Report for Scene Classification Task

盛凯枫*

2018 年 4 月 23 日

Abstract

A deep residual convolutional neural network, which has been proved efficient in numerous classification and detection benchmarks, has been used as the core of the classifier in current solving of scene classification task. The code was written from scratch in Python and with the help of Tensorflow modula. After adopting some broadly used techniques, i.e. data augmentation, batch normalization, regularization, uniform std initialization, and with a proper choose of the hyperparameters, including the number of layers, in regard to a balance between accuracy and training time, a final top-3 accuracy of 75 percents is achieved on a validation set with 1,000 samples.

Method

network structure

A 34-layer residual network described in *Kaiming He, etc*[1] is implemented with some minor changes. The shortcuts between layers of different dimensions work as follow: the output from the preceding layer with double size and half filters goes through a max pooling layer with a stride of 2 and then is concatenated with zero padding entries to match the filter number of the following layer. The output from the last 3×3 convolutional layer is transformed into a feature map by a 1×1 convolutional layer, whose filter number being equal to 80, the number of classes in question. A global average pooling is then performed on the feature map, yielding the classification result after a final softmax layer. The specific structure is shown in the code *network structure*.

batch normalization

In order to reduce the covariance shift between batches, a batch normalization layer is inserted before each convolutional layer[2]. The normalization is taken between batches for each channel of each pixel, i.e. a pair of normalization parameter is learned for each value of the input picture vector. The specific procedure is shown in the code *batch normalization*.

regularization; global average pooling

A huge overfitting was found in my first prototype of the network model, which has a fully connected layer at the end of the data passageway. To cope with this problem, a global average pooling layer is added at the end, right after the final 1×1 convolutional layer, from which a feature map is produced for each class, in change of the fully connected layer. Besides, a weight regularization cost is added to the *loss function*, which has always been a efficient way of reducing overfitting as we know it. The weight is stored in a weight array at the time it is initialized, and the coefficient is chosen to be 0.01 after some

*Peking University, School of Physic. Email:1500011404@pku.edu.cn

experiments. Overfitting is largely reduced after the adoption of the above two techniques, as shown in the training curve1.

weight variable initialization

To prevent learning rate from decaying exponentially as the network go deeper, all convolutional weight variables should be initialized in the way that the standard deviation being equal to 2(considering the relu layer after each convolution)[3].

data augmentation

All pictures are feeded to the network after being resized to 256×256 pixels. Data augmentation includes horizontally reversing and clipping. Details are shown in the code *data augmentation*.

training procedure

The training procedure take places on a NVIDIA GeForce-1070 GPU with the Tensorflow-gpu platform. In some early experiments, it is found that the time cost of data loading and processing is actually larger that the training procedure, so a multi-thread feature is invoked to load the data with the CPU in the same time of the training, as shown in the code.

Result

Refer to Figure 1.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition" arXiv:1512.03385 [cs.CV], 10 Dec 2015.

[2] Sergey Ioffe, Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" arXiv:1502.03167v3 [cs.LG], 2 Mar 2015.

[3] Dmytro Mishkin, Jiri Matas, "ALL YOU NEED IS A GOOD INIT" arXiv:1511.06422v7 [cs.LG], 19 Feb 2016.

---

network structure

```
def resnet_34(x):
    h_conv1_input = max_pool(tf.nn.relu(conv2d(batch_norm(x), 3, 64, size=7, stride
        =2)))
    h_conv1_output = make_layers(h_conv1_input, 6, enlarge=False)
    h_conv2_output = make_layers(h_conv1_output, 8)
    h_conv3_output = make_layers(h_conv2_output, 12)
    h_conv4_output = make_layers(h_conv3_output, 6)
    feature_map = conv2d(batch_norm(tf.nn.relu(h_conv4_output)), 512, 80, size=1)
    global_pool = tf.reduce_mean(feature_map, [1, 2])
    y = tf.nn.softmax(tf.reshape(global_pool, [-1, 80]))
    return y
```

batch normalization

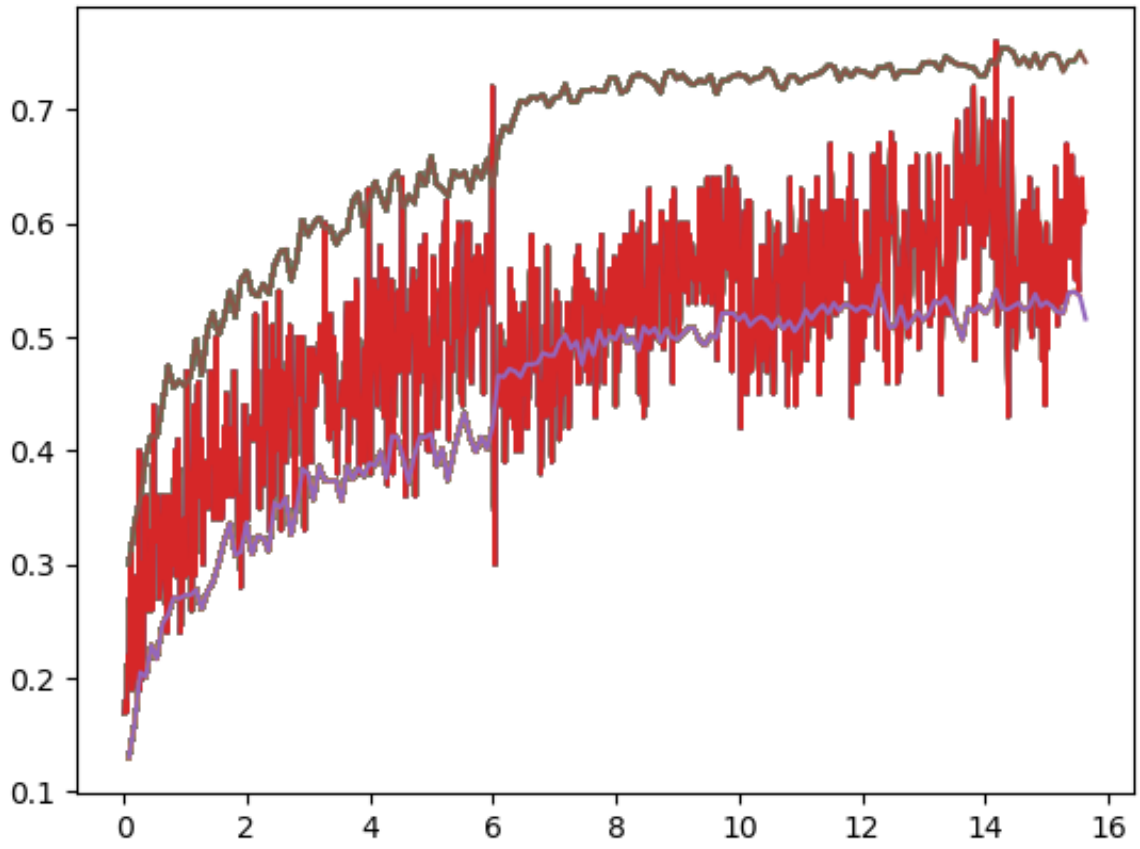

Figure 1: Accuracy curve of the training procedure. Purple line: top-1 accuracy on the validation set. Brown line: top-3 accuracy on the validation set. Red line: top-1 accuracy on the training set.

```
def batch_norm(x):
    shape = x.shape[1:]
    x_mean, x_var = tf.nn.moments(x, [0])
    offset = tf.Variable(tf.constant(0.0, shape=shape))
    scale = tf.Variable(tf.constant(1.0, shape=shape))
    return tf.nn.batch_normalization(x, x_mean, x_var, offset, scale, 0.001)
```

loss function

```
loss = -tf.reduce_sum(y_*tf.log(tf.clip_by_value(y_conv, 1e-10, 1.0))) + alpha*sum([
    tf.reduce_sum(w*w) for w in weight_array])
```

data augmentation

```
def image_clip(image):
    strip = 80
    if image.width >= image.height:
        edge = image.height
        clips = []
        while edge < image.width:
            clips.append(image.crop((edge-image.height, 0, edge, image.height)))
            edge += strip
        return clips
    else:
        edge = image.width
        clips = []
        while edge < image.height:
            clips.append(image.crop((0, edge-image.width, image.width, edge)))
            edge += strip
        return clips


def load_data(description, augment=False):
    pair = re.split(' ', description)
    sy = np.zeros(80)
    sy[int(pair[1])] = 1
    with Image.open("data/"+pair[0]) as im:
        if not augment:
            return [[np.array(im.resize((256, 256), Image.ANTIALIAS)), sy]]
        else:
            im_array = []
            im_array.append(im)
            im_array += image_clip(im)
            im_array.append(im.transpose(Image.FLIP_LEFT_RIGHT))
            im_array += image_clip(im.transpose(Image.FLIP_LEFT_RIGHT))
            im_array.append(im.filter(ImageFilter.BLUR))
```

```
33              data_array = []
                for ima in im_array:
35                  data_array.append([np.array(ima.resize((256, 256), Image.ANTIALIAS)),
                        sy])
                return data_array
```