Neural Network Project Report

Jiahui Gao
May 13, 2020

1 Topic
In this project I build a neural network and train it for the action recognition task.

2 Dataset
New: The main reason for the model didn't work well is because the dataset is not suitable for this project. The HMDB51 dataset is mainly video clips from movies, so the jump in this dataset is quite different with the jump for a ordinary people in our normal life. Another issue is that in the earlier work, I only use walk as the no jump class. The result of this choice is that the model can not recognize many other actions in the test video which are not jump. To fix these problem, I used the video clips filmed by my self which is more natural in daily life and contains more different actions. The result seems much better than before.

Old: 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 clips distributed in 51 action categories, and 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, etc.
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. To train the neural network for this project, jump is the target class and walk has been used for contract class.

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 these videos have the following 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. (It'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. The jump is a body movement expected to have a move in the vertical direction. So I think that the body movement should have a feature along the time axis. So the neural network with a 3D convolutional kernel is intuitive for video processing.

3.1 Preprocess
The video clips for training quite short and it's reasonable to sample it into same size so that the training can be done. In our case, the colored video is transferred into the gray scale video and we uniformly sample 16 frames for each video and the resolution is resized to 100 by 100.

I use the same down sample method to sample the data into a fixed size (100,100, 17) and we use these 17 frames to generate 16 frames of optical flow. The optical flow data of size (100, 100, 16) is used to be the input data for the neural network. The optical flow images are expected to emphasize the body movement in the video and ignore the background information which is fixed in the video selected. I think the quality of optical flow is crucial to the performance and the better the optical flow is the better performance should be obtained.

We have 114 videos from both classes, jump and walk, and they are sampled and resized in the desired shape. So the entire dataset is of size (228, 100 ,100 ,16 ,1) and there will be a split for training and validation.

3.2 3D-CNN Model

Input data is of shape (228, 100 ,100 ,16 ,1)
There are four 3D convolutional layer with the same parameter. 32 kernels of size (3, 3, 3) are in each 3D convolutional layer followed by a max-pooling layer with kernels of size (2, 2, 1) for the first two layers and (2, 2, 2) for the last two layers. Output data is a vector shows the binary classification result of shape (2,1)

Shape of each layer

Layer (type)	Output Shape	Param #
conv3d_73 (Conv3D)	(None, 98, 98, 14, 32)	896
max_pooling3d_69 (MaxPooling	(None, 49, 49, 14, 32)	0
conv3d_74 (Conv3D)	(None, 47, 47, 12, 32)	27680
max_pooling3d_70 (MaxPooling	(None, 23, 23, 12, 32)	0
conv3d_75 (Conv3D)	(None, 21, 21, 10, 32)	27680
max_pooling3d_71 (MaxPooling	(None, 10, 10, 5, 32)	0
conv3d_76 (Conv3D)	(None, 8, 8, 3, 32)	27680
max_pooling3d_72 (MaxPooling	(None, 4, 4, 1, 32)	0
dropout_52 (Dropout)	(None, 4, 4, 1, 32)	0
flatten_17 (Flatten)	(None, 512)	0
dense_52 (Dense)	(None, 512)	262656
dropout_53 (Dropout)	(None, 512)	0
dense_53 (Dense)	(None, 128)	65664
dropout_54 (Dropout)	(None, 128)	0
dense_54 (Dense)	(None, 2)	258
activation_17 (Activation)	(None, 2)	0

4 Hyperparameters

The hyperparameter used in the training process:
Epochs 16
Dropout 0.3 for the convolutional layers
Dropout 0.5 for the fully connected layers

5 Annotated Code

Import the necessary libraries

```
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 keras.models import model_from_json
import json
from sklearn.model_selection import train_test_split
```

```
img_row = 100
img_col = 100
img_depth = 16
newDimension = (img_row,img_col)
```

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)
```

```
(228, 100, 100, 16)
```

```
num_sample = len(X_tr)

label = np.ones((num_sample))
label[0:113] = 0
label[113:230] = 1
#print(label)

train_data = [X_tr,label]

(X_train,Y_train) = (train_data[0],train_data[1])
print(X_train.shape)
train_set = np.zeros((num_sample,img_row,img_col,img_depth,1))
print(train_set[0,:,:,:,0].shape)
for sample in range(num_sample):
    train_set[sample,:,:,:,0] = X_train[sample][:][:][:]

patch_size = 10
batch_size = 1
num_class = 2
num_epoch = 16

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

num_filter = [32,32,32,32,32]
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_last',filters = num_filter[0],kernel_size = (3,3,3),input_shape = (img
_row,img_col,img_depth,1),activation='relu'))
model.add(MaxPooling3D(pool_size = (2,2,1)))
model.add(Convolution3D(data_format = 'channels_last',filters = num_filter[1],kernel_size = (3,3,3),input_shape = (img
_row,img_col,img_depth,1),activation='relu'))
model.add(MaxPooling3D(pool_size = (2,2,1)))
model.add(Convolution3D(data_format = 'channels_last',filters = num_filter[2],kernel_size = (3,3,3),input_shape = (img
_row,img_col,img_depth,1),activation='relu'))
model.add(MaxPooling3D(pool_size = (2,2,2)))
model.add(Convolution3D(data_format = 'channels_last',filters = num_filter[0],kernel_size = (3,3,3),input_shape = (img
_row,img_col,img_depth,1),activation='relu'))
model.add(MaxPooling3D(pool_size = (2,2,2)))

model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(512,init = 'normal',activation = 'relu'))
model.add(Dropout(0.5))
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 = 'binary_crossentropy',optimizer = 'RMSprop',metrics = ['accuracy'])
model.summary()
```

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(1) +'.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

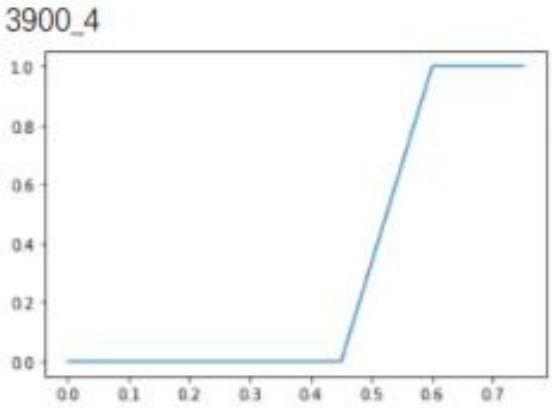
Train on 190 samples, validate on 48 samples
Epoch 1/16
190/190 [==============================] - 107s 564ms/step - loss: 0.8025 - accuracy: 0.6237 - val_loss: 0.4799 - val_accuracy: 0.7083
Epoch 2/16
190/190 [==============================] - 107s 562ms/step - loss: 0.4977 - accuracy: 0.8395 - val_loss: 0.4588 - val_accuracy: 0.8750
Epoch 3/16
190/190 [==============================] - 107s 565ms/step - loss: 0.3493 - accuracy: 0.8711 - val_loss: 0.3099 - val_accuracy: 0.8958
Epoch 4/16
190/190 [==============================] - 107s 563ms/step - loss: 0.2397 - accuracy: 0.9158 - val_loss: 0.5889 - val_accuracy: 0.9167
Epoch 5/16
190/190 [==============================] - 107s 563ms/step - loss: 0.3335 - accuracy: 0.9237 - val_loss: 0.2667 - val_accuracy: 0.9167
Epoch 6/16
190/190 [==============================] - 107s 563ms/step - loss: 0.2863 - accuracy: 0.9605 - val_loss: 0.2605 - val_accuracy: 0.9167
Epoch 7/16
190/190 [==============================] - 107s 563ms/step - loss: 0.3105 - accuracy: 0.9553 - val_loss: 0.2525 - val_accuracy: 0.8958
Epoch 8/16
190/190 [==============================] - 107s 564ms/step - loss: 0.2364 - accuracy: 0.9316 - val_loss: 0.5621 - val_accuracy: 0.8854
Epoch 9/16
190/190 [==============================] - 108s 567ms/step - loss: 0.3814 - accuracy: 0.9421 - val_loss: 0.5198 - val_accuracy: 0.8958
Epoch 10/16
190/190 [==============================] - 107s 565ms/step - loss: 0.3585 - accuracy: 0.9579 - val_loss: 0.4696 - val_accuracy: 0.9375
Epoch 11/16
190/190 [==============================] - 108s 567ms/step - loss: 0.2491 - accuracy: 0.9684 - val_loss: 0.4145 - val_accuracy: 0.9167
Epoch 12/16
190/190 [==============================] - 107s 563ms/step - loss: 0.1486 - accuracy: 0.9632 - val_loss: 0.6781 - val_accuracy: 0.9167
Epoch 13/16
190/190 [==============================] - 107s 564ms/step - loss: 0.0742 - accuracy: 0.9868 - val_loss: 0.8372 - val_accuracy: 0.9583
Epoch 14/16
190/190 [==============================] - 108s 566ms/step - loss: 0.0457 - accuracy: 0.9895 - val_loss: 4.8633 - val_accuracy: 0.8958
Epoch 15/16
190/190 [==============================] - 108s 570ms/step - loss: 0.3324 - accuracy: 0.9895 - val_loss: 1.4325 - val_accuracy: 0.9167
Epoch 16/16
190/190 [==============================] - 109s 573ms/step - loss: 0.4787 - accuracy: 0.9579 - val_loss: 1.0741 - val_accuracy: 0.9375

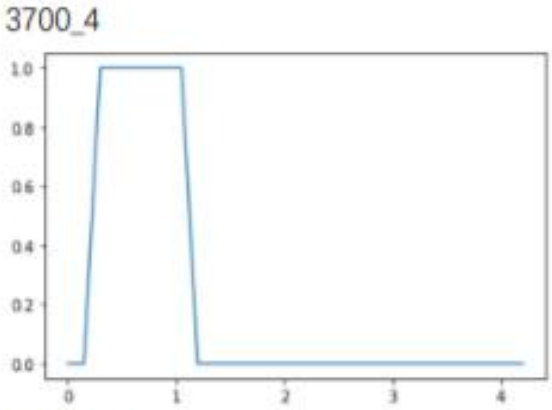
The accuracy for training and validation set is over 90% after 16 epochs training.

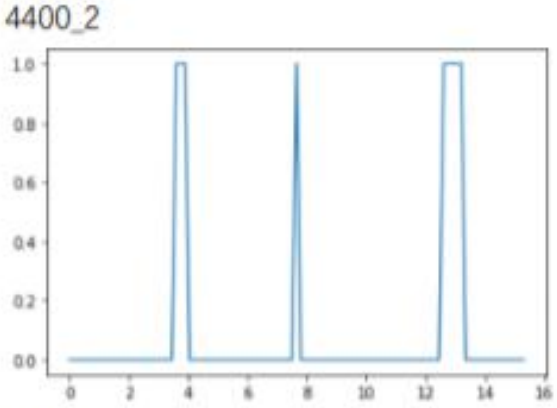
7 Compare of the New and Old Result on Test Videos

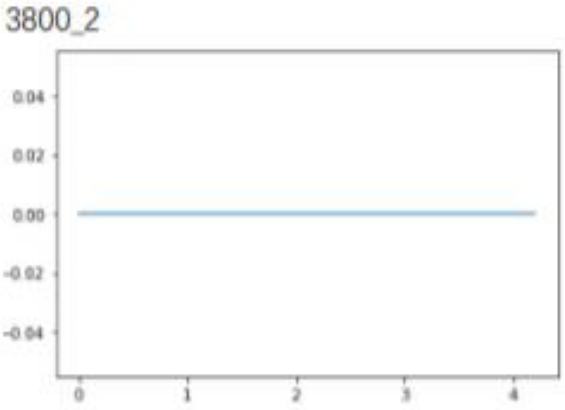
The result of old model is shown on the left while the result of new is on the right.

Old:

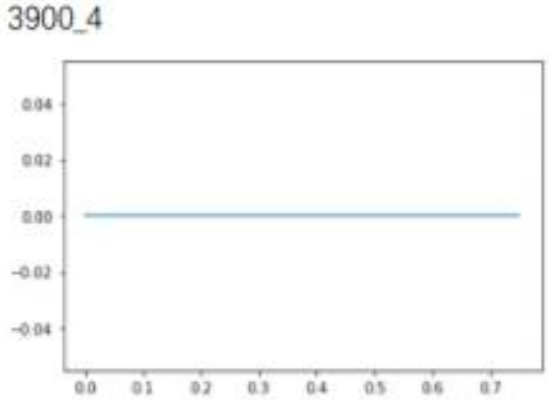


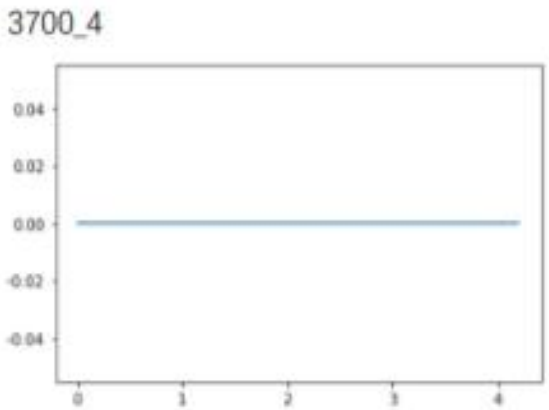


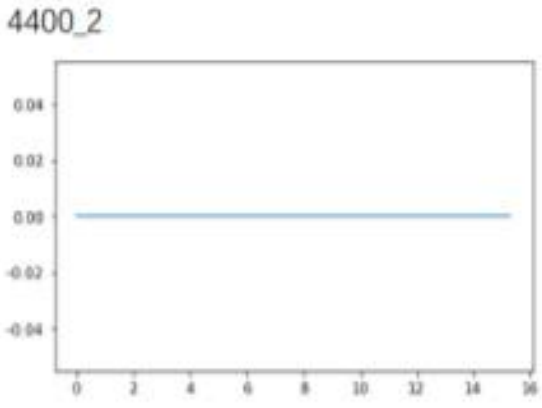


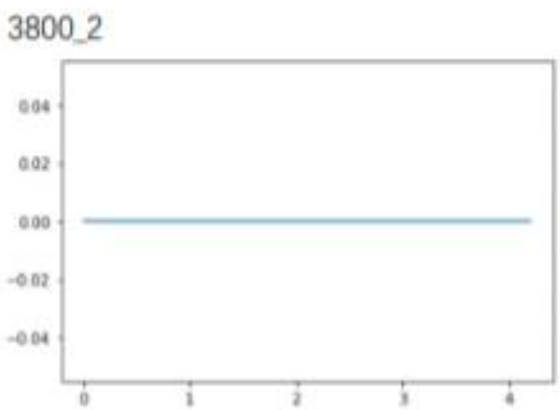


New:

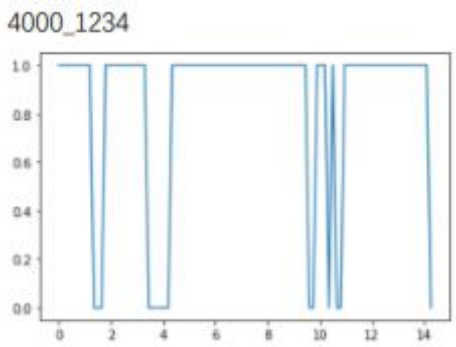


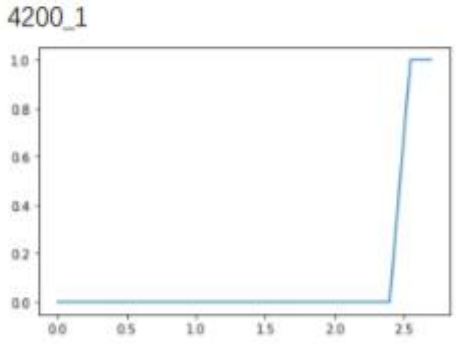


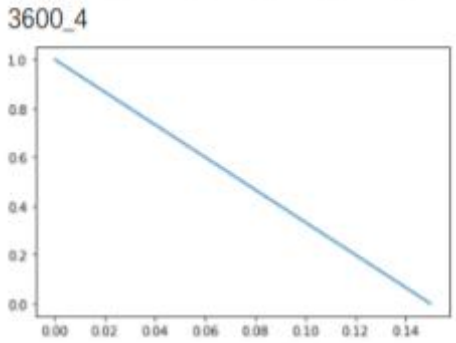


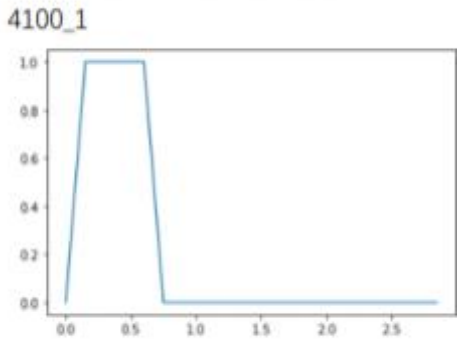


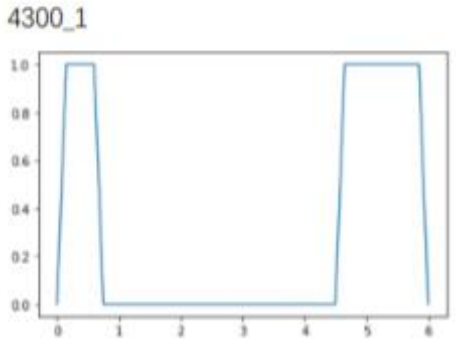
Old


4000_1234

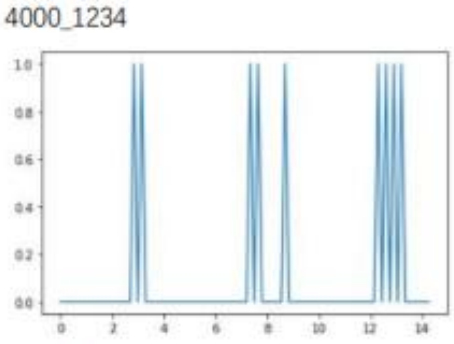

4200_1

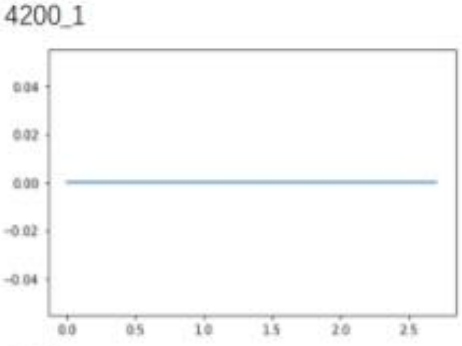

3600_4

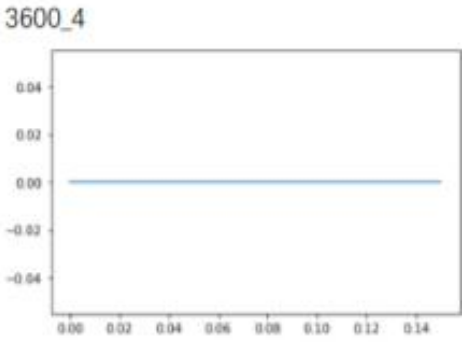

4100_1

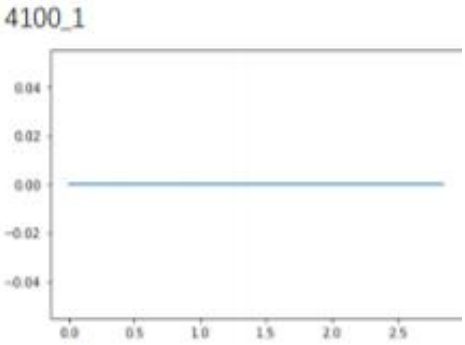

4300_1

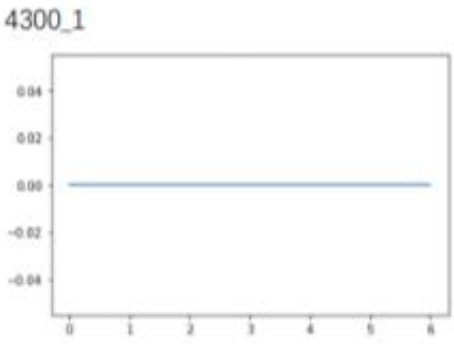
New


4000_1234


4200_1


3600_4


4100_1


4300_1

8 Instruction on how to test the trained DNN

Install Dependencies:

- python 3
- Keras
- Tensorflow
- Anaconda
- openCV

9 Future Work

If we have more time, we try the following method to improve the model.

1. Try a better optical flow estimation model to obtain the optical flow for the video.
2. Try to use the colored frames instead of gray scale frames.
3. Try to design a better structure of the neural network.