CSCE 636-601
Phat Nguyen
UIN: 726006842

Project Submission 4

Video Classification

1. Topic

The goal of this project is to build a neural network that is capable of detecting human doing housework. For this initial submission, I'll narrow down the target of the neural network to detecting one housework activity: **mopping the floor**.

2. Dataset

The dataset being used in this project is the UCF101. It has videos of humans mopping the floor and 100 other different activities. The training set will include both correct and incorrect samples. The correct samples are those mopping videos. For the incorrect samples, if we use all videos of 100 other activities, there will be a significant bias in the training set, which may lead to low prediction performance. Thus, I'll select 4 out of 100 activities:

- Billiards
- Floor Gymnastics
- Golf Swing
- Javelin Throw

Those 4 activities are chosen because they involve interaction with the floor or using devices that have the similar shape to the mop. The final dataset includes **360** videos:

- Training set: **246** videos
  - Split into **1966** total frames.
  - Training set: **1572** frames.
  - Validation set: **394** frames.
- Testing set: **114** videos
  - Split into **937** total frames.

3. DNN Model

3.1. Architecture

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

3.2. Input tensor

Initial training:	(1572. 224, 224, 3)
Initial validating:	(394, 224, 224, 3)
Initial testing	(937, 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:	(1572, 25088)
Final validating:	(394, 25088)
Final testing:	(937, 25088)

3.3. Output tensor

Output training:	(1572, 2)
Output validating:	(394, 2)
Output testing:	(937, 2)

4. Hyperparameters

4.1. Range of Hyperparameters Tried

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

4.2. Optimal Hyperparameters:

Batch size	128
Epochs	15
Dropout	0.5

5. 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)

```
# extract the frames from training videos
for i in tqdm(range(train.shape[0])):
    count = 0
    videoFile = train['video_name'][i]
    cap = cv2.VideoCapture(videoFile.split(' ')[0].split('/')[1])   # capturing the video from the given path
    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):
            # storing the frames in a new folder named training_frames
            filename ='training_frames/' + videoFile.split('/')[1].split(' ')[0] +"_frame%d.jpg" % count;count+=1
            cv2.imwrite(filename, frame)
    cap.release()

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

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

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

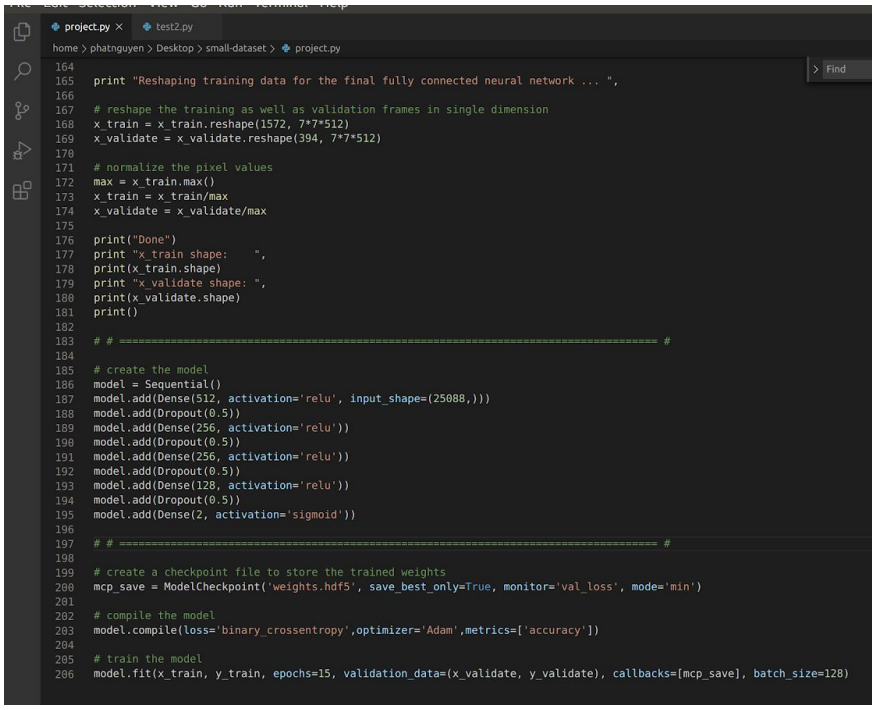
Listing 1: Extract the frames from the training dataset and label each frame.

```
# 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)
```

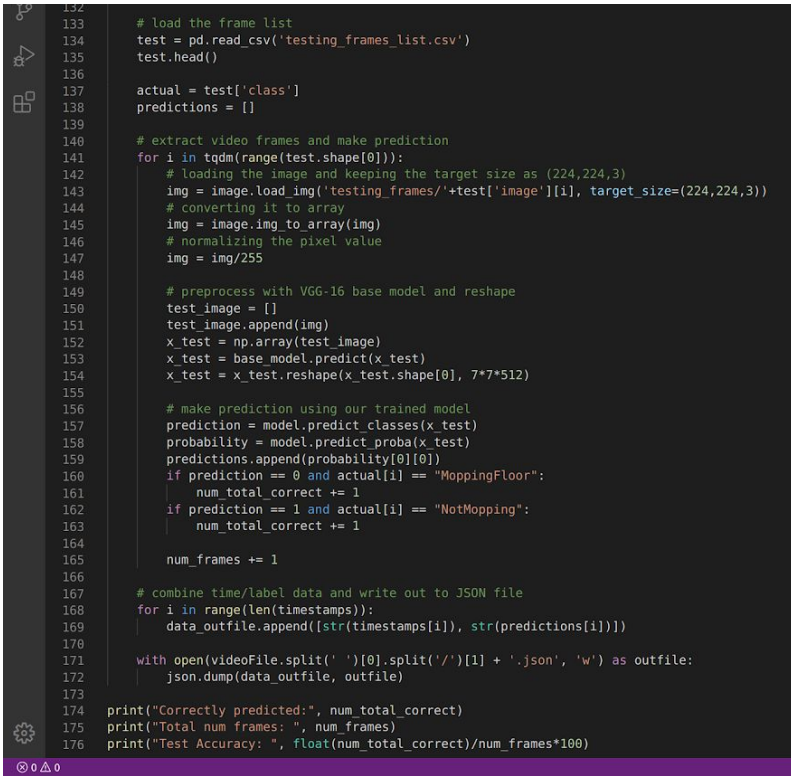
Listing 2: Split all the frames into training set and validation set

```
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

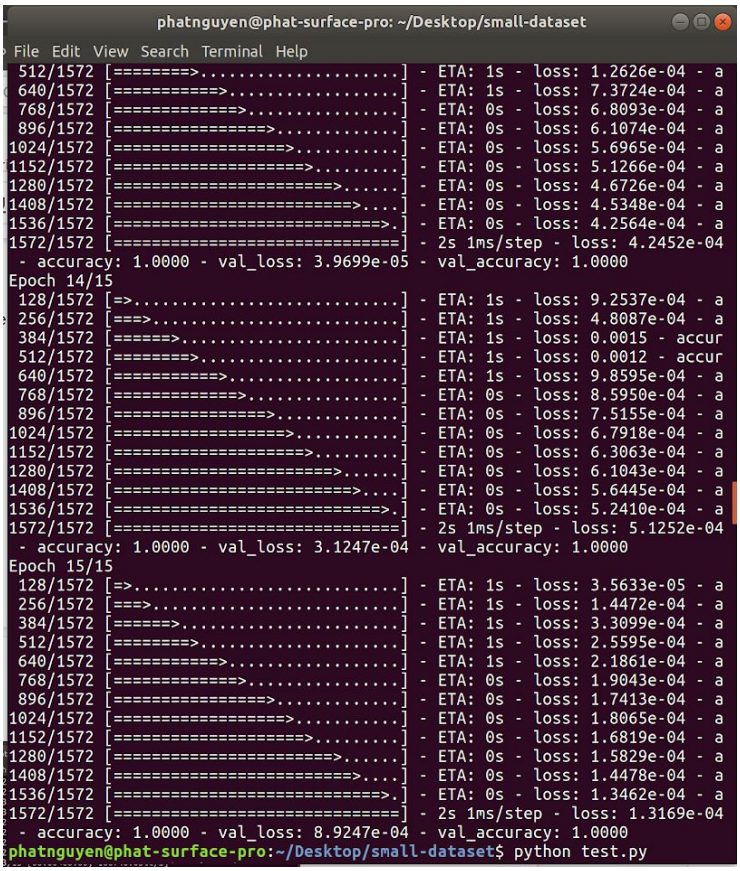


Listing 4: Creating, compiling and training the model

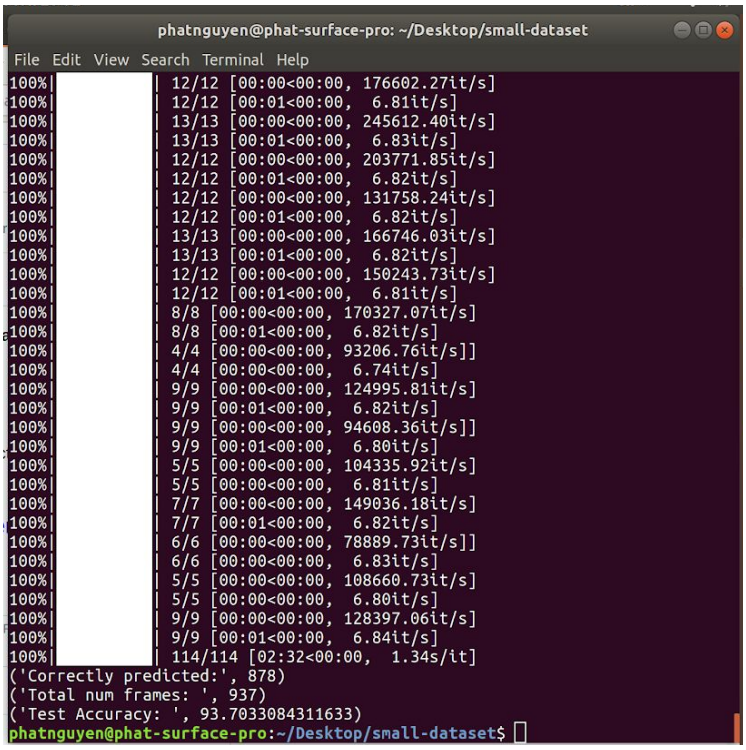


Listing 5: Extracting frames from the testing set and feeding to the model

6. Training and Testing Performance



Listing 6: Validating accuracy 100%



Listing 7: Testing accuracy 93.7%

7. 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 same folder as the training.py file
    - Create an empty folder named training_frames.
    - Run the test.py file 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 same folder as the testing.py file
    - Create an empty folder named testing_frames.
    - Run the test.py file using command: python testing.py

8. 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/