Transfer a photo into a Monet style picture

(This part we got inspiration and learning from https://www.kaggle.com/basu369victor/style-transfer-deep-learning-algorithm (https://www.kaggle.com/basu369victor/style-transfer-deep-learning-algorithm))

First, we have a `original image` and a `style image` . Then, we still use VGG16 but only pick the convolution layers for extracting image's features and delete the layers for classification. Finally, we will get a new picture combine the style of the `style image` and the objects from `original image` .

In [1]:

```
import os
import numpy as np
import tensorflow
from tensorflow.keras import applications
from tensorflow.keras.applications import vgg16
from tensorflow.keras.models import Model
from scipy.optimize import fmin_l_bfgs_b
from tensorflow.keras.preprocessing.image import load_img, save_img, img_to_
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
import tensorflow.keras.backend as K
K.clear_session()
ori_path = './photo.jpg'
style_path = './Claude_Monet.jpg'
```

Generation image's size

In [3]:

```
ori_image = load_img(ori_path)
nrow, ncol = ori_image.size
nrow = nrow // 10
ncol = ncol // 10
print(str(nrow), str(ncol))
```

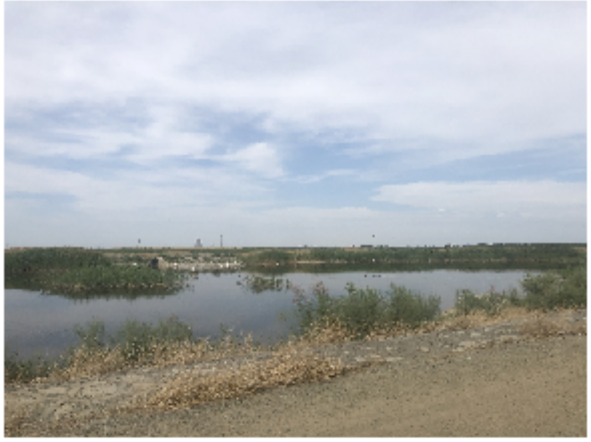
```
403 302
```

Show 2 images

In [4]:
```
ori_image = load_img(ori_path,target_size=(ncol, nrow))
print(ori_image.size)
style_image = load_img(style_path, target_size=(ncol, nrow))
plt.axis('off')
plt.title('Photo')
plt.imshow(ori_image)
plt.show()
plt.axis('off')
plt.title('Claude_Monet')
plt.imshow(style_image)
```

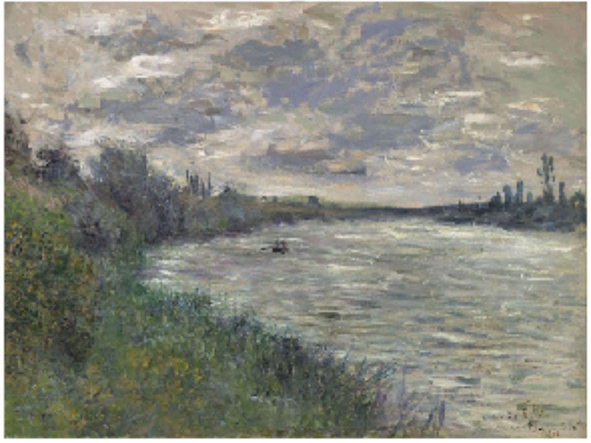
```
(403, 302)
```

Photo



Out[4]:
```
<matplotlib.image.AxesImage at 0x7f028571a358>
```

Claude_Monet



Pre-Process the images as the VGG16 input.Using tensor to represent the image,and bulid a new imgae as ouput

In [5]:

```
def preprocess(image):
    img = img_to_array(image)
    img = vgg16.preprocess_input(img[np.newaxis,:])
    img = K.variable(img)
    return img
ori_img = preprocess(ori_image)
style_img = preprocess(style_image)
result_image = K.placeholder((1,ncol, nrow,3)) # 'channels_last' format()
result_image.shape
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflo
w/python/ops/resource_variable_ops.py:435: colocate_with (from tensorflo
w.python.framework.ops) is deprecated and will be removed in a future ver
sion.
Instructions for updating:
Colocations handled automatically by placer.
```

Out[5]:

```
TensorShape([Dimension(1), Dimension(302), Dimension(403), Dimension(3)])
```

Then intergrating original image, style image and result image.

In [6]:

```
vgg16_input = K.concatenate([ori_img,style_img,result_image], axis=0)
vgg16_input
```

Out[6]:

```
<tf.Tensor 'concat:0' shape=(3, 302, 403, 3) dtype=float32>
```

Import VGG16 model, and make a dictionary

```
In [7]: model = vgg16.VGG16(input_tensor=vgg16_input,
                            weights='imagenet', include_top=False)
        model.summary()
        #dictionary
        model_dict = dict([(layer.name, layer.output) for layer in model.layers])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (3, 302, 403, 3)          0         
_________________________________________________________________
block1_conv1 (Conv2D)        (3, 302, 403, 64)         1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (3, 302, 403, 64)         36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (3, 151, 201, 64)         0         
_________________________________________________________________
block2_conv1 (Conv2D)        (3, 151, 201, 128)        73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (3, 151, 201, 128)        147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (3, 75, 100, 128)         0         
_________________________________________________________________
block3_conv1 (Conv2D)        (3, 75, 100, 256)         295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (3, 75, 100, 256)         590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (3, 75, 100, 256)         590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (3, 37, 50, 256)          0         
_________________________________________________________________
block4_conv1 (Conv2D)        (3, 37, 50, 512)          1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (3, 37, 50, 512)          2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (3, 37, 50, 512)          2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (3, 18, 25, 512)          0         
_________________________________________________________________
block5_conv1 (Conv2D)        (3, 18, 25, 512)          2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (3, 18, 25, 512)          2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (3, 18, 25, 512)          2359808   
_________________________________________________________________
block5_pool (MaxPooling2D)   (3, 9, 12, 512)           0         
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Check the dictionary

In [8]:

```
model_dict
```

Out[8]:

```
{'block1_conv1': <tf.Tensor 'block1_conv1/Relu:0' shape=(3, 302, 403, 64)
dtype=float32>,
 'block1_conv2': <tf.Tensor 'block1_conv2/Relu:0' shape=(3, 302, 403, 64)
dtype=float32>,
 'block1_pool': <tf.Tensor 'block1_pool/MaxPool:0' shape=(3, 151, 201, 6
4) dtype=float32>,
 'block2_conv1': <tf.Tensor 'block2_conv1/Relu:0' shape=(3, 151, 201, 12
8) dtype=float32>,
 'block2_conv2': <tf.Tensor 'block2_conv2/Relu:0' shape=(3, 151, 201, 12
8) dtype=float32>,
 'block2_pool': <tf.Tensor 'block2_pool/MaxPool:0' shape=(3, 75, 100, 12
8) dtype=float32>,
 'block3_conv1': <tf.Tensor 'block3_conv1/Relu:0' shape=(3, 75, 100, 256)
dtype=float32>,
 'block3_conv2': <tf.Tensor 'block3_conv2/Relu:0' shape=(3, 75, 100, 256)
dtype=float32>,
 'block3_conv3': <tf.Tensor 'block3_conv3/Relu:0' shape=(3, 75, 100, 256)
dtype=float32>,
 'block3_pool': <tf.Tensor 'block3_pool/MaxPool:0' shape=(3, 37, 50, 256)
dtype=float32>,
 'block4_conv1': <tf.Tensor 'block4_conv1/Relu:0' shape=(3, 37, 50, 512)
dtype=float32>,
 'block4_conv2': <tf.Tensor 'block4_conv2/Relu:0' shape=(3, 37, 50, 512)
dtype=float32>,
 'block4_conv3': <tf.Tensor 'block4_conv3/Relu:0' shape=(3, 37, 50, 512)
dtype=float32>,
 'block4_pool': <tf.Tensor 'block4_pool/MaxPool:0' shape=(3, 18, 25, 512)
dtype=float32>,
 'block5_conv1': <tf.Tensor 'block5_conv1/Relu:0' shape=(3, 18, 25, 512)
dtype=float32>,
 'block5_conv2': <tf.Tensor 'block5_conv2/Relu:0' shape=(3, 18, 25, 512)
dtype=float32>,
 'block5_conv3': <tf.Tensor 'block5_conv3/Relu:0' shape=(3, 18, 25, 512)
dtype=float32>,
 'block5_pool': <tf.Tensor 'block5_pool/MaxPool:0' shape=(3, 9, 12, 512)
dtype=float32>,
 'input_1': <tf.Tensor 'concat:0' shape=(3, 302, 403, 3) dtype=float32>}
```

Vgg16 is usually used to classificate, however, here we need to delete the classification purpose layyers.

In this case, we decided to use 6 convolution layers to get the image's feature and one layers as output style feature

In [9]:

```
# 6 convolution layers and one output style feature
feature_layers = ['block1_conv1','block1_conv2',
                'block2_conv1','block3_conv1',
               'block4_conv1','block5_conv1']
nfeature_layer = 6
output_layers = ['block5_conv2']
```

Using dictionary and layers we choosed late step to gerenate features

In [10]:

```
features = model_dict['block5_conv2']
ori_features = features[0, :, :, :]
new_features = features[2, :, :, :]
```

We need a optimizer to minimize the loss between style image and new image. After reading the "Optimizers Guide"(https://qiskit.org/documentation/aqua/optimizers.html) (https://qiskit.org/documentation/aqua/optimizers.html)), we decided to use the Limited-memory Broyden-Fletcher-Goldfarb-Shanno Bound, because this optimizer has a direct callable function by scipy (https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html (https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html))

In [11]:

```
from scipy.optimize import fmin_l_bfgs_b
# x, f, d= fmin_l_bfgs_b(func, x0)

# x : array_like(Estimated position of the minimum.)
# f : float(Value of func at the minimum.)
# d : dict(Information dictionary.)
# func : callable f(x,*args) Function to minimise.
# x0 : ndarray Initial guess.
```

Here we decided to use the error square as loss and gradients descent which we learned in lecture.

Define Output Loss Function, using error square (keras.backend.square(x) https://keras.io/backend/ (https://keras.io/backend/))

In [12]:

```
# we has total 7 layers, the last ouput layer get smallest weight
output_weight=0.03
output_loss = output_weight * K.sum(K.square(new_features - ori_features))
```

Define Style Loss Function, using Gram Matrices

A Gram matrix of vectors a1, ... ,an is a matrix G

s.t. G=⟨ai,aj⟩ for all i,j if vectors a1, ... ,an are columns of a matrix A, then G=ATA a Gram matrix is Positive Definite and Symmetric if vectors a1, ... ,an are the rows of A (A would be so-called "Data Matrix"), then G=AAT, and it's called left Gram matrix

Then still using error square to caclutate 2 matrices loss

In [13]:

```
def style_loss_function(style_features, new_features):
    #overturn style_features tensor
    style_features = tensorflow.reshape(style_features,
                                         [-1, style_features.shape[-1]])
    # G=AAT
    style_gram_mtr = tensorflow.matmul(style_features, style_features,
                                        transpose_a=True,transpose_b=False)
    new_features = tensorflow.reshape(new_features,
                                         [-1, style_features.shape[-1]])
    # G=AAT
    output_gram_mtr = tensorflow.matmul(new_features, new_features,
                                        transpose_a=True,transpose_b=False)
    # Error square to caclutate 2 matrices loss
    style_loss = K.sum(K.square(style_gram_mtr - output_gram_mtr))
    return style_loss
```

Define How to Caculate the initial gradients, uising keras.backend.gradients(loss, variables) [https://keras.io/backend/#gradients (https://keras.io/backend/#gradients)](https://keras.io/backend/#gradients)

In [14]:

```
for layer_name in feature_layers:
    features = model_dict[layer_name]
    style_features = features[1, :, :, :]
    new_features = features[2, :, :, :]
    style_loss = style_loss_function(style_features, new_features)
    # 1 output layer, 6 convolution layers(features)
    output_loss = output_loss + (1 / nfeature_layer) * style_loss
gradient = K.gradients(output_loss, result_image)
loss_list = [output_loss]
loss_list += gradient #Record gradient
keras_function = K.function([result_image], loss_list)
```

With loss function and initial gradients, now we define the callable function to minimise using in optimizer.

In [15]:

```
class func(object):
    def __init__(self):
        self.loss_value = None

    def loss(self, x):
        x = x.reshape((1, ncol, nrow, 3))
        result = keras_function([x])
        self.loss_value = result[0]
        self.grad_values = np.array(result[1:]).flatten().astype('float64')
        return self.loss_value

    def grad(self, x):
        grad_values = np.copy(self.grad_values)
        self.loss_value = None
        self.grad_values = None
        return grad_values
function = func()
```

Using original image as the input

In [16]:
```
# run scipy-based optimization (L-BFGS) over the pixels of the generated ima
# so as to minimize the neural style loss
X = load_img(ori_path, target_size=(ncol, nrow))
X = np.expand_dims(img_to_array(X), axis=0)
X = vgg16.preprocess_input(X)
```

Start iterating using Limited-memory Broyden-Fletcher-Goldfarb-Shanno Bound

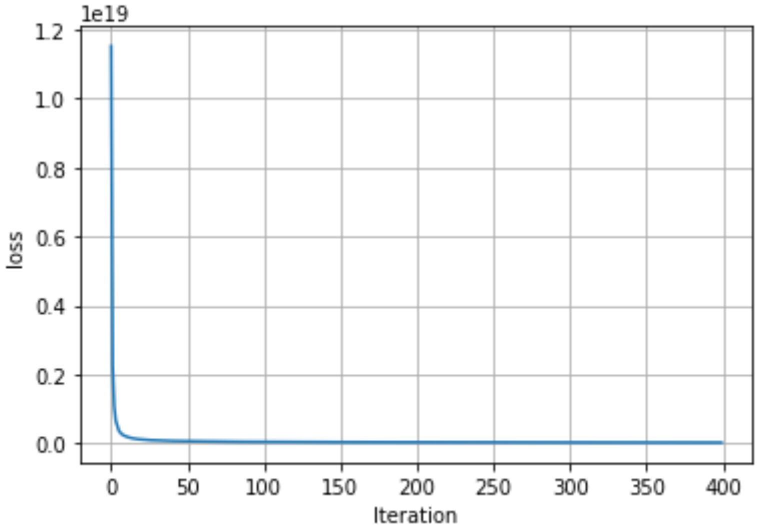
In [17]:
```
nitr = 400
neval = 20 #Maximum number of function evaluations.
loss = []
for i in range(nitr):
    print('Iteration = {}'.format(i))
    if i == 0:
        opt_loss = float('Inf') #initial min loss
    X, func_min, d= fmin_l_bfgs_b(function.loss, X.flatten(), fprime=functic
    loss.append(func_min)
    print(d['grad'])
    if func_min < opt_loss:
        opt_loss = func_min
        opt_X = X.copy()
```

```
Iteration = 0
[ 1.12871153e+13  9.67454595e+12  3.44800965e+12 ...  1.10163237e+12
 -4.79169118e+11 -1.46461442e+11]
Iteration = 1
[-1.38585270e+12 -7.72080337e+11  1.25274489e+12 ...  2.99249140e+11
  2.53802136e+11  2.26965545e+10]
Iteration = 2
[ 1.42048231e+12  3.60088601e+11 -3.27851016e+11 ...  7.72686807e+10
  2.60116019e+09  9.72613386e+10]
Iteration = 3
[2.75204997e+11 6.76066263e+10 2.63078314e+11 ... 6.59419955e+10
 1.10414971e+11 8.19095142e+10]
Iteration = 4
[ 2.99091149e+10 -6.28181565e+10 -3.54626109e+10 ...  1.47346883e+10
  8.29418865e+10  7.99861637e+10]
Iteration = 5
[-7.14945331e+10  3.54930065e+11  4.05361787e+10 ...  1.99629906e+10
  2.46782198e+10  5.56122235e+10]
Iteration = 6
```

Plot the minimum loss each iteration.

```
In [28]: plt.plot(loss)
         plt.grid()
         plt.xlabel('Iteration')
         plt.ylabel('loss')
```

```
Out[28]: Text(0, 0.5, 'loss')
```



Then convert output array to image.

VGG-16 was trained using Caffe, and Caffe uses OpenCV to load images which uses BGR by default, so both VGG models are expecting BGR images.

The official mean that gets used in the vgg16.py file is [123.68, 116.779, 103.939] (https://forums.fast.ai/t/how-is-vgg16-mean-calculated/4577/9 (https://forums.fast.ai/t/how-is-vgg16-mean-calculated/4577/9))

So we need to covert BGR to RGB before imshow

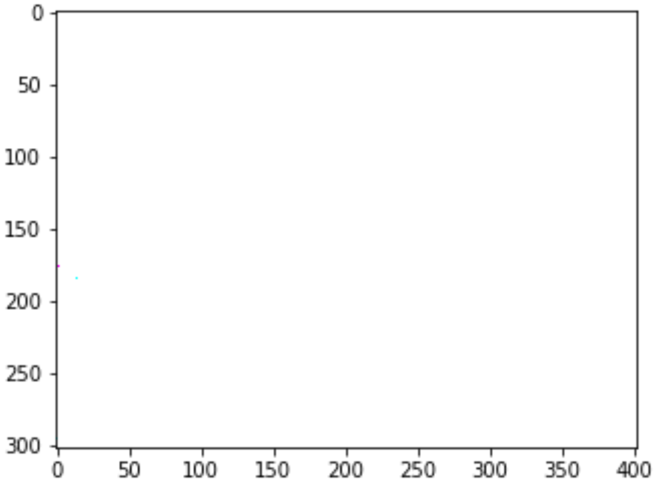
In [19]:

```
opt_X = opt_X.reshape((ncol, nrow,  3))
vgg_mean = [103.939,116.779,123.68]
for i in range(3):
    opt_X[:, :, i] = opt_X[:, :, i] + vgg_mean[i]
plt.imshow(opt_X)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] f
or floats or [0..255] for integers).
```

Out[19]:

```
<matplotlib.image.AxesImage at 0x7f022aa3c0b8>
```



Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

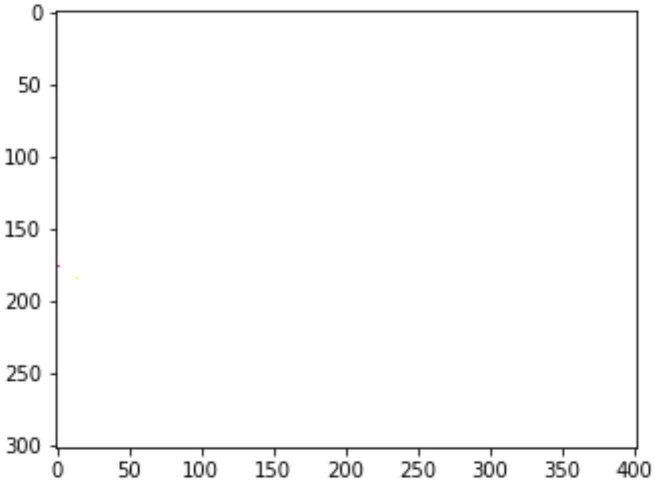
In [20]:

```
opt_X = opt_X[:, :, ::-1]
opt_X = np.clip(opt_X, 0, 255)
plt.imshow(opt_X)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] f
or floats or [0..255] for integers).
```

Out[20]:

```
<matplotlib.image.AxesImage at 0x7f022aafc518>
```

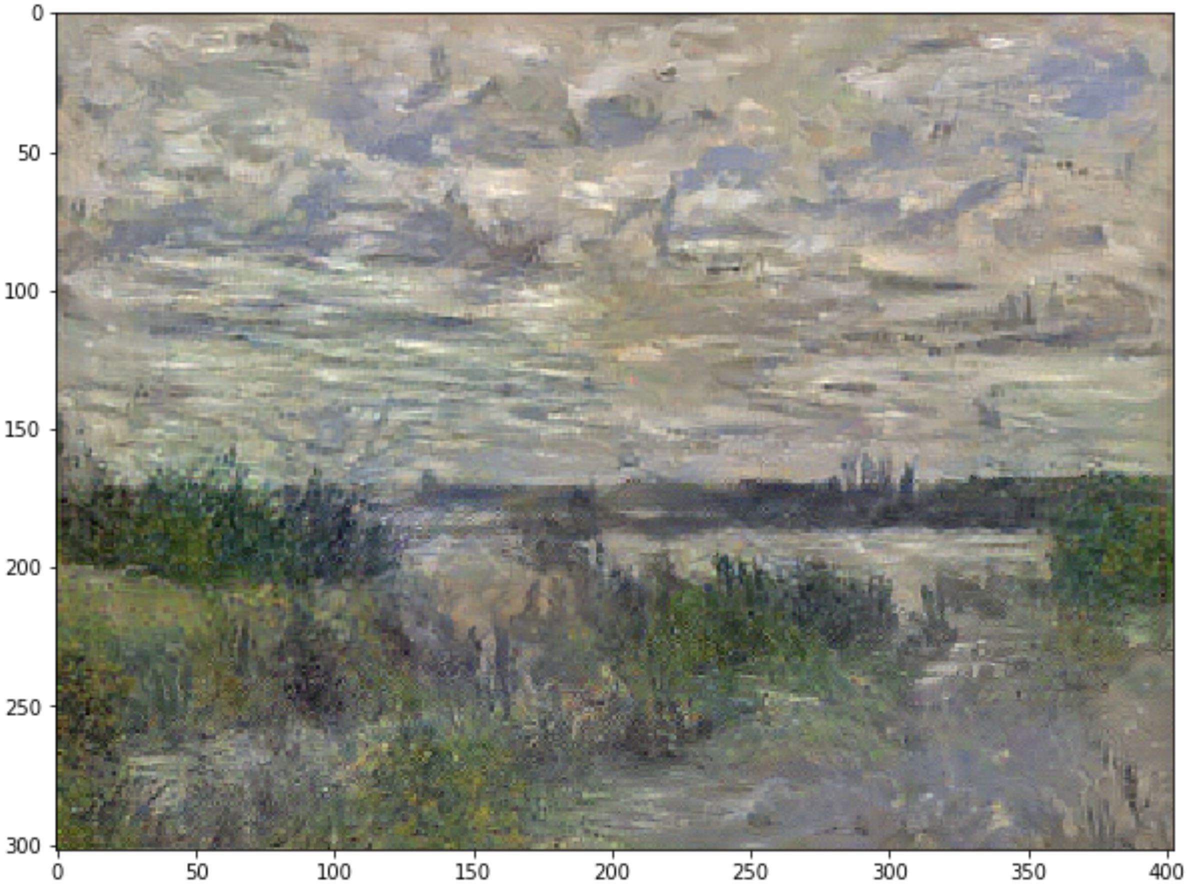


Convert the numpy arrays to uint8 before passing them to Image.fromarray np array should have data type as uint8

([https://stackoverflow.com/questions/10443295/combine-3-separate-numpy-arrays-to-an-rgb-image-in-python (https://stackoverflow.com/questions/10443295/combine-3-separate-numpy-arrays-to-an-rgb-image-in-python)](https://stackoverflow.com/questions/10443295/combine-3-separate-numpy-arrays-to-an-rgb-image-in-python))

```
In [24]: opt_X = opt_X.astype('uint8')
         plt.figure(figsize=(10,10))
         plt.imshow(opt_X)
```

```
Out[24]: <matplotlib.image.AxesImage at 0x7f022ab5b5c0>
```

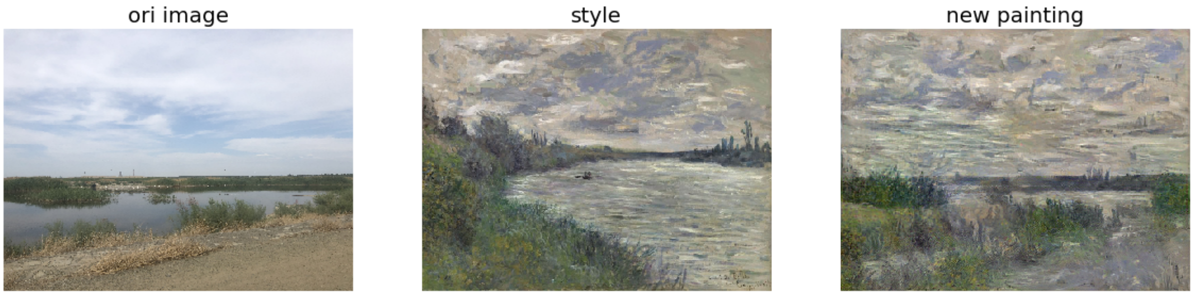


In [22]:

```
# Ori
plt.figure(figsize=(20,20))
plt.subplot(1,3,1)
plt.title("ori image",fontsize=20)
plt.axis('off')
plt.imshow(ori_image)
# style
plt.subplot(1,3,2)
plt.title("style",fontsize=20)
img_style = load_img(style_path)
plt.axis('off')
plt.imshow(style_image)
# new painting
plt.subplot(1,3,3)
plt.title("new painting",fontsize=20)
plt.axis('off')
plt.imshow(opt_X)
```

Out[22]:

```
<matplotlib.image.AxesImage at 0x7f022ac37b70>
```



In [ ]:

image-style-change (another example)

(This part we got inspiration and learning from https://www.kaggle.com/basu369victor/style-transfer-deep-learning-algorithm (https://www.kaggle.com/basu369victor/style-transfer-deep-learning-algorithm))

First, we have a `original image` and a `style image` . Then, we still use VGG16 but only pick the convolution layers for extracting image's features and delete the layers for classification. Finally, we will get a new picture combine the style of the `style image` and the objects from `original image` .

(This part we got inspiration and learning from https://www.kaggle.com/basu369victor/style-transfer-deep-learning-algorithm (https://www.kaggle.com/basu369victor/style-transfer-deep-learning-algorithm))

In [1]:

```
import os
import numpy as np
import tensorflow
from tensorflow.keras import applications
from tensorflow.keras.applications import vgg16
from tensorflow.keras.models import Model
from scipy.optimize import fmin_l_bfgs_b
from tensorflow.keras.preprocessing.image import load_img, save_img, img_to_
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

```
/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarnin
g: Conversion of the second argument of issubdtype from `float` to `np.fl
oating` is deprecated. In future, it will be treated as `np.float64 == n
p.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

In [2]:

```
import tensorflow.keras.backend as K
K.clear_session()
ori_path = './Edouard_Manet_49.jpg'
style_path = './Joan_Miro_38.jpg'
```

Generation image's size

In [3]:

```
ori_image = load_img(ori_path)
nrow, ncol = ori_image.size
nrow = nrow // 2
ncol = ncol // 2
print(str(nrow), str(ncol))
```

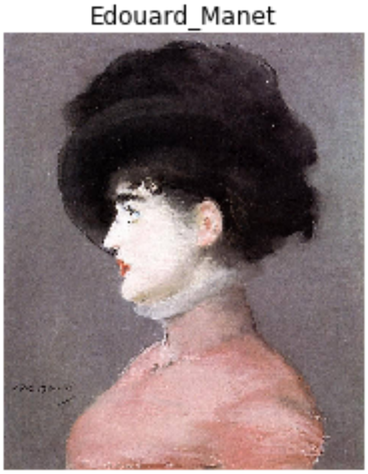
```
486 590
```

Show 2 images

In [4]:

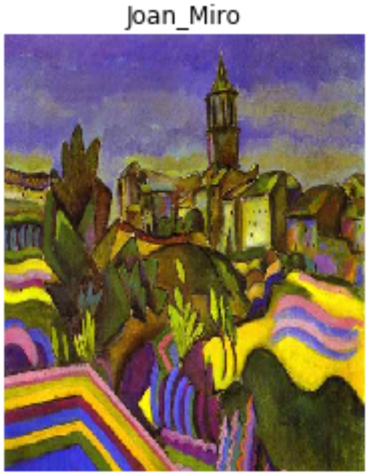
```
ori_image = load_img(ori_path,target_size=(ncol, nrow))
print(ori_image.size)
style_image = load_img(style_path, target_size=(ncol, nrow))
plt.axis('off')
plt.title('Edouard_Manet')
plt.imshow(ori_image)
plt.show()
plt.axis('off')
plt.title('Joan_Miro')
plt.imshow(style_image)
```

```
(486, 590)
```



Out[4]:

```
<matplotlib.image.AxesImage at 0x132812f550>
```



Pre-Process the images as the VGG16 input.Using tensor to represent the image,and bulid a new imgae as ouput

In [5]:
```
def preprocess(image):
    img = img_to_array(image)
    img = vgg16.preprocess_input(img[np.newaxis,:])
    img = K.variable(img)
    return img
ori_img = preprocess(ori_image)
style_img = preprocess(style_image)
result_image = K.placeholder((1,ncol, nrow,3)) # 'channels_last' format()
result_image.shape
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflo
w/python/ops/resource_variable_ops.py:435: colocate_with (from tensorflo
w.python.framework.ops) is deprecated and will be removed in a future ver
sion.
Instructions for updating:
Colocations handled automatically by placer.
```

Out[5]:
```
TensorShape([Dimension(1), Dimension(590), Dimension(486), Dimension(3)])
```

Then intergrating original image, style image and result image.

In [6]:
```
vgg16_input = K.concatenate([ori_img,style_img,result_image], axis=0)
vgg16_input
```

Out[6]:
```
<tf.Tensor 'concat:0' shape=(3, 590, 486, 3) dtype=float32>
```

Import VGG16 model, and make a dictionary

In [7]:

```python
model = vgg16.VGG16(input_tensor=vgg16_input,
                    weights='imagenet', include_top=False)
model.summary()
#dictionary
model_dict = dict([(layer.name, layer.output) for layer in model.layers])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (3, 590, 486, 3)          0         
_________________________________________________________________
block1_conv1 (Conv2D)        (3, 590, 486, 64)         1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (3, 590, 486, 64)         36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (3, 295, 243, 64)         0         
_________________________________________________________________
block2_conv1 (Conv2D)        (3, 295, 243, 128)        73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (3, 295, 243, 128)        147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (3, 147, 121, 128)        0         
_________________________________________________________________
block3_conv1 (Conv2D)        (3, 147, 121, 256)        295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (3, 147, 121, 256)        590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (3, 147, 121, 256)        590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (3, 73, 60, 256)          0         
_________________________________________________________________
block4_conv1 (Conv2D)        (3, 73, 60, 512)          1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (3, 73, 60, 512)          2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (3, 73, 60, 512)          2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (3, 36, 30, 512)          0         
_________________________________________________________________
block5_conv1 (Conv2D)        (3, 36, 30, 512)          2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (3, 36, 30, 512)          2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (3, 36, 30, 512)          2359808   
_________________________________________________________________
block5_pool (MaxPooling2D)   (3, 18, 15, 512)          0         
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Check the dictionary

```
In [8]: model_dict
```

```
Out[8]: {'block1_conv1': <tf.Tensor 'block1_conv1/Relu:0' shape=(3, 590, 486, 64)
        dtype=float32>,
         'block1_conv2': <tf.Tensor 'block1_conv2/Relu:0' shape=(3, 590, 486, 64)
        dtype=float32>,
         'block1_pool': <tf.Tensor 'block1_pool/MaxPool:0' shape=(3, 295, 243, 6
        4) dtype=float32>,
         'block2_conv1': <tf.Tensor 'block2_conv1/Relu:0' shape=(3, 295, 243, 12
        8) dtype=float32>,
         'block2_conv2': <tf.Tensor 'block2_conv2/Relu:0' shape=(3, 295, 243, 12
        8) dtype=float32>,
         'block2_pool': <tf.Tensor 'block2_pool/MaxPool:0' shape=(3, 147, 121, 12
        8) dtype=float32>,
         'block3_conv1': <tf.Tensor 'block3_conv1/Relu:0' shape=(3, 147, 121, 25
        6) dtype=float32>,
         'block3_conv2': <tf.Tensor 'block3_conv2/Relu:0' shape=(3, 147, 121, 25
        6) dtype=float32>,
         'block3_conv3': <tf.Tensor 'block3_conv3/Relu:0' shape=(3, 147, 121, 25
        6) dtype=float32>,
         'block3_pool': <tf.Tensor 'block3_pool/MaxPool:0' shape=(3, 73, 60, 256)
        dtype=float32>,
         'block4_conv1': <tf.Tensor 'block4_conv1/Relu:0' shape=(3, 73, 60, 512)
        dtype=float32>,
         'block4_conv2': <tf.Tensor 'block4_conv2/Relu:0' shape=(3, 73, 60, 512)
        dtype=float32>,
         'block4_conv3': <tf.Tensor 'block4_conv3/Relu:0' shape=(3, 73, 60, 512)
        dtype=float32>,
         'block4_pool': <tf.Tensor 'block4_pool/MaxPool:0' shape=(3, 36, 30, 512)
        dtype=float32>,
         'block5_conv1': <tf.Tensor 'block5_conv1/Relu:0' shape=(3, 36, 30, 512)
        dtype=float32>,
         'block5_conv2': <tf.Tensor 'block5_conv2/Relu:0' shape=(3, 36, 30, 512)
        dtype=float32>,
         'block5_conv3': <tf.Tensor 'block5_conv3/Relu:0' shape=(3, 36, 30, 512)
        dtype=float32>,
         'block5_pool': <tf.Tensor 'block5_pool/MaxPool:0' shape=(3, 18, 15, 512)
        dtype=float32>,
         'input_1': <tf.Tensor 'concat:0' shape=(3, 590, 486, 3) dtype=float32>}
```

Vgg16 is usually used to classificate, however, here we need to delete the classification purpose layyers.

In this case, we decided to use 6 convolution layers to get the image's feature and one layers as output style feature

```
In [9]: # 6 convolution layers and one output style feature
        feature_layers = ['block1_conv1','block1_conv2',
                          'block2_conv1','block3_conv1',
                         'block4_conv1','block5_conv1']
        nfeature_layer = 6
        output_layers = ['block5_conv2']
```

Using dictionary and layers we choosed late step to gerenate features

In [10]:

```
features = model_dict['block5_conv2']
ori_features = features[0, :, :, :]
new_features = features[2, :, :, :]
```

We need a optimizer to minimize the loss between stype image and new image. After reading the "Optimizers Guide"(https://qiskit.org/documentation/aqua/optimizers.html) (https://qiskit.org/documentation/aqua/optimizers.html)), we decided to use the Limited-memory Broyden-Fletcher-Goldfarb-Shanno Bound, because this optimizer has a direct callable function by scipy (https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html (https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html))

In [11]:

```
from scipy.optimize import fmin_l_bfgs_b
# x, f, d= fmin_l_bfgs_b(func, x0)

# x : array_like(Estimated position of the minimum.)
# f : float(Value of func at the minimum.)
# d : dict(Information dictionary.)
# func : callable f(x,*args) Function to minimise.
# x0 : ndarray Initial guess.
```

Here we decided to use the error square as loss and gradients descent which we learned in lecture.

Define Output Loss Function, using error square (keras.backend.square(x) https://keras.io/backend/ (https://keras.io/backend/))

In [12]:

```
# we has total 7 layers, the last ouput layer get smallest weight
output_weight=0.03
output_loss = output_weight * K.sum(K.square(new_features - ori_features))
```

Define Style Loss Function, using Gram Matrices

A Gram matrix of vectors a1, ... ,an is a matrix G

s.t. G=⟨ai,aj⟩ for all i,j if vectors a1, ... ,an are columns of a matrix A, then G=ATA a Gram matrix is Positive Definite and Symmetric if vectors a1, ... ,an are the rows of A (A would be so-called "Data Matrix"), then G=AAT, and it's called left Gram matrix

Then still using error square to caclutate 2 matrices loss

In [13]:
```
def style_loss_function(style_features, new_features):
    #overturn style_features tensor
    style_features = tensorflow.reshape(style_features,
                                        [-1, style_features.shape[-1]])
    # G=AAT
    style_gram_mtr = tensorflow.matmul(style_features, style_features,
                                       transpose_a=True,transpose_b=False)
    new_features = tensorflow.reshape(new_features,
                                        [-1, style_features.shape[-1]])
    # G=AAT
    output_gram_mtr = tensorflow.matmul(new_features, new_features,
                                       transpose_a=True,transpose_b=False)
    # Error square to caclutate 2 matrices loss
    style_loss = K.sum(K.square(style_gram_mtr - output_gram_mtr))
    return style_loss
```

Define How to Caculate the initial gradients, uising keras.backend.gradients(loss, variables) https://keras.io/backend/#gradients (https://keras.io/backend/#gradients)

In [14]:
```
for layer_name in feature_layers:
    features = model_dict[layer_name]
    style_features = features[1, :, :, :]
    new_features = features[2, :, :, :]
    style_loss = style_loss_function(style_features, new_features)
    # 1 output layer, 6 convolution layers(features)
    output_loss = output_loss + (1 / nfeature_layer) * style_loss
gradient = K.gradients(output_loss, result_image)
loss_list = [output_loss]
loss_list += gradient #Record gradient
keras_function = K.function([result_image], loss_list)
```

With loss function and initial gradients, now we define the callable function to minimise using in optimizer.

In [15]:
```
class func(object):
    def __init__(self):
        self.loss_value = None

    def loss(self, x):
        x = x.reshape((1, ncol, nrow, 3))
        result = keras_function([x])
        self.loss_value = result[0]
        self.grad_values = np.array(result[1:]).flatten().astype('float64')
        return self.loss_value

    def grad(self, x):
        grad_values = np.copy(self.grad_values)
        self.loss_value = None
        self.grad_values = None
        return grad_values
function = func()
```

Using original image as the input

In [16]:
```
# run scipy-based optimization (L-BFGS) over the pixels of the generated ima
# so as to minimize the neural style loss
X = load_img(ori_path, target_size=(ncol, nrow))
X = np.expand_dims(img_to_array(X), axis=0)
X = vgg16.preprocess_input(X)
```

Start iterating using Limited-memory Broyden-Fletcher-Goldfarb-Shanno Bound

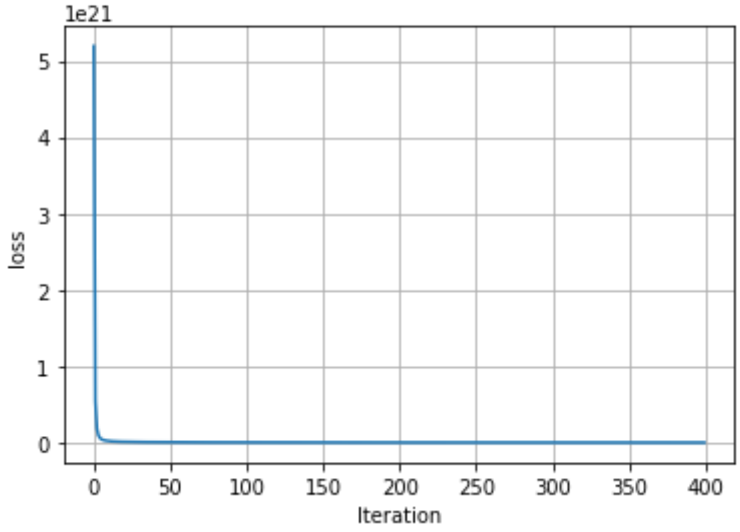
In [17]:
```
nitr = 400
neval = 20 #Maximum number of function evaluations.
loss = []
for i in range(nitr):
    print('Iteration = {}'.format(i))
    if i == 0:
        opt_loss = float('Inf') #initial min loss
    X, func_min, d= fmin_l_bfgs_b(function.loss, X.flatten(), fprime=functic
    loss.append(func_min)
    print(d['grad'])
    if func_min < opt_loss:
        opt_loss = func_min
        opt_X = X.copy()
```

```
Iteration = 0
[-1.13819259e+14  2.11088499e+13  7.24103636e+13 ...  5.96972038e+12
  1.85995991e+13  1.44182818e+12]
Iteration = 1
[ 9.70059153e+12  1.30129005e+13  1.63229123e+10 ...  4.57716374e+12
 -6.35074773e+12 -1.16962584e+12]
Iteration = 2
[-7.88490564e+13 -6.56955681e+13 -6.55839828e+13 ...  1.81990851e+12
 -6.15783072e+12 -9.84203788e+11]
Iteration = 3
[-5.32359309e+12  2.57250443e+13  1.50716348e+13 ...  1.94227130e+12
  4.34667520e+09 -1.62685688e+12]
Iteration = 4
[-7.81255893e+13 -6.35482690e+13 -9.43444428e+13 ...  3.86211303e+12
 -1.52068869e+12 -1.64111424e+12]
Iteration = 5
[-1.63219086e+13 -1.65134074e+11 -9.60866497e+12 ...  2.12884980e+12
 -9.67161414e+11 -6.63994434e+11]
Iteration = 6
```

Plot the minimum loss each iteration.

In [18]:
```
plt.plot(loss)
plt.grid()
plt.xlabel('Iteration')
plt.ylabel('loss')
```

Out[18]:
```
Text(0, 0.5, 'loss')
```



Then convert output array to image.

VGG-16 was trained using Caffe, and Caffe uses OpenCV to load images which uses BGR by default, so both VGG models are expecting BGR images.

The official mean that gets used in the vgg16.py file is [123.68, 116.779, 103.939] (https://forums.fast.ai/t/how-is-vgg16-mean-calculated/4577/9 (https://forums.fast.ai/t/how-is-vgg16-mean-calculated/4577/9))

So we need to covert BGR to RGB before imshow

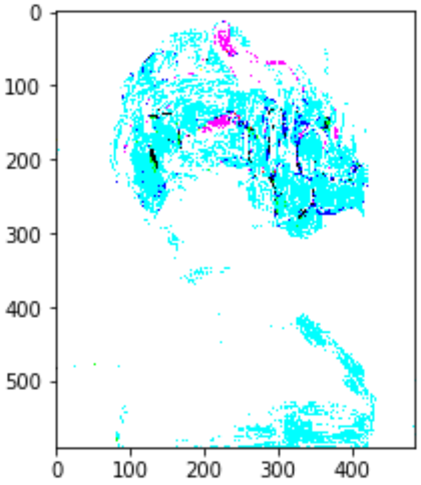
In [19]:

```
opt_X = opt_X.reshape((ncol, nrow,  3))
vgg_mean = [103.939,116.779,123.68]
for i in range(3):
    opt_X[:, :, i] = opt_X[:, :, i] + vgg_mean[i]
plt.imshow(opt_X)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] f
or floats or [0..255] for integers).
```

Out[19]:

```
<matplotlib.image.AxesImage at 0x7f3dd4526198>
```



Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

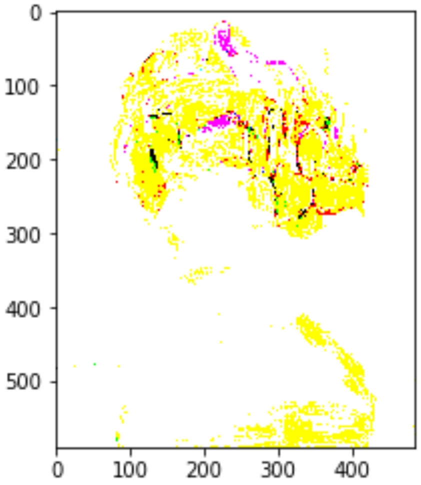
In [20]:

```
opt_X = opt_X[:, :, ::-1]
opt_X = np.clip(opt_X, 0, 255)
plt.imshow(opt_X)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] f
or floats or [0..255] for integers).
```

Out[20]:

```
<matplotlib.image.AxesImage at 0x7f3dd435b390>
```



Convert the numpy arrays to uint8 before passing them to Image.fromarray np array should have data type as uint8

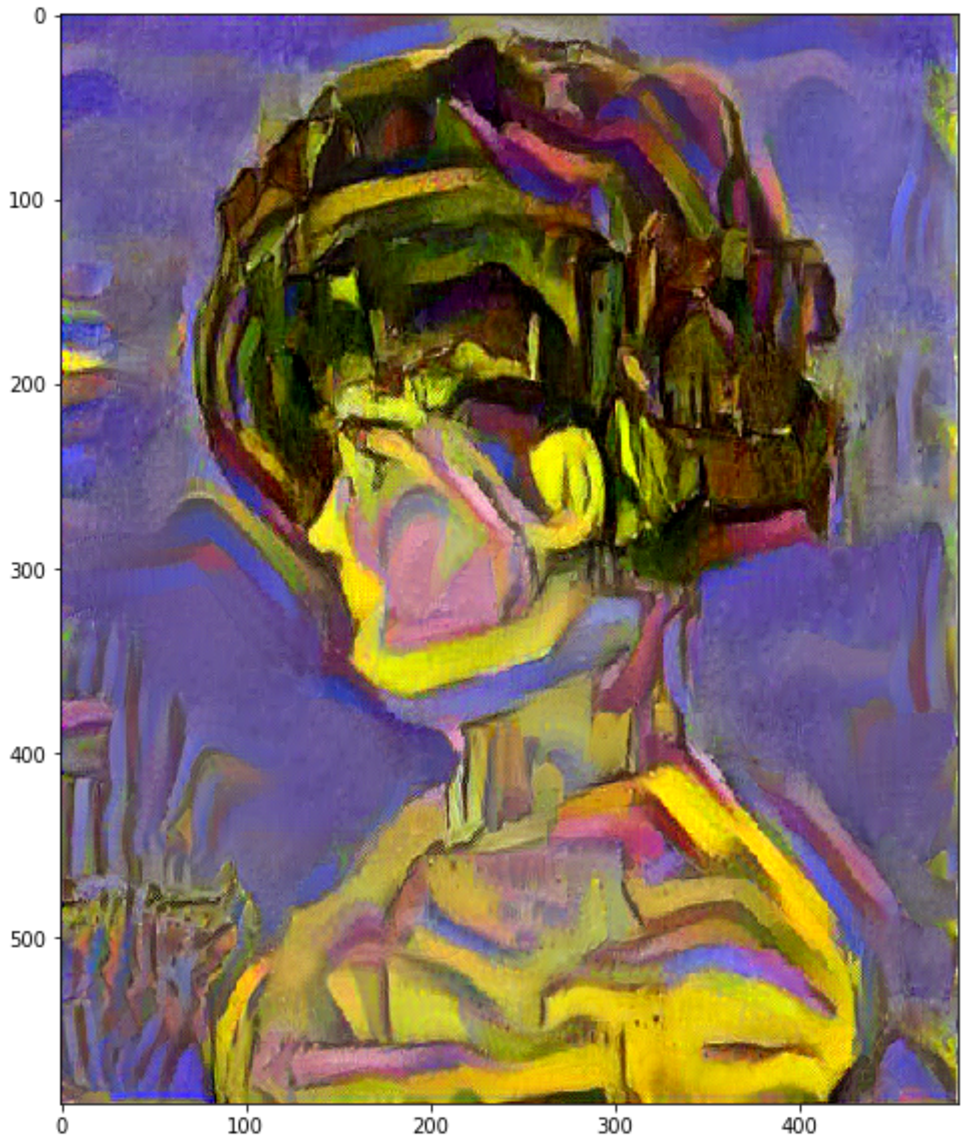
(https://stackoverflow.com/questions/10443295/combine-3-separate-numpy-arrays-to-an-rgb-image-in-python (https://stackoverflow.com/questions/10443295/combine-3-separate-numpy-arrays-to-an-rgb-image-in-python))

In [21]:

```
opt_X = opt_X.astype('uint8')
plt.figure(figsize=(10,10))
plt.imshow(opt_X)
```

Out[21]:

```
<matplotlib.image.AxesImage at 0x7f3dd4310a58>
```

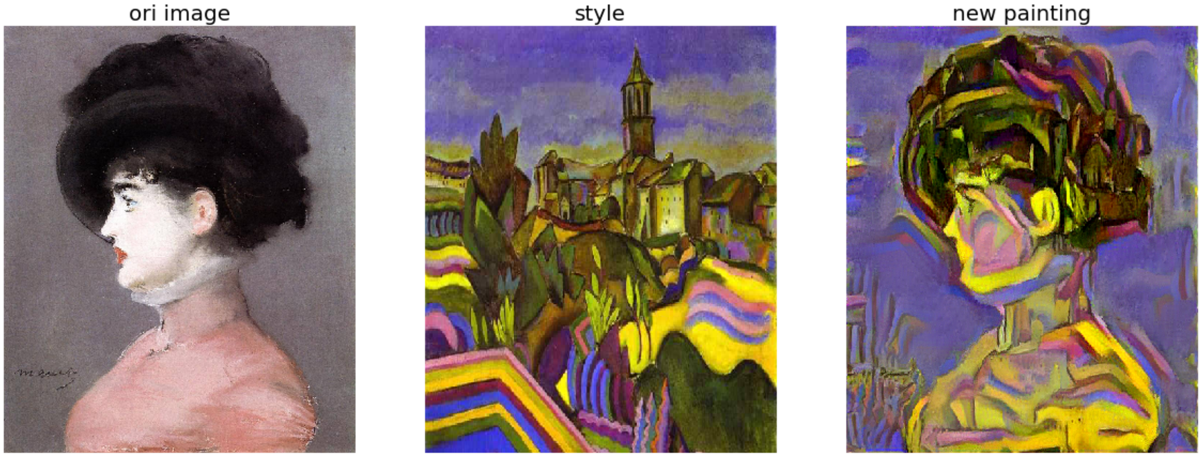


In [22]:

```
# Ori
plt.figure(figsize=(20,20))
plt.subplot(1,3,1)
plt.title("ori image",fontsize=20)
plt.axis('off')
plt.imshow(ori_image)
# style
plt.subplot(1,3,2)
plt.title("style",fontsize=20)
img_style = load_img(style_path)
plt.axis('off')
plt.imshow(style_image)
# new painting
plt.subplot(1,3,3)
plt.title("new painting",fontsize=20)
plt.axis('off')
plt.imshow(opt_X)
```

Out[22]:

```
<matplotlib.image.AxesImage at 0x7f3dd42b91d0>
```



ori image

style

new painting

In [ ]:

```

```