Creating Art with Deep Learning

Sanket Nagrale 185053
Shubhang Bhagat 185060
Abhishek Kapoor 185056

Computer Science and Enginering
NIT Hamirpur
July 17, 2020

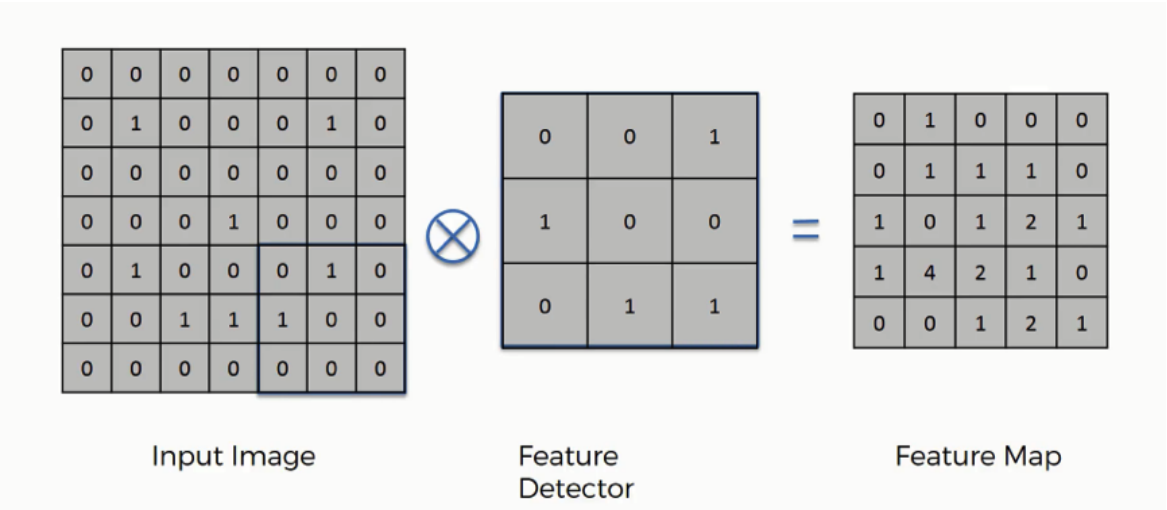
Abstract

Neural Style Transfer

Neural style transfer is an algorithm that combines the content of one image with the style of another image using CNN. Given a content image and a style image, the goal is to generate a target image that minimizes the content difference with the content image and the style difference with the style image. [1]

The system uses neural representations to separate and recombine content and style of arbitrary images, providing a neural algorithm for the creation of artistic images.

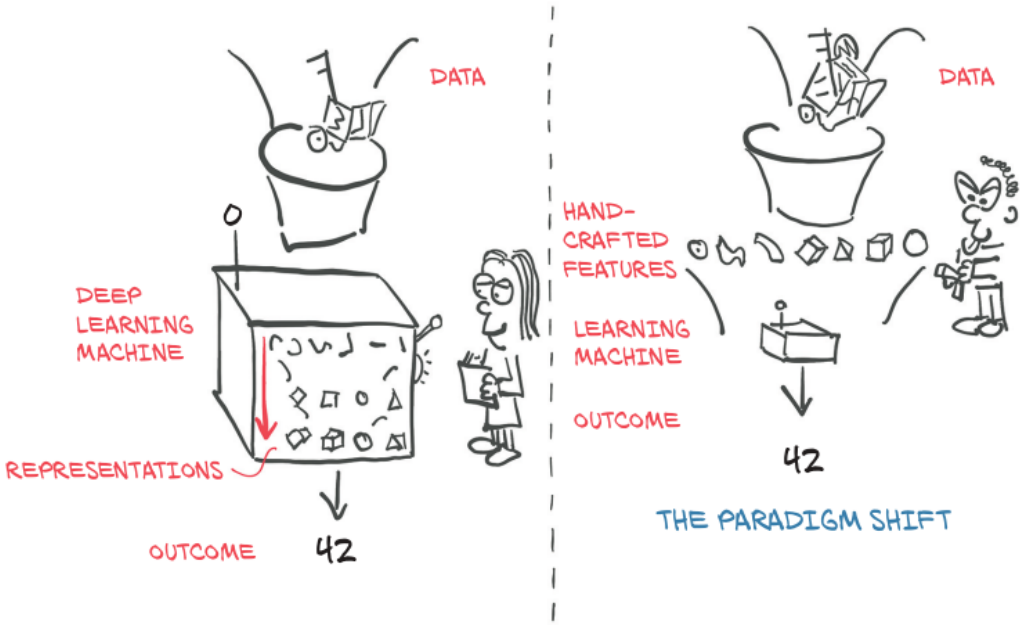
What are Convolutional Neural Networks?



A Convolution Operation

A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image.

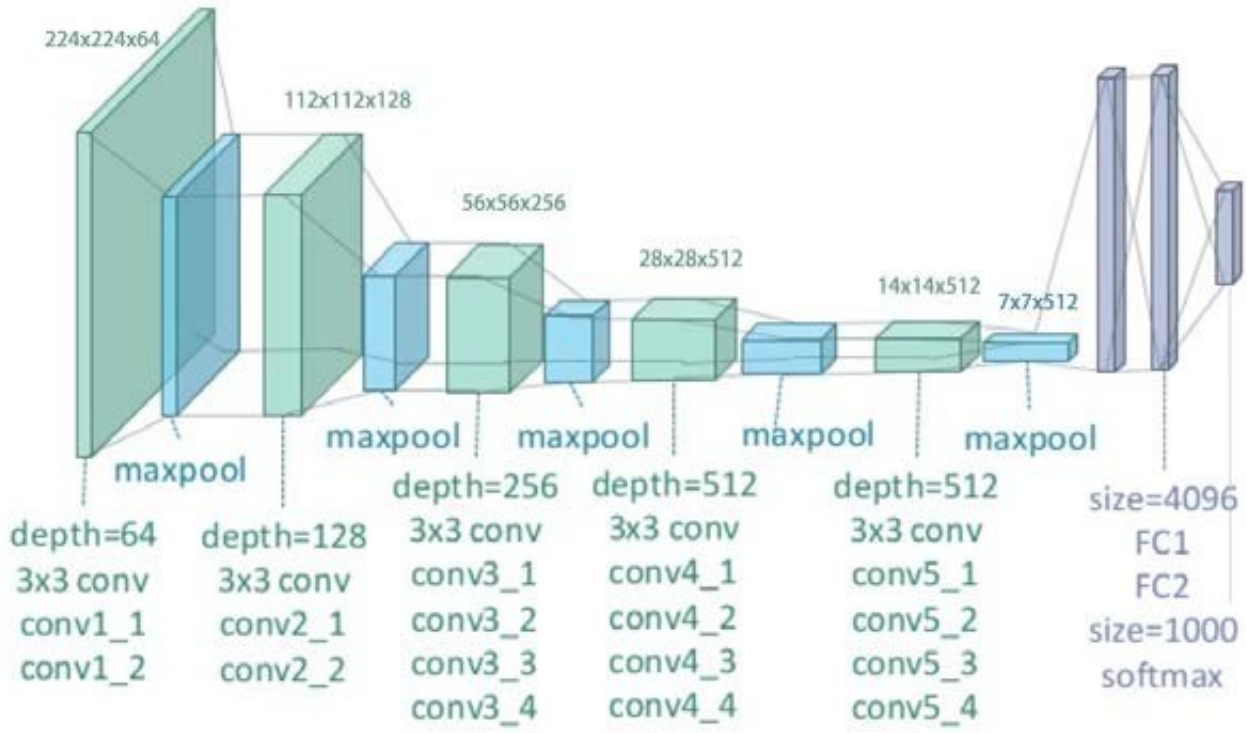
A CNN algorithm can take an input image, extract learnable weights and biases to various objects in image and be able to tell from one another. While the old methods required to be hand-engineered, with enough training, ConvNets learn the filters themselves.



Transfer Learning

One way to quickly get results (and often also get by with much less data) is to start not from random initializations but from a network trained on some task with related data. This is called transfer learning. And we will selectively train some layers accordinig to our needs. This is called *fine-tuning*. Transfer learning has the benefit of decreasing the training time for a neural network model and can result in lower generalization error. For this project, training a model from scratch will turn out to be an arduous task, hence we use pretrained models that come with Pytorch library.

VGGNet



VGGNet19 Architecture

The VGG network architecture was introduced by Simonyan and Zisserman in their 2014 paper, *Very Deep Convolutional Networks for Large Scale Image Recognition.*

This network is characterized by its simplicity, using only 3×3 convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier (above).

Torchvision has a pretrained VGG19 model. Pytorch provides access to a number of top-performing pre-trained models that were developed for image recognition tasks. The first time a pre-trained model is loaded, Pytorch will download the required model weights, which may take some time given the speed of your internet connection.
Note: It is highly recommended to use cuda.

Code Explanation

Loading the Libraries and Installing Requirements

```
from __future__ import division
from torchvision import models
from torchvision import transforms
from PIL import Image
import argparse
import torch
import torchvision
import torch.nn as nn
import numpy as np
```

The requirements to install the libraries are given in requirements.txt file.

Clone the github repository.

```
git clone https://github.com/SankeNrale/Creating-Art-with-Deep-Learning.git
```

Install requirements

```
cd Creating-Art-with-Deep-Learning
pip install -requirements.txt
```

Preprocessing the Image

```
14
15 def load_image(image_path, transform=None, max_size=None, shape=None):
16     """converting image to a torch tensor."""
17     image = Image.open(image_path)
18
19     if max_size:
20         scale = max_size / max(image.size)
21         size = np.array(image.size) * scale
22         image = image.resize(size.astype(int), Image.ANTIALIAS)
23
24     if shape:
25         image = image.resize(shape, Image.LANCZOS)
26
27     if transform:
28         image = transform(image).unsqueeze(0)
29
30     return image.to(device)
31
```

Tensors are similar to NumPy's ndarrays, with the addition being that Tensors can also be used on a GPU to accelerate computing.

Defining The Model

```
32
33 class VGGNet(nn.Module):
34     def __init__(self):
35         """Select conv1_1 ~ conv5_1 activation maps."""
36         super(VGGNet, self).__init__()
37         self.select = ['0', '5', '10', '19', '28']
38         self.vgg = models.vgg19(pretrained=True).features
39
40     def forward(self, x):
41         """Extract multiple convolutional feature maps."""
42         features = []
43         for name, layer in self.vgg._modules.items():
44             x = layer(x)
45             if name in self.select:
46                 features.append(x)
47         return features
48
```

- *self.select* chooses the layers to activate.
- *self.VGG* selects the model VGG and *pretrained=True* returns a pretrained model.

Pytorch has a unique way of defining the model.

In PyTorch, a model is represented by a regular Python class that inherits from the Module class. The most fundamental methods it needs to implement are:

- __init__(self)): it defines the parts that make up the model.
- forward(self, x): it performs the actual computation, that is, it outputs a prediction, given the input x.

Training the Model

```
50 def main(config):
51
52     transform = transforms.Compose([
53         transforms.ToTensor(),
54         transforms.Normalize(mean=(0.485, 0.456, 0.406),
55                              std=(0.229, 0.224, 0.225))])
56
57
58     content = load_image(config.content, transform, max_size=config.max_size)
59     style = load_image(config.style, transform, shape=[content.size(2), content.size(3)])
60
61
62     target = content.clone().requires_grad_(True)
63
64     optimizer = torch.optim.Adam([target], lr=config.lr, betas=[0.5, 0.999])
65     vgg = VGGNet().to(device).eval()
66
```

VGGNet was trained on ImageNet where images are normalized by mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]. We use the same normalization statistics here. Load content and style images with previously created load_image function. Make the style image same size as the content image. We use the adam optimizer here. And we create a taget image using clone().

```
66
67  for step in range(config.total_step):
68
69      # Extract multiple(5) conv feature vectors
70      target_features = vgg(target)
71      content_features = vgg(content)
72      style_features = vgg(style)
73
74      style_loss = 0
75      content_loss = 0
76      for f1, f2, f3 in zip(target_features, content_features, style_features):
77
78          content_loss += torch.mean((f1 - f2)**2)
79
80
81          _, c, h, w = f1.size()
82          f1 = f1.view(c, h * w)
83          f3 = f3.view(c, h * w)
84
85
86          f1 = torch.mm(f1, f1.t())
87          f3 = torch.mm(f3, f3.t())
88
89
90          style_loss += torch.mean((f1 - f3)**2) / (c * h * w)
91
```

view() basically reshapes the image in dimensions(c, h*w) where c is the number of the channels and h and w are height and width respectively. torch.mm computes gram matrix which tells the correlation between the matrices. content loss and style loss are calculated as given in the figure. total loss is just a combination of both losses.

```
92
93      loss = content_loss + config.style_weight * style_loss
94      optimizer.zero_grad()
95      loss.backward()
96      optimizer.step()
97
98      if (step+1) % config.log_step == 0:
99          print ('Step [{}/{}], Content Loss: {:.4f}, Style Loss: {:.4f}'
100                .format(step+1, config.total_step, content_loss.item(), style_loss.item()))
101
102     if (step+1) % config.sample_step == 0:
103         # Save the generated image
104         denorm = transforms.Normalize((-2.12, -2.04, -1.80), (4.37, 4.46, 4.44))
105         img = target.clone().squeeze()
106         img = denorm(img).clamp_(0, 1)
107         torchvision.utils.save_image(img, 'output-{}.png'.format(step+1))
108
```

The above code trains the model for a given number of epochs.

Passing the Arguments

```
112
113
114 if __name__ == "__main__":
115     parser = argparse.ArgumentParser()
116     parser.add_argument('--content', type=str, default='png/content.png')
117     parser.add_argument('--style', type=str, default='png/style.png')
118     parser.add_argument('--max_size', type=int, default=400)
119     parser.add_argument('--total_step', type=int, default=2000)
120     parser.add_argument('--log_step', type=int, default=10)
121     parser.add_argument('--sample_step', type=int, default=500)
122     parser.add_argument('--style_weight', type=float, default=100)
123     parser.add_argument('--lr', type=float, default=0.003)
124     config = parser.parse_args()
125     print(config)
126     main(config)
```

We can run our code using

python main.py —content png/content.png —style ;png/style.p

content.png and style.png should be in the png folder.

Important optional arguments are:

- total_step: Number of Epochs
- style_weight: To change the amount of artistic style
- lr: Learning rate

Results

Bibliography

[1] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, 2015. cite arxiv:1508.06576.

[2] https://towardsdatascience.com/neural-networks- intuitions-2-dot-product-gram-matrix-and-neural-style- transfer-5d39653e7916

[3] https://github.com/eriklindernoren/Fast-Neural-Style-Transfer

[4] Coursera Deep Learning Specialization