Human Activity Recognition

Big-Data Systems and Intelligence Analytics

INFO 7245 - SPRING 2018

PROFESSOR:
Prof. Nicholas Brown

TEAM MEMBERS:

TEJESH RATHI (https://linkedin.com/in/tejesh-rathi)
SAMEER SUMAN (https://www.linkedin.com/in/sameersuman)

Table of Contents

Abstract:

The purpose of this project is to create a model that can identify the basic human actions like running, jogging, walking, clapping, hand-waving and boxing. Local space-time features capture local events in video and can be adapted to the size, the frequency and the velocity of moving patterns. In this paper we demonstrate how such features can be used for recognizing complex motion patterns. We construct video representations in terms of local space-time features and integrate such representations with Convolution Neural Networks for recognition. For the purpose of evaluation, we introduce a new video database containing 598 video sequences of six human actions performed by 25 people in four different scenarios. The presented results of action recognition justify the proposed method and demonstrate its advantage compared to other relative approaches for action recognition. To test the approach, we worked with 598 video sequences downloaded from Swedish University NADA Research ([1]). Our framework accurately classified the human action 67% of the times and classified the data into those 6 categories.

There are potentially a lot of applications of video recognition such as: Real-time tracking of an object - This could be very helpful for tracking the location of an object (like a vehicle) or a person from the video recorded by a CCTV. Learning the patterns involved in the movement of humans - If we can create a model that can learn how we (humans) perform various activities (like walking, running, exercising etc.), we can use this model for proper functioning of the movement mechanisms in autonomous robots.

Introduction:

Computers are getting better at solving some very complex problems (like understanding an image) due to the advances in computer vision. Models are being made wherein, if an image is given to the model, it can predict what the image is about, or it can detect whether an object is present in the image or not. These models are known as neural networks (or artificial neural networks) which are inspired by the structure and functionality of a human brain. Applications such as surveillance, video retrieval and human-computer interaction require methods for recognizing human actions in various scenarios. Typical scenarios include scenes with cluttered, moving backgrounds, nonstationary camera, scale variations, individual variations in appearance and cloth of people, changes in light and view point and so forth. All of these conditions introduce challenging problems. Deep learning, a subfield of Machine learning is the study of these neural networks and over the time, many variations of these networks have been implemented for a variety of different problems. This project aims at implementing video recognition application using the convolution neural network. In this paper we demonstrate that action recognition can be achieved using local measurements in terms of spatiotemporal interest points (local features) [9]. Such features capture local motion events in video and can be adapted to the size, the frequency and the velocity of moving patterns, hence, resulting in video representations that are stable with respect to corresponding transformations. This project uses deep learning for Video Recognition - given a set of labeled videos, train a model so that it can give a label/prediction for a new video. Here, the label might represent what is being performed in the video, or what the video is about. Once the model has been trained on the training data, its performance will be evaluated using the test data.

Code with Documentation:

Data Exploration:

The dataset can be obtained here – Recognition of Human Actions

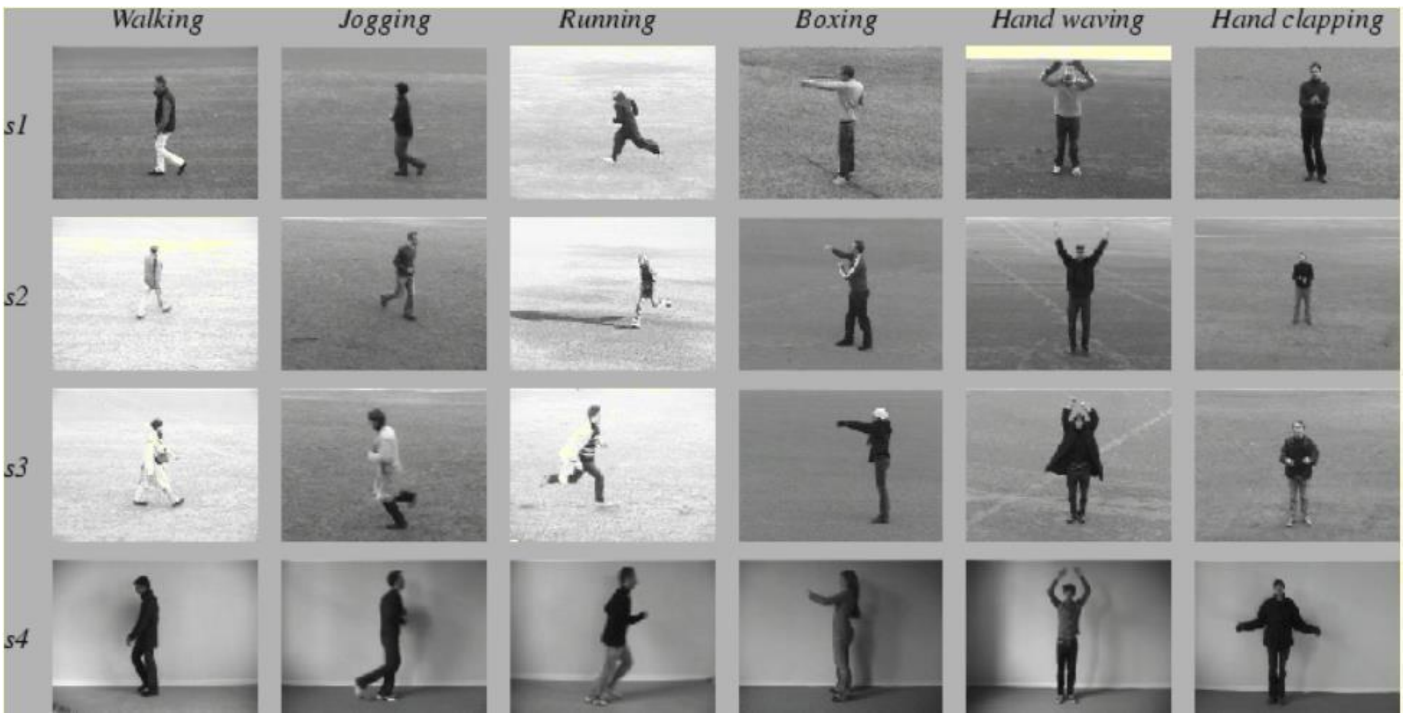
→ The video dataset contains six types of human actions (boxing, handclapping, handwaving, jogging, running and walking) performed several times by 25 different subjects in 4 different scenarios - outdoors *s1*, outdoors with scale variation *s2*, outdoors with different clothes *s3* and indoors

s4. The model will be constructed irrespective of these scenarios.

→ The videos were captures at a frame rate of 25fps and each frame was down-sampled to the resolution of 160x120 pixels.

→ The dataset contains 598 videos – 100 videos for each of the 6 categories (with the exception of Handclapping and Boxing having 99 videos)

Next, there are some sample frames for some videos from the dataset.

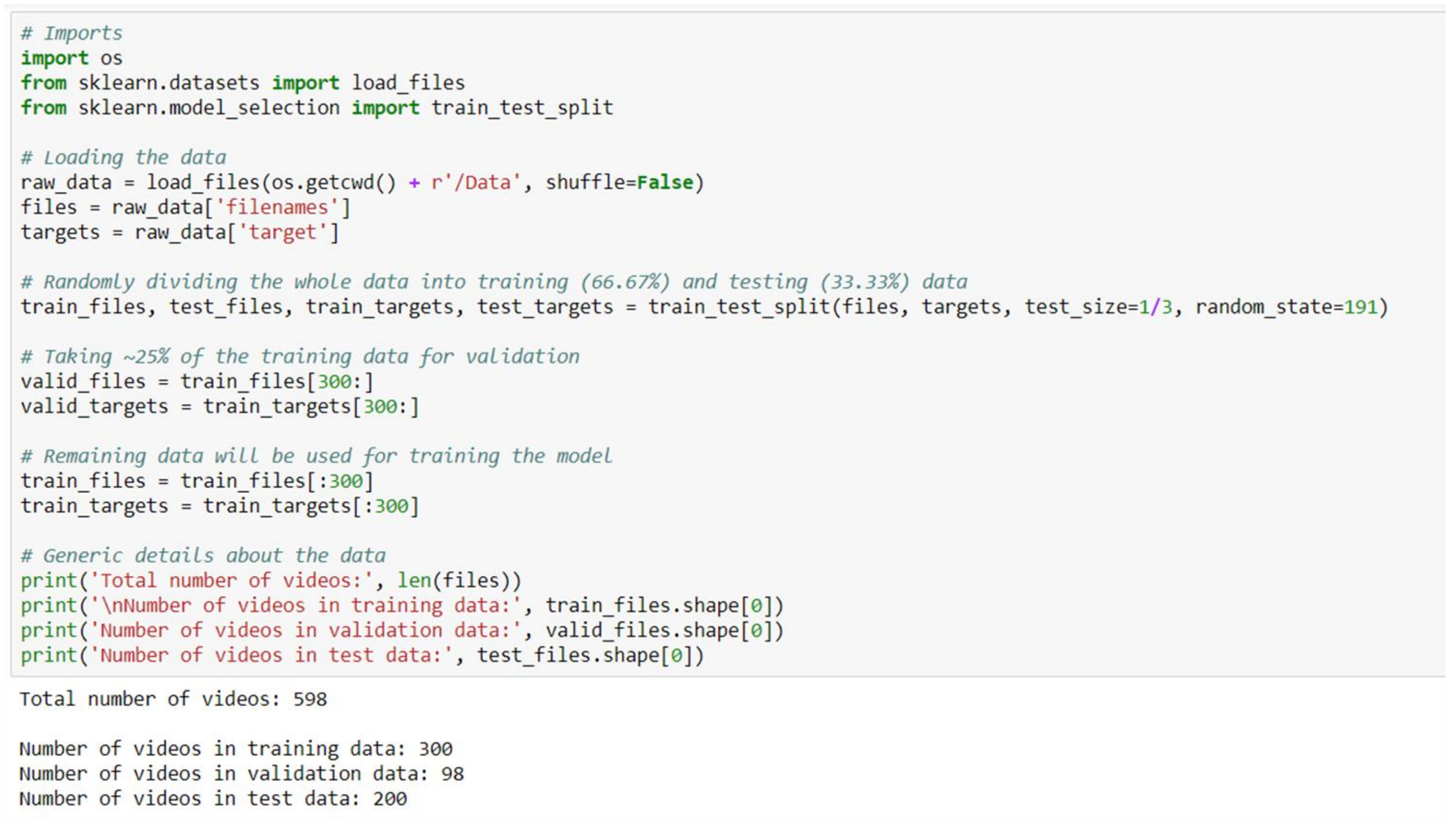


There is a total of 6 categories - boxing, handclapping, handwaving, jogging, running and walking. While loading the data, we convert these text labels into integers according to the following mapping:

Boxing	0
Handclapping	1
Handwaving	2
Jogging	3
Running	4
Walking	5

Further instructions to obtain the data are mentioned here - Link

Train, Test and Validation data distribution Code:

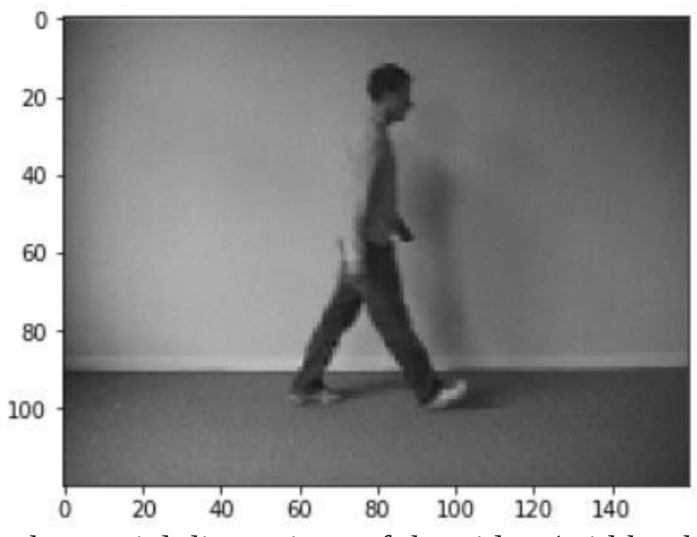


```python
# Imports
import os
from sklearn.datasets import load_files
from sklearn.model_selection import train_test_split

# Loading the data
raw_data = load_files(os.getcwd() + r'/Data', shuffle=False)
files = raw_data['filenames']
targets = raw_data['target']

# Randomly dividing the whole data into training (66.67%) and testing (33.33%) data
train_files, test_files, train_targets, test_targets = train_test_split(files, targets, test_size=1/3, random_state=191)

# Taking ~25% of the training data for validation
valid_files = train_files[300:]
valid_targets = train_targets[300:]

# Remaining data will be used for training the model
train_files = train_files[:300]
train_targets = train_targets[:300]

# Generic details about the data
print('Total number of videos:', len(files))
print('\nNumber of videos in training data:', train_files.shape[0])
print('Number of videos in validation data:', valid_files.shape[0])
print('Number of videos in test data:', test_files.shape[0])
```

```
Total number of videos: 598

Number of videos in training data: 300
Number of videos in validation data: 98
Number of videos in test data: 200
```

Exploratory Visualization:

Below is a single frame of a sample video of walking



It can be observed that the spatial dimensions of the video (width x height) are 160 x 120 pixels. Also, on loading a single video into a NumPy array in python, the shape of the array obtained was – (1, 515, 120, 160, 3)

This indicates that:

There is 1 video

The video has 515 frames

The spatial dimension of the video is 160 x 120 (width x height) pixels

Each frame has 3 channels – Red(R), Green(G) and Blue(B)

A similar methodology would be used for reading in the entire dataset.

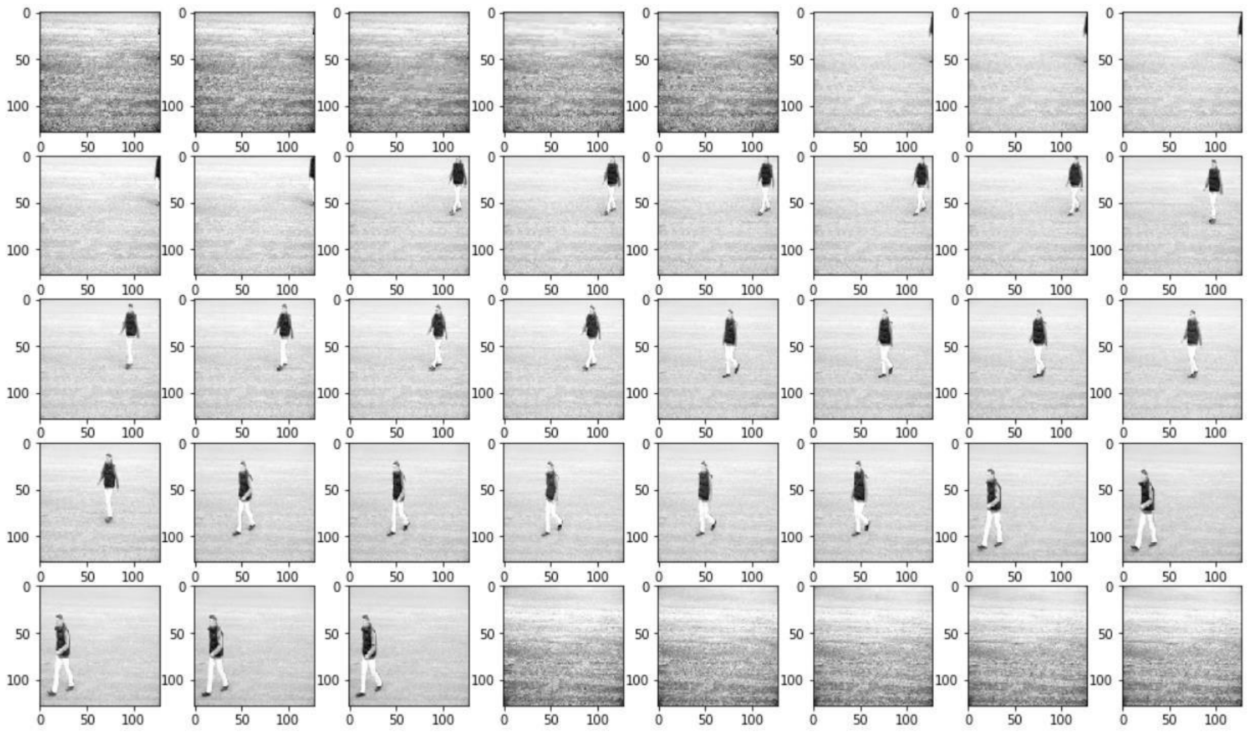


Fig. The frames of a sample video of 'Walking'

The above visual shows that in the videos of activities involving the movement of the whole body, a lot of frames in the video might be empty (no person performing any action). Also, if the person is moving very slowly, then most of the frames would be redundant.

It would be a major challenge for the model to deal with such a problem.

Data pre-processing

- Reading in the video frame-by-frame.
- The videos were captured at a frame rate of 25fps. This means that for each second of the video, there will be 25 frames. We know that within a second, a human body does not perform very significant movement. This implies that most of the frames (per second) in our video will be redundant. Therefore, only a subset of all the frames in a video needs

to be extracted. This will also reduce the size of the input data which will in turn help the model train faster and can also prevent over-fitting.

- Different strategies would be used for frame extraction like:
- Extracting a fixed number of frames from the total frames in the video – say only the first 200 frames (i.e., first 8 seconds of the video).
- Extracting a fixed number of frames each second from the video – say we need only 5 frames per second from a video whose duration is of 10 seconds. This would return a total of 50 frames from the video. This approach is better in the sense that we are extracting the frames sparsely and uniformly from the entire video.
- Each frame needs to have the same spatial dimensions (height and width). Hence each frame in a video will have to be resized to the required size.
- In order to simplify the computations, the frames are converted to grayscale.
- Normalization – The pixel values ranges from 0 to 255. These values would have to be normalized in order to help our model converge faster and get a better performance. Different normalization techniques can be applied such as:
    - Min-max Normalization – Get the values of the pixels in a given range (say 0 to 1)
    - Z-score Normalization – This basically determines the number of standard deviations from the mean a data point is.
- We would finally get a 5-dimensional tensor of shape –

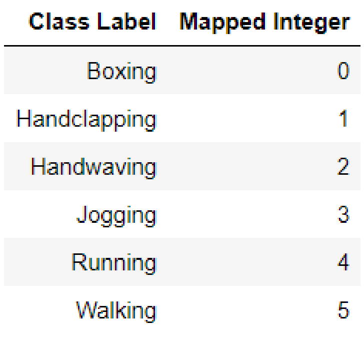
  (<number of videos>, <number of frames>, <width>, <height>, <channels>)

  ‘channels’ can have the value 1 (grayscale) or 3 (RGB)

  ‘number of frames’ - the extracted frames (will have to be the same for each video)

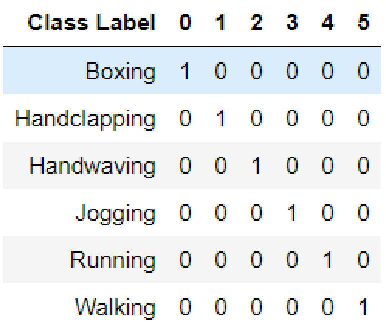
  Also, the categorical labels should be encoded using a technique called One-hot Encoding.

  One-hot Encoding converts the categorical labels into a format that works better with both classification and regression models.



Class Label	Mapped Integer
Boxing	0
Handclapping	1
Handwaving	2
Jogging	3
Running	4
Walking	5

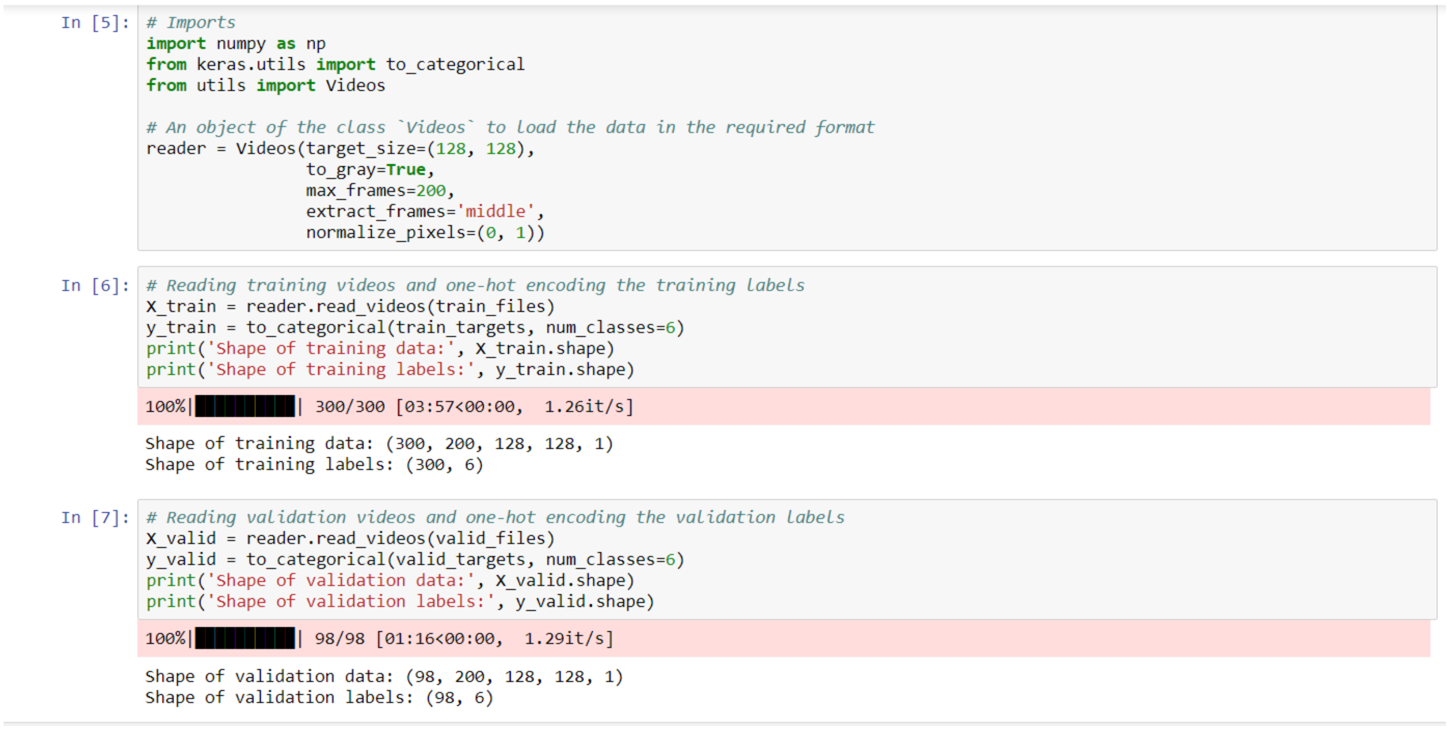
Before One-hot Encoding



Class Label	0	1	2	3	4	5
Boxing	1	0	0	0	0	0
Handclapping	0	1	0	0	0	0
Handwaving	0	0	1	0	0	0
Jogging	0	0	0	1	0	0
Running	0	0	0	0	1	0
Walking	0	0	0	0	0	1

After One-hot Encoding

Data pre-processing Code:



```
In [5]: # Imports
        import numpy as np
        from keras.utils import to_categorical
        from utils import Videos

        # An object of the class `Videos` to load the data in the required format
        reader = Videos(target_size=(128, 128),
                        to_gray=True,
                        max_frames=200,
                        extract_frames='middle',
                        normalize_pixels=(0, 1))
```

```
In [6]: # Reading training videos and one-hot encoding the training labels
        X_train = reader.read_videos(train_files)
        y_train = to_categorical(train_targets, num_classes=6)
        print('Shape of training data:', X_train.shape)
        print('Shape of training labels:', y_train.shape)

        100%|██████████| 300/300 [03:57<00:00,  1.26it/s]

        Shape of training data: (300, 200, 128, 128, 1)
        Shape of training labels: (300, 6)
```

```
In [7]: # Reading validation videos and one-hot encoding the validation labels
        X_valid = reader.read_videos(valid_files)
        y_valid = to_categorical(valid_targets, num_classes=6)
        print('Shape of validation data:', X_valid.shape)
        print('Shape of validation labels:', y_valid.shape)

        100%|██████████| 98/98 [01:16<00:00,  1.29it/s]

        Shape of validation data: (98, 200, 128, 128, 1)
        Shape of validation labels: (98, 6)
```

```python
# Reading training videos and one-hot encoding the training labels
X_train = reader.read_videos(train_files)
y_train = to_categorical(train_targets, num_classes=6)
print('Shape of training data:', X_train.shape)
print('Shape of training labels:', y_train.shape)
```

```
100%|██████████| 300/300 [03:57<00:00,  1.26it/s]
```

```
Shape of training data: (300, 200, 128, 128, 1)
Shape of training labels: (300, 6)
```

```python
# Reading validation videos and one-hot encoding the validation labels
X_valid = reader.read_videos(valid_files)
y_valid = to_categorical(valid_targets, num_classes=6)
print('Shape of validation data:', X_valid.shape)
print('Shape of validation labels:', y_valid.shape)
```

```
100%|██████████| 98/98 [01:16<00:00,  1.29it/s]
```

```
Shape of validation data: (98, 200, 128, 128, 1)
Shape of validation labels: (98, 6)
```

```python
# Reading testing videos and one-hot encoding the testing labels
X_test = reader.read_videos(test_files)
y_test = to_categorical(test_targets, num_classes=6)
print('Shape of testing data:', X_test.shape)
print('Shape of testing labels:', y_test.shape)
```

```
100%|██████████| 200/200 [02:39<00:00,  1.26it/s]
```

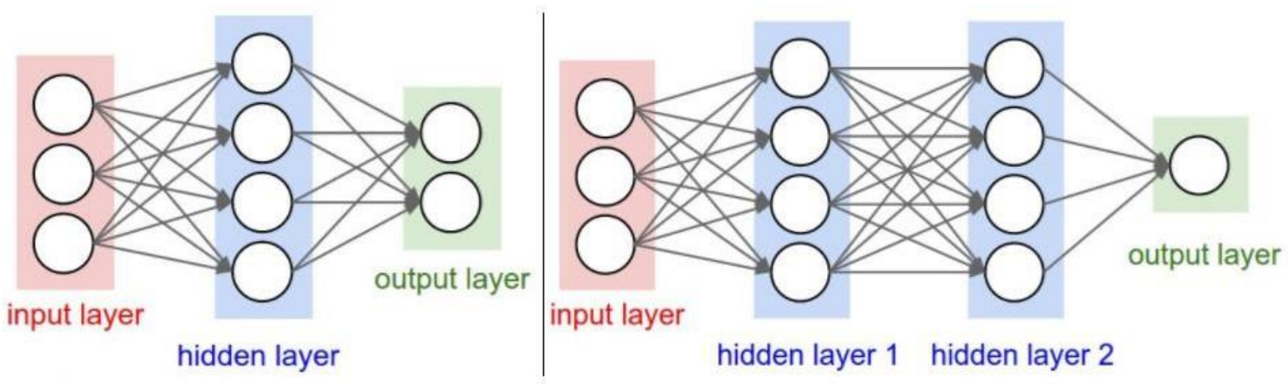
```
Shape of testing data: (200, 200, 128, 128, 1)
Shape of testing labels: (200, 6)
```

Algorithms and Techniques:

We already know that neural networks perform very well for image recognition. In particular, a specific type of neural networks called Convolutional Neural Networks (CNNs) are best suited for the task of image recognition. I will now explain how the approach of convolutional neural networks differ from that of normal neural networks.

Traditional Neural Networks

The image is flattened into a 1-dimensional array, and this array is given as the input to our neural network. The problem with this approach is that the spatial pattern of the pixels (their position in their 2-d form) is not taken into account. Also, suppose we have an image whose dimension is 256 x 256 pixels. The input vector will then comprise of 65,536 nodes, one for each pixel in the image. That's a very large input vector, which could make the weight matrix very large and in turn, make the model very computationally intensive. And even after being so complex, the network would not be able to give any significant accuracy. As a result, this approach was not suited well for tasks like image recognition.

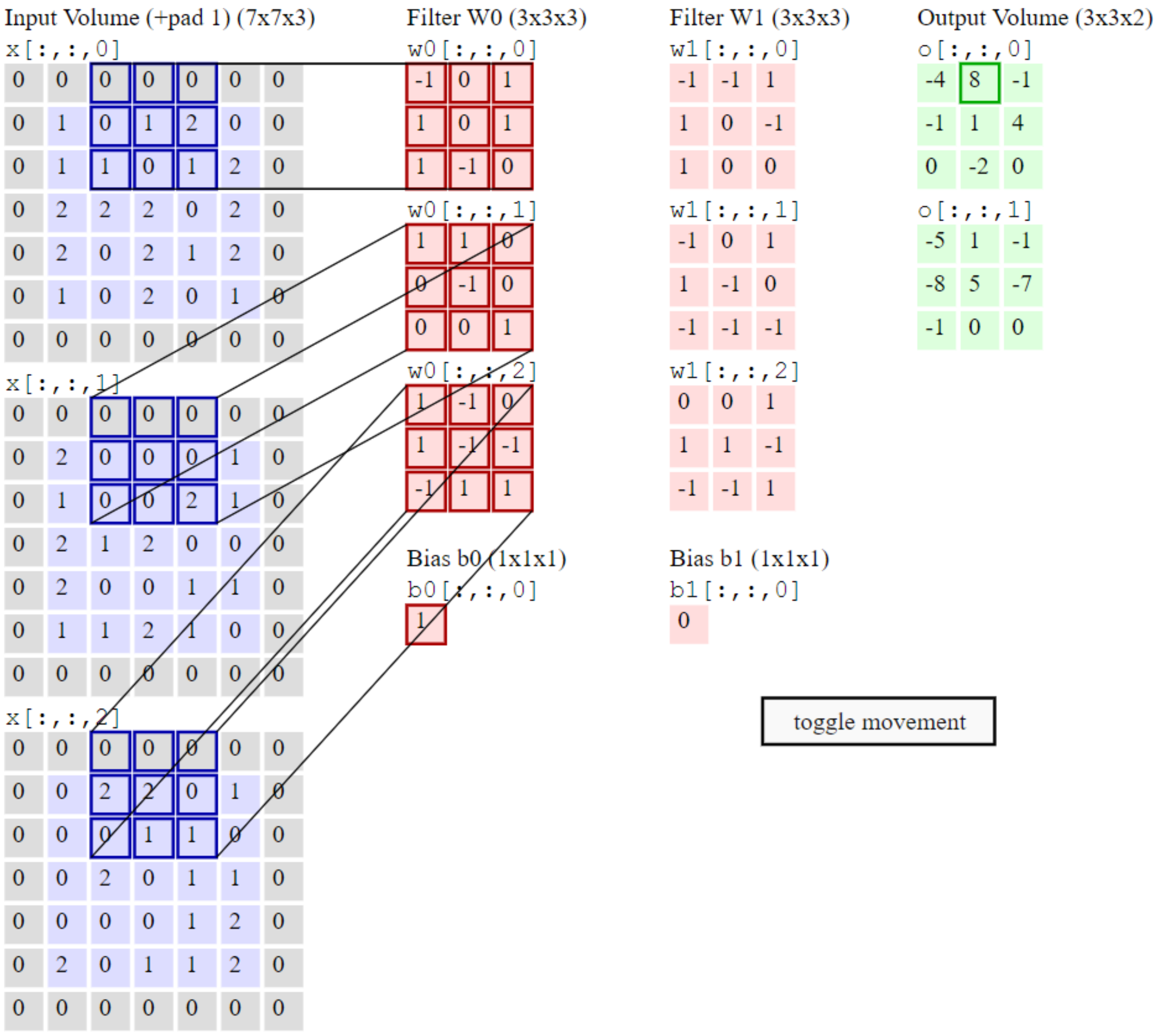


Left: A 2-layer Neural Network (one hidden layer of 4 neurons (or units) and one output layer with 2 neurons), and three inputs. **Right:** A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer.

Convolutional Neural Networks

The image is divided into regions, and each region is then assigned to different hidden nodes. Each hidden node finds pattern in only one of the regions in the image. This region is determined by a kernel (also called a filter/window). A filter is convolved over both x-axis and y-axis. Multiple filters are used in order to extract different patterns from the image. The output of one filter when convolved throughout the entire image generates a 2-d layer of neurons called a feature map. Each filter is responsible for one features map.

These feature maps can be stacked into a 3-d array, which can then be used as the input to the layers further. This is performed by the layer known as Convolutional layer in a CNN. These layers are followed by the Pooling layers, that reduce the spatial dimensions of the output (obtained from the convolutional layers). In short, a window is slid in both the axes and the max value in that filter/window is taken (Max- Pooling layer). Sometimes Average pooling layer is also used where the only difference is to take the average value within the window instead of the maximum value. Therefore, the convolutional layers increase the depth of the input image, whereas the pooling layers decreases the spatial dimensions (height and width). The importance of such an architecture is that it encodes the content of an image that can be flattened into a 1-dimensional array.



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	0	1	2	0	0
0	1	1	0	1	2	0
0	2	2	2	0	2	0
0	2	0	2	1	2	0
0	1	0	2	0	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	0	0	0	1	0
0	1	0	0	2	1	0
0	2	1	2	0	0	0
0	2	0	0	1	1	0
0	1	1	2	1	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	0	2	2	0	1	0
0	0	0	1	1	0	0
0	0	2	0	1	1	0
0	0	0	0	1	2	0
0	2	0	1	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
1	0	1
1	-1	0

w0[:,:,1]

1	1	0
0	-1	0
0	0	1

w0[:,:,2]

1	-1	0
1	-1	-1
-1	1	1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	-1	1
1	0	-1
1	0	0

w1[:,:,1]

-1	0	1
1	-1	0
-1	-1	-1

w1[:,:,2]

0	0	1
1	1	-1
-1	-1	1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

-4	8	-1
-1	1	4
0	-2	0

o[:,:,1]

-5	1	-1
-8	5	-7
-1	0	0

We discussed how CNNs can be used in case of images. What we use is specifically known as 2-d convolutional layers and pooling layers. It’s 2-dimensional because the filter is convolved along the x-axis and y-axis of the image. But in case of a video, we have an additional temporal axis – z-axis. So, a 3-d convolutional layer is used – where the filter (also 3-dimensional) is convolved across all the three axes.

Multiple convolutional and pooling layers are stacked together. These are followed by some fully-connected layers, where the last layer is the output layer. The output layer contains 6 neurons (one for each category). The network gives a probability of an input to belong to each category/class.

A brief procedure:

- The entire dataset is divided into 3 parts – training data, validation data and test data.
- The model is trained on the training data repeatedly for a number of iterations. These iterations are known as epochs. After each epoch, the model is tested using the validation data.
- Finally, the model that performed the best on the validation data is loaded.
- The performance of this model is then evaluated using the test data.

Model Parameters

For each convolutional layer, we have to configure the following parameters:

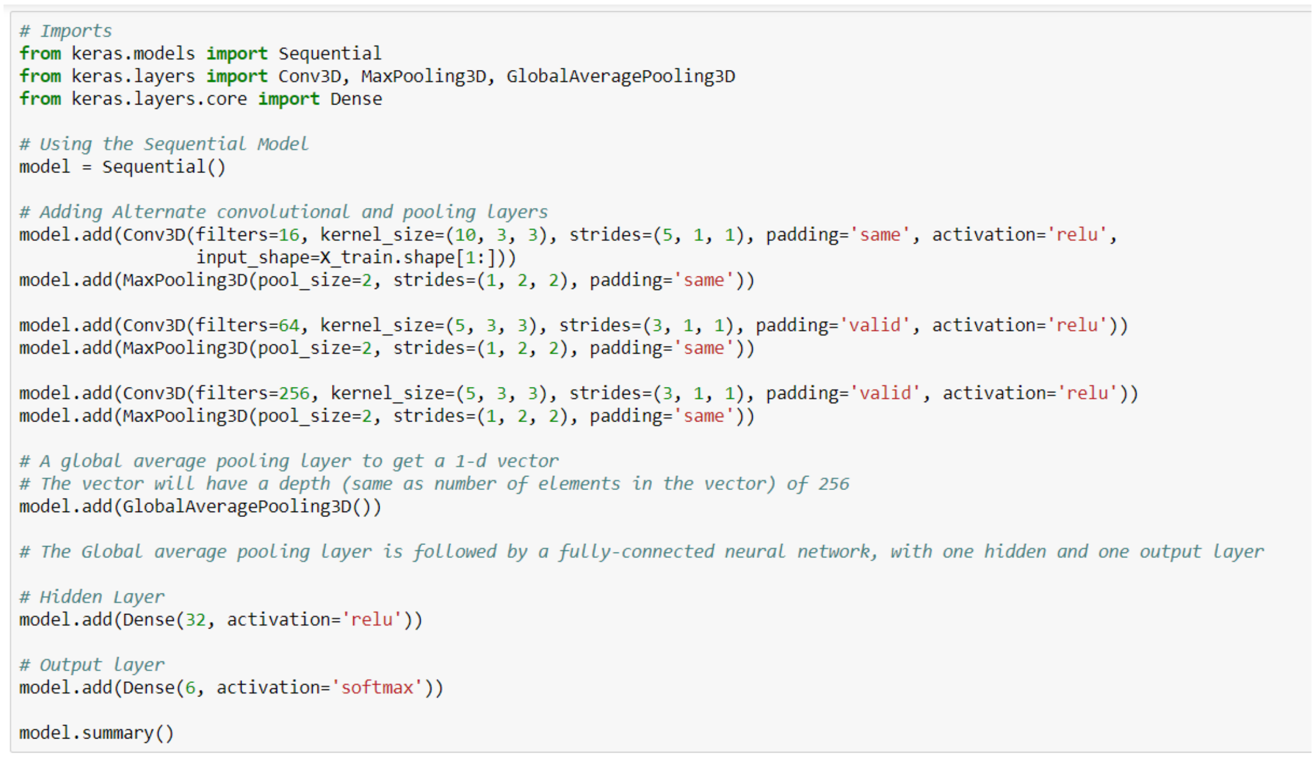
- filters - This is the number of feature maps required as the output of that convolutional layer
- kernel_size - The size of the window that will get convolved along all the axes of the input data to produce a single feature map
- strides - The number of pixels by which the convolutional window should shift by
- padding - To decide what happens on the edges - either the input gets cropped (valid) or the input is padded with zeros to maintain the same dimensionality (same)
- activation - The activation function to be used for that layer. (ReLU is proven to work best with deep neural networks because of its non-linearity, and it property of avoiding the vanishing gradient problem)

For each pooling layer, we have to configure the following parameters:

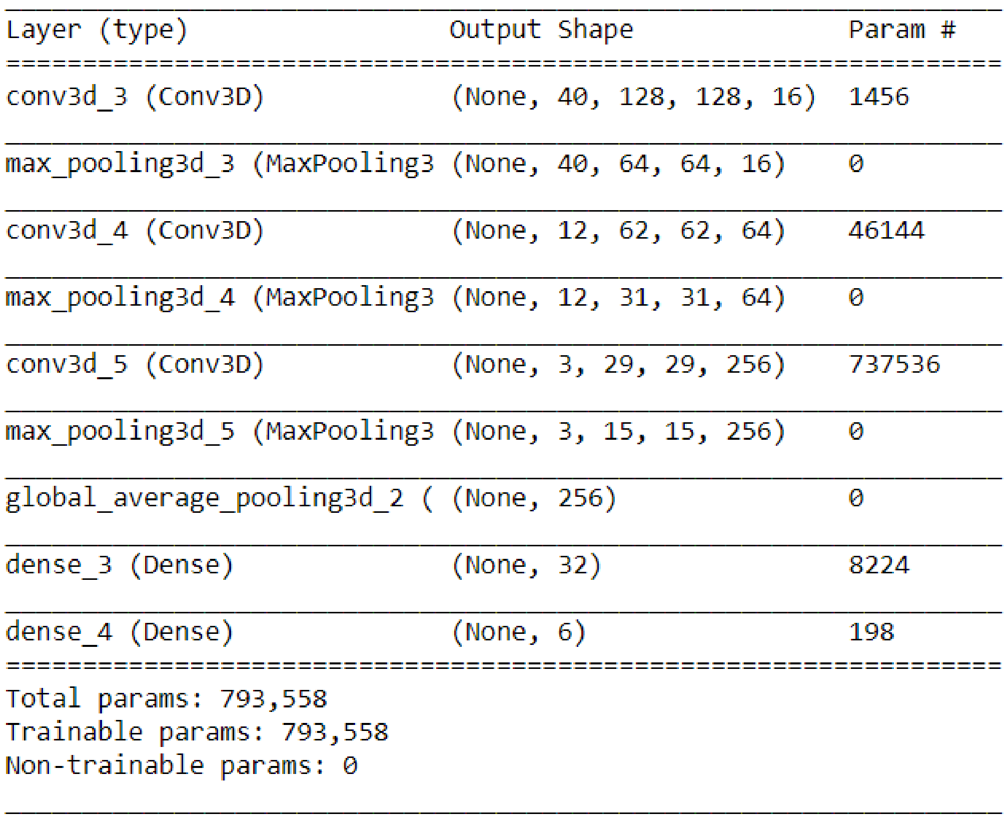
- pool_size - The size of the window.
- strides - The number of pixels by which the pooling window should shift by.

- padding - To decide what happens on the edges - either the input gets cropped (valid) or the input is padded with zeros to maintain the same dimensionality (same).

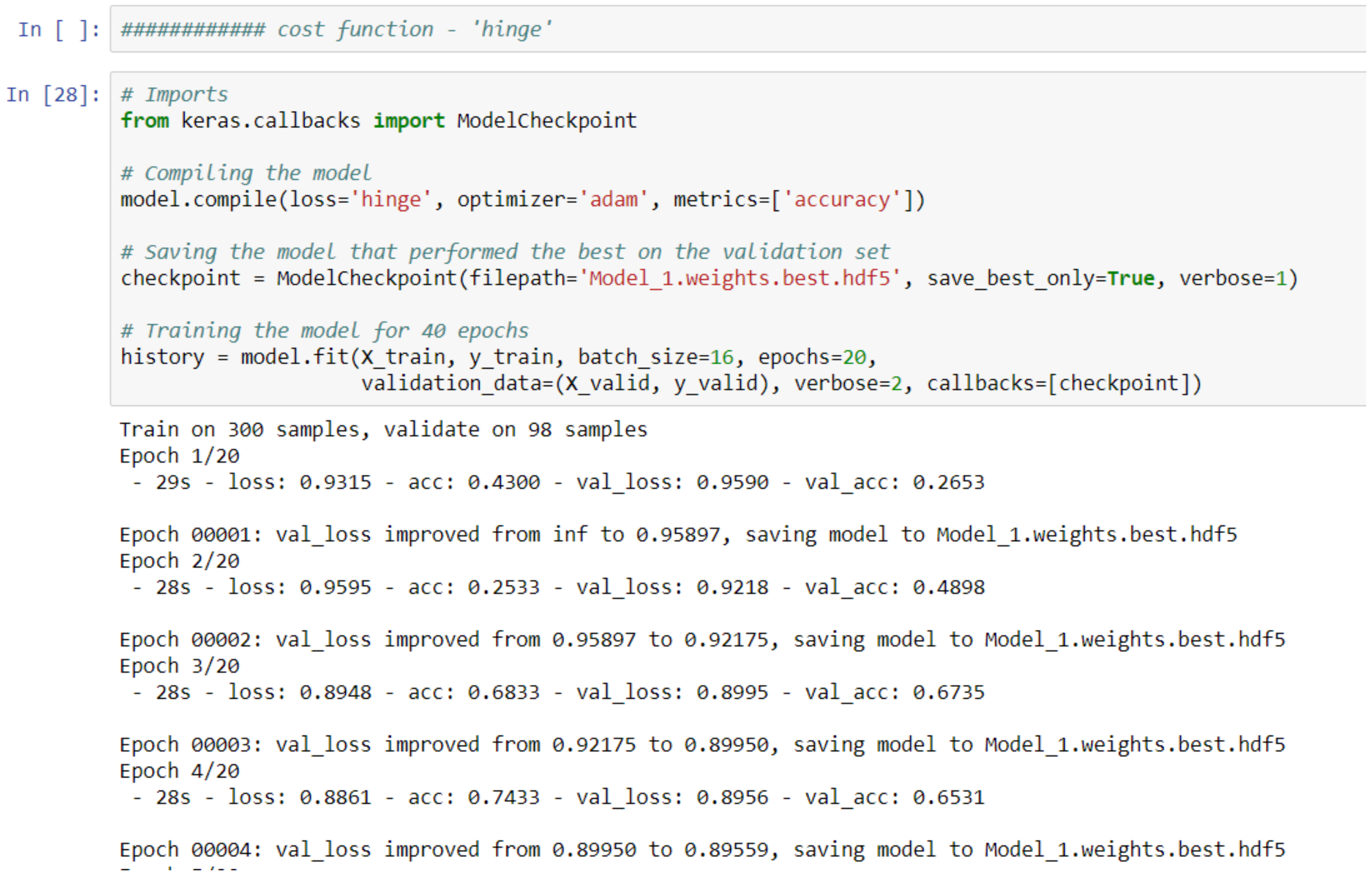
Model Implementation Code:



```
# Imports
from keras.models import Sequential
from keras.layers import Conv3D, MaxPooling3D, GlobalAveragePooling3D
from keras.layers.core import Dense

# Using the Sequential Model
model = Sequential()

# Adding Alternate convolutional and pooling layers
model.add(Conv3D(filters=16, kernel_size=(10, 3, 3), strides=(5, 1, 1), padding='same', activation='relu',
                 input_shape=X_train.shape[1:]))
model.add(MaxPooling3D(pool_size=2, strides=(1, 2, 2), padding='same'))

model.add(Conv3D(filters=64, kernel_size=(5, 3, 3), strides=(3, 1, 1), padding='valid', activation='relu'))
model.add(MaxPooling3D(pool_size=2, strides=(1, 2, 2), padding='same'))

model.add(Conv3D(filters=256, kernel_size=(5, 3, 3), strides=(3, 1, 1), padding='valid', activation='relu'))
model.add(MaxPooling3D(pool_size=2, strides=(1, 2, 2), padding='same'))

# A global average pooling layer to get a 1-d vector
# The vector will have a depth (same as number of elements in the vector) of 256
model.add(GlobalAveragePooling3D())

# The Global average pooling layer is followed by a fully-connected neural network, with one hidden and one output layer

# Hidden Layer
model.add(Dense(32, activation='relu'))

# Output layer
model.add(Dense(6, activation='softmax'))

model.summary()
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv3d_3 (Conv3D)            (None, 40, 128, 128, 16)  1456      
_________________________________________________________________
max_pooling3d_3 (MaxPooling3 (None, 40, 64, 64, 16)    0         
_________________________________________________________________
conv3d_4 (Conv3D)            (None, 12, 62, 62, 64)    46144     
_________________________________________________________________
max_pooling3d_4 (MaxPooling3 (None, 12, 31, 31, 64)    0         
_________________________________________________________________
conv3d_5 (Conv3D)            (None, 3, 29, 29, 256)    737536    
_________________________________________________________________
max_pooling3d_5 (MaxPooling3 (None, 3, 15, 15, 256)    0         
_________________________________________________________________
global_average_pooling3d_2 ( (None, 256)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 32)                8224      
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 198       
=================================================================
Total params: 793,558
Trainable params: 793,558
Non-trainable params: 0
_________________________________________________________________
```



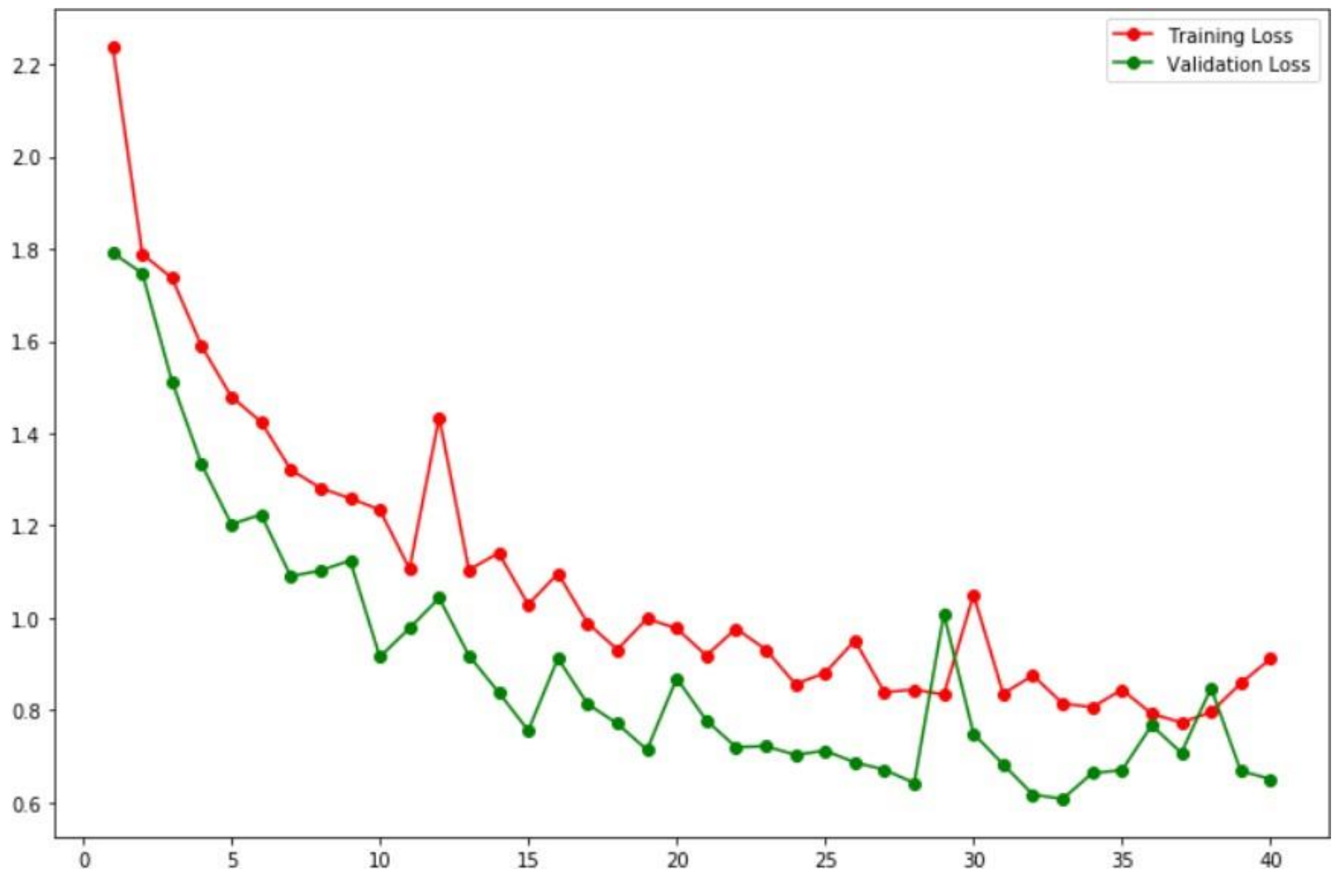
```
In [ ]: ############ cost function - 'hinge'
```

```
In [28]: # Imports
         from keras.callbacks import ModelCheckpoint

         # Compiling the model
         model.compile(loss='hinge', optimizer='adam', metrics=['accuracy'])

         # Saving the model that performed the best on the validation set
         checkpoint = ModelCheckpoint(filepath='Model_1.weights.best.hdf5', save_best_only=True, verbose=1)

         # Training the model for 40 epochs
         history = model.fit(X_train, y_train, batch_size=16, epochs=20,
                             validation_data=(X_valid, y_valid), verbose=2, callbacks=[checkpoint])
```

```
Train on 300 samples, validate on 98 samples
Epoch 1/20
 - 29s - loss: 0.9315 - acc: 0.4300 - val_loss: 0.9590 - val_acc: 0.2653

Epoch 00001: val_loss improved from inf to 0.95897, saving model to Model_1.weights.best.hdf5
Epoch 2/20
 - 28s - loss: 0.9595 - acc: 0.2533 - val_loss: 0.9218 - val_acc: 0.4898

Epoch 00002: val_loss improved from 0.95897 to 0.92175, saving model to Model_1.weights.best.hdf5
Epoch 3/20
 - 28s - loss: 0.8948 - acc: 0.6833 - val_loss: 0.8995 - val_acc: 0.6735

Epoch 00003: val_loss improved from 0.92175 to 0.89950, saving model to Model_1.weights.best.hdf5
Epoch 4/20
 - 28s - loss: 0.8861 - acc: 0.7433 - val_loss: 0.8956 - val_acc: 0.6531

Epoch 00004: val_loss improved from 0.89950 to 0.89559, saving model to Model_1.weights.best.hdf5
```

Results:

The last model (Model-3) gave the highest accuracy on the test data (70.5%). Given below is the learning curve of the model over 40 epochs.



We chose the model weights that performed the best on the validation set, which gave us the highest accuracy on the test data.

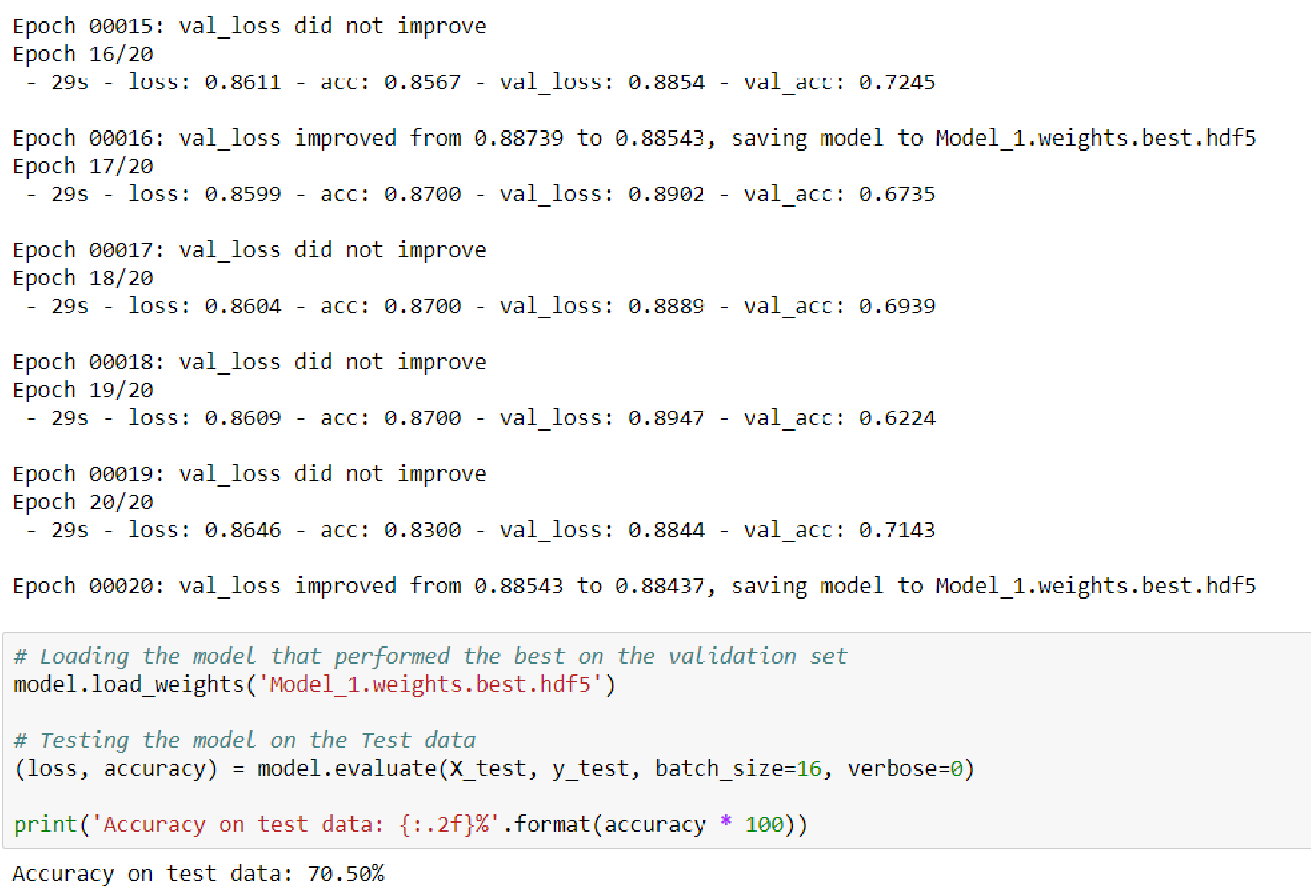
Since the input data had been reduced while pre-processing (by 5 times), the previous models would have taken considerably less time for training. It's because of this reduction in the data, that a more complex model was constructed and used. If we had the same data, this model would have taken a very long time to train (the model had approx. 5 million trainable parameters). Hence, making the model deeper (more complex) and compensating the increase in training time by reduction of data is how the model gave the highest accuracy.

Following are some of the important specifications of the final model:

- The depth of the vector obtained by the last convolutional layer is 1024.
- A Global Average Pooling layer (GAP) then takes the average value from each of these 1024 dimensions and gives a 1-dimensional vector representing the entire video.

- The GAP is followed by a fully-connected layer containing 32 neurons. This fully-connected layer also has a dropout of 0.5, meaning that for each epoch, 50% of the neurons of this layer will be deactivated. This is what helps the model prevent overfitting.
- Finally, there is the output layer with 6 neurons (one for each category). The network gives a probability for the input video to belong to each of the 6 categories.
- All the convolutional layers have 'ReLU' as the activation function. It gives the best performance out of a CNN.

Accuracy Code:



```
Epoch 00015: val_loss did not improve
Epoch 16/20
 - 29s - loss: 0.8611 - acc: 0.8567 - val_loss: 0.8854 - val_acc: 0.7245

Epoch 00016: val_loss improved from 0.88739 to 0.88543, saving model to Model_1.weights.best.hdf5
Epoch 17/20
 - 29s - loss: 0.8599 - acc: 0.8700 - val_loss: 0.8902 - val_acc: 0.6735

Epoch 00017: val_loss did not improve
Epoch 18/20
 - 29s - loss: 0.8604 - acc: 0.8700 - val_loss: 0.8889 - val_acc: 0.6939

Epoch 00018: val_loss did not improve
Epoch 19/20
 - 29s - loss: 0.8609 - acc: 0.8700 - val_loss: 0.8947 - val_acc: 0.6224

Epoch 00019: val_loss did not improve
Epoch 20/20
 - 29s - loss: 0.8646 - acc: 0.8300 - val_loss: 0.8844 - val_acc: 0.7143

Epoch 00020: val_loss improved from 0.88543 to 0.88437, saving model to Model_1.weights.best.hdf5
```

```python
# Loading the model that performed the best on the validation set
model.load_weights('Model_1.weights.best.hdf5')

# Testing the model on the Test data
(loss, accuracy) = model.evaluate(X_test, y_test, batch_size=16, verbose=0)

print('Accuracy on test data: {:.2f}%'.format(accuracy * 100))
```

```
Accuracy on test data: 70.50%
```

Discussion or Future Prospects:

- There is a huge scope of improvement in the approach that has been used in order to address the problem.
    - The data could be processed more efficiently. As stated earlier, the pre-processing step should take care of the frames that are empty (with no human performing any action). This could significantly improve the performance of the model by reducing the false positives rate.
    - The proposed model, as seen in the learning curve, overfitted to the training data after certain number of epochs. This suggests that there is a lot of tuning that can be on the model in order to prevent overfitting, and hence improving the results.

- Also, there is one potential model that can give a much better performance than our current model. The part where our model is lagging is that it is not able to extract features from the video and convert it into a 1-d vector, without losing much information. If we are able to use the concept of transfer learning in order to extract featured from the videos, it would give much better results – given that the model used is pre-trained on some similar dataset. But since no pre-trained models exist for video recognition, we can use the following approach –
    - Use a pre-trained model (like InceptionV3 or ResNet) to encode each frame of the video into a 1-d vector. This will give us a sequence of 1-dimensional vectors (each representing a frame).
    - We can now use a sequence-to-sequence model (like LSTM) to capture the temporal relationship between adjacent frames.
    - Since this model will extract more information from each video, the performance of such a model might be a lot better than our proposed model.

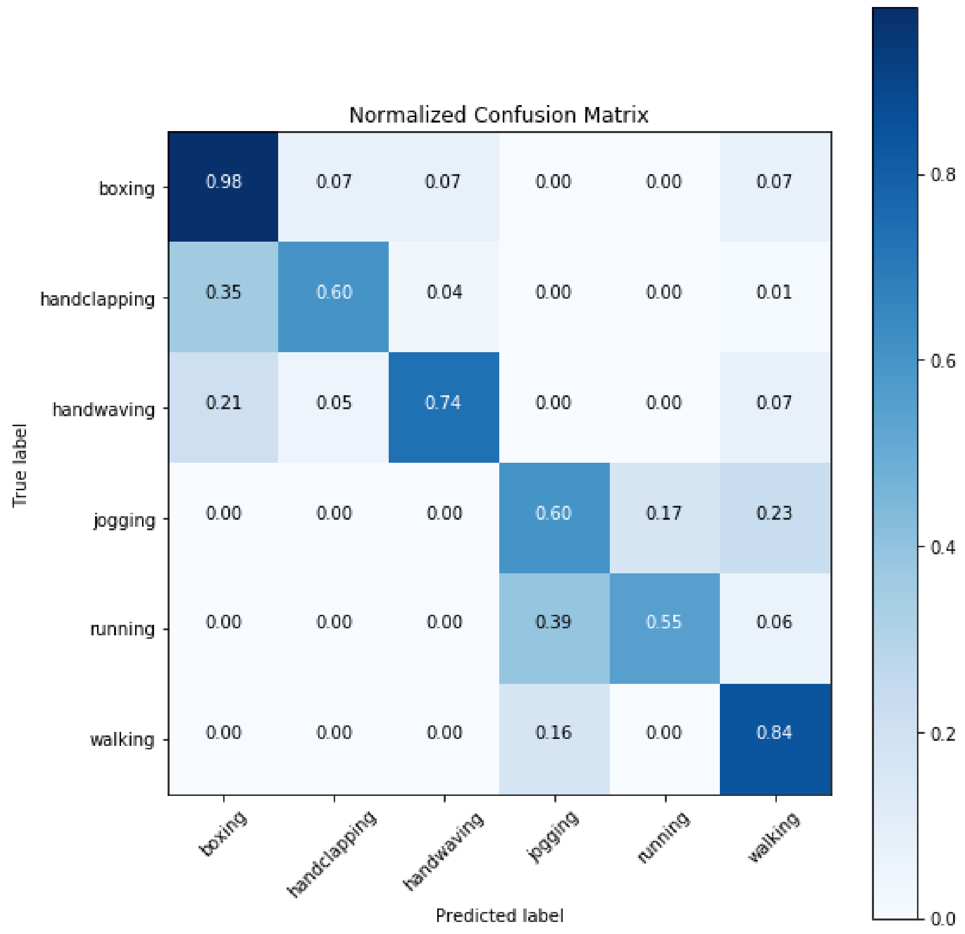
- A simple web-application could have been developed, where the user can perform some action. This would be captured by a webcam and the model would give real-time predictions of the action being performed

Conclusion:

We built a 3D CNN model and predicted the human actions in the video with an accuracy of about 64%. We compared our model with the benchmark model by running the same set of data in both the models and obtaining a confusion matrix for the same.

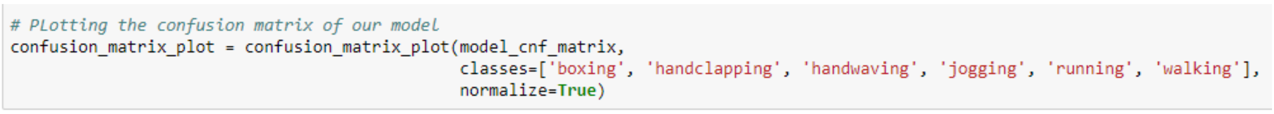
The confusion matrix of the benchmark model as well as the proposed model have been converted in the same format. Also, the confusion matrix has been normalized.

```
# Plotting the Confusion matrix of the Benchmark Model
confusion_matrix_plot(benchmark_cnf_matrix,
                      classes=['boxing', 'handclapping', 'handwaving', 'jogging', 'running', 'walking'])
```

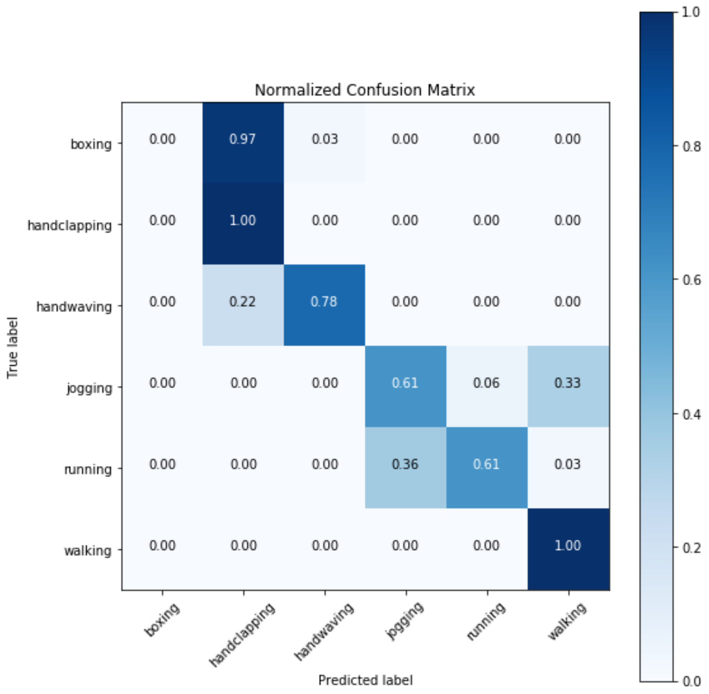


Benchmark Confusion Matrix (Normalized)

Below is the confusion matrix of our model converted in the same format.



```
# PLotting the confusion matrix of our model
confusion_matrix_plot = confusion_matrix_plot(model_cnf_matrix,
                                               classes=['boxing', 'handclapping', 'handwaving', 'jogging', 'running', 'walking'],
                                               normalize=True)
```



<u>Proposed Model Confusion Matrix (Normalized)</u>

The data used to get the results of the benchmark model was not mentioned. Although the research paper said that the test data had 9 persons. So, I randomly selected 9 different persons, processed all the videos of these 9 persons (9 x 4 = 216 videos) and constructed the confusion matrix using the proposed model.

We look at the diagonal values of the confusion matrix.

In case of 'Walking', when the actual label was 'walking' – the benchmark predicted the label as 'walking'

84% of the time, whereas our model predicted 'walking' 100% of the time.

Similar results were obtained for –

ACTION	BENCHMARK MODEL	PROPOSED MODEL
WALKING	0.84	1.00
JOGGING	0.60	0.61
HANDWAVING	0.74	0.78
HANDCLAPPING	0.60	1.00
RUNNING	0.55	0.61
BOXING	0.98	0.00

This suggests that our model was better (or at par) at predicting these actions than the benchmark model.

In fact, for 'handclapping' and 'walking' our model gave 100% accurate results.

- Our model was able to distinguish 'running' from 'jogging' better than the benchmark model. When the actual label was 'running', the benchmark model predicted 'jogging' 39% of the time, whereas the proposed model predicted 'jogging' 36% of the time
- It can be observed from the confusion matrix of both the models that the overall recognition rate of both the models is almost equal (except for 'Boxing' where our model performed extremely poor)
- If 'Boxing' is kept aside, our model has a much lower false positive rate than the benchmark model

This means that videos in which the action being performed was 'Boxing' were predicted by the benchmark model with a better recognition rate than that of our model. Our model confused 'Boxing' with 'Handclapping' 97% of the time.

To conclude, our model predicted better in recognizing 5 of the 6 human actions as compared to the benchmark model, while it failed in recognizing one and confused handclapping with boxing.

References:

[1] Recognition of Human Actions – NADA Research

[2] Recognizing Human Actions: A Local SVM Approach

[3] Deep Learning

[4] Metrics to Evaluate ML Algorithms

[5] Intro to Neural Networks

[6] HumanActivityRecognition_Smartphones

[7] Deep Learning_ActionRecognition

[8] Convolution Neural Network

[9] Confusion Matrix Terminology