CSCE636 Project Part VIII

Name: Yiting Luo UIN: 427008285

PART I TOPIC

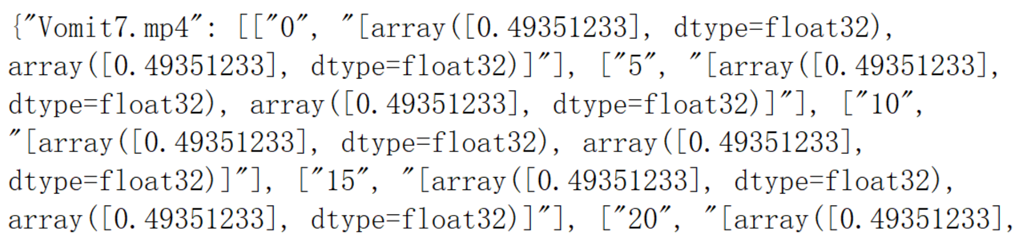
The purpose of the project is to train a neural network so it could be applied to video detection in the future smart home environment. The topic assigned to me is “detecting vomit”. The project is meaningful since vomiting is the symptom of many sudden illnesses.

PART II PREVIOUS EXISTING PROBLEMS

In the Part IV submission, the network is trained and the trained model is generated. The video used for testing the accuracy of the generated model is interpreted and each frame of the video is tested to get the probability of vomiting.

In the Part IV, although I could generate the model and predict the input video, the .json file is not correct.

As shown below, the probability of vomiting stays the same.

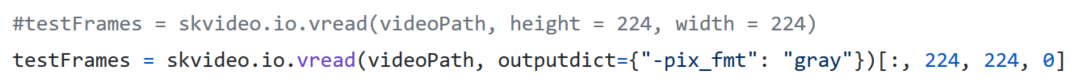


```
{"Vomit7.mp4": [["0", "[array([0.49351233], dtype=float32),
array([0.49351233], dtype=float32)]"], ["5", "[array([0.49351233],
dtype=float32), array([0.49351233], dtype=float32)]"], ["10",
"[array([0.49351233], dtype=float32), array([0.49351233],
dtype=float32)]"], ["15", "[array([0.49351233], dtype=float32),
array([0.49351233], dtype=float32)]"], ["20", "[array([0.49351233],
```

At first, I thought it’s because the video is too short and the differences between the frames is too subtle so the prediction stays the same. Then I change several input videos but the prediction stays the same.

To find out if I saved and loaded the trained model correctly, I change the epochs and learning rate of the training process to get a new model. The probability changes as I change the parameters but the probability of different frames still stays the same. Which indicates that the model is saved and loaded correctly.

After testing I found out there is a dimension mismatching while I was using the module skvideo (the code which caused error is shown below). I searched for the solution but it seems that skvideo is out of date and there is no more updates.



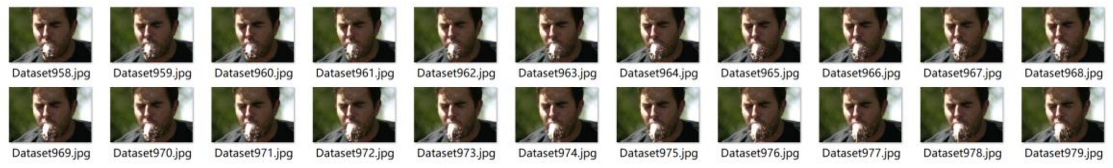
```
#testFrames = skvideo.io.vread(videoPath, height = 224, width = 224)
testFrames = skvideo.io.vread(videoPath, outputdict={"-pix_fmt": "gray"})[:, 224, 224, 0]
```

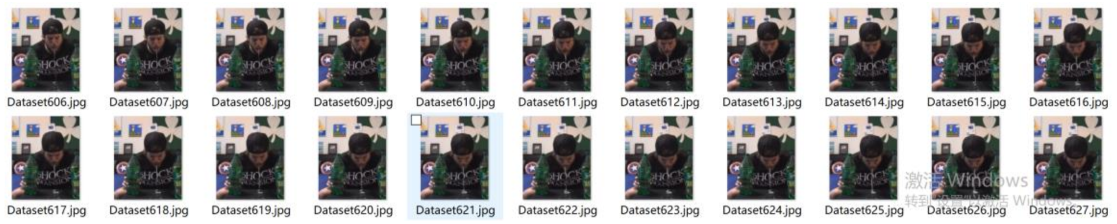
So one of the most important improvements in PART V I made is that I find out the problem which causes the wrong output .json file. After changing the video capture module to opencv2 and resizing the images I get, the json file seems correct for now.

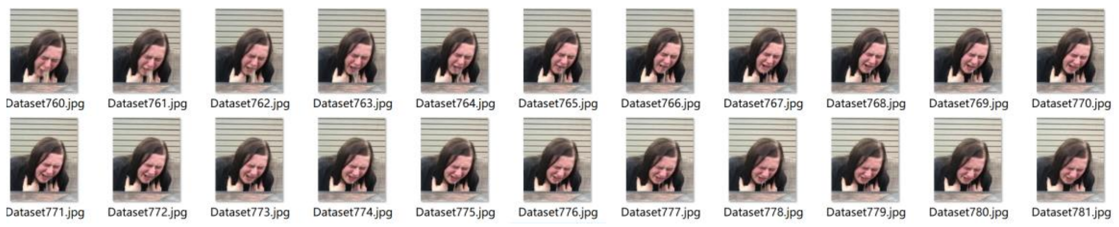
PART III DATASET

Since the topic is a little weird, I looked through the training dataset and I couldn't find related videos/images which I could make use of. I searched Youtube and most of the uploaded videos don't have any actual vomit scene. I managed to find several videos and transfer them into images. By sorting and labeling the images I get a preliminary dataset which could be used for training the network.

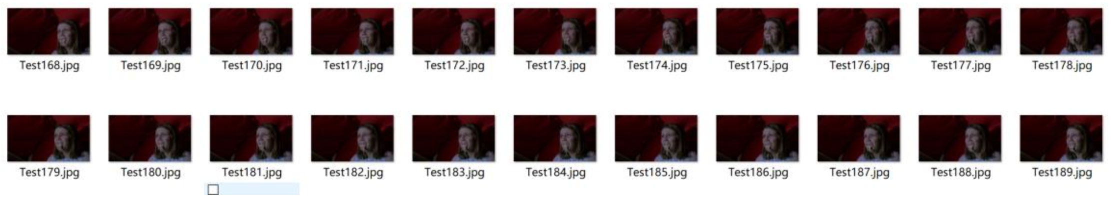
Shown below are parts of the images used for training in PART IV.







Shown below are parts of the images used for testing:

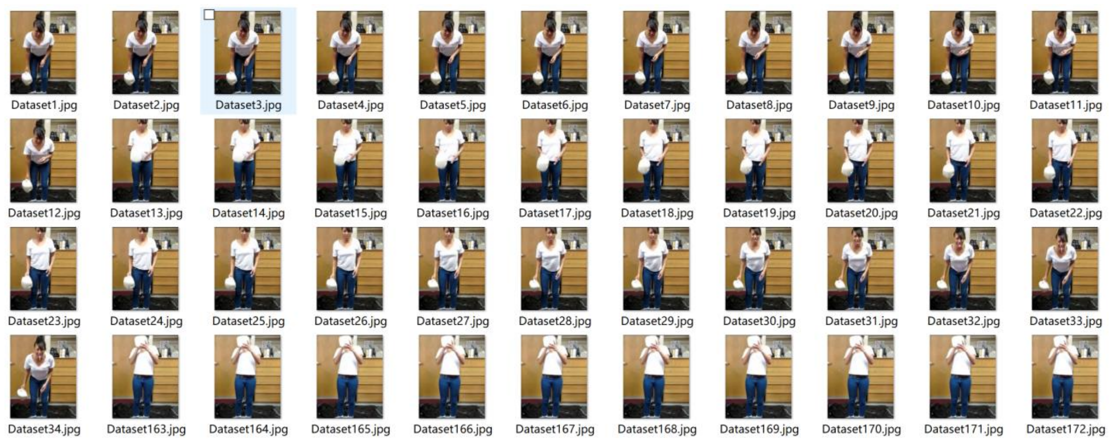


	Training Dataset	Testing Dataset
# of Vomit Images	805	168
# of Non-vomit Images	199	48

# of All Images	1004	216

The images used for training in the Part IV, most of them are vomiting faces. Since the cameras in the home may also capture the whole motion of the body, the dataset needs to be enlarged.

Shown below are parts of the images I added to the training dataset.



Since the images contain the whole body of a vomiting person, the trained network would be more accurate while dealing with videos captured by the cameras in a smart home.

By enlarging the training dataset, the prediction would be more accurate theoretically. However one of the main difficulties I met is that the related videos and images of vomiting is rare, since those data could cause people’s discomfortable.

PART IV ARCHITECTURE

The layers I use in this attempt are all dense layers. But in some cases, the model will over fit.

In the future improvements I will attempt using different layers and adding more training data to avoid over fitting.

Input shape of tensor:

X_train shape is (1004,32,32)

Y_train shape is (216,32,32)

Output shape of tensor:

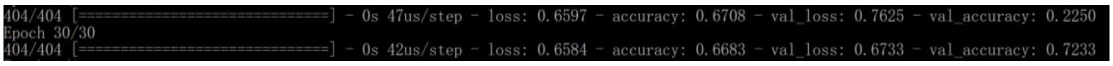
x_train shape is (1004,32,32)

y_train shape is (216,32,32)

Parameter Adjusting Attempt1.

The parameters I attempted to tune include epochs and batch size. The epoch I am using for optimization is 30. The batch size I am using is 256.

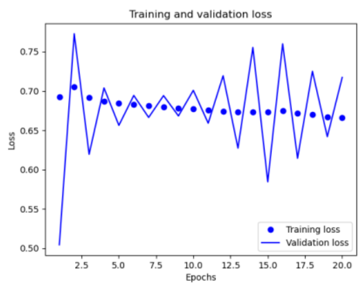
Testing performance is shown below:

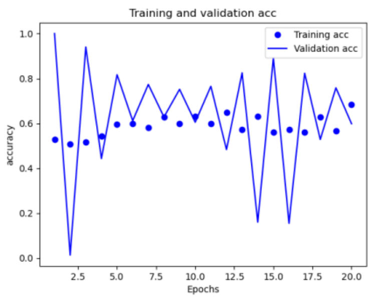


```
404/404 [==============================] - 0s 47us/step - loss: 0.6597 - accuracy: 0.6708 - val_loss: 0.7625 - val_accuracy: 0.2250
Epoch 30/30
404/404 [==============================] - 0s 42us/step - loss: 0.6584 - accuracy: 0.6683 - val_loss: 0.6733 - val_accuracy: 0.7233
```

The loss is 0.6733 and the accuracy is 72.33%.

Shown below is the accuracy and loss of the trained network.

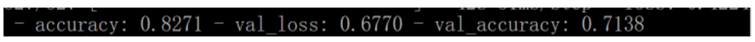




Parameter Adjusting Attempt2.

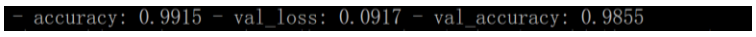
The parameters I attempted to tune include epochs and batch size. The epoch I am using for optimization is 30. The new batch size I am using is 64 instead of 32. The increasing of the batch size could also tune the performance.

The previous testing performance of the first epoch is shown below:



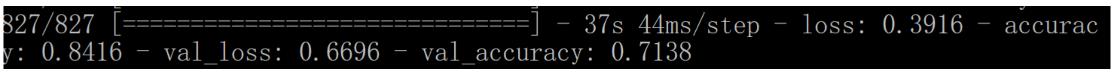
```
 - accuracy: 0.8271 - val_loss: 0.6770 - val_accuracy: 0.7138
```

The previous testing performance of the last (30th) epoch is shown below:



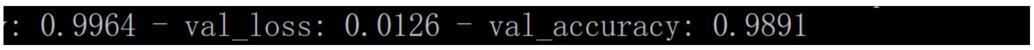
```
 - accuracy: 0.9915 - val_loss: 0.0917 - val_accuracy: 0.9855
```

As for now, the testing performance of the first epoch is shown below:



```
827/827 [==============================] - 37s 44ms/step - loss: 0.3916 - accurac
y: 0.8416 - val_loss: 0.6696 - val_accuracy: 0.7138
```

As for now, the testing performance of the last epoch is shown below:



```
: 0.9964 - val_loss: 0.0126 - val_accuracy: 0.9891
```

The accuracy is higher for now and the increasing of the batch size could make the training procedure much faster. Theoretically is the batch size is too large, the convergence could get worse but here the situation didn't appear. If the batch size is too small, the result could be worse.

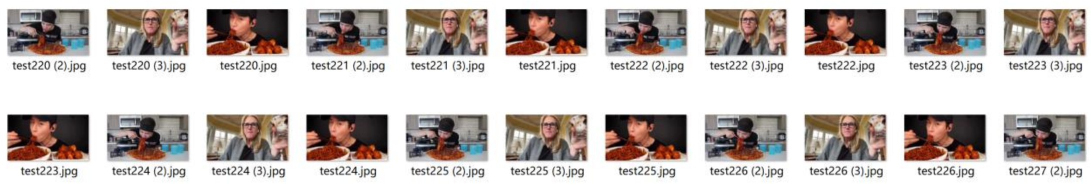
Another important parameter is the ratio of the vomiting/non-vomiting images. According to the paper, if the training data contains a large proportion of vomiting images, the model would tend to predict that the testing result would be vomit instead of predicting correctly.

Previously I generated the dataset directly from the input video and the ratio of the vomiting/non-vomiting images is not even close to 1:1. The proportion of non-vomiting images is much smaller. The paper's instruction include that I could copy the non-vomiting images to increase the proportion of the non-vomiting images.

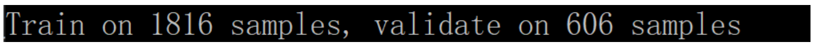
Parameter Adjusting Attempt3.

In order to enlarge the dataset and improve the accuracy of the model, I also add some negative examples which could be confused with vomiting. For example, eating and drinking are common activities which could be captured by cameras and those motions could be confused with vomiting.

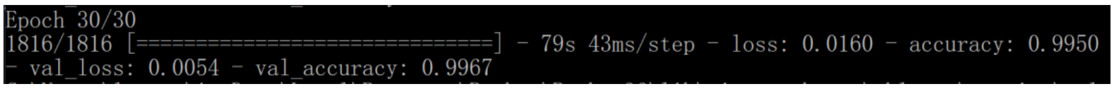
Shown below are some of the images that I added to the non-vomiting dataset.



The training result is shown below:

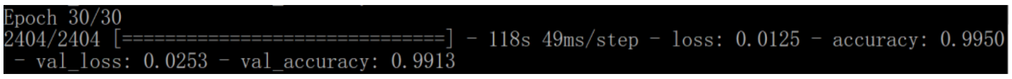


```
Train on 1816 samples, validate on 606 samples
```



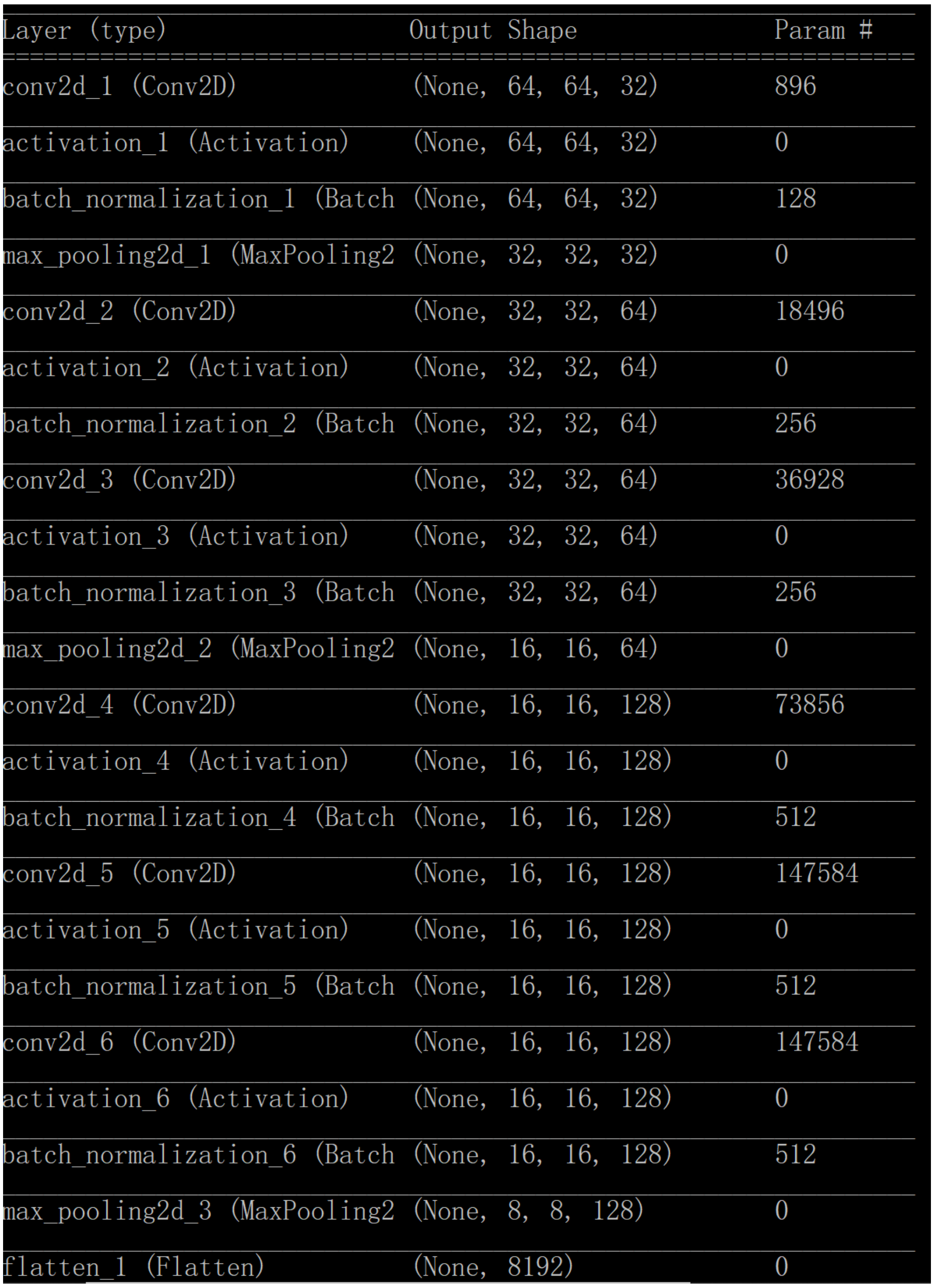
```
Epoch 30/30
1816/1816 [==============================] - 79s 43ms/step - loss: 0.0160 - accuracy: 0.9950
- val_loss: 0.0054 - val_accuracy: 0.9967
```

As for now, the ratio of the vomiting/non-vomiting images is 784:1638, which is not even close to 1:1. The proportion of vomiting images is much smaller. The paper's instruction include that I could copy the non-vomiting images to increase the proportion of the non-vomiting images. After balancing the ratio of vomiting/non-vomiting images, the performance is shown below.

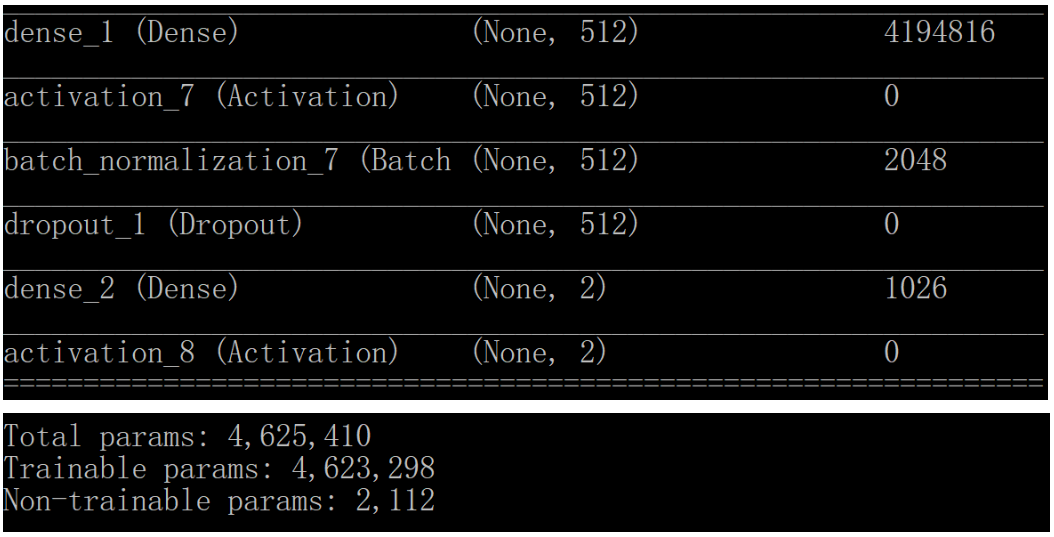


```
Epoch 30/30
2404/2404 [==============================] - 118s 49ms/step - loss: 0.0125 - accuracy: 0.9950
 - val_loss: 0.0253 - val_accuracy: 0.9913
```

The summary of the model looks like:



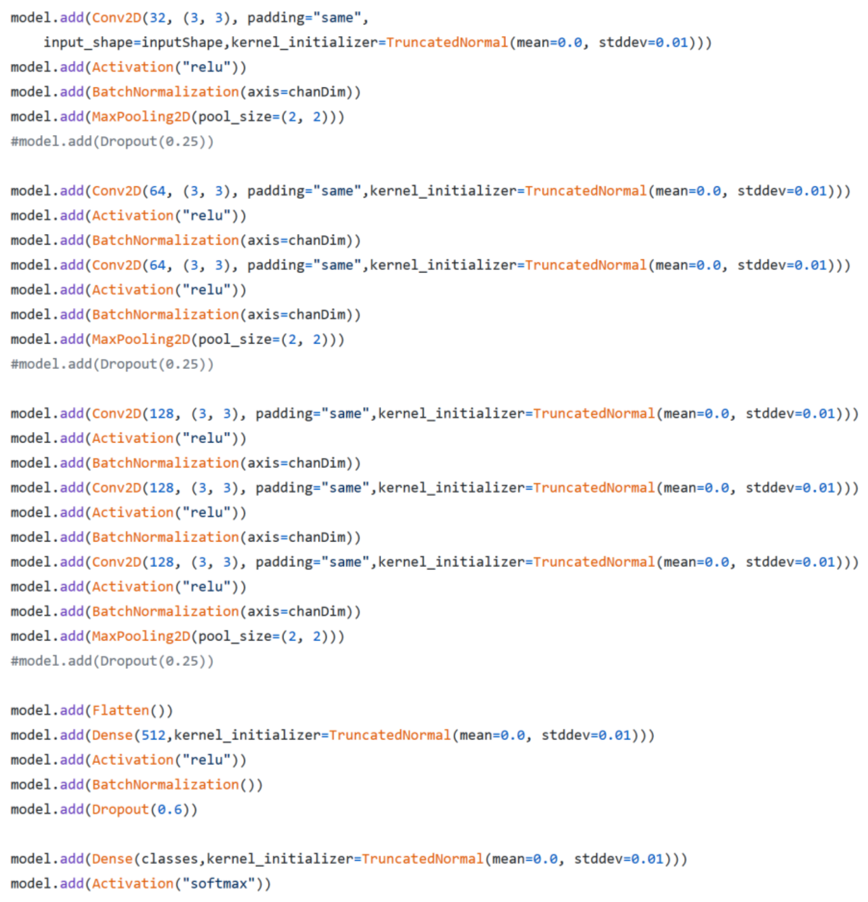
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 64, 64, 32)        896
_________________________________________________________________
activation_1 (Activation)    (None, 64, 64, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 64, 64, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
activation_4 (Activation)    (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 128)       147584
_________________________________________________________________
activation_5 (Activation)    (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_5 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 128)       147584
_________________________________________________________________
activation_6 (Activation)    (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_6 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
```



```
dense_1 (Dense)              (None, 512)               4194816
_________________________________________________________________
activation_7 (Activation)    (None, 512)               0
_________________________________________________________________
batch_normalization_7 (Batch (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 1026
_________________________________________________________________
activation_8 (Activation)    (None, 2)                 0
=================================================================
Total params: 4,625,410
Trainable params: 4,623,298
Non-trainable params: 2,112
```

PART V CODE

Shown below is the structure of the network:



```python
model.add(Conv2D(32, (3, 3), padding="same",
    input_shape=inputShape,kernel_initializer=TruncatedNormal(mean=0.0, stddev=0.01)))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
#model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding="same",kernel_initializer=TruncatedNormal(mean=0.0, stddev=0.01)))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same",kernel_initializer=TruncatedNormal(mean=0.0, stddev=0.01)))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
#model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), padding="same",kernel_initializer=TruncatedNormal(mean=0.0, stddev=0.01)))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(128, (3, 3), padding="same",kernel_initializer=TruncatedNormal(mean=0.0, stddev=0.01)))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(128, (3, 3), padding="same",kernel_initializer=TruncatedNormal(mean=0.0, stddev=0.01)))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
#model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512,kernel_initializer=TruncatedNormal(mean=0.0, stddev=0.01)))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.6))

model.add(Dense(classes,kernel_initializer=TruncatedNormal(mean=0.0, stddev=0.01)))
model.add(Activation("softmax"))
```

Shown below is the code used for training the network:

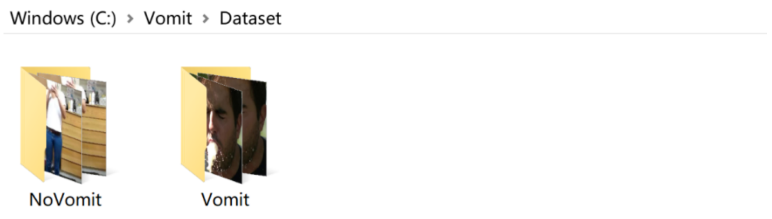
```
data = []
labels = []

imagePaths = sorted(list(getpath.list_images('C:\Vomit\Dataset')))
random.seed(42)
random.shuffle(imagePaths)


for imagePath in imagePaths:
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (64, 64))
    data.append(image)
    label = imagePath.split(os.path.sep)[-2]
    #print(label)
    labels.append(label)


data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)


(trainX, testX, trainY, testY) = train_test_split(data,labels, test_size=0.25, random_state=42)

lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)


aug = ImageDataGenerator(rotation_range=30, width_shift_range=0.1,
    height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
    horizontal_flip=True, fill_mode="nearest")

model = SimpleVGGNet.build(width=64, height=64, depth=3,classes=len(lb.classes_))

INIT_LR = 0.01
EPOCHS = 30
BS = 32

opt = SGD(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="sparse_categorical_crossentropy", optimizer=opt,metrics=["accuracy"])
'''
H = model.fit_generator(aug.flow(trainX, trainY, batch_size=BS),
    validation_data=(testX, testY), steps_per_epoch=len(trainX)//BS,
    epochs=EPOCHS)
'''

H = model.fit(trainX, trainY, validation_data=(testX, testY),
    epochs=EPOCHS, batch_size=32)

predictions = model.predict(testX, batch_size=32)
print(classification_report(testY.argmax(axis=1),
    predictions.argmax(axis=1), target_names=lb.classes_))

N = np.arange(0, EPOCHS)
plt.style.use("ggplot")
plt.figure()
plt.plot(N, H.history["loss"], label="train_loss")
plt.plot(N, H.history["val_loss"], label="val_loss")
plt.plot(N, H.history["accuracy"], label="train_acc")
plt.plot(N, H.history["val_accuracy"], label="val_acc")

plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.savefig('./cnn_plot.png')

model.save('./cnn.model')
f = open('./cnn_lb.pickle', "wb")
f.write(pickle.dumps(lb))
f.close()
```

Shown below is the code used for predicting the input video:

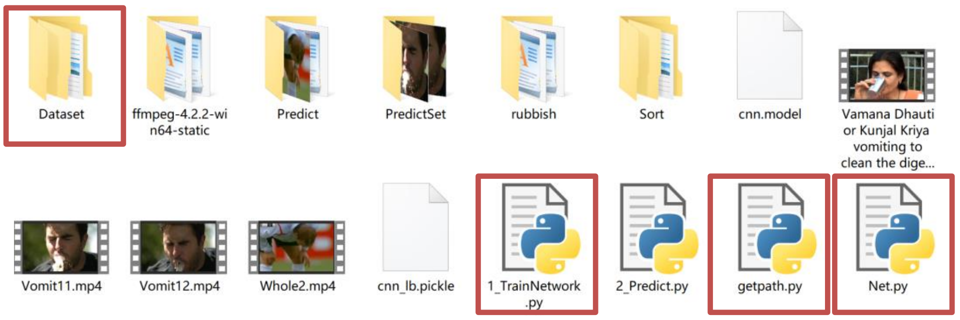
```
model = load_model('./cnn.model')
lb = pickle.loads(open('./cnn_lb.pickle', "rb").read())
fileJson = {}
fileJson["Vomit10.mp4"] = []

import cv2
vc=cv2.VideoCapture("Vomit10.mp4")
c=1
if vc.isOpened():
      rval,frame=vc.read()
else:
      rval=False
while rval:
      cv2.imwrite('C:\\Vomit\\PredictSet\\Predict'+str(c)+'.jpg',frame)
      rval,frame=vc.read()
      image = cv2.imread('C:\\Vomit\\PredictSet\\Predict'+str(c)+'.jpg')
      output = image.copy()
      image = cv2.resize(image, (64, 64))
      image = image.astype("float") / 255.0
      image = image.reshape((1, image.shape[0],
      image.shape[1],image.shape[2]))
      preds = model.predict(image)
      i = preds.argmax(axis=1)[0]
      label = lb.classes_[i]
      text = "{}: {:.2f}%".format(label, preds[0][i] * 100)
      fileJson["Vomit10.mp4"].append([str(c),str([preds[0][0],preds[0][1]])])

      c=c+1
      cv2.waitKey(1)
vc.release()

import json

with open("File.json", "w") as outfile:
    json.dump(fileJson, outfile)

fileJson
```

PART VI IMPLEMENTATION

To train the model, create a folder which contains sorted training data (as shown below). The training set is divided into two folders; one contains images in which the person is vomiting, while the other one contains images in which the person is not vomiting.

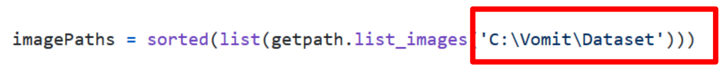


The necessary files include: ./Dataset; 1_TrainNetwork.py; Net.py; getpath.py



To change the network structure, edit Net.py
To train the network, use the demand:
python 1_TrainNetwork.py

Need to notice, the file path of Dataset should be edited in file 1_TrainNetwork.py

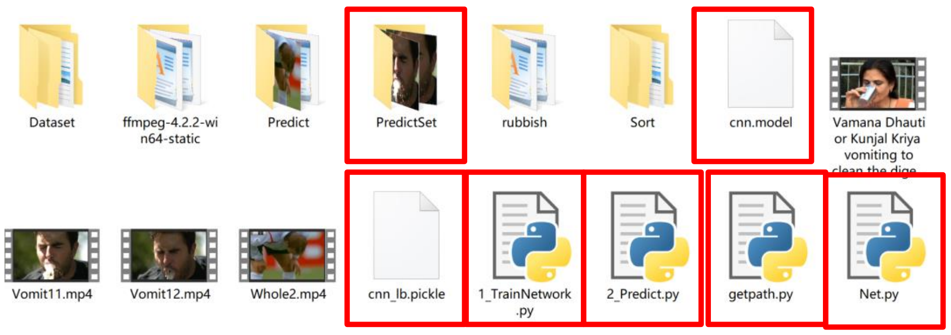


```
imagePaths = sorted(list(getpath.list_images('C:\Vomit\Dataset')))
```

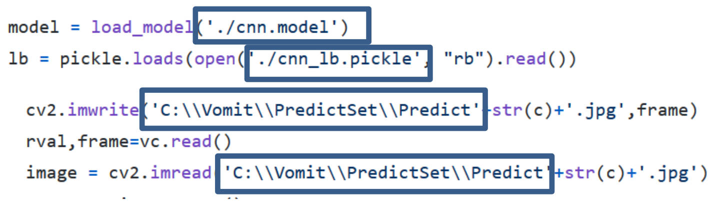
If the file cnn.model and the cnn_lb.pickle are generated then the model is trained and saved successfully.

To use the trained network model and predict an input video, use the demand:
python 2_Predict.py

The necessary files include: ./Predictset; 1_TrainNetwork.py; 2_Predict.py; Net.py; getpath.py; cnn.model; cnn_lb.pickle



Need to notice, the file path of the model and the input video should be edited in file 2_Predict.py

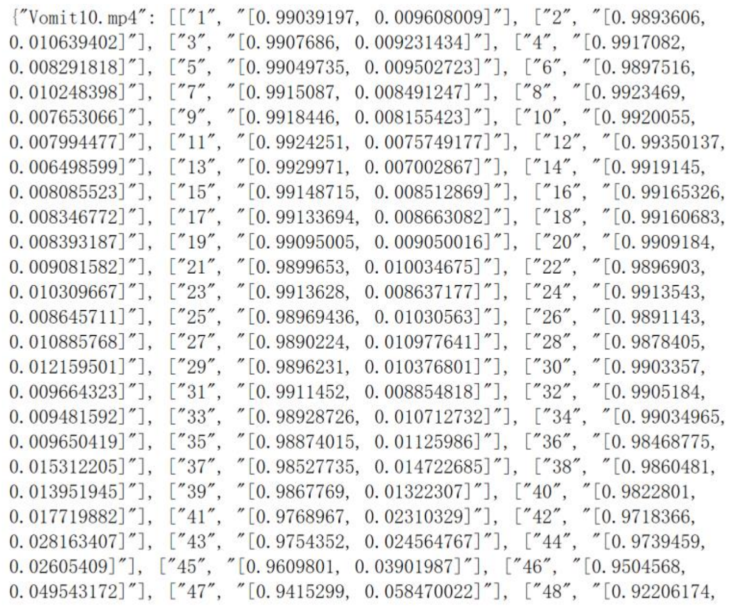


```
model = load_model('./cnn.model')
lb = pickle.loads(open('./cnn_lb.pickle', "rb").read())

  cv2.imwrite('C:\\Vomit\\PredictSet\\Predict'+str(c)+'.jpg',frame)
  rval,frame=vc.read()
  image = cv2.imread('C:\\Vomit\\PredictSet\\Predict'+str(c)+'.jpg')
```

Since the model is too large to be uploaded to Github, the model is uploaded to Google Drive for accessing.

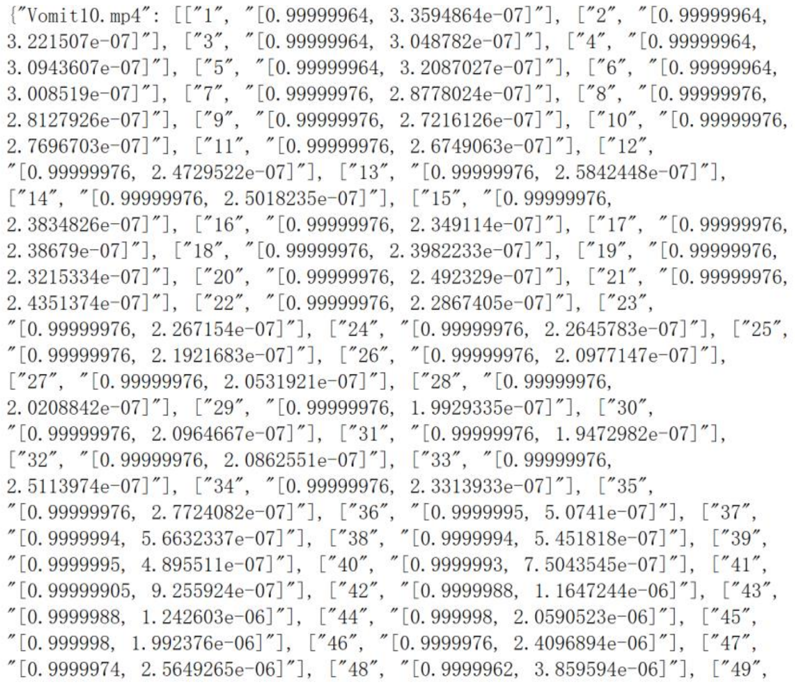
PART VI RESULTS

The previously generated json file for input video Vomit10.mp4 is like:



```
{"Vomit10.mp4": [["1", "[0.99039197, 0.009608009]"], ["2", "[0.9893606,
0.010639402]"], ["3", "[0.9907686, 0.009231434]"], ["4", "[0.9917082,
0.008291818]"], ["5", "[0.99049735, 0.009502723]"], ["6", "[0.9897516,
0.010248398]"], ["7", "[0.9915087, 0.008491247]"], ["8", "[0.9923469,
0.007653066]"], ["9", "[0.9918446, 0.008155423]"], ["10", "[0.9920055,
0.007994477]"], ["11", "[0.9924251, 0.0075749177]"], ["12", "[0.99350137,
0.006498599]"], ["13", "[0.9929971, 0.007002867]"], ["14", "[0.9919145,
0.008085523]"], ["15", "[0.99148715, 0.008512869]"], ["16", "[0.99165326,
0.008346772]"], ["17", "[0.99133694, 0.008663082]"], ["18", "[0.99160683,
0.008393187]"], ["19", "[0.99095005, 0.009050016]"], ["20", "[0.9909184,
0.009081582]"], ["21", "[0.9899653, 0.010034675]"], ["22", "[0.9896903,
0.010309667]"], ["23", "[0.9913628, 0.008637177]"], ["24", "[0.9913543,
0.008645711]"], ["25", "[0.98969436, 0.01030563]"], ["26", "[0.9891143,
0.010885768]"], ["27", "[0.9890224, 0.010977641]"], ["28", "[0.9878405,
0.012159501]"], ["29", "[0.9896231, 0.010376801]"], ["30", "[0.9903357,
0.009664323]"], ["31", "[0.9911452, 0.008854818]"], ["32", "[0.9905184,
0.009481592]"], ["33", "[0.98928726, 0.010712732]"], ["34", "[0.99034965,
0.009650419]"], ["35", "[0.98874015, 0.01125986]"], ["36", "[0.98468775,
0.015312205]"], ["37", "[0.98527735, 0.014722685]"], ["38", "[0.9860481,
0.013951945]"], ["39", "[0.9867769, 0.01322307]"], ["40", "[0.9822801,
0.017719882]"], ["41", "[0.9768967, 0.02310329]"], ["42", "[0.9718366,
0.028163407]"], ["43", "[0.9754352, 0.024564767]"], ["44", "[0.9739459,
0.02605409]"], ["45", "[0.9609801, 0.03901987]"], ["46", "[0.9504568,
0.049543172]"], ["47", "[0.9415299, 0.058470022]"], ["48", "[0.92206174,
```

[["1", "[0. 99039197, 0. 009608009]"] indicate that at the first frame, the probability of non-vomiting is 99.039197%, the probability of vomiting is 0.9608009%.
["55", "[0. 45397803, 0. 54602194]"] indicate that at the 55th frame, the probability of non-vomiting is 45.397803%, the probability of vomiting is 54.602194%.
The latest result is like:



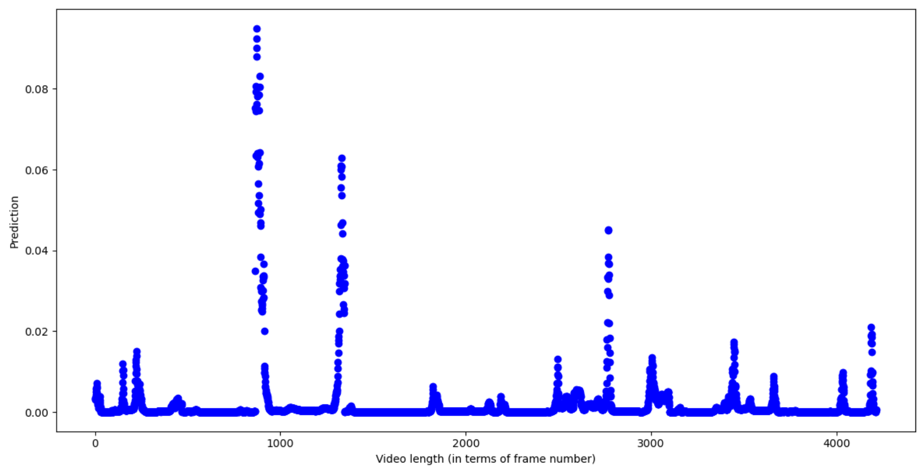
```
{"Vomit10.mp4": [["1", "[0.99999964, 3.3594864e-07]"], ["2", "[0.99999964,
3.221507e-07]"], ["3", "[0.99999964, 3.048782e-07]"], ["4", "[0.99999964,
3.0943607e-07]"], ["5", "[0.99999964, 3.2087027e-07]"], ["6", "[0.99999964,
3.008519e-07]"], ["7", "[0.99999976, 2.8778024e-07]"], ["8", "[0.99999976,
2.8127926e-07]"], ["9", "[0.99999976, 2.7216126e-07]"], ["10", "[0.99999976,
2.7696703e-07]"], ["11", "[0.99999976, 2.6749063e-07]"], ["12",
"[0.99999976, 2.4729522e-07]"], ["13", "[0.99999976, 2.5842448e-07]"],
["14", "[0.99999976, 2.5018235e-07]"], ["15", "[0.99999976,
2.3834826e-07]"], ["16", "[0.99999976, 2.349114e-07]"], ["17", "[0.99999976,
2.38679e-07]"], ["18", "[0.99999976, 2.3982233e-07]"], ["19", "[0.99999976,
2.3215334e-07]"], ["20", "[0.99999976, 2.492329e-07]"], ["21", "[0.99999976,
2.4351374e-07]"], ["22", "[0.99999976, 2.2867405e-07]"], ["23",
"[0.99999976, 2.267154e-07]"], ["24", "[0.99999976, 2.2645783e-07]"], ["25",
"[0.99999976, 2.1921683e-07]"], ["26", "[0.99999976, 2.0977147e-07]"],
["27", "[0.99999976, 2.0531921e-07]"], ["28", "[0.99999976,
2.0208842e-07]"], ["29", "[0.99999976, 1.9929335e-07]"], ["30",
"[0.99999976, 2.0964667e-07]"], ["31", "[0.99999976, 1.9472982e-07]"],
["32", "[0.99999976, 2.0862551e-07]"], ["33", "[0.99999976,
2.5113974e-07]"], ["34", "[0.99999976, 2.3313933e-07]"], ["35",
"[0.99999976, 2.7724082e-07]"], ["36", "[0.9999995, 5.0741e-07]"], ["37",
"[0.9999994, 5.6632337e-07]"], ["38", "[0.9999994, 5.451818e-07]"], ["39",
"[0.9999995, 4.895511e-07]"], ["40", "[0.9999993, 7.5043545e-07]"], ["41",
"[0.99999905, 9.255924e-07]"], ["42", "[0.9999988, 1.1647244e-06]"], ["43",
"[0.9999988, 1.242603e-06]"], ["44", "[0.999998, 2.0590523e-06]"], ["45",
"[0.999998, 1.992376e-06]"], ["46", "[0.9999976, 2.4096894e-06]"], ["47",
"[0.9999974, 2.5649265e-06]"], ["48", "[0.9999962, 3.859594e-06]"], ["49",
```

The generated json file and the corresponding videos are also uploaded to Github.

The latest prediction for the eight types of actions provided by professor looks like:

```
{"eight.mp4": [["1", "[0.9968988, 0.0031012252]"], ["2", "[0.99669695,
0.003303025]"], ["3", "[0.99658775, 0.0034122062]"], ["4", "[0.9966091,
0.0033909176]"], ["5", "[0.9958125, 0.004187546]"], ["6", "[0.99504966,
0.00495028]"], ["7", "[0.99425864, 0.005741303]"], ["8", "[0.9929329,
0.007067144]"], ["9", "[0.9939102, 0.0060898485]"], ["10", "[0.9940502,
0.0059497263]"], ["11", "[0.99358094, 0.0064190305]"], ["12", "[0.995106,
0.0048939995]"], ["13", "[0.9964224, 0.003577592]"], ["14", "[0.9971643,
0.0028357236]"], ["15", "[0.9976235, 0.002376475]"], ["16", "[0.9980184,
0.0019816095]"], ["17", "[0.99848264, 0.0015173409]"], ["18", "[0.9989561,
0.0010439513]"], ["19", "[0.998348, 0.0016519804]"], ["20", "[0.99814916,
0.001850861]"], ["21", "[0.9975501, 0.00244996]"], ["22", "[0.99733967,
0.002660318]"], ["23", "[0.99706143, 0.0029385511]"], ["24", "[0.9965013,
0.003498719]"], ["25", "[0.9960955, 0.00390448]"], ["26", "[0.9971726,
0.002827373]"], ["27", "[0.997623, 0.0023770132]"], ["28", "[0.9982664,
0.001733612]"], ["29", "[0.9988452, 0.0011547183]"], ["30", "[0.99935204,
0.0006479757]"], ["31", "[0.999752, 0.00024803967]"], ["32", "[0.9998766,
0.00012336622]"], ["33", "[0.99991477, 8.5197935e-05]"], ["34",
"[0.99993396, 6.600416e-05]"], ["35", "[0.9999639, 3.6171863e-05]"], ["36",
"[0.9999763, 2.3754914e-05]"], ["37", "[0.999985, 1.5060354e-05]"], ["38",
"[0.9999851, 1.4905751e-05]"], ["39", "[0.99997675, 2.3264249e-05]"], ["40",
"[0.9999788, 2.1192283e-05]"], ["41", "[0.99998, 1.997314e-05]"], ["42",
"[0.99997556, 2.4457846e-05]"], ["43", "[0.9999759, 2.4138357e-05]"], ["44",
"[0.999969, 3.098214e-05]"], ["45", "[0.9999697, 3.0239938e-05]"], ["46",
"[0.9999691, 3.086371e-05]"], ["47", "[0.999972, 2.8026334e-05]"], ["48",
"[0.99997234, 2.7669306e-05]"], ["49", "[0.999969, 3.0950687e-05]"], ["50",
"[0.99996984, 3.0199248e-05]"], ["51", "[0.9999691, 3.0905067e-05]"], ["52",
```

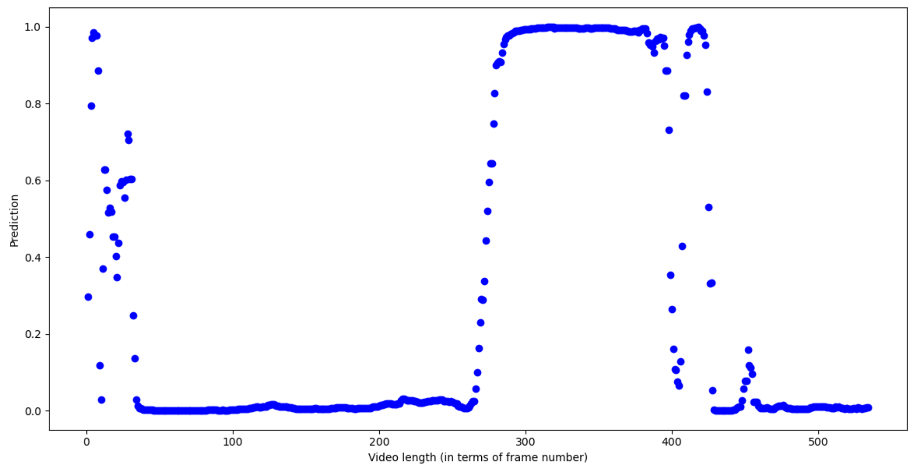
The time label figure looks like:



Since the probability of vomit is very small, which is smaller than 0.01. It's because the eight actions don't include vomiting.

Another test case is 2.mp4, the output time label figure looks like:

And i



In the later half of the video, the person starts vomiting. And the probability of prediction becomes close to 1.

However, the prediction could be influenced by the lightning condition and the angle of the camera. It’s hard to improve since the training data I found is not that much.