Neural Network Project Report 2

Jiahui Gao
April 2, 2020

0 What Have Been Done This time

0.1 Data set

I check the video clip one by one and surprisingly find that some of them are not the same as the label at all. So I trimmed the data by select some of videos in the original HMDB51 dataset. And this video may or may not have this properties:

1 the video is filmed from a camera which has a fixed view. (It's reasonable that the home monitor system capture the video from a video camera set at a fixed angle.)

2 people in this video shows most of their body. (Its's reasonable that the camera has a proper distance from the people.)

0.2 Preprocessing

I use the same down sample method to sample the data into a fixed size (100,100, 11) and we use these 11 frames to generate 10 frames of optical flow. The optical flow data of size (100, 100, 10) is used to be the input data for the neural network. Since the data size remain the same, the structure of neural network doesn't have to be change (of course this can be a very important method to improve the performance of the system). By using the optical follow, the impact of the background in the video is undermined. I think the better the optical flow is the better performance can be obtained. By doing this, the classifier starts to work rather than last time it didn't work for the test video at all.

1 Topic

In this project I build a neural network and train it for the action recognition task. The dataset for training is HMDB51.

2 Dataset

HMDB51 is a dataset collected from various sources, mostly from movies, and a small proportion from public databases such as the Prelinger archive, YouTube and Google videos. The dataset contains 6849 clios distributed in 51 action categories, Each containing at least 101 clips. The actions categories can be grouped in five types:

1 general facial action smile, laugh, chew, talk.

2 Facial actions with object manipulation: smoke, eat, drink.

3 General body movements: cartwheel, clap hands, climb, etc.

4 Body movements with object interaction: brush hair, catch, draw sword, etc.

5 Body movements for human interaction: fencing, hug, kiss, etc.

My goal is to detect jump in the video and is classified as general body movements. In this case, some of the video of general body movements will be used to train the neural network. For this week's training, only jump and walk has been used for train the neural network.

I check the video clip one by one and surprisingly find that some of them are not the same as the label at all. So I trimmed the data by select some of videos in the original HMDB51 dataset. And this video may or may not have this properties:

1 the video is filmed from a camera which has a fixed view. (It's reasonable that the home monitor system capture the video from a video camera set at a fixed angle.)

2 people in this video shows most of their body. (Its's reasonable that the camera has a proper distance from the people.)

3 DNN Model

In this project, I would like to design a 3D neural network for this action recognition task since I think neural network with a 3D convolutional kernel is intuitive for video processing. For the first week, the main goal is to make sure all the part of this project is working and the video data can be processed step by step as I want.

3.1 Preprocess

The video clips for training quite short and its reasonable to sample it into same size so that the the training can be done. In our case, the colored video is transferred into the gray scale video and we uniformly sample 10 frames for each video and the resolution is resized to 100 by 100. So for each video, the input data is of size (100,100,10,1).

We have 150 videos from both classes, jump and walk, and they are sampled and resized in the desired size. So the entire dataset is of size (300,100,100,10,1) and there will be a split for training and validation.

3.2 3D-CNN Model

Input data is of shape (300,100,100,10,1)
Output data is a vector shows the binary classification result of shape (2,1)
Shape of each layer

- Output of convolution layer: (96,96,6)
- Output of the pooling layer:(32,32,2)
- Flatten layer: (1536,1)
- Fully connect layer:(128,1)

4 Hyperparameters

The hyperparameter used in the training process:
Batch Size 2
Epochs 15
Dropout 0.3

5 Annotated Code

Import the necessary libraries

[1]:
```
import cv2
import math
import matplotlib.pyplot as plt
import pandas as pd

from keras.preprocessing import image
import numpy as np
from keras.utils import np_utils
from skimage.transform import resize
from keras.models import Sequential
from keras.layers import Dense, Flatten,Dropout,Conv3D, MaxPooling3D,Activation
from keras.layers.convolutional import Convolution3D, MaxPooling3D
import os

from sklearn.model_selection import train_test_split
```

```
Using TensorFlow backend.
```

[2]:
```
img_row = 100
img_col = 100
img_depth = 10
```

Read and preprocess of the training video

```
X_tr = []
a = 0
listing = os.listdir('videofortraining/jump/')
for vid in listing:
    vid = 'videofortraining/jump/'+vid
    frames = []
    newDimension = (img_row,img_col)
    cap = cv2.VideoCapture(vid)

    ret,first_frame = cap.read()
    first_frame = cv2.resize(first_frame, newDimension, interpolation = cv2.INTER_AREA)
    prev_gray = cv2.cvtColor(first_frame,cv2.COLOR_BGR2GRAY)
    hsv = np.zeros_like(first_frame)
    hsv[...,1] = 255

    frameRate = cap.get(7)/(img_depth)
    #print(frameRate)
    newDimension = (img_row,img_col)
    a = a + frameRate
    frame_list = []

    for i in range(img_depth):
        f = math.floor((i + 1) * frameRate)
        frame_list.append(f)
    frame_list.append(int(cap.get(7)-2))

    while (cap.isOpened()):
        frameId = cap.get(1)
        ret, frame = cap.read()
        if (ret != True):
            break
        if frameId in frame_list:
            frame = cv2.resize(frame, newDimension, interpolation = cv2.INTER_AREA)
            gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
            flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None, 0.5, 3, 15, 3, 5, 1.2, 0)
            mag,ang = cv2.cartToPolar(flow[...,0],flow[...,1])
            hsv[...,0] = ang*180/np.pi/2
            hsv[...,2] = cv2.normalize(mag,None,0,255,cv2.NORM_MINMAX)
            flow = cv2.cvtColor(hsv,cv2.COLOR_HSV2BGR)
            flow = cv2.cvtColor(flow,cv2.COLOR_BGR2GRAY)
            frames.append(flow)
            prev_gray = gray

    cap.release()
    cv2.destroyAllWindows()

    input=np.array(frames)

    ipt=np.rollaxis(np.rollaxis(input,2,0),2,0)
    print(ipt.shape)
    X_tr.append(ipt)
```

Hyperparameters

```
X_tr = np.array(X_tr)
print(X_tr.shape)
```

```
(230, 100, 100, 10)
```

```
num_sample = len(X_tr)

label = np.ones((num_sample))
label[0:115] = 0
label[115:230] = 1
print(label)

train_data = [X_tr, label]

(X_train, Y_train) = (train_data[0], train_data[1])

train_set = np.zeros((num_sample, 1, img_row, img_col, img_depth))

for sample in range(num_sample):
    train_set[sample][0][:][:][:] = X_train[sample][:][:][:]

patch_size = 10
#print(train_set.shape)

batch_size = 1
num_class = 2
num_epoch = 15

Y_train = np_utils.to_categorical(Y_train, num_class)
#print(Y_train)

num_filter = [32, 32]
num_pooling = [3, 3]
num_conv = [5, 5]

train_set = train_set.astype('float32')
train_set -= np.mean(train_set)
train_set /= np.max(train_set)
```

Model

```
model = Sequential()
model.add(Convolution3D(data_format = 'channels_first', filters = num_filter[0], kernel_size = (5, 5, 5), input_shape = (1, img_row, img_col, img_
model.add(MaxPooling3D(pool_size = [3, 3, 3]))
model.add(Dropout(0.5))
model.add(Flatten())

model.add(Dense(128, init = 'normal', activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(2, init = 'normal'))
model.add(Activation('softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer = 'RMSprop', metrics = ['accuracy'])
```

```
/home/jasongao/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7: UserWarning: Update your `Dense` call to the Keras 2 API:
`Dense(128, activation="relu", kernel_initializer="normal")`
  import sys
/home/jasongao/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:9: UserWarning: Update your `Dense` call to the Keras 2 API:
`Dense(2, kernel_initializer="normal")`
  if __name__ == '__main__':
```

```
X_train_new, X_val_new, Y_train_new, Y_val_new = train_test_split(train_set, Y_train, test_size = 0.2)
```

```
hist = model.fit(X_train_new, Y_train_new, validation_data = (X_val_new, Y_val_new), batch_size = batch_size, nb_epoch = num_epoch, shuffle = Tr
```

Save the result and plot the curve

```
model_json = model.to_json()
with open("model.json","w") as json_file:
    json_file.write(model_json)

model.save_weights("model_w.json")
```

```
train_loss=hist.history['loss']
val_loss=hist.history['val_loss']
train_acc=hist.history['accuracy']
val_acc=hist.history['val_accuracy']
xc=range(num_epoch)

plt.figure(1,figsize=(7,5))
plt.plot(xc,train_loss)
plt.plot(xc,val_loss)
plt.xlabel('num of Epochs')
plt.ylabel('loss')
plt.title('train_loss vs val_loss')
plt.grid(True)
plt.legend(['train','val'])
#print plt.style.available # use bmh, classic,ggplot for big pictures
plt.style.use(['classic'])

plt.figure(2,figsize=(7,5))
plt.plot(xc,train_acc)
plt.plot(xc,val_acc)
plt.xlabel('num of Epochs')
plt.ylabel('accuracy')
plt.title('train_acc vs val_acc')
plt.grid(True)
plt.legend(['train','val'],loc=4)
#print plt.style.available # use bmh, classic,ggplot for big pictures
plt.style.use(['classic'])
```

Load the test video and give a classification curve over time

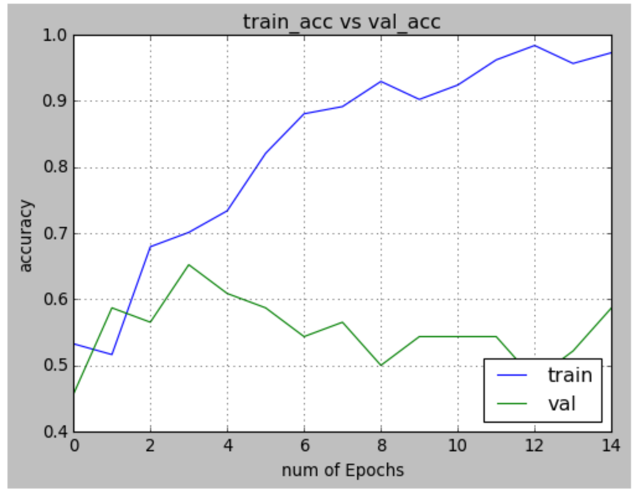
```
for video in range(5):
    vid = 'video_filmed/' + str(video) +'.MP4'
#for i in range(1):
#    vid = 'video_filmed/' + str(i) +'.MP4'
    X_TEST = []
    frames = []
    count = 0
    cap = cv2.VideoCapture(vid)
    ret,first_frame = cap.read()
    first_frame = cv2.resize(first_frame, newDimension, interpolation = cv2.INTER_
AREA)
    prev_gray = cv2.cvtColor(first_frame,cv2.COLOR_BGR2GRAY)
    hsv = np.zeros_like(first_frame)
    hsv[...,1] = 255

    frameRate = round(cap.get(5))
    num_frame = round(cap.get(7))
    #print(frameRate)
    #print(num_frame)

    newDimension = (img_row,img_col)
    sample_rate = 0.2 * frameRate

    s = 0
    sample_frame = []
    sample_frame.append(0)
    while (s <= num_frame - 1 - img_depth * sample_rate):
        s += round(sample_rate)
        sample_frame.append(s)

    #print(sample_frame)
```

```
    while (cap.isOpened()):
        frameId = cap.get(1)
        ret, frame = cap.read()
        if (ret != True):
            break
        if frameId in sample_frame:
            frame = cv2.resize(frame, newDimension, interpolation = cv2.INTER_AREA
)
            gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
            flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None, 0.5, 3, 15,
 3, 5, 1.2, 0)
            mag,ang = cv2.cartToPolar(flow[...,0],flow[...,1])
            hsv[...,0] = ang*180/np.pi/2
            hsv[...,2] = cv2.normalize(mag,None,0,255,cv2.NORM_MINMAX)
            flow = cv2.cvtColor(hsv,cv2.COLOR_HSV2BGR)
            flow = cv2.cvtColor(flow,cv2.COLOR_BGR2GRAY)
            frames.append(flow)
            #plt.figure()
            #plt.imshow(flow)
            prev_gray = gray
            count +=1

    cap.release()
    cv2.destroyAllWindows()

    frames = np.array(frames)

    #print(frames.shape)



    for i in range(count-img_depth):
        frame_pack = []
        for j in range(img_depth):
            frame_pack.append(frames[i+j][:][:])
        X_TEST.append(frame_pack)

       #f = np.array(frame_pack)
       #print(f.shape)

    X_TEST = np.array(X_TEST)
    X_TEST = np.rollaxis(np.rollaxis(X_TEST,3,1),3,1)
    #print(X_TEST.shape)

    TEST_set = np.zeros((count-img_depth,1,img_row,img_col,img_depth))

    for sample in range(count-img_depth):
        TEST_set[sample][0][:][:][:] = X_TEST[sample][:][:][:]


    prediction = model.predict(TEST_set)
    time = []
    prob = []
    time_label = []
    for i in range(count-img_depth):
        time.append(i*0.2)
        prob.append(prediction[i][0])
        time_label.append([i*0.2,prediction[i][0]])
    plt.figure()
    plt.plot(time,prob)
    with open('time_label_op' + str(video),'w') as f:
        json.dump(str(time_label),f)
```

6 Training and Testing Performance

Train on 184 samples, validate on 46 samples

Epoch 1/15
184/184 [==============================] - 57s 311ms/step - loss: 1.1536 - accuracy: 0.5326 - val_loss: 1.6904 - val_accuracy: 0.4565
Epoch 2/15
184/184 [==============================] - 57s 310ms/step - loss: 0.9574 - accuracy: 0.5163 - val_loss: 0.7503 - val_accuracy: 0.5870
Epoch 3/15
184/184 [==============================] - 57s 309ms/step - loss: 0.8319 - accuracy: 0.6793 - val_loss: 0.9058 - val_accuracy: 0.5652
Epoch 4/15
184/184 [==============================] - 57s 310ms/step - loss: 0.8117 - accuracy: 0.7011 - val_loss: 0.8780 - val_accuracy: 0.6522
Epoch 5/15
184/184 [==============================] - 57s 310ms/step - loss: 0.6756 - accuracy: 0.7337 - val_loss: 1.2986 - val_accuracy: 0.6087
Epoch 6/15
184/184 [==============================] - 58s 313ms/step - loss: 0.5898 - accuracy: 0.8207 - val_loss: 2.2406 - val_accuracy: 0.5870
Epoch 7/15
184/184 [==============================] - 57s 308ms/step - loss: 0.5212 - accuracy: 0.8804 - val_loss: 2.3896 - val_accuracy: 0.5435
Epoch 8/15
184/184 [==============================] - 57s 309ms/step - loss: 0.4453 - accuracy: 0.8913 - val_loss: 2.8918 - val_accuracy: 0.5652
Epoch 9/15
184/184 [==============================] - 57s 309ms/step - loss: 0.4390 - accuracy: 0.9293 - val_loss: 3.4549 - val_accuracy: 0.5000
Epoch 10/15
184/184 [==============================] - 57s 309ms/step - loss: 0.4828 - accuracy: 0.9022 - val_loss: 4.3352 - val_accuracy: 0.5435
Epoch 11/15
184/184 [==============================] - 57s 308ms/step - loss: 0.4770 - accuracy: 0.9239 - val_loss: 3.3740 - val_accuracy: 0.5435
Epoch 12/15
184/184 [==============================] - 57s 308ms/step - loss: 0.1675 - accuracy: 0.9620 - val_loss: 3.8184 - val_accuracy: 0.5435
Epoch 13/15
184/184 [==============================] - 57s 308ms/step - loss: 0.0777 - accuracy: 0.9837 - val_loss: 4.8502 - val_accuracy: 0.4783
Epoch 14/15
184/184 [==============================] - 57s 308ms/step - loss: 0.1340 - accuracy: 0.9565 - val_loss: 5.2807 - val_accuracy: 0.5217
Epoch 15/15
184/184 [==============================] - 57s 308ms/step - loss: 0.1699 - accuracy: 0.9728 - val_loss: 4.7831 - val_accuracy: 0.5870



7 Instruction on how to test the trained DNN

Install Dpendencies:

- python 3
- Keras
- Tensorflow
- Anaconda