In [5]:

```
#######################################################################
# Implementation of
# A Sliced Wasserstein Loss for Neural Texture Synthesis
# Heitz et al., CVPR 2021
#######################################################################

import numpy as np
import torch
import imageio
from torchvision import transforms

#######################################################################
# scaling factor of the optimized texture
# wrt the example texture
#######################################################################
SCALING_FACTOR = 1


#######################################################################
# Load example texture
#######################################################################
STYLE_IMAGE_PATH = 'images/Vassily_Kandinsky_1913_-_Composition_7.jpg'
CONTENT_IMAGE_PATH = 'images/YellowLabradorLooking_new.jpg'

def saveImage(filename, image):
    imageTMP = np.clip(image * 255.0, 0, 255).astype('uint8')
    imageio.imwrite("SWD/"+filename, imageTMP)

def loadImage(filename, size=128):
    image = imageio.imread(filename).astype("float32") / 255.0  # Normalize pixel values
    image = image[:, :, :3]  # Ensure image is RGB without alpha channel
    image = np.transpose(image, (2, 0, 1))  # Change dimension order to CxHxW
    image = image[np.newaxis, ...]  # Add a batch dimension
    # Convert to tensor
    image = torch.from_numpy(image)
    # Resize image
    resize = transforms.Resize((size, size))
    image = resize(image)
    return image

# Load style and content images
image_style = loadImage(STYLE_IMAGE_PATH)
image_content = loadImage(CONTENT_IMAGE_PATH)

# Ensure both are on the same device
device = torch.device('cpu')  # Use 'cuda' if you have GPU
image_style = image_style.to(device)
image_content = image_content.to(device)

#######################################################################
# Load pretrained VGG19
#######################################################################

class VGG19(torch.nn.Module):

    def __init__(self):
        super(VGG19, self).__init__()

        self.block1_conv1 = torch.nn.Conv2d(3, 64, (3,3), padding=(1,1), padding_mode='r
eflect')
        self.block1_conv2 = torch.nn.Conv2d(64, 64, (3,3), padding=(1,1), padding_mode='
reflect')

        self.block2_conv1 = torch.nn.Conv2d(64, 128, (3,3), padding=(1,1), padding_mode=
'reflect')
        self.block2_conv2 = torch.nn.Conv2d(128, 128, (3,3), padding=(1,1), padding_mode
='reflect')
```

```
        self.block3_conv1 = torch.nn.Conv2d(128, 256, (3,3), padding=(1,1), padding_mode
='reflect')
        self.block3_conv2 = torch.nn.Conv2d(256, 256, (3,3), padding=(1,1), padding_mode
='reflect')
        self.block3_conv3 = torch.nn.Conv2d(256, 256, (3,3), padding=(1,1), padding_mode
='reflect')
        self.block3_conv4 = torch.nn.Conv2d(256, 256, (3,3), padding=(1,1), padding_mode
='reflect')

        self.block4_conv1 = torch.nn.Conv2d(256, 512, (3,3), padding=(1,1), padding_mode
='reflect')
        self.block4_conv2 = torch.nn.Conv2d(512, 512, (3,3), padding=(1,1), padding_mode
='reflect')
        self.block4_conv3 = torch.nn.Conv2d(512, 512, (3,3), padding=(1,1), padding_mode
='reflect')
        self.block4_conv4 = torch.nn.Conv2d(512, 512, (3,3), padding=(1,1), padding_mode
='reflect')

        self.relu = torch.nn.ReLU(inplace=True)
        self.downsampling = torch.nn.AvgPool2d((2,2))

    def forward(self, image):

        # RGB to BGR
        image = image[:, [2,1,0], :, :]

        # [0, 1] --> [0, 255]
        image = 255 * image

        # remove average color
        image[:,0,:,:] -= 103.939
        image[:,1,:,:] -= 116.779
        image[:,2,:,:] -= 123.68

        # block1
        block1_conv1 = self.relu(self.block1_conv1(image))
        block1_conv2 = self.relu(self.block1_conv2(block1_conv1))
        block1_pool = self.downsampling(block1_conv2)

        # block2
        block2_conv1 = self.relu(self.block2_conv1(block1_pool))
        block2_conv2 = self.relu(self.block2_conv2(block2_conv1))
        block2_pool = self.downsampling(block2_conv2)

        # block3
        block3_conv1 = self.relu(self.block3_conv1(block2_pool))
        block3_conv2 = self.relu(self.block3_conv2(block3_conv1))
        block3_conv3 = self.relu(self.block3_conv3(block3_conv2))
        block3_conv4 = self.relu(self.block3_conv4(block3_conv3))
        block3_pool = self.downsampling(block3_conv4)

        # block4
        block4_conv1 = self.relu(self.block4_conv1(block3_pool))
        block4_conv2 = self.relu(self.block4_conv2(block4_conv1))
        block4_conv3 = self.relu(self.block4_conv3(block4_conv2))
        block4_conv4 = self.relu(self.block4_conv4(block4_conv3))

        return [block1_conv1, block1_conv2, block2_conv1, block2_conv2, block3_conv1, bl
ock3_conv2, block3_conv3, block3_conv4, block4_conv1, block4_conv2, block4_conv3, block4
_conv4]


######################################################################
# Initialize optimized texture
# LBFGS optimization with the slicing loss
######################################################################

def run_texture_synthesis(lambda_content=1.0, max_iterations=20):
    device = torch.device('cpu')
    image_style = loadImage(STYLE_IMAGE_PATH).to(device)
    image_content = loadImage(CONTENT_IMAGE_PATH).to(device)
```

```
    vgg = VGG19().to(device)
    vgg.load_state_dict(torch.load("vgg19.pth", map_location=device))

    image_optimized = torch.nn.Parameter(torch.randn_like(image_style) * 0.01 + image_st
yle.mean(dim=(2, 3), keepdim=True))

    optimizer = torch.optim.LBFGS([image_optimized], lr=1, max_iter=max_iterations, tole
rance_grad=0.0)
    def slicing_loss(image_generated, image_style):

        # generate VGG19 activations
        list_activations_generated = vgg(image_generated)
        list_activations_example   = vgg(image_style)

        # iterate over layers
        loss = 0
        for l in range(len(list_activations_example)):
            # get dimensions
            b = list_activations_example[l].shape[0]
            dim = list_activations_example[l].shape[1]
            n = list_activations_example[l].shape[2]*list_activations_example[l].shape[3
]
            # linearize layer activations and duplicate example activations according to
scaling factor
            activations_example = list_activations_example[l].view(b, dim, n).repeat(1,
1, SCALING_FACTOR*SCALING_FACTOR)
            activations_generated = list_activations_generated[l].view(b, dim, n*SCALING
_FACTOR*SCALING_FACTOR)
            # sample random directions
            Ndirection = dim
            directions = torch.randn(Ndirection, dim).to('cpu')  # After
            directions = directions / torch.sqrt(torch.sum(directions**2, dim=1, keepdim
=True))
            # project activations over random directions
            projected_activations_example = torch.einsum('bdn,md->bmn', activations_exam
ple, directions)
            projected_activations_generated = torch.einsum('bdn,md->bmn', activations_ge
nerated, directions)
            # sort the projections
            sorted_activations_example = torch.sort(projected_activations_example, dim=2
)[0]
            sorted_activations_generated = torch.sort(projected_activations_generated, d
im=2)[0]
            # L2 over sorted lists
            loss += torch.mean( (sorted_activations_example-sorted_activations_generated
)**2 )
        return loss

    def content_loss(image_generated, image_content):
        # generate VGG19 activations for the generated and content images
        activations_generated = vgg(image_generated)
        activations_content = vgg(image_content)

        # Choose layers to compare content
        content_layers = [4]

        loss = 0
        for l in content_layers:
            loss += torch.mean((activations_generated[l] - activations_content[l])**2)
        return loss

    current_loss = {'total_loss': 0, 'style_loss': 0, 'content_loss': 0}  # A dictionary
to hold current loss values

    def closure():
        optimizer.zero_grad()
        style_loss = slicing_loss(image_optimized, image_style)
        cont_loss = content_loss(image_optimized, image_content)
        total_loss = style_loss + lambda_content * cont_loss
        total_loss.backward()
```

```
        # Store losses in the dictionary to access outside the closure
        current_loss['total_loss'] = total_loss.item()
        current_loss['style_loss'] = style_loss.item()
        current_loss['content_loss'] = cont_loss.item()

        return total_loss

    for iteration in range(max_iterations):
        optimizer.step(closure)  # This will execute the closure and update the `current_
loss` dictionary

        # Now you can access and print the loss values stored in `current_loss`
        print(f'Iteration {iteration}: lambda: {lambda_content}, Style Loss: {current_lo
ss["style_loss"]}, Content Loss: {current_loss["content_loss"]}, Total Loss: {current_los
s["total_loss"]}')

        with torch.no_grad():
            img_np = image_optimized.detach().cpu().squeeze(0).permute(1, 2, 0).numpy()
            saveImage(f'optimized_texture_lambda_{lambda_content}_{iteration}.png', img_
np)

    print("Optimization Completed")
```

```
C:\Users\agish\AppData\Local\Temp\ipykernel_14236\365520272.py:30: DeprecationWarning: St
arting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread
. To keep the current behavior (and make this warning disappear) use `import imageio.v2 a
s imageio` or call `imageio.v2.imread` directly.
  image = imageio.imread(filename).astype("float32") / 255.0  # Normalize pixel values
```

In [7]:

```
# Example usage:
run_texture_synthesis(lambda_content=100.0, max_iterations=20)
run_texture_synthesis(lambda_content=10.0, max_iterations=20)
run_texture_synthesis(lambda_content=1.0, max_iterations=20)
```

```
C:\Users\agish\AppData\Local\Temp\ipykernel_14236\365520272.py:30: DeprecationWarning: St
arting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread
. To keep the current behavior (and make this warning disappear) use `import imageio.v2 a
s imageio` or call `imageio.v2.imread` directly.
  image = imageio.imread(filename).astype("float32") / 255.0  # Normalize pixel values
```

```
Iteration 0: lambda: 100.0, Style Loss: 5.27976131439209, Content Loss: 0.053211849182844
16, Total Loss: 10.600946426391602
Iteration 1: lambda: 100.0, Style Loss: 5.181183338165283, Content Loss: 0.01991446129977
703, Total Loss: 7.172629356384277
Iteration 2: lambda: 100.0, Style Loss: 5.299943923950195, Content Loss: 0.01299806125462
0552, Total Loss: 6.59975004196167
Iteration 3: lambda: 100.0, Style Loss: 5.289909362792969, Content Loss: 0.00974303111433
9828, Total Loss: 6.264212608337402
Iteration 4: lambda: 100.0, Style Loss: 5.3857035636901855, Content Loss: 0.0079182637855
41058, Total Loss: 6.177529811859131
Iteration 5: lambda: 100.0, Style Loss: 5.33715295791626, Content Loss: 0.007158598862588
406, Total Loss: 6.053012847900391
Iteration 6: lambda: 100.0, Style Loss: 5.279743671417236, Content Loss: 0.00669075828045
6066, Total Loss: 5.948819637298584
Iteration 7: lambda: 100.0, Style Loss: 5.374063491821289, Content Loss: 0.00661295140162
1103, Total Loss: 6.035358428955078
Iteration 8: lambda: 100.0, Style Loss: 5.210229873657227, Content Loss: 0.00666816998273
1342, Total Loss: 5.877047061920166
Iteration 9: lambda: 100.0, Style Loss: 5.230064868927002, Content Loss: 0.00647206371650
09975, Total Loss: 5.8772711753845215
Iteration 10: lambda: 100.0, Style Loss: 5.247893333435059, Content Loss: 0.0064246295951
30682, Total Loss: 5.890356063842773
Iteration 11: lambda: 100.0, Style Loss: 5.122081756591797, Content Loss: 0.0064046578481
7934, Total Loss: 5.762547492980957
Iteration 12: lambda: 100.0, Style Loss: 5.146628379821777, Content Loss: 0.0064882882870
73374, Total Loss: 5.795457363128662
Iteration 13: lambda: 100.0, Style Loss: 5.33413553237915, Content Loss: 0.00656204577535
39085, Total Loss: 5.990340232849121
Iteration 14: lambda: 100.0, Style Loss: 5.147825241088867, Content Loss: 0.0065876739099
62177, Total Loss: 5.8065924644470215
Iteration 15: lambda: 100.0, Style Loss: 5.168637752532959, Content Loss: 0.0067040450252
```

Iteration 15: lambda: 100.0, Style Loss: 5.168637752532959, Content Loss: 0.006704045925
893925, Total Loss: 5.8390421867370605
Iteration 16: lambda: 100.0, Style Loss: 5.302617073059082, Content Loss: 0.0067851468920
7077, Total Loss: 5.981131553649902
Iteration 17: lambda: 100.0, Style Loss: 5.367295742034912, Content Loss: 0.0067385332658
88691, Total Loss: 6.041149139404297
Iteration 18: lambda: 100.0, Style Loss: 5.195492744445801, Content Loss: 0.0067027360200
88196, Total Loss: 5.865766525268555
Iteration 19: lambda: 100.0, Style Loss: 5.125655174255371, Content Loss: 0.0068208286538
72013, Total Loss: 5.807738304138184
Optimization Completed
Iteration 0: lambda: 10.0, Style Loss: 1.941745400428772, Content Loss: 0.247397094964981
08, Total Loss: 4.415716171264648
Iteration 1: lambda: 10.0, Style Loss: 1.7931164503097534, Content Loss: 0.18040844798088
074, Total Loss: 3.597200870513916
Iteration 2: lambda: 10.0, Style Loss: 1.7249104976654053, Content Loss: 0.16581609845161
438, Total Loss: 3.3830714225769043
Iteration 3: lambda: 10.0, Style Loss: 1.7415781021118164, Content Loss: 0.15575399994850
16, Total Loss: 3.2991180419921875
Iteration 4: lambda: 10.0, Style Loss: 1.7110183238983154, Content Loss: 0.15525174140930
176, Total Loss: 3.263535737991333
Iteration 5: lambda: 10.0, Style Loss: 1.7270162105560303, Content Loss: 0.15460585057735
443, Total Loss: 3.2730746269226074
Iteration 6: lambda: 10.0, Style Loss: 1.7214856147766113, Content Loss: 0.15096312761306
763, Total Loss: 3.231116771697998
Iteration 7: lambda: 10.0, Style Loss: 1.6952719688415527, Content Loss: 0.14916004240512
848, Total Loss: 3.1868724822998047
Iteration 8: lambda: 10.0, Style Loss: 1.7576907873153687, Content Loss: 0.14800395071506
5, Total Loss: 3.2377302646636963
Iteration 9: lambda: 10.0, Style Loss: 1.6570630073547363, Content Loss: 0.14674408733844
757, Total Loss: 3.1245038509368896
Iteration 10: lambda: 10.0, Style Loss: 1.7122598886489868, Content Loss: 0.1466636955738
0676, Total Loss: 3.178896903991699
Iteration 11: lambda: 10.0, Style Loss: 1.7026772499084473, Content Loss: 0.1454511880874
6338, Total Loss: 3.157189130783081
Iteration 12: lambda: 10.0, Style Loss: 1.657796859741211, Content Loss: 0.14459298551082
61, Total Loss: 3.103726863861084
Iteration 13: lambda: 10.0, Style Loss: 1.6894550323486328, Content Loss: 0.1451061069965
3625, Total Loss: 3.1405160427093506
Iteration 14: lambda: 10.0, Style Loss: 1.7368263006210327, Content Loss: 0.1448594480752
945, Total Loss: 3.1854207515716553
Iteration 15: lambda: 10.0, Style Loss: 1.6654022932052612, Content Loss: 0.1443416178226
471, Total Loss: 3.108818531036377
Iteration 16: lambda: 10.0, Style Loss: 1.6555118560791016, Content Loss: 0.1445381492376
3275, Total Loss: 3.100893497467041
Iteration 17: lambda: 10.0, Style Loss: 1.7220063209533691, Content Loss: 0.1427540779113
7695, Total Loss: 3.1495471000671387
Iteration 18: lambda: 10.0, Style Loss: 1.732527256011963, Content Loss: 0.14301526546478
271, Total Loss: 3.16267991065979
Iteration 19: lambda: 10.0, Style Loss: 1.672137975692749, Content Loss: 0.14228072762489
32, Total Loss: 3.094945192337036
Optimization Completed
Iteration 0: lambda: 1.0, Style Loss: 0.41386330127716064, Content Loss: 1.06659686565399
17, Total Loss: 1.4804601669311523
Iteration 1: lambda: 1.0, Style Loss: 0.31046822667121887, Content Loss: 0.77671146392822
27, Total Loss: 1.0871796607971191
Iteration 2: lambda: 1.0, Style Loss: 0.31121203303337097, Content Loss: 0.67398852109909
06, Total Loss: 0.9852005243301392
Iteration 3: lambda: 1.0, Style Loss: 0.31421902775764465, Content Loss: 0.63167637586593
63, Total Loss: 0.9458954334259033
Iteration 4: lambda: 1.0, Style Loss: 0.30325621366500854, Content Loss: 0.61087393760681
15, Total Loss: 0.9141301512718201
Iteration 5: lambda: 1.0, Style Loss: 0.3108363151550293, Content Loss: 0.596941113471984
9, Total Loss: 0.9077774286270142
Iteration 6: lambda: 1.0, Style Loss: 0.3025412857532501, Content Loss: 0.587234556674957
3, Total Loss: 0.8897758722305298
Iteration 7: lambda: 1.0, Style Loss: 0.30151027441024780, Content Loss: 0.581413269042968
8, Total Loss: 0.8829235434532166
Iteration 8: lambda: 1.0, Style Loss: 0.3165889382362366, Content Loss: 0.576135575771331
8, Total Loss: 0.8927245140075684
Iteration 9: lambda: 1.0, Style Loss: 0.315253347158432, Content Loss: 0.5732884407043457
, Total Loss: 0.8885418176651001
Iteration 10: lambda: 1.0, Style Loss: 0.3067370152473449, Content Loss: 0.5712137994384

```
Iteration 10: lambda: 1.0, Style Loss: 0.29673701524734497, Content Loss: 0.571313798427
818, Total Loss: 0.8680508136749268
Iteration 11: lambda: 1.0, Style Loss: 0.2982594966888428, Content Loss: 0.56904262304306
03, Total Loss: 0.8673021197319031
Iteration 12: lambda: 1.0, Style Loss: 0.3050053119659424, Content Loss: 0.56639170646667
48, Total Loss: 0.8713970184326172
Iteration 13: lambda: 1.0, Style Loss: 0.3047199547290802, Content Loss: 0.56458628177642
82, Total Loss: 0.869306206703186
Iteration 14: lambda: 1.0, Style Loss: 0.3065562844276428, Content Loss: 0.56318062543869
02, Total Loss: 0.869736909866333
Iteration 15: lambda: 1.0, Style Loss: 0.31112462282180786, Content Loss: 0.5639705657958
984, Total Loss: 0.8750951886177063
Iteration 16: lambda: 1.0, Style Loss: 0.309491366147995, Content Loss: 0.562542200088501
, Total Loss: 0.8720335960388184
Iteration 17: lambda: 1.0, Style Loss: 0.307576447725296, Content Loss: 0.562174022197723
4, Total Loss: 0.8697504997253418
Iteration 18: lambda: 1.0, Style Loss: 0.3099428117275238, Content Loss: 0.56059885025024
41, Total Loss: 0.8705416917800903
Iteration 19: lambda: 1.0, Style Loss: 0.3068004250526428, Content Loss: 0.55923104286193
85, Total Loss: 0.8660314679145813
Optimization Completed
```

In [8]:

```
run_texture_synthesis(lambda_content=0.5, max_iterations=20)
```

```
C:\Users\agish\AppData\Local\Temp\ipykernel_14236\365520272.py:30: DeprecationWarning: St
arting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread
. To keep the current behavior (and make this warning disappear) use `import imageio.v2 a
s imageio` or call `imageio.v2.imread` directly.
  image = imageio.imread(filename).astype("float32") / 255.0  # Normalize pixel values
```

```
Iteration 0: lambda: 0.5, Style Loss: 0.26880425214767456, Content Loss: 1.47116780281066
9, Total Loss: 1.0043880939483643
Iteration 1: lambda: 0.5, Style Loss: 0.21440470218658447, Content Loss: 1.07951712608337
4, Total Loss: 0.7541632652282715
Iteration 2: lambda: 0.5, Style Loss: 0.20608563721179962, Content Loss: 0.93413931131362
92, Total Loss: 0.6731553077697754
Iteration 3: lambda: 0.5, Style Loss: 0.20681165158748627, Content Loss: 0.85610586404800
42, Total Loss: 0.6348645687103271
Iteration 4: lambda: 0.5, Style Loss: 0.203163281083107, Content Loss: 0.8292345404624939
, Total Loss: 0.6177805662155151
Iteration 5: lambda: 0.5, Style Loss: 0.196353942155838, Content Loss: 0.8037357330322266
, Total Loss: 0.5982217788696289
Iteration 6: lambda: 0.5, Style Loss: 0.1987880915403366, Content Loss: 0.792955636978149
4, Total Loss: 0.5952659249305725
Iteration 7: lambda: 0.5, Style Loss: 0.19665615260601044, Content Loss: 0.77980601787567
14, Total Loss: 0.5865591764450073
Iteration 8: lambda: 0.5, Style Loss: 0.1900598108768463, Content Loss: 0.769750773906707
8, Total Loss: 0.5749351978302002
Iteration 9: lambda: 0.5, Style Loss: 0.19467341899871826, Content Loss: 0.76496344804763
79, Total Loss: 0.5771551132202148
Iteration 10: lambda: 0.5, Style Loss: 0.20064949989318848, Content Loss: 0.7544479370117
188, Total Loss: 0.5778734683990479
Iteration 11: lambda: 0.5, Style Loss: 0.19671915471553802, Content Loss: 0.7498811483383
179, Total Loss: 0.5716597437858582
Iteration 12: lambda: 0.5, Style Loss: 0.1991158127784729, Content Loss: 0.750235080718994
1, Total Loss: 0.57423335313797
Iteration 13: lambda: 0.5, Style Loss: 0.19801244139671326, Content Loss: 0.7461951971054
077, Total Loss: 0.5711100101470947
Iteration 14: lambda: 0.5, Style Loss: 0.1909932643175125, Content Loss: 0.74426174163818
36, Total Loss: 0.5631241202354431
Iteration 15: lambda: 0.5, Style Loss: 0.1927773505449295, Content Loss: 0.73861837387084
96, Total Loss: 0.5620865225791931
Iteration 16: lambda: 0.5, Style Loss: 0.19265824556350708, Content Loss: 0.7386921644210
815, Total Loss: 0.5620043277740479
Iteration 17: lambda: 0.5, Style Loss: 0.19634030759334564, Content Loss: 0.7363210320472
717, Total Loss: 0.5645008087158203
Iteration 18: lambda: 0.5, Style Loss: 0.19500130414962769, Content Loss: 0.7363296747207
642, Total Loss: 0.5631661415100098
Iteration 19: lambda: 0.5, Style Loss: 0.19607394933700562, Content Loss: 0.7308056950569
153, Total Loss: 0.5614768266677856
Optimization Completed
```

Optimization Completed

In [10]:

```
run_texture_synthesis(lambda_content=0.1, max_iterations=20)
```

```
C:\Users\agish\AppData\Local\Temp\ipykernel_14236\365520272.py:30: DeprecationWarning: St
arting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread
. To keep the current behavior (and make this warning disappear) use `import imageio.v2 a
s imageio` or call `imageio.v2.imread` directly.
  image = imageio.imread(filename).astype("float32") / 255.0  # Normalize pixel values
```

```
Iteration 0: lambda: 0.1, Style Loss: 0.214693084359169, Content Loss: 2.0820412635803223
, Total Loss: 0.42289721965789795
Iteration 1: lambda: 0.1, Style Loss: 0.14281392097473145, Content Loss: 1.84021615982055
66, Total Loss: 0.3268355429172516
Iteration 2: lambda: 0.1, Style Loss: 0.12375868111848831, Content Loss: 1.68947756290435
8, Total Loss: 0.2927064299583435
Iteration 3: lambda: 0.1, Style Loss: 0.11649229377508163, Content Loss: 1.58319139480590
82, Total Loss: 0.27481144666671753
Iteration 4: lambda: 0.1, Style Loss: 0.11200540512800217, Content Loss: 1.51156067848205
57, Total Loss: 0.26316148042678833
Iteration 5: lambda: 0.1, Style Loss: 0.11082682013511658, Content Loss: 1.45933663845062
26, Total Loss: 0.25676047801971436
Iteration 6: lambda: 0.1, Style Loss: 0.10654297471046448, Content Loss: 1.42372429370880
13, Total Loss: 0.2489154040813446
Iteration 7: lambda: 0.1, Style Loss: 0.10700571537017822, Content Loss: 1.38753080368042
, Total Loss: 0.2457588016986847
Iteration 8: lambda: 0.1, Style Loss: 0.10520437359809875, Content Loss: 1.37435030937194
82, Total Loss: 0.24263940751552582
Iteration 9: lambda: 0.1, Style Loss: 0.10494611412286758, Content Loss: 1.34877991676330
57, Total Loss: 0.239824116230011
Iteration 10: lambda: 0.1, Style Loss: 0.10351203382015228, Content Loss: 1.3416504859924
316, Total Loss: 0.23767708241939545
Iteration 11: lambda: 0.1, Style Loss: 0.10321284830570221, Content Loss: 1.3297357559204
102, Total Loss: 0.2361864298582077
Iteration 12: lambda: 0.1, Style Loss: 0.10271663963794708, Content Loss: 1.3211748600006
104, Total Loss: 0.23483413457870483
Iteration 13: lambda: 0.1, Style Loss: 0.09994783997535706, Content Loss: 1.3112720251083
374, Total Loss: 0.23107504844665527
Iteration 14: lambda: 0.1, Style Loss: 0.1031724065542221, Content Loss: 1.30057120323181
15, Total Loss: 0.23322953283786774
Iteration 15: lambda: 0.1, Style Loss: 0.10370147228240967, Content Loss: 1.2863115072250
366, Total Loss: 0.23233263194561005
Iteration 16: lambda: 0.1, Style Loss: 0.09993018209934235, Content Loss: 1.2802337408065
796, Total Loss: 0.22795355319976807
Iteration 17: lambda: 0.1, Style Loss: 0.1013152003288269, Content Loss: 1.27232956886291
5, Total Loss: 0.22854815423488617
Iteration 18: lambda: 0.1, Style Loss: 0.10001422464847565, Content Loss: 1.2700164318084
717, Total Loss: 0.22701586782932281
Iteration 19: lambda: 0.1, Style Loss: 0.09968072175979614, Content Loss: 1.2619624137878
418, Total Loss: 0.22587697207927704
Optimization Completed
```

In [11]:

```
run_texture_synthesis(lambda_content=0.01, max_iterations=20)
```

```
C:\Users\agish\AppData\Local\Temp\ipykernel_14236\365520272.py:30: DeprecationWarning: St
arting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread
. To keep the current behavior (and make this warning disappear) use `import imageio.v2 a
s imageio` or call `imageio.v2.imread` directly.
  image = imageio.imread(filename).astype("float32") / 255.0  # Normalize pixel values
```

```
Iteration 0: lambda: 0.01, Style Loss: 0.18532966077327728, Content Loss: 2.2850191593170
166, Total Loss: 0.20817984640598297
Iteration 1: lambda: 0.01, Style Loss: 0.13927492499351501, Content Loss: 2.2192232608795
166, Total Loss: 0.16146716475486755
Iteration 2: lambda: 0.01, Style Loss: 0.11293163150548935, Content Loss: 2.2075960636138
916, Total Loss: 0.1350075900554657
Iteration 3: lambda: 0.01, Style Loss: 0.10766863822937012, Content Loss: 2.2095847129821
777, Total Loss: 0.12976448237895966
Iteration 4: lambda: 0.01, Style Loss: 0.09975562244653702, Content Loss: 2.1932382583618
```

```
164, Total Loss: 0.12168800830841064
Iteration 5: lambda: 0.01, Style Loss: 0.0953354462981224, Content Loss: 2.18598604202270
5, Total Loss: 0.11719530820846558
Iteration 6: lambda: 0.01, Style Loss: 0.09478658437728882, Content Loss: 2.1789073944091
797, Total Loss: 0.11657565832138062
Iteration 7: lambda: 0.01, Style Loss: 0.09295006096363068, Content Loss: 2.1751182079315
186, Total Loss: 0.11470124125480652
Iteration 8: lambda: 0.01, Style Loss: 0.08995184302330017, Content Loss: 2.1666066646575
928, Total Loss: 0.11161790788173676
Iteration 9: lambda: 0.01, Style Loss: 0.08719481527805328, Content Loss: 2.1640236377716
064, Total Loss: 0.10883504897356033
Iteration 10: lambda: 0.01, Style Loss: 0.08696721494197845, Content Loss: 2.160902023315
4297, Total Loss: 0.10857623815536499
Iteration 11: lambda: 0.01, Style Loss: 0.08479148149490356, Content Loss: 2.153447151184
082, Total Loss: 0.10632595419883728
Iteration 12: lambda: 0.01, Style Loss: 0.08407074958086014, Content Loss: 2.155191659927
368, Total Loss: 0.10562266409397125
Iteration 13: lambda: 0.01, Style Loss: 0.08464830368757248, Content Loss: 2.155472040176
3916, Total Loss: 0.10620301961898804
Iteration 14: lambda: 0.01, Style Loss: 0.08392493426799774, Content Loss: 2.155487775802
6123, Total Loss: 0.10547981411218643
Iteration 15: lambda: 0.01, Style Loss: 0.08311319351196289, Content Loss: 2.147831439971
924, Total Loss: 0.1045915037393557
Iteration 16: lambda: 0.01, Style Loss: 0.08237247914075851, Content Loss: 2.146595954895
0195, Total Loss: 0.10383843630552292
Iteration 17: lambda: 0.01, Style Loss: 0.08137483894824982, Content Loss: 2.146344900131
2256, Total Loss: 0.10283828526735306
Iteration 18: lambda: 0.01, Style Loss: 0.0804399847984314, Content Loss: 2.1389291286468
506, Total Loss: 0.10182927548885345
Iteration 19: lambda: 0.01, Style Loss: 0.07976898550987244, Content Loss: 2.142666816711
426, Total Loss: 0.10119565576314926
Optimization Completed
```