Neural Style Transfer

Soo Young Moon
Paper Implementation Project

Paper: L.A.Gatys, A.S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* p2414-2423, 2016.

Motivation

- Paper implementation is important in deep learning because the experience can be helpful to design my own architecture.
- A deep understanding about Convolutional Neural Network (CNN) and PyTorch was expected.
- One of the most fun and exciting applications of CNN recently has been Neural Style Transfer (NST).
- A model part of current tutorial code is slightly implicit [1]. I strive to implement more explicitly so that my code can be used to figure out the model more anatomically.

Introduction

Deep learning vs. Machine learning

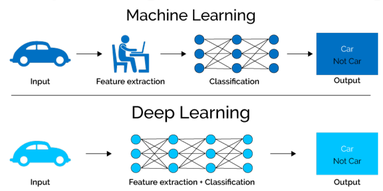

Figure 1: A difference of machine learning and deep learning

When solving a machine learning problem, you follow a specific workflow. You start with an image and then you extract relevant features from it. Then you create a model that describes or predicts the object. On the other hand, with deep learning, you skip the manual step of extracting features from images; instead you feed images directly into the deep learning algorithm. So deep learning is a subtype of machine learning. It deals directly with images and is often more complex.

Convolutional Neural Network

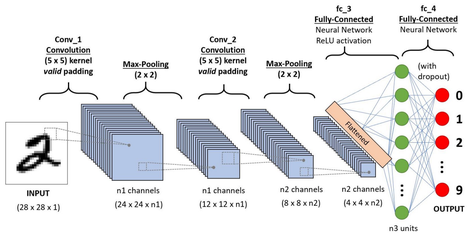

Figure 2: A architecture of a CNN

CNN is a network architecture for deep learning. It learns directly from images. A CNN is made up of several layers that process and transform an input to produce an output. You can train a CNN to do image analysis tasks including scene classification, object detection and segmentation and image processing. A CNN is a network architecture for deep learning which learns directly from images. The CNN can have tens or hundreds of hidden layers that each learns to detect different features in an image. In this feature map, we can see that every hidden layer increases the complexity of the learned image features. For example, the first hidden layer learns how to detect edges and the last learns how to detect more complex shapes. There are three ways to use CNNs for image analysis. The first method is to train the CNN from scratch. The second method relies on transfer learning which is based on the idea that you can use knowledge of one type of problem to solve a similar problem. With the third method, you can use a pre-trained CNN to extract features for training a machine learning model. For example, a hidden layer that has learned how to detect edges in an image is broadly relevant to images from many different domains. I used pre-trained models for this project.

Methodology

Pretrained Model: VGG-19

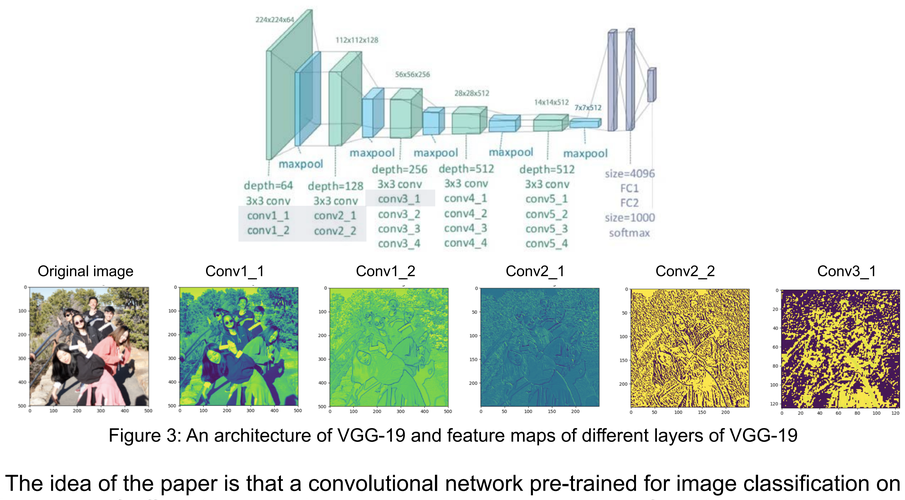

Figure 3: An architecture of VGG-19 and feature maps of different layers of VGG-19

The idea of the paper is that a convolutional network pre-trained for image classification on thousands of different images already knows how to encode the information contained in an image. The pre-trained filters at each layer can detect certain generalized features. There is no information loss in the feature map which is visualized as an output from the lower layers of the network. On the other hand, from the higher layers, lots of information about the actual pixel values or in local image structures seem to be lost. Nevertheless, the features in the upper layers still preserve the information about the general and abstract content of the image, such as the global image. NST model used only up until the fifth convolutional layer based on Gatys' work. This is because NST is not for classification but for transforming my input. Fully connected layers and softmax function, which is excluded for NST, help classify images by squashing the dimensionality of the feature map and outputting a probability.

Neural Style Transfer

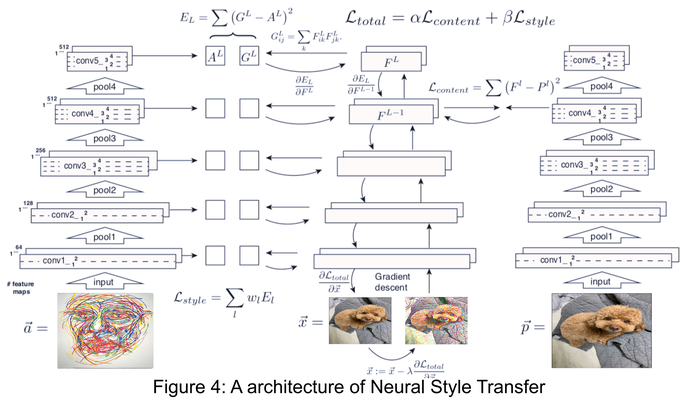

Figure 4: A architecture of Neural Style Transfer

First content and style features are extracted and stored. Then a random white noise image (or a copy of the content image) is passed through the network and its style features (*G*) and content features (*F*) are computed. On each layer included in the style representation (*A*), the element-wise mean squared difference between *G* and *A* is computed to give the style loss *Lstyle*. Also the mean squared difference between *F* and the content representation (*P*) is computed to give the content loss *L*content (right). The total loss *Ltotal* is then a linear combination between the content and the style loss. Its derivative with respect to the pixel values can be computed using error back-propagation (middle). This gradient is used to iteratively update the image (*x*) until it simultaneously matches the style features of the style image (*a*) and the content features of the content image (*p*) (middle, bottom) [2].

Results

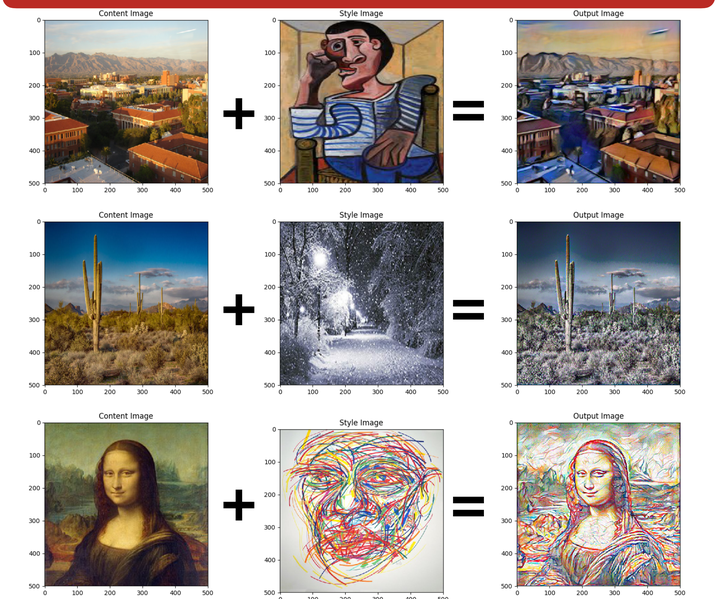

Figure 5: Content (left), style (middle) and output (right) image

Direct Code Comparison

Implementation of the model

```
# desired depth layers to compute style/content losses :
content_layers_default = ['conv_4']
style_layers_default = ['conv_1', 'conv_2', 'conv_3', 'conv_4', 'conv_5']

def get_style_model_and_losses(cnn, normalization_mean, normalization_std,
                               style_img, content_img,
                               content_layers=content_layers_default,
                               style_layers=style_layers_default):
    cnn = copy.deepcopy(cnn)

    # normalization module
    normalization = Normalization(normalization_mean, normalization_std).to(device)

    # just in order to have an iterable access to or list of content/syle
    # losses
    content_losses = []
    style_losses = []

    # assuming that cnn is a nn.Sequential, so we make a new nn.Sequential
    # to put in modules that are supposed to be activated sequentially
    model = nn.Sequential(normalization)

    i = 0  # increment every time we see a conv
    for layer in cnn.children():
        if isinstance(layer, nn.Conv2d):
            i += 1
            name = 'conv_{}'.format(i)
        elif isinstance(layer, nn.ReLU):
            name = 'relu_{}'.format(i)
            layer = nn.ReLU(inplace=False)
        elif isinstance(layer, nn.MaxPool2d):
            name = 'pool_{}'.format(i)
        elif isinstance(layer, nn.BatchNorm2d):
            name = 'bn_{}'.format(i)
        else:
            raise RuntimeError('Unrecognized layer: {}'.format(layer.__class__.__name__))

        model.add_module(name, layer)

        if name in content_layers:
            # add content loss:
            target = model(content_img).detach()
            content_loss = ContentLoss(target)
            model.add_module("content_loss_{}".format(i), content_loss)
            content_losses.append(content_loss)

        if name in style_layers:
            # add style loss:
            target_feature = model(style_img).detach()
            style_loss = StyleLoss(target_feature)
            model.add_module("style_loss_{}".format(i), style_loss)
            style_losses.append(style_loss)

    # now we trim off the layers after the last content and style losses
    for i in range(len(model) - 1, -1, -1):
        if isinstance(model[i], ContentLoss) or isinstance(model[i], StyleLoss):
            break

    model = model[:(i + 1)]

    return model, style_losses, content_losses
```

```
def nst_model(content_img, style_img):
    vgg = models.vgg19(pretrained=True).features.eval()
    normalization = Normalization(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))

    content_img = content_img.detach()
    style_img = style_img.detach()
    content_losses = []
    style_losses = []

    model = nn.Sequential(normalization)
    i = 0
    for name, layer in vgg._modules.items():
        if name in ['0','2','5','10']:
            model.add_module('conv_{}'.format(i),layer)
            style_target = model(style_img)
            style_loss = SL(style_target)
            style_losses.append(style_loss)
            model.add_module('styleloss_{}'.format(i),style_loss)
            i += 1

        elif name in ['7']:
            model.add_module('conv_{}'.format(i),layer)
            content_target = model(content_img)
            content_loss = CL(content_target)
            content_losses.append(content_loss)
            model.add_module('contentloss_{}'.format(i),content_loss)
            style_target = model(style_img)
            style_loss = SL(style_target)
            style_losses.append(style_loss)
            model.add_module('styleloss_{}'.format(i),style_loss)
            i += 1

        elif name in ['1','3','6','8']:
            layer = nn.ReLU(inplace=False)
            model.add_module('relu_{}'.format(i),layer)
            i += 1

        elif name in ['4','9']:
            model.add_module('maxpool_{}'.format(i),layer)
            i += 1

        elif name == '11':
            break

    return model, style_losses, content_losses
```

Source code: https://github.com/symoon94/Neural-Style-Transfer-pytorch

Figure 6: Tutorial code (left) and my code (right)

Implementation for getting images from the user

```
style_img = image_loader("./data/images/neural-style/picasso.jpg")
content_img = image_loader("./data/images/neural-style/dancing.jpg")

output = run_style_transfer(cnn, cnn_normalization_mean, cnn_normalization_std,
                            content_img, style_img, input_img)
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('--content_img', type=str, default = 'images/dancing.jpg')
    parser.add_argument('--style_img', type=str, default = 'images/picasso.jpg')
    parser.add_argument('--size', type=int, default = 128, help='if you want to get more clear pictures, increase the size')
    parser.add_argument('--steps', type=int, default = 300)
    parser.add_argument('--c_weight', type=int, default = 1, help='weighting factor for content reconstruction')
    parser.add_argument('--s_weight', type=int, default = 100000, help='weighting factor for style reconstruction')

    args = parser.parse_args()
    print(args)
    output = main(args)
```

```
Usage

To train a model with images you want to merge:

  $ python train.py --c_weight=1 \
                    --s_weight=100000 \
                    --content_img='images/dancing.jpg' \
                    --style_img='images/picasso.jpg' \
                    --size=128 --steps=300
```

Figure 7: Tutorial code (top) and my code (bottom)

[1] Neural Transfer Using Pytorch. https://pytorch.org/tutorials/advanced/neural_style_tutorial.html
[2] L.A.Gatys, A.S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* p2414-2423, 2016.