

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	512
Hidden layer	256
Hidden layer	256
Hidden layer	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)

```

59
60 # extract the frames from training videos
61 for i in tqdm(range(train.shape[0])):
62     count = 0
63     videoFile = train['video_name'][i]
64     cap = cv2.VideoCapture(videoFile.split('/')[0].split('/')[1]) # capturing the video from the given path
65     frameRate = cap.get(5) #frame rate
66     x=1
67     while(cap.isOpened()):
68         frameId = cap.get(1) #current frame number
69         ret, frame = cap.read()
70         if (ret != True):
71             break
72         if (frameId % math.floor(frameRate) == 0):
73             # storing the frames in a new folder named training frames
74             filename = 'training_frames/' + videoFile.split('/')[0].split('/')[1] + "_frame%d.jpg" % count; count+=1
75             cv2.imwrite(filename, frame)
76     cap.release()
77
78 # ===== #
79
80 # get label for all images
81 images = glob("training_frames/*.jpg")
82 train_image = []
83 train_class = []
84 for i in tqdm(range(len(images))):
85     # creating the image name
86     train_image.append(images[i].split('/')[1])
87     # creating the class of image
88     if (images[i].split('/')[1].split('.')[1]) == "MoppingFloor":
89         train_class.append(images[i].split('/')[1].split('.')[1])
90     else:
91         train_class.append("NotMopping")
92
93 # storing the images and their class in a dataframe
94 train_data = pd.DataFrame()
95 train_data['image'] = train_image
96 train_data['class'] = train_class
97
98 # convert the dataframe into csv file
99 train_data.to_csv('training_frames_list.csv', header=True, index=False)
100
101 # ===== #

```

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

```

129
130
131 # split the videos into training and validation set
132 y = train['class']
133 x_train, x_validate, y_train, y_validate = train_test_split(x, y, random_state=42, test_size=0.2, stratify = y)
134
135 # create dummies of target variable for train and validation set
136 y_train = pd.get_dummies(y_train)
137 y_validate = pd.get_dummies(y_validate)
138
139 print "y_train shape: ",
140 print(y_train.shape)
141 print "y_validate shape: ",
142 print(y_validate.shape)
143

```

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

```

145
146 print "Processing training data through VGG16 ...",
147
148 # create the base model of pre-trained VGG16 model
149 base_model = VGG16(weights='imagenet', include_top=False)
150
151 # extract features for training frames
152 x_train = base_model.predict(x_train)
153
154 # extract features for validation frames
155 x_validate = base_model.predict(x_validate)
156
157 print("Done")
158 print "x_train shape: ",
159 print(x_train.shape)
160 print "x_validate shape: ",
161 print(x_validate.shape)
162
163 # ===== #
164
165 print "Reshaping training data for the final fully connected neural network ... ",
166
167 # reshape the training as well as validation frames in single dimension
168 x_train = x_train.reshape(1572, 7*7*512)
169 x_validate = x_validate.reshape(394, 7*7*512)
170
171 # normalize the pixel values
172 max = x_train.max()
173 x_train = x_train/max
174 x_validate = x_validate/max
175
176 print("Done")
177 print "x_train shape: ",
178 print(x_train.shape)
179 print "x_validate shape: ",
180 print(x_validate.shape)
181 print()
182

```

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

```

164 print "Reshaping training data for the final fully connected neural network ... ",
165
166 # reshape the training as well as validation frames in single dimension
167 x_train = x_train.reshape(1572, 7*7*512)
168 x_validate = x_validate.reshape(394, 7*7*512)
169
170 # normalize the pixel values
171 max = x_train.max()
172 x_train = x_train/max
173 x_validate = x_validate/max
174
175 print("Done")
176 print "x_train shape: ",
177 print(x_train.shape)
178 print "x_validate shape: ",
179 print(x_validate.shape)
180 print()
181
182 # # ===== #
183
184 # create the model
185 model = Sequential()
186 model.add(Dense(512, activation='relu', input_shape=(25088,)))
187 model.add(Dropout(0.5))
188 model.add(Dense(256, activation='relu'))
189 model.add(Dropout(0.5))
190 model.add(Dense(256, activation='relu'))
191 model.add(Dropout(0.5))
192 model.add(Dense(128, activation='relu'))
193 model.add(Dropout(0.5))
194 model.add(Dense(2, activation='sigmoid'))
195
196 # # ===== #
197
198 # create a checkpoint file to store the trained weights
199 mcp_save = ModelCheckpoint('weights.hdf5', save_best_only=True, monitor='val_loss', mode='min')
200
201 # compile the model
202 model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=['accuracy'])
203
204 # train the model
205 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

```

132 # load the frame list
133 test = pd.read_csv('testing_frames_list.csv')
134 test.head()
135
136 actual = test['class']
137 predictions = []
138
139 # extract video frames and make prediction
140 for i in tqdm(range(test.shape[0])):
141     # loading the image and keeping the target size as (224,224,3)
142     img = image.load_img('testing_frames/'+test['image'][i], target_size=(224,224,3))
143     # converting it to array
144     img = image.img_to_array(img)
145     # normalizing the pixel value
146     img = img/255
147
148     # preprocess with VGG-16 base model and reshape
149     test_image = []
150     test_image.append(img)
151     x_test = np.array(test_image)
152     x_test = base_model.predict(x_test)
153     x_test = x_test.reshape(x_test.shape[0], 7*7*512)
154
155     # make prediction using our trained model
156     prediction = model.predict_classes(x_test)
157     probability = model.predict_proba(x_test)
158     predictions.append(probability[0][0])
159     if prediction == 0 and actual[i] == "MoppingFloor":
160         num_total_correct += 1
161     if prediction == 1 and actual[i] == "NotMopping":
162         num_total_correct += 1
163
164     num_frames += 1
165
166 # combine time/label data and write out to JSON file
167 for i in range(len(timestamps)):
168     data_outfile.append([str(timestamps[i]), str(predictions[i])])
169
170 with open(videoFile.split(' ')[0].split('/')[1] + '.json', 'w') as outfile:
171     json.dump(data_outfile, outfile)
172
173 print("Correctly predicted:", num_total_correct)
174 print("Total num frames: ", num_frames)
175 print("Test Accuracy: ", float(num_total_correct)/num_frames*100)

```

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

6. Training and Testing Performance

```
phatnguyen@phat-surface-pro: ~/Desktop/small-dataset
File Edit View Search Terminal Help
512/1572 [=====>.....] - ETA: 1s - loss: 1.2626e-04 - a
640/1572 [=====>.....] - ETA: 1s - loss: 7.3724e-04 - a
768/1572 [=====>.....] - ETA: 0s - loss: 6.8093e-04 - a
896/1572 [=====>.....] - ETA: 0s - loss: 6.1074e-04 - a
1024/1572 [=====>.....] - ETA: 0s - loss: 5.6965e-04 - a
1152/1572 [=====>.....] - ETA: 0s - loss: 5.1266e-04 - a
1280/1572 [=====>.....] - ETA: 0s - loss: 4.6726e-04 - a
1408/1572 [=====>.....] - ETA: 0s - loss: 4.5348e-04 - a
1536/1572 [=====>.....] - ETA: 0s - loss: 4.2564e-04 - a
1572/1572 [=====>.....] - 2s 1ms/step - loss: 4.2452e-04
- accuracy: 1.0000 - val_loss: 3.9699e-05 - val_accuracy: 1.0000
Epoch 14/15
128/1572 [=>.....] - ETA: 1s - loss: 9.2537e-04 - a
256/1572 [==>.....] - ETA: 1s - loss: 4.8087e-04 - a
384/1572 [=====>.....] - ETA: 1s - loss: 0.0015 - accur
512/1572 [=====>.....] - ETA: 1s - loss: 0.0012 - accur
640/1572 [=====>.....] - ETA: 1s - loss: 9.8595e-04 - a
768/1572 [=====>.....] - ETA: 0s - loss: 8.5950e-04 - a
896/1572 [=====>.....] - ETA: 0s - loss: 7.5155e-04 - a
1024/1572 [=====>.....] - ETA: 0s - loss: 6.7918e-04 - a
1152/1572 [=====>.....] - ETA: 0s - loss: 6.3063e-04 - a
1280/1572 [=====>.....] - ETA: 0s - loss: 6.1043e-04 - a
1408/1572 [=====>.....] - ETA: 0s - loss: 5.6445e-04 - a
1536/1572 [=====>.....] - ETA: 0s - loss: 5.2410e-04 - a
1572/1572 [=====>.....] - 2s 1ms/step - loss: 5.1252e-04
- accuracy: 1.0000 - val_loss: 3.1247e-04 - val_accuracy: 1.0000
Epoch 15/15
128/1572 [=>.....] - ETA: 1s - loss: 3.5633e-05 - a
256/1572 [==>.....] - ETA: 1s - loss: 1.4472e-04 - a
384/1572 [=====>.....] - ETA: 1s - loss: 3.3099e-04 - a
512/1572 [=====>.....] - ETA: 1s - loss: 2.5595e-04 - a
640/1572 [=====>.....] - ETA: 1s - loss: 2.1861e-04 - a
768/1572 [=====>.....] - ETA: 0s - loss: 1.9043e-04 - a
896/1572 [=====>.....] - ETA: 0s - loss: 1.7413e-04 - a
1024/1572 [=====>.....] - ETA: 0s - loss: 1.8065e-04 - a
1152/1572 [=====>.....] - ETA: 0s - loss: 1.6819e-04 - a
1280/1572 [=====>.....] - ETA: 0s - loss: 1.5829e-04 - a
1408/1572 [=====>.....] - ETA: 0s - loss: 1.4478e-04 - a
1536/1572 [=====>.....] - ETA: 0s - loss: 1.3462e-04 - a
1572/1572 [=====>.....] - 2s 1ms/step - loss: 1.3169e-04
- accuracy: 1.0000 - val_loss: 8.9247e-04 - val_accuracy: 1.0000
phatnguyen@phat-surface-pro:~/Desktop/small-dataset$ python test.py
```

Listing 6: Validating accuracy 100%

```
phatnguyen@phat-surface-pro: ~/Desktop/small-dataset
File Edit View Search Terminal Help
100%| 12/12 [00:00<00:00, 176602.27it/s]
100%| 12/12 [00:01<00:00, 6.81it/s]
100%| 13/13 [00:00<00:00, 245612.40it/s]
100%| 13/13 [00:01<00:00, 6.83it/s]
100%| 12/12 [00:00<00:00, 203771.85it/s]
100%| 12/12 [00:01<00:00, 6.82it/s]
100%| 12/12 [00:00<00:00, 131758.24it/s]
100%| 12/12 [00:01<00:00, 6.82it/s]
100%| 13/13 [00:00<00:00, 166746.03it/s]
100%| 13/13 [00:01<00:00, 6.82it/s]
100%| 12/12 [00:00<00:00, 150243.73it/s]
100%| 12/12 [00:01<00:00, 6.81it/s]
100%| 8/8 [00:00<00:00, 170327.07it/s]
100%| 8/8 [00:01<00:00, 6.82it/s]
100%| 4/4 [00:00<00:00, 93206.76it/s]
100%| 4/4 [00:00<00:00, 6.74it/s]
100%| 9/9 [00:00<00:00, 124995.81it/s]
100%| 9/9 [00:01<00:00, 6.82it/s]
100%| 9/9 [00:00<00:00, 94608.36it/s]
100%| 9/9 [00:01<00:00, 6.80it/s]
100%| 5/5 [00:00<00:00, 104335.92it/s]
100%| 5/5 [00:00<00:00, 6.81it/s]
100%| 7/7 [00:00<00:00, 149036.18it/s]
100%| 7/7 [00:01<00:00, 6.82it/s]
100%| 6/6 [00:00<00:00, 78889.73it/s]
100%| 6/6 [00:00<00:00, 6.83it/s]
100%| 5/5 [00:00<00:00, 108660.73it/s]
100%| 5/5 [00:00<00:00, 6.80it/s]
100%| 9/9 [00:00<00:00, 128397.06it/s]
100%| 9/9 [00:01<00:00, 6.84it/s]
100%| 114/114 [02:32<00:00, 1.34s/it]
('Correctly predicted:', 878)
('Total num frames:', 937)
('Test Accuracy:', 93.7033084311633)
phatnguyen@phat-surface-pro:~/Desktop/small-dataset$
```

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/>