

基于卷积神经网络的图像识别设计与实现

摘要: 图像分类任务是计算机视觉的核心, 实际应用广泛, 图像分类对人类来说是个十分简单的问题, 而对于计算机来说却并不容易。卷积神经网络的能力强大, 其结构特别适用于解决计算机视觉任务。通过卷积神经网络对采集结果的训练, 得到用来识别的各类特征, 识别的结果可以得到图像类别信息。本文就深度学习在计算机视觉领域的应用进行简单的实践与叙述, 目标是实现基于卷积神经网络的图像识别与分类。卷积神经网络在图像识别领域属于较经典、较成熟的模型。本文中利用卷积神经网络实现图像的分类问题, 实现使用的是谷歌深度学习框架 Tensorflow, 使用果蔬图像数据集训练模型后, 可以达到随机输入单张果蔬图片, 返回图片分类结果的效果。

关键词: 卷积神经网络; 深度学习; 图像识别; Tensorflow

0 引言

随着科学技术的飞速发展, 图像识别技术在社会各领域得以应用。图形识别技术可以作为一项基础技术应用于如工业零件分类、人脸识别以及手势识别等。当前的图像识别也是作为一项十分热门的技术被大众所广泛讨论。深度学习是机器学习的一个新的热门研究方向, 其旨在模仿人类的学习模式, 通过对输入样本的训练与测试, 由简及深地提取特征来区分样本。通过深度学习来进行图像识别也是如此, 通过对于图像样本的训练和测试, 对样本进行分类。一般来说, 深度学习网络满足多层结构、自动进行特征提取和自动更新网络权重等特点。

卷积神经网络(Convolutional Neural Network, CNN)作为深度学习的经典模型, 一直备受关注。卷积神经网络, 简称 CNN, 是多层神经网络模型的一个变种, 受到生物学的启发, 卷积神经网络在图像领域得到了广泛的应用。最早的神经网络模型是 1998 年由 LeCun 等人提出的 LeNet5m, 它以交替出现的卷积层和池化层作为基础的主干网络, 结合全连接层组成完整的网络结构。在 2012 年, Krizhevsky 等人设计了 AlexNet 网络, 它的主干网络包含了五个卷积层, 全连接层增加到三个, 并且将传统的激活函数替换成 ReLU 函数。2013 年, MinLin 在 Network in Network 中首次明确提出了在进行卷积运算的时候使用 1×1 的卷积核。2014 年, Szegedy 等人提出了并行卷积的 Inception 模块。2015 年, HeK 等人提出了使用两个 3×3 的卷积核代替原来的 5×5 的卷积核的 MSRA-Net[S], 使网络的性能得到非常大的提升。同年, HeK 等人提出的 ResNet 网络, 进一步提升了网络性能。本文讨论卷积神经网络应用在图像识别这一课题。

1 卷积神经网络简介

卷积神经网络是一种带有卷积结构的深度神经网络, 卷积结构可以减少深层网络占用的内存量, 其三个关键的操作, 其一是局部感受野, 其二是权值共享, 其三是 pooling 层, 有效

的减少了网络的参数个数, 缓解了模型的过拟合问题。

1.1 网络结构

1.1.1 卷积神经网络整体架构

卷积神经网络是一种多层的监督学习神经网络, 隐含层的卷积层和池采样层是实现卷积神经网络特征提取功能的核心模块。该网络模型通过采用梯度下降法最小化损失函数对网络中的权重参数逐层反向调节, 通过频繁的迭代训练提高网络的精度。卷积神经网络的低隐层是由卷积层和最大池采样层交替组成, 高层是全连接层对应传统多层感知器的隐含层和逻辑回归分类器。第一个全连接层的输入是由卷积层和子采样层进行特征提取得到的特征图像。最后一层输出层是一个分类器, 可以采用逻辑回归, Softmax 回归甚至是支持向量机对输入图像进行分类。

1.1.2 卷积神经网络结构包括

卷积层, 降采样层, 全链接层。每一层有多个特征图, 每个特征图通过一种卷积滤波器提取输入的一种特征, 每个特征图有多个神经元。输入图像统计和滤波器进行卷积之后, 提取该局部特征, 一旦该局部特征被提取出来之后, 它与其他特征的位置关系也随之确定下来了, 每个神经元的输入和前一层的局部感受野相连, 每个特征提取层都紧跟一个用来求局部平均与二次提取的计算层, 也叫特征映射层, 网络的每个计算层由多个特征映射平面组成, 平面上所有的神经元的权重相等。通常将输入层到隐藏层的映射称为一个特征映射, 也就是通过卷积层得到特征提取层, 经过 pooling 之后得到特征映射层。

1.2 局部感受野与权值共享

卷积神经网络的核心思想就是局部感受野、是权值共享和 pooling 层, 以此来达到简化网络参数并使得网络具有一定程度的位移、尺度、缩放、非线性形变稳定性。

1.2.1 局部感受野

由于图像的空间联系是局部的, 每个神经元不需要对全部的图像做感受, 只需要感受局部特征即可, 然后在更高层将这些感受得到的不同的局部神经元综合起来就可以得到全局的信

息了，这样可以减少连接的数目。

1.2.2 权值共享

不同神经元之间的参数共享可以减少需要求解的参数，使用多种滤波器去卷积图像就会得到多种特征映射。权值共享其实就是对图像用同样的卷积核进行卷积操作，也就意味着第一个隐藏层的所有神经元所能检测到处于图像不同位置的完全相同的特征。其主要的功能就能检测到不同位置的同一类型特征，也就是卷积网络能很好的适应图像的小范围的平移性，即有较好的平移不变性。

1.3 卷积层、下采样层、全连接层

1.3.1 卷积层

因为通过卷积运算我们可以提取出图像的特征，通过卷积运算可以使得原始信号的某些特征增强，并且降低噪声。用一个可训练的滤波器 $f(x)$ 去卷积一个输入的图像（第一阶段是输入的图像，后面的阶段就是卷积特征 map），然后加一个偏置

b_x ，得到卷积层 c_x 。

1.3.2 下采样层

下采样层也叫池化层，其具体操作与卷积层的操作基本相同，只不过下采样的卷积核为只取对应位置的最大值、平均值等（最大池化、平均池化），并且不经过反向传播的修改。对图像进行下采样，可以减少数据处理量同时保留有用信息，采样可以混淆特征的具体位置，当某个特征找出来之后，它的位置已经不重要了，我们只需要这个特征和其他特征的相对位置，就可以应对形变和扭曲带来的同类物体的变化。

1.3.3 全连接层

采用 softmax 全连接，得到的激活值即卷积神经网络提取到的图片特征。

2 Tensorflow 简介

Tensorflow 是一个采用数据流图(data flow graphs), 用于数值计算的开源软件库。TensorFlow 最初由 Google 大脑小组(隶属于 Google 机器智能研究机构)的研究员和工程师们开发出来，用于机器学习和深度学习方面的研究，但这个系统的通用性使其也可广泛用于其他计算领域。它是谷歌基于 DistBelief 进行研发的第二代人工智能学习系统。2015 年 11 月 9 日，Google 发布人工智能系统 TensorFlow 并宣布开源。其命名来源于本身的原理，Tensor(张量)意味着 N 维数组，Flow(流)意味着基于数据流图的计算。Tensorflow 运行过程就是张量从图的一端流动到另一端的计算过程。张量从图中流过的直观图像是其取名为“TensorFlow”的原因。TensorFlow 的关键点是：“Data Flow Graphs”，表示 TensorFlow 是一种基于图的计算框架，其中节点(Nodes)在图中表示数学操作，线(Edges)则表示在节点间相互联系的多维数据数组，即张量(Tensor)，这种基于流的架构让 TensorFlow 具有非常高的灵活性，该灵活性也让 TensorFlow 框架可以在多个平台上进行计算，例如：台式

计算机、服务器、移动设备等。

2.1 核心概念：数据流图

数据流图(如下图 1)用“结点”(nodes)和“线”(edges)的有向图来描述数学计算。“节点”一般用来表示施加的数学操作，但也可以表示数据输入(feed in)的起点/输出(push out)的终点，或者是读取/写入持久变量(persistent variable)的终点。“线”表示“节点”之间的输入/输出关系。这些数据“线”可以输运“size 可动态调整”的多维数据数组，即“张量”(tensor)。张量从图中流过的直观图像是这个工具取名为“Tensorflow”的原因。一旦输入端的所有张量准备好，节点将被分配到各种计算设备完成异步并行地执行运算。图中包含：输入(input)、塑形(reshape)、ReLU 层(ReLU Layer)、Logit 层(Logit Layer)、Softmax、交叉熵(cross entropy)、梯度(gradient)、SGD 训练(SGD Trainer)等部分，是一个简单的回归模型。

图中包含：输入(input)、塑形(reshape)、ReLU 层(ReLU Layer)、Logit 层(Logit Layer)、Softmax、交叉熵(cross entropy)、梯度(gradient)、SGD 训练(SGD Trainer)等部分，是一个简单的回归模型。

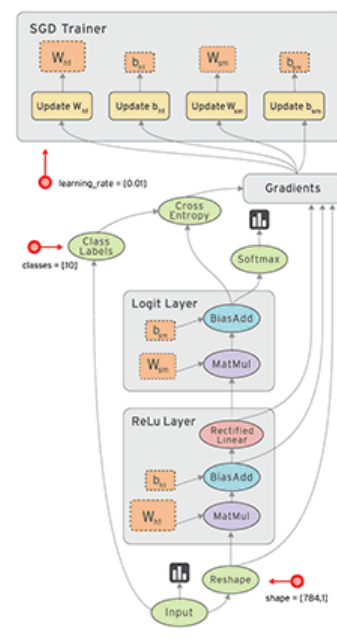


图 1 数据流图

2.2 Tensorflow 特性

- 1) 高度的灵活性：只要能够将计算表示成为一个数据流图，那么就可以使用 TensorFlow；
- 2) 可移植性：TensorFlow 支持 CPU 和 GPU 的运算，并且可以运行在台式机、服务器、手机移动端设备等等；
- 3) 自动求微分：TensorFlow 内部实现了自动对于各种给定目标函数求导的方式；
- 4) 多种语言支持：Python、C++；
- 5) 性能高度优化。

3 Tensorflow 构建 CNN

3.1 构建 CNN 准备

3.1.1 创建权重函数

```
def weight_variable(shape, name="w"):
    initial = tf.contrib.layers.xavier_initializer()
    return tf.get_variable(name, initializer=initial,
                           shape=shape)
```

注意：权重初始化值选择 xavier，这个初始化器是用来保持每一层的梯度大小都差不多相同。

3.1.2 创建偏置函数

```
def biases_variable(shape):
    initial = tf.constant(0.0, shape=shape)
    return tf.Variable(initial)
```

3.1.3 创建二维卷积函数

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

如果 padding='SAME'，则输出的卷积后图像大小与输入的大小一样。如果 padding='VALID'，则输出的卷积后图像大小为 $N=(imgSize-kSize)/Strides$ ，这里 imgSize 为原来图像的宽或者高，kSize 为卷积核大小，Strides 为卷积步长。

3.1.4 创建最大池化层函数

```
def max_pool_2x2(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1]):
    return tf.nn.max_pool(x, ksize, strides, padding='VALID')
```

同样的，如果 padding='SAME'，则输出的卷积后图像大小与输入的大小一样。如果 padding='VALID'，则输出的卷积后图像大小为 $N=(imgSize-kSize)/Strides$ ，这里 imgSize 为原来图像的宽或者高，kSize 为卷积核大小，Strides 为卷积步长。

3.1.5 定义占位符

```
xs = tf.placeholder(tf.float32, [None, 465, 128, 1])
ys = tf.placeholder(tf.float32, [None, 10])
```

定义占位符只需要知道自己输入图的大小就可以按照相应的改写。例如输入 100 张 20X20 的图则： $xs = tf.placeholder(tf.float32, [None, 20, 20, 1])$ 。其中 None 表示可以输入任意多的图。

3.2 构建 CNN 结构

下图 2 为卷积操作的示意图，可以知道，卷积层需要突触权值，偏置（可以选择不要偏置）激活函数，最后得到输出。

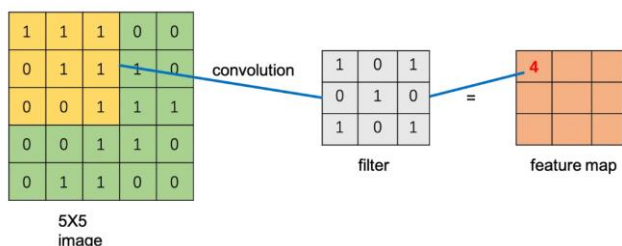


图 2 卷积操作示意举例

3.2.1 创建卷积层，并且用 relu 激活

```
kernerl_size = [3, 3, 1, 32]
strides = [1, 1, 1, 1]
W_conv1 = weight_variable(kernerl_size, name="w1") ##patch
3x3 ,in size 1,out size 32
b1 = biases_variable([kernerl_size[3]])
#b1=0
conv2_1 = tf.nn.relu(conv2d(xs, W_conv1)+b1) # N x 465 x
128 x 32
```

假设需要一个 3x3 的卷积核去卷积输入，并且输出 32 层，每次卷积的步长为 1。那么卷积核大小就为 [3, 3, 1, 32] => [卷积宽度，卷积高度，卷积输入，卷积输出]，卷积步长 [1, 1, 1, 1] => [固定 1，卷积步长宽度，卷积步长高度，固定 1]，因为权重是要随着训练改变的，因此定义权重变量，变量的大小为卷积核的大小。偏置是加入对应每个输出上的，因此是卷积核大小为卷积输出大小，最后卷积用激活函数 relu 去激活。

3.2.2 创建池化层

```
pool_2 = max_pool_2x2(x=conv2_2, ksize=[1, 2, 2, 1],
strides=[1, 2, 2, 1]) # N x 116 x 32 x 64
```

池化层是对卷积后的结果进行降维。降维后每个通道图大小为 $N=(imgSize-kSize)/Strides$ ，这里 imgSize 为原来图像的宽或者高，kSize 为池化核大小，Strides 为池化步长。

3.2.3 对池化得到的结果压平用于全连接层

```
Ft = tf.reshape(pool_2, [-1, 116 * 32 * 64]) # N x 3 x 2
x 64 =>> N x 384
```

压平操作是指矩阵转换成一维矩阵，reshape 即可，需注意矩阵大小要计算准确，最后一维矩阵大小为 $N=high*wide*channel$ ，也就是输出通道数乘以图的宽度高度。

3.2.4 创建全连接层

```
w_fc_1 = tf.Variable(tf.truncated_normal([116 * 32 * 64,
100], stddev=0.01)) # N x 384 =>> N x 100
b3 = biases_variable([100])
x = tf.nn.relu(tf.matmul(Ft, w_fc_1)+b3)
```

全连接是对压平后的数据再次变小，用矩阵乘法得到更新的维度再激活函数激活。

3.2.5 预测

```
out_size = tf.Variable(tf.truncated_normal([100, 10],
stddev=0.01)) # N x 100 =>> N x 10
prediction_values = tf.matmul(x, out_size)
```

预测也是矩阵相乘，压缩输出，到此一个 CNN 构建完成，卷积池化全连接大小可以根据实际情况增加或者减少。

3.3 训练模型

训练模型首先需要定义损失，优化损失方法，然后进行训练。因为训练数据量很大，需要对数据按照 batch 划分，一个小的 batch 进行训练。

3.3.1 定义损失

```
loss=tf.losses.softmax_cross_entropy(ys,prediction_value
s)
```

如果损失函数选择了 softmax, 那么 ys 也就是标签需要定义成 one-hot。比如有 10 个类, 那么第一类 0 的 one-hot 编码为 [1,0,0,0,0,0,0,0,0,0]。

3.3.2 定义训练

```
LEARNING_RATE_BASE = 0.001
LEARNING_RATE_DECAY = 0.1
LEARNING_RATE_STEP = 300
gloabl_steps = tf.Variable(0, trainable=False)
learning_rate=tf.train.exponential_decay(LEARNING_RATE_B
ASE,gloabl_steps,LEARNING_RATE_STEP,LEARNING_RATE_DECAY,
staircase=True)
train_step=tf.train.AdamOptimizer(learning_rate).minimiz
e(loss,global_step=gloabl_steps)
```

最小化损失 minimize 之后把损失函数放进去。

4 实验过程

4.1 准备数据集

数据集分为两个部分, 训练集和测试集, 训练集包含 12 个项目类别共 841 张图片, 测试集同样包含 12 个项目类别共 391 张图片。



图 3 数据输入情况

4.2 CNN 构建

```
# 构建 CNN 模型
def model_load(IMG_SHAPE=(224, 224, 3), class_num=12):
.....
# 输出模型信息
model.summary()
# 指明模型的训练参数, 优化器为 SGD 优化器, 损失函数为交叉熵
# 损失函数
model.compile(optimizer='sgd',
loss='categorical_crossentropy', metrics=['accuracy'])
# 返回模型
return model
```

4.3 网络训练

```
def train(epochs):
# 开始训练, 记录开始时间
begin_time = time()
# todo 加载数据集, 修改为你的数据集的路径
train_ds, val_ds, class_names =
```

```
data_load(r"C:\studysoftware\Anaconda3\pyqt5\pyqt\disting
uish\data\vegetable_fruit\image_data", r"C:\studysoftwar
e\Anaconda3\pyqt5\pyqt\distinguish\data\vegetable_fruit\
test_image_data", 224, 224, 16)
```

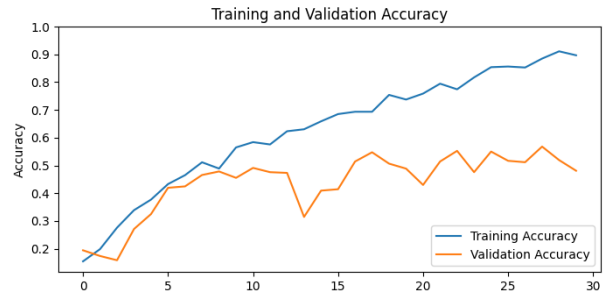


图 4 训练与验证过程的准确率变化趋势






图 5 训练与验证过程的损失值变化趋势

4.4 网络预测

```
# 预测图片
def predict_img(self):
img = Image.open('images/target.png') # 读取图片
img = np.asarray(img) # 将图片转化为 numpy 的数组
outputs = self.model.predict(img.reshape(1, 224, 224,
3)) # 将图片输入模型得到结果
result_index = int(np.argmax(outputs))
result = self.class_names[result_index] # 获得对应的水
果名称
self.result.setText(result) # 在界面上做显示
```

4.5 输出结果

表 1 输出结果对比

测试图			
预测概率	0.608815	0.706612	0.675839
推理结果	胡萝卜	苹果	梨

5 总结与展望

卷积神经网络在计算机视觉方面的应用, 包括图像的分类识别、视频识别等都有着明显的优势, 在语音识别方面也取得了突破性的进展, 但是图像识别在实际运用中仍然存在一些挑战, 值得研究者进一步探讨研究与解决。

1) 图像是识别的基础数据, 图像识别中首要的挑战就是模糊图像、受环境影响的图像(如受光线、噪声影响)、遮挡的图像等情况。虽然前人提出了各种各样的技术来尽可能地减少这种挑战, 但是这些问题依然存在于计算机视觉任务中。

2) 卷积神经网络在检测中需要对数据进行标注, 一个模型的训练过程中往往需要大量手工标注的数据, 随着大规模数据量的涌现, 无标签的未知数据占绝大多数, 为这些数据做标签已经变得不够现实了, 而且找专家来标注数据是非常昂贵的。这个时候我们就要学会从无标注的数据里面进行学习, 现有的研究方法包括生成对抗网络、主动学习(Active Learning)等。

3) 非欧空间数据不存在平移不变性, 采用图卷积神经网络可以处理图数据。在该领域中, 目前存在的主要方法为谱方法和空间方法, 研究表明, 虽然图卷积神经网络取得了一定的成果, 但仍然有很多问题需要解决。

4) 目前在深度学习中训练网络模型需要大量的数据样本, 如何在少量样本情况下即保证识别精度又能大大提高网络的训练速度是很多研究者的一个重要目标。数据扩充(Data Augmentation)是一项非常重要的技术, 其可以从现有的数据中产生更多的有用数据。神经网络中Dropout的引入使得样本不足的条件也能够比较高的识别率, 文献提出采用滑动窗口技术增加训练数据的方法。Chen等人提出了一种GridMask的数据扩增策略, 该策略删除均匀分布的区域, 最后形成网络

形状, 使用此形状删除信息比设置完全随机位置更加有效。

总之, CNN 目前还存在很多待解决的问题, 这些问题不影响在各领域中图像识别的运用和发展, 仍然是研究的一大热点。

6 参考文献

- [1] 李彦冬.卷积神经网络研究综述[J].计算机应用,2016,(09):5~7.
- [2] 刘小文,郭大波,李聪.卷积神经网络中激活函数的一种改进[J].测试技术学报,2019,33(02):121-125.
- [3] 刘万军,梁雪剑,曲海成.自适应增强卷积神经网络图像识别[J].中国图象图形学报,2017,22(12):1723-1736.
- [4] 杨估.基于卷积神经网络的手写汉字识别研究[J].信息技术与信息化,2018(12):223-225.
- [5] 杨真真,匡楠,范露,康彬.基于卷积神经网络的图像分类算法综述[J].信号处理,2018,34(12):1474-1489.
- [6] 陈全,王泽,贾伟.基于深度卷积神经网络的车标识别研究[J].工业控制计算机,2018,31(12):36-38.
- [7] 李丹.基于 Lenet-5 的卷积神经网络改进算法[J].计算机时代,2016,(08):28~30.