CSCE636 Project Part V

Name: Yiting Luo UIN: 427008285

PART I TOPIC

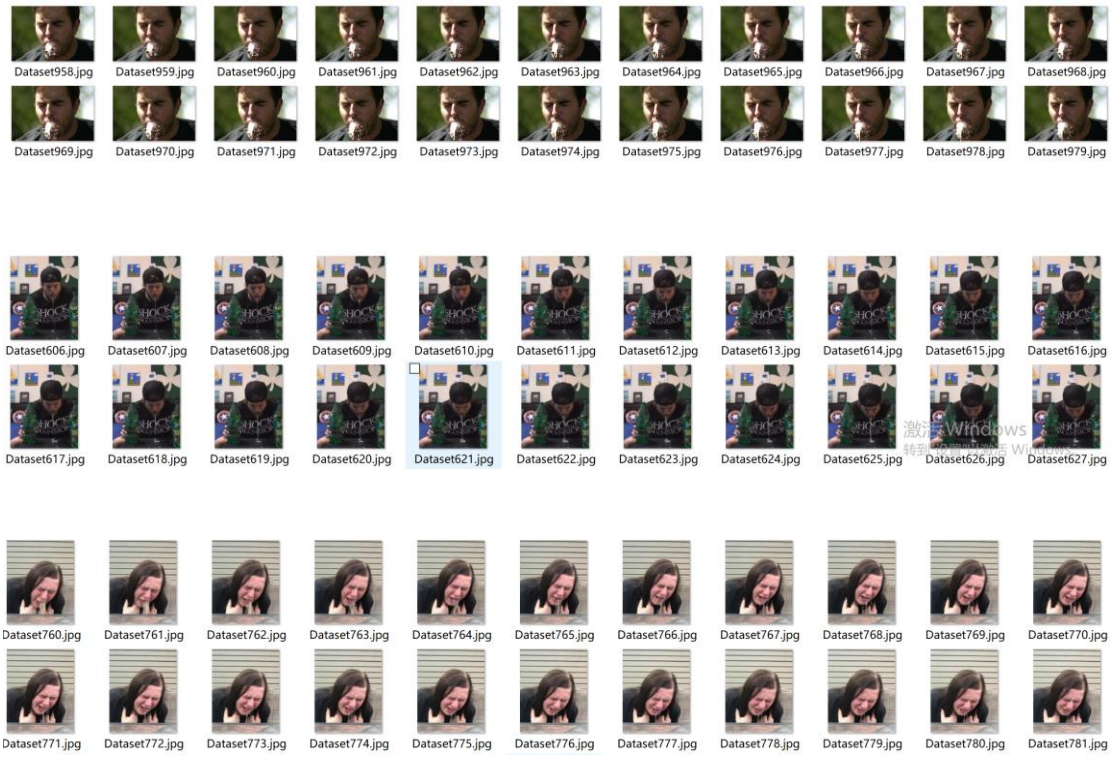
The purpose of the project is to train a neural network so it could be applied to video detection in the future smart home environment. The topic assigned to me is "detecting vomit".

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.

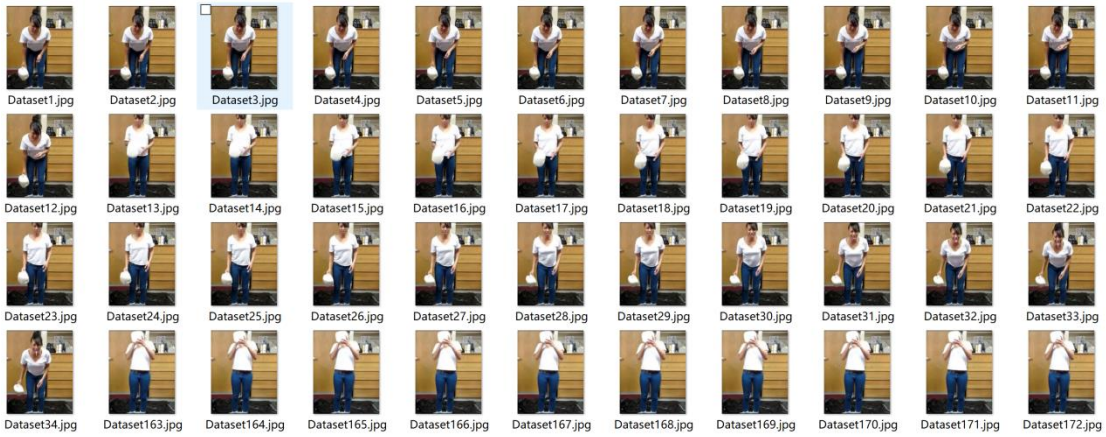
PART II DATASET

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 the Part IV, most of them are vomiting faces.



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.

	Training Dataset
# of Vomit Images	784
# of Non-vomit Images	319
# of All Images	1103

PART III ARCHITECTURE IMPROVEMENTS

Previously there are over fitting problems, but as I added the epochs and the number of layers, the over fitting problem is solved. Also I resize the image to 64 since the resized 32 doesn't work well for images which include the whole body of a person.

Privious input shape of tensor:

X_train shape is (1004,32,32)

Y_train shape is (216,32,32)

Previous output shape of tensor:

x_train shape is (1004,32,32)

y_train shape is (216,32,32)

New input shape of tensor:

X_train shape is (827,64,64)

Y_train shape is (276,64,64)

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

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:

```
Epoch 1/30
 32/827 [>.............................] - ETA: 45s - loss: 0.7401 - accuracy: 0.
 64/827 [=>............................] - ETA: 30s - loss: 0.7119 - accuracy: 0.
 96/827 [==>...........................] - ETA: 25s - loss: 0.6840 - accuracy: 0.
128/827 [===>..........................] - ETA: 22s - loss: 0.6692 - accuracy: 0.
160/827 [====>.........................] - ETA: 19s - loss: 0.6269 - accuracy: 0.
192/827 [=====>........................] - ETA: 18s - loss: 0.5887 - accuracy: 0.
224/827 [=======>......................] - ETA: 17s - loss: 0.5847 - accuracy: 0.
256/827 [========>.....................] - ETA: 15s - loss: 0.5667 - accuracy: 0.
288/827 [=========>....................] - ETA: 27s - loss: 0.5425 - accuracy: 0.
320/827 [==========>...................] - ETA: 24s - loss: 0.5295 - accuracy: 0.
352/827 [===========>..................] - ETA: 21s - loss: 0.5254 - accuracy: 0.
384/827 [============>.................] - ETA: 19s - loss: 0.5181 - accuracy: 0.
416/827 [==============>...............] - ETA: 17s - loss: 0.5032 - accuracy: 0.
448/827 [===============>..............] - ETA: 15s - loss: 0.4979 - accuracy: 0.
480/827 [================>.............] - ETA: 13s - loss: 0.4819 - accuracy: 0.
512/827 [=================>............] - ETA: 12s - loss: 0.4651 - accuracy: 0.
544/827 [==================>...........] - ETA: 10s - loss: 0.4550 - accuracy: 0.
576/827 [===================>..........] - ETA: 9s - loss: 0.4432 - accuracy: 0.8
608/827 [=====================>........] - ETA: 9s - loss: 0.4360 - accuracy: 0.8
640/827 [======================>.......] - ETA: 8s - loss: 0.4247 - accuracy: 0.8
672/827 [=======================>......] - ETA: 6s - loss: 0.4156 - accuracy: 0.8
704/827 [========================>.....] - ETA: 5s - loss: 0.4079 - accuracy: 0.8
736/827 [=========================>....] - ETA: 3s - loss: 0.3976 - accuracy: 0.8
768/827 [==========================>...] - ETA: 2s - loss: 0.3961 - accuracy: 0.8
800/827 [===========================>..] - ETA: 1s - loss: 0.3957 - accuracy: 0.8
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.

PART IV CODE

Shown below is the structure of the network:

```
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:

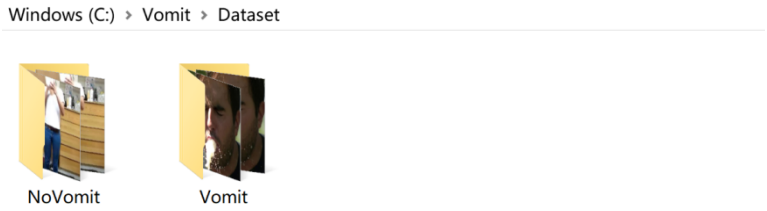
```
data = []
labels = []

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:

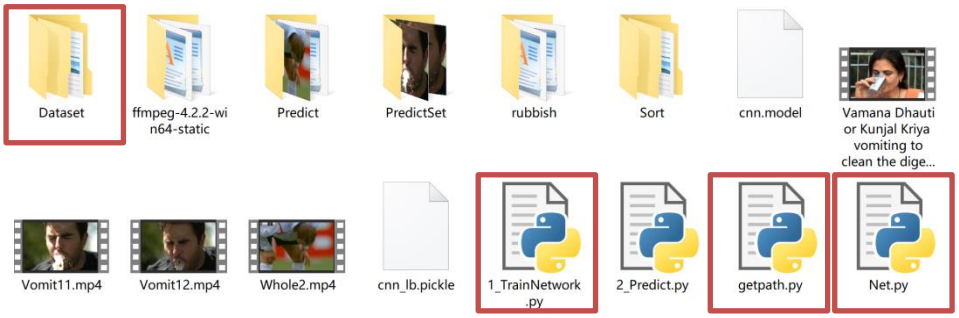
```
model = load_model('./cnn.model')
lb = pickle.loads(open('./cnn_lb.pickle', "rb").read())
fileJson = {}
fileJson["Vomit10.mp4"] = []

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 V 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

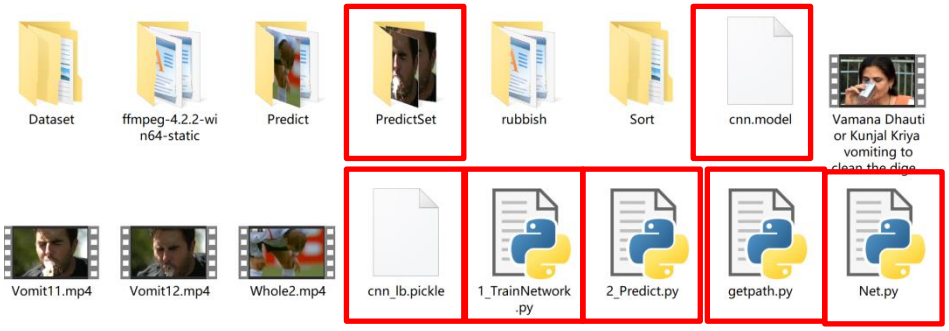
Need to notice, the file path of Dataset should be edited in file 1_TrainNetwork.py

```
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.