揭开深度学习的神秘面纱

不知不觉，人工智能已经悄然的走进我们的生活。在我们的潜意识里，还是将人工智能和智能人形机器人联系在一起，这是我们对人工智能的固有偏见。在2016年的云栖大会上，马云的演讲飞速的被翻译成文字，显示在现场的大屏幕上；自动驾驶是汽车产业与人工智能、物联网、高性能计算等新一代信息技术深度融合的产物，是当前全球汽车与交通出行领域智能化和网联化发展的主要方向，已成为各国争抢的战略制高点；手机中的语音助手Siri，总能和我们愉快的聊天；iPhone手机上的照片-相簿-人物中自动对照片中的人脸进行识别和归类，将包含一个人脸的照片放在同一个相簿里面；今日头条中推荐的新闻总是你感兴趣的东西，不同人打开淘宝App出来的界面总不一样。人工智能已经应用在我们生活的很多领域了。

到底什么是人工智能（AI）,创新工场的李开复对AI在《人工智能》中列举了AI的五个常见定义：1）AI就是让人觉得不可思议的计算机程序；2）AI就是与人类思考方式相似的计算机程序；3） AI就是与人类行为相似的计算机程序；4）AI就是会学习的计算机程序；5）AI就是根据对环境的感知，做出合理的行动，并获得很大收益的计算机程序。从其五点定义中，我们可以很轻易的得到这个结论：AI就是计算机程序。当然并不仅仅是这样，详细点的概括为，AI是一种为了解决某些问题而人工设计出的计算机程序，这个程序的算法是模仿人类思考问题的方式，并且具有学习能力。

2016年，AlphaGo与李世石的围棋大战无疑成为人工智能的一道里程碑，也是人工智能的再一次复兴。反观AlphaGo低层算法原理，如下图2所示，需要收集大量棋手的棋谱，通说深度神经网络来训练模型。人工智能的这一次复兴的最大特点是，AI在语音识别、机器视觉、数据挖掘等多个领域走进了业界多个真实的应用场景，与商业模式紧密结合，开始在各个行业发挥出真正的价值。可以说新时期的人工智能模式为：深度学习+大数据。

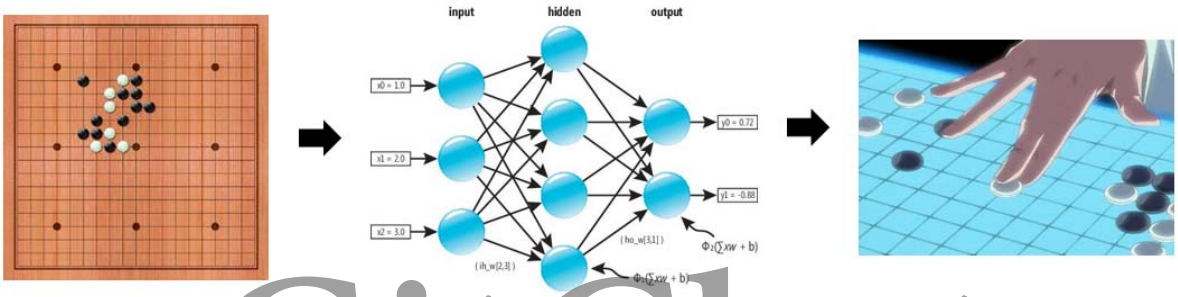

图2 AlphaGo算法模式

现在我们走进深度学习，是机器学习的一个分支。深度学习算法可以从数据中学习更加复杂的特征表达，是最后一步的特征学习变得更加简单有效。就图像识别而言，深度学习可以一层一层地将简单的像素特征逐步转化成更加复杂的特征，从而使不同类别的图像更加可分。和传统机器学习相比，深度学习的特点主要体现在“深度”二字中，主要体现在网络结构的深度和提取的特征深度。本文主要针对的是深度学习在机器视觉上的应用，那么我们就绕不开一个重要的网络结构——卷积神经网路。

在早期的图像识别中，主要是从图像低层特征入手，对图像特征进行归纳分析，当然，组织特征也是图像识别技术最大的难点。卷积神经网络，英文全称为Convolutional Neural Network，简称CNN，在最开始就是为了用来解决图像中的物体识别等问题，现

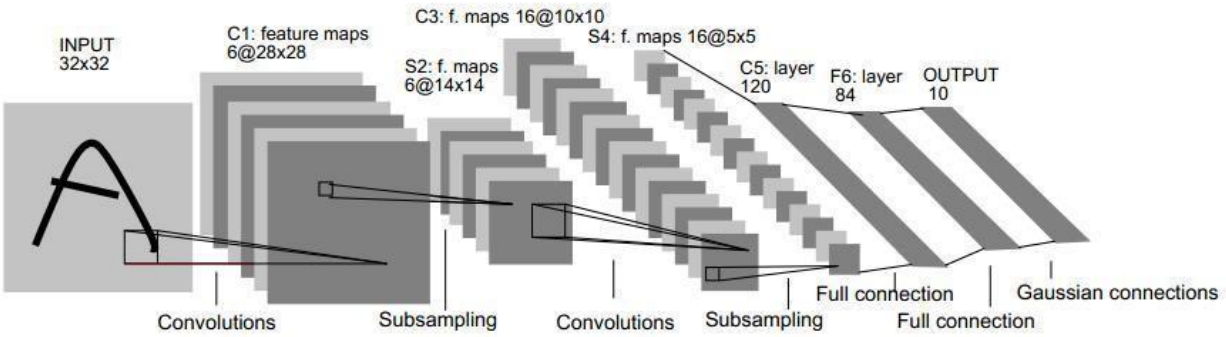

图3 LeNet网络结构

经典的CNN的网络结构如图3所示，卷积神经网络主要结构有：卷积层、池化层和全连接层。在LeNet的网络结构中，对于一个32×32的图像，首先用6个卷积核对图像进行卷积得到6×28×28的一对数据，然后对数据的每一维做一个池化，得到6×14×14的数据结构，后面再用有16个卷积核的卷积层对数据进行卷积得到16×10×10的数据结构，再用一个池化层对将数据简化成16×5×5，后面接一个全连接的神经网络。

图4所示为我们常用的卷积神经网络实例，变化比较多的可能是卷积层和池化层的层数，卷积层和池化层的之中可能会连接数次，还可以在所用卷积核的Size上面做文章，也能达到不同的效果。

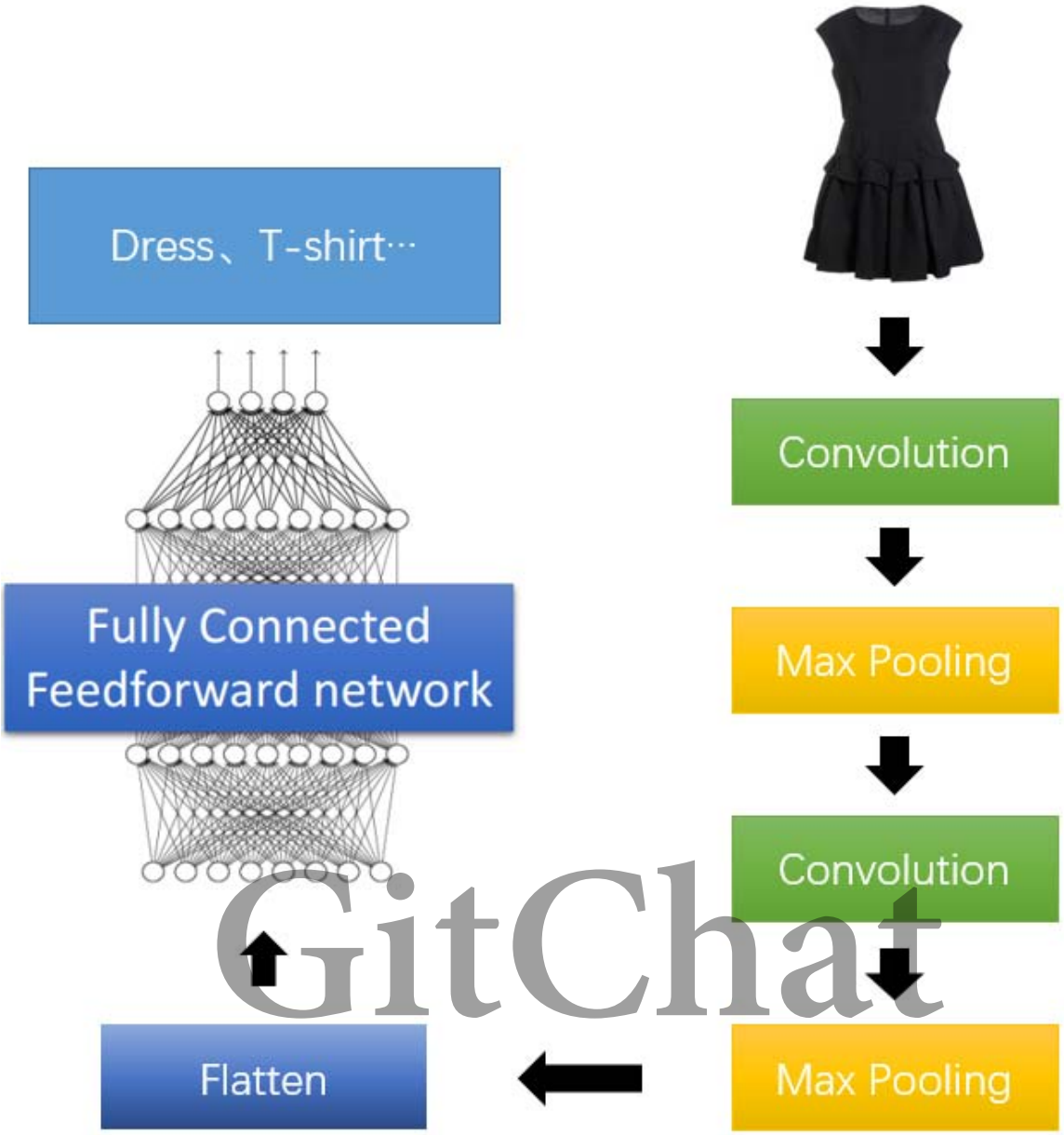

图4 CNN运用实例

以Mnist数字集为例来构建自己的CNN网络，下文中分别用Keras和TensorFlow对CNN网络进行了实现，实现代码如下所示。

```
seed = 7
numpy.random.seed(seed)
```

加载数据：

```
(X_train,y_train),(X_test,y_test) = mnist.load_data()

X_train =
X_train.reshape(X_train.shape[0],1,28,28).astype('float32')
X_test =
X_test.reshape(X_test.shape[0],1,28,28).astype('float32')
X_train = X_train/255
X_test = X_test/255
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]

def baseline_model():
```

构建模型：


```
model = Sequential()

model.add(Convolution2D(30,5,5,border_mode='valid',input_shape=
(1,28,28),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Convolution2D(15,3,3,activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dense(50,activation='relu'))
model.add(Dense(num_classes,activation='softmax'))
```

```
    from tensorflow.examples.tutorials.mnist import input_data
    import tensorflow as tf
    mnist = input_data.read_data_sets("MNIST_data/",
one_hot=True)# 读取图片数据集
    sess = tf.InteractiveSession()# 创建session
```

一、函数声明部分

```
        def weight_variable(shape):
            # 正态分布，标准差为0.1，默认最大为1，最小为-1，均值为0
                initial = tf.truncated_normal(shape, stddev=0.1)
                return tf.Variable(initial)
        def bias_variable(shape):
            # 创建一个结构为shape矩阵也可以说是数组shape声明其行列，初始
化所有值为0.1
                initial = tf.constant(0.1, shape=shape)
                return tf.Variable(initial)
        def conv2d(x, W):
            # 卷积遍历各方向步数为1，SAME：边缘外自动补0，遍历相乘
            return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1],
padding='SAME')
        def max_pool_2x2(x):
            # 池化卷积结果（conv2d）池化层采用kernel大小为2*2，步数也为
2，周围补0，取最大值。数据量缩小了4倍
            return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],strides=
[1, 2, 2, 1], padding='SAME')
```


二、定义输入输出结构

```
        # 声明一个占位符，None表示输入图片的数量不定，28*28图片分辨率
        xs = tf.placeholder(tf.float32, [None, 28*28])
        # 类别是0-9总共10个类别，对应输出分类结果
        ys = tf.placeholder(tf.float32, [None, 10])
        keep_prob = tf.placeholder(tf.float32)
```

```
        # 图片乘以卷积核，并加上偏执量，卷积结果28x28x32
        h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
        # 池化结果14x14x32 卷积结果乘以池化卷积核
        h_pool1 = max_pool_2x2(h_conv1)

        ## 第二层卷积操作 ##
        # 32通道卷积，卷积出64个特征
        w_conv2 = weight_variable([5,5,32,64])
        # 64个偏执数据
        b_conv2  = bias_variable([64])
        # 注意h_pool1是上一层的池化结果，#卷积结果14x14x64
        h_conv2 = tf.nn.relu(conv2d(h_pool1,w_conv2)+b_conv2)
        # 池化结果7x7x64
        h_pool2 = max_pool_2x2(h_conv2)
        # 原图像尺寸28*28，第一轮图像缩小为14*14，共有32张，第二轮后图像缩小为7*7，共有64张

        ## 第三层全连接操作 ##
        # 二维张量，第一个参数7*7*64的patch，也可以认为是只有一行7*7*64个数据的卷积，第二个参数代表卷积个数共1024个
        W_fc1 = weight_variable([7*7*64, 1024])
        # 1024个偏执数据
        b_fc1 = bias_variable([1024])
        # 将第二层卷积池化结果reshape成只有一行7*7*64个数据#[n_samples, 7, 7, 64] ->> [n_samples, 7*7*64]
        h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
        # 卷积操作，结果是1*1*1024，单行乘以单列等于1*1矩阵，matmul实现最基本的矩阵相乘，不同于tf.nn.conv2d的遍历相乘，自动认为是前行向量后列向量
        h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

        # dropout操作，减少过拟合，其实就是降低上一层某些输入的权重scale，甚至置为0，升高某些输入的权值，甚至置为2，防止评测曲线出现震荡，个人觉得样本较少时很必要
        # 使用占位符，由dropout自动确定scale，也可以自定义，比如0.5，根据tensorflow文档可知，程序中真实使用的值为1/0.5=2，也就是某些输入乘以2，同时某些输入乘以0
        keep_prob = tf.placeholder(tf.float32)
```

```
        cross_entropy = -tf.reduce_sum(ys * tf.log(y_conv)) # 定
义交叉熵为loss函数
        train_step =
tf.train.DradientDescentOptimizer(0.5).minimize(cross_entropy) #
调用优化器优化，其实就是通过喂数据争取cross_entropy最小化
```

五、开始数据训练以及评测

```
        correct_prediction = tf.equal(tf.argmax(y_conv,1),
tf.argmax(ys,1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction,
tf.float32))
        tf.global_variables_initializer().run()
        for i in range(20000):
            batch = mnist.train.next_batch(50)
            if i%100 == 0:
                    train_accuracy = accuracy.eval(feed_dict=
{x:batch[0], ys: batch[1], keep_prob: 1.0})
                    print("step %d, training accuracy %g"%(i,
train_accuracy))
            train_step.run(feed_dict={x: batch[0], ys: batch[1],
keep_prob: 0.5})
        print("test accuracy %g"%accuracy.eval(feed_dict={x:
mnist.test.images, ys: mnist.test.labels, keep_prob: 1.0}))
```


两种方式实现CNN都是比较容易，但是在代码量上，Keras占据很大优势，但是就代码的灵活程度上，Keras远远不及TensorFlow。事实上，Keras是TensorFlow的简化接口。具体地，Keras简单易学，对于想把深度学习作为黑箱子使用的人，Keras非常容易上手，在扩展性上面，Keras将大部分的内部运算代码都隐藏了，我们就自然而然的认为它的功能比较狭窄，但是我们还是可以通过theano和Tensorflow的语句来扩展功能和Keras结合使用。但是对初学者而言，不建议直接上手Keras，因为这样会忽略很多细节和低层原理。

以简单的CNN为基础出现了很多先进的卷积网络结构，AlexNet、VGGNet、Google