CSCE 636-601
Phat Nguyen
UIN: 726006842

Project Submission 8

Video Classification

1. **Topic**

The goal of this project is to build a neural network that is capable of detecting human doing housework. Housework activities include cleaning, cooking, washing dishes, etc. For this project, I narrow down the definition of housework to two activities: **house cleaning and washing dishes.**

2. **What's new in this submission**

2.1. The model

Surprisingly, the modified neural network model in submission 6 performed poorly on untrained data, even though it got a very good testing accuracy:

- Accuracy when tested with the sample video provided on YouTube by the professor: **0%.**
- Accuracy when tested with 19 unmet/untrained activities: **50%.**

Thus, for this final submission, I made several changes in the neural network architecture and the dataset. Three most significant modifications made on the neural network architecture are:

- The number of layers and neurons per layer are changed.
- Increase the number of epochs to 25.
- Changed the loss function to *categorical* from *binary* cross-entropy.

Input	Hidden	Hidden	Hidden	Hidden	Hidden	Output
25088	1024	512	256	256	128	3

The training dataset has been increased to include 5 more indoor activities. The architecture of the neural network has also been improved to be able to learn from this more complex dataset. I've also found that the best number of epochs increase from 15 to 25. This gives more time for the model to learn.

In last submission, the output of the neural network is binary: housework or not_housework. This setup is problematic because we have two distinct housework activities: house cleaning and washing dishes. Combining them into a group makes it hard for the neural network to detect because it is difficult to find the common between the two activities.

Thus, in this submission, the output is changed to *categorical*: house cleaning, washing dishes and not housework.

With this configuration, it is easier for the neural model to learn and therefore increases the prediction accuracy:

- Accuracy when tested with the sample video provided on YouTube by the professor: **100%.**
- Accuracy when tested with 19 unmet/untrained activities: **81%.**

2.2. The training/testing sample videos naming format

In previous submissions, names of the input videos have to be put in *testlist.txt* and *trainlist.txt* in a very specific and complex way (see section 9 in previous submissions). This was because the code was written to directly handle the videos from the UCF101 dataset.

For this submission, the code has been rewritten to handle the full name of input videos. Tester just needs to type in the original name of the video in *testlist.txt* or *trainlist.txt.* No reformatting is needed (see section 9 for more information).

3. Dataset

A total of 44 selected activities in the UCF101 and STAIR datasets are used. The final dataset includes a total of **3,091** videos grouped into 3 sets: *training set, testing set*, and *unmet testing set*. The *unmet testing set* is used to test the performance of the model against several activities that it has never encountered or been trained on.

- Training set: **1,622** videos
    - 312 videos of house cleaning
    - 350 videos of washing dishes
    - 960 videos of other 23 indoor activities:
        - Apply Eye Makeup
        - Apply Lipstick
        - Baby Crawling
        - Blow Dry Hair
        - Blow Candles
        - Brushing Teeth
        - Cutting in Kitchen
        - Folding Laundry
        - Ironing
        - Knitting
        - Playing Cello
        - Playing Guitar
        - Playing Piano
        - Playing Violin
        - Pouring Coffee or Tea
        - Pull Ups
        - Push Ups

        - Shaving Beard
        - Studying
        - Telephoning
        - Washing Hands
        - Wiping Window
        - Wearing Shoes
    - Training videos are split into **11,166** total frames
        - Training frames: **8,932**
        - Validation frames: **2,234**
- Testing set: **967** videos
- Unmet testing set: **502** videos of 19 untrained activities.
    - Eating meal
    - Exercising
    - Haircutting
    - Lying on floor
    - Punching
    - Reading newspaper
    - Rock climbing indoor
    - Rowing
    - Salsa Spin
    - Smoking
    - Soccer Juggling
    - Taichi
    - Throwing Trash
    - Using Computer
    - Walking with Dog
    - Wall Pushup
    - Washing Face
    - Writing on Board
    - Yoyo

4. DNN Model

4.1. Architecture

Input vector	25088
Hidden layer Hidden layer Hidden layer Hidden layer Hidden layer	1024 512 256 256 128
Output layer	3

4.2. Input tensor

Initial training: (8932. 224, 224, 3)
Initial validating: (2234, 224, 224, 3)
Initial testing (7110, 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: (8932, 25088)
Final validating: (2234, 25088)
Final testing: (7110, 25088)

4.3. Output tensor

Output training: (8932, 3)
Output validating: (2234, 3)
Output testing: (7110, 3)

5. Hyperparameters

5.1. Range of Hyperparameters Tried

Batch size	64, 128, 256
Epochs	15, 25, 35, 45
Dropout	0.1 - 0.5

5.2. Optimal Hyperparameters:

Batch size	128
Epochs	25
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.
  - Video frames labeling
  - Output time/label data to JSON file.
  - Testing method (per frame instead of per video as in the reference code)

training.py × testing.py

home > phatnguyen > Documents > CSCE-636-Video-Classification > training.py

```python
# create a dataframe
train = pd.DataFrame()
train['video_name'] = videos
train = train[:-1]
train.head()

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

# remove old frames in the extracted_frames folder
files = glob('training_videos/extracted_frames/*.jpg')
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)   # capturing the video from the given path
    frameRate = cap.get(5) #frame rate
    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+"_frame%d.jpg" % count;count+=1
            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].find('MoppingFloor') != -1:
        train_class.append("housecleaning")
    elif images[i].find('WashingDishes') != -1:
        train_class.append("washingdishes")
    else:
        train_class.append("not_housework")

# 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(x_train.shape[0], 7*7*512)
x_validate = x_validate.reshape(x_validate.shape[0], 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
# ================================================================================= #

# create the model
model = Sequential()

model.add(Dense(1024, activation='relu', input_shape=(25088,)))
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu'))
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(3, activation='softmax'))

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

# 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='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])

# train the model
model.fit(x_train, y_train, epochs=25, validation_data=(x_validate, y_validate), callbacks=[mcp_save], batch_size=128)

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

Listing 4: Creating, compiling and training the model

```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 == housecleaning, 1 == not_housework, 2 == washingdishes
    probability = model.predict_proba(x_test)       # [1,0,0] == housecleaning, [0,1,0] == not_housework, [0,0,1] == washingdishes
    predictions.append(1.0 - probability[0][1])     # probability[0][1] == probability of not_housework
    if prediction == 0 and actual[i] == "housecleaning":
        num_total_correct += 1
        correctness_list.append('yes')
    elif prediction == 1 and actual[i] == "not_housework":
        num_total_correct += 1
        correctness_list.append('yes')
    elif prediction == 2 and actual[i] == "washingdishes":
        num_total_correct += 1
        correctness_list.append('yes')
    else:
        correctness_list.append('no')
    num_frames += 1
    predictions_list.append(prediction)
    probability_list.append(1.0 - probability[0][1])

# 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.ylim(-0.25,1.25)
plt.plot(timestamps, predictions)
plt.savefig(videoFile + '.png')
plt.close()

with open(videoFile + '.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

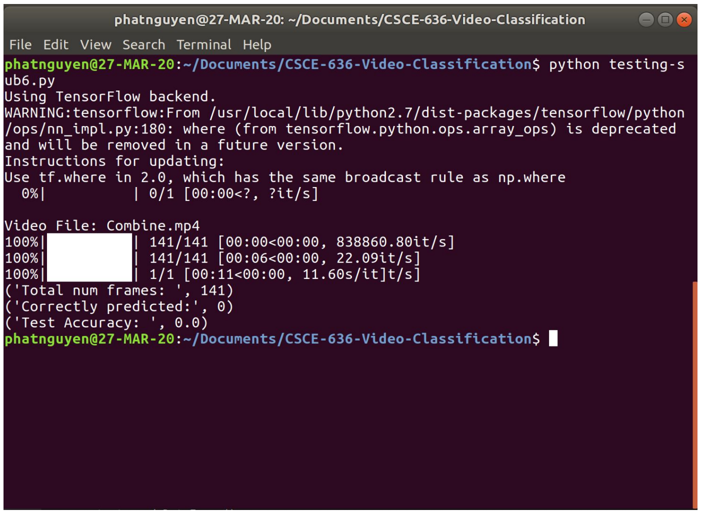
```
phatnguyen@27-MAR-20: ~/Documents/CSCE-636-Video-Classification
File  Edit  View  Search  Terminal  Help
4992/8932 [===============>.............] - ETA: 4s - loss: 0.0511 - accuracy:
5120/8932 [================>............] - ETA: 4s - loss: 0.0501 - accuracy:
5248/8932 [================>............] - ETA: 4s - loss: 0.0503 - accuracy:
5376/8932 [=================>...........] - ETA: 4s - loss: 0.0525 - accuracy:
5504/8932 [=================>...........] - ETA: 4s - loss: 0.0529 - accuracy:
5632/8932 [=================>...........] - ETA: 3s - loss: 0.0522 - accuracy:
5760/8932 [==================>..........] - ETA: 3s - loss: 0.0523 - accuracy:
5888/8932 [==================>..........] - ETA: 3s - loss: 0.0523 - accuracy:
6016/8932 [===================>.........] - ETA: 3s - loss: 0.0517 - accuracy:
6144/8932 [===================>.........] - ETA: 3s - loss: 0.0515 - accuracy:
6272/8932 [====================>.........] - ETA: 3s - loss: 0.0524 - accuracy:
6400/8932 [====================>.........] - ETA: 2s - loss: 0.0516 - accuracy:
6528/8932 [====================>.........] - ETA: 2s - loss: 0.0511 - accuracy:
6656/8932 [=====================>........] - ETA: 2s - loss: 0.0512 - accuracy:
6784/8932 [=====================>........] - ETA: 2s - loss: 0.0514 - accuracy:
6912/8932 [======================>.......] - ETA: 2s - loss: 0.0513 - accuracy:
7040/8932 [======================>.......] - ETA: 2s - loss: 0.0513 - accuracy:
7168/8932 [=======================>......] - ETA: 2s - loss: 0.0507 - accuracy:
7296/8932 [=======================>......] - ETA: 1s - loss: 0.0508 - accuracy:
7424/8932 [=======================>......] - ETA: 1s - loss: 0.0502 - accuracy:
7552/8932 [========================>.....] - ETA: 1s - loss: 0.0495 - accuracy:
7680/8932 [========================>.....] - ETA: 1s - loss: 0.0501 - accuracy:
7808/8932 [=========================>....] - ETA: 1s - loss: 0.0498 - accuracy:
7936/8932 [=========================>....] - ETA: 1s - loss: 0.0492 - accuracy:
8064/8932 [==========================>...] - ETA: 1s - loss: 0.0492 - accuracy:
8192/8932 [==========================>...] - ETA: 0s - loss: 0.0496 - accuracy:
8320/8932 [==========================>...] - ETA: 0s - loss: 0.0500 - accuracy:
8448/8932 [===========================>..] - ETA: 0s - loss: 0.0501 - accuracy:
8576/8932 [===========================>..] - ETA: 0s - loss: 0.0497 - accuracy:
8704/8932 [============================>.] - ETA: 0s - loss: 0.0492 - accuracy:
8832/8932 [============================>.] - ETA: 0s - loss: 0.0495 - accuracy:
8932/8932 [==============================] - 11s 1ms/step - loss: 0.0491 - accur
acy: 0.9865 - val_loss: 0.0325 - val_accuracy: 0.9919
```

Listing 6: Validating accuracy 99.19%

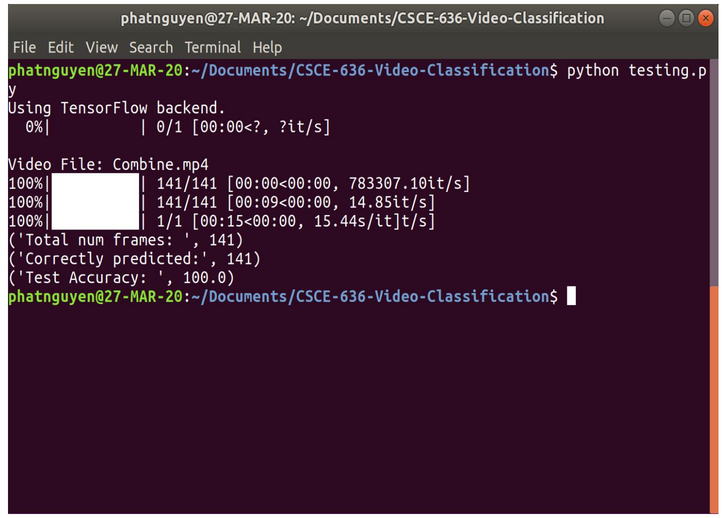
```
phatnguyen@27-MAR-20: ~/Documents/CSCE-636-Video-Classification
File  Edit  View  Search  Terminal  Help
100%|██████████| 963/967 [09:29<00:03,  1.17it/s]

Video File: WipingWindows-0183C.mp4
100%|██████████| 7/7 [00:00<00:00, 206761.46it/s]
100%|██████████| 7/7 [00:00<00:00, 18.22it/s]
100%|██████████| 964/967 [09:30<00:02,  1.17it/s]

Video File: WipingWindows-0184C.mp4
100%|██████████| 7/7 [00:00<00:00, 233016.89it/s]
100%|██████████| 7/7 [00:00<00:00, 18.37it/s]
100%|██████████| 965/967 [09:31<00:01,  1.17it/s]

Video File: WipingWindows-0185C.mp4
100%|██████████| 7/7 [00:00<00:00, 225847.14it/s]
100%|██████████| 7/7 [00:00<00:00, 18.83it/s]
100%|██████████| 966/967 [09:32<00:00,  1.16it/s]

Video File: WipingWindows-0186C.mp4
100%|██████████| 7/7 [00:00<00:00, 225847.14it/s]
100%|██████████| 7/7 [00:00<00:00, 18.24it/s]
100%|██████████| 967/967 [09:32<00:00,  1.69it/s]
('Total num frames: ', 7110)
('Correctly predicted:', 6339)
('Test Accuracy: ', 89.15611814345992)
phatnguyen@27-MAR-20:~/Documents/CSCE-636-Video-Classification$
```

Listing 7: Testing accuracy 89.16%

8. Improvements over Last Submission

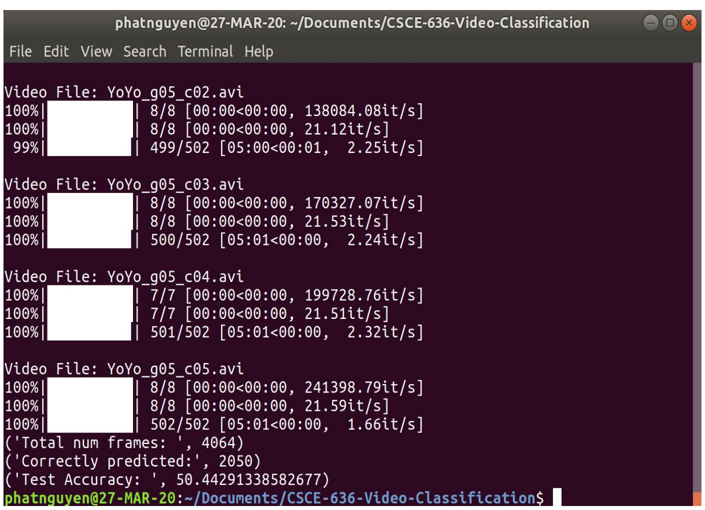


Submission 6 (0%)

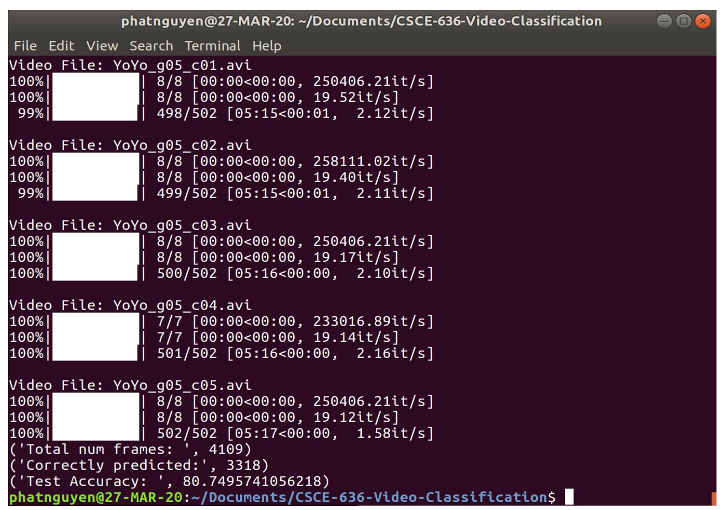


Submission 8 (100%)

Listing 8: Accuracy on YouTube sample video provided by professor Jiang



Submission 6 (50.44%)



Submission 8 (80.75%)

Listing 9: Accuracy on 19 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 full 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 full name of the training videos in the *testlist.txt* file
  - Put the testing video files in the *testing_videos* folder
  - Run the test using command: python testing.py
- For testing script and other instructions, please visit the GitHub page of this project:
  https://github.com/phatnguyen0430/CSCE-636-Video-Classification

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/