In [1]:

```python
from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, BatchNormalization, Activation
from keras.models import Model
from keras import backend as K

def Encoder():
    input_img = Input(shape=(256, 384, 3))  # adapt this if using `channels_first` image data format
    x = Conv2D(64, (3, 3), padding='same')(input_img)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(32, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(16, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    encoded = MaxPooling2D((2, 2), padding='same')(x)
    return Model(input_img, encoded)

def Decoder():
    input_img = Input(shape=(32, 48, 16))  # adapt this if using `channels_first` image data format
    x = Conv2D(16, (3, 3), padding='same')(input_img)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(32, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(64, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(3, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    decoded = Activation('sigmoid')(x)
    return Model(input_img, decoded)
```

```
/anaconda3/envs/panicroom/lib/python3.6/site-packages/h5py/__init_
_.py:36: FutureWarning: Conversion of the second argument of issub
dtype from `float` to `np.floating` is deprecated. In future, it w
ill be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [2]:

```
# define input to the model:
x = Input(shape=(256, 384, 3))  # adapt this if using `channels_first` image d
ata format

# make the model:
autoencoder = Model(x, Decoder()(Encoder()(x)))

# compile the model:
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
```

In [3]:

```
Encoder().summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_4 (InputLayer)         (None, 256, 384, 3)       0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 256, 384, 64)      1792      
_________________________________________________________________
batch_normalization_8 (Batch (None, 256, 384, 64)      256       
_________________________________________________________________
activation_8 (Activation)    (None, 256, 384, 64)      0         
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 128, 192, 64)      0         
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 128, 192, 32)      18464     
_________________________________________________________________
batch_normalization_9 (Batch (None, 128, 192, 32)      128       
_________________________________________________________________
activation_9 (Activation)    (None, 128, 192, 32)      0         
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 64, 96, 32)        0         
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 64, 96, 16)        4624      
_________________________________________________________________
batch_normalization_10 (Batc (None, 64, 96, 16)        64        
_________________________________________________________________
activation_10 (Activation)   (None, 64, 96, 16)        0         
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 32, 48, 16)        0         
=================================================================
Total params: 25,328
Trainable params: 25,104
Non-trainable params: 224
_________________________________________________________________
```

In [4]:

```
Decoder().summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_5 (InputLayer)         (None, 32, 48, 16)        0         
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 32, 48, 16)        2320      
_________________________________________________________________
batch_normalization_11 (Batc (None, 32, 48, 16)        64        
_________________________________________________________________
activation_11 (Activation)   (None, 32, 48, 16)        0         
_________________________________________________________________
up_sampling2d_4 (UpSampling2 (None, 64, 96, 16)        0         
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 64, 96, 32)        4640      
_________________________________________________________________
batch_normalization_12 (Batc (None, 64, 96, 32)        128       
_________________________________________________________________
activation_12 (Activation)   (None, 64, 96, 32)        0         
_________________________________________________________________
up_sampling2d_5 (UpSampling2 (None, 128, 192, 32)      0         
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 128, 192, 64)      18496     
_________________________________________________________________
batch_normalization_13 (Batc (None, 128, 192, 64)      256       
_________________________________________________________________
activation_13 (Activation)   (None, 128, 192, 64)      0         
_________________________________________________________________
up_sampling2d_6 (UpSampling2 (None, 256, 384, 64)      0         
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 256, 384, 3)       1731      
_________________________________________________________________
batch_normalization_14 (Batc (None, 256, 384, 3)       12        
_________________________________________________________________
activation_14 (Activation)   (None, 256, 384, 3)       0         
=================================================================
Total params: 27,647
Trainable params: 27,417
Non-trainable params: 230
_________________________________________________________________
```

In [5]:

```
# load all data, normalise, reshape
import os
from PIL import Image
import numpy as np

all_data = []
path = './images-small/'
data_size = 2000
for filename in os.listdir(path)[:data_size]:
    img_path = path + filename
    image = Image.open(img_path)
    all_data.append(np.array(image))
    print('Processed ', len(all_data), ' of ', data_size, ' images.', end='\r'
)

x_train = np.array(all_data[:1500])
x_train = x_train.astype('float32') / 255.
print('x_train shape:', x_train.shape)

x_test = np.array(all_data[1500:])
x_test = x_test.astype('float32') / 255.
print('x_train shape:', x_test.shape)
```

```
x_train shape: (1500, 256, 384, 3)images. of  2000  images.
x_train shape: (500, 256, 384, 3)
```

In [8]:

```
autoencoder.fit(x_train, x_train,
            epochs=25,
            batch_size=64,
            shuffle=True,
            validation_data=(x_test, x_test))
```

```
Train on 1500 samples, validate on 500 samples
Epoch 1/25
1500/1500 [==============================] - 1342s 895ms/step - lo
ss: 0.5230 - val_loss: 0.5289
Epoch 2/25
1500/1500 [==============================] - 1296s 864ms/step - lo
ss: 0.5172 - val_loss: 0.5154
Epoch 3/25
1500/1500 [==============================] - 1297s 864ms/step - lo
ss: 0.5128 - val_loss: 0.5138
Epoch 4/25
1500/1500 [==============================] - 1292s 861ms/step - lo
ss: 0.5095 - val_loss: 0.5105
Epoch 5/25
1500/1500 [==============================] - 1305s 870ms/step - lo
ss: 0.5066 - val_loss: 0.5060
Epoch 6/25
1500/1500 [==============================] - 1311s 874ms/step - lo
ss: 0.5038 - val_loss: 0.5106
Epoch 7/25
```

```
1500/1500 [==============================] - 1305s 870ms/step - lo
ss: 0.5014 - val_loss: 0.5139
Epoch 8/25
1500/1500 [==============================] - 1294s 863ms/step - lo
ss: 0.4997 - val_loss: 0.5072
Epoch 9/25
1500/1500 [==============================] - 1302s 868ms/step - lo
ss: 0.4976 - val_loss: 0.5048
Epoch 10/25
1500/1500 [==============================] - 1303s 869ms/step - lo
ss: 0.4964 - val_loss: 0.4963
Epoch 11/25
1500/1500 [==============================] - 1303s 869ms/step - lo
ss: 0.4950 - val_loss: 0.5006
Epoch 12/25
1500/1500 [==============================] - 1302s 868ms/step - lo
ss: 0.4939 - val_loss: 0.4926
Epoch 13/25
1500/1500 [==============================] - 1300s 867ms/step - lo
ss: 0.4925 - val_loss: 0.4970
Epoch 14/25
1500/1500 [==============================] - 1300s 867ms/step - lo
ss: 0.4916 - val_loss: 0.4898
Epoch 15/25
1500/1500 [==============================] - 1287s 858ms/step - lo
ss: 0.4907 - val_loss: 0.4903
Epoch 16/25
1500/1500 [==============================] - 1298s 866ms/step - lo
ss: 0.4899 - val_loss: 0.4929
Epoch 17/25
1500/1500 [==============================] - 1310s 873ms/step - lo
ss: 0.4891 - val_loss: 0.4883
Epoch 18/25
1500/1500 [==============================] - 1305s 870ms/step - lo
ss: 0.4890 - val_loss: 0.4880
Epoch 19/25
1500/1500 [==============================] - 1303s 868ms/step - lo
ss: 0.4880 - val_loss: 0.4872
Epoch 20/25
1500/1500 [==============================] - 1298s 865ms/step - lo
ss: 0.4873 - val_loss: 0.4917
Epoch 21/25
1500/1500 [==============================] - 1298s 865ms/step - lo
ss: 0.4872 - val_loss: 0.4868
Epoch 22/25
1500/1500 [==============================] - 1296s 864ms/step - lo
ss: 0.4865 - val_loss: 0.4876
Epoch 23/25
1500/1500 [==============================] - 1292s 861ms/step - lo
ss: 0.4861 - val_loss: 0.4869
Epoch 24/25
1500/1500 [==============================] - 1290s 860ms/step - lo
ss: 0.4857 - val_loss: 0.4868
Epoch 25/25
1500/1500 [==============================] - 1303s 869ms/step - lo
ss: 0.4855 - val_loss: 0.4856
```

Out[8]:

```
<keras.callbacks.History at 0x186cbacdd8>
```

In [11]:

```python
autoencoder.save('./lomo-autoencoder-25epoch-backup')
```

In [12]:

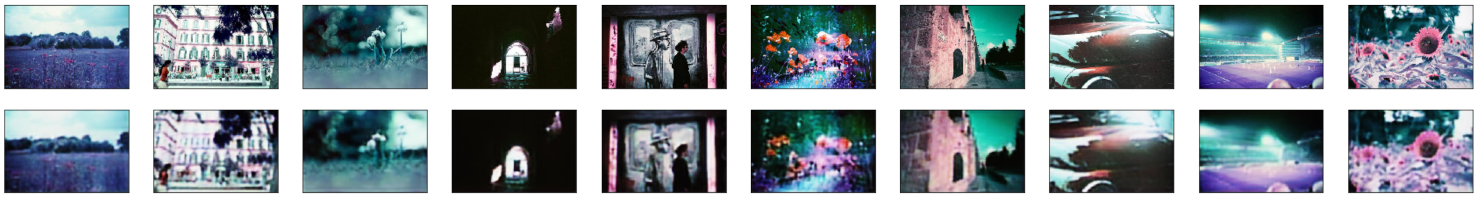
```python
# definition to show original image and reconstructed image
def showOrigDec(orig, dec, num=10):
    import matplotlib.pyplot as plt
    %matplotlib inline
    n = num
    plt.figure(figsize=(30, 4))

    for i in range(n):
        # display original
        ax = plt.subplot(2, n, i+1)
        plt.imshow(orig[i].reshape(256, 384, 3))
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

        # display reconstruction
        ax = plt.subplot(2, n, i +1 + n)
        plt.imshow(dec[i].reshape(256, 384, 3))
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
    plt.show()
```

In [15]:

```python
encoded_imgs = Encoder().predict(x_train)
decoded_imgs = Decoder().predict(encoded_imgs)
autodecoded_imgs = autoencoder.predict(x_train)
```

In [16]:

```python
showOrigDec(x_train, autodecoded_imgs, num=10)
```



In [134]:

```python
input_size = np.prod(x_train.shape[1:])
code_size = np.prod(encoded_imgs.shape[1:])
compression_factor = input_size/code_size
print ("Compression factor is ", compression_factor)
```

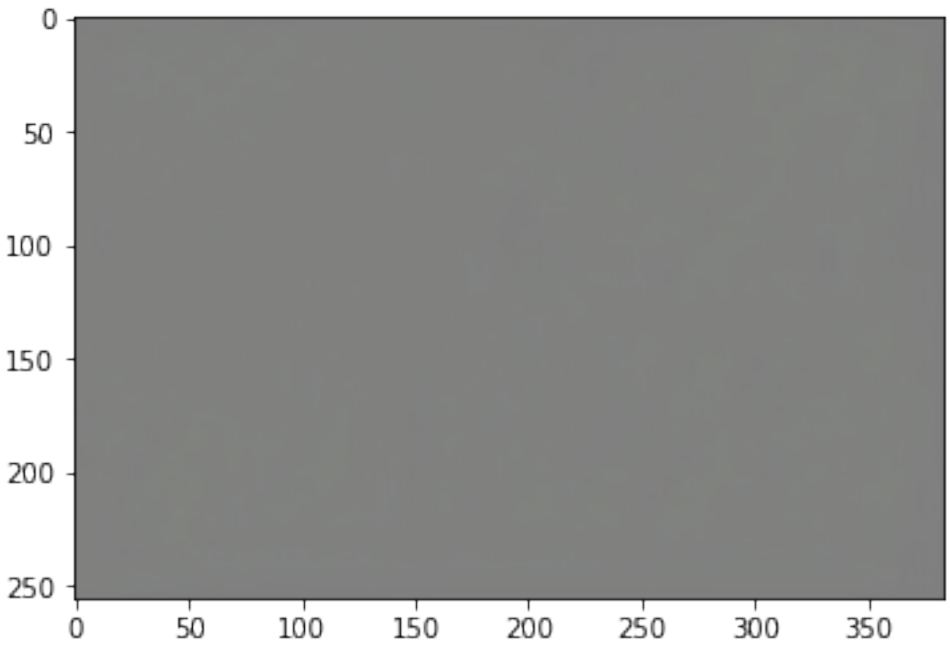
```
Compression factor is  12.0
```

In [30]:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.imshow(decoded_imgs[1])
```

Out[30]:

```
<matplotlib.image.AxesImage at 0x186d4afba8>
```

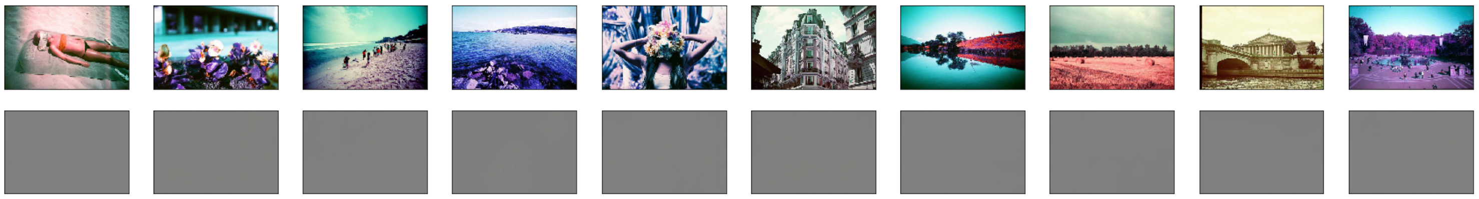


In [13]:

```
encoded_imgs = Encoder().predict(x_test)
decoded_imgs = Decoder().predict(encoded_imgs)
```

In [31]:

```
showOrigDec(x_test, decoded_imgs, num=10)
```

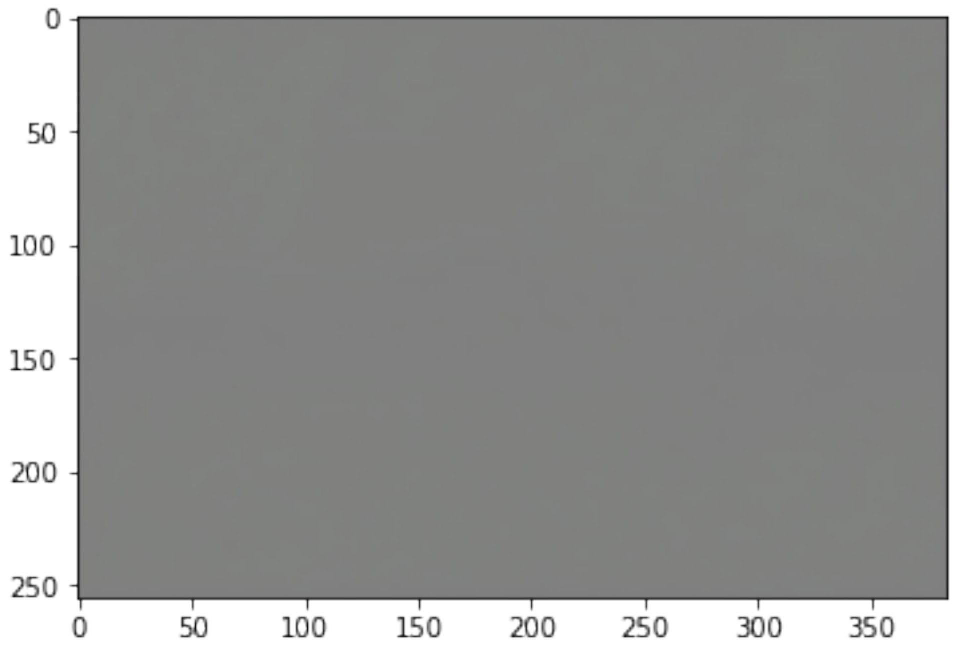


In [28]:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.imshow(decoded_imgs[-1])
```

Out[28]:

```
<matplotlib.image.AxesImage at 0x1870377320>
```



In [ ]:

```
decoded_imgs[0]
```

In [32]:

```
# load new data for testing the network
import os
from PIL import Image
import numpy as np

new_data = []
path = './images-small/'
for filename in os.listdir(path)[2000:2500]:
    img_path = path + filename
    image = Image.open(img_path)
    new_data.append(np.array(image))
    print('Processed ', len(new_data), ' of ', 500, ' images.', end='\r')

x_val = np.array(new_data[:1500])
x_val = x_val.astype('float32') / 255.
print('x_val shape:', x_val.shape)
```

```
x_val shape: (500, 256, 384, 3).
```

In [34]:

```
encoded_imgs = Encoder().predict(x_val)
decoded_imgs = Decoder().predict(encoded_imgs)
autodecoded_imgs = autoencoder.predict(x_val)
```

In [40]:

```
showOrigDec(x_val, autodecoded_imgs, num=10)
```

In [127]:

```
for data in range(0, :
    img = Image.fromarray((data[0] * 255).astype(np.uint8))
filename = 1
img.save('./images-decoded/'+str(filename)+'.jpg')
```

In [107]:

```
scaledup = encoded_imgs*150
```

In [108]:

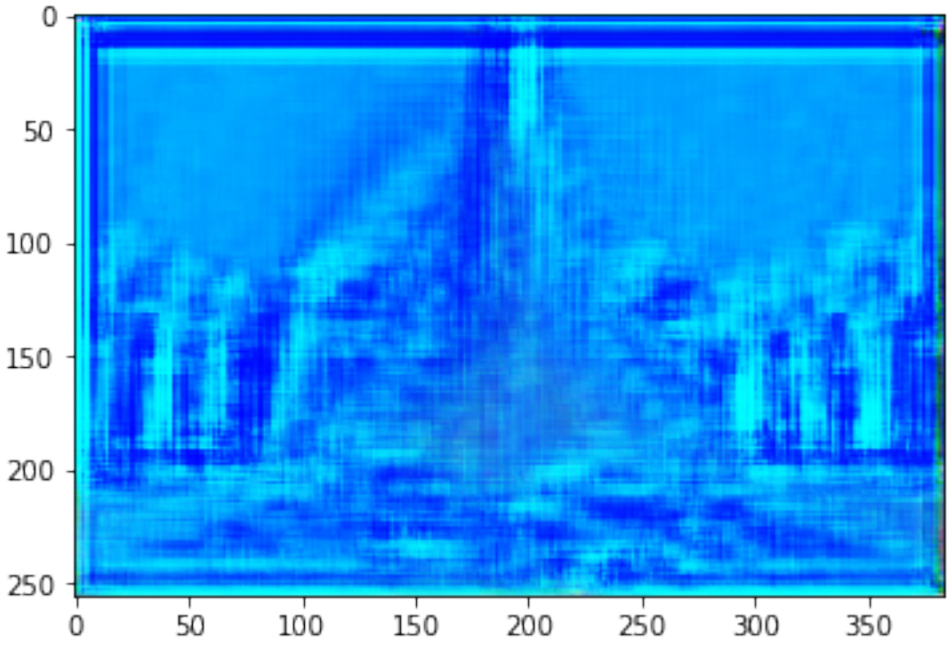
```
scaledup = Decoder().predict(scaledup[0:1])
```

In [109]:

```
plt.imshow(scaledup[0])
```

Out[109]:

```
<matplotlib.image.AxesImage at 0x191da30e10>
```

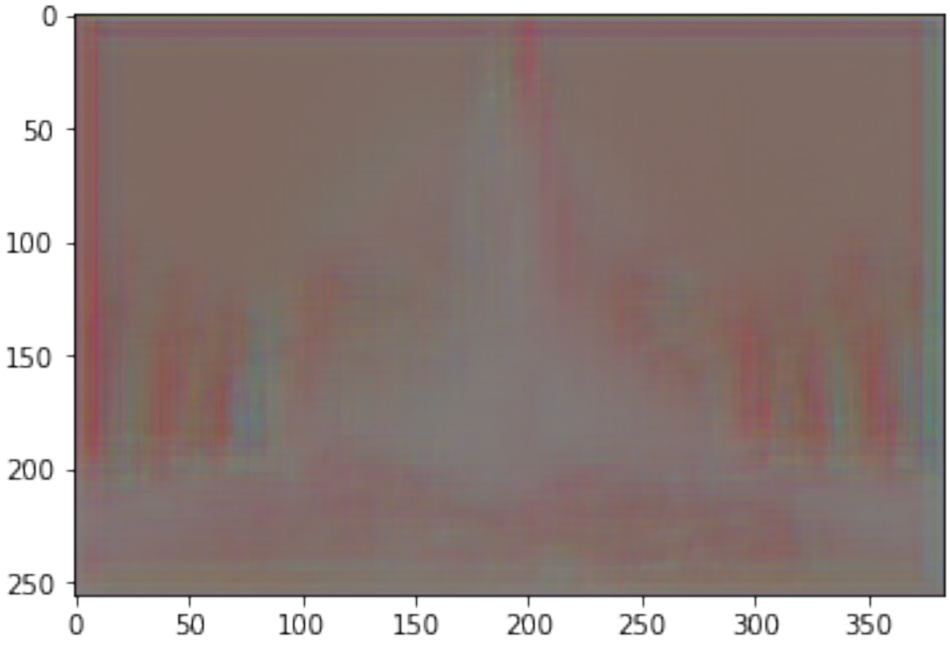


In [77]:

```
from __future__ import print_function
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

In [85]:

```
def show(scale):
    scaledup = encoded_imgs*scale
    scaledup = Decoder().predict(scaledup[0:1])
    plt.imshow(scaledup[0])

interactive(show, scale=10)
```


0
50
100
150
200
250
0
50
100
150
200
250
300
350