Human Interaction Recognition Using Deep Learning and Time-series Models

A mini project report submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

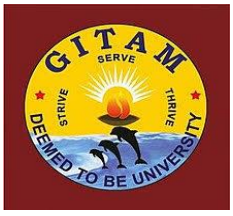
Submitted by

VEERAMALLA NITIN	**2215316458**
T. INDU PRABHAT	**2215316452**
AKULA SAI TEJA	**2215316465**
SANNEGANDLA SAIDA MASTAN	**2215316466**

Under the guidance of

Mrs. T. ARUNA SRI(M. Tech)

(Assistant Professor**)**



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM

(Deemed to be University)

HYDERABAD CAMPUS

APRIL-2019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM INSTITUTE OF TECHNOLOGY

GITAM

(Deemed to be University)

DECLARATION

I/We, hereby declare that the project report entitled "**Human Interaction Recognition Using Deep Learning and Time-series Models**" is an original work done in the Department of Computer Science and Engineering, GITAM Institute of Technology, GITAM (Deemed to be University) submitted in partial fulfilment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

VEERAMALLA NITIN

T. INDU PRABHAT

AKULA SAI TEJA

SANNEGANDLA SAIDA MASTAN

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM INSTITUTE OF TECHNOLOGY

GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that the project report **"Human Interaction Recognition Using Deep Learning and Time-series Models" is submitted by VEERAMALLA NITIN (2215316458), T. INDU PRABHAT (2215316452), AKULA SAI TEJA (2215316465), SANNEGANDLA SAIDA MASTAN (2215316466)** in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

Mrs. T. Aruna Sri **Prof. S. Phani Kumar**

(Project Guide) **(Project Coordinator)** **Head of the Department**

PANEL MEMBERS

1.

2.

3.

ACKNOWLEDGEMENT

Our project would not have been successful without the help of several people. we would like to thank the personalities who were part of our project in numerous ways, those who gave us outstanding support from the birth of the project.

We are extremely thankful to our honorable Pro-Vice Chancellor, Prof. N.Siva Prasad for providing necessary infrastructure and resources for the accomplishment of our project.

We are highly indebted to Prof. N. Seetharamaiah, Principal, School of Technology, for his support during the tenure of the project.

We are very much obliged to our beloved Prof. S. Phani Kumar, Head of the Department of Computer Science & Engineering for providing the opportunity to undertake this project and encouragement in completion of this project.

We hereby wish to express our deep sense of gratitude to Mrs. T. Aruna Sri, Department of Computer Science and Engineering, School of Technology for the esteemed guidance, moral support and invaluable advice provided by him for the success of the project.

We are also thankful to all the staff members of Computer Science and Engineering department who have cooperated in making our project a success. We would like to thank all our parents and friends who extended their help, encouragement and moral support either directly or indirectly in our project work.

Sincerely,

VEERAMALLA NITIN

T. INDU

PRABHAT AKULA SAI TEJA

SANNEGANDLA SAIDA MASTA

TABLE OF CONTENTS

ABSTRACT

Human Interaction Recognition Using Deep Learning and Time-series Models

Human Interaction has its applications in various fields like Surveillance, Human-Computer interaction, Threat Detection and Analysis and Recommendation System. Interaction can be observed on a Single Person (Ex: Gestures), Behavior of 2 People or Multi-Person Interaction. Understanding Interactions between 2 or more people is often more complex with respect to Understanding and Computation. Deep Learning Techniques can be applied to understand the features and Times series models can be used to recognize and action.

OpenCV (Computer Vision) is open source technology that can be used to convert motion video into frames of images which can be used to find out the key frames. For Extraction of Features into a Vector we can use CNN (AlexNet , CNN for small images), these features can be used to classify or label for frames using LSTM. The Features Extracted are used for Classification and Recognition by using activation function.

LIST OF FIGURES

1. INTRODUCTION

Computer vision is the field of computer science that focuses on replicating parts of the complexity of the human vision system and enabling computers to identify and process objects in images and videos in the same way that humans do. Until recently, computer vision only worked in limited capacity.

Thanks to advances in artificial intelligence and innovations in deep learning and neural networks, the field has been able to take great leaps in recent years and has been able to surpass humans in some tasks related to detecting and labeling objects. One of the driving factors behind the growth of computer vision is the amount of data we generate today that is then used to train and make computer vision better.

On a certain level Computer vision is all about pattern recognition. So one way to train a computer how to understand visual data is to feed it images, lots of images thousands, millions if possible that have been labeled, and then subject those to various software techniques, or algorithms, that allow the computer to hunt down patterns in all the elements that relate to those labels.

1.1 PURPOSE OF THE PROJECT

We work on UT-interaction dataset to be able to classify 6 action classes namely Handshaking, Kicking, Punching, Hugging, Pointing and Pushing. This enables us to be able to apply this concept in different areas of the Real-world Scenarios. Being able to understand the sequence of pattern that lead to an interaction is primary objective of this project.

1.2 OVERVIEW OF THE PROJECT

We will be using UT-Interaction Dataset as our Standard dataset to perform the Modelling and classification.

1.2.1 UT INTERACTION DATASET

The UT-Interaction dataset contains videos of continuous executions of 6 classes of

human-human interactions: shake-hands, point, hug, push, kick and punch. Ground truth labels for these interactions are provided, including time intervals and bounding boxes. There is a total of 20 video sequences whose lengths are around 1 minute. Each video contains at least one execution per interaction, providing us 8 executions of human activities per video on average. Several participants with more than 15 different clothing conditions appear in the videos. The videos are taken with the resolution of 720*480, 30fps, and the height of a person in the video is about 200 pixels.

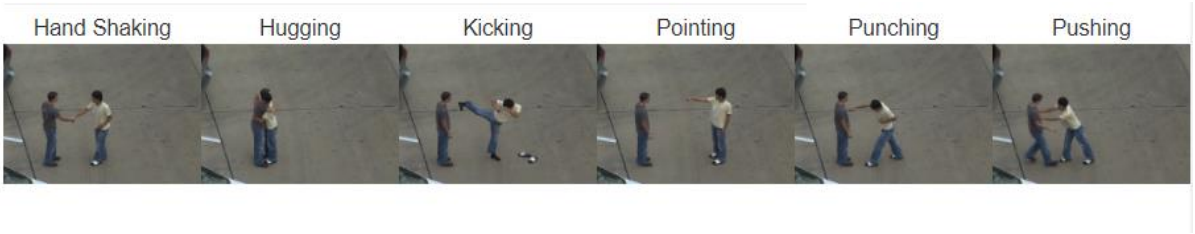


Figure 1.1 Six Action Sequences

1.2.2 PROCESSING OF VIDEOS

To Reduce the memory load on the ram we convert these videos into Keyframes, these Keyframes are then Loading into the memory by converting them into a numpy array. This is then saved into a Pickel Object.

1.2.3 MODEL AND CLASSIFICATION

The Pickel object is then reshaped into train-test splits and feed into the Convolution Neural Network to extract features from the Last Layer(i.e before the classification layer).These features are reshaped into blocks and then feed into and LSTM model to get the Output Classifications.

2. LITERATURE REVIEW

Many Deep Learning articles and tutorials primarily focus on three data domains: images, speech, and text. These data domains are popular for their applications in image classification, speech recognition, and text sentiment classification. Another very interesting data modality is video. From a dimensionality and size perspective, videos are one of the most interesting data types alongside datasets such as social networks or genetic codes. Video uploading platforms such as YouTube are collecting enormous datasets, empowering Deep Learning research.

Deep Learning is not new concept but has been researched since the mid-19th Century. There have been a lot of developments and advancements in the AI, ML, and DL fields over the past 60 years.
To boil it down to a rough timeline, deep learning might look something like this:

- 1960s: Shallow neural networks
- 1960-70s: Backpropagation emerges
- 1974-80: First AI Winter
- 1980s: Convolution emerges
- 1987-93: Second AI Winter
- 1990s: Unsupervised deep learning
- 1990s-2000s: Supervised deep learning
- 2006s-present: Modern deep learning

There are two immediate concerns relating to the video datasets used. The first is the enormous size of the dataset, thus making the prospect of loading the entire dataset into local memory impractical. A workaround for this is to use a URL parsing library to dynamically download the videos from their YouTube links and overwrite the videos currently in memory which were used in a previous batch size.

A second concern with this dataset is one which is frequently found with text mining applications, the variable length of each instance. For example, one video might be 30 seconds, while another is 2 minutes.

2.1 EXISTING METHODS

There are various methods that can help classify Videos few of them are the following:

- Classifying one frame at a time with a ConvNet.
- Using a time-distributed ConvNet and passing the features to an RNN, in one network.
- Using a 3D convolutional network.
- Extracting features from each frame with a ConvNet and passing the sequence to a separate RNN.
- Extracting features from each frame with a ConvNet and passing the sequence to a separate MLP.
- Multi-Stream Networks

Since a video is just a series of frames, A General Algorithm for Video Classification is:
Loop over all frames in the video file.
For each frame, pass the frame through the CNN.
Classify each frame individually and independently of each other.
Choose the label with the largest corresponding probability.
Label the frame and write the output frame to disk.

2.2 RELATED WORKS

There have been previous works on this field of computer vision, to be more specific in the terms of action classification/recognition. One important thing we must remember the difference between action and interaction. Action is the behavior of a single person whereas interaction is the collective behavior of multiple people. In this project we focus on 2-person Interactions.

There are 2 different types of subtasks that is needed to classify Interactions namely, Skeleton based Action Recognition and 3d Action Recognition. Both of them proved to be very useful in terms of accurate and efficient Predictions. In the past year, Advancement in technology and Computation helped in developing an open source library OpenPose which can detect skeletons of people in real-time. Human interaction recognition has been widely studied because it has great scientific importance and many potential practical applications.

However, recognizing human interactions is also very challenging especially in realistic environments where the background is dynamic or has varying lighting conditions. Most existing methods rely on either spatio-temporal local features (i.e. SIFT) or human poses, or human joints to model human interactions. As a result, they are not fully unsupervised processes because they require either hand-designed features or human detection results. Often the Challenge would be in Understanding the patterns in both spatial and sequential.

3. PROBLEM IDENTIFICATION & OBJECTIVES

There are Certain Challenges involved in this project for example local machine computation limitations, accuracy results , letting the model learn patterns.

3.1 PROBLEM STATEMENT

- Research on the best combination of models that yield high accuracy or better model performance on a data set.
- Also, The model combination that give the least Computational Complexity.
- Implementation of real-time multi-person Interaction Recognition.

3.2 TECHNOLOGIES USED

OpenCV- OpenCV (Open Source Computer Vision Library: http://opencv.org) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. The document describes the so-called OpenCV 2.x API, which is essentially a C++ API, as opposed to the C-based OpenCV 1.x API (C API is deprecated and not tested with "C" compiler since OpenCV 2.4 releases).OpenCV has a modular structure, which means that the package includes several shared or static libraries.

Python Programming Language- Python is a great object-oriented, interpreted, and interactive programming language. It is often compared to Lisp, Tcl, Perl, Ruby, C#, Visual Basic, Visual Fox Pro, Scheme or Java... and it's much more fun. Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing.
There are interfaces to many system calls and libraries, as well as to various windowing systems. New built-in modules are easily written in C or C++ (or other languages, depending on the chosen implementation). Python is also usable as an extension language for applications written in other languages that need easy-to-use scripting or automation interfaces.

TensorFlow- TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that

lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

Keras- Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.
It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research, if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

3.3 FIELD OF THE PROJECT

Each of the application areas described above employ a range of computer vision tasks; more or less well-defined measurement problems or processing problems, which can be solved using a variety of methods. Some examples of typical computer vision tasks are presented below.

Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, *e.g.*, in the forms of decisions. Understanding in this context means the transformation of visual images (the input of the retina) into descriptions of the world that can interface with other thought processes and elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

Several tasks relate to motion estimation where an image sequence is processed to produce an estimate of the velocity either at each points in the image or in the 3D scene, or even of the camera that produces the images.

4. SYSTEM METHODOLGY

4.1 WORK FLOW OF THE PROJECT

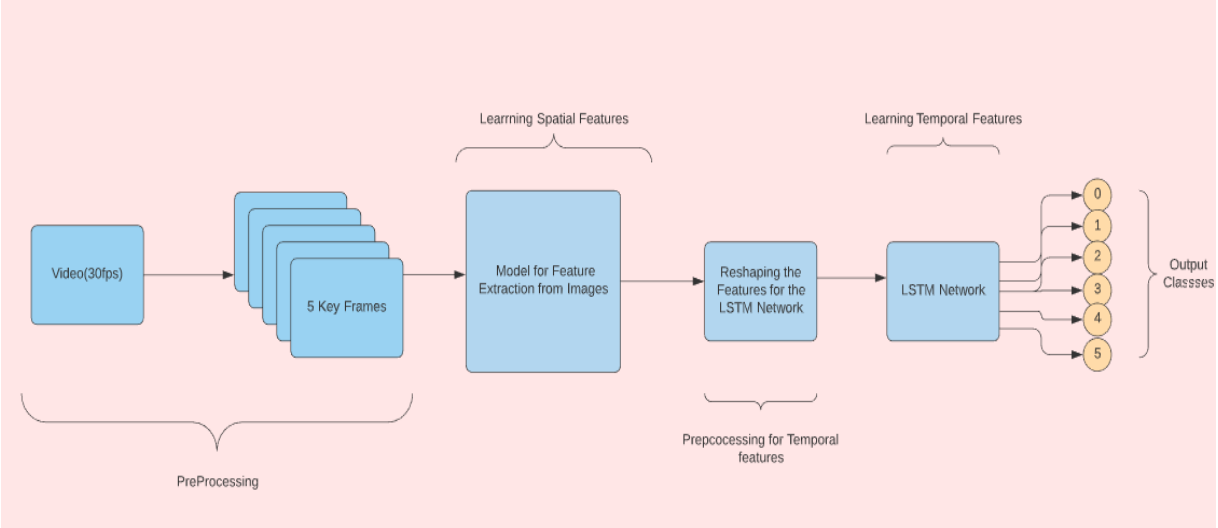


Figure 4.1 Work Flow

4.1.1 ORGANIZING THE DATASET

We split the UT-Interaction dataset into train(80%) and validation(20%) files respectively.
Train folder-96 videos
Test folder-24 videos
We have equal sets of same class videos therefore we will not be facing the class imbalance problem.

4.1.2 KEY-FRAME EXTRACTION

For Key frames we are using DFKE which is Differential key evolution algorithm [Appendix A] which is genetic algorithm that iterates through all the frames and returns best possible parent frames that are differentiated using Euclidean Distance.
This helps us reduce the manual extraction of Key frames
We can also use Entropy as our metric for differentiating but Euclidian distance helps perceptually differentiate the frames.

4.1.3 CONVERTING FRAMES INTO NUMPY ARRAYS

It is always advisable to load our data into main memory before feeding it to the neural network therefore we use a dataframe of shape samples x shape of the sample x no of channels i.e (num_sample,dim=[widthxheight],num_of_channels).
Channels here are 3 as we are dealing with rgb images ,we can also use b/w images for faster train time.

4.1.4 BUILDING FEATURE EXTRACTION MODELS

Feature extraction is a process of dimensionality reduction by which an initial set of raw data is reduced to more manageable groups for processing. A characteristic of these large data sets is a large number of variables that require a lot of computing resources to process. Feature extraction is the name for methods that select and /or combine variables into features, effectively reducing the amount of data that must be processed, while still accurately and completely describing the original data set.

The process of feature extraction is useful when you need to reduce the number of resources needed for processing without losing important or relevant information. Feature extraction can also reduce the amount of redundant data for a given analysis. Also, the reduction of the data and the machine's efforts in building variable combinations (features) facilitate the speed of learning and generalization steps in the machine learning process
This is done by Popping the last layer form the Image Recognition Model and predicting the features from the last layer.

4.1.5 RESHAPING THE DATA TO BE LSTM READY

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work.1 They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!
We have to remember that the LSTM input shape is 3D tensor. We split and re-append the

dataset to shape the data as(num_samples x num_seq x num_features).In our case the shape of test data set would be (96 x 5 x 512).

4.1.6 FEEDING DATA TO LSTM NETWORK

Training a LSTM would be faster if we use Dropout and Normalization Layers in our Network.
We have to experiment with different epochs and learning rates(adam,adagrad etc).

4.1.7 ANALYZING THE RESULTS

To Analyze our results we will be plotting accuracy vs epochs and loss vs epochs for both test and train sets so that we know whether the model Is learning properly or not

5. OVERVIEW OF TECHNOLOGIES

5.1 SETTING UP CONDA ENVIRONMENT

Anaconda is a virtual environment that enables us to run several different version of python using the conda environment.
We install Anaconda and create a new environment for our tensor flow project, it is always advisable to create a new conda environment before installing packages on to your base environment.
Anaconda provides us with several IDEs such as Jupyter Notebook, Spyder, VS Code etc

Run the following in cmd after installing the Anaconda:
conda env create -v -f tensorflow.yml

python -m ipykernel install --user --name tensorflow --display-name "Python 3.7 (tensorflow)"

.yml files contain information about the package versions

5.2 PACKAGES REQUIRED

These are the necessary packages required:
name: tensorflow_gpu
dependencies:

- python=3.7(general python files)
- pip>=19.0(for easy installation)
- jupyter(notebook for interactive environment)
- tensorflow-gpu=2.0(running models on GPU)
- scikit-learn(General Machine Learning utils)
- scipy(Useful Scientific functions)
- pandas(Dataframes)
- pandas-datareader
- matplotlib(Ploting images and graphs)
- pillow

- tqdm
- requests
- h5py(saving models ie weights/biases)
- pyyaml(reading .yaml files)
- flask(to deploy models)
- opencv-python

5.3 HARDWARE SPECIFICATIONS

This Project was run on a System with
- 8GB DDR4 RAM
- 4GB GTX 1650Ti
-i7 2.60GHZ(12CPUs)

An Excellent Alternative would be Google Cloab which provides us with free GPU AND TPU if available (No Installation Required for Colab)

5.4 SOFTWARE SPECIFICATION

The following NVIDIA® software must be installed on your system:

NVIDIA® GPU drivers —CUDA 10.1 requires 418.x or higher.
CUDA® Toolkit —TensorFlow supports CUDA 10.1 (TensorFlow >= 2.1.0)
CUPTI ships with the CUDA Toolkit.
cuDNN SDK (>= 7.6)
(Optional) TensorRT 6.0 to improve latency and throughput for inference on some models.

6. IMPLEMENTATION

We import necessary the Modules and load in all the helper functions

In [2]:
```
try:

    # --------General------------
    import cv2 as cv2
    import os
    import pickle
    import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
    import time as t

    # ----sklearn----------------
    from sklearn.utils import shuffle
    from sklearn.model_selection import train_test_split

    from keras.utils import np_utils
    # --------Tensorflow---------
    import tensorflow as tf


    # --------Keras--------------
    import keras
    from keras import backend as K
    # K.set_image_dim_ordering('th')
    from keras.callbacks import ModelCheckpoint, EarlyStopping
    from keras import layers, callbacks
    from keras.models import Model
    from keras.models import Sequential
    from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, BatchNormalization,LSTM
    from keras.optimizers import SGD, RMSprop, Adam, Adagrad, Adadelta, RMSprop
    print("-------------------Libraries Loaded Successfully!!!----------------------------------------")
except:
    print("Library not Found ! ")
```

```
Using TensorFlow backend.
```

```
-------------------Libraries Loaded Successfully!!!----------------------------------------
```

Figure 6.1 Importing Necessary Modules

6.1 CONVERTING VIDEOS INTO KEY FRAMES

To Convert the Video into Key-Frames we use Differential Evolution Algorithm with Euclidean distance as the differentiating metric.

The Reference paper Says the 5 key-frames can ideally represent an action.

Each Video is converted into 5 Key-frames that retain the sequence action information

6.2 KEYFRAME-IMAGES DUMPED INTO PICKLE OBJECT

The 5 frames are converted into into a numpy array and stored in a dataframe.

The datatype is float32.

In [14]:

```
epochs =  30
batch_size = int(.25 * 480)
if batch_size > 128:
    batch_size = 128 # batch size caps at 128

print("Shape of train set and labels:",X_train.shape,Y_train.shape)
print("Shape of test set and labels:",X_test.shape,Y_test.shape)



x_train = X_train.astype('float32')
x_test = X_test.astype('float32')
x_train /= 255
x_test /= 255

#input shape
input_shape=X_train[0].shape
print("Input Shape of Each Image:",input_shape)
```

```
Shape of train set and labels: (480, 64, 64, 3) (480, 6)
Shape of test set and labels: (120, 64, 64, 3) (120, 6)
Input Shape of Each Image: (64, 64, 3)
```

Figure 6.2 Shaping The Data for Neural Network

6.3 BUILDING KERAS MODELS

- Conv2d()-performs convolution operations on the input tensor..
- Batch Normalization()-Normalizes the data for faster training.
- Dense()-simple layer with n number of neurons.
- Dropout ()-randomly drops a portion of neurons to speed up training process.
- MaxPooling2d()-given the size of kernel it reduces the shape for input tensor.

6.3.1 ALEXNET

AlexNet is the name of a convolutional neural network (CNN), designed by Alex Krizhevsky, and published with Ilya Sutskever and Krizhevsky's doctoral advisor Geoffrey Hinton.

AlexNet competed in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012. The network achieved a top-5 error of 15.3%, more than 10.8

percentage points lower than that of the runner up. The original paper's primary result was that the depth of the model was essential for its high performance, which was computationally expensive, but made feasible due to the utilization of graphics processing units (GPUs) during training.

In [15]:

```
model = Sequential()
model.add(Conv2D(64, (11,11),padding='same', input_shape=(64,64,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (5,5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(512, (3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(1024))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5,name='last_layer'))

model.add(Dense(6))
model.add(BatchNormalization())
model.add(Activation('softmax'))

model.summary()
```

```
Model: "sequential_1"
```

Figure 6.3 Modified AlexNet

AlexNet Contains 5 convolution layers and 3 fully connected layers and standard input shape is 224x224x3.We optimized out Alexnet with 4 convolution layers and 2 fully connected layers for an input of 64x64 which helped reduce the computation costs.

We Observed that the deeper our convolutions layers the lesser feature we extract for 64x64 images therefore removing 1 Convolution layer and 1 Fully Connected Layer.

6.3.2 CNN FOR 64X64 IMAGES

This our own model which we landed on after trying several hyper parameters

We Follow a similar approach to AlexNet we perform 2 Convolutions followed by a MaxPoolig 2D initially.

Then Use Maxpooling after every Conv Layer

```
model = Sequential()

model.add(Conv2D(32, (5,5),strides=(1,1),padding='same', input_shape=(64,64,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Conv2D(32, (5,5),strides=(1,1),padding='same', input_shape=(64,64,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))


model.add(Conv2D(64, (3,3),strides=(1,1),padding='same', input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Conv2D(64, (3,3),strides=(1,1),padding='same', input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))


model.add(Conv2D(96, (3,3),strides=(1,1),padding='same', input_shape=(16,16,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Conv2D(96, (3,3),strides=(1,1),padding='same', input_shape=(16,16,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))


model.add(Conv2D(128, (3,3),strides=(1,1),padding='same', input_shape=(8,8,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Conv2D(128, (3,3),strides=(1,1),padding='same', input_shape=(8,8,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
```

Figure 6.4 Convolutions layers for Neural Network

The 2 Fully Connected layers are added to the Network after Flattening then finally a softmax layer to classify into outputs.

```
model.add(Flatten())
model.add(Dense(4096))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(1024))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5,name='last_layer'))

model.add(Dense(6))
model.add(BatchNormalization())
model.add(Activation('softmax'))

model.summary()
```

Figure 6.4 Fully Connected Layers For The Neural Network

6.4 FEATURE EXTRACTION

Once The model is Trained we extract all the features from the last layer by running prediction()

On the training and testing samples.

We simply save the model from the initial layer to the last but 1 layer there by saving all the 512 features to another dataframe.

```
In [17]:
# define the layer for feature extraction
intermediate_layer = Model(inputs=model.input, outputs=model.get_layer('last_layer').output)

feature_engineered_train = intermediate_layer.predict(x_train)
feature_engineered_train = pd.DataFrame(feature_engineered_train)

feature_engineered_test = intermediate_layer.predict(x_test)
feature_engineered_test = pd.DataFrame(feature_engineered_test)


print(feature_engineered_train.shape)
print(feature_engineered_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(480, 512)
(120, 512)
(480, 6)
(120, 6)
```

Figure 6.5 Extracting Features From The Neural Network

6.5 RESHAPING THE DATA FOR LSTM

We Shape the data into blocks of 5 features each representing a sequence as the LSTM cannot input (num_samples,num_features) directly, simply speaking it requires another input shape to be fed into the Network.

The LSTM only take in (num_of_data_samples, num_sequences,num_features) as its input.
The data_blocks() returns a numpy array of shape (num_of_data_samples, num_sequences,num_features).

The label_blocker() function returns the label in blocked format 5 labels are clubbed into 1 as all the 5 are representing the same Data_sample.

This part of the Code is same for both the Networks as the input shape for the LSTM is the same for both the Networks

```
In [21]: def data_blocks(n,x):
             blocks=int(len(x)/n)
             X_train=np.array(np.split(x,blocks))
             print(X_train.shape,type(X_train))
             return X_train

In [22]: X=data_blocks(5,X)

         (120, 5, 512) <class 'numpy.ndarray'>

In [23]: def label_blocker(n,y):
             Y=[]
             for i in range(0,len(y),5):
                 Y.append(y[i])
             return Y

In [24]: y=label_blocker(5,Features_labels)
         type(y)

Out[24]: list
```

Figure 6.6 Helper Functions For Reshaping the Data

6.6 TRAINING LSTM

Since Our Data is ready for being fed into the LSTM Network we then create a LSTM model that can finally give us our model classifications.
One thing to add is a dropout layer which significantly help in decreasing the train time.

In [30]:
```
model1=Sequential()
model1.add(LSTM(512,input_shape=(5,512),activation='sigmoid',return_sequences=False))
model1.add(Dropout(0.2))
model1.add(Dense(256,activation='sigmoid'))

#model1.add(Dense(100,activation='sigmoid'))

model1.add(Dense(6,activation='softmax'))
```

In [31]:
```
# Training the Model
```

In [32]:
```
model1.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

s=t.time()
model1.fit(X_train,y_train,validation_data=[X_test,y_test],epochs=15)
e=t.time()
print("traintime-",((e-s)/60))
```

```
Train on 96 samples, validate on 24 samples
```

Figure 6.7 LSTM Model for AlexNet

In [41]:
```
model1=Sequential()
model1.add(LSTM(512,input_shape=(5,512),activation='sigmoid',return_sequences=False))
model1.add(Dropout(0.2))
model1.add(Dense(256,activation='sigmoid'))

model1.add(Dense(100,activation='sigmoid'))

model1.add(Dense(6,activation='softmax'))
```

In [42]:
```
# Training the Model
```

In [43]:
```
model1.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

s=t.time()
model1.fit(X_train,y_train,validation_data=[X_test,y_test],epochs=20)
e=t.time()
print("traintime-",((e-s)/60))
```

```
Train on 96 samples, validate on 24 samples
```

Figure 6.8 LSTM Model for CNN

7. RESULTS

7.1 ALEXNET+LSTM

The Model is trained for 15 epochs returns results as follows

```
Epoch 8/15
96/96 [==============================] - 0s 499us/step - loss: 0.3555 - accuracy: 0.8854 - val_loss: 0.3614 - val_accuracy: 0.8
750
Epoch 9/15
96/96 [==============================] - 0s 486us/step - loss: 0.3230 - accuracy: 0.8750 - val_loss: 0.3855 - val_accuracy: 0.8
750
Epoch 10/15
96/96 [==============================] - 0s 528us/step - loss: 0.2835 - accuracy: 0.8958 - val_loss: 0.3968 - val_accuracy: 0.8
750
Epoch 11/15
96/96 [==============================] - 0s 512us/step - loss: 0.2833 - accuracy: 0.8854 - val_loss: 0.4401 - val_accuracy: 0.8
750
Epoch 12/15
96/96 [==============================] - 0s 500us/step - loss: 0.2437 - accuracy: 0.9062 - val_loss: 0.4321 - val_accuracy: 0.8
750
Epoch 13/15
96/96 [==============================] - 0s 509us/step - loss: 0.2298 - accuracy: 0.9167 - val_loss: 0.4153 - val_accuracy: 0.8
750
Epoch 14/15
96/96 [==============================] - 0s 478us/step - loss: 0.2111 - accuracy: 0.9271 - val_loss: 0.3870 - val_accuracy: 0.8
750
Epoch 15/15
96/96 [==============================] - 0s 509us/step - loss: 0.1926 - accuracy: 0.9271 - val_loss: 0.4150 - val_accuracy: 0.8
750
traintime- 0.07752947807312012
```

Figure 7.1 Training Results for AlexNet+LSTM

7.1.1 PLOTTING METRICS

Once we train the model we story all the metrics it returns in an object and use matplotlib to plot the following graphs

Accuracy vs epochs and loss vs epochs for both train and test splits

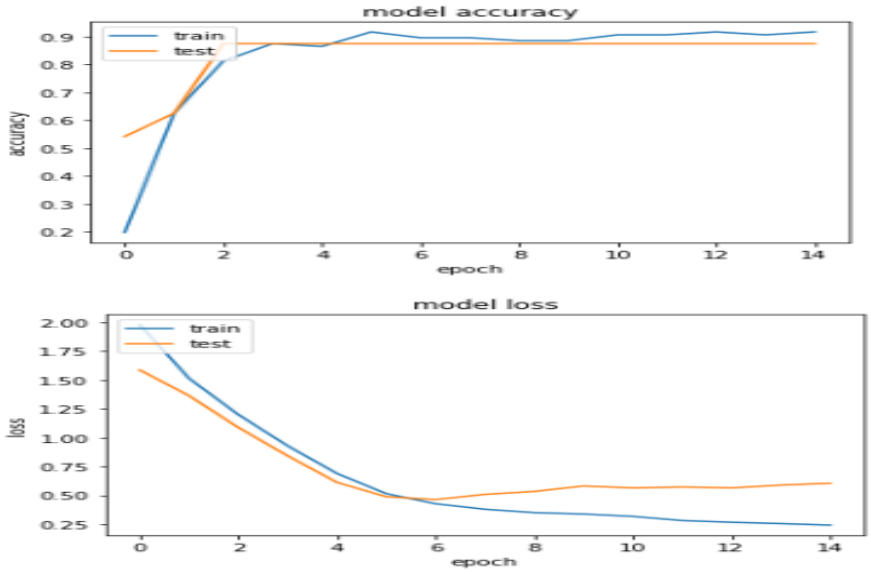


Figure 7.2 Graphs for Alex+LSTM

By observing the graphs we can say the for increase in accuracy the loss is gradually

decreasing hence the model was able to learn both spatio-temporal patterns.

7.2 CNN FOR SMALL IMAGES+LSTM

The Model is trained for 20 epochs returns results as follows

```
Epoch 9/20
96/96 [==============================] - 0s 519us/step - loss: 0.6792 - accuracy: 0.8646 - val_loss: 0.6300 - val_accuracy: 0.9
167
Epoch 10/20
96/96 [==============================] - 0s 509us/step - loss: 0.5931 - accuracy: 0.8750 - val_loss: 0.5707 - val_accuracy: 0.9
167
Epoch 11/20
96/96 [==============================] - 0s 497us/step - loss: 0.5267 - accuracy: 0.8854 - val_loss: 0.5362 - val_accuracy: 0.9
167
Epoch 12/20
96/96 [==============================] - 0s 499us/step - loss: 0.4647 - accuracy: 0.8854 - val_loss: 0.4946 - val_accuracy: 0.9
167
Epoch 13/20
96/96 [==============================] - 0s 519us/step - loss: 0.4238 - accuracy: 0.9062 - val_loss: 0.4583 - val_accuracy: 0.9
167
Epoch 14/20
96/96 [==============================] - 0s 561us/step - loss: 0.3841 - accuracy: 0.8854 - val_loss: 0.4369 - val_accuracy: 0.9
167
Epoch 15/20
96/96 [==============================] - 0s 499us/step - loss: 0.3420 - accuracy: 0.9271 - val_loss: 0.4294 - val_accuracy: 0.9
167
Epoch 16/20
96/96 [==============================] - 0s 519us/step - loss: 0.3201 - accuracy: 0.9167 - val_loss: 0.4253 - val_accuracy: 0.9
167
Epoch 17/20
96/96 [==============================] - 0s 493us/step - loss: 0.2900 - accuracy: 0.9271 - val_loss: 0.4091 - val_accuracy: 0.9
167
Epoch 18/20
96/96 [==============================] - 0s 488us/step - loss: 0.2694 - accuracy: 0.9479 - val_loss: 0.4242 - val_accuracy: 0.8
750
Epoch 19/20
96/96 [==============================] - 0s 499us/step - loss: 0.2503 - accuracy: 0.9479 - val_loss: 0.4318 - val_accuracy: 0.8
750
Epoch 20/20
96/96 [==============================] - 0s 519us/step - loss: 0.2257 - accuracy: 0.9688 - val_loss: 0.4205 - val_accuracy: 0.8
750
traintime- 0.10325285991032919
```

Figure 7.3 Training Results For CNN+LSTM

7.2.1 PLOTTING GRAPHS

Accuracy vs epochs and loss vs epochs for both train and test splits shows that the model vas able to learn properly.

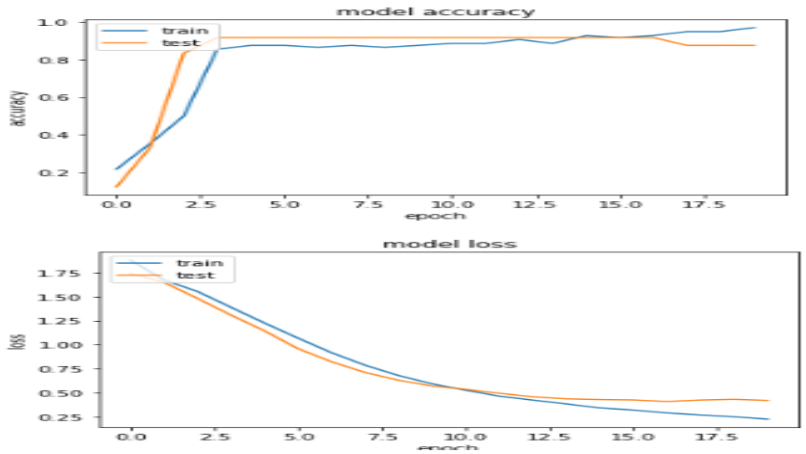


Figure 7.4 Graphs for CNN+LSTM

8. CONCLUSION AND FUTURE SCOPE

By Using the Differential Key Evolution algorithm that can generate key-frames(best possible) we were able to eliminate the manual process of extracting the key frames, however we must still observe these generated key-frames to ensure that the data we feed into the neural network is noise free and precise.

Another thing that was noticed is that the validation losses for both the hybrid models Is noticeable and must be reduced this maybe due the less number of sequences present in the validation set but this can also be improved in the future work.

We noticed that the Shorter Sequences are easily understood by the model, hence we want to improve the work by extending the model to be able to understand Multi-person Interactions(more than 2).

9. REFERENCES

Websites:

Kaggle : https://www.kaggle.com/

Medium : https://medium.com/

Documentations:

Keras : https://keras.io/

Tensorflow : https://www.tensorflow.org/versions/r2.0/api_docs/python/tf

Opencv : https://docs.opencv.org/master/

Papers:

Appendix A: Differential Key Evolution

Abraham, Kevin & Ashwin, M. & Sundar, Darshak & Ashoor, Tharic & Jeyakumar, Gurusamy. (2017). An evolutionary computing approach for solving key frame extraction problem in video analytics. 1615-1619. 10.1109/ICCSP.2017.8286663.

Appendix B:Human Interaction Recognition based on Deep Learning and HMM

A. Gong, C. Chen and M. Peng, "Human Interaction Recognition Based on Deep Learning and HMM," in IEEE Access, vol. 7, pp. 161123-161130, 2019.