|  |
| --- |
| Class FastStyleTransfer: |
|  | CONTENT\_LAYER = 'relu3\_3' |
|  | STYLE\_LAYERS = ('relu1\_2', 'relu2\_2', 'relu3\_3', 'relu4\_3') |
|  |  |
|  | def \_\_init\_\_(self, vgg\_path, |
|  | style\_image, content\_shape, content\_weight, |
|  | style\_weight, tv\_weight, batch\_size, device): |
|  | with tf.device(device): |
|  | vgg = vgg\_network.VGG(vgg\_path) |
|  | self.style\_image = style\_image |
|  | self.batch\_size = batch\_size |
|  | self.batch\_shape = (batch\_size,) + content\_shape |
|  |  |
|  | self.input\_batch = tf.placeholder(tf.float32, |
|  | shape=self.batch\_shape, |
|  | name="input\_batch") |
|  |  |
|  | self.stylized\_image = transform.net(self.input\_batch) |
|  |  |
|  | loss\_calculator = LossCalculator(vgg, self.stylized\_image) |
|  |  |
|  | self.content\_loss = loss\_calculator.content\_loss( |
|  | self.input\_batch, |
|  | self.CONTENT\_LAYER, |
|  | content\_weight) / self.batch\_size |
|  |  |
|  | self.style\_loss = loss\_calculator.style\_loss( |
|  | self.style\_image, |
|  | self.STYLE\_LAYERS, |
|  | style\_weight) / self.batch\_size |
|  |  |
|  | self.total\_variation\_loss = loss\_calculator.tv\_loss( |
|  | self.stylized\_image, |
|  | tv\_weight) / batch\_size |
|  |  |
|  | self.loss = self.content\_loss + self.style\_loss + self.total\_variation\_loss |
|  |  |
|  |  |
|  | def \_current\_loss(self, feed\_dict): |
|  | losses = {} |
|  | losses['content'] = self.content\_loss.eval(feed\_dict=feed\_dict) |
|  | losses['style'] = self.style\_loss.eval(feed\_dict=feed\_dict) |
|  | losses['total\_variation'] = self.total\_variation\_loss.eval(feed\_dict=feed\_dict) |
|  | losses['total'] = self.loss.eval(feed\_dict=feed\_dict) |
|  | return losses |
|  |  |
|  | def train(self, content\_training\_images,learning\_rate, |
|  | epochs, checkpoint\_iterations): |
|  |  |
|  | def is\_checkpoint\_iteration(i): |
|  | return (checkpoint\_iterations and i % checkpoint\_iterations == 0) |
|  |  |
|  | def print\_progress(i): |
|  | stdout.write('Iteration %d\n' % (i + 1)) |
|  |  |
|  | train\_step = tf.train.AdamOptimizer(learning\_rate).minimize(self.loss) |
|  |  |
|  | with tf.Session() as sess: |
|  | sess.run(tf.global\_variables\_initializer()) |
|  | iterations = 0 |
|  | for epoch in range(epochs): |
|  | for i in range(0, len(content\_training\_images), self.batch\_size): |
|  | print\_progress(iterations) |
|  |  |
|  | batch = self.\_load\_batch(content\_training\_images[i: i+self.batch\_size]) |
|  |  |
|  | train\_step.run(feed\_dict={self.input\_batch:batch}) |
|  |  |
|  | if is\_checkpoint\_iteration(iterations): |
|  | yield ( |
|  | iterations, |
|  | sess, |
|  | self.stylized\_image.eval(feed\_dict={self.input\_batch:batch})[0], |
|  | self.\_current\_loss({self.input\_batch:batch}) |
|  | ) |
|  | iterations += 1 |
|  |  |
|  | def \_load\_batch(self, image\_paths): |
|  | return np.array([utils.load\_image(img\_path) for j, img\_path in enumerate(image\_paths)]) |
|  |  |
|  |  |
|  |  |
|  | def \_tensor\_size(tensor): |
|  | from operator import mul |
|  | return reduce(mul, (d.value for d in tensor.get\_shape()), 1) |

FastTransfer 함수를 보고 import하여 활용