案例八：基于3DCNN的人体动作识别

1 案例目的

• 理解3DCNN的应用场景；

• 掌握3DCNN算法。

2 案例内容

使用3DCNN进行时空特征学习，用TRECVID 2008数据集进行实验，寻找最佳架构，来对数据集视频中进行人体动作识别

3 案例知识点

• Python语言编程

• 人体动作识别网络架构

• 3DCNN算法

4 案例时长

共2.5学时，具体安排如下：

• 数据预处理（0.5学时）

• 进行模型架构与训练（1学时）

• 查看训练结果（0.5学时）

• 对比分析结果（0.5学时）

5 案例实验环境

•操作系统：

1）Windows 10 x64位操作系统

•软件环境：

1）Python 3.7

•开发环境与工具：

1. tensorflow 1.2

6 案例分析

本案例主要分为以下4部分：

1）数据预处理，把视频数据处理成分类模型需要的格式；

2）训练模型，输入到一个多类线性支持向量机训练模型进行训练；

3）查看训练结果，与基于框架的2D CNN模型进行对比；

4）处理模型输出的结果，增加结果的可读性。

7 案例实验过程

7.1准备数据集；

TRECVID 2008开发数据集包括在伦敦盖特威克机场用5个不同的摄像头拍摄的49小时视频，分辨率为720 x576帧，每秒25帧。4号摄像机拍摄的视频被排除了，因为这个场景发生的事件很少。在当前的实验中，我们主要关注三个动作类(CellToEar, ObjectPut和Pointing)的识别。每一个动作都以一对一的方式进行分类，大量的负样本是由不属于这三个类别的动作产生的。该数据集采集时间为5天(20071101、20071106、20071107、20071108、20071112)。

7.2训练分类模型

计算原始视频帧的特征，基于获得的特征学习分类器。3D CNN架构包含一个硬连线hardwired层、3个卷积层、2个下采样层和一个全连接层，每个3D卷积核卷积的立方体是连续7帧，没帧patch大小是60x40。模型规范化，对于每一个需要训练的行为，我们提取其长时间的行为信息，作为其高级行为特征。模型组合，对一个特定的输入，每个模型都做出预测，然后组合这些模型的预测得到最后的决策。所有的子采样层都是最大子采样，用于训练规范化模型的整体代价函数是有真实的行为类的误差和高层特征的辅助输入代价项的线性加权得到，，权值分别是1和0.005，模型的所有参数都是随机初始化，然后通过diagonal Levenberg-Marquardt方法来优化训练。

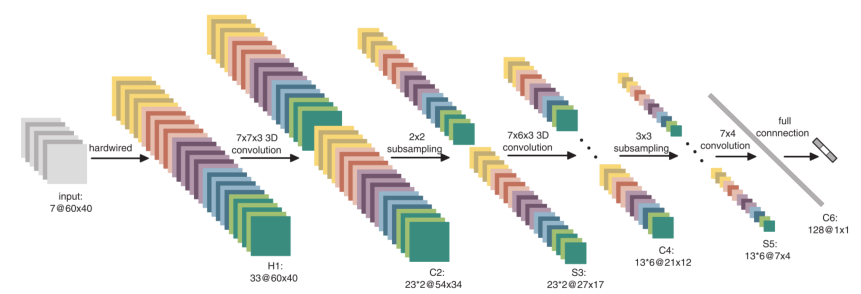


图1 3DCNN模型

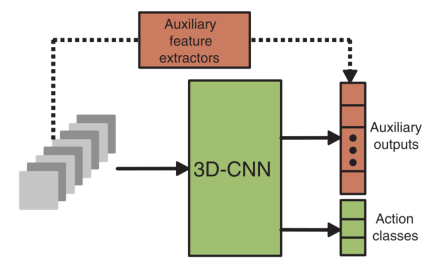


图2模型规范化



图3从相机编号1、2、3和5(从左到右)取样人体检测和跟踪结果

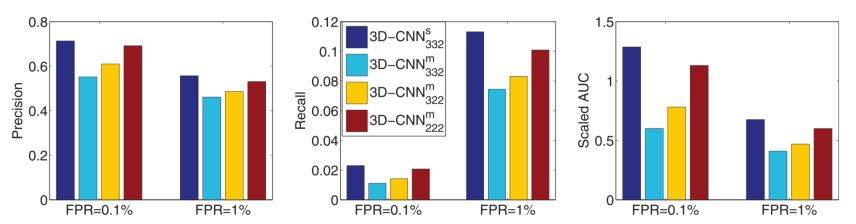


图3四种不同的3D CNN架构在不同假阳性率下的平均性能比较FPR = 0.1%和1%时的AUC分数分别乘以105和103，以获得更好的视觉效果

7.3与基于框架的2D CNN模型做对比分析

表3给出了3DCNN与两种基线和基于框架的2D CNN模型动作识别精度对比。我们报告了5倍交叉验证的结果，其中一天的数据被用作一个折叠。我们采用的测量方法是精度、召回率和多重假阳性率(FPR)下的ROC曲线下面积(ACU)。表2总结了这七种方法的性能，图3绘制了所有操作类的平均性能。从这些结果中我们可以观察到，3D CNN模型在所有情况下的动作类celltoearandobjectputin上都显著优于基于框架的2D CNN模型、SPMcube灰色模型和SPMcube mehisi。对于动作类Pointing, 3D CNN模型的性能略低于其他三种方法。相对于单个特征，SPMcube灰度和SPMcube MEHIfeatures的连接性能有所提高，但仍低于3D CNN模型的性能。我们还可以观察到，我们的模型也优于基于时空特征TISR的方法。总的来说，3D CNN模型的性能始终优于其他方法，从图7的平均性能可以看出。此外，在所有情况下，正则化的模型比没有正则化的模型产生更高的性能。虽然正则化模型的改进不明显，但实验表明，将两种模型结合可以获得显著的性能改进。

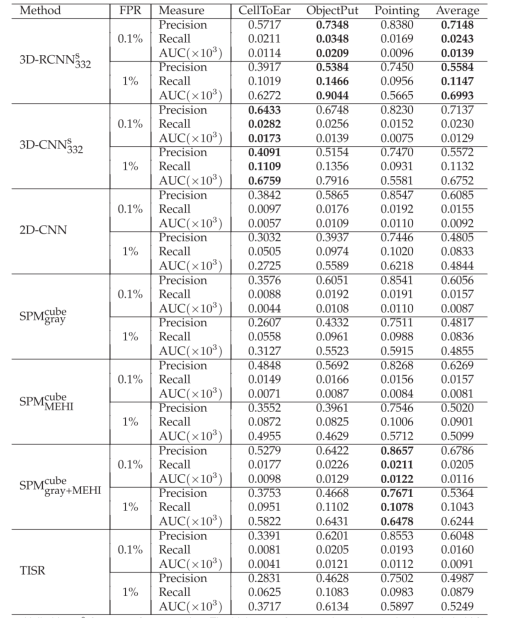


表1. 七种方法在多重假阳性率下的表现

为了评估模型组合在CNN背景下对动作识别的有效性，我们开发了三种可供选择的3DCNN架构，如下所述:。3D- cnnm 332表示不同通道“混合”的架构，前两个卷积层使用3D卷积，最后一层使用2D卷积。“混合”意味着c ha n n e l s otheametype(ie,grdaIent- x and gradient-y,optflow-x,和optflow-y)分别进行卷积，但它们在第一卷积层中连接到同一组特征平面。在第二个卷积层中，所有5个通道都连接到同一组特征平面。相比之下，对于带有上标s的模型，所有5个通道都连接到所有层的分离特征平面上。3D- cnnm 322表示一个类似于3D- cnnm 332的模型，但只有第一个卷积层使用3D卷积，其他两个层使用2D卷积。3D-CNNm 222表示一个类似于3D-CNNm 332的模型，但所有三个卷积层都使用2D卷积。

8 案例代码

|  |
| --- |
|  |
|  | import os |
|  |  |
|  | import matplotlib  import argparse |
|  | matplotlib.use('AGG') |
|  | import matplotlib.pyplot as plt |
|  | import numpy as np |
|  | from keras.datasets import cifar10 |
|  | from keras.layers import (Activation, Conv3D, Dense, Dropout, Flatten, |
|  | MaxPooling3D) |
|  | from keras.layers.advanced\_activations import LeakyReLU |
|  | from keras.losses import categorical\_crossentropy |
|  | from keras.models import Sequential |
|  | from keras.optimizers import Adam |
|  | from keras.utils import np\_utils |
|  | from keras.utils.vis\_utils import plot\_model |
|  | from sklearn.model\_selection import train\_test\_split |
|  |  |
|  | import videoto3d |
|  | from tqdm import tqdm |
|  |  |
|  |  |
|  | def plot\_history(history, result\_dir): |
|  | plt.plot(history.history['acc'], marker='.') |
|  | plt.plot(history.history['val\_acc'], marker='.') |
|  | plt.title('model accuracy') |
|  | plt.xlabel('epoch') |
|  | plt.ylabel('accuracy') |
|  | plt.grid() |
|  | plt.legend(['acc', 'val\_acc'], loc='lower right') |
|  | plt.savefig(os.path.join(result\_dir, 'model\_accuracy.png')) |
|  | plt.close() |
|  |  |
|  | plt.plot(history.history['loss'], marker='.') |
|  | plt.plot(history.history['val\_loss'], marker='.') |
|  | plt.title('model loss') |
|  | plt.xlabel('epoch') |
|  | plt.ylabel('loss') |
|  | plt.grid() |
|  | plt.legend(['loss', 'val\_loss'], loc='upper right') |
|  | plt.savefig(os.path.join(result\_dir, 'model\_loss.png')) |
|  | plt.close() |
|  |  |
|  |  |
|  | def save\_history(history, result\_dir): |
|  | loss = history.history['loss'] |
|  | acc = history.history['acc'] |
|  | val\_loss = history.history['val\_loss'] |
|  | val\_acc = history.history['val\_acc'] |
|  | nb\_epoch = len(acc) |
|  |  |
|  | with open(os.path.join(result\_dir, 'result.txt'), 'w') as fp: |
|  | fp.write('epoch\tloss\tacc\tval\_loss\tval\_acc\n') |
|  | for i in range(nb\_epoch): |
|  | fp.write('{}\t{}\t{}\t{}\t{}\n'.format( |
|  | i, loss[i], acc[i], val\_loss[i], val\_acc[i])) |
|  |  |
|  |  |
|  | def loaddata(video\_dir, vid3d, nclass, result\_dir, color=False, skip=True): |
|  | files = os.listdir(video\_dir) |
|  | X = [] |
|  | labels = [] |
|  | labellist = [] |
|  |  |
|  | pbar = tqdm(total=len(files)) |
|  |  |
|  | for filename in files: |
|  | pbar.update(1) |
|  | if filename == '.DS\_Store': |
|  | continue |
|  | name = os.path.join(video\_dir, filename) |
|  | label = vid3d.get\_UCF\_classname(filename) |
|  | if label not in labellist: |
|  | if len(labellist) >= nclass: |
|  | continue |
|  | labellist.append(label) |
|  | labels.append(label) |
|  | X.append(vid3d.video3d(name, color=color, skip=skip)) |
|  |  |
|  | pbar.close() |
|  | with open(os.path.join(result\_dir, 'classes.txt'), 'w') as fp: |
|  | for i in range(len(labellist)): |
|  | fp.write('{}\n'.format(labellist[i])) |
|  |  |
|  | for num, label in enumerate(labellist): |
|  | for i in range(len(labels)): |
|  | if label == labels[i]: |
|  | labels[i] = num |
|  | if color: |
|  | return np.array(X).transpose((0, 2, 3, 4, 1)), labels |
|  | else: |
|  | return np.array(X).transpose((0, 2, 3, 1)), labels |
|  |  |
|  |  |
|  | def main(): |
|  | parser = argparse.ArgumentParser( |
|  | description='simple 3D convolution for action recognition') |
|  | parser.add\_argument('--batch', type=int, default=128) |
|  | parser.add\_argument('--epoch', type=int, default=100) |
|  | parser.add\_argument('--videos', type=str, default='UCF101', |
|  | help='directory where videos are stored') |
|  | parser.add\_argument('--nclass', type=int, default=101) |
|  | parser.add\_argument('--output', type=str, required=True) |
|  | parser.add\_argument('--color', type=bool, default=False) |
|  | parser.add\_argument('--skip', type=bool, default=True) |
|  | parser.add\_argument('--depth', type=int, default=10) |
|  | args = parser.parse\_args() |
|  |  |
|  | img\_rows, img\_cols, frames = 32, 32, args.depth |
|  | channel = 3 if args.color else 1 |
|  | fname\_npz = 'dataset\_{}\_{}\_{}.npz'.format( |
|  | args.nclass, args.depth, args.skip) |
|  |  |
|  | vid3d = videoto3d.Videoto3D(img\_rows, img\_cols, frames) |
|  | nb\_classes = args.nclass |
|  | if os.path.exists(fname\_npz): |
|  | loadeddata = np.load(fname\_npz) |
|  | X, Y = loadeddata["X"], loadeddata["Y"] |
|  | else: |
|  | x, y = loaddata(args.videos, vid3d, args.nclass, |
|  | args.output, args.color, args.skip) |
|  | X = x.reshape((x.shape[0], img\_rows, img\_cols, frames, channel)) |
|  | Y = np\_utils.to\_categorical(y, nb\_classes) |
|  |  |
|  | X = X.astype('float32') |
|  | np.savez(fname\_npz, X=X, Y=Y) |
|  | print('Saved dataset to dataset.npz.') |
|  | print('X\_shape:{}\nY\_shape:{}'.format(X.shape, Y.shape)) |
|  |  |
|  | # Define model |
|  | model = Sequential() |
|  | model.add(Conv3D(32, kernel\_size=(3, 3, 3), input\_shape=( |
|  | X.shape[1:]), border\_mode='same')) |
|  | model.add(Activation('relu')) |
|  | model.add(Conv3D(32, kernel\_size=(3, 3, 3), border\_mode='same')) |
|  | model.add(Activation('softmax')) |
|  | model.add(MaxPooling3D(pool\_size=(3, 3, 3), border\_mode='same')) |
|  | model.add(Dropout(0.25)) |
|  |  |
|  | model.add(Conv3D(64, kernel\_size=(3, 3, 3), border\_mode='same')) |
|  | model.add(Activation('relu')) |
|  | model.add(Conv3D(64, kernel\_size=(3, 3, 3), border\_mode='same')) |
|  | model.add(Activation('softmax')) |
|  | model.add(MaxPooling3D(pool\_size=(3, 3, 3), border\_mode='same')) |
|  | model.add(Dropout(0.25)) |
|  |  |
|  | model.add(Flatten()) |
|  | model.add(Dense(512, activation='sigmoid')) |
|  | model.add(Dropout(0.5)) |
|  | model.add(Dense(nb\_classes, activation='softmax')) |
|  |  |
|  | model.compile(loss=categorical\_crossentropy, |
|  | optimizer=Adam(), metrics=['accuracy']) |
|  | model.summary() |
|  | plot\_model(model, show\_shapes=True, |
|  | to\_file=os.path.join(args.output, 'model.png')) |
|  |  |
|  | X\_train, X\_test, Y\_train, Y\_test = train\_test\_split( |
|  | X, Y, test\_size=0.2, random\_state=43) |
|  |  |
|  | history = model.fit(X\_train, Y\_train, validation\_data=(X\_test, Y\_test), batch\_size=args.batch, |
|  | epochs=args.epoch, verbose=1, shuffle=True) |
|  | model.evaluate(X\_test, Y\_test, verbose=0) |
|  | model\_json = model.to\_json() |
|  | if not os.path.isdir(args.output): |
|  | os.makedirs(args.output) |
|  | with open(os.path.join(args.output, 'ucf101\_3dcnnmodel.json'), 'w') as json\_file: |
|  | json\_file.write(model\_json) |
|  | model.save\_weights(os.path.join(args.output, 'ucf101\_3dcnnmodel.hd5')) |
|  |  |
|  | loss, acc = model.evaluate(X\_test, Y\_test, verbose=0) |
|  | print('Test loss:', loss) |
|  | print('Test accuracy:', acc) |
|  | plot\_history(history, args.output) |
|  | save\_history(history, args.output) |
|  |  |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | main() |

//2DCNN

|  |
| --- |
|  |
|  | import numpy as np |
|  | import matplotlib  import argparse |
|  | matplotlib.use('Agg') |
|  | import matplotlib.pyplot as plt |
|  | import os |
|  | import videoto3d |
|  | from sklearn.cross\_validation import train\_test\_split |
|  | from tqdm import tqdm |
|  |  |
|  | from keras.datasets import cifar10 |
|  | from keras.models import Sequential |
|  | from keras.layers import Dense, Dropout, Activation, Flatten |
|  | from keras.layers import Convolution2D, MaxPooling2D |
|  | from keras.utils import np\_utils |
|  | from keras.utils.vis\_utils import plot\_model |
|  |  |
|  |  |
|  | def plot\_history(history, result\_dir): |
|  | plt.plot(history.history['acc'], marker='.') |
|  | plt.plot(history.history['val\_acc'], marker='.') |
|  | plt.title('model accuracy') |
|  | plt.xlabel('epoch') |
|  | plt.ylabel('accuracy') |
|  | plt.grid() |
|  | plt.legend(['acc', 'val\_acc'], loc='lower right') |
|  | plt.savefig(os.path.join(result\_dir, 'model\_accuracy.png')) |
|  | plt.close() |
|  |  |
|  | plt.plot(history.history['loss'], marker='.') |
|  | plt.plot(history.history['val\_loss'], marker='.') |
|  | plt.title('model loss') |
|  | plt.xlabel('epoch') |
|  | plt.ylabel('loss') |
|  | plt.grid() |
|  | plt.legend(['loss', 'val\_loss'], loc='upper right') |
|  | plt.savefig(os.path.join(result\_dir, 'model\_loss.png')) |
|  | plt.close() |
|  |  |
|  |  |
|  | def save\_history(history, result\_dir): |
|  | loss = history.history['loss'] |
|  | acc = history.history['acc'] |
|  | val\_loss = history.history['val\_loss'] |
|  | val\_acc = history.history['val\_acc'] |
|  | nb\_epoch = len(acc) |
|  |  |
|  | with open(os.path.join(result\_dir, 'result.txt'), 'w') as fp: |
|  | fp.write('epoch\tloss\tacc\tval\_loss\tval\_acc\n') |
|  | for i in range(nb\_epoch): |
|  | fp.write('{}\t{}\t{}\t{}\t{}\n'.format( |
|  | i, loss[i], acc[i], val\_loss[i], val\_acc[i])) |
|  |  |
|  |  |
|  | def loaddata(video\_dir, vid3d, nclass, result\_dir): |
|  | files = os.listdir(video\_dir) |
|  | X = [] |
|  | labels = [] |
|  | labellist = [] |
|  |  |
|  | pbar = tqdm(total=len(files)) |
|  |  |
|  | for filename in files: |
|  | if filename == '.DS\_Store': |
|  | continue |
|  | name = os.path.join(video\_dir, filename) |
|  | #print('loading video:{}'.format(name)) |
|  | label = vid3d.get\_UCF\_classname(filename) |
|  | # print('label:{}'.format(label)) |
|  | if label not in labellist: |
|  | if len(labellist) >= nclass: |
|  | continue |
|  | labellist.append(label) |
|  | labels.append(label) |
|  | X.append(vid3d.video3d(name)) |
|  | pbar.update(1) |
|  |  |
|  | pbar.close() |
|  | with open(os.path.join(result\_dir, 'classes.txt'), 'w') as fp: |
|  | for i in range(len(labellist)): |
|  | fp.write('{}\n'.format(labellist[i])) |
|  |  |
|  | for num, label in enumerate(labellist): |
|  | for i in range(len(labels)): |
|  | if label == labels[i]: |
|  | labels[i] = num |
|  | return np.array(X), labels |
|  |  |
|  |  |
|  | def main(): |
|  | parser = argparse.ArgumentParser(description='2D convolution') |
|  | parser.add\_argument('--batch', type=int, default=128) |
|  | parser.add\_argument('--epoch', type=int, default=100) |
|  | parser.add\_argument('--videos', type=str, default='videos', |
|  | help='directory where videos are stored') |
|  | parser.add\_argument('--nclass', type=int, default=101) |
|  | parser.add\_argument('--output', type=str, required=True) |
|  | args = parser.parse\_args() |
|  |  |
|  | img\_rows, img\_cols = 32, 32 |
|  |  |
|  | vid3d = videoto3d.Videoto3D(img\_rows, img\_cols, 1) |
|  | x, y = loaddata(args.videos, vid3d, args.nclass, args.output) |
|  | X = x.reshape(x.shape[0], img\_cols, img\_rows, 1) |
|  | nb\_classes = max(y) + 1 |
|  | Y = np\_utils.to\_categorical(y, nb\_classes) |
|  | X = X.astype('float32') |
|  | print('X shape:{}\nYshape:{}'.format(X.shape, Y.shape)) |
|  |  |
|  | # define model |
|  | model = Sequential() |
|  |  |
|  | model.add(Convolution2D(32, 3, 3, border\_mode='same', |
|  | input\_shape=X.shape[1:])) |
|  | model.add(Activation('relu')) |
|  | model.add(Convolution2D(32, 3, 3)) |
|  | model.add(Activation('relu')) |
|  | model.add(MaxPooling2D(pool\_size=(2, 2))) |
|  | model.add(Dropout(0.25)) |
|  |  |
|  | model.add(Convolution2D(64, 3, 3, border\_mode='same')) |
|  | model.add(Activation('relu')) |
|  | model.add(Convolution2D(64, 3, 3)) |
|  | model.add(Activation('relu')) |
|  | model.add(MaxPooling2D(pool\_size=(2, 2))) |
|  | model.add(Dropout(0.25)) |
|  |  |
|  | model.add(Flatten()) |
|  | model.add(Dense(512)) |
|  | model.add(Activation('relu')) |
|  | model.add(Dropout(0.5)) |
|  | model.add(Dense(nb\_classes)) |
|  | model.add(Activation('softmax')) |
|  |  |
|  | model.compile(loss='categorical\_crossentropy', |
|  | optimizer='adam', |
|  | metrics=['accuracy']) |
|  | model.summary() |
|  | plot\_model(model, show\_shapes=True, to\_file=os.path.join(args.output, 'model.png')) |
|  |  |
|  | X\_train, X\_test, Y\_train, Y\_test = train\_test\_split( |
|  | X, Y, test\_size=0.2, random\_state=4) |
|  |  |
|  | history = model.fit(X\_train, Y\_train, |
|  | batch\_size=args.batch, |
|  | nb\_epoch=args.epoch, |
|  | validation\_data=(X\_test, Y\_test), |
|  | shuffle=True) |
|  | model\_json = model.to\_json() |
|  | with open(os.path.join(args.output, 'ucf101cnnmodel.json'), 'w') as json\_file: |
|  | json\_file.write(model\_json) |
|  | model.save\_weights(os.path.join(args.output, 'ucf101cnnmodel.hd5')) |
|  |  |
|  | loss, acc = model.evaluate(X\_test, Y\_test, verbose=0) |
|  | print('Test loss:', loss) |
|  | print('Test accuracy:', acc) |
|  | plot\_history(history, args.output) |
|  | save\_history(history, args.output) |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | main() |