Find the best interpolation

April 20, 2020

1 Finding the best way to interpolate

[1]:
```
import d3dshot
import torch
import time
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import torch.nn.functional as F
```

[2]:
```
d = d3dshot.create(capture_output="pytorch_float_gpu")
```

[3]:
```
#high-speed screen capture
d.capture(region=(0, 0, 1920, 1080))
```

[3]: True

[4]:
```
# stop it
d.stop()
```

[4]: True

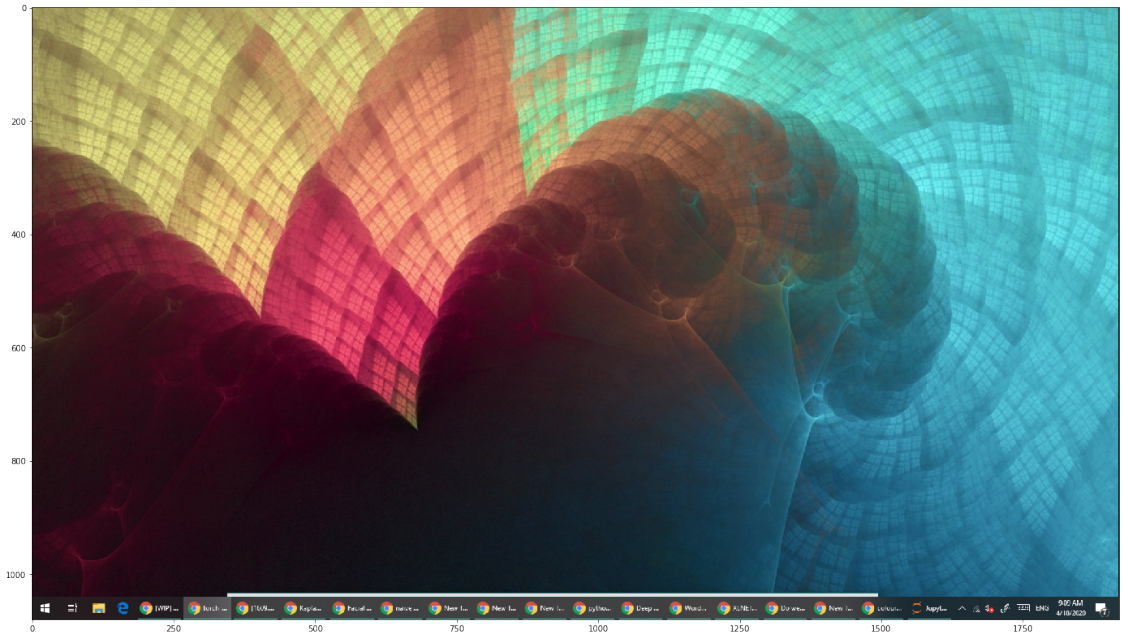
[4]:
```
image = d.get_latest_frame()
```

[5]:
```
def downsample(tensor, scale_factor=0.03, mode='area', align_corners=None):
    """source tensor should be in 3D (H, W, C) tensor"""
    # interpolate accept size of (B, C, H, W) while B is the batch size, only 1
    tensor = tensor.permute(2, 0, 1).unsqueeze(0)
    downsampled = F.interpolate(tensor, scale_factor=scale_factor, mode=mode,␣
 ↪align_corners=align_corners)
    # back to (H, W, C), for imshow right now
    downsampled = downsampled.squeeze(0).permute(1, 2, 0)
    return downsampled
```

[6]:
```
def downupsample(tensor, scale_factor=0.03, mode='area', mode2='nearest',␣
 ↪align_corners=None, align_corners2=None):
    """source tensor should be in 3D (H, W, C) tensor"""
```

```
    # interpolate accept size of (B, C, H, W) while B is the batch size, only 1
    tensor = tensor.permute(2, 0, 1).unsqueeze(0)
    downsampled = F.interpolate(tensor, scale_factor=scale_factor/2, mode=mode,␣
↪align_corners=align_corners)
    upsampled = F.interpolate(downsampled, scale_factor=2, mode=mode2,␣
↪align_corners=align_corners2)
    # back to (H, W, C), for imshow right now
    upsampled = upsampled.squeeze(0).permute(1, 2, 0)
    return upsampled
```

[7]:
```
plt.figure(figsize=(24,16))
plt.imshow(image.cpu())
plt.show()
```



[8]:
```
start_time = time.time()
downsample(image)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 0.0009987354278564453 seconds ---
```

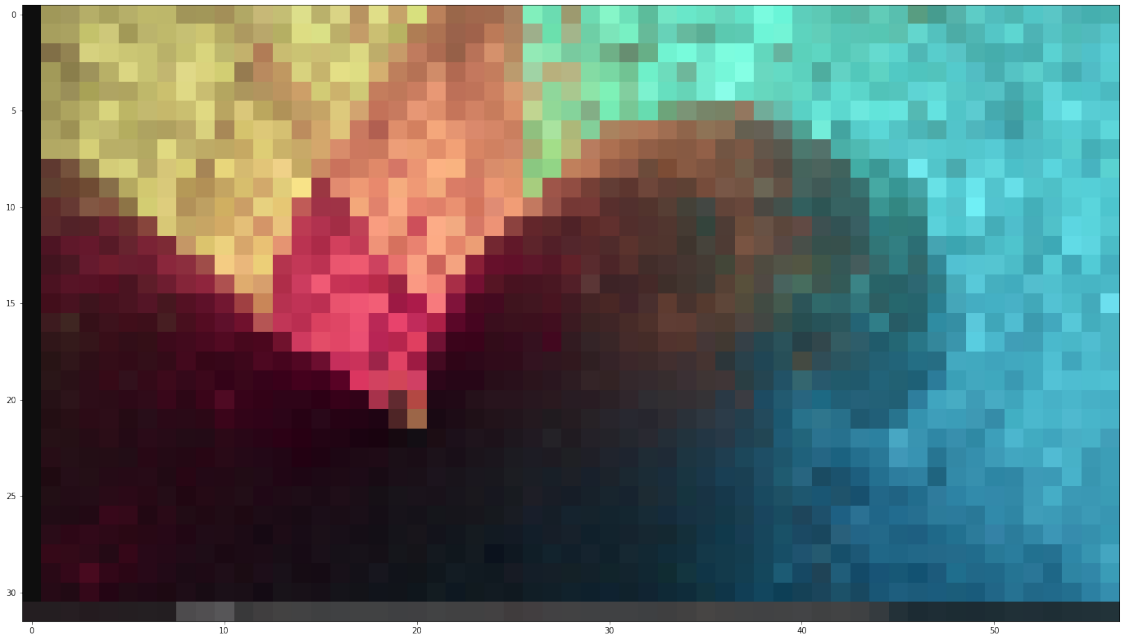
[9]:
```
downsample(image).shape
```

[9]:
```
torch.Size([32, 57, 3])
```

[10]:
```
# area
image_r = downsample(image)
```

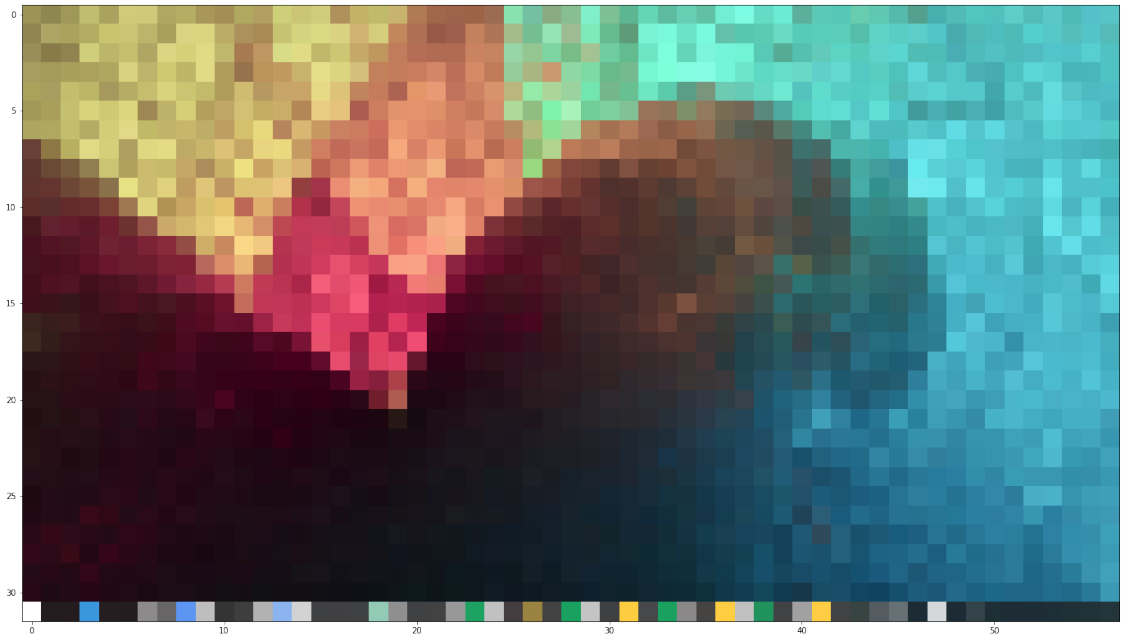
```
plt.figure(figsize=(24,16))
plt.imshow(image_r.cpu())
plt.show()
```

[11]:
```
image_r1 = downsample(image, mode='nearest')
plt.figure(figsize=(24,16))
plt.imshow(image_r1.cpu())
plt.show()
```

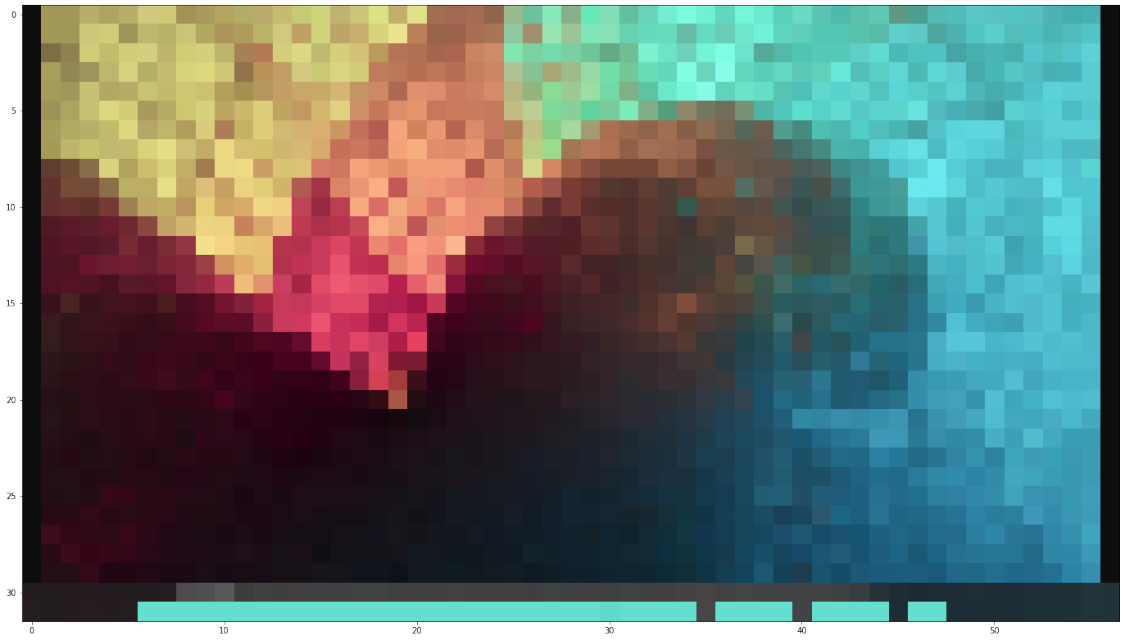


[13]:
```
image_r2 = downsample(image, mode='bilinear')
plt.figure(figsize=(24,16))
plt.imshow(image_r2.cpu())
plt.show()
```

```
C:\Users\King\Anaconda3\lib\site-packages\torch\nn\functional.py:2423:
UserWarning: Default upsampling behavior when mode=bilinear is changed to
align_corners=False since 0.4.0. Please specify align_corners=True if the old
behavior is desired. See the documentation of nn.Upsample for details.
  "See the documentation of nn.Upsample for details.".format(mode))
```

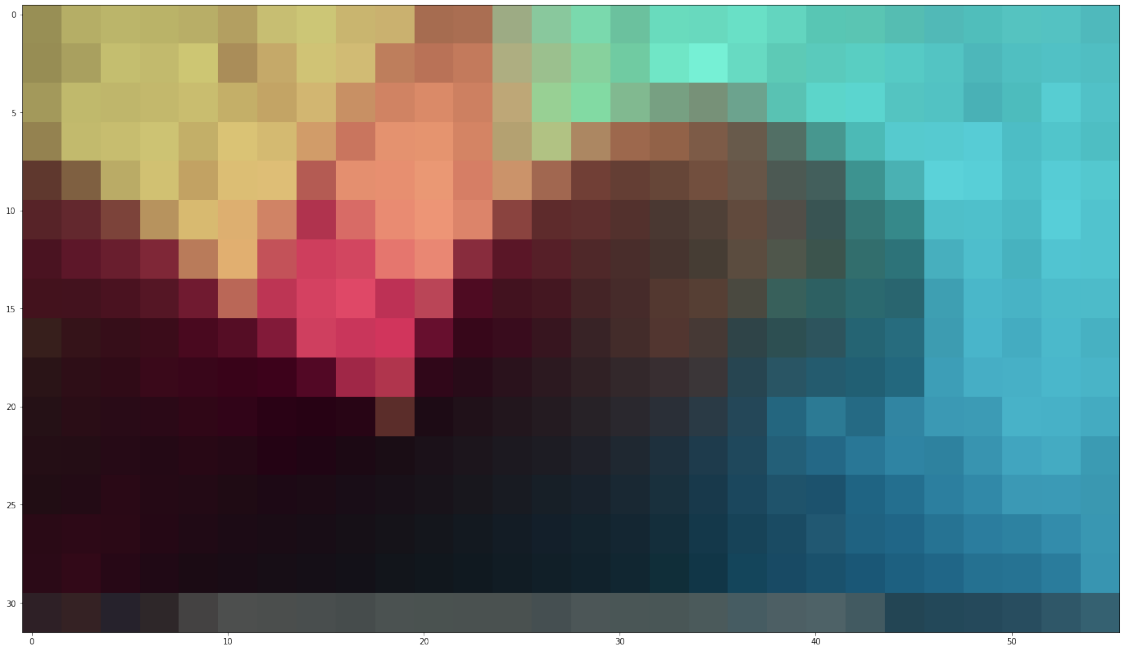


[14]:
```
image_r2 = downsample(image, mode='bilinear', align_corners=True)
plt.figure(figsize=(24,16))
plt.imshow(image_r2.cpu())
plt.show()
```

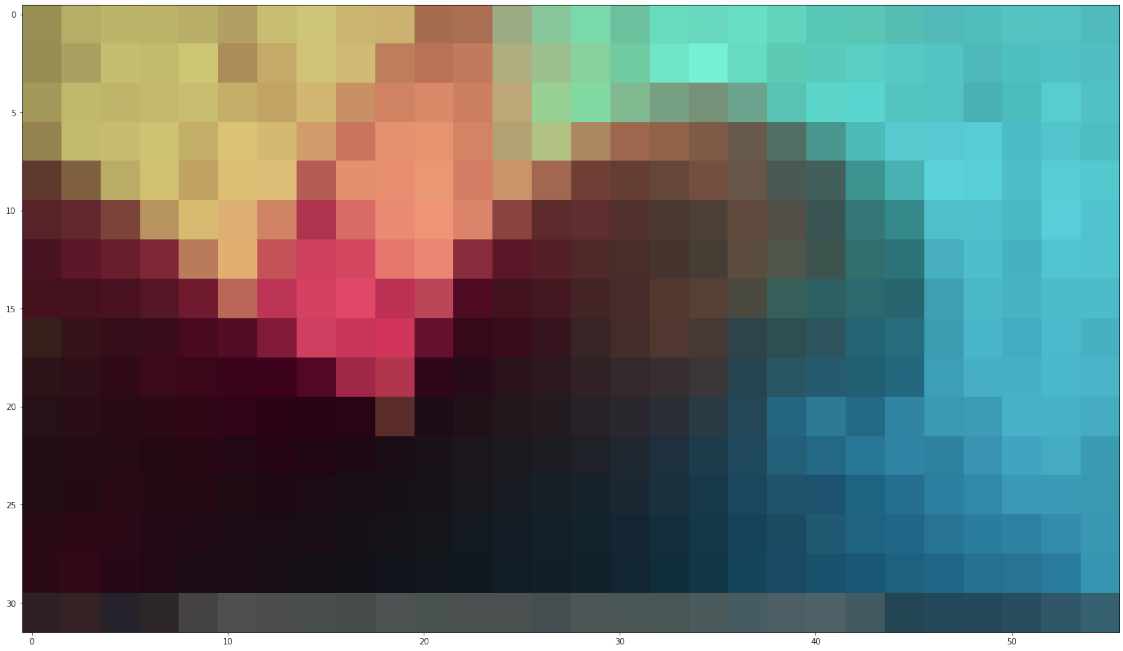


2 Downsample /2 and upsample *2

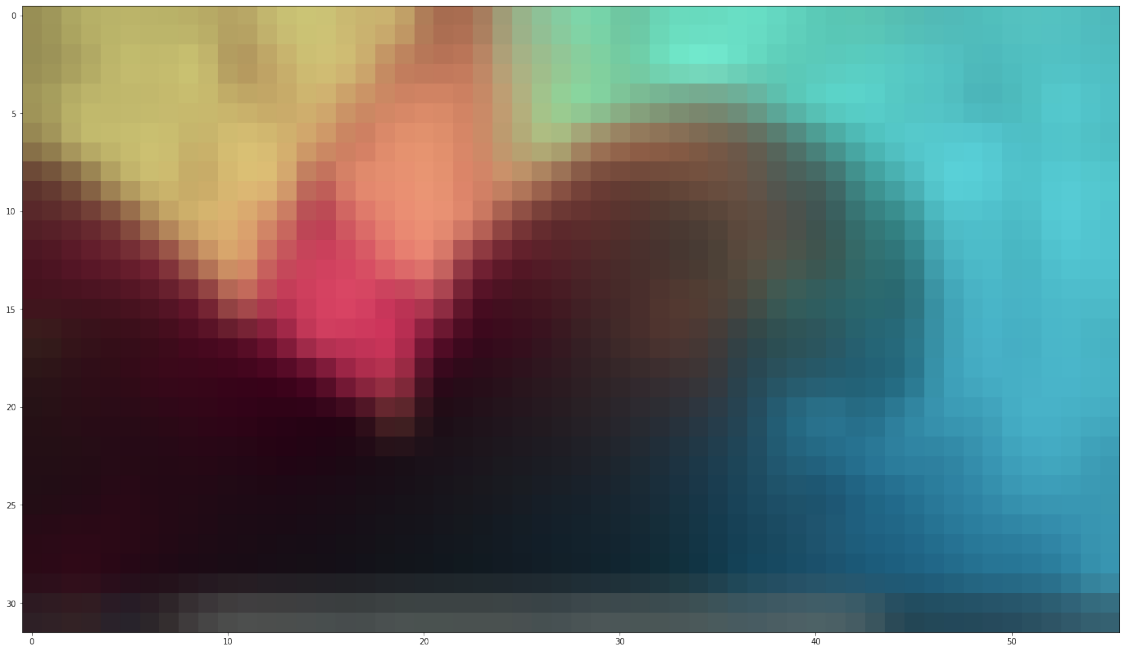
[15]:
```
# default is area
image_ru = downupsample(image)
plt.figure(figsize=(24,16))
plt.imshow(image_ru.cpu())
plt.show()
```



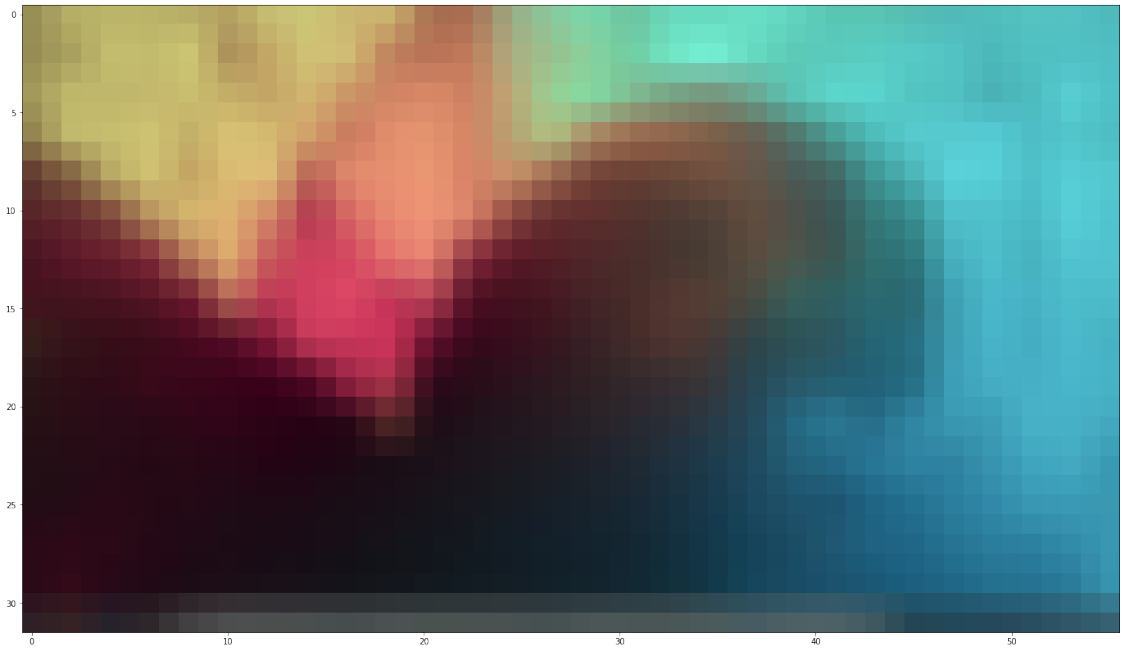
[16]:
```
image_ru1 = downupsample(image, mode2='nearest')
plt.figure(figsize=(24,16))
plt.imshow(image_ru1.cpu())
plt.show()
```



[17]:
```
image_ru2 = downupsample(image, mode2='bilinear')
plt.figure(figsize=(24,16))
plt.imshow(image_ru2.cpu())
plt.show()
```



[18]:
```
image_ru3 = downupsample(image, mode2='bilinear', align_corners2=True)
plt.figure(figsize=(24,16))
plt.imshow(image_ru3.cpu())
plt.show()
```



2.1 bilinear without aligning the corners seems to best

3 Let's invert the image for the left and bottom bar

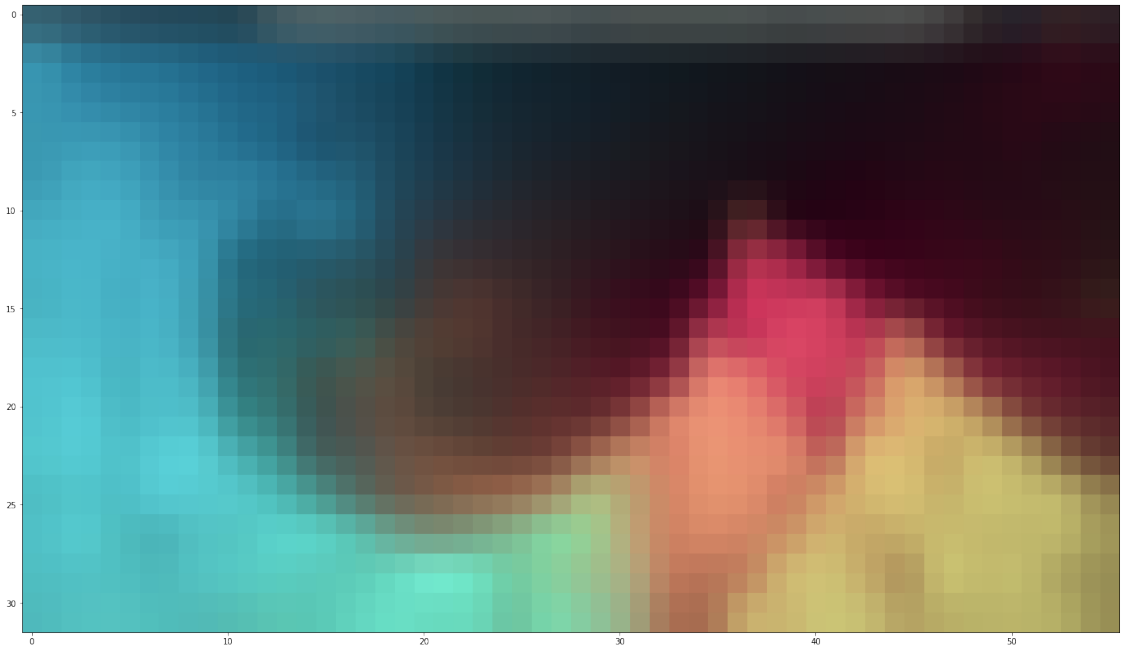
[19]:
```
# bilinear down up sample
image_ru2.shape
```

[19]:
```
torch.Size([32, 56, 3])
```

[20]:
```
image_ru2_F = torch.flip(image_ru2, [0, 1])
image_ru2_F.shape
```

[20]:
```
torch.Size([32, 56, 3])
```

[21]:
```
# Just to confirm
plt.figure(figsize=(24,16))
plt.imshow(image_ru2_F.cpu())
plt.show()
```



4 Select and resize again

[22]:
```
def resize_top_bottom(tensor, size):
    """source tensor should be in 2D (W, C) tensor"""
    # interpolate accept size of (B, C, H, W) while B is the batch size, only 1
    tensor = tensor.unsqueeze(0).permute(2, 0, 1).unsqueeze(0)
    resized = F.interpolate(tensor, size=size, mode='bilinear')
    # back to (H, W, C), for imshow right now
    resized = resized.squeeze(0).permute(1, 2, 0).squeeze(0)
    return resized
```

[24]:
```
top = image_ru2[0, :, :]
top.shape
```

[24]:
```
torch.Size([56, 3])
```

[25]:
```
# unsqueeze(0) = Insert an extra dimension at position 0, as the height -> for␣
 ↪plotting image
plt.figure(figsize=(24,16))
plt.imshow(top.unsqueeze(0).cpu())
plt.show()
```

Top needs 55

[26]:
```
top_r = resize_top_bottom(top, size=(1, 55))
top_r.shape
```

[26]:
```
torch.Size([55, 3])
```

[27]:
```
plt.figure(figsize=(24,16))
plt.imshow(top_r.unsqueeze(0).cpu())
plt.show()
```

[28]:
```
bottom = image_ru2_F[0, :, :]
```

[29]:
```
bottom.shape
```

[29]:
```
torch.Size([56, 3])
```

[31]:
```
plt.figure(figsize=(24,16))
plt.imshow(bottom.unsqueeze(0).cpu())
plt.show()
```

Bottom needs 59

[32]:
```
bottom_r = resize_top_bottom(bottom, size=(1, 59))
bottom_r.shape
```

[32]:
```
torch.Size([59, 3])
```

[33]:
```
plt.figure(figsize=(24,16))
plt.imshow(bottom_r.unsqueeze(0).cpu())
plt.show()
```

[61]:
```
right = image_ru2[1:-1, -1, :] #excluding first and last pixels (corners)
right.shape
```

[61]:
```
torch.Size([30, 3])
```

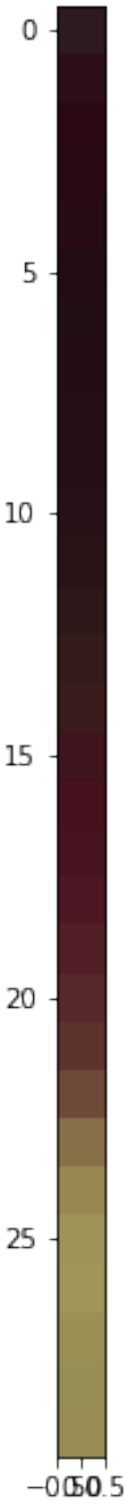
[62]:
```
# unsqueeze(1) = Insert an extra dimension at position 1, as the width -> for␣
↪plotting image
plt.figure(figsize=(24,10))
plt.imshow(right.unsqueeze(1).cpu())
plt.show()
```


0
5
10
15
20
25
−0.5
0.5

[63]:
```
left = image_ru2_F[1:-1, -1, :] #excluding first and last pixels (corners)
left.shape
```

[63]:
```
torch.Size([30, 3])
```

[64]:
```
plt.figure(figsize=(24,10))
plt.imshow(left.unsqueeze(1).cpu())
plt.show()
```


0
5
10
15
20
25
−0.5
0.0
0.5

[65]:
```
data = torch.cat([bottom_r, left, top_r, right], 0)
data.shape
```

[65]:
```
torch.Size([174, 3])
```

[66]:
```
data[:3]
```

[66]:
```
tensor([[0.2113, 0.3840, 0.4458],
        [0.2058, 0.3742, 0.4372],
        [0.1950, 0.3548, 0.4200]], device='cuda:0')
```

[67]:
```
channel_data = torch.flatten(data)
channel_data.shape
```

[67]:
```
torch.Size([522])
```

[68]:
```
channel_data[:9]
```

[68]:
```
tensor([0.2113, 0.3840, 0.4458, 0.2058, 0.3742, 0.4372, 0.1950, 0.3548, 0.4200],
       device='cuda:0')
```

[69]:
```
channel_data_byte = torch.mul(channel_data, 255).to(torch.uint8)
channel_data_byte[:10]
```

[69]:
```
tensor([ 53,  97, 113,  52,  95, 111,  49,  90, 107,  46], device='cuda:0',
       dtype=torch.uint8)
```

[ ]:
```

```