Neural Network Project Report

Jiahui Gao
March 2020

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.

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 pyperparameter used in the training process:
Batch Size 2
Epochs 50
Dropout 0.5

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

[4]:

```
X_tr = []
listing = os.listdir('videofortraining/jump/')
for vid in listing:
    vid = 'videofortraining/jump/'+vid
    frames = []
    cap = cv2.VideoCapture(vid)
    frameRate = cap.get(7)/(img_depth - 1)
    newDimension = (img_row,img_col)

    frame_list = []
    frame_list.append(0)
    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)
            r, g, b = frame[:,:,0], frame[:,:,1], frame[:,:,2]
            gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
            frames.append(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

```
num_sample = len(X_tr)

label = np.ones((num_sample))
label[0:150] = 0
label[150:300] = 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 = 2
num_class = 2
num_epoch = 50

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

```
X_TEST = []
frames = []
count = 0
listing = os.listdir('video_filmed/')
for vid in listing:
    vid = 'video_filmed/'+vid
    frames = []
    cap = cv2.VideoCapture(vid)
    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 += 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)
            r, g, b = frame[:,:,0], frame[:,:,1], frame[:,:,2]
            gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
            frames.append(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_classes(TEST_set)
```

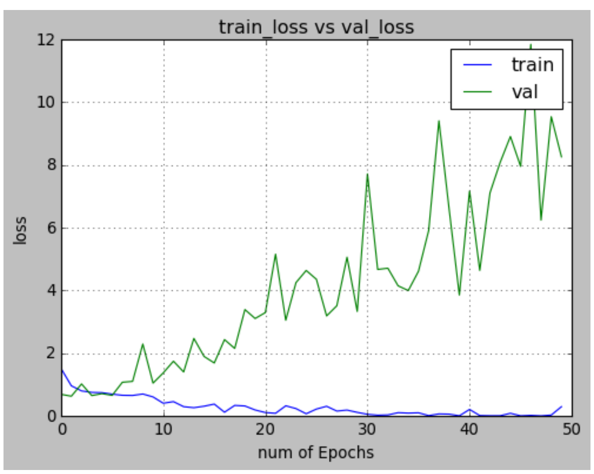
```
plt.plot(prediction)
```

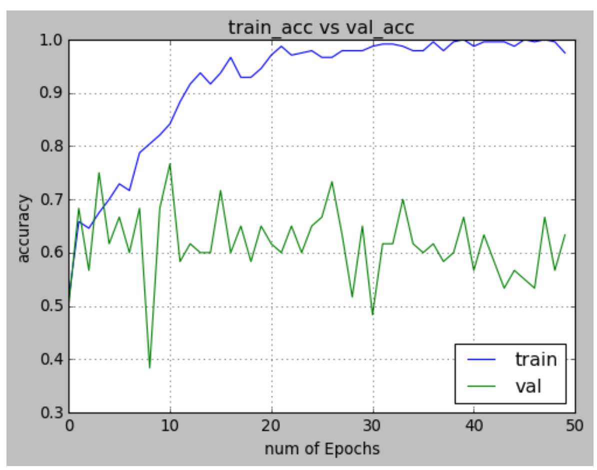
6 Training and Testing Performance

```
Train on 240 samples, validate on 60 samples
Epoch 1/50
240/240 [==============================] - 66s 277ms/step - loss: 1.5075 - accuracy: 0.5125 - val_loss: 0.6865
- val_accuracy: 0.5000
Epoch 2/50
240/240 [==============================] - 66s 275ms/step - loss: 0.9586 - accuracy: 0.6583 - val_loss: 0.6266
- val_accuracy: 0.6833
Epoch 3/50
240/240 [==============================] - 66s 274ms/step - loss: 0.7959 - accuracy: 0.6458 - val_loss: 1.0179
- val_accuracy: 0.5667
Epoch 4/50
240/240 [==============================] - 67s 277ms/step - loss: 0.7487 - accuracy: 0.6750 - val_loss: 0.6473
- val_accuracy: 0.7500
Epoch 5/50
240/240 [==============================] - 84s 349ms/step - loss: 0.7405 - accuracy: 0.7000 - val_loss: 0.7077
- val_accuracy: 0.6167
Epoch 6/50
240/240 [==============================] - 86s 357ms/step - loss: 0.6936 - accuracy: 0.7292 - val_loss: 0.6541
- val_accuracy: 0.6667
Epoch 7/50
240/240 [==============================] - 85s 356ms/step - loss: 0.6565 - accuracy: 0.7167 - val_loss: 1.0720
- val_accuracy: 0.6000
Epoch 8/50
240/240 [==============================] - 82s 343ms/step - loss: 0.6507 - accuracy: 0.7875 - val_loss: 1.0999
- val_accuracy: 0.6833
Epoch 9/50
240/240 [==============================] - 86s 358ms/step - loss: 0.6973 - accuracy: 0.8042 - val_loss: 2.2906
- val_accuracy: 0.3833
Epoch 10/50
240/240 [==============================] - 86s 358ms/step - loss: 0.6045 - accuracy: 0.8208 - val_loss: 1.0445
- val_accuracy: 0.6833
Epoch 11/50
240/240 [==============================] - 86s 356ms/step - loss: 0.4027 - accuracy: 0.8417 - val_loss: 1.3697
- val_accuracy: 0.7667
Epoch 12/50
```

```
240/240 [==============================] - 86s 357ms/step - loss: 0.4546 - accuracy: 0.8833 - val_loss: 1.7435
- val_accuracy: 0.5833
Epoch 13/50
240/240 [==============================] - 86s 357ms/step - loss: 0.2934 - accuracy: 0.9167 - val_loss: 1.3997
- val_accuracy: 0.6167
Epoch 14/50
240/240 [==============================] - 86s 357ms/step - loss: 0.2608 - accuracy: 0.9375 - val_loss: 2.4698
- val_accuracy: 0.6000
Epoch 15/50
240/240 [==============================] - 84s 351ms/step - loss: 0.3080 - accuracy: 0.9167 - val_loss: 1.8943
- val_accuracy: 0.6000
Epoch 16/50
240/240 [==============================] - 69s 288ms/step - loss: 0.3762 - accuracy: 0.9375 - val_loss: 1.6865
- val_accuracy: 0.7167
Epoch 17/50
240/240 [==============================] - 66s 275ms/step - loss: 0.1167 - accuracy: 0.9667 - val_loss: 2.4315
- val_accuracy: 0.6000
Epoch 18/50
240/240 [==============================] - 66s 275ms/step - loss: 0.3344 - accuracy: 0.9292 - val_loss: 2.1497
- val_accuracy: 0.6500
Epoch 19/50
240/240 [==============================] - 66s 275ms/step - loss: 0.3158 - accuracy: 0.9292 - val_loss: 3.3882
- val_accuracy: 0.5833
Epoch 20/50
240/240 [==============================] - 66s 275ms/step - loss: 0.1881 - accuracy: 0.9458 - val_loss: 3.1039
- val_accuracy: 0.6500
Epoch 21/50
240/240 [==============================] - 66s 275ms/step - loss: 0.1067 - accuracy: 0.9708 - val_loss: 3.2927
- val_accuracy: 0.6167
Epoch 22/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0825 - accuracy: 0.9875 - val_loss: 5.1549
- val_accuracy: 0.6000
Epoch 23/50
240/240 [==============================] - 66s 276ms/step - loss: 0.3215 - accuracy: 0.9708 - val_loss: 3.0484
- val_accuracy: 0.6500
Epoch 24/50
240/240 [==============================] - 66s 275ms/step - loss: 0.2325 - accuracy: 0.9750 - val_loss: 4.2478
- val_accuracy: 0.6000
Epoch 25/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0675 - accuracy: 0.9792 - val_loss: 4.6353
- val_accuracy: 0.6500
Epoch 26/50
240/240 [==============================] - 66s 275ms/step - loss: 0.2170 - accuracy: 0.9667 - val_loss: 4.3540
- val_accuracy: 0.6667
Epoch 27/50
240/240 [==============================] - 66s 275ms/step - loss: 0.3063 - accuracy: 0.9667 - val_loss: 3.1856
- val_accuracy: 0.7333
Epoch 28/50
240/240 [==============================] - 66s 275ms/step - loss: 0.1535 - accuracy: 0.9792 - val_loss: 3.5095
- val_accuracy: 0.6333
Epoch 29/50
240/240 [==============================] - 66s 276ms/step - loss: 0.1846 - accuracy: 0.9792 - val_loss: 5.0589
- val_accuracy: 0.5167
Epoch 30/50
240/240 [==============================] - 66s 275ms/step - loss: 0.1124 - accuracy: 0.9792 - val_loss: 3.3284
- val_accuracy: 0.6500
Epoch 31/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0500 - accuracy: 0.9875 - val_loss: 7.7134
- val_accuracy: 0.4833
Epoch 32/50
240/240 [==============================] - 68s 282ms/step - loss: 0.0214 - accuracy: 0.9917 - val_loss: 4.6674
- val_accuracy: 0.6167
Epoch 33/50
240/240 [==============================] - 69s 288ms/step - loss: 0.0305 - accuracy: 0.9917 - val_loss: 4.7075
- val_accuracy: 0.6167
Epoch 34/50
240/240 [==============================] - 66s 275ms/step - loss: 0.1022 - accuracy: 0.9875 - val_loss: 4.1461
- val_accuracy: 0.7000
Epoch 35/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0887 - accuracy: 0.9792 - val_loss: 3.9944
- val_accuracy: 0.6167
Epoch 36/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0989 - accuracy: 0.9792 - val_loss: 4.6120
- val_accuracy: 0.6000
Epoch 37/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0080 - accuracy: 0.9958 - val_loss: 5.9138
- val_accuracy: 0.6167
Epoch 38/50
```

```
240/240 [==============================] - 66s 275ms/step - loss: 0.0628 - accuracy: 0.9792 - val_loss: 9.4067
- val_accuracy: 0.5833
Epoch 39/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0555 - accuracy: 0.9958 - val_loss: 6.5859
- val_accuracy: 0.6000
Epoch 40/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 3.8519
- val_accuracy: 0.6667
Epoch 41/50
240/240 [==============================] - 66s 275ms/step - loss: 0.2057 - accuracy: 0.9875 - val_loss: 7.1671
- val_accuracy: 0.5667
Epoch 42/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0100 - accuracy: 0.9958 - val_loss: 4.6357
- val_accuracy: 0.6333
Epoch 43/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0036 - accuracy: 0.9958 - val_loss: 7.1082
- val_accuracy: 0.5833
Epoch 44/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0041 - accuracy: 0.9958 - val_loss: 8.0833
- val_accuracy: 0.5333
Epoch 45/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0827 - accuracy: 0.9875 - val_loss: 8.9059
- val_accuracy: 0.5667
Epoch 46/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 7.9528
- val_accuracy: 0.5500
Epoch 47/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0158 - accuracy: 0.9958 - val_loss:
11.8426 - val_accuracy: 0.5333
Epoch 48/50
240/240 [==============================] - 66s 275ms/step - loss: 3.5186e-04 - accuracy: 1.0000 - val_loss:
6.2398 - val_accuracy: 0.6667
Epoch 49/50
240/240 [==============================] - 66s 275ms/step - loss: 0.0257 - accuracy: 0.9958 - val_loss: 9.5364
- val_accuracy: 0.5667
Epoch 50/50
240/240 [==============================] - 66s 275ms/step - loss: 0.2942 - accuracy: 0.9750 - val_loss: 8.2571
- val_accuracy: 0.6333
```





7 Instruction on how to test the trained DNN

Install Dpendencies:

- python 3
- Keras
- Tensorflow
- Anaconda