CSCE 636-601
Phat Nguyen
UIN: 726006842

Project Submission 5

Video Classification

1. Topic

The goal of this project is to build a neural network that is capable of detecting human doing housework. Since housework includes too many activities, from cleaning, cooking to washing dishes, etc. Thus, for this project, I narrow down the target of the neural network to detect one housework activity: **cleaning up the floor**.

2. What's new in this submission

The target of this submission is to answer the question: is your neural network good enough to be deployed in a real smart home system? In this case, the neural network can only be good for a real home monitoring system if it can distinguish between cleaning and other common indoor activities.

In the last submission, the neural network was trained with the UCF101 dataset to detect mopping activity against 4 other activities: billiards, floor gymnastics, golf swing, javelin throw and showed great performance (accuracy = 94%). However, when tested against untrained activities, the model did not work very well (accuracy = 35%). Moreover, it could not even detect any single mopping frame (accuracy = 0%) when tested with 100 cleaning examples taken from another dataset (STAIR-ACTION). This means there is much room for improvement.

Thus, the objectives of this submission are:

- Improve the accuracy of the neural engine in detecting housecleaning activity.
- Improve the performance of the neural engine in distinguishing between cleaning with other indoor activities.

3. Dataset

In submission 4, the model was trained with 246 video clips from the UCF101 dataset. Only 76 of which were house cleaning. The rest (170 videos) were about other activities. Thus, *small and biased training dataset* is the reason why the neural network performs poorly when tested with examples outside of UCF101.

In this submission, I include *all indoor activities* in the UCF101 dataset and add some house cleaning examples from the STAIR-ACTION dataset [cite] to make a bigger and more balanced training data. Thus, the final dataset has a total of **2,226** videos:

- Training set: **1,451** videos
    - 726 videos of house cleaning
    - 725 videos of 15 indoor activities:
        - Apply Eye Makeup
        - Apply Lipstick

        - Baby Crawling
        - Blow Dry Hair
        - Blow Candles
        - Brushing Teeth
        - Cutting in Kitchen
        - Knitting
        - Playing Cello
        - Playing Guitar
        - Playing Piano
        - Playing Violin
        - Pull Ups
        - Push Ups
        - Shaving Beard
    - Training videos are split into **7,181** total frames
        - Training frames: **5,744**
        - Validation frames: **1,437**

- Testing set: **775** videos
    - **663** videos to test the model on trained activities
    - **112** videos to test the model on unmet activities:
        - Ironing
        - Washing Dishes
        - Haircutting
        - Writing on Board
        - Playing Yoyo

4. DNN Model

4.1. Architecture

Input vector	25088
Hidden layer	512
Hidden layer	256
Hidden layer	256
Hidden layer	128
Output layer	2

4.2. Input tensor [need fix]

Initial training:	(5744. 224, 224, 3)
Initial validating:	(1437, 224, 224, 3)
Initial testing	(4549, 224, 224, 3)

Training and validating data are preprocessed with the pretrained VGG-16 model and then reshaped [1]. Thus, the final training and validating data is:

Final training:	(5744, 25088)
Final validating:	(1437, 25088)
Final testing:	(4549, 25088)

4.3. Output tensor [need fix]

Output training:	(5744, 2)
Output validating:	(1437, 2)
Output testing:	(4549, 2)

5. Hyperparameters

5.1. Range of Hyperparameters Tried

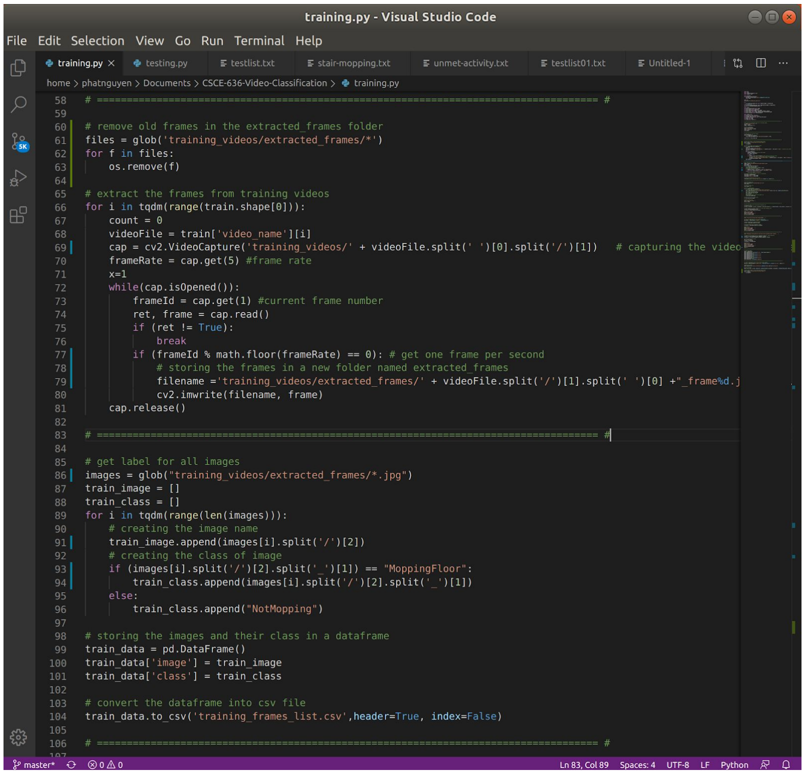
Batch size	32, 64, 128
Epochs	15, 20, 25
Dropout	0.1 - 0.5

5.2. Optimal Hyperparameters:

Batch size	128
Epochs	15
Dropout	0.5

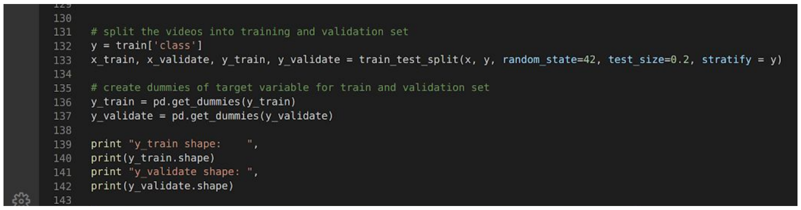
6. Annotated Code

- Part of the code is referenced from this article [1].
- Part of them is added/modified to fit the project requirements and optimize the model performance, including:
    - Model architecture.
    - Writing time/label data to output JSON file.
    - Testing method (per frame instead of per video as in the reference code)


training.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
training.py
testing.py
testlist.txt
stair-mopping.txt
unmet-activity.txt
testlist01.txt
Untitled-1
home > phatnguyen > Documents > CSCE-636-Video-Classification > training.py
master*
0 0
Ln 83, Col 89 Spaces: 4 UTF-8 LF Python

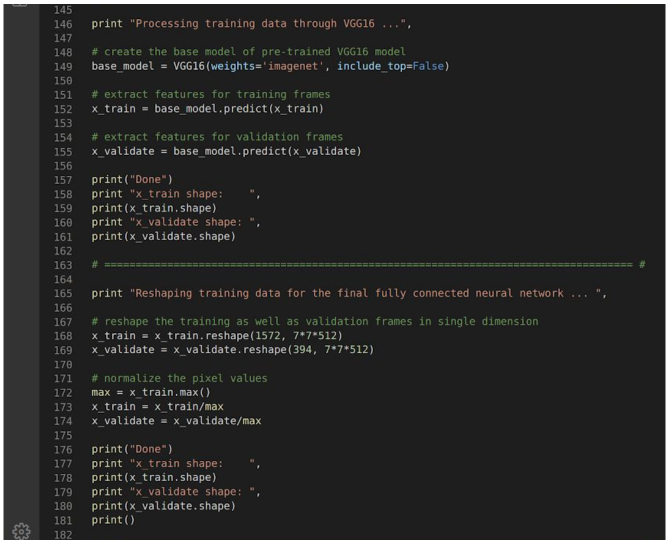
```
# ==================================================================================== #

# remove old frames in the extracted_frames folder
files = glob('training_videos/extracted_frames/*')
for f in files:
    os.remove(f)

# extract the frames from training videos
for i in tqdm(range(train.shape[0])):
    count = 0
    videoFile = train['video_name'][i]
    cap = cv2.VideoCapture('training_videos/' + videoFile.split(' ')[0].split('/')[1])   # capturing the video
    frameRate = cap.get(5) #frame rate
    x=1
    while(cap.isOpened()):
        frameId = cap.get(1) #current frame number
        ret, frame = cap.read()
        if (ret != True):
            break
        if (frameId % math.floor(frameRate) == 0): # get one frame per second
            # storing the frames in a new folder named extracted_frames
            filename ='training_videos/extracted_frames/' + videoFile.split('/')[1].split(' ')[0] +"_frame%d.j
            cv2.imwrite(filename, frame)
    cap.release()

# ==================================================================================== #

# get label for all images
images = glob("training_videos/extracted_frames/*.jpg")
train_image = []
train_class = []
for i in tqdm(range(len(images))):
    # creating the image name
    train_image.append(images[i].split('/')[2])
    # creating the class of image
    if (images[i].split('/')[2].split('_')[1]) == "MoppingFloor":
        train_class.append(images[i].split('/')[2].split('_')[1])
    else:
        train_class.append("NotMopping")

# storing the images and their class in a dataframe
train_data = pd.DataFrame()
train_data['image'] = train_image
train_data['class'] = train_class

# convert the dataframe into csv file
train_data.to_csv('training_frames_list.csv',header=True, index=False)

# ==================================================================================== #
```

Listing 1: Extract the frames from the training dataset and label each frame.



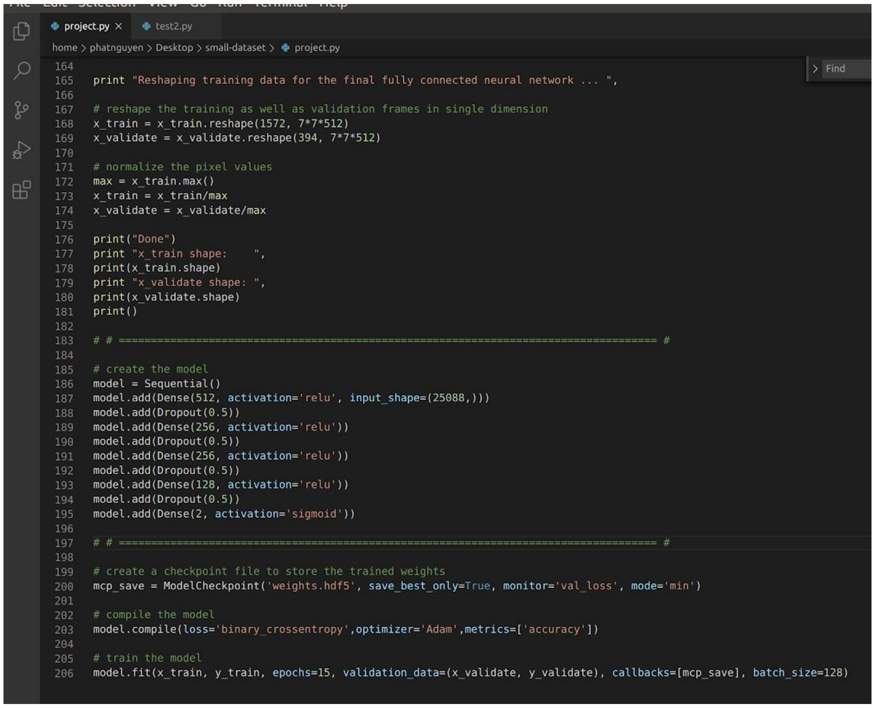
```

# split the videos into training and validation set
y = train['class']
x_train, x_validate, y_train, y_validate = train_test_split(x, y, random_state=42, test_size=0.2, stratify = y)

# create dummies of target variable for train and validation set
y_train = pd.get_dummies(y_train)
y_validate = pd.get_dummies(y_validate)

print "y_train shape:    ",
print(y_train.shape)
print "y_validate shape: ",
print(y_validate.shape)

```

Listing 2: Split all the frames into training set and validation set



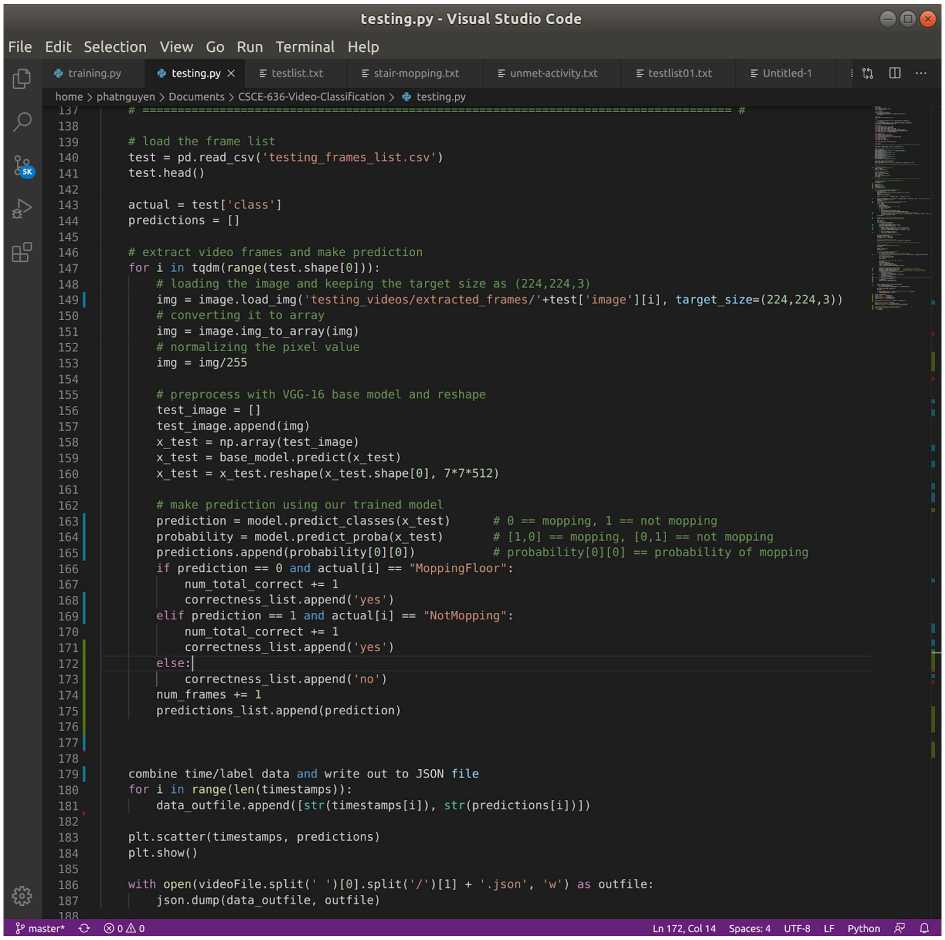
```

print "Processing training data through VGG16 ...",

# create the base model of pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False)

# extract features for training frames
x_train = base_model.predict(x_train)

# extract features for validation frames
x_validate = base_model.predict(x_validate)

print("Done")
print "x_train shape:    ",
print(x_train.shape)
print "x_validate shape: ",
print(x_validate.shape)

# ==================================================================================== #

print "Reshaping training data for the final fully connected neural network ... ",

# reshape the training as well as validation frames in single dimension
x_train = x_train.reshape(1572, 7*7*512)
x_validate = x_validate.reshape(394, 7*7*512)

# normalize the pixel values
max = x_train.max()
x_train = x_train/max
x_validate = x_validate/max

print("Done")
print "x_train shape:    ",
print(x_train.shape)
print "x_validate shape: ",
print(x_validate.shape)
print()

```

Listing 3: Preprocessing the training data using the VGG-16 pretrained model and reshaping the data



```python
print "Reshaping training data for the final fully connected neural network ... ",

# reshape the training as well as validation frames in single dimension
x_train = x_train.reshape(1572, 7*7*512)
x_validate = x_validate.reshape(394, 7*7*512)

# normalize the pixel values
max = x_train.max()
x_train = x_train/max
x_validate = x_validate/max

print("Done")
print "x_train shape:    ",
print(x_train.shape)
print "x_validate shape: ",
print(x_validate.shape)
print()

# # ================================================================================== #

# create the model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(25088,)))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='sigmoid'))

# # ================================================================================== #

# create a checkpoint file to store the trained weights
mcp_save = ModelCheckpoint('weights.hdf5', save_best_only=True, monitor='val_loss', mode='min')

# compile the model
model.compile(loss='binary_crossentropy',optimizer='Adam',metrics=['accuracy'])

# train the model
model.fit(x_train, y_train, epochs=15, validation_data=(x_validate, y_validate), callbacks=[mcp_save], batch_size=128)
```

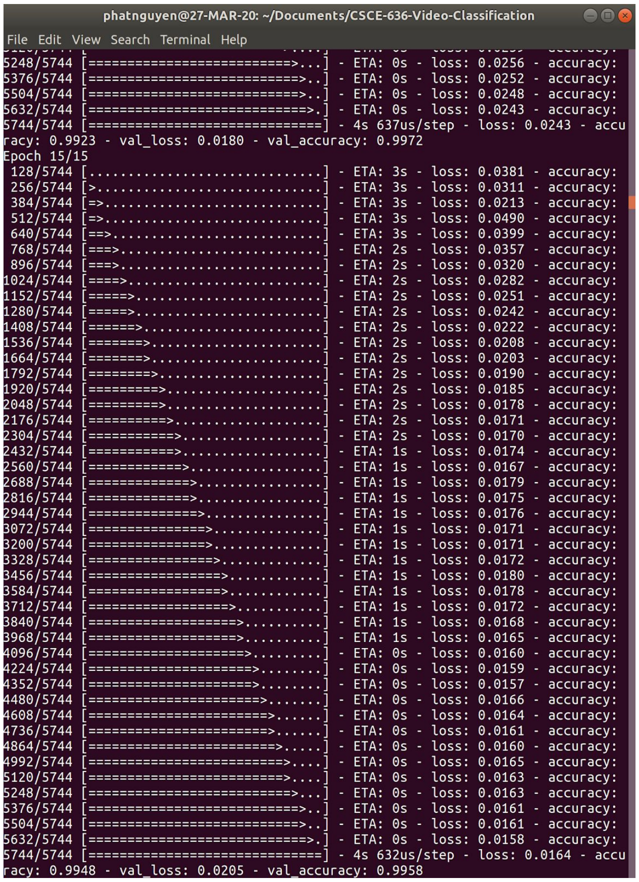
Listing 4: Creating, compiling and training the model


testing.py - Visual Studio Code
File Edit Selection View Go Run Terminal Help
training.py
testing.py
testlist.txt
stair-mopping.txt
unmet-activity.txt
testlist01.txt
Untitled-1
home > phatnguyen > Documents > CSCE-636-Video-Classification > testing.py
5K
master*
0
0
Ln 172, Col 14
Spaces: 4
UTF-8
LF
Python

```python
    # ================================================================================== #

    # load the frame list
    test = pd.read_csv('testing_frames_list.csv')
    test.head()

    actual = test['class']
    predictions = []

    # extract video frames and make prediction
    for i in tqdm(range(test.shape[0])):
        # loading the image and keeping the target size as (224,224,3)
        img = image.load_img('testing_videos/extracted_frames/'+test['image'][i], target_size=(224,224,3))
        # converting it to array
        img = image.img_to_array(img)
        # normalizing the pixel value
        img = img/255

        # preprocess with VGG-16 base model and reshape
        test_image = []
        test_image.append(img)
        x_test = np.array(test_image)
        x_test = base_model.predict(x_test)
        x_test = x_test.reshape(x_test.shape[0], 7*7*512)

        # make prediction using our trained model
        prediction = model.predict_classes(x_test)      # 0 == mopping, 1 == not mopping
        probability = model.predict_proba(x_test)       # [1,0] == mopping, [0,1] == not mopping
        predictions.append(probability[0][0])           # probability[0][0] == probability of mopping
        if prediction == 0 and actual[i] == "MoppingFloor":
            num_total_correct += 1
            correctness_list.append('yes')
        elif prediction == 1 and actual[i] == "NotMopping":
            num_total_correct += 1
            correctness_list.append('yes')
        else:
            correctness_list.append('no')
        num_frames += 1
        predictions_list.append(prediction)



    combine time/label data and write out to JSON file
    for i in range(len(timestamps)):
        data_outfile.append([str(timestamps[i]), str(predictions[i])])

    plt.scatter(timestamps, predictions)
    plt.show()

    with open(videoFile.split(' ')[0].split('/')[1] + '.json', 'w') as outfile:
        json.dump(data_outfile, outfile)
```

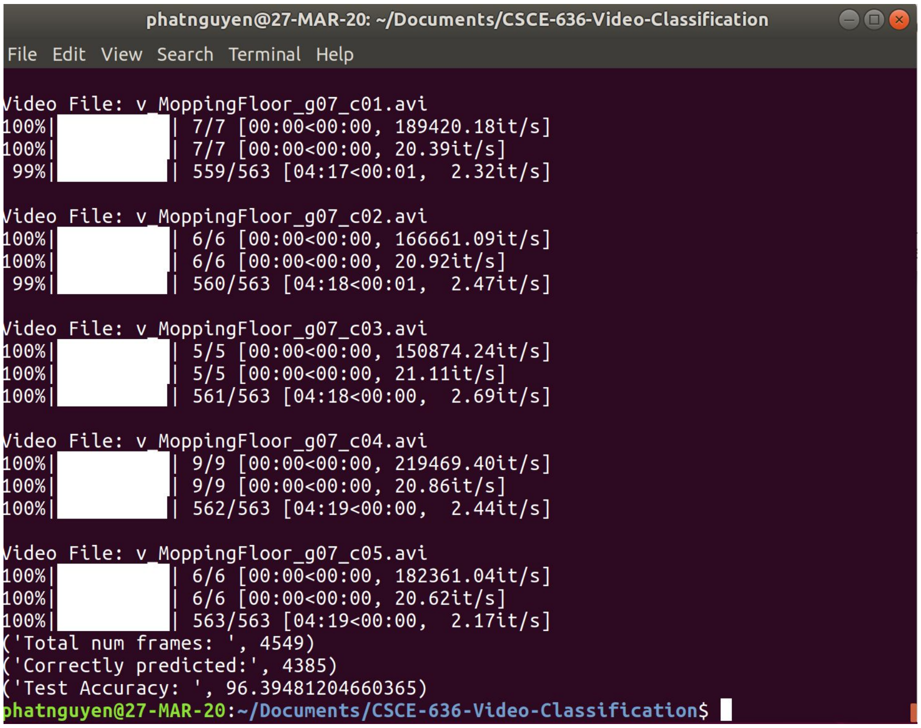
Listing 5: Extracting frames from the testing set and feeding to the model

7. Training and Testing Performance


phatnguyen@27-MAR-20: ~/Documents/CSCE-636-Video-Classification
File Edit View Search Terminal Help

```
5248/5744 [==========================>...] - ETA: 0s - loss: 0.0256 - accuracy:
5376/5744 [===========================>..] - ETA: 0s - loss: 0.0252 - accuracy:
5504/5744 [===========================>..] - ETA: 0s - loss: 0.0248 - accuracy:
5632/5744 [============================>.] - ETA: 0s - loss: 0.0243 - accuracy:
5744/5744 [==============================] - 4s 637us/step - loss: 0.0243 - accu
racy: 0.9923 - val_loss: 0.0180 - val_accuracy: 0.9972
Epoch 15/15
 128/5744 [..............................] - ETA: 3s - loss: 0.0381 - accuracy:
 256/5744 [>.............................] - ETA: 3s - loss: 0.0311 - accuracy:
 384/5744 [=>............................] - ETA: 3s - loss: 0.0213 - accuracy:
 512/5744 [=>............................] - ETA: 3s - loss: 0.0490 - accuracy:
 640/5744 [==>...........................] - ETA: 3s - loss: 0.0399 - accuracy:
 768/5744 [===>..........................] - ETA: 2s - loss: 0.0357 - accuracy:
 896/5744 [===>..........................] - ETA: 2s - loss: 0.0320 - accuracy:
1024/5744 [====>.........................] - ETA: 2s - loss: 0.0282 - accuracy:
1152/5744 [=====>........................] - ETA: 2s - loss: 0.0251 - accuracy:
1280/5744 [=====>........................] - ETA: 2s - loss: 0.0242 - accuracy:
1408/5744 [======>.......................] - ETA: 2s - loss: 0.0222 - accuracy:
1536/5744 [=======>......................] - ETA: 2s - loss: 0.0208 - accuracy:
1664/5744 [=======>......................] - ETA: 2s - loss: 0.0203 - accuracy:
1792/5744 [========>.....................] - ETA: 2s - loss: 0.0190 - accuracy:
1920/5744 [=========>....................] - ETA: 2s - loss: 0.0185 - accuracy:
2048/5744 [=========>....................] - ETA: 2s - loss: 0.0178 - accuracy:
2176/5744 [==========>...................] - ETA: 2s - loss: 0.0171 - accuracy:
2304/5744 [===========>..................] - ETA: 2s - loss: 0.0170 - accuracy:
2432/5744 [===========>..................] - ETA: 1s - loss: 0.0174 - accuracy:
2560/5744 [============>.................] - ETA: 1s - loss: 0.0167 - accuracy:
2688/5744 [=============>................] - ETA: 1s - loss: 0.0179 - accuracy:
2816/5744 [=============>................] - ETA: 1s - loss: 0.0175 - accuracy:
2944/5744 [==============>...............] - ETA: 1s - loss: 0.0176 - accuracy:
3072/5744 [===============>..............] - ETA: 1s - loss: 0.0171 - accuracy:
3200/5744 [===============>..............] - ETA: 1s - loss: 0.0171 - accuracy:
3328/5744 [================>.............] - ETA: 1s - loss: 0.0172 - accuracy:
3456/5744 [=================>............] - ETA: 1s - loss: 0.0180 - accuracy:
3584/5744 [=================>............] - ETA: 1s - loss: 0.0178 - accuracy:
3712/5744 [==================>...........] - ETA: 1s - loss: 0.0172 - accuracy:
3840/5744 [===================>..........] - ETA: 1s - loss: 0.0168 - accuracy:
3968/5744 [===================>..........] - ETA: 1s - loss: 0.0165 - accuracy:
4096/5744 [====================>.........] - ETA: 0s - loss: 0.0160 - accuracy:
4224/5744 [=====================>........] - ETA: 0s - loss: 0.0159 - accuracy:
4352/5744 [=====================>........] - ETA: 0s - loss: 0.0157 - accuracy:
4480/5744 [======================>.......] - ETA: 0s - loss: 0.0166 - accuracy:
4608/5744 [=======================>......] - ETA: 0s - loss: 0.0164 - accuracy:
4736/5744 [=======================>......] - ETA: 0s - loss: 0.0161 - accuracy:
4864/5744 [========================>.....] - ETA: 0s - loss: 0.0160 - accuracy:
4992/5744 [=========================>....] - ETA: 0s - loss: 0.0165 - accuracy:
5120/5744 [=========================>....] - ETA: 0s - loss: 0.0163 - accuracy:
5248/5744 [==========================>...] - ETA: 0s - loss: 0.0163 - accuracy:
5376/5744 [===========================>..] - ETA: 0s - loss: 0.0161 - accuracy:
5504/5744 [===========================>..] - ETA: 0s - loss: 0.0161 - accuracy:
5632/5744 [============================>.] - ETA: 0s - loss: 0.0158 - accuracy:
5744/5744 [==============================] - 4s 632us/step - loss: 0.0164 - accu
racy: 0.9948 - val_loss: 0.0205 - val_accuracy: 0.9958
```

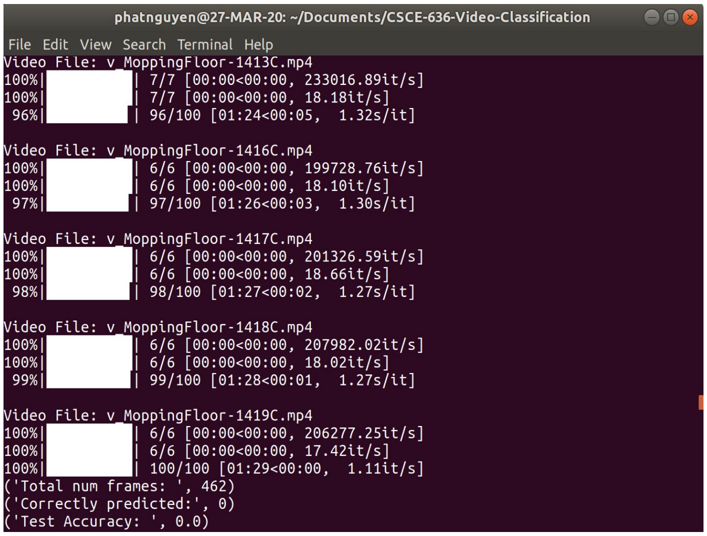
Listing 6: Validating accuracy 99.58%


phatnguyen@27-MAR-20: ~/Documents/CSCE-636-Video-Classification
File Edit View Search Terminal Help

```
Video File: v_MoppingFloor_g07_c01.avi
100%|██████████| 7/7 [00:00<00:00, 189420.18it/s]
100%|██████████| 7/7 [00:00<00:00, 20.39it/s]
 99%|██████████| 559/563 [04:17<00:01,  2.32it/s]

Video File: v_MoppingFloor_g07_c02.avi
100%|██████████| 6/6 [00:00<00:00, 166661.09it/s]
100%|██████████| 6/6 [00:00<00:00, 20.92it/s]
 99%|██████████| 560/563 [04:18<00:01,  2.47it/s]

Video File: v_MoppingFloor_g07_c03.avi
100%|██████████| 5/5 [00:00<00:00, 150874.24it/s]
100%|██████████| 5/5 [00:00<00:00, 21.11it/s]
100%|██████████| 561/563 [04:18<00:00,  2.69it/s]

Video File: v_MoppingFloor_g07_c04.avi
100%|██████████| 9/9 [00:00<00:00, 219469.40it/s]
100%|██████████| 9/9 [00:00<00:00, 20.86it/s]
100%|██████████| 562/563 [04:19<00:00,  2.44it/s]

Video File: v_MoppingFloor_g07_c05.avi
100%|██████████| 6/6 [00:00<00:00, 182361.04it/s]
100%|██████████| 6/6 [00:00<00:00, 20.62it/s]
100%|██████████| 563/563 [04:19<00:00,  2.17it/s]
('Total num frames: ', 4549)
('Correctly predicted:', 4385)
('Test Accuracy: ', 96.39481204660365)
phatnguyen@27-MAR-20:~/Documents/CSCE-636-Video-Classification$
```

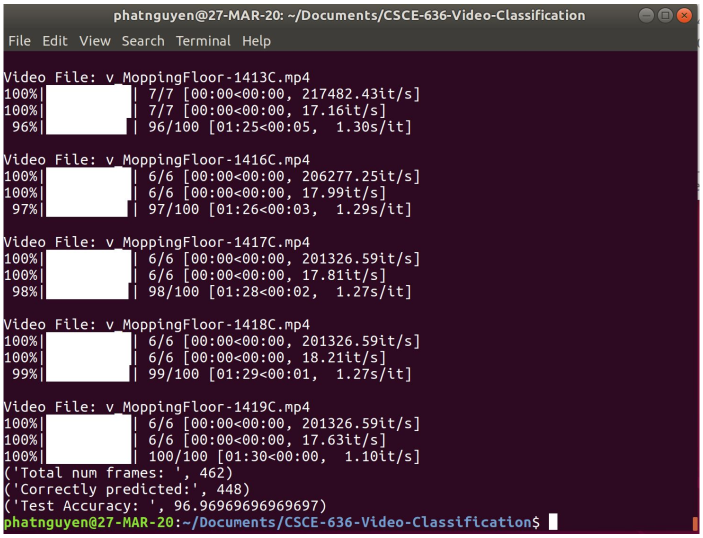
Listing 7: Testing accuracy 96.4%

8. Improvements over Last Submission


phatnguyen@27-MAR-20: ~/Documents/CSCE-636-Video-Classification
File Edit View Search Terminal Help

```
Video File: v_MoppingFloor-1413C.mp4
100%|██████████| 7/7 [00:00<00:00, 233016.89it/s]
100%|██████████| 7/7 [00:00<00:00, 18.18it/s]
 96%|█████████▌| 96/100 [01:24<00:05,  1.32s/it]

Video File: v_MoppingFloor-1416C.mp4
100%|██████████| 6/6 [00:00<00:00, 199728.76it/s]
100%|██████████| 6/6 [00:00<00:00, 18.10it/s]
 97%|█████████▋| 97/100 [01:26<00:03,  1.30s/it]

Video File: v_MoppingFloor-1417C.mp4
100%|██████████| 6/6 [00:00<00:00, 201326.59it/s]
100%|██████████| 6/6 [00:00<00:00, 18.66it/s]
 98%|█████████▊| 98/100 [01:27<00:02,  1.27s/it]

Video File: v_MoppingFloor-1418C.mp4
100%|██████████| 6/6 [00:00<00:00, 207982.02it/s]
100%|██████████| 6/6 [00:00<00:00, 18.02it/s]
 99%|█████████▉| 99/100 [01:28<00:01,  1.27s/it]

Video File: v_MoppingFloor-1419C.mp4
100%|██████████| 6/6 [00:00<00:00, 206277.25it/s]
100%|██████████| 6/6 [00:00<00:00, 17.42it/s]
100%|██████████| 100/100 [01:29<00:00,  1.11it/s]
('Total num frames: ', 462)
('Correctly predicted:', 0)
('Test Accuracy: ', 0.0)
```

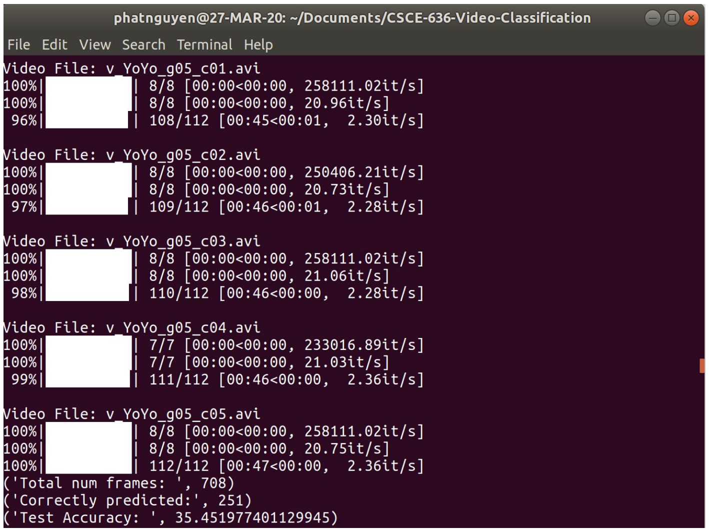
Before (0%)


phatnguyen@27-MAR-20: ~/Documents/CSCE-636-Video-Classification
File Edit View Search Terminal Help

```
Video File: v_MoppingFloor-1413C.mp4
100%|██████████| 7/7 [00:00<00:00, 217482.43it/s]
100%|██████████| 7/7 [00:00<00:00, 17.16it/s]
 96%|█████████▌| 96/100 [01:25<00:05,  1.30s/it]

Video File: v_MoppingFloor-1416C.mp4
100%|██████████| 6/6 [00:00<00:00, 206277.25it/s]
100%|██████████| 6/6 [00:00<00:00, 17.99it/s]
 97%|█████████▋| 97/100 [01:26<00:03,  1.29s/it]

Video File: v_MoppingFloor-1417C.mp4
100%|██████████| 6/6 [00:00<00:00, 201326.59it/s]
100%|██████████| 6/6 [00:00<00:00, 17.81it/s]
 98%|█████████▊| 98/100 [01:28<00:02,  1.27s/it]

Video File: v_MoppingFloor-1418C.mp4
100%|██████████| 6/6 [00:00<00:00, 201326.59it/s]
100%|██████████| 6/6 [00:00<00:00, 18.21it/s]
 99%|█████████▉| 99/100 [01:29<00:01,  1.27s/it]

Video File: v_MoppingFloor-1419C.mp4
100%|██████████| 6/6 [00:00<00:00, 201326.59it/s]
100%|██████████| 6/6 [00:00<00:00, 17.63it/s]
100%|██████████| 100/100 [01:30<00:00,  1.10it/s]
('Total num frames: ', 462)
('Correctly predicted:', 448)
('Test Accuracy: ', 96.96969696969697)
phatnguyen@27-MAR-20:~/Documents/CSCE-636-Video-Classification$ 
```

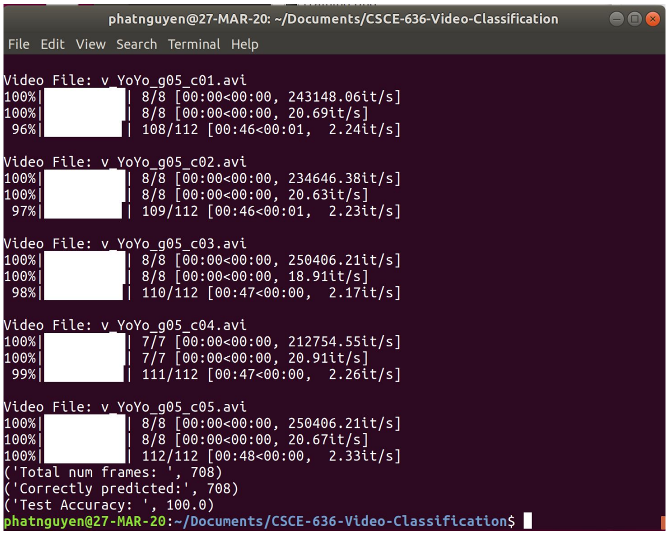
After (97%)

Listing 8: Testing with 100 house cleaning videos from STAIR-ACTION dataset


phatnguyen@27-MAR-20: ~/Documents/CSCE-636-Video-Classification
File Edit View Search Terminal Help

```
Video File: v_YoYo_g05_c01.avi
100%|██████████| 8/8 [00:00<00:00, 258111.02it/s]
100%|██████████| 8/8 [00:00<00:00, 20.96it/s]
 96%|█████████▋| 108/112 [00:45<00:01,  2.30it/s]

Video File: v_YoYo_g05_c02.avi
100%|██████████| 8/8 [00:00<00:00, 250406.21it/s]
100%|██████████| 8/8 [00:00<00:00, 20.73it/s]
 97%|█████████▋| 109/112 [00:46<00:01,  2.28it/s]

Video File: v_YoYo_g05_c03.avi
100%|██████████| 8/8 [00:00<00:00, 258111.02it/s]
100%|██████████| 8/8 [00:00<00:00, 21.06it/s]
 98%|█████████▊| 110/112 [00:46<00:00,  2.28it/s]

Video File: v_YoYo_g05_c04.avi
100%|██████████| 7/7 [00:00<00:00, 233016.89it/s]
100%|██████████| 7/7 [00:00<00:00, 21.03it/s]
 99%|█████████▉| 111/112 [00:46<00:00,  2.36it/s]

Video File: v_YoYo_g05_c05.avi
100%|██████████| 8/8 [00:00<00:00, 258111.02it/s]
100%|██████████| 8/8 [00:00<00:00, 20.75it/s]
100%|██████████| 112/112 [00:47<00:00,  2.36it/s]
('Total num frames: ', 708)
('Correctly predicted:', 251)
('Test Accuracy: ', 35.451977401129945)
```

Before (35.5%)


phatnguyen@27-MAR-20: ~/Documents/CSCE-636-Video-Classification
File Edit View Search Terminal Help

```
Video File: v_YoYo_g05_c01.avi
100%|██████████| 8/8 [00:00<00:00, 243148.06it/s]
100%|██████████| 8/8 [00:00<00:00, 20.69it/s]
 96%|█████████▋| 108/112 [00:46<00:01,  2.24it/s]

Video File: v_YoYo_g05_c02.avi
100%|██████████| 8/8 [00:00<00:00, 234646.38it/s]
100%|██████████| 8/8 [00:00<00:00, 20.63it/s]
 97%|█████████▋| 109/112 [00:46<00:01,  2.23it/s]

Video File: v_YoYo_g05_c03.avi
100%|██████████| 8/8 [00:00<00:00, 250406.21it/s]
100%|██████████| 8/8 [00:00<00:00, 18.91it/s]
 98%|█████████▊| 110/112 [00:47<00:00,  2.17it/s]

Video File: v_YoYo_g05_c04.avi
100%|██████████| 7/7 [00:00<00:00, 212754.55it/s]
100%|██████████| 7/7 [00:00<00:00, 20.91it/s]
 99%|█████████▉| 111/112 [00:47<00:00,  2.26it/s]

Video File: v_YoYo_g05_c05.avi
100%|██████████| 8/8 [00:00<00:00, 250406.21it/s]
100%|██████████| 8/8 [00:00<00:00, 20.67it/s]
100%|██████████| 112/112 [00:48<00:00,  2.33it/s]
('Total num frames: ', 708)
('Correctly predicted:', 708)
('Test Accuracy: ', 100.0)
phatnguyen@27-MAR-20:~/Documents/CSCE-636-Video-Classification$ 
```

After (100%)

Listing 9: Testing with 112 videos of unmet (untrained) activities

9. Instructions on How to Test the Trained DNN

- Dependencies:
  - Python 2.7
  - Keras
  - Tensorflow
  - OpenCV
  - Scipy, sklearn, skimage, glob, tqdm
- How to train:
  - Put the name of the training videos in the trainlist.txt file
  - Put the training video files in the training_videos folder
  - Start the training process using command: python training.py
- How to test:
  - Put the name of the testing videos in the testlist.txt file
  - Put the testing video files in the testing_videos folder
  - Run the test using command: python testing.py

10. Reference

[1] Step-by-Step Deep Learning Tutorial to Build your own Video Classification Model. https://www.analyticsvidhya.com/blog/2019/09/step-by-step-deep-learning-tutorial-video-classification-python/