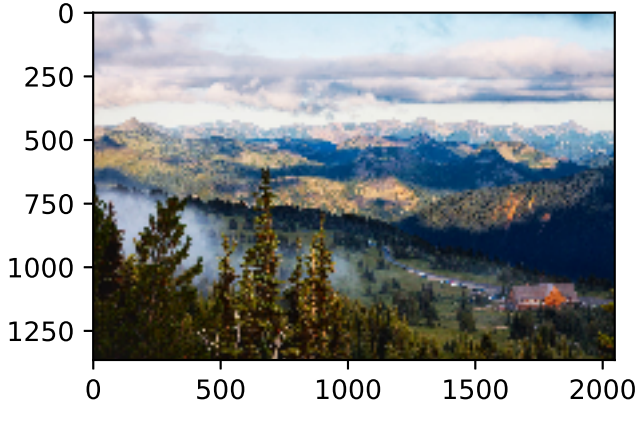
Neural Style Transfer

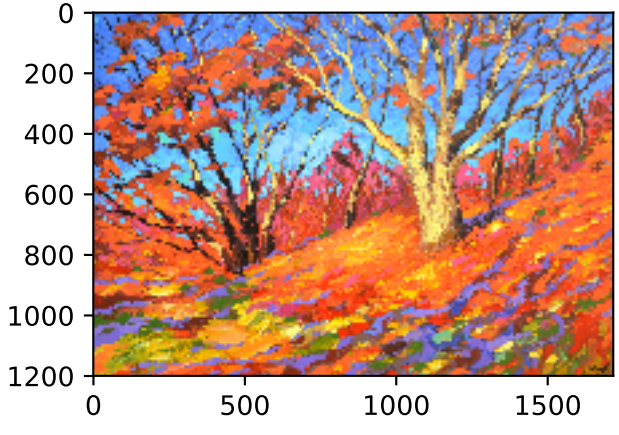
In [ ]:
```
%matplotlib inline
import d2l
from mxnet import autograd, gluon, image, init, nd
from mxnet.gluon import model_zoo, nn
import time
```

Read the Content and Style Images

In [8]:
```
d2l.set_figsize()
content_img = image.imread('rainier.jpg')
d2l.plt.imshow(content_img.asnumpy());
```



In [9]:
```
style_img = image.imread('autumn_oak.jpg')
d2l.plt.imshow(style_img.asnumpy());
```



Features

Preprocessing and Postprocessing

In [3]:
```
rgb_mean = nd.array([0.485, 0.456, 0.406])
rgb_std = nd.array([0.229, 0.224, 0.225])

def preprocess(img, image_shape):
    img = image.imresize(img, *image_shape)
    img = (img.astype('float32') / 255 - rgb_mean) / rgb_std
    return img.transpose((2, 0, 1)).expand_dims(axis=0)

def postprocess(img):
    img = img[0].as_in_context(rgb_std.context)
    return (img.transpose((1, 2, 0)) * rgb_std + rgb_mean).clip(0, 1)
```

Select Content and Style Layers

In [5]:
```
pretrained_net = model_zoo.vision.vgg19(pretrained=True)
style_layers, content_layers = [0, 5, 10, 19, 28], [25]
net = nn.Sequential()
for i in range(max(content_layers + style_layers) + 1):
    net.add(pretrained_net.features[i])
```

Extract Features

In [7]:
```
def extract_features(X, content_layers, style_layers):
    contents = []
    styles = []
    for i in range(len(net)):
        X = net[i](X)
        if i in style_layers:
            styles.append(X)
        if i in content_layers:
            contents.append(X)
    return contents, styles
```

Get Contents and Style Features

In [8]:
```
def get_contents(image_shape, ctx):
    content_X = preprocess(content_img, image_shape).copyto(ctx)
    contents_Y, _ = extract_features(content_X, content_layers, style_layers)
    return content_X, contents_Y

def get_styles(image_shape, ctx):
    style_X = preprocess(style_img, image_shape).copyto(ctx)
    _, styles_Y = extract_features(style_X, content_layers, style_layers)
    return style_X, styles_Y
```

Define the Loss Function

Content Loss

In [9]:
```
def content_loss(Y_hat, Y):
    return (Y_hat - Y).square().mean()
```

Style Loss

In [10]:
```
def gram(X):
    num_channels, n = X.shape[1], X.size // X.shape[1]
    X = X.reshape((num_channels, n))
    return nd.dot(X, X.T) / (num_channels * n)

def style_loss(Y_hat, gram_Y):
    return (gram(Y_hat) - gram_Y).square().mean()
```

Total Variance Loss

In [12]:
```
def tv_loss(Y_hat):
    return 0.5 * ((Y_hat[:, :, 1:, :] - Y_hat[:, :, :-1, :]).abs().mean() +
                  (Y_hat[:, :, :, 1:] - Y_hat[:, :, :, :-1]).abs().mean())
```

Loss Function

In [13]:
```
style_channels = [net[l].weight.shape[0] for l in style_layers]
content_weight, style_weight, tv_weight = 1, 1e3, 10

def compute_loss(X, contents_Y_hat, styles_Y_hat, contents_Y, styles_Y_gram):
    contents_l = [content_loss(Y_hat, Y) * content_weight for Y_hat, Y in zip(
        contents_Y_hat, contents_Y)]
    styles_l = [style_loss(Y_hat, Y) * style_weight for Y_hat, Y in zip(
        styles_Y_hat, styles_Y_gram)]
    tv_l = tv_loss(X) * tv_weight
    l = nd.add_n(*styles_l) + nd.add_n(*contents_l) + tv_l
    return contents_l, styles_l, tv_l, l
```

Create and Initialize the Composite Image

In [14]:
```
class GeneratedImage(nn.Block):
    def __init__(self, img_shape, **kwargs):
        super(GeneratedImage, self).__init__(**kwargs)
        self.weight = self.params.get('weight', shape=img_shape)

    def forward(self):
        return self.weight.data()
```

Initialize

In [15]:
```
def get_inits(X, ctx, lr, styles_Y):
    gen_img = GeneratedImage(X.shape)
    gen_img.initialize(init.Constant(X), ctx=ctx, force_reinit=True)
    trainer = gluon.Trainer(gen_img.collect_params(), 'adam',
                            {'learning_rate': lr})
    styles_Y_gram = [gram(Y) for Y in styles_Y]
    return gen_img(), styles_Y_gram, trainer
```

Train

In [16]:
```
def train(X, contents_Y, styles_Y, ctx, lr, max_epochs, lr_decay_epoch):
    X, styles_Y_gram, trainer = get_inits(X, ctx, lr, styles_Y)
    for i in range(max_epochs):
        start = time.time()
        with autograd.record():
            contents_Y_hat, styles_Y_hat = extract_features(
                X, content_layers, style_layers)
            contents_l, styles_l, tv_l, l = compute_loss(
                X, contents_Y_hat, styles_Y_hat, contents_Y, styles_Y_gram)
        l.backward()
        trainer.step(1)
        nd.waitall()
        if i % 50 == 0 and i != 0:
            print('epoch %3d, content loss %.2f, style loss %.2f, '
                  'TV loss %.2f, %.2f sec'
                  % (i, nd.add_n(*contents_l).asscalar(),
                     nd.add_n(*styles_l).asscalar(), tv_l.asscalar(),
                     time.time() - start))
        if i % lr_decay_epoch == 0 and i != 0:
            trainer.set_learning_rate(trainer.learning_rate * 0.1)
            print('change lr to %.1e' % trainer.learning_rate)
    return X
```

Train a 150 x 225 composite image

In [17]:
```
ctx, image_shape = d2l.try_gpu(), (225, 150)
net.collect_params().reset_ctx(ctx)
content_X, contents_Y = get_contents(image_shape, ctx)
style_X, styles_Y = get_styles(image_shape, ctx)
output = train(content_X, contents_Y, styles_Y, ctx, 0.01, 500, 200)
d2l.plt.imsave('neural-style-1.png', postprocess(output).asnumpy())
```

```
epoch  50, content loss 10.12, style loss 29.39, TV loss 3.46, 0.03 sec
epoch 100, content loss 7.50, style loss 15.44, TV loss 3.90, 0.03 sec
epoch 150, content loss 6.31, style loss 10.38, TV loss 4.15, 0.03 sec
epoch 200, content loss 5.66, style loss 8.10, TV loss 4.29, 0.03 sec
change lr to 1.0e-03
epoch 250, content loss 5.60, style loss 7.93, TV loss 4.30, 0.03 sec
epoch 300, content loss 5.54, style loss 7.78, TV loss 4.31, 0.03 sec
epoch 350, content loss 5.48, style loss 7.63, TV loss 4.31, 0.03 sec
epoch 400, content loss 5.42, style loss 7.48, TV loss 4.32, 0.03 sec
change lr to 1.0e-04
epoch 450, content loss 5.41, style loss 7.47, TV loss 4.32, 0.03 sec
```

Visualize

Train a 300 x 450 composite image

In [19]:
```
image_shape = (450, 300)
content_X, content_Y = get_contents(image_shape, ctx)
style_X, style_Y = get_styles(image_shape, ctx)
X = preprocess(postprocess(output) * 255, image_shape)
output = train(X, content_Y, style_Y, ctx, 0.01, 300, 100)
d2l.plt.imsave('neural-style-2.png', postprocess(output).asnumpy())
```

```
epoch  50, content loss 13.89, style loss 13.71, TV loss 2.38, 0.10 sec
epoch 100, content loss 9.54, style loss 8.79, TV loss 2.65, 0.10 sec
change lr to 1.0e-03
epoch 150, content loss 9.23, style loss 8.46, TV loss 2.68, 0.10 sec
epoch 200, content loss 8.95, style loss 8.17, TV loss 2.70, 0.10 sec
change lr to 1.0e-04
epoch 250, content loss 8.92, style loss 8.14, TV loss 2.70, 0.10 sec
```

Visualize