Reconstructing pictures with machine learning [demonstration]

Feb 9, 2016 • Alex Rogozhnikov

Reconstructing pictures with machine learning [demonstration] ¶

In this post I demonstrate how different techniques of machine learning are working.

The idea used is very simple:

- each black & white image can be treated as complex function of 2 variables - x and y, position of pixel
- intensity of pixel is output
- we can leave only small fraction of pixels, treating others as 'lost'
- by looking how different reression algorithms reconstruct picture, we can get some understanding of how those are operating

Don't treat this demonstration as some 'comparison of approaches', because this problem (reconstructing a picture) is very specific and has very few in common with typical ML datasets. And of course, this approach is not to be used in practice to reconstruc pictures :)

I am using scikit-learn and making use of its API, enabling user to construct new models via meta-ensembling and pipelines.

First, we import lots of things ¶

[1]:
```
# !pip install image
from PIL import Image
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
import numpy
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestRegressor, BaggingRegressor, Gra

from sklearn.cross_validation import train_test_split
from sklearn.random_projection import GaussianRandomProjection
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.kernel_approximation import RBFSampler
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor

from rep.metaml import FoldingRegressor
from rep.estimators import XGBoostRegressor, TheanetsRegressor
```

Download the picture¶

I took quite complex picture with many little details

[3]:
```
!wget http://orig05.deviantart.net/1d93/f/2009/084/5/2/new_york_black_anc
```

```
--2016-02-09 04:25:20--  http://orig05.deviantart.net/1d93/f/2009/084/5/2
Resolving orig05.deviantart.net... 216.137.63.99, 216.137.63.108, 216.137
Connecting to orig05.deviantart.net|216.137.63.99|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 657820 (642K) [image/jpeg]
Saving to: 'image.jpg'

image.jpg           100%[====================>] 642.40K  65.2KB/s   in 1

2016-02-09 04:25:32 (60.0 KB/s) - 'image.jpg' saved [657820/657820]
```

[4]:
```
image = numpy.asarray(Image.open('./image.jpg')).mean(axis=2)

plt.figure(figsize=[20, 10])
plt.imshow(image, cmap='gray')
```

t[4]:
```
<matplotlib.image.AxesImage at 0x10fc35690>
```

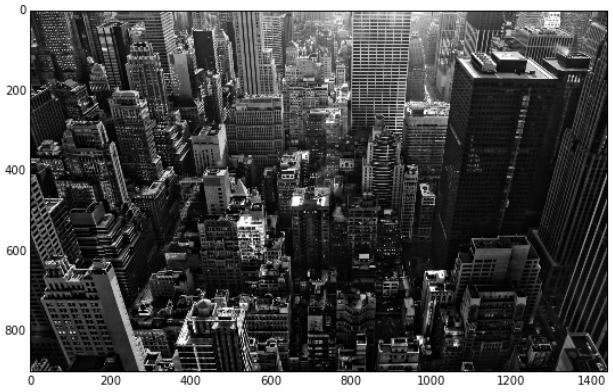
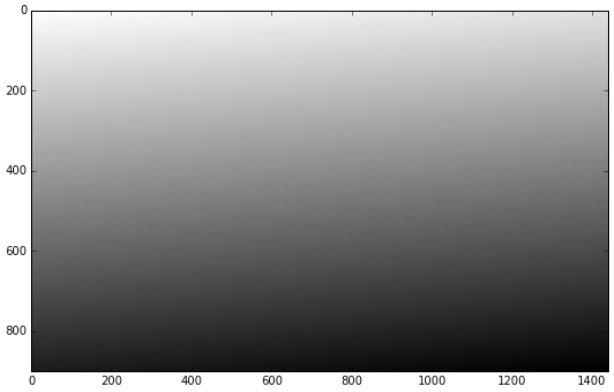
Define a function to train regressor¶

`train_size` is how many pixels shall be used in reconstructing the picture

[5]:
```
def train_display(regressor, image, train_size=0.02):
    height, width = image.shape
    flat_image = image.reshape(-1)
    xs = numpy.arange(len(flat_image)) % width
    ys = numpy.arange(len(flat_image)) // width
    data = numpy.array([xs, ys]).T
    target = flat_image
    trainX, testX, trainY, testY = train_test_split(data, target, train_s
    mean = trainY.mean()
    regressor.fit(trainX, trainY - mean)
    new_flat_picture = regressor.predict(data + mean)
    plt.figure(figsize=[20, 10])
    plt.subplot(121)
    plt.imshow(image, cmap='gray')
    plt.subplot(122)
    plt.imshow(new_flat_picture.reshape(height, width), cmap='gray')
```

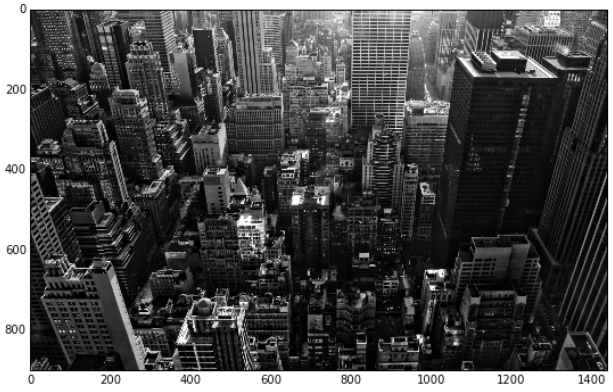
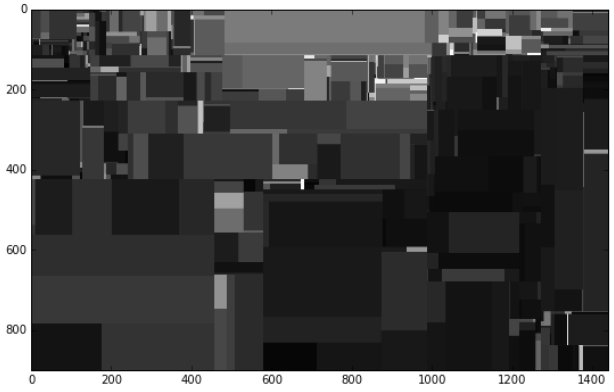
Linear regression¶

[6]:
```
train_display(LinearRegression(), image)
```

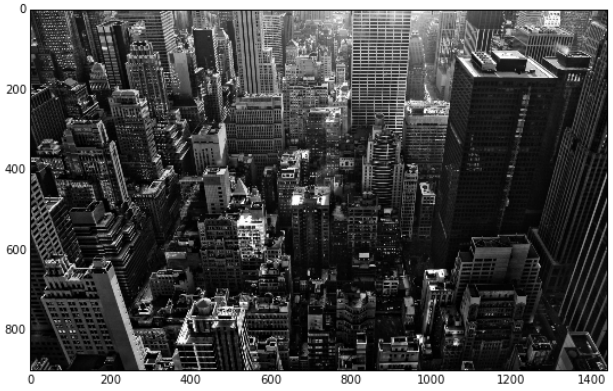
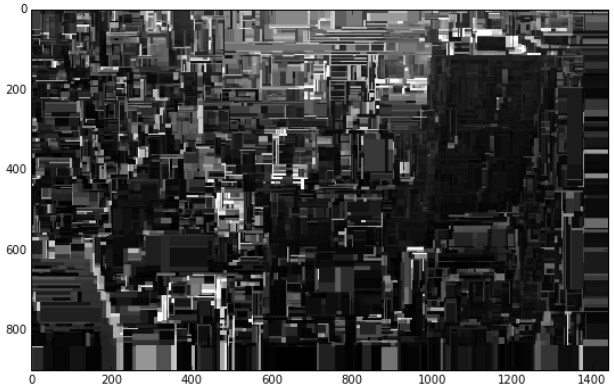



Deicsion tree limited by depth¶

[7]:
```
train_display(DecisionTreeRegressor(max_depth=10), image)
```

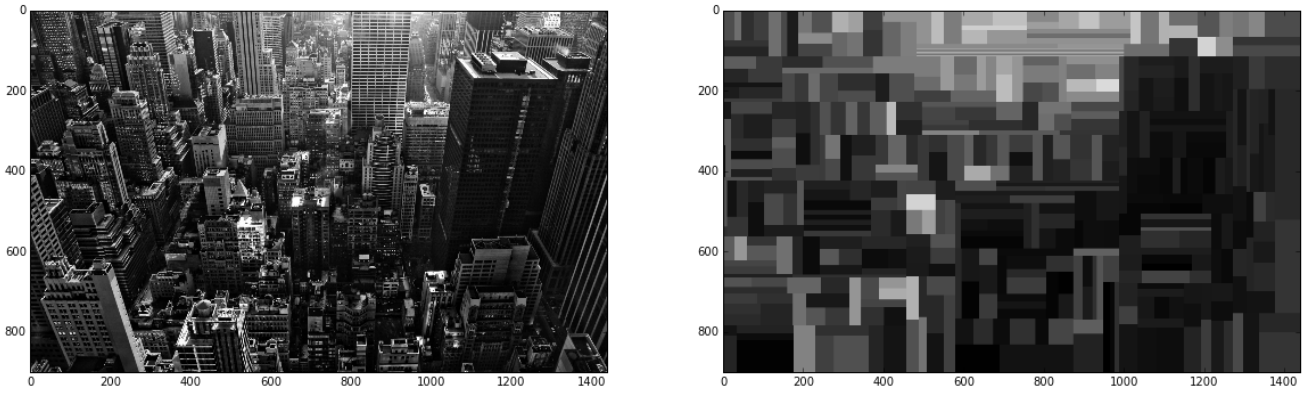



[8]:
```
train_display(DecisionTreeRegressor(max_depth=20), image)
```

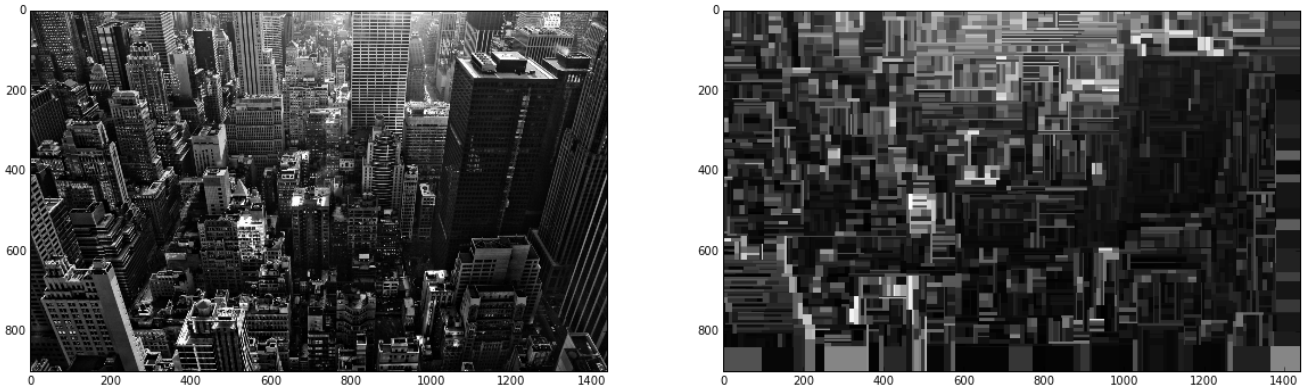



Decision tree limited by minimal number of samples in a leaf¶

[9]:
```
train_display(DecisionTreeRegressor(min_samples_leaf=40), image)
```

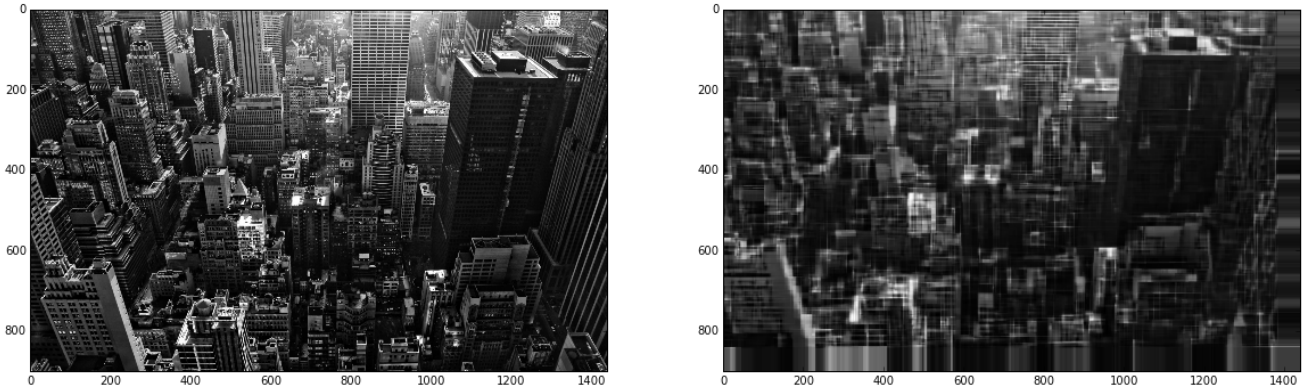


10]:
```
train_display(DecisionTreeRegressor(min_samples_leaf=5), image)
```



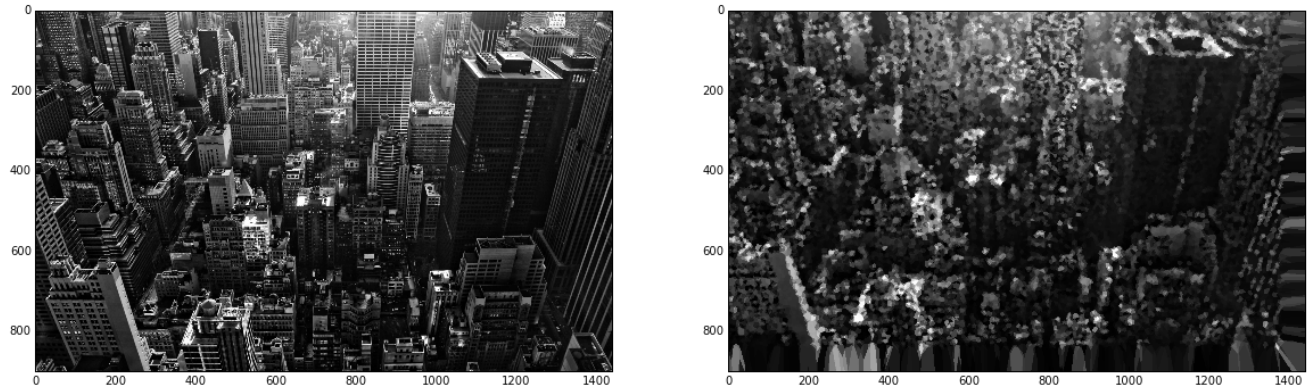
RandomForest¶

11]:
```
train_display(RandomForestRegressor(n_estimators=100), image)
```



K Nearest Neighbours¶

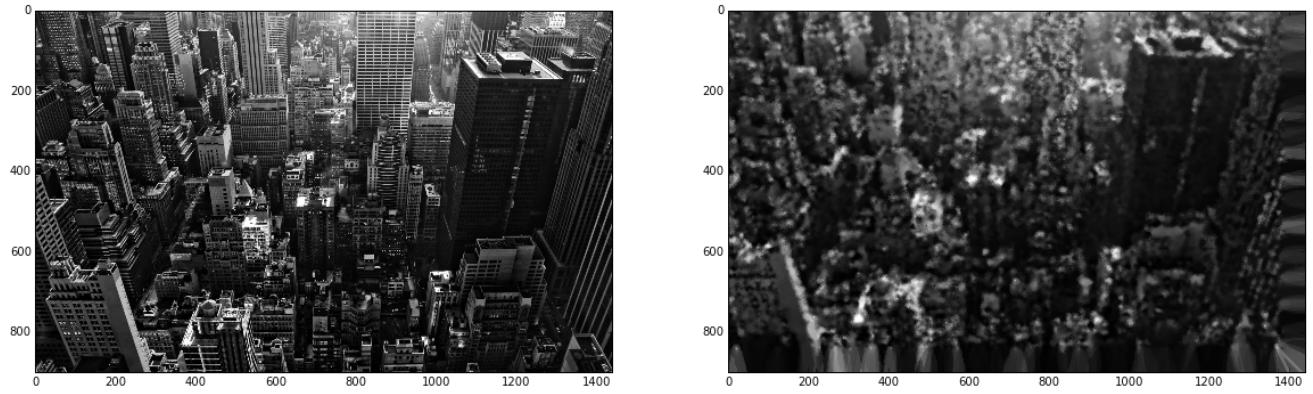
12]:
```
train_display(KNeighborsRegressor(n_neighbors=2), image)
```



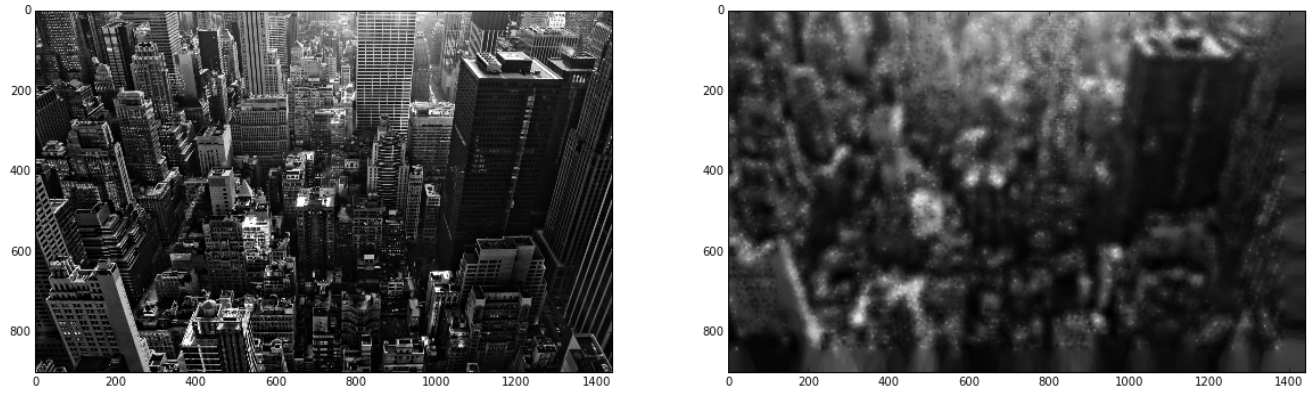
more neighbours + weighting according to distance¶

to make predictions smoother

13]:
```
train_display(KNeighborsRegressor(n_neighbors=5, weights='distance'), ima
```

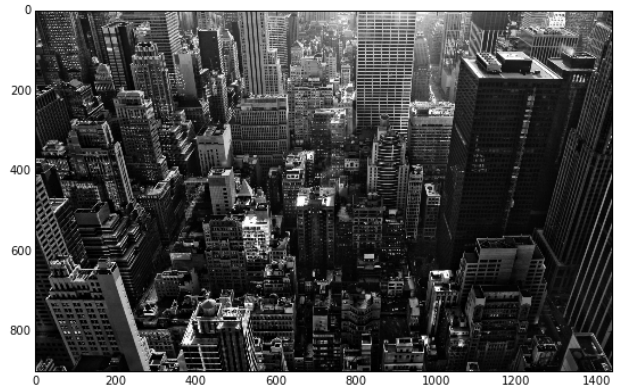
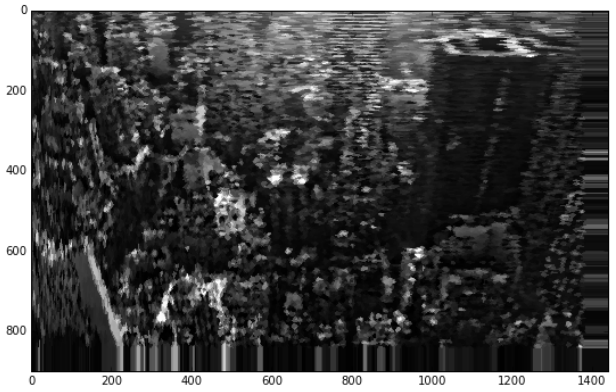


14]:
```
train_display(KNeighborsRegressor(n_neighbors=25, weights='distance'), im
```



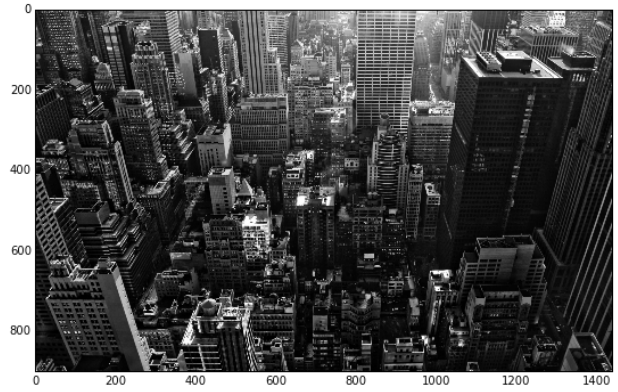
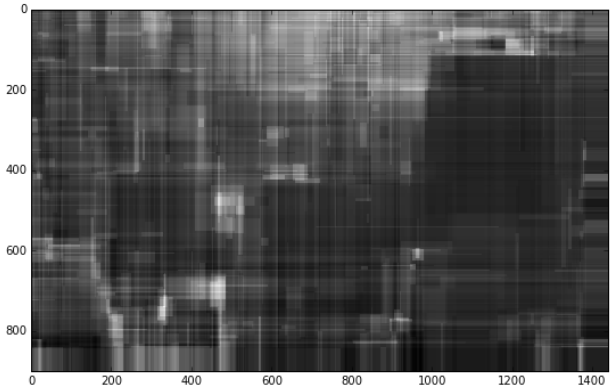
KNN with canberra metric¶

15]:
```
train_display(KNeighborsRegressor(n_neighbors=2, metric='canberra'), imag
```

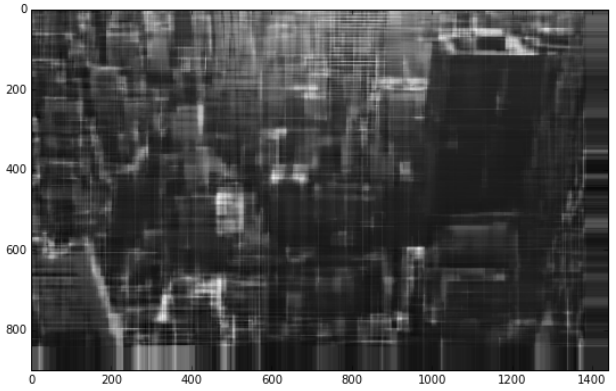
Gradient Boosting¶

16]:
```
train_display(XGBoostRegressor(max_depth=5, n_estimators=100, subsample=0
```

Gradient Boosting with deep trees¶

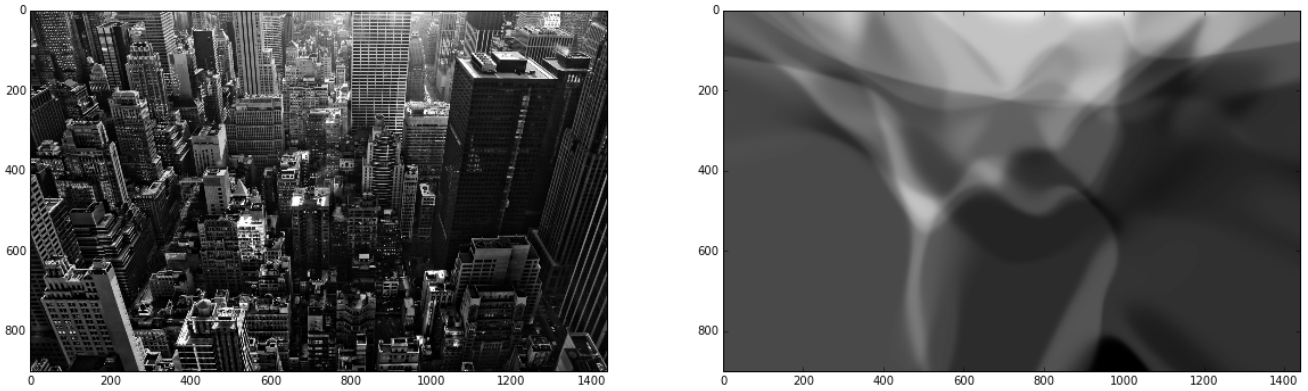
17]:
```
train_display(XGBoostRegressor(max_depth=12, n_estimators=100, subsample=
```

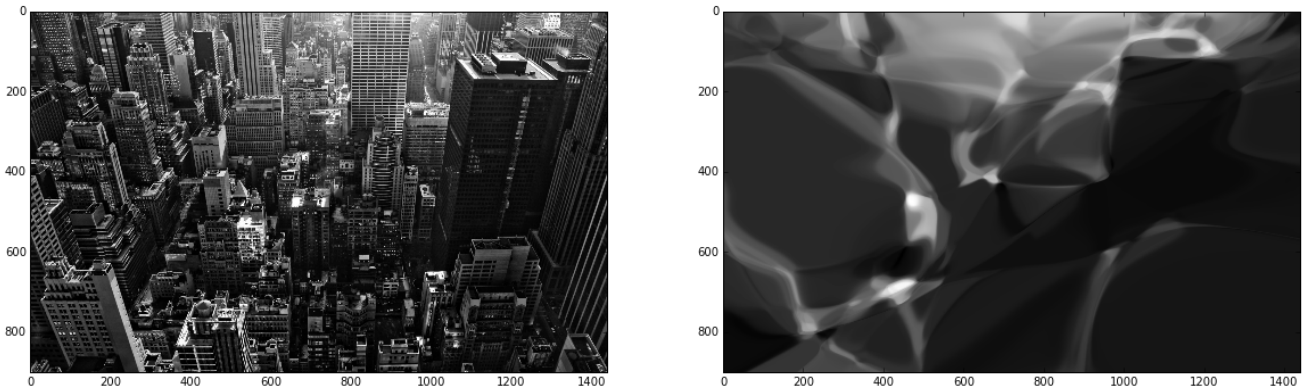
Neural networks¶

neural networks provide smooth predictions and are not able to deal with tiny sharp details of pictures.

18]:
```
train_display(TheanetsRegressor(layers=[20, 20], hidden_activation='tanh'
                                trainers=[{'algo': 'adadelta', 'learning_
```



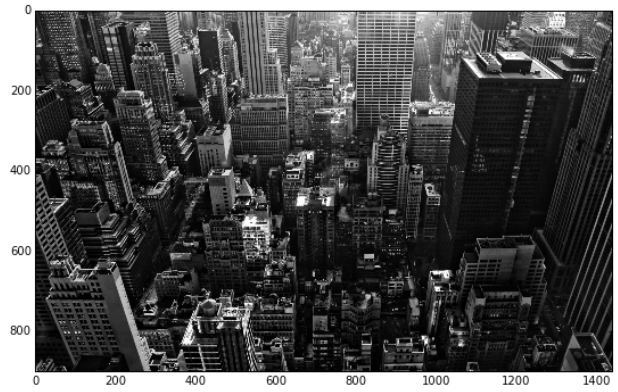
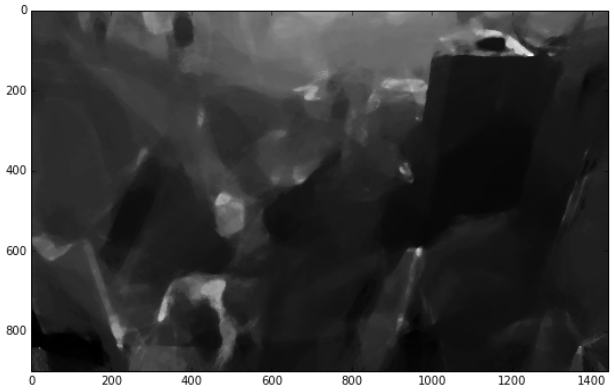
19]:
```
train_display(TheanetsRegressor(layers=[40, 40, 40, 40], hidden_activatic
                                trainers=[{'algo': 'adadelta', 'learning_
```



AdaBoost over Decision Trees using random projections¶

20]:
```
base = make_pipeline(GaussianRandomProjection(n_components=10),
                     DecisionTreeRegressor(max_depth=10, max_features=5))
train_display(AdaBoostRegressor(base, n_estimators=50, learning_rate=0.05
```

```
/Users/axelr/.conda/envs/rep/lib/python2.7/site-packages/sklearn/random_p
  DataDimensionalityWarning)
```

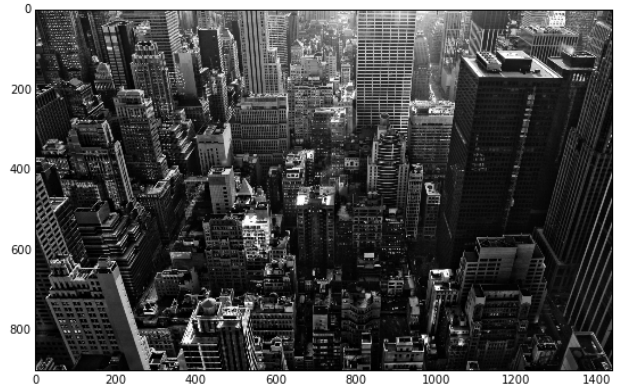
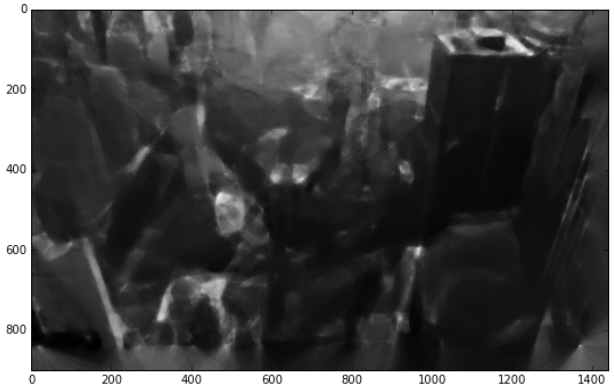



Bagging over decision trees using random projections¶

This is sometimes referred as Random Forest too (since this idea was proposed by Leo Breiman in the same paper).

21]:
```
base = make_pipeline(GaussianRandomProjection(n_components=15),
                     DecisionTreeRegressor(max_depth=12, max_features=5))
train_display(BaggingRegressor(base, n_estimators=100), image)
```

```
/Users/axelr/.conda/envs/rep/lib/python2.7/site-packages/sklearn/random_p
  DataDimensionalityWarning)
```

See also:¶

- Drawing an image with deep neural network Andrej Karpathy
- Artificial artist by Tim Head
- Painting photo in different artistic styles. Note that the study in this paper is quite different from things demonstrated in the post.

Feel free to download the notebook from repository and play with other images / parameters.

This post was written in IPython. You can download the notebook from repository.

Brilliantly wrong (subscribe via RSS)

Alex Rogozhnikov
alex.rogozhnikov@yandex.ru
arogozhnikov

Brilliantly Wrong - Alex Rogozhnikov's blog about math, machine learning, programming and high energy physics.