saravanabalagi / deep_image_prior Public

Code Issues Pull requests Actions Projects Wiki Security Insights

master

deep_image_prior / Denoising with zero pre-training.ipynb

saravanabalagi Adjusting learning rate

1 contributor

2.22 MB

In [1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import cv2

from keras.models import Sequential
from keras.models import Model
from keras.layers import Dense
from keras.layers import Convolution2D
from keras.layers import AveragePooling2D
from keras.layers import UpSampling2D
from keras.layers import Reshape
from keras.layers import Flatten
from keras.layers import Input
from keras.optimizers import Adam
```

```
Using TensorFlow backend.
```

From [0-255] to [-1 to 1]

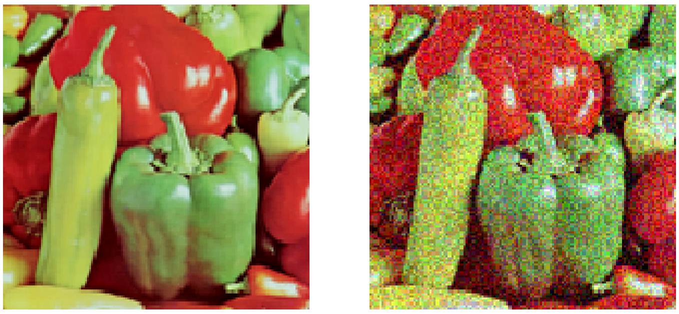
In [2]:

```
def normalize(image):             return (image/255 - 0.5)*2
def to_image(normalized_image):   return ((normalized_image/2 + 0.5) * 255).as
```

Generate noisy image

In [3]:

```
im = cv2.imread("peppers.png")[:,:,::-1]  #BGR to RGB
noise_intensity = 50
noise = np.random.randint(-noise_intensity, noise_intensity, size = im.shape)
im_noise = (im + noise).clip(0,255).astype(np.uint8)

plt.subplot(121); plt.axis('off'); plt.imshow(to_image(normalize(im)))
plt.subplot(122); plt.axis('off'); plt.imshow(im_noise)
plt.show()
```



Build the model

In [7]:

```
def deep_image_prior_model():
    encoding_size = 128

    encoder = Sequential([
        Convolution2D(32, 3, padding='same', input_shape=[128,128,3], activat
        Convolution2D(32, 3, padding='same', activation='relu'),
        AveragePooling2D(),
        Convolution2D(64, 3, padding='same', activation='relu'),
        Convolution2D(64, 3, padding='same', activation='relu'),
        AveragePooling2D(),
        Convolution2D(128, 3, padding='same', activation='relu')
        Convolution2D(128, 3, padding='same', activation='relu'),
        Convolution2D(128, 3, padding='same', activation='relu'),
        Flatten(),
        Dense(encoding_size, activation='tanh')
    ])

    decoder = Sequential([
        Dense(192, input_shape=(encoding_size,), activation='relu'),
        Reshape((8, 8, 3)),
        Convolution2D(128, 3, padding='same', activation='relu'),
        Convolution2D(128, 3, padding='same', activation='relu'),
        UpSampling2D(),
        Convolution2D(64, 3, padding='same', activation='relu'),
        Convolution2D(64, 3, padding='same', activation='relu'),
        UpSampling2D(),
        Convolution2D(32, 3, padding='same', activation='relu'),
        Convolution2D(32, 3, padding='same', activation='relu'),
        UpSampling2D(),
        Convolution2D(16, 3, padding='same', activation='relu'),
        Convolution2D(16, 3, padding='same', activation='relu'),
        UpSampling2D(),
        Convolution2D(8, 3, padding='same', activation='relu'),
        Convolution2D(3, 3, padding='same', activation='tanh')
    ])

    autoencoder = Sequential([
        encoder,
        decoder
    ])

    autoencoder.compile(loss='mse', optimizer=Adam(lr=0.0001))
    return autoencoder
```

In [8]:

```
x = np.random.random(size=((1,) + im.shape)) * 2 - 1
y = normalize(im_noise[None, :])
[x.shape, y.shape]
```

Out[8]:

```
[(1, 128, 128, 3), (1, 128, 128, 3)]
```

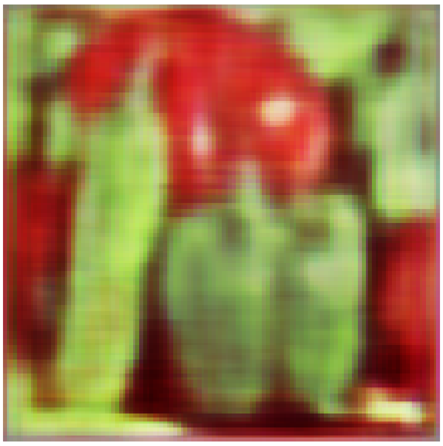
Fit noisy image and produce rectified image

In [9]:

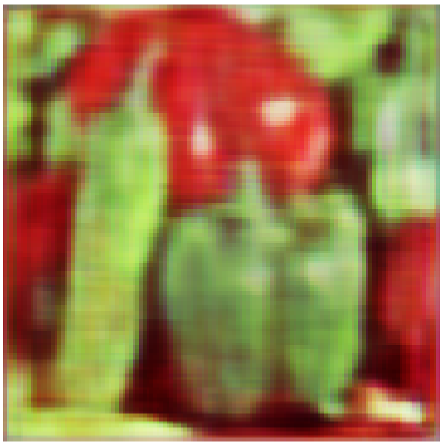
```
plt.axis('off')
plt.title('Iteration 0')
plt.imshow(to_image(x[0]))
plt.show()

model = deep_image_prior_model()
iterations = 30                     # in hundreds
results = np.empty(x.shape)

for i in range(iterations):
    model.fit(x, y, epochs=100, batch_size=1, verbose=0)
    output = model.predict(x)
    results = np.append(results, output, axis=0)
    plt.axis('off')
    plt.title('Iteration '+ str((i+1)*100))
    plt.imshow(to_image(output[0]))
    plt.show()
```

Iteration 100

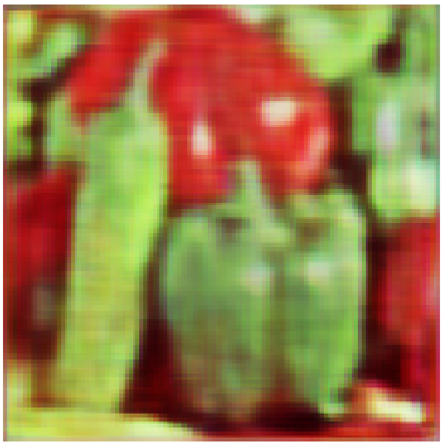
Iteration 200



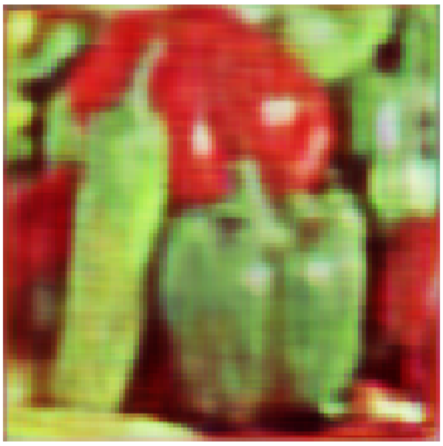
Iteration 300



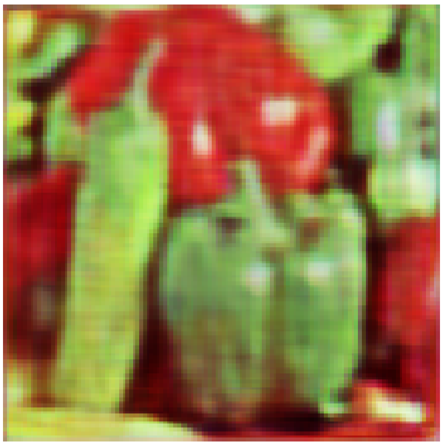
Iteration 400



Iteration 500



Iteration 600



Iteration 700

Iteration 800

Iteration 900

Iteration 1000

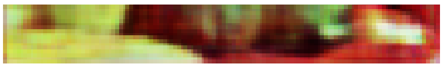
Iteration 1100

Iteration 1200

Iteration 1300

Iteration 1400

Iteration 1500



Iteration 1600

Iteration 1700

Iteration 1800

Iteration 1900

Iteration 2000

Iteration 2100

Iteration 2200

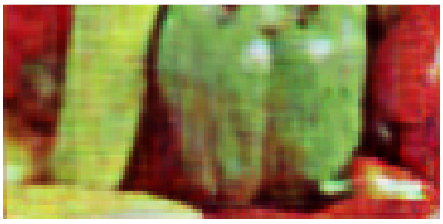
Iteration 2300

Iteration 2400

Iteration 2500

Iteration 2600

Iteration 2700



Iteration 2800

Iteration 2900

Iteration 3000