Recognizing Hand Gestures

Anagha Fatale
Dept. of Electrical Engineering and Computer Science
Syracuse University
afatale@syr.edu,SUID 438039600

Namrata Arkalgud
Dept. of Electrical Engineering and Computer Science
Syracuse University
narkalgu@syr.edu,SUID 937006299

Abstract*—In the current world of automation, voice recognition is used to perform tasks like playing music, lowering room temperature etc. (example Alexa). This idea can further be extended to use hand gestures for performing tasks like switching off of light, closing door, typing, writing etc. This can also be extended to recognition when someone is waving for help or a baby is crawling out of its space in real time. In this paper, we have discussed different architectures we have implemented for recognising hand gestures using neural network models. The three architecture models implemented are 3D-CNN, LSTM and CNN-LSTM networks. Comparison of these architecture is described in this our paper. We have included in this paper the implementation, results and future work for the project.

I. Introduction

For this project, the data flow or overview of the implementation steps are: (1) Data segregation in to test and train. This split is provided by the dataset. Our job was to move the files into separate folders which we did using python. (2) CNN model requires images as input where as LSTM requires features extracted from these images. The next step was to extract images from videos and features from these images. Image extraction was done using *ffmpeg*, an open source multimedia tool. Its default frame rate is 40 fps. We then used Google's Inceptionv3 model pre trained with set weights on Imagenet dataset to get features for images from its final layer. This feature numpy output was then stored to be later passed as input to our LSTM model. (3) Data cleaning done was simple resizing all images to the same size along with rescaling. Here rescaling is nothing but having equal length of sequences of images from input videos. When we give videos of different length as input for image extraction, different counts of images are extracted for different videos. For training the model, each input videos will have equal length of sequence images. We have also performed one hot encoding for all labels. (4) Summarize the data in a csv file to be fed as input to the preprocessing steps. Video name, if it is test or train, label and frame count is the data in the csv. The implementation uses this data to fetch the images for each video, its label and data type. (5) Training the model with the data was done using three models 3D-CNN, LSTM, CNN-LSTM. (6) Predict using the three models to compare performance of the networks.

II. Dataset

We have implemented neural network for hand gesture recognition using data from UCF data set. The dataset contains videos like writing, typing. *There are 13,320 videos from 101 action categories. They are diverse in terms of actions, variations in camera motion, object appearance and pose. They are taken with cluttered background[4].* The link for the dataset is given below. Since the data set contains 101 classes and requires huge computational power (one epoch took 8 hours), we have trained with only 1/10th of the data set.

`http://crcv.ucf.edu/data/UCF101/UCF101.rar`

III. Architecture

A. 3D-Convolution

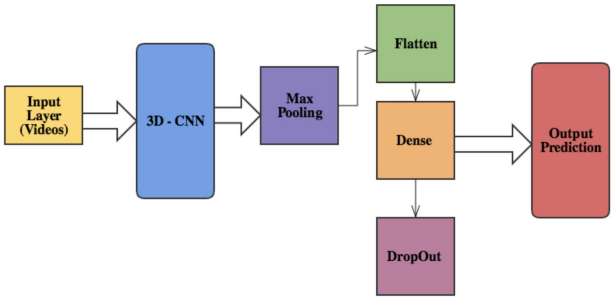

Fig. 1. Architecture of 3D-CNN

Figure 1. describes the 3D-CNN architecture which we implemented for our system. This architecture consists of five basic layers. First layer is input layer which is nothing but the layer in which videos are provided as input and extraction of images from videos is done. Next layer is the 3D-CNN layer. In 3D-CNN layer, a three dimensional filter to the dataset is applied and this filter moves in three direction to calculate the output. This output is three dimensional volume space such as a cube or cuboid.Next layer is of Maxpool 3D, this layer down samples the input data. *It also helps to avoid overfitting of the data by providing an abstract form of representation. It also reduces computational cost by reducing the number of parameters to learn[6]*. After flatten layer, there is dense layer. Dense layer, actually performs classification on the features extracted by convolution layer. Dropout layer is a regularization technique to reduce complexity of model with a goal of preventing overfitting. Using dropout we randomly deactivate certain units in input layer. Last layer is the prediction layer where the output is predicted on test data.

Layer (type)	Output Shape	Param #
conv3d_1 (Conv3D)	(None, 38, 78, 78, 32)	2624
max_pooling3d_1 (MaxPooling3	(None, 38, 39, 39, 32)	0
conv3d_2 (Conv3D)	(None, 36, 37, 37, 64)	55360
max_pooling3d_2 (MaxPooling3	(None, 36, 18, 18, 64)	0
conv3d_3 (Conv3D)	(None, 34, 16, 16, 128)	221312
conv3d_4 (Conv3D)	(None, 32, 14, 14, 128)	442496
max_pooling3d_3 (MaxPooling3	(None, 32, 7, 7, 128)	0
conv3d_5 (Conv3D)	(None, 31, 6, 6, 256)	262400
conv3d_6 (Conv3D)	(None, 30, 5, 5, 256)	524544
max_pooling3d_4 (MaxPooling3	(None, 30, 2, 2, 256)	0
flatten_1 (Flatten)	(None, 30720)	0
dense_1 (Dense)	(None, 1024)	31458304
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 1024)	1049600
dropout_2 (Dropout)	(None, 1024)	0
dense_3 (Dense)	(None, 5)	5125

Total params: 34,021,765
Trainable params: 34,021,765
Non-trainable params: 0

None

Fig. 2. Model summary of 3D-CNN

B. Long Short Term Memory

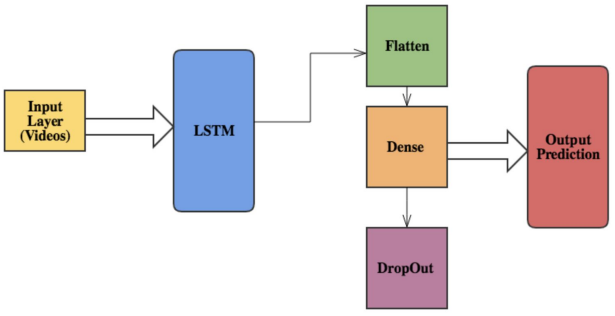

Fig. 3. Architecture of LSTM

Figure 3 describes the LSTM architecture for our model. In this architecture, LSTM layer is used instead of 3D-CNN layer. LSTM (Long Short Term Memory) recurrent type of neural networks. These store internal contextual state cells which act as long and short term memory. These cells are then further used for computing results for modulating the output. The other layers Flatten, dense and dropout remains the same as described in 3D-Convolution architecture. Last layer is output prediction layer.

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 2048)	33562624
dense_1 (Dense)	(None, 512)	1049088
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 5)	2565

Total params: 34,614,277
Trainable params: 34,614,277
Non-trainable params: 0

None

Fig. 4. Model summary of LSTM

C. CNN-LSTM

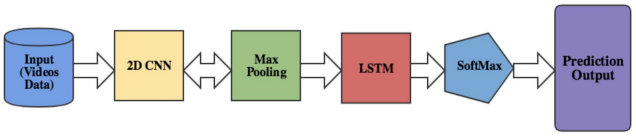

Fig. 5. Architecture of CNN-LSTM

Figure 5 describes the CNN-LSTM architecture we implemented for recognizing hands gestures. In this architecture, instead of 3D-CNN, 2D-CNN layer is applied as the first layer in the system. As

described earlier, Maxpool performs down sampling of input data and helps in avoiding overfitting. Fourth layer is of LSTM which helps to model longer temporal relations in the videos. Then softmax layer is applied to network before predicting the output.

Layer (type)	Output Shape	Param #
time_distributed_1 (TimeDist	(None, 40, 40, 40, 32)	4736
time_distributed_2 (TimeDist	(None, 40, 38, 38, 32)	9248
time_distributed_3 (TimeDist	(None, 40, 19, 19, 32)	0
time_distributed_4 (TimeDist	(None, 40, 19, 19, 64)	18496
time_distributed_5 (TimeDist	(None, 40, 19, 19, 64)	36928
time_distributed_6 (TimeDist	(None, 40, 9, 9, 64)	0
time_distributed_7 (TimeDist	(None, 40, 9, 9, 128)	73856
time_distributed_8 (TimeDist	(None, 40, 9, 9, 128)	147584
time_distributed_9 (TimeDist	(None, 40, 4, 4, 128)	0
time_distributed_10 (TimeDis	(None, 40, 4, 4, 256)	295168
time_distributed_11 (TimeDis	(None, 40, 4, 4, 256)	590080
time_distributed_12 (TimeDis	(None, 40, 2, 2, 256)	0
time_distributed_13 (TimeDis	(None, 40, 2, 2, 512)	1180160
time_distributed_14 (TimeDis	(None, 40, 2, 2, 512)	2359808
time_distributed_15 (TimeDis	(None, 40, 1, 1, 512)	0
time_distributed_16 (TimeDis	(None, 40, 512)	0
dropout_1 (Dropout)	(None, 40, 512)	0
lstm_1 (LSTM)	(None, 256)	787456
dense_1 (Dense)	(None, 5)	1285

Total params: 5,504,805
Trainable params: 5,504,805
Non-trainable params: 0

None

Fig. 6. Model summary of CNN-LSTM

IV. IMPLEMENTATION

Some of the libraries used are keras modules like keras.preprocessing for preprocessing images, keras.applications.inception_v3, keras.models,keras.optimizers,keras.layers.convolutional. The pseudo code for the logic flow given in introduction is given below *and is reproduced from [3].*

A. Data segregation

```
testFiles = os.path.join('ucfTrainTestlist', 'testlist' + '.txt')
trainFiles= os.path.join('ucfTrainTestlist', 'trainlist' + '.txt')
groups = {'train': trainVideosGotFromTrainFiles, 'test': testVideosGotFromTestFiles}
for each video in groups:
if not os.path.exists(os.path.join(trainOrTest, label)):
os.makedirs(os.path.join(trainOrTest, label))
dest = os.path.join(trainOrTest, Label, videoFileName)
os.rename(videoFileName, dest)
```

B. Image and feature extraction

```
for each video:
if not alreadyExtracted(video):
src = os.path.join(trainOrTest, label, videoFileName)
dest = os.path.join(trainOrTest, label, videoFileName + '.jpg')
call(["ffmpeg", "-i", videoSrc, imgDest])
```

C. Data cleaning

```
for each label/class:
encodedLabel = label.index(labelName)
labelHot=keras.utils.to_categorical(encodedLabel, len(self.labels))
```

D. CSV creation

```
for each video:
csvFile.append([trainOrTest,label, videoFileName, numberOfFrames])
writer = csv.writer("data_file.csv" in write mode)
writer.writerows(csvFile)
```

E. Model training

All models were compiled with Adam optimizer with a learning rate of 0.000001 and categorical cross entropy loss function.

```
Function train():
stepsInEpoch = (len(data) * 0.7)
x_train, y_train = data.getData('train', trainOrTest)
x_test, y_test = data.getData('test', trainOrTest)
model = Models(numberOfLabels, model, seqLength)
train = model.fit(x_train, y_train, batchSize, validation_data = (x_test, y_test), verbose=1, epochs=20)

Call function train:
model = 'conv_3d'
classLimit = 5
seqLength = 40
batchSize = 32
```

train(images, seqLength, model, classLimit, batchSize=batchSize)

F. Predicting

Results of the prediction are presented in the *Results* section. Only implementation of prediction logic is given here.

```
Function predict():
model = keras.models.load_model(savedModel)
sample=data.getImgFrames(videoFileName, imgOrFeature)
prediction=model.predict(np.expand_dims(sample, axis=0))

Call function predict():
model = 'conv_3d'
saved_model= 'data/checkpoints/conv_3d-images.080-1.589.hdf5
'videoFileName = 'v_BabyCrawling_g01_c02'
predict(savedModel,videoFileName, imgOrFeature='img')
```

V. RESULTS

The results of running the training on all three models for 20 epochs is given below.

TABLE I. Training Result comparison

Performance Parameters	Models		
	3D-CNN	***LSTM***	***CNN-LSTM***
Time taken (per step)	4 seconds	145 ms	459 ms
Accuracy	0.31	0.655	0.414
Loss	1.52	1.12	1.6

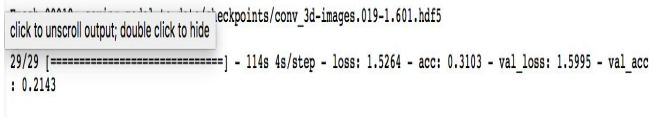

Fig. 7. 3D-CNN Training

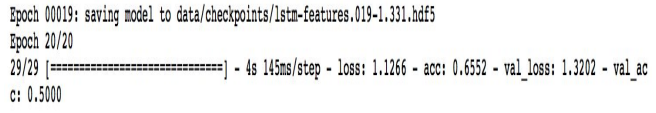

Fig. 8. LSTM Training

Fig. 9. CNN-LSTM Training

The prediction results are given in the below table for the above training performance parameters.

TABLE II. Prediction Result Comparison

Performance Parameters	Models		
	3D-CNN	***LSTM***	***CNN-LSTM***
Baby Crawling	0.26	0.46	0.22
BalanceBeam	0.23	0.29	0.22
Apply Makeup	0.22	0.16	0.20
Apply Lipstick	0.21	0.06	0.19
Archery	0.09	0.03	0.17

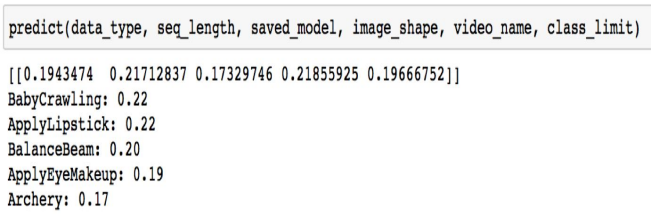

Fig. 10. CNN-LSTM Prediction

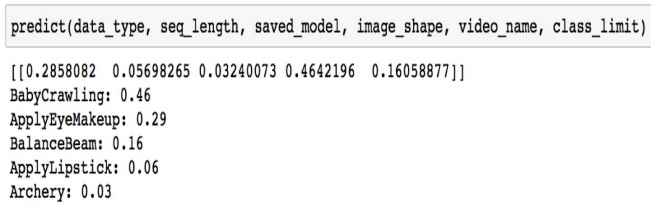

Fig. 11. LSTM Prediction

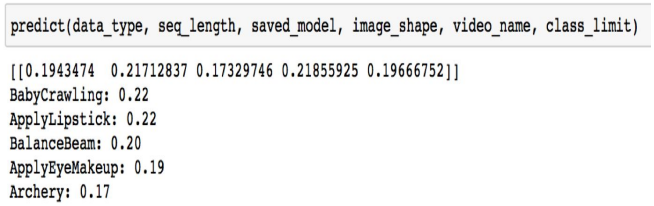

Fig. 12. 3D-CNN Prediction

VI. COMPARISON OF DIFFERENT MODELS

The LSTM model performs much better than the 3D CNN model which is the second best. We do not want to make conclusions on which model performs best as these models are trained with a very small dataset and for very little time as we were not able to obtain any machine with huge computational power. Their performances on train

and test data are shown in the graphs below. We believe the LSTM model works better than the 3D CNN because to the LSTM model we are feeding features got from running the pretrained Inceptionv3 model on the images. We believe that with an extensively trained 3D-CNN model, it should work better than LSTM.

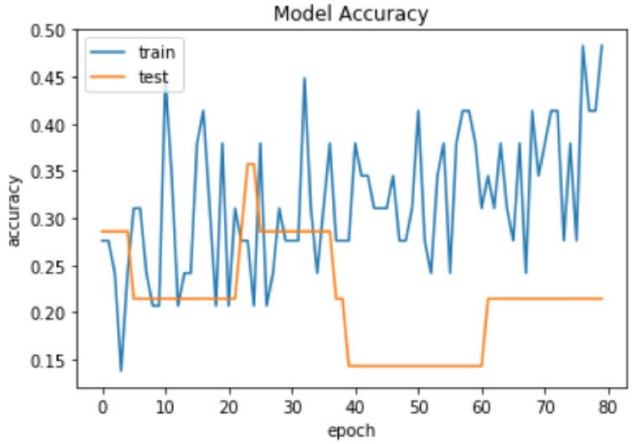

Fig. 13. Graph for accuracy v/s epochs on train and test data for CNN-LSTM

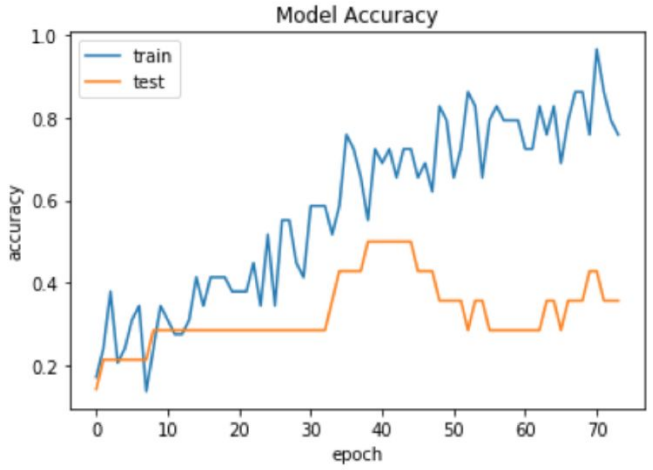

Fig. 14. Graph for accuracy v/s epochs on train and test data for 3D-CNN

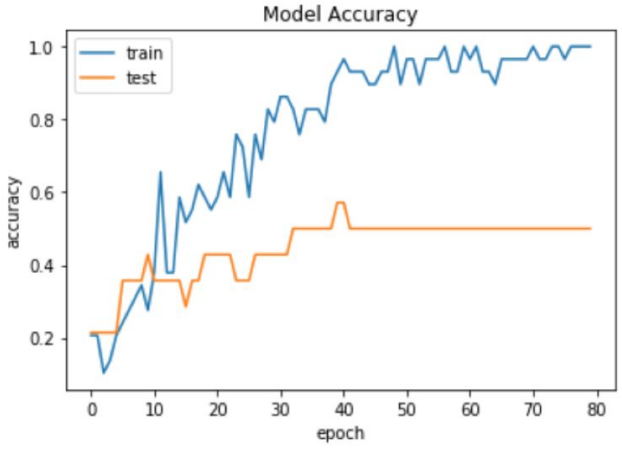

Fig. 15. Graph for accuracy v/s epochs on train and test data for LSTM

VII. Future Work

The current models are supervised learning models. We would like to work on an unsupervised model to be able to dynamically predict gestures outside the labels. We would like to be able to train the model extensively so that we have a saved model which can predict hand gestures live in real time. Working with a streaming data set would probably help with obtaining a better model i.e. with better accuracy and without overfitting. This could help with real world applications like alerting when in a video feed of a crowd, a person is waving for help.

VIII. References

[1] P. Molchanov, S. Gupta, J. Kautz, K. Kim and NVIDIA, “Hand Gesture Recognition with 3D Convolutional Neural Networks” Santa Clara, CA, USA

[2] Twenty Billion Neurons, “Gesture recognition using end-to-end learning from a large video database”, Sept 14th 2017
https://medium.com/twentybn/gesture-recognition-using-end-to-end-learning-from-a-large-video-database-2ecbfb4659ff

[3] Matt Harvey, “Video classification methods implemented in Keras”, Mar 21st 2017
https://blog.coast.ai/five-video-classification-methods-implemented-in-keras-and-tensorflow-99cad29cc0b5

[4] University of Central Florida, “UCF-101, Action Recognition Dataset”, Oct 17th 2013
http://crcv.ucf.edu/data/UCF101.php

[5] A lot issues were resolved using StackOverflow, GitHub issues pages.

[6] Quora website for explanation of use of Max Pooling3D.
https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks

IX. Appendix

Additionally, we have successfully implemented the other architectures like 3D-CNN and LSTM for our model. We have compared the results from all three architectures and found that model with LSTM works better for data set of UCF-101 classes in particular.