictions. The first one is a combination of "fit" and "prediction", which correspond to use for "cross validation" and "grid search". We will talk about “grid search” later. The other one, which named "Predict_test" is used for predict the actual test data.

2.3. Grid Search

Since we have to train 12 models, the GridSearchCV impletemented in sklearn won't work, thus we create a new grid search function to find the paramters corresponding to each model.

What we did is using for-loop to go through all 3 channels and 4 models per channel. Also, we can set the function to work parallelly to speed up

2.4. Cross Validation



We devide our data in n_folds (defalut 3), then iteratedly,use (n_folds - 1) folds as training set and the other fold as validation set, train the models and then predict value in the validation set.

2.5. Super resolution

Here, given a picture, we firstly do the padding. Then, instead of randomply pick 100 points, we use all the points to construct features. Then, we implement our models to get y, which has ($dim_y = (height * weight)X4$). To be more specific, every row has four points, which correspond to the lower resolution image's i-th pixel's super resolution value. Also, we need to add the central back.

3. Prepare data

Get our features and responds

```
Time: 71.74953961372375
```

```
 X shape: (150000, 8, 3)
 y shape: (150000, 4, 3)
```

3.1. Train & Test split

```
 X_train shape: (120000, 8, 3)
 y_train shape: (120000, 4, 3)

 X_test shape: (30000, 8, 3)
 y_test shape: (30000, 4, 3)
```

4. Baseline model (gbm)

4.1. Tuning models

4.1.1. Grid Search

```
Model 0
Best params {'learning_rate': 0.2, 'min_samples_leaf': 8, 'min_samples_spl
it': 2}
Best MSE: 148.95659474923593
Model 1
Best params {'learning_rate': 0.1, 'min_samples_leaf': 8, 'min_samples_spl
it': 2}
Best MSE: 149.9695441668164
Model 2
Best params {'learning_rate': 0.1, 'min_samples_leaf': 10, 'min_samples_sp
lit': 10}
Best MSE: 155.06416635649742
Model 3
Best params {'learning_rate': 0.1, 'min_samples_leaf': 10, 'min_samples_sp
lit': 2}
Best MSE: 150.561882053127
Model 4
Best params {'learning_rate': 0.2, 'min_samples_leaf': 2, 'min_samples_spl
it': 5}
Best MSE: 141.21947290195465
Model 5
Best params {'learning_rate': 0.1, 'min_samples_leaf': 10, 'min_samples_sp
lit': 2}
Best MSE: 143.65584948963195
Model 6
Best params {'learning_rate': 0.1, 'min_samples_leaf': 10, 'min_samples_sp
lit': 5}
Best MSE: 147.99993570980564
Model 7
Best params {'learning_rate': 0.1, 'min_samples_leaf': 10, 'min_samples_sp
lit': 10}
Best MSE: 143.84592645414705
Model 8
Best params {'learning_rate': 0.2, 'min_samples_leaf': 10, 'min_samples_sp
lit': 2}
Best MSE: 153.43463651232437
Model 9
Best params {'learning_rate': 0.1, 'min_samples_leaf': 4, 'min_samples_spl
it': 5}
Best MSE: 155.41662506564035
Model 10
Best params {'learning_rate': 0.1, 'min_samples_leaf': 10, 'min_samples_sp
lit': 2}
Best MSE: 158.1472241277593
Model 11
Best params {'learning_rate': 0.1, 'min_samples_leaf': 10, 'min_samples_sp
lit': 2}
Best MSE: 154.6713901026176
Time: 2282.761451482773
```

```
Time: 0.0009737014770507812
```

4.1.2. Cross validation

```
 Validation PSNR: [26.04, 26.2, 26.16]
 Train PSNR: [26.47, 26.39, 26.41]
```

```
Time: 272.263188123703
```

From the cross validation, we can know that our baseline method are not overfit, since the test PSNR are close to train PSNR.

4.2. Make Prediction

4.2.1. Train models

```
Time: 130.25535941123962
```

4.2.2. Predict Test set

```
PSNR: 26.10305467926151
```

```
Time: 0.9045805931091309
```

5. Advanced algorithm (Xgboost)

5.1. Tuning Model

5.1.1 Grid Search

```
Time: 0.0002613067626953125
```

5.1.2. Cross Validation

```
 Validation PSNR: [26.07, 26.23, 26.2]
 Train PSNR: [26.92, 26.85, 26.88]
```

```
Time: 75.56308078765869
```

5.2. Make Prediction

5.2.1 Train models

```
Time: 277.3800723552704
```

5.2.2 Predict Test set

```
PSNR: 26.039720724299634
```

```
Time: 4.063607215881348
```

6. Advanced algorithm (Light GMB)

6.1. Tuning Model

6.1.1. Grid Search

6.1.2. Cross Validation

```
 Validation PSNR: [26.1, 26.24, 26.24]
 Train PSNR: [27.05, 26.99, 26.98]
```

```
Time: 72.69590163230896
```

6.2. Make Prediction

6.2.1 Train models

```
Time: 19.83057188987732
```

6.2.2 Predict Test set

```
PSNR: 26.172132575193324
```

```
Time: 3.1158692836761475
```

7. Advanced algorithm (Random Forest)

7.1. Tuning Model

7.1.1. Grid Search

```
Model 0
Best params {'min_samples_split': 5, 'min_samples_leaf': 12, 'n_estimator
s': 50}
Best MSE: 111.90318899241801
Model 1
Best params {'min_samples_split': 15, 'min_samples_leaf': 12, 'n_estimator
s': 50}
Best MSE: 109.75202331094341
Model 2
Best params {'min_samples_split': 10, 'min_samples_leaf': 12, 'n_estimator
s': 50}
Best MSE: 113.04683253950004
Model 3
Best params {'min_samples_split': 15, 'min_samples_leaf': 12, 'n_estimator
s': 50}
Best MSE: 110.0267683279919
Model 4
Best params {'min_samples_split': 10, 'min_samples_leaf': 12, 'n_estimator
s': 50}
Best MSE: 107.6238266829209
Model 5
Best params {'min_samples_split': 5, 'min_samples_leaf': 12, 'n_estimator
s': 50}
Best MSE: 104.81182284849491
Model 6
Best params {'min_samples_split': 15, 'min_samples_leaf': 12, 'n_estimator
s': 50}
Best MSE: 108.04929922380353
Model 7
Best params {'min_samples_split': 15, 'min_samples_leaf': 12, 'n_estimator
s': 50}
Best MSE: 105.21240289117571
Model 8
Best params {'min_samples_split': 10, 'min_samples_leaf': 12, 'n_estimator
s': 50}
Best MSE: 114.81466678306992
Model 9
Best params {'min_samples_split': 10, 'min_samples_leaf': 12, 'n_estimator
s': 50}
Best MSE: 113.6038601714536
Model 10
Best params {'min_samples_split': 5, 'min_samples_leaf': 12, 'n_estimator
s': 50}
Best MSE: 114.862307746691
Model 11
Best params {'min_samples_split': 10, 'min_samples_leaf': 12, 'n_estimator
s': 50}
Best MSE: 113.19167738166782
Time: 1415.9164199829102
```

```
Time: 0.00497126579284668
```

7.1.2 Cross Validation

```
Validation PSNR: [26.06, 26.22, 26.19]
Train PSNR: [27.72, 27.63, 27.65]
```

```
Time: 140.64839267730713
```

7.2. Make Prediction

7.2.1 Train models

```
Time: 34.490190505981445
```

7.2.2 Predict Test set

```
PSNR: 26.20172578819789
```

```
Time: 1.3583359718322754
```

8. Summary

Out[240]:

	PSNR on Test	Train Time	Predict Time
GBM (n_sample = 100)	26.14	73.92	0.52
GBM (n_sample = 1000)	25.95	739.2	5.52
INTER_NEAREST	25.78	nan	298.13
Light GBM (n_sample = 100)	26.23	6.14	0.5
Light GBM (n_sample = 1000)	26.07	30.61	4.73
Random Forest (n_sample = 100)	26.2	33.69	1.36
Xgboost (n_sample = 100)	26.19	19.13	0.28
Xgboost (n_sample = 1000)	26.03	277.5	4.06

As we can see from the above table, we can conclude as foloowing:

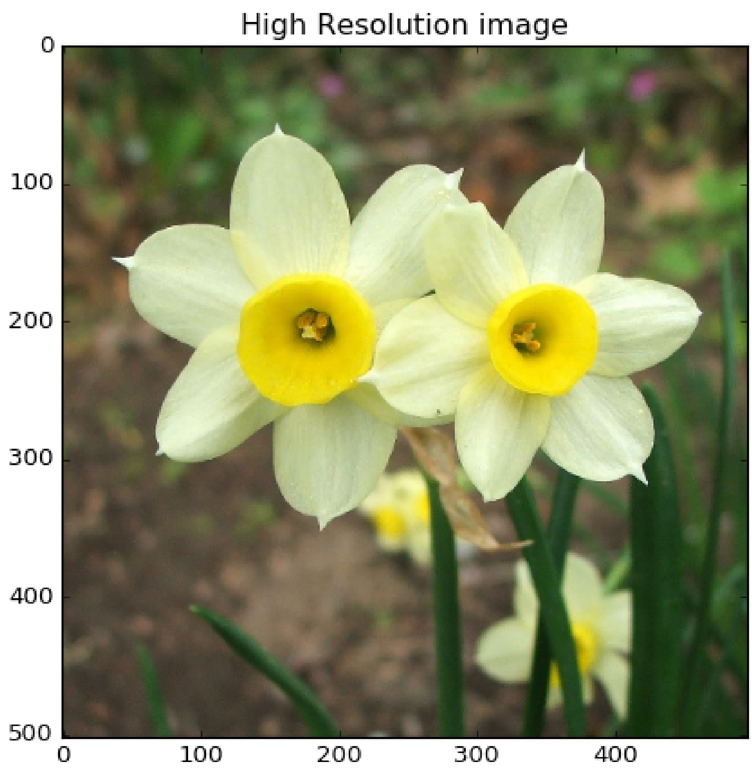
- All of our advanced models get better performance compared to the baseline model as well as OpenCv implemented function.
- Light GBM has highest score and highest speed of train.
- Xgboost has highest speed for prediction.
- When n_sample = 1000 rather than 100, our performance become worse since we are not tunning paramters for the reason that it consume to much time, however, from cross validation we can see it become more robust when our sample become larger.
- When sample size become larger, Xgboost become slower, however, light GBM still fast.

After balancing time consumption and PSNR score, we decide to choose model: Light GBM (n_sample = 100)

8.1 Super resolution 100 images with light gbm

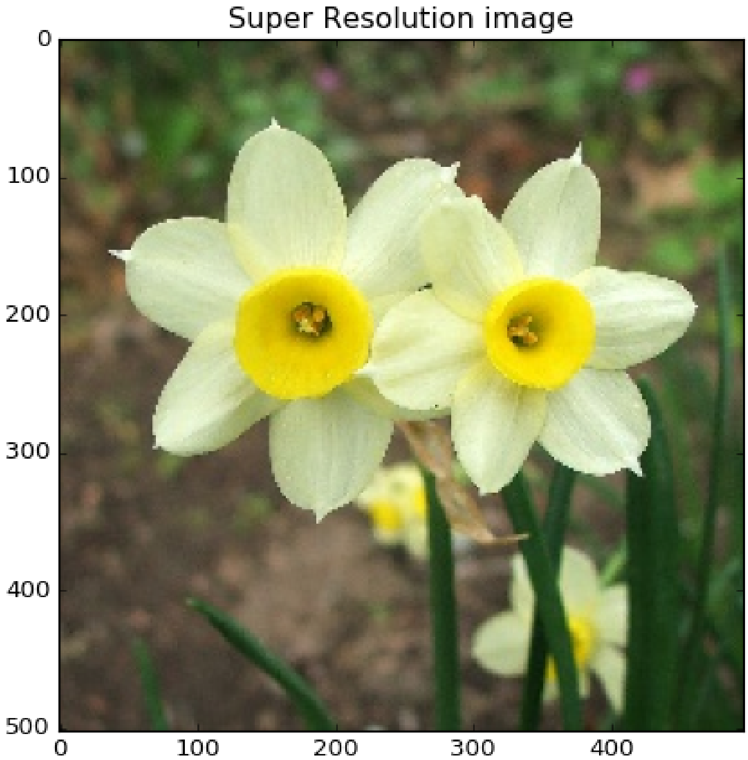
```
PSNR is 28.1
Time: 551.9331457614899
```

8.2 Compare our result image with original high resolution image



Out[282]:

```
<matplotlib.text.Text at 0x7ff7d40bf588>
```



Out[283]:

```
<matplotlib.text.Text at 0x7ff7d4095940>
```

```
PSNR:  31.34139313405133
```

9. Super Resolution for Test LR Images

```
Time: 64.94702672958374
```

```
Time: 95.643887758255
```