HUMAN ACTION RECOGNITION

212CS015

NIKITA PATIL

COMPUTER SCIENCE AND ENGINNERING | NITK

18/05/2022

Introduction

Due to increasing generation of video data, there is need for automated analysis of videos based on semantic interpretation. Thus, a well-established HAR system is integral in development of smart Computer vision systems. Both ConvLSTM and LRCN approaches are implemented and compared for this problem.

MOTIVATION

Human Activity Recognition has the potential to aid in video surveillance and anomaly detection. Automatic human action detection and recognition systems can aid in real-time CCTV video surveillance and reduce reliance on costly, labor-intensive manual analysis. In addition, human-machine interaction (HMI) could benefit greatly from human action recognition. Furthermore, human action recognition also has the potential to assist in behavior analysis and athletic rehabilitation. A variety of real-life mobile sensing applications are becoming available, especially in the life-logging, fitness tracking, and health monitoring domains. These applications use mobile sensors embedded in smartphones to recognize human activities in order to get a better understanding of human behavior. With CNN-LSTM approach, we are able to capture both temporal and spatial features of a video. Finally, an evaluation of the resulting model needs to be done.

PLAN OF WORK

1) Literature Survey
2) Visualize the Data with its Labels
3)Preprocess the Dataset
4) Split the Data into Train and Test Set
5) Implement the ConvLSTM Approach
6) Construct the Model
7) Compile and Train the Model
8) Plot Model's Loss and Accuracy Curves
9) Implement the LRCN Approach
10) Construct the Model
11) Compile and Train the Model
12) Plot Model's Loss and Accuracy Curves

LITERATURE SURVEY

[1] Shi, Xingjian & Chen, Zhourong & Wang, Hao & Yeung, Dit-Yan & Wong, Wai Kin & WOO, Wang-chun. (2015). Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting.

[2] J. Donahue et al., "Long-Term Recurrent Convolutional Networks for Visual Recognition and Description," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 4, pp. 677-691, 1 April 2017, doi: 10.1109/TPAMI.2016.2599174.

[3] C. J. Dhamsania and T. V. Ratanpara, "A survey on Human action recognition from videos," 2016 Online International Conference on Green Engineering and Technologies (IC-GET), 2016, pp. 1-5, doi: 10.1109/GET.2016.7916717.

[4] M. Zeng et al., "Convolutional Neural Networks for human activity recognition using mobile sensors," 6th International Conference on Mobile Computing, Applications and Services, 2014, pp. 197-205, doi: 10.4108/icst.mobicase.2014.257786.

DATASET

UCF50 - Action Recognition Dataset, consisting of realistic videos taken from youtube which differentiates this data set from most of the other available action recognition data sets as they are not realistic and are staged by actors.
The Dataset contains:
50 Action Categories
25 Groups of Videos per Action Category
133 Average Videos per Action Category
199 Average Number of Frames per Video
320 Average Frames Width per Video
240 Average Frames Height per Video
26 Average Frames Per Seconds per Video

Preprocessing dataset

After visualizing dataset, we preprocess it.

First, we will read the video files from the dataset and resize the frames of the videos to a fixed width and height, to reduce the computations and normalized the data to range [0-1] by dividing the pixel values with 255, which makes convergence faster while training the network.

We consider following 4 classes for our experiment: "WalkingWithDog", "TaiChi", "Swing", "HorseRace"

We extract frames from each video in above selected classes with no of frames equal to sequence length and prepare the dataset. We convert labels (class indexes) into one-hot encoded vectors. We split our data to create training and testing sets with (0.75,0.25) division. We will also shuffle the dataset before the split to avoid any bias and get splits representing the overall distribution of the data.

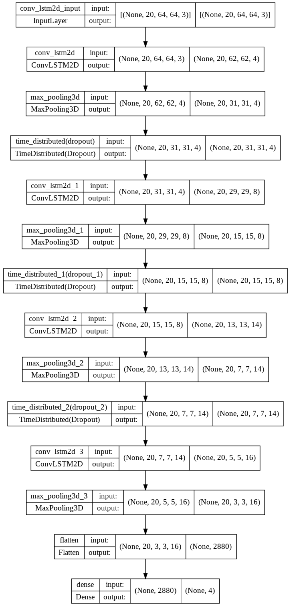
Implement the ConvLSTM Approach

We implement the first approach by using a combination of ConvLSTM cells. A ConvLSTM cell is a variant of an LSTM network that contains convolutions operations in the network. it is an LSTM with convolution embedded in the architecture, which makes it capable of identifying spatial features of the data while keeping into account the temporal relation.
For video classification, this approach effectively captures the spatial relation in the individual frames and the temporal relation across the different frames. As a result of this convolution structure, the ConvLSTM is capable of taking in 3-dimensional input: (width, height, num of channels) whereas a simple LSTM only takes in 1-dimensional input hence an LSTM is incompatible for modeling Spatio-temporal data on its own.

Implement the ConvLSTM Approach

To construct the model, we use Keras ConvLSTM2D recurrent layers. The ConvLSTM2D layer also takes in the number of filters and kernel size required for applying the convolutional operations. The output of the layers is flattened in the end and is fed to the Dense layer with softmax activation which outputs the probability of each action category. We also use MaxPooling3D layers to reduce the dimensions of the frames and avoid unnecessary computations and Dropout layers to prevent overfitting the model on the data. The architecture is a simple one and has a small number of trainable parameters. Then ,we add an early stopping callback to prevent overfitting and start the training after compiling the model. After training, we evaluate the model on the test set. Finally, we save the model. And plot training and validation loss, accuracy curves

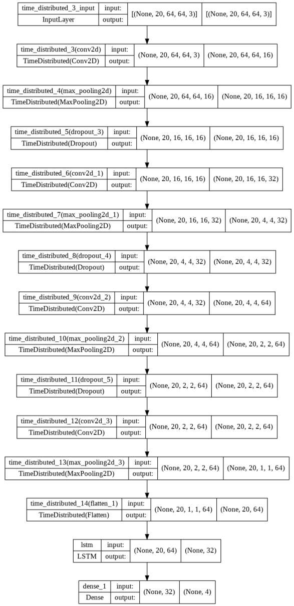
ConvLSTM


conv_lstm2d_input InputLayer input: output: [(None, 20, 64, 64, 3)] [(None, 20, 64, 64, 3)]
conv_lstm2d ConvLSTM2D input: output: (None, 20, 64, 64, 3) (None, 20, 62, 62, 4)
max_pooling3d MaxPooling3D input: output: (None, 20, 62, 62, 4) (None, 20, 31, 31, 4)
time_distributed(dropout) TimeDistributed(Dropout) input: output: (None, 20, 31, 31, 4) (None, 20, 31, 31, 4)
conv_lstm2d_1 ConvLSTM2D input: output: (None, 20, 31, 31, 4) (None, 20, 29, 29, 8)
max_pooling3d_1 MaxPooling3D input: output: (None, 20, 29, 29, 8) (None, 20, 15, 15, 8)
time_distributed_1(dropout_1) TimeDistributed(Dropout) input: output: (None, 20, 15, 15, 8) (None, 20, 15, 15, 8)
conv_lstm2d_2 ConvLSTM2D input: output: (None, 20, 15, 15, 8) (None, 20, 13, 13, 14)
max_pooling3d_2 MaxPooling3D input: output: (None, 20, 13, 13, 14) (None, 20, 7, 7, 14)
time_distributed_2(dropout_2) TimeDistributed(Dropout) input: output: (None, 20, 7, 7, 14) (None, 20, 7, 7, 14)
conv_lstm2d_3 ConvLSTM2D input: output: (None, 20, 7, 7, 14) (None, 20, 5, 5, 16)
max_pooling3d_3 MaxPooling3D input: output: (None, 20, 5, 5, 16) (None, 20, 3, 3, 16)
flatten Flatten input: output: (None, 20, 3, 3, 16) (None, 2880)
dense Dense input: output: (None, 2880) (None, 4)

Implement the LRCN Approach

We implement another approach known as the Long-term Recurrent Convolutional Network (LRCN), which combines CNN and LSTM layers in a single model. The Convolutional layers are used for spatial feature extraction from the frames, and the extracted spatial features are fed to LSTM layer(s) at each time-steps for temporal sequence modeling. This way the network learns spatiotemporal features directly in an end-to-end training, resulting in a robust model. We will also use TimeDistributed wrapper layer, which allows applying the same layer to every frame of the video independently. So it makes a layer (around which it is wrapped) capable of taking input of shape (no of frames, width, height, num of channels) if originally the layer's input shape was (width, height, num of channels) which is very beneficial as it allows to input the whole video into the model in a single shot.
we use time-distributed Conv2D layers which will be followed by MaxPooling2D and Dropout layers. The feature extracted from the Conv2D layers will be then flattened using the Flatten layer and will be fed to a LSTM layer. The Dense layer with softmax activation will then use the output from the LSTM layer to predict the action being performed.

LRCN

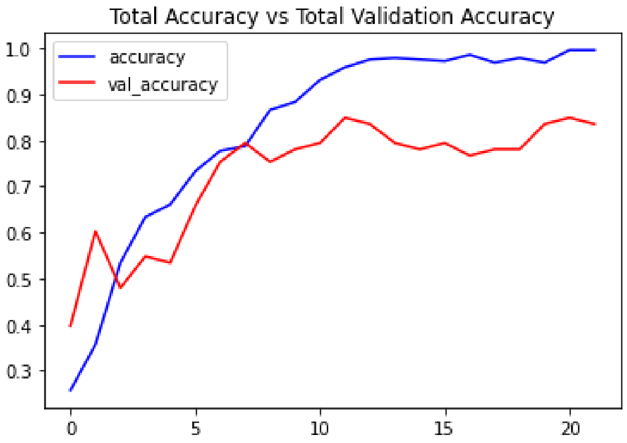

time_distributed_3_input | InputLayer | input: [(None, 20, 64, 64, 3)] | output: [(None, 20, 64, 64, 3)]
time_distributed_3(conv2d) | TimeDistributed(Conv2D) | input: (None, 20, 64, 64, 3) | output: (None, 20, 64, 64, 16)
time_distributed_4(max_pooling2d) | TimeDistributed(MaxPooling2D) | input: (None, 20, 64, 64, 16) | output: (None, 20, 16, 16, 16)
time_distributed_5(dropout_3) | TimeDistributed(Dropout) | input: (None, 20, 16, 16, 16) | output: (None, 20, 16, 16, 16)
time_distributed_6(conv2d_1) | TimeDistributed(Conv2D) | input: (None, 20, 16, 16, 16) | output: (None, 20, 16, 16, 32)
time_distributed_7(max_pooling2d_1) | TimeDistributed(MaxPooling2D) | input: (None, 20, 16, 16, 32) | output: (None, 20, 4, 4, 32)
time_distributed_8(dropout_4) | TimeDistributed(Dropout) | input: (None, 20, 4, 4, 32) | output: (None, 20, 4, 4, 32)
time_distributed_9(conv2d_2) | TimeDistributed(Conv2D) | input: (None, 20, 4, 4, 32) | output: (None, 20, 4, 4, 64)
time_distributed_10(max_pooling2d_2) | TimeDistributed(MaxPooling2D) | input: (None, 20, 4, 4, 64) | output: (None, 20, 2, 2, 64)
time_distributed_11(dropout_5) | TimeDistributed(Dropout) | input: (None, 20, 2, 2, 64) | output: (None, 20, 2, 2, 64)
time_distributed_12(conv2d_3) | TimeDistributed(Conv2D) | input: (None, 20, 2, 2, 64) | output: (None, 20, 2, 2, 64)
time_distributed_13(max_pooling2d_3) | TimeDistributed(MaxPooling2D) | input: (None, 20, 2, 2, 64) | output: (None, 20, 1, 1, 64)
time_distributed_14(flatten_1) | TimeDistributed(Flatten) | input: (None, 20, 1, 1, 64) | output: (None, 20, 64)
lstm | LSTM | input: (None, 20, 64) | output: (None, 32)
dense_1 | Dense | input: (None, 32) | output: (None, 4)

Evaluation of models on test data

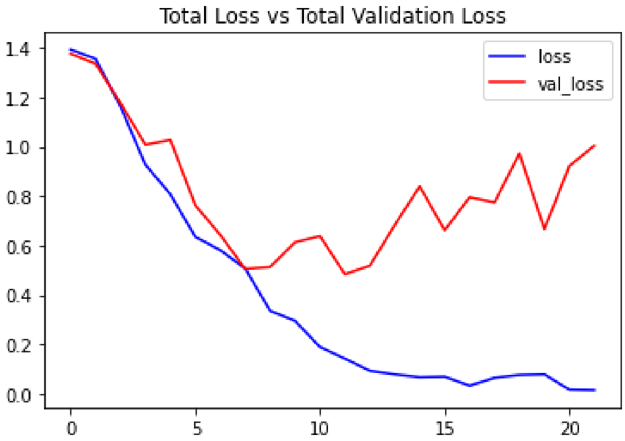
ConvLSTM:
1) accuracy: 0.7377
2) loss: 0.7642

LRCN:
1) accuracy: 0.8197
2) loss: 0.4610

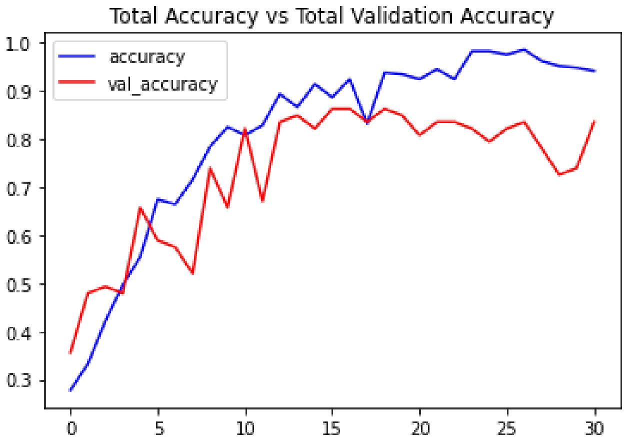
Plot for training and validation accuracy for ConvLSTM model



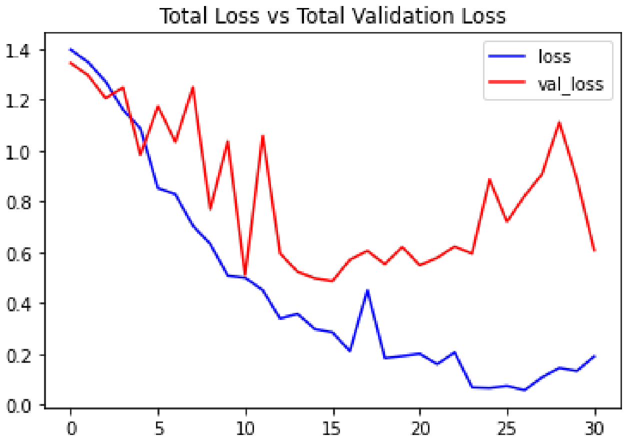
Plot for training and validation loss for ConvLSTM model



Plot for training and validation accuracy for LRCN model



Plot for training and validation loss for LRCN model



S/W REQUIREMENTS

Software requirements: Keras, Numpy, Matplotlib, Pandas, sklearn, TensorFlow

Thank you!