Artistic Style Transfer Implementation with Tensorflow

GR5242 Advanced Machine Learning Final Project

Tian Gao

tg2585

tian.gao@columbia.edu

GR5242 Advanced Machine Learning Final Project

October 26, 2017

Contents

List of Figures

List of Tables

1 Introduction

1.1 Project background

1.2 Related works

1.3 Development environment and hardware information

2 Mathematical Foundation

2.1 Covolutional neural network

2.2 VGG-network

2.3 Loss function

3 Tensorflow Implementation

3.1 System overview

The style transfer system is implemented mainly with Python Tensorflow, with auxiliary packages including NumPy, SciPy, and Pillows. A flowchart of the system overview is provided in Fig. 3–1.
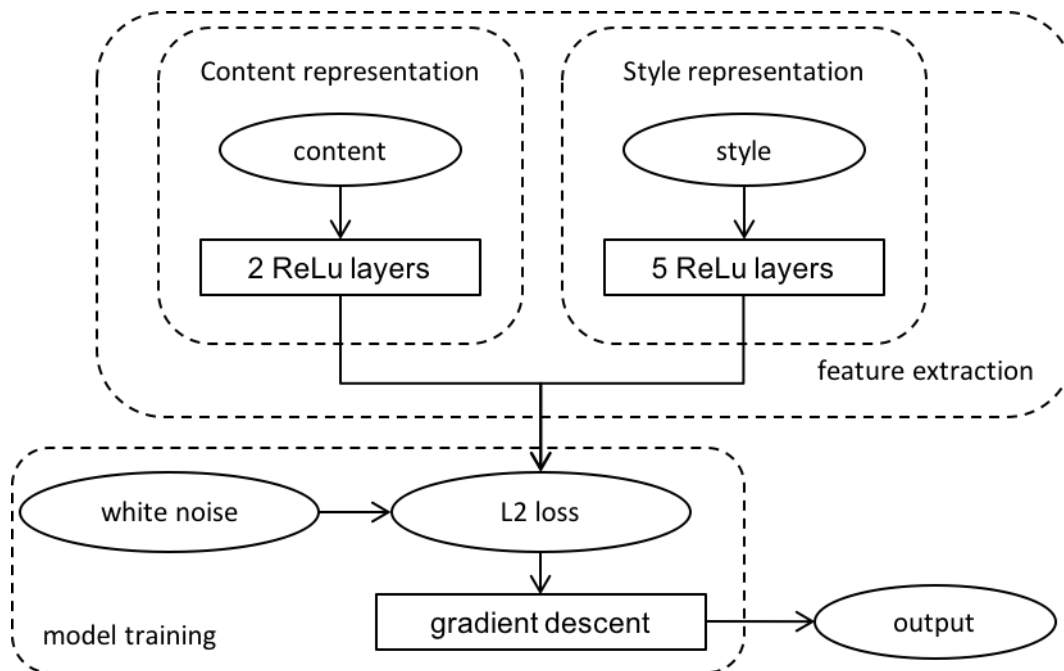

Figure 3–1: Tensorflow implementation system overview

In feature extraction part, the pre-trained VGG network is used for both content representation and style representation. One ReLu layers (relu4_2) is for content feature extraction and five ReLu layers (relu1_1, relu2_1, relu3_1, relu4_1, relu5_1) are for stye representation, which are different from the original work [1].

In model training part, the final output is initialized with a white noise picture, and a gradient-based optimization methodology known as Adam is used. As is described in [2], Adam optimizer is based on first-order gradient. Other optimizers including traditional gradient descent method can also be implemented.

3.2 Pre-defined parameters

Some tuning parameters are hard-wired in the system. Most of the choices follow the original work [1], or the reference [3] and some are modified.

There is plenty of room with cross-validatio optimization for performace improvement (see Sec 5.1 for more details). For simplicity, they are fixed in this system for now and the user can easily modify part of them in `constants.py` following instruction in A.

Some pre-defined parameters are listed below in Table 3–1. The ratio of content loss and style loss is arbitrarily set while [1] explores different loss ratios and corresponding output results. Parameters can always be modified and optimized with techniques like cross-validation.

parameter type	parameter value
content weight	5
style weight	500
total variation weight	100
Adam learning rate	10
Adam β_1	0.9
Adam β_2	0.999
Adam ϵ	10^{-8}
maximum iteration	1000
pooling layer method	max

Table 3–1: Pre-defined model parameters

Apart from model parameters, neural network choice in this work is also different.

		layers used
	this work	relu4_2
content representation	original work	conv1_1, conv4_2
	Anish work	relu4_2, relu5_2 (weight adjustable)
	this work	relu1_1, relu2_1, relu3_1, relu4_1, relu5_1
style representation	original work	conv1_1, conv2_1, conv3_1, conv4_1, conv5_1
	Anish work	relu1_1, relu2_1, relu3_1, relu4_1, relu5_1

Table 3–2: Pre-defined feature extraction layers

Specifically, all layers in style representation are equally weighted. The random initialization, i.e. process to generate the white noise graph is arbitrary as well but has little impact on the output results, and thus is omitted here.

4   Sample Results

4.1   When New York meets Van Gogh

4.2   More examples

5   Discussion

5.1   Tuning parameters optimization

5.2   Running time optimization

A Code Manuscript

Please refer to `https://github.com/tian-gao/AdvML-fall17-project` for detailed instructions. All source code are provided.

B Python Code

B.1 style_transfer.py

```python
import os
import sys
import time
import scipy.misc
from argparse import ArgumentParser
from utils import read_image, save_image

from logger import logger
from settings import PATH_INPUT_STYLE, PATH_INPUT_CONTENT, PATH_OUTPUT, TRAINED_NETWORK_DATA
from constants import (
    CONTENT_WEIGHT, STYLE_WEIGHT, TV_WEIGHT, POOLING,
    LEARNING_RATE, BETA1, BETA2, EPSILON, MAX_ITERATION
)
from visual_geometry_group import VGG
from neural_network import NeuralNetwork


def style_transfer(
        content_name, style_name, output_name, content_weight, style_weight, tv_weight,
        pooling, learning_rate, beta1, beta2, epsilon, max_iteration, check_point):
    time_start = time.time()

    # read images
    content = read_image(PATH_INPUT_CONTENT + content_name)
    style = read_image(PATH_INPUT_STYLE + style_name)
    style = scipy.misc.imresize(style, content.shape[1] / style.shape[1])

    # initialize objects
    vgg = VGG(TRAINED_NETWORK_DATA, pooling)
    nn = NeuralNetwork(content, style, vgg, content_weight, style_weight, tv_weight)

    # train model
    for k, output_image in nn.train_model(learning_rate, beta1, beta2, epsilon,
        max_iteration, check_point):
        name_list = output_name.split('.')
        image_name = PATH_OUTPUT + '.'.join(name_list[:-1]) + '_{}.{}'.format(str(k) if not
            k % check_point else 'final', name_list[-1])
        save_image(output_image, image_name)

    time_end = time.time()
    logger.info('Time elapsed: {} seconds'.format(round(time_end - time_start)))
```

```python
42  def build_parser():
43      parser = ArgumentParser()
44      parser.add_argument('—content', dest='content', required=True,
45                          help='Content image, e.g. "input.jpg"')
46      parser.add_argument('—style', dest='style', required=True,
47                          help='Style image, e.g. "style.jpg"')
48      parser.add_argument('—output', dest='output', required=True,
49                          help='Output image, e.g. "output.jpg"')
50      return parser
51
52
53  if __name__ == '__main__':
54      parser = build_parser()
55      args = parser.parse_args()
56
57      # check if network data file exists
58      if not os.path.isfile(TRAINED_NETWORK_DATA):
59          logger.error('Cannot find pre—trained network data file!')
60          sys.exit()
61
62      style_transfer(
63          content_name=args.content,
64          style_name=args.style,
65          output_name=args.output,
66
67          content_weight=CONTENT_WEIGHT,
68          style_weight=STYLE_WEIGHT,
69          tv_weight=TV_WEIGHT,
70          pooling=POOLING,
71
72          learning_rate=LEARNING_RATE,
73          beta1=BETA1,
74          beta2=BETA2,
75          epsilon=EPSILON,
76          max_iteration=MAX_ITERATION,
77          check_point=MAX_ITERATION / 10
78      )
```

B.2   neural_network.py

```
1   import numpy as np
2   import tensorflow as tf
3   from functools import reduce
4   from operator import mul
5
6   from logger import logger
7   from constants import CONTENT_LAYERS, STYLE_LAYERS
8   from utils import process_image, unprocess_image
9
10
11  class NeuralNetwork(object):
12      """NeuralNetwork provides an interface to formulate the Tensorflow neural network model
13      and perform style transfer algorithm"""
14      def __init__(self, content, style, vgg, content_weight, style_weight, tv_weight):
15          logger.info('Initializing neural network......')
16          self.content = content
17          self.style = style
18          self.vgg = vgg
19
20          self.content_weight = content_weight
21          self.style_weight = style_weight
22          self.tv_weight = tv_weight
23
24          self.content_shape, self.style_shape, self.content_layer_weights, self.
                style_layer_weights = self.get_parameters()
25          self.content_features, self.style_features = self.get_features()
26
27      def get_parameters(self):
28          logger.info('Fetching images parameters......')
29          content_shape = (1, ) + self.content.shape
30          style_shape = (1, ) + self.style.shape
31
32          # get content layer weights
33          content_layer_weights = {}
34          content_layer_weights['relu4_2'] = 1.0
35          content_layer_weights['relu5_2'] = 0.0
36
37          # get style layer weights
38          style_layer_weights = {}
39          for style_layer in STYLE_LAYERS:
40              style_layer_weights[style_layer] = 1.0 / len(STYLE_LAYERS)
41
42          return content_shape, style_shape, content_layer_weights, style_layer_weights
43
44      def get_features(self):
45          content_features = self._get_content_feature()
46          style_features = self._get_style_feature()
```

```
47          return content_features, style_features
48
49      def _get_content_feature(self):
50          logger.info('Fetching content features......')
51          content_features = {}
52          graph = tf.Graph()
53          with graph.as_default(), graph.device('/cpu:0'), tf.Session() as session:
54              content_image = tf.placeholder('float', shape=self.content_shape)
55              content_net = self.vgg.load_net(content_image)
56              content_pre = np.array([
57                  process_image(self.content, self.vgg.mean_pixel)])
58              for content_layer in CONTENT_LAYERS:
59                  content_features[content_layer] = content_net[content_layer].eval(feed_dict
                        ={content_image: content_pre})
60
61          return content_features
62
63      def _get_style_feature(self):
64          logger.info('Fetching style features......')
65          style_features = {}
66          graph = tf.Graph()
67          with graph.as_default(), graph.device('/cpu:0'), tf.Session() as session:
68              style_image = tf.placeholder('float', shape=self.style_shape)
69              style_net = self.vgg.load_net(style_image)
70              style_pre = np.array([
71                  process_image(self.style, self.vgg.mean_pixel)])
72              for style_layer in STYLE_LAYERS:
73                  feature = style_net[style_layer].eval(feed_dict={style_image: style_pre})
74                  feature = np.reshape(feature, (-1, feature.shape[3]))
75                  gram = feature.T.dot(feature) / feature.size
76                  style_features[style_layer] = gram
77
78          return style_features
79
80      def train_model(self, learning_rate, beta1, beta2, epsilon, max_iteration, check_point):
81          with tf.Graph().as_default():
82              # initialize with random guess
83              logger.info('Initializing tensorflow graph with random guess......')
84              noise = np.random.normal(size=self.content_shape, scale=np.std(self.content) *
                    0.1)
85              initial_guess = tf.random_normal(self.content_shape) * 0.256
86              input_image = tf.Variable(initial_guess)
87              parsed_net = self.vgg.load_net(input_image)
88
89              # calculate loss
90              content_loss = self._calculate_content_loss(parsed_net)
91              style_loss = self._calculate_style_loss(parsed_net)
92              tv_loss = self._calculate_tv_loss(input_image)
93              loss = content_loss + style_loss + tv_loss
```

11

```
 94
 95            # summary statistics
 96            tf.summary.scalar('content_loss', content_loss)
 97            tf.summary.scalar('style_loss', style_loss)
 98            tf.summary.scalar('tv_loss', tv_loss)
 99            tf.summary.scalar('total_loss', loss)
100            summary_loss = tf.summary.merge_all()
101
102            # initialize optimization
103            train_step = tf.train.AdamOptimizer(learning_rate, beta1, beta2, epsilon).
                   minimize(loss)
104
105            with tf.Session() as session:
106                summary_writer = tf.summary.FileWriter('logs/neural_network', session.graph)
107                logger.info('Saving graph......')
108
109                session.run(tf.global_variables_initializer())
110                logger.info('Initializing optimization......')
111                logger.info('Current total loss: {}'.format(loss.eval()))
112
113                for k in range(max_iteration):
114                    logger.info('Iteration {} total loss {}'.format(str(k+1), loss.eval()))
115                    train_step.run()
116                    summary = session.run(summary_loss)
117                    summary_writer.add_summary(summary, k)
118
119                    # save intermediate images at checkpoints
120                    if (check_point and (not k % check_point)) or k == max_iteration - 1:
121                        output_temp = input_image.eval()
122                        output_image = unprocess_image(output_temp.reshape(self.
                            content_shape[1:]), self.vgg.mean_pixel)
123                        yield k, output_image
124
125    def _calculate_content_loss(self, parsed_net):
126        logger.info('Calculating content loss......')
127        losses = []
128        for content_layer in CONTENT_LAYERS:
129            losses += [
130                self.content_layer_weights[content_layer] * self.content_weight * (
131                    2 * tf.nn.l2_loss(
132                        parsed_net[content_layer] - self.content_features[content_layer]
133                    ) / self.content_features[content_layer].size)]
134        return reduce(tf.add, losses)
135
136    def _calculate_style_loss(self, parsed_net):
137        logger.info('Calculating style loss......')
138        losses = []
139        for style_layer in STYLE_LAYERS:
140            layer = parsed_net[style_layer]
```

```
141            _, height, width, number = map(lambda x: x.value, layer.get_shape())
142            size = height * width * number
143            feats = tf.reshape(layer, (-1, number))
144            gram = tf.matmul(tf.transpose(feats), feats) / size
145            style_gram = self.style_features[style_layer]
146            losses += [
147                self.style_layer_weights[style_layer] * 2 * tf.nn.l2_loss(gram - style_gram)
                        / style_gram.size]
148        return self.style_weight * reduce(tf.add, losses)
149
150    def _calculate_tv_loss(self, image):
151        # total variation denoising
152        logger.info('Calculating total variation loss......')
153        tv_y_size = self._get_tensor_size(image[:, 1:, :, :])
154        tv_x_size = self._get_tensor_size(image[:, :, 1:, :])
155        tv_loss = self.tv_weight * 2 * ((
156            tf.nn.l2_loss(image[:, 1:, :, :] - image[:, :self.content_shape[1]-1, :, :]) /
                    tv_y_size) + (
157            tf.nn.l2_loss(image[:, :, 1:, :] - image[:, :, :self.content_shape[2]-1, :]) /
                    tv_x_size))
158        return tv_loss
159
160    def _get_tensor_size(self, tensor):
161        return reduce(mul, (d.value for d in tensor.get_shape()), 1)
```

B.3 visual_geometry_group.py

```python
import numpy as np
import scipy.io
import tensorflow as tf

from logger import logger
from constants import VGG19_LAYERS


class VGG(object):
    """VGG provides an interface to extract parameter from pre-trained neural network
    and formulate Tensorflow layers"""
    def __init__(self, trained, pooling):
        logger.info('Loading pre-trained network data......')
        self.network = scipy.io.loadmat(trained)
        self.layers, self.mean_pixel = self.init_net()
        self.pooling = pooling

    def init_net(self):
        mean_mat = self.network['normalization'][0][0][0]  # shape: (224, 224, 3)
        mean_pixel = np.mean(mean_mat, axis=(0, 1))  # length: 3
        layers = self.network['layers'].reshape(-1)  # length: 43
        return layers, mean_pixel

    def load_net(self, input_image):
        # construct layers using parameters
        logger.info('Parsing layers......')
        parsed_net = {}
        current_image = input_image

        for layer_name, input_layer in zip(VGG19_LAYERS, self.layers):
            layer_kind = layer_name[:4]

            if layer_kind == 'conv':
                current_image = self._get_conv_layer(current_image, input_layer)
            elif layer_kind == 'relu':
                current_image = self._get_relu_layer(current_image)
            elif layer_kind == 'pool':
                current_image = self._get_pool_layer(current_image)
            parsed_net[layer_name] = current_image

        assert len(parsed_net) == len(VGG19_LAYERS)
        return parsed_net

    def _get_conv_layer(self, input_image, input_layer):
        # get kernel and bias
        # matconvnet: weights are [width, height, in_channels, out_channels]
        # tensorflow: weights are [height, width, in_channels, out_channels]
```

```python
48        kernels, bias = input_layer[0][0][0][0]
49        kernels = np.transpose(kernels, (1, 0, 2, 3))
50        bias = bias.reshape(-1)
51
52        # formulate conv layer
53        conv = tf.nn.conv2d(input_image, tf.constant(kernels), strides=(1, 1, 1, 1), padding
              ='SAME')
54        layer = tf.nn.bias_add(conv, bias)
55        return layer
56
57    def _get_relu_layer(self, input_image):
58        return tf.nn.relu(input_image)
59
60    def _get_pool_layer(self, input_image):
61        if self.pooling == 'avg':
62            layer = tf.nn.avg_pool(input_image, ksize=(1, 2, 2, 1), strides=(1, 2, 2, 1),
                  padding='SAME')
63        elif self.pooling == 'max':
64            layer = tf.nn.max_pool(input_image, ksize=(1, 2, 2, 1), strides=(1, 2, 2, 1),
                  padding='SAME')
65        return layer
```

References

[1] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR*, pages 2414–2423. IEEE, 2016.

[2] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] Anish Athalye. Neural style. `https://github.com/anishathalye/neural-style`, 2015.