In [1]:

```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np
import os
from keras import backend as K
from keras.preprocessing.image import load_img, save_img, img_to_array
import matplotlib.pyplot as plt
from keras.applications import vgg19
from keras.models import Model
#from keras import optimizers
from scipy.optimize import fmin_l_bfgs_b
#from keras.applications.vgg19 import VGG19
#vgg19_weights = '../input/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5'
#vgg19 = VGG19(include_top = False, weights=vgg19_weights)
print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

```
Using TensorFlow backend.
```

```
['image-classification', 'best-artworks-of-all-time', 'vgg19']
```

In [2]:

```
StylePath = '../input/best-artworks-of-all-time/images/images/'
ContentPath = '../input/image-classification/validation/validation/travel and adventure/'
```

In [3]:

```
base_image_path = ContentPath+'13.jpg'
style_image_path = StylePath+'Pablo_Picasso/Pablo_Picasso_92.jpg'
```

In [4]:

```
# dimensions of the generated picture.
width, height = load_img(base_image_path).size
img_nrows = 400
img_ncols = int(width * img_nrows / height)
```

In [5]:

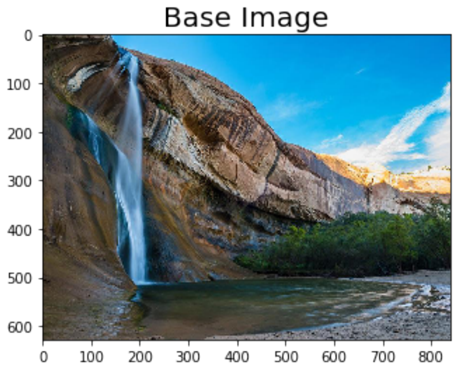
```
def preprocess_image(image_path):
    from keras.applications import vgg19
    img = load_img(image_path, target_size=(img_nrows, img_ncols))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = vgg19.preprocess_input(img)
    return img
```

In [6]:

```
plt.figure()
plt.title("Base Image",fontsize=20)
img1 = load_img(ContentPath+'13.jpg')
plt.imshow(img1)
```

Out[6]:

```
<matplotlib.image.AxesImage at 0x7f3814618f60>
```

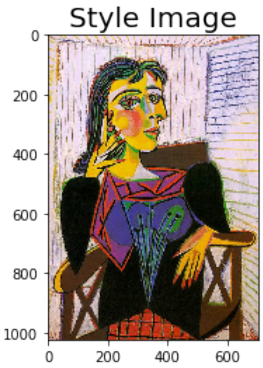


In [7]:

```
plt.figure()
plt.title("Style Image",fontsize=20)
img1 = load_img(StylePath+'Pablo_Picasso/Pablo_Picasso_92.jpg')
plt.imshow(img1)
```

Out[7]:

```
<matplotlib.image.AxesImage at 0x7f3811dae198>
```



In [8]:

```
# get tensor representations of our images

base_image = K.variable(preprocess_image(base_image_path))
style_reference_image = K.variable(preprocess_image(style_image_path))
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/op_def_library.py:263:
colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

In [9]:

```
K.image_data_format()
```

Out[9]:

```
'channels_last'
```

In [10]:

```
# this will contain our generated image
if K.image_data_format() == 'channels_first':
    combination_image = K.placeholder((1,3,img_nrows, img_ncols))
else:
    combination_image = K.placeholder((1,img_nrows, img_ncols,3))
```

In [11]:

```
# combine the 3 images into a single Keras tensor
input_tensor = K.concatenate([base_image,
                              style_reference_image,
                              combination_image
                              ], axis=0)
```

In [12]:

```
# build the VGG19 network with our 3 images as input
# the model will be loaded with pre-trained ImageNet weights
from keras.applications.vgg19 import VGG19
vgg19_weights = '../input/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5'
model = VGG19(input_tensor=input_tensor,
              include_top = False,
              weights=vgg19_weights)
#model = vgg19.VGG19(input_tensor=input_tensor,
#                    weights='imagenet', include_top=False)
print('Model loaded.')

```

```
Model loaded.
```

In [13]:

```
# Content layer where will pull our feature maps
content_layers = ['block5_conv2']

# Style layer we are interested in
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1'
               ]

num_content_layers = len(content_layers)
num_style_layers = len(style_layers)
```

In [14]:

```
outputs_dict = dict([(layer.name, layer.output) for layer in model.layers])
print(outputs_dict['block5_conv2'])
```

```
Tensor("block5_conv2/Relu:0", shape=(3, 25, 33, 512), dtype=float32)
```

In [15]:

```python
# an auxiliary loss function
# designed to maintain the "content" of the
# base image in the generated image
def get_content_loss(base_content, target):
    return K.sum(K.square(target - base_content))
```

In [16]:

```python
import tensorflow as tf
# the gram matrix of an image tensor (feature-wise outer product)
def gram_matrix(input_tensor):
    assert K.ndim(input_tensor)==3
    #if K.image_data_format() == 'channels_first':
    #    features = K.batch_flatten(input_tensor)
    #else:
    #    features = K.batch_flatten(K.permute_dimensions(input_tensor,(2,0,1)))
    #gram = K.dot(features, K.transpose(features))
    channels = int(input_tensor.shape[-1])
    a = tf.reshape(input_tensor, [-1, channels])
    n = tf.shape(a)[0]
    gram = tf.matmul(a, a, transpose_a=True)
    return gram#/tf.cast(n, tf.float32)

def get_style_loss(style, combination):
    assert K.ndim(style) == 3
    assert K.ndim(combination) == 3
    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = img_nrows*img_ncols
    return K.sum(K.square(S - C))#/(4.0 * (channels ** 2) * (size ** 2))

```

In [17]:

```python
# Get output layers corresponding to style and content layers
#style_outputs = [model.get_layer(name).output for name in style_layers]
#content_outputs = [model.get_layer(name).output for name in content_layers]
#model_outputs = style_outputs + content_outputs
```

In [18]:

```python
# Get the style and content feature representations from our model
#style_features = [style_layer[0] for style_layer in model_outputs[:num_style_layers]]
#content_features = [content_layer[1] for content_layer in model_outputs[num_style_layers:]]
```

In [19]:

```python
 #gram_style_features = [gram_matrix(style_feature) for style_feature in style_features]
```

In [20]:

```python
#style_output_features = model_outputs[:num_style_layers]
#content_output_features = model_outputs[num_style_layers:]
# Accumulate style losses from all layers
# Here, we equally weight each contribution of each loss layer
#weight_per_style_layer = 1.0 / float(num_style_layers)
#loss = K.variable(0.0)
#style_score = 0
#content_score = 0

#for target_style, comb_style in zip(gram_style_features, style_output_features):
#    style_score += weight_per_style_layer * get_style_loss(comb_style[0], target_style)
# Accumulate content losses from all layers
#weight_per_content_layer = 1.0 / float(num_content_layers)
#for target_content, comb_content in zip(content_features, content_output_features):
#    content_score += weight_per_content_layer* get_content_loss(comb_content[0], target_content)

#style_score *= style_weight
#content_score *= content_weight

# Get total loss
#loss = style_score + content_score
```

In [21]:

```python
content_weight=0.025
style_weight=1.0
# combine these loss functions into a single scalar
loss = K.variable(0.0)
layer_features = outputs_dict['block5_conv2']
base_image_features = layer_features[0, :, :, :]
combination_features = layer_features[2, :, :, :]
print('Layer Feature for Content Layers :: '+str(layer_features))
print('Base Image Feature :: '+str(base_image_features))
print('Combination Image Feature for Content Layers:: '+str(combination_features)+'\n')
loss += content_weight * get_content_loss(base_image_features,
                                          combination_features)

feature_layers = ['block1_conv1', 'block2_conv1',
                  'block3_conv1', 'block4_conv1',
                  'block5_conv1']
for layer_name in feature_layers:
    layer_features = outputs_dict[layer_name]
    style_reference_features = layer_features[1, :, :, :]
    combination_features = layer_features[2, :, :, :]
    print('Layer Feature for Style Layers :: '+str(layer_features))
    print('Style Image Feature :: '+str(style_reference_features))
    print('Combination Image Feature for Style Layers:: '+str(combination_features)+'\n')
    sl = get_style_loss(style_reference_features, combination_features)
    loss += (style_weight / len(feature_layers)) * sl

```

```
Layer Feature for Content Layers :: Tensor("block5_conv2/Relu:0", shape=(3, 25, 33, 512), dtype=float32)
Base Image Feature :: Tensor("strided_slice:0", shape=(25, 33, 512), dtype=float32)
Combination Image Feature for Content Layers:: Tensor("strided_slice_1:0", shape=(25, 33, 512), dtype=float32)

WARNING:tensorflow:Variable += will be deprecated. Use variable.assign_add if you want assignment to the variable
value or 'x = x + y' if you want a new python Tensor object.
Layer Feature for Style Layers :: Tensor("block1_conv1/Relu:0", shape=(3, 400, 535, 64), dtype=float32)
Style Image Feature :: Tensor("strided_slice_2:0", shape=(400, 535, 64), dtype=float32)
Combination Image Feature for Style Layers:: Tensor("strided_slice_3:0", shape=(400, 535, 64), dtype=float32)

Layer Feature for Style Layers :: Tensor("block2_conv1/Relu:0", shape=(3, 200, 267, 128), dtype=float32)
Style Image Feature :: Tensor("strided_slice_6:0", shape=(200, 267, 128), dtype=float32)
Combination Image Feature for Style Layers:: Tensor("strided_slice_7:0", shape=(200, 267, 128), dtype=float32)

Layer Feature for Style Layers :: Tensor("block3_conv1/Relu:0", shape=(3, 100, 133, 256), dtype=float32)
Style Image Feature :: Tensor("strided_slice_10:0", shape=(100, 133, 256), dtype=float32)
Combination Image Feature for Style Layers:: Tensor("strided_slice_11:0", shape=(100, 133, 256), dtype=float32)

Layer Feature for Style Layers :: Tensor("block4_conv1/Relu:0", shape=(3, 50, 66, 512), dtype=float32)
Style Image Feature :: Tensor("strided_slice_14:0", shape=(50, 66, 512), dtype=float32)
Combination Image Feature for Style Layers:: Tensor("strided_slice_15:0", shape=(50, 66, 512), dtype=float32)

Layer Feature for Style Layers :: Tensor("block5_conv1/Relu:0", shape=(3, 25, 33, 512), dtype=float32)
Style Image Feature :: Tensor("strided_slice_18:0", shape=(25, 33, 512), dtype=float32)
Combination Image Feature for Style Layers:: Tensor("strided_slice_19:0", shape=(25, 33, 512), dtype=float32)
```

In [22]:

```python
def deprocess_image(x):
    if K.image_data_format() == 'channels_first':
        x = x.reshape((3, img_nrows, img_ncols))
        x = x.transpose((1, 2, 0))
    else:
        x = x.reshape((img_nrows, img_ncols, 3))
    # Remove zero-center by mean pixel
    x[:, :, 0] += 103.939
    x[:, :, 1] += 116.779
    x[:, :, 2] += 123.68
    # 'BGR'->'RGB'
    x = x[:, :, ::-1]
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

In [23]:

```python
# get the gradients of the generated image wrt the loss
grads = K.gradients(loss, combination_image)
grads
```

Out[23]:

```
[<tf.Tensor 'gradients/concat_grad/Slice_2:0' shape=(1, 400, 535, 3) dtype=float32>]
```

In [24]:

```python
outputs = [loss]
if isinstance(grads, (list,tuple)):
    outputs += grads
else:
    outputs.append(grads)
f_outputs = K.function([combination_image], outputs)
f_outputs
```

Out[24]:

```
<keras.backend.tensorflow_backend.Function at 0x7f380b4c4b00>
```

In [25]:

```
# run scipy-based optimization (L-BFGS) over the pixels of the generated image
# so as to minimize the neural style loss
x_opt = preprocess_image(base_image_path)
```

In [26]:

```
def eval_loss_and_grads(x):
    if K.image_data_format() == 'channels_first':
        x = x.reshape((1, 3, img_nrows, img_ncols))
    else:
        x = x.reshape((1, img_nrows, img_ncols, 3))
    outs = f_outputs([x])
    loss_value = outs[0]
    if len(outs[1:]) == 1:
        grad_values = outs[1].flatten().astype('float64')
    else:
        grad_values = np.array(outs[1:]).flatten().astype('float64')
    return loss_value, grad_values

```

In [27]:

```
class Evaluator(object):

    def __init__(self):
        self.loss_value = None
        self.grads_values = None

    def loss(self, x):
        assert self.loss_value is None
        loss_value, grad_values = eval_loss_and_grads(x)
        self.loss_value = loss_value
        self.grad_values = grad_values
        return self.loss_value

    def grads(self, x):
        assert self.loss_value is not None
        grad_values = np.copy(self.grad_values)
        self.loss_value = None
        self.grad_values = None
        return grad_values
```

In [28]:

```
evaluator = Evaluator()
```

In [29]:

```
iterations=400
# Store our best result
best_loss, best_img = float('inf'), None
for i in range(iterations):
    print('Start of iteration', i)
    x_opt, min_val, info= fmin_l_bfgs_b(evaluator.loss,
                                        x_opt.flatten(),
                                        fprime=evaluator.grads,
                                        maxfun=20,
                                        disp=True,
                                       )
    print('Current loss value:', min_val)
    if min_val < best_loss:
        # Update best loss and best image from total loss.
        best_loss = min_val
        best_img = x_opt.copy()
```

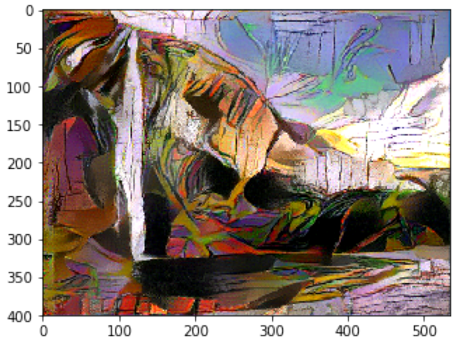
```
Current loss value: 1.2474705e+20
Start of iteration 54
Current loss value: 1.23089605e+20
Start of iteration 55
Current loss value: 1.2110161e+20
Start of iteration 56
Current loss value: 1.1818676e+20
Start of iteration 57
Current loss value: 1.15673655e+20
Start of iteration 58
Current loss value: 1.1447074e+20
Start of iteration 59
Current loss value: 1.1316408e+20
Start of iteration 60
Current loss value: 1.12150336e+20
Start of iteration 61
Current loss value: 1.1121649e+20
Start of iteration 62
Current loss value: 1.0979155e+20
Start of iteration 63
```

In [30]:

```
# save current generated image
imgx = deprocess_image(best_img.copy())
plt.imshow(imgx)
```

Out[30]:

```
<matplotlib.image.AxesImage at 0x7f3808d83e48>
```

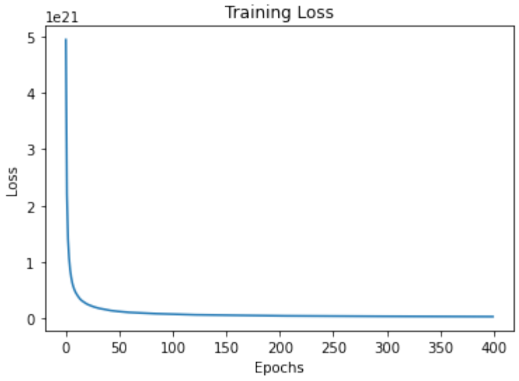


In [30]:

```
import numpy
import matplotlib.pyplot as plt
from graph import epochs, loss
```

In [33]:

```
plt.plot(epochs,loss)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.show()
```

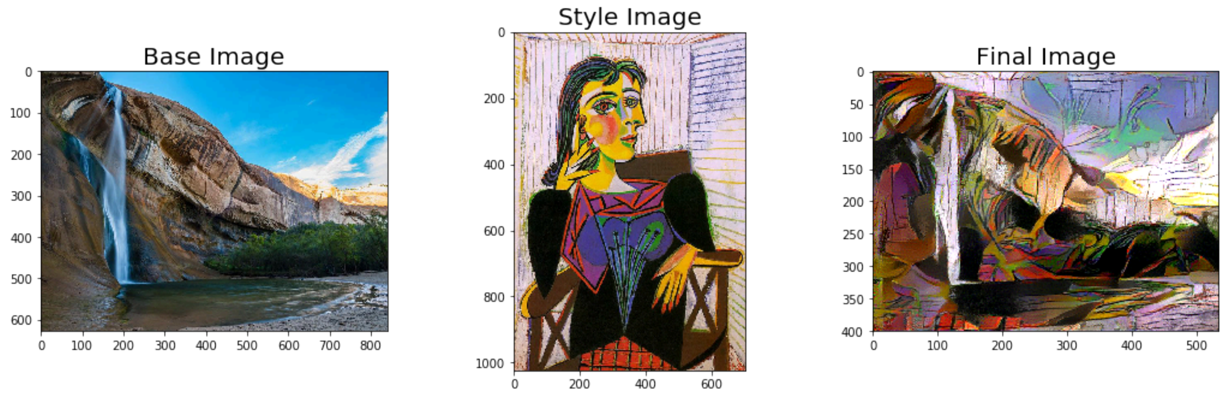


In [31]:

```
plt.figure(figsize=(30,30))
plt.subplot(5,5,1)
plt.title("Base Image",fontsize=20)
img_base = load_img(base_image_path)
plt.imshow(img_base)

plt.subplot(5,5,1+1)
plt.title("Style Image",fontsize=20)
img_style = load_img(style_image_path)
plt.imshow(img_style)

plt.subplot(5,5,1+2)
plt.title("Final Image",fontsize=20)
plt.imshow(imgx)
```

Out[31]:

```
<matplotlib.image.AxesImage at 0x7f3808e045f8>
```



In [32]:

```
def preprocess_image_instantiator(image_path,img_nrows,img_ncols):
    from keras.applications import vgg19
    img = load_img(image_path, target_size=(img_nrows, img_ncols))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = vgg19.preprocess_input(img)
    return img
```

In [33]:

```python
def Run_StyleTransfer(base_image_path, style_image_path):

    width, height = load_img(base_image_path).size
    img_nrows = 400
    img_ncols = int(width * img_nrows / height)

    base_image = K.variable(preprocess_image_instantiator(base_image_path,img_nrows,img_ncols))
    style_reference_image = K.variable(preprocess_image_instantiator(style_image_path,img_nrows,img_ncols))

    if K.image_data_format() == 'channels_first':
        combination_image = K.placeholder((1,3,img_nrows, img_ncols))
    else:
        combination_image = K.placeholder((1,img_nrows, img_ncols,3))

    input_tensor = K.concatenate([base_image,
                                  style_reference_image,
                                  combination_image
                                  ], axis=0)
    from keras.applications.vgg19 import VGG19
    vgg19_weights = '../input/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5'
    model = VGG19(input_tensor=input_tensor,
                  include_top = False,
                  weights=vgg19_weights)
    outputs_dict = dict([(layer.name, layer.output) for layer in model.layers])

    content_weight=0.025
    style_weight=1.0
    # combine these loss functions into a single scalar
    loss = K.variable(0.0)
    layer_features = outputs_dict['block5_conv2']
    base_image_features = layer_features[0, :, :, :]
    combination_features = layer_features[2, :, :, :]
    #print('Layer Feature for Content Layers :: '+str(layer_features))
    #print('Base Image Feature :: '+str(base_image_features))
    #print('Combination Image Feature for Content Layers:: '+str(combination_image_features))
    loss += content_weight * get_content_loss(base_image_features,
                                              combination_features)

    feature_layers = ['block1_conv1', 'block2_conv1',
                      'block3_conv1', 'block4_conv1',
                      'block5_conv1']
    for layer_name in feature_layers:
        layer_features = outputs_dict[layer_name]
        style_reference_features = layer_features[1, :, :, :]
        combination_features = layer_features[2, :, :, :]
        #print('Layer Feature for Style Layers :: '+str(layer_features))
        #print('Style Image Feature :: '+str(style_reference_features))
        #print('Combination Image Feature for Style Layers:: '+str(combination_features))
        sl = get_style_loss(style_reference_features, combination_features)
        loss += (style_weight / len(feature_layers)) * sl

    grads = K.gradients(loss, combination_image)

    outputs = [loss]
    if isinstance(grads, (list,tuple)):
        outputs += grads
    else:
        outputs.append(grads)
    f_outputs = K.function([combination_image], outputs)

    x_opt = preprocess_image(base_image_path)

    evaluator = Evaluator()
    iterations=200
    # Store our best result
    best_loss, best_img = float('inf'), None
    for i in range(iterations):
        #print('Start of iteration', i)
        x_opt, min_val, info= fmin_l_bfgs_b(evaluator.loss,
                                            x_opt.flatten(),
                                            fprime=evaluator.grads,
                                            maxfun=20,
                                            disp=True,
                                           )
        #print('Current loss value:', min_val)
        if min_val < best_loss:
            # Update best loss and best image from total loss.
            best_loss = min_val
            best_img = x_opt.copy()
    imgx = deprocess_image(best_img.copy())

    return imgx
```

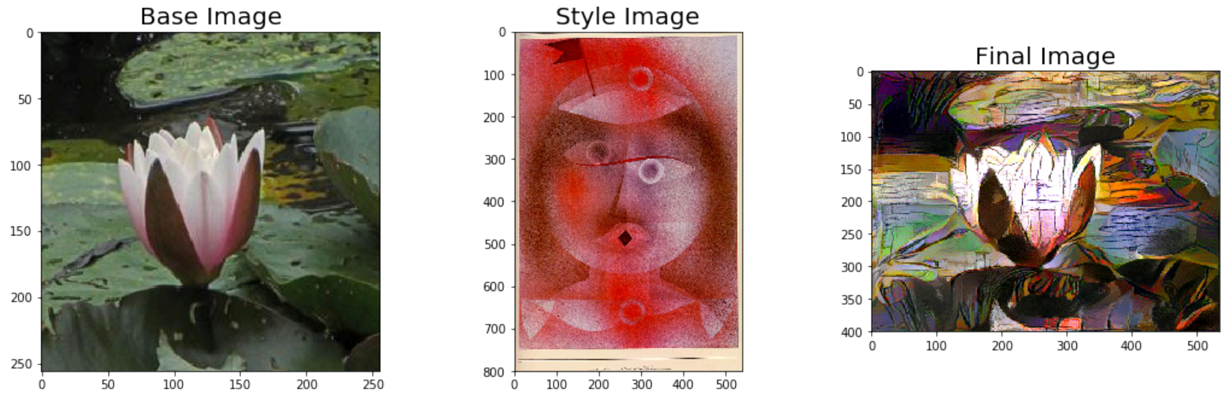
In [34]:

```
base_image_path_1 = '../input/image-classification/images/images/travel and adventure/Places365_val_00005821.jpg'
plt.figure(figsize=(30,30))
plt.subplot(5,5,1)
plt.title("Base Image",fontsize=20)
img_base = load_img(base_image_path_1)
plt.imshow(img_base)

style_image_path_1 = '../input/best-artworks-of-all-time/images/images/Paul_Klee/Paul_Klee_96.jpg'
plt.subplot(5,5,1+1)
plt.title("Style Image",fontsize=20)
img_style = load_img(style_image_path_1)
plt.imshow(img_style)

plt.subplot(5,5,1+2)
imgg = Run_StyleTransfer(base_image_path_1, style_image_path_1)
plt.title("Final Image",fontsize=20)
plt.imshow(imgg)
```

```
WARNING:tensorflow:Variable += will be deprecated. Use variable.assign_add if you want assignment to the variable value or 'x = x + y' if you want a new python Tensor object.
```

Out[34]:

```
<matplotlib.image.AxesImage at 0x7f380b113978>
```

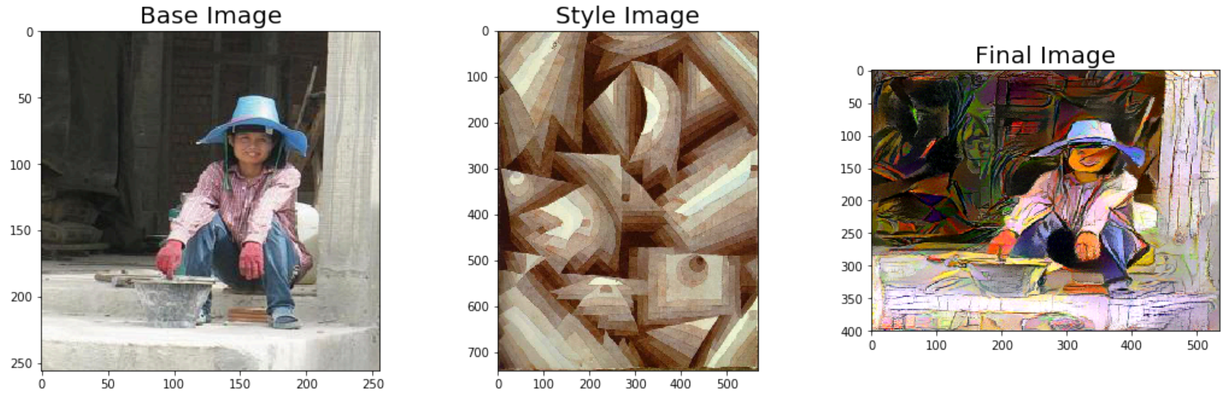


In [35]:

```
base_image_path_2 = '../input/image-classification/images/images/travel and adventure/Places365_val_00005982.jpg'
plt.figure(figsize=(30,30))
plt.subplot(5,5,1)
plt.title("Base Image",fontsize=20)
img_base = load_img(base_image_path_2)
plt.imshow(img_base)

style_image_path_2 = '../input/best-artworks-of-all-time/images/images/Paul_Klee/Paul_Klee_24.jpg'
plt.subplot(5,5,1+1)
plt.title("Style Image",fontsize=20)
img_style = load_img(style_image_path_2)
plt.imshow(img_style)

plt.subplot(5,5,1+2)
imga = Run_StyleTransfer(base_image_path_2, style_image_path_2)
plt.title("Final Image",fontsize=20)
plt.imshow(imga)
```

Out[35]:

```
<matplotlib.image.AxesImage at 0x7f380a946898>
```

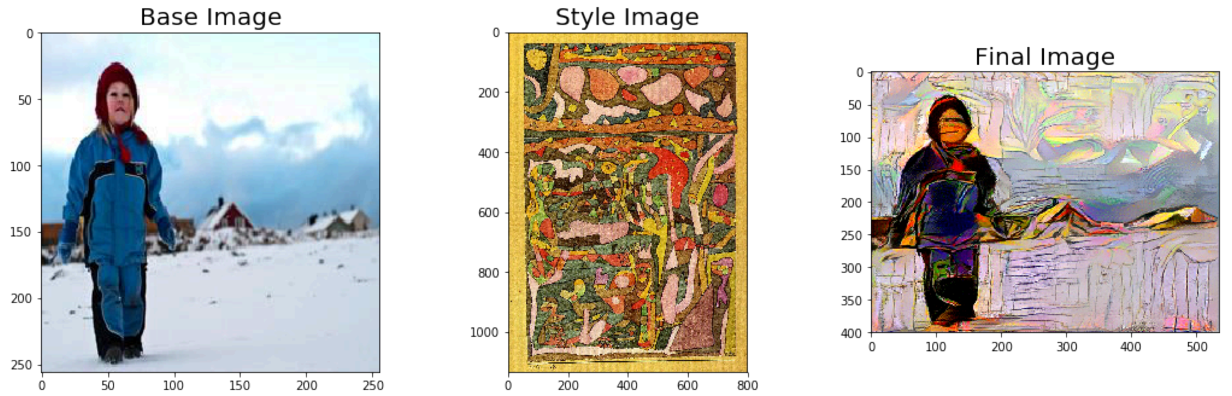


In [36]:

```
base_image_path_3 = '../input/image-classification/images/images/travel and adventure/Places365_val_00005752.jpg'
plt.figure(figsize=(30,30))
plt.subplot(5,5,1)
plt.title("Base Image",fontsize=20)
img_base = load_img(base_image_path_3)
plt.imshow(img_base)

style_image_path_3 = '../input/best-artworks-of-all-time/images/images/Paul_Klee/Paul_Klee_83.jpg'
plt.subplot(5,5,1+1)
plt.title("Style Image",fontsize=20)
img_style = load_img(style_image_path_3)
plt.imshow(img_style)

plt.subplot(5,5,1+2)
imgy = Run_StyleTransfer(base_image_path_3, style_image_path_3)
plt.title("Final Image",fontsize=20)
plt.imshow(imgy)
```

Out[36]:

```
<matplotlib.image.AxesImage at 0x7f380013f358>
```



In [37]:

```
```