案例七：基于C3D的行为识别

1 案例目的

• 理解C3D的应用场景场景；

• 掌握C3D算法。

2 案例内容

使用C3D进行时空特征学习，用UCF101(一个中等规模的数据集)进行实验，寻找最佳架构，来对数据集视频中进行行为识别

3 案例知识点

• Python语言编程

• 行为识别网络架构

• C3D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）提取特征，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，与目前最好的手工特征进行对比；

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；

我们在UCF101数据集上评估C3D特征。该数据集包含101个人类动作类别的13320个视频。我们使用此数据集提供的三个分割设置。

该网络以视频片段作为输入，预测属于101种不同动作的类别标签。所有视频帧的大小都调整为128 × 171。这大约是UCF101帧的一半分辨率。视频被分割成不重叠的16帧剪辑，然后作为网络的输入。输入尺寸为3 × 16 × 128 × 171。我们还使用抖动，在训练期间使用大小为3 × 16 × 112 × 112的输入剪辑的随机作物。

7.2训练分类模型

我们提取C3D特征，输入到一个多类线性支持向量机训练模型。我们使用3种不同的网络对C3D描述子进行实验:在I380K上训练的C3D，在Sports-1M上训练的C3D，以及在I380K上训练并在Sports-1M上进行调整的C3D。在多重网络设置中，我们连接这些网络的l2 -标准化C3D描述符。使用30个片段的小批量从头开始训练网络，初始学习率为0.003。学习速率每4个时点后除以10。训练在16个时代之后停止。

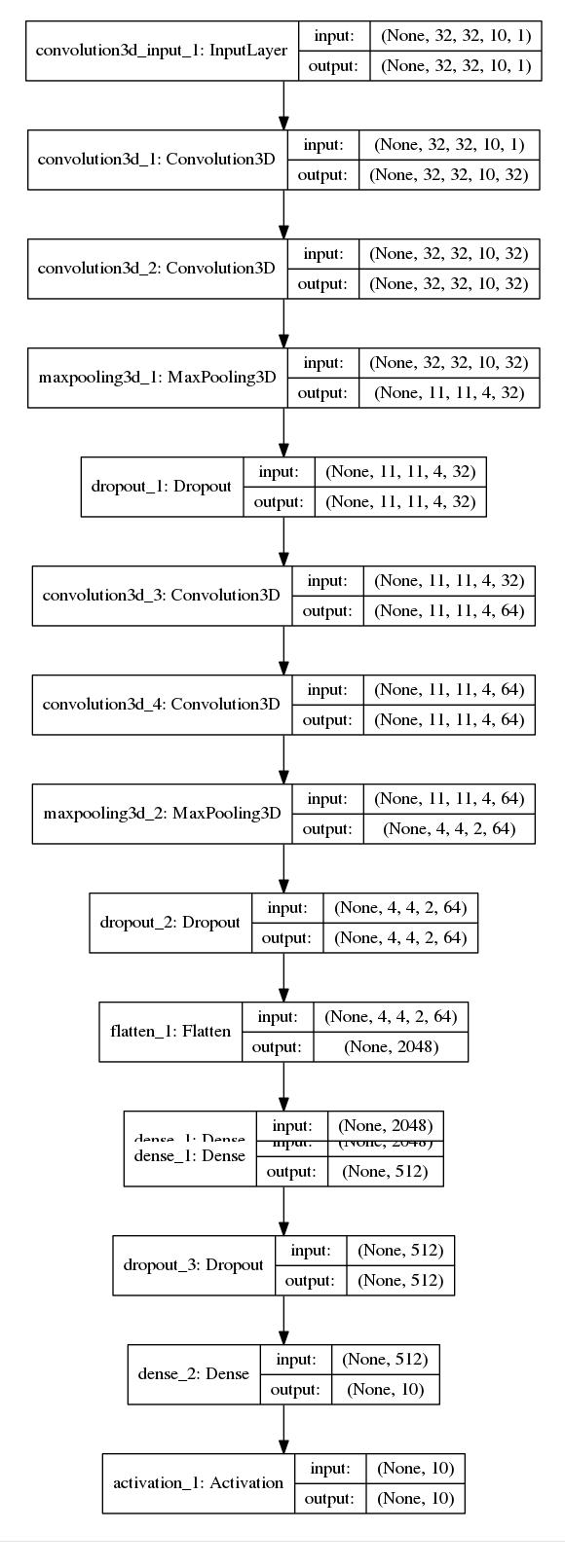


图1 模型

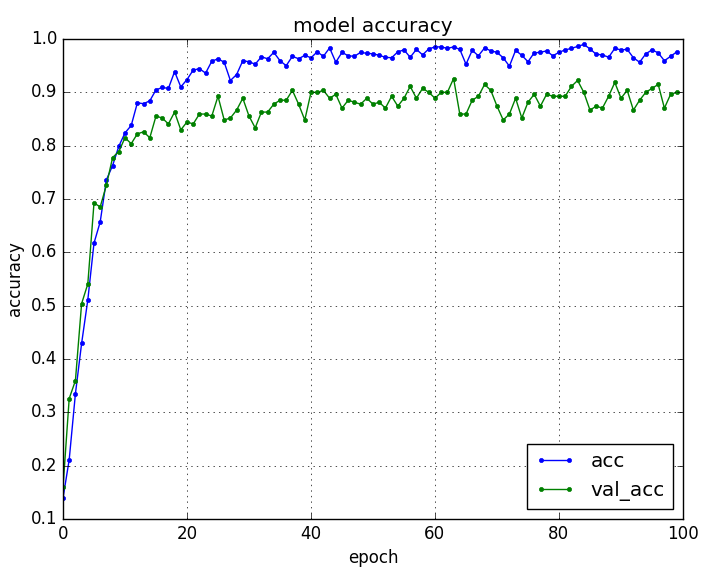


图2 模型准确率

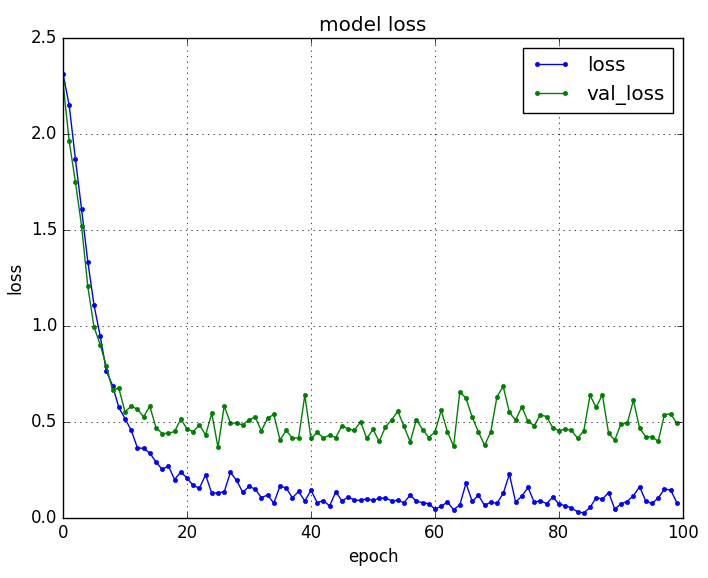


图3 模型损失

7.3与其他手工特征做对比分析

我们将C3D特征与几个基线进行比较:目前最好的手工特征，即改进的密集轨迹(iDT)和常用的深度图像特征，即Imagenet，使用Caffe的Imagenet预处理模型。对于iDT，我们对iDT的每个特征通道(轨迹、HOG、HOF、MBHx和MBHy)使用单词袋表示，码本大小为5000。我们使用L1-norm分别对每个通道的直方图进行归一化，并将这些归一化直方图拼接成一个视频的25K特征向量。对于Imagenet基线，类似于C3D，我们对每一帧提取Imagenet fc6特征，平均这些帧特征，制作视频描述符。多类线性支持向量机也用于这两个基线，以公平的比较。

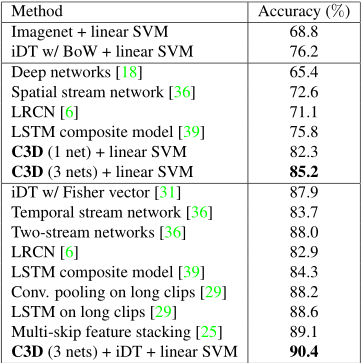


表3.UCF101动作识别结果。C3D与基线和当前最先进的方法的比较。顶部:简单特征与线性支持向量机;中间:只将RGB帧作为输入的方法;底部:使用多个特征组合的方法。

表3给出了C3D与两种基线和目前最佳方法的动作识别精度对比。上面的部分显示了两个基线的结果。中间部分给出了只使用RGB帧作为输入的方法。下面的部分报告了使用所有可能的特征组合(例如光流，iDT)的所有当前最好的方法（所有结果均为视频级精度）。

C3D微调网在前面描述的三种C3D网络中表现最好。然而，这三种网之间的性能差距很小(1%)。从现在开始，除非另有说明，我们将微调网络称为C3D。C3D使用一个只有4096个尺寸的网获得了82.3%的精度。带有3个网的C3D将精度提高到85.2%，尺寸增加到12288。当C3D与iDT结合时，准确率进一步提高到90.4%，而当与Imagenet结合时，我们只观察到0.6%的提高。这说明C3D可以很好地捕获外观和运动信息，因此与基于外观的深度特征Imagenet结合没有好处。另一方面，C3D和iDT具有很强的互补性，结合起来是有益的。事实上，iDT是基于光流跟踪和底层梯度直方图手工制作的特征，而C3D则捕捉高层抽象/语义信息。

3网C3D达到85.2%，分别比iDT和Imagenet基线好9%和16.4%。在唯一的RGB输入设置下，与基于cnn的算法相比，我们的C3D算法比深度网络和空间流网络分别高出19.8%和12.6%。深度网络和空间流网络都采用了AlexNet架构。网络是根据Sports-1M的预训练模型进行微调的，空间流网络是根据Imagenet的预训练模型进行微调的。我们的C3D在网络架构和基本操作方面不同于这些基于cnn的方法。此外，C3D是训练体育- 1m和使用没有任何微调。与基于递归神经网络(RNN)的方法相比，C3D算法比长期递归卷积网络(LRCN)和LSTM复合模型的性能分别高出14.1%和9.4%。只有RGB输入的C3D在同时使用光流和RGB以及中的时序流网络时，仍然优于这两种基于rnn的方法。然而，C3D需要与iDT结合，才能超过双流网络、其他基于iDT的方法，以及专注于长期建模的方法。与其他方法相比，C3D除了具有良好的数值前景外，还具有简单的优点。

8 案例代码

import argparse

import os

import matplotlib

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()

#使用量标准化添加新模型

import argparse

import os

import matplotlib

matplotlib.use('AGG')

import matplotlib.pyplot as plt

import numpy as npåå

from keras.datasets import cifar10

from keras.layers import (Activation, Conv3D, Dense, Dropout, Flatten,

MaxPooling3D, BatchNormalization)

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:]), padding="same"))

model.add(Activation('relu'))

model.add(Conv3D(32, padding="same", kernel\_size=(3, 3, 3)))

model.add(Activation('relu'))

model.add(MaxPooling3D(pool\_size=(3, 3, 3), padding="same"))

model.add(Dropout(0.25))

model.add(Conv3D(64, padding="same", kernel\_size=(3, 3, 3)))

model.add(Activation('relu'))

model.add(Conv3D(64, padding="same", kernel\_size=(3, 3, 3)))

model.add(Activation('relu'))

model.add(MaxPooling3D(pool\_size=(3, 3, 3), padding="same"))

model.add(Dropout(0.25))

model.add(Conv3D(64, padding="same", kernel\_size=(3, 3, 3)))

model.add(Activation('relu'))

model.add(Conv3D(64, padding="same", kernel\_size=(3, 3, 3)))

model.add(Activation('relu'))

model.add(MaxPooling3D(pool\_size=(3, 3, 3), padding="same"))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation='relu'))

model.add(BatchNormalization())

model.add(Dropout(0.5))

model.add(Dense(nb\_classes, activation='softmax'))

model.compile(loss=categorical\_crossentropy,

optimizer='rmsprop', 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()