机器学习团队技术报告-人脸关键点识别

17231087 章玉婷

17231088 田语

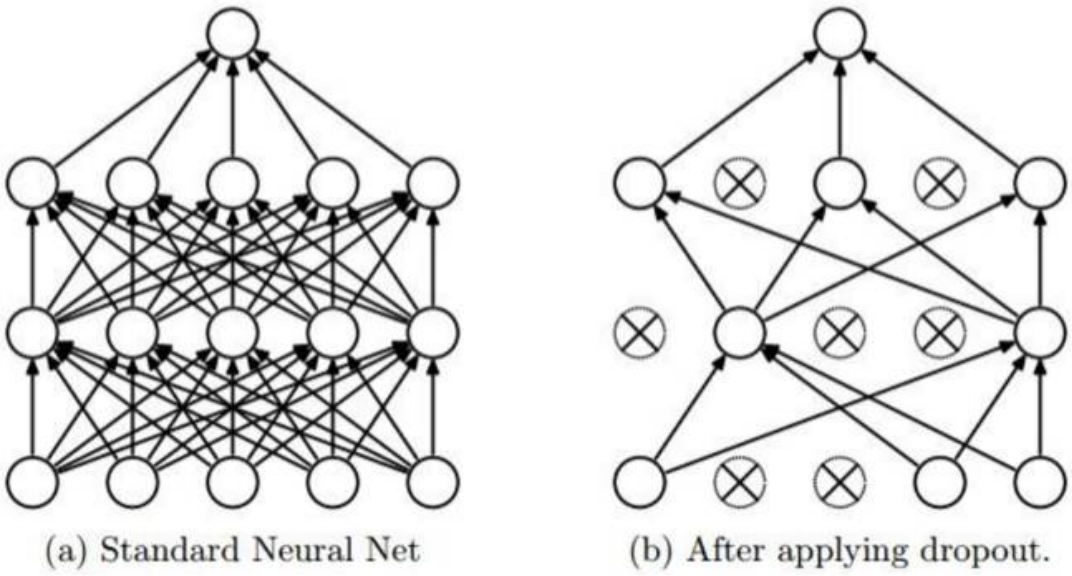
17377118 栗亚舟

使用方法介绍

在全连接神经网络中，每两层之间的节点都有边相连。

对于卷积神经网络，相邻两层之间只有部分节点相连。在卷积神经网络的前几层中，每一层的节点都被组织成一个三维矩阵。前几层中每一个节点只和上一层中部分节点相连。

二者对比见下图



(a) Standard Neural Net　　(b) After applying dropout.

利用全连接神经网络会造成参数过多，而利用卷积可以有效减小参数个数，有效地减少过拟合问题，提高优化效率，于是我们利用卷积神经网络进行学习以及预测。

卷积神经网络主要由以下结构组成：

①输入层

②卷积层：尝试利用多个卷积核的卷积操作从输入层提取更高的特征。

③池化层：保留最重要的部分并提高模型的畸变容限,本次作业中使用了最大池化，提取最大特征。

④全连接层：将图像抽象为具有较高信息含量的特征，然后使用神经网络完成后续的分类和其他任务。

⑤输出层：通常利用$softmax$用于输出概率值或分类结果

实现细节

使用tensorflow框架，实现cnn网络进行深度学习。

首先因为图片太大，计算需要大量内存，自己的电脑无法胜任，所以要先对图像进行处理使图像大小变小。用PIL库读取图片并将其转换为灰度图片以降低维数

```
I=Image.open(path)
L=I.convert("L")
```

再将图片转换成numpy数组，并用opencv库，将不同大小的数组变成统一的96*96像素，

`x=cv2.resize(x,(96,96))`

并对.pts中的标准坐标进行统一

```
pos_x=float(pos_x)/x_size
pos_y=float(pos_y)/y_size
```

将数据以9：1的比例分为训练集与验证集。

将训练集的图像数组和坐标输入神经网络中进行训练，以均方根误差为损失函数

每次训练前将数据shuffle增强随机性与泛化性能。

```
np.random.shuffle(train_index)  #每个epoch都shuffle一下效果更好
X_train, y_train = X_train[train_index], y_train[train_index]
```

对原数据进行数据增强，包括对比度、色度和饱和度的设置

```
random_brightness = tf.image.random_brightness(img,max_delta=30)
    #随机设置图片的对比度
random_contrast = tf.image.random_contrast(img,lower=0.2,upper=1.8)
    #随机设置图片的色度
random_hue = tf.image.random_hue(img,max_delta=0.3)
    #随机设置图片的饱和度
random_satu = tf.image.random_saturation(img,lower=0.2,upper=1.8)
```

将验证集数据输入神经网络训练出的模型，计算出loss来检验模型的性能，其中网络的具体实现见下节模型结构。

最后将测试集输入网络得出结果，并将坐标变为原来图像的大小。为了实现结果的可视化，利用opencv将坐标的点画在图像上

`cv2.circle(x_test,(int(pt[1][i]*x_size),int(pt[0][i]*y_size)),2, (255, 0, 0),-1)`

效果如图：

模型结构

- **输入层**

  将图像转换为灰度图像，reshape变成标准96*96的np矩阵，将训练集关键点的横纵左边规范化到0-1区间

- **卷积和池化**

  在模型的权重初始化中添加少量噪声，以打破对称性并避免零梯度。使用一个小的正数来初始化偏差项，以避免出现神经元节点的输出始终为0的问题。

```
def weight_variable(shape, namew='w'):

    initial = tf.truncated_normal(shape, stddev=0.1)

    return tf.Variable(initial, name=namew)

def bias_variable(shape, nameb='b'):

    initial = tf.constant(0.1, shape=shape)

    return tf.Variable(initial, name=nameb)
```

卷积使用1步，0边距模板。为确保输出和输入大小相同，池化使用简单的传统2x2尺寸模板进行最大池化

```
def conv2d(x, W):

  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):

  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME')
```

- **中间隐层**

  第一层

  它由卷积和最大池化完成。卷积在每个3x3的$patch$中计算32个特征。卷积的权重张量形状为[3,3,1,32]，前两个维是$patch$的大小，其次是输入通道的数量，最后是输出通道的数量。对于每个输出通道，都有一个对应的偏移量。

```
        W_conv1 = weight_variable([3, 3, 1, 32], 'w1') # 32个3*3*1的卷积核

        b_conv1 = bias_variable([32], 'b1')
```

我们把x_image和权值向量进行卷积，加上偏置项，然后应用$ReLu$激活函数，最后进行池化。

```
        h_conv1 = tf.nn.relu(conv2d(x, W_conv1) + b_conv1)

        h_pool1 = max_pool_2x2(h_conv1)
```

第二层

把几个类似的层堆叠起来，为64个2232的卷积核，经池化变成23*23

```
    W_conv2 = weight_variable([2, 2, 32, 64], 'w2') # 64个2*2*32的卷积核

    b_conv2 = bias_variable([64], 'b2')

    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)

    h_pool2 = max_pool_2x2(h_conv2)
```

第三层

把几个类似的层堆叠起来，为128个2264的卷积核，经池化变成11*11。

```
    W_conv3 = weight_variable([2, 2, 64, 128], 'w3') # 128个2*2*32的卷积核

    b_conv3 = bias_variable([128], 'b3')

    h_conv3 = tf.nn.relu(conv2d(h_pool2, W_conv3) + b_conv3)

    h_pool3 = max_pool_2x2(h_conv3)
```

- **全连接层**
  将第三层卷积输出为一维向量，大小为500，我们添加了一个具有500个神经元的完整连接层来处理整个图像。我们将池化层的输出张量通过reshape转换为向量，将其乘以权重矩阵，添加偏移量，然后对其使用$ReLu$激活函数。

```
    W_fc1 = weight_variable([11 * 11 * 128, 500], 'wf1') # 全连接层

    b_fc1 = bias_variable([500], 'bf1')

    h_pool3_flat = tf.reshape(h_pool3, [-1, 11 * 11 * 128]) # 把第三层卷积的输出一维向量

    h_fc1 = tf.nn.relu(tf.matmul(h_pool3_flat, W_fc1) + b_fc1)
```

- **输出层**

  由于需要输出98个点的横纵坐标，于是经过linear转换为196的列向量

```
    W_fc2 = weight_variable([500, 500], 'wf2')

    b_fc2 = bias_variable([500], 'bf2')

    h_fc2 = tf.nn.relu(tf.matmul(h_fc1, W_fc2) + b_fc2, name='hfc2')

    h_fc2_drop = tf.nn.dropout(h_fc2, keep_prob)

    W_fc3 = weight_variable([500, 196], 'wf3')

    b_fc3 = bias_variable([196], 'bf3')

    y_conv = tf.add(tf.matmul(h_fc2_drop, W_fc3) + b_fc3, 0.0, name='output')

    # 以均方根误差为代价函数，Adam为优化器

    rmse = tf.sqrt(tf.reduce_mean(tf.square(y_ - y_conv)))

    train_step = tf.train.AdamOptimizer(1e-3).minimize(rmse)
```

Dropout

为了减少过拟合，我们在输出层之前加入$dropout$。我们用一个$placeholder$来代表一个神经元的输出在$dropout$中保持不变的概率。在训练过程中启用$dropout$，在测试过程中关闭$dropout$。

```
    keep_prob = tf.placeholder("float", name='keep_prob')  # dropout概率值
```

- **训练和评估模型**

  利用均方根误差计算loss，采用ADAM优化器

```
    rmse = tf.sqrt(tf.reduce_mean(tf.square(y_ - y_conv)))

    train_step = tf.train.AdamOptimizer(1e-3).minimize(rmse)
```

- **test集预测**

  此时将keep_prob设置为1.0,求解出规范化的横纵坐标，再按照比例得到真实图片中的关键点的横纵坐标

```
predictions = sess.run(y_conv, feed_dict={x: np.reshape(x_test, [-1, 96, 96, 1]),keep_prob: 1.0})

pt = np.vstack(np.split(predictions[0], 98)).T

x_test=cv2.resize(x_test,(x_size,y_size))

for j in range(98):
    print(",%.6f,%.6f" % (pt[1][j] * x_size, pt[0][j] * y_size), file=output, end="")
```

调参

本模型可以调的参数较多。

- $epoch$

  对于$epoch=5$，$epoch=10$，$epoch=20$，$epoch=40$的情况进行了测试，并使$learning\ rate$随$epoch$增大而减小，随$epoch$增加准确率升高，但由于计算机算力限制，没有测试$epoch$更多的情况。

- $batch_size$

  在固定$epoch=10$的情况下，对于$batch_size=64,32,16,8$的情况进行了测试，测试结果如下：

  $batch\_size=16$：

6	17.520897	submission.zip	12/22/2019 18:03:43	Finished	

  $batch\_size=64$：

4	18.344597	submission.zip	12/22/2019 15:44:29	Finished	

  $batch\_size=32$：

10	19.222054	submission.zip	12/22/2019 19:29:26	Finished		+

$batch\_size = 8$：

7	16.931937	submission.zip	12/22/2019 18:22:50	Finished	✔

可以发现，$batch_size$减小对准确率的提升有较大帮助。

- $learning\ rate$：

  在固定$epoch = 10$，$batch\_size = 8$，图像规范后大小为$96 * 96$的情况下，对$learning\ rate = 1e-1, 1e-2, 1e-3, 1e-4, 1e-5$的情况进行了测试，测试结果如下：

  - $learning\ rate = 1e-2$：

8	39.201087	submission.zip	12/22/2019 18:45:26	Finished	+

  - $learning\ rate = 1e-3$：

7	16.931937	submission.zip	12/22/2019 18:22:50	Finished	✔

  - $learning\ rate = 1e-4$：

9	25.660331	submission.zip	12/22/2019 19:10:50	Finished	+

  可以发现在$epoch = 10$时，$learning\ rate = 1e-3$的效果最好，猜测可能由于$epoch$较小，而$learning\ rate = 1e-4$的时候收敛较慢，故效果不如$1e-3$的情况

遇到的问题和解决方法

1. 训练集过大，内存不够，在读取数据的过程中，如果一次性全部读取，就会出现$kernel\ died$的报错

   解决办法：先对图片进行处理，将三通道的RGB图像转换为灰度图并保存，在训练模型构造训练集的时候，直接读取灰度图。

2. 将得到的关键点坐标在原图上显示的时候，发现点集中在一个方形区域内，和原图不符。

   解决方法：由于在训练和测试的过程中，我们对图像进行了规范化，而后续对得到的关键点坐标进行还原映射的过程中，x坐标和y坐标取反了。

3. 训练速度太慢。

   解决方法：没有找到本地跑的好的解决方法。

4. 训练$loss$就出现了NaN。

   解决方法：在网上搜了搜发现原因因为交叉熵$loss$中用到了log函数，当输入为0时就会出现NaN。于是将$loss$函数改为均方根误差。

分工

神经网络结构及数据读入：栗亚舟

数据增强：田语

调参：章玉婷

实验报告：合作完成

最终结果及截图

15	17373342_邓力友	9	12/22/19	**队伍1**	14.0954 (14)
16	17231139_张庆玉	8	12/21/19	**jojo的奇妙检测**	14.4689 (15)
17	17231059_丁菲	20	12/22/19	**摸鱼小分队**	15.7552 (16)
18	17377118_栗亚舟	5	12/22/19	**single three**	16.9319 (17)