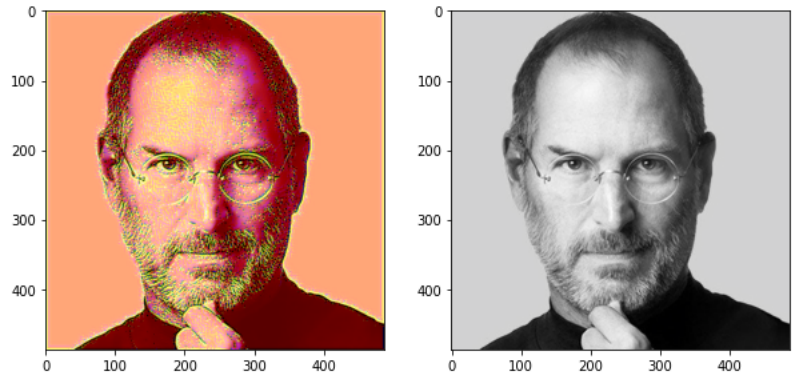
Deep Learning with PyTorch : Neural Style Transfer

Set Google Colab runtime

```
<matplotlib.image.AxesImage at 0x7f41b2acaeb0>
```



```
!pip install torch torchvision
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: torch in /usr/local/lib/python3.8/dist-packages (1.13.1+cu116)
Requirement already satisfied: torchvision in /usr/local/lib/python3.8/dist-packages (0.14.1+cu116)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.8/dist-packages (from torch) (4.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from torchvision) (1.21.6)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.8/dist-packages (from torchvision) (7.1.2)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from torchvision) (2.25.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests->torchvision) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests->torchvision) (2022.12.7)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->torchvision) (2.10)
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->torchvision) (4.0.0)
```

+ Code + Text

Loading VGG Pretrained Model

```
import torch
from torchvision import models
vgg = models.vgg19(pretrained = True)
print(vgg)
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace=True)
    (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (24): ReLU(inplace=True)
    (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
    (26): ReLU(inplace=True)
    (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): ReLU(inplace=True)
    (32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (33): ReLU(inplace=True)
    (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (35): ReLU(inplace=True)
    (36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

```
vgg = vgg.features
print(vgg)
```

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace=True)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace=True)
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace=True)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace=True)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace=True)
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU(inplace=True)
  (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace=True)
  (16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (17): ReLU(inplace=True)
  (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (20): ReLU(inplace=True)
  (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (22): ReLU(inplace=True)
  (23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (24): ReLU(inplace=True)
  (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (26): ReLU(inplace=True)
  (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (29): ReLU(inplace=True)
  (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (31): ReLU(inplace=True)
  (32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (33): ReLU(inplace=True)
  (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (35): ReLU(inplace=True)
  (36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
```

```
for parameters in vgg.parameters() :
  parameters.requires_grad_(False)
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda
```

```
vgg.to(device)
```

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
  (1): ReLU(inplace=True)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace=True)
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace=True)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace=True)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace=True)
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU(inplace=True)
  (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace=True)
  (16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (17): ReLU(inplace=True)
  (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (20): ReLU(inplace=True)
  (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (22): ReLU(inplace=True)
  (23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (24): ReLU(inplace=True)
  (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (26): ReLU(inplace=True)
  (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (29): ReLU(inplace=True)
  (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (31): ReLU(inplace=True)
  (32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (33): ReLU(inplace=True)
  (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (35): ReLU(inplace=True)
  (36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
```

Preprocess image

```
from PIL import Image
from torchvision import transforms as T

def preprocess(img_path , max_size = 500):
  image = Image.open(img_path).convert('RGB')
  if max(image.size) >max_size :
    size = max_size
  else:
    size = max(image.size)
  img_transform = T.Compose([
      T.Resize(size),
      T.ToTensor(),
      T.Normalize(
          mean=[0.485, 0.456, 0.406],
          std = [0.229, 0.224, 0.225]

                 )
  ])
  image = img_transform(image)
  image = image.unsqueeze(0)
  return image




content_p = preprocess("/content/Project-NST/content10.jpg")
style_p = preprocess('/content/Project-NST/style10.jpg')

content_p = content_p.to(device)
style_p = style_p.to(device)

print(content_p.shape)
print(style_p.shape)
```

```
torch.Size([1, 3, 487, 487])
torch.Size([1, 3, 500, 765])
```

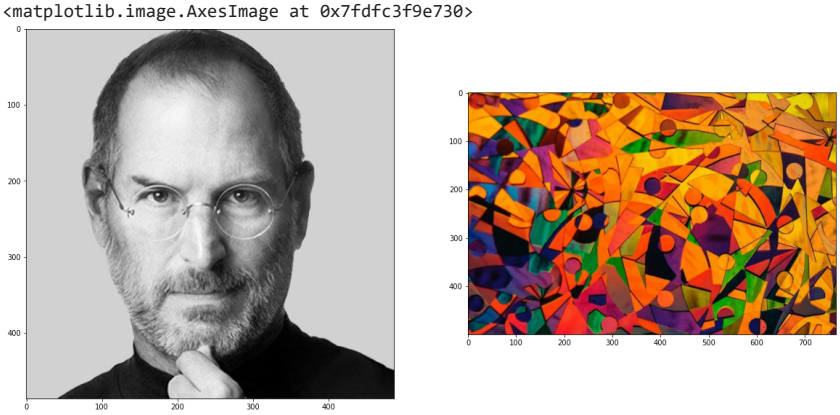
Task 4 : Deprocess image

```
import numpy as np
import matplotlib.pyplot as plt

def deprocess(tensor):
  image = tensor.to('cpu').clone()
  image = image.numpy()
  image = image.squeeze(0)
  image = image.transpose(1,2,0)
  image = image * np.array([0.229, 0.224, 0.225]) + np.array([0.485, 0.456, 0.406])
  image = image.clip(0,1)
  return image


content_d = deprocess(content_p)
style_d = deprocess(style_p)

print(content_d.shape)
print(style_d.shape)
```

```
(487, 487, 3)
(500, 765, 3)
```

```
fig , (ax1,ax2) = plt.subplots(1, 2, figsize=(20,10))
ax1.imshow(content_d)
ax2.imshow(style_d)
```

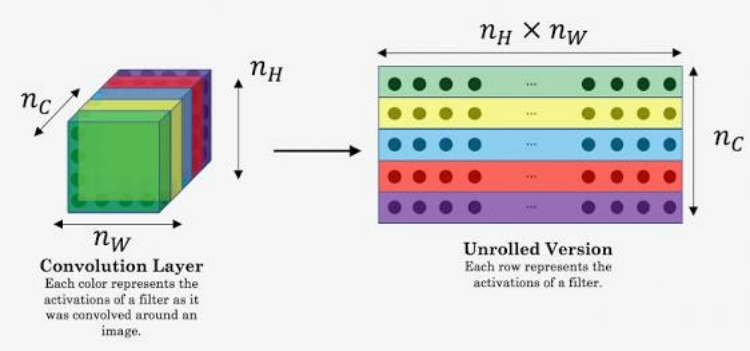
```
<matplotlib.image.AxesImage at 0x7fdfc3f9e730>
```

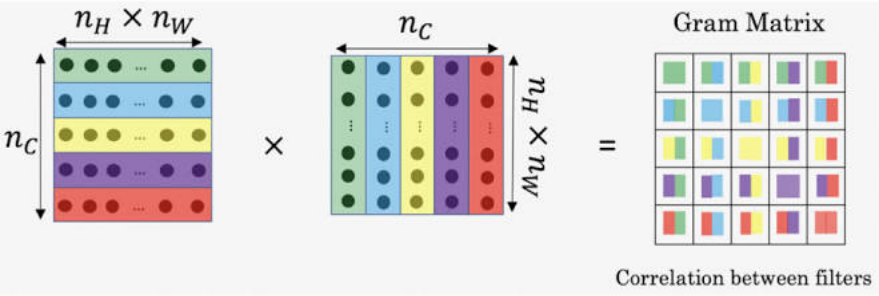


Get content,style features and create gram matrix

```
def get_features(image , model):
  layers ={
      '0':'conv1_1',
      '5':'conv2_1',
      '10':'conv3_1',
      '19': 'conv4_1',
      '21' : 'conv4_2',
      '28' : 'conv5_1'
  }
  x = image
  Features = {}
  for name , layer in model._modules.items():
    x = layer(x)

    if name in layers :
      Features[layers[name]] = x
```

```
  return Features

content_f = get_features(content_p , vgg)
style_f = get_features(style_p , vgg)
```






```
def gram_matrix(tensor):
  b,c,h,w = tensor.size()
  tensor = tensor.view(c,h*w)
  gram = torch.mm(tensor,tensor.t())
  return gram


style_grams = {layer : gram_matrix(style_f[layer]) for layer in style_f}
```

Task 6 : Creating Style and Content loss function

```
def content_loss(target_conv4_2 , content_conv4_2):
  loss = torch.mean((target_conv4_2-content_conv4_2)**2)
  return loss


style_weights = {
    'conv1_1':1.0,
    'conv2_1' : 0.75,
    'conv3_1' : 0.2,
    'conv4_1' : 0.2,
    'conv5_1' : 0.2
}


def style_loss(style_weights, target_features, style_grams):
  loss = 0
  for layer in style_weights:
    target_f = target_features[layer]
    targer_gram = gram_matrix(target_f)
    style_gram = style_grams[layer]
    b,c,h,w = target_f.shape
    layer_loss = style_weights[layer] + torch.mean((targer_gram-style_gram)**2)
    loss += layer_loss / (c*h*w)

    return loss


target = content_p.clone().requires_grad_(True).to(device)
target_f = get_features(target , vgg)
```

```
print("Content loss:", content_loss(target_f['conv4_2'] , content_f['conv4_2']))
print("style loss:" ,style_loss(style_weights, target_f, style_grams) )
```

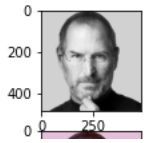
```
Content loss: tensor(0., device='cuda:0', grad_fn=<MeanBackward0>)
style loss: tensor(1126.2665, device='cuda:0', grad_fn=<AddBackward0>)
```

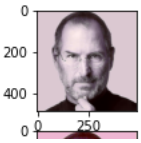
Training loop

```
from torch import optim

optimizer = optim.Adam([target], lr=0.003)
alpha = 1
beta = 1e5
epochs = 500
show_every = 50


def total_loss(c_loss, s_loss , alpha, beta):
  loss = alpha * c_loss + beta * s_loss
  return loss


results = []

for i in range(epochs):
  target_f = get_features(target , vgg)
  c_loss = content_loss(target_f['conv4_2'],content_f['conv4_2'])
  s_loss = style_loss(style_weights, target_f, style_grams)
  t_loss = total_loss(c_loss, s_loss , alpha, beta)

  optimizer.zero_grad()
  t_loss.backward()
  optimizer.step()
  if i % show_every == 0 :
    print("Total loss at epoch {} : {}".format(i,t_loss))
    results.append(deprocess(target.detach()))
```

```
Total loss at epoch 0 : 112626648.0
Total loss at epoch 50 : 102456064.0
Total loss at epoch 100 : 87804720.0
Total loss at epoch 150 : 75324184.0
Total loss at epoch 200 : 64434732.0
Total loss at epoch 250 : 55043984.0
Total loss at epoch 300 : 46937272.0
Total loss at epoch 350 : 37820080.0
Total loss at epoch 400 : 29664820.0
Total loss at epoch 450 : 23506814.0
```

```
plt.figure(figsize=(10,8))
for i in range(len(results)):
  plt.subplot(5,2,i+1)
  plt.imshow(results[i])
plt.show()
```






```
target_copy = deprocess(target.detach())
content_copy = deprocess(content_p)
fig , (ax1,ax2) = plt.subplots(1,2,figsize=(10,5))
ax1.imshow(target_copy)
ax2.imshow(content_copy)
```

```
<matplotlib.image.AxesImage at 0x7fdf3c5ca820>
```