Machine Learning Engineer Nanodegree

Capstone Project

Mrinal Jain
December 16th, 2017

Definition

Project Overview

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 a particular 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. Deep learning, a subfield of Machine learning is the study of these neural networks and over the time, a number of variations of these networks have been implemented for a variety of different problems.

This project uses deep learning for Video Recognition - given a set of labelled 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.

A very interesting application of this kind of problem could be identifying objects in videos in real-time. A project called Clarifai, which is an Image and Video recognition API worth to have a look at. Check out the demo here - Link

Problem Statement

The aim of this project is to create a model that can identify the basic human actions like running, jogging, walking, clapping, hand-waving and boxing. The model will be given a set of videos where in each video, a person will be performing an action. The label of a video will be the action that is being performed in that particular video. The model will have to learn this relationship, and then it should be able to predict the label of an input (video) that it has never seen. Technically, the model would have to learn to differentiate between various human actions, given some examples of these actions.

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 are able to 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.

The tasks involved are the following:

- ✓ Downloading, extracting and pre-processing a video dataset
- ✓ Dividing the dataset into training and testing data
- ✓ Create a neural network and train it on the training data
- ✓ Test the model on the test data
- ✓ Compare the performance of the model with some pre-existing models

Metrics

Once the model has been trained on the training data, its performance will be evaluated using the test data. The following metrics will be used:

Accuracy – will be used for evaluating the performance of the model on the test data.

Confusion Matrix - will be used in order to compare the model with the Benchmark model.

A *confusion matrix* is used to describe the performance of a classification model.

TOTAL	PREDICTED: YES	PREDICTED: NO
ACTUAL: YES	True Positives	False Negatives
ACTUAL: NO	False Positives	True Negatives

True Positives (TP) – The model predicted 'YES' and the prediction was *True*

False Positives (FP) – The model predicted 'YES' but the prediction was *False*

False Negatives (FN) – The model predicted 'NO' but the prediction was *False*

True Negatives (TN) – The model predicted 'NO' and the prediction was *True*

Accuracy = $\frac{True\ Positives + True\ Negatives}{Total\ size\ of\ Data}$

Accuracy is the most common evaluation metric used for classification problems. In particular, accuracy is very useful when there are an **equal number of samples in each class**. Since our dataset have similar characteristics, accuracy would be a suitable metric to evaluate the model.

Analysis

Data Exploration

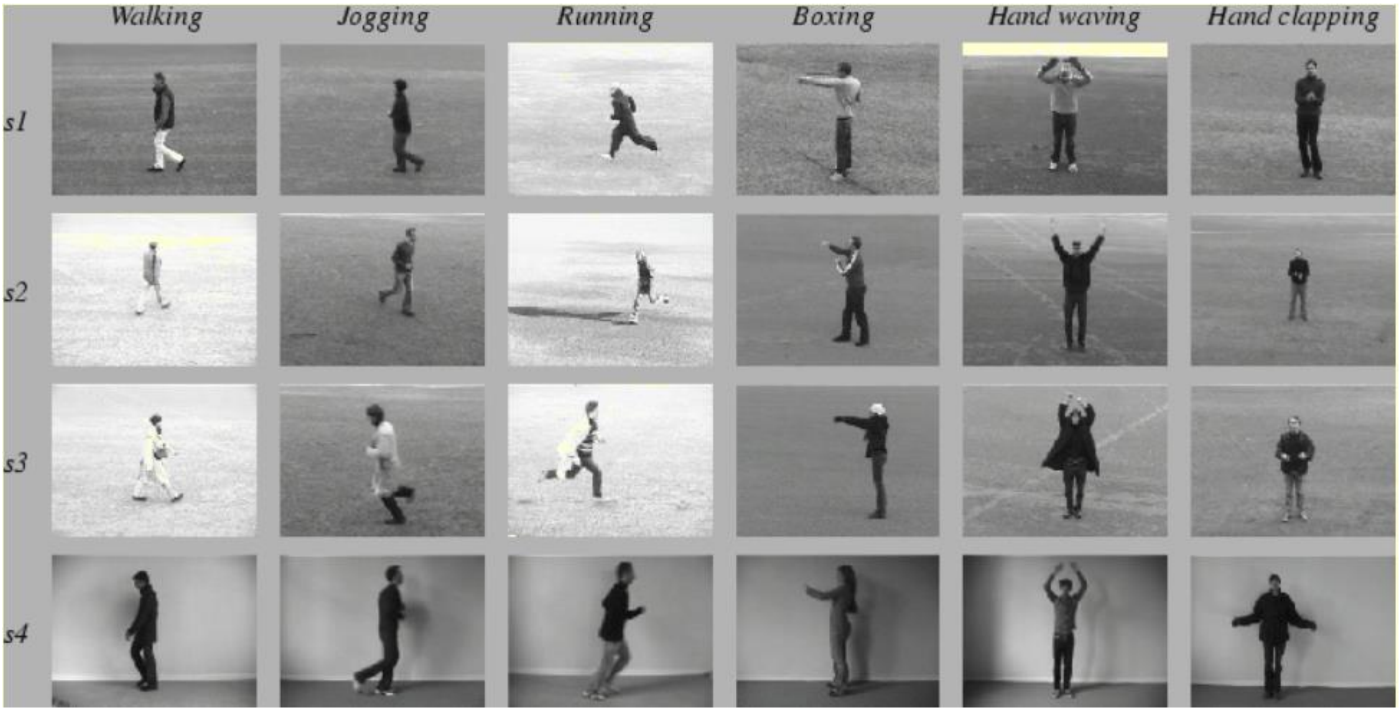
The dataset can be obtained here – Recognition of Human Actions

→ 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 599 videos – 100 videos for each of the 6 categories (with the exception of Handclapping 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.

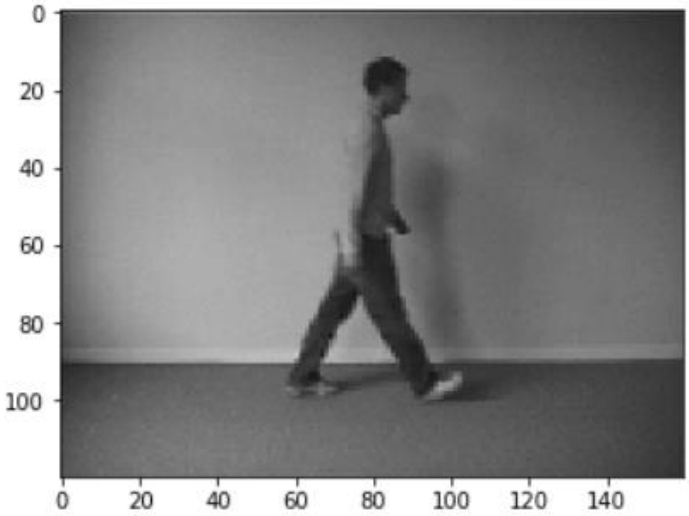
While loading the data, we convert these text labels into integers according to the following mapping:

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

Further instructions to obtain the data are mentioned here - [Link]

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.

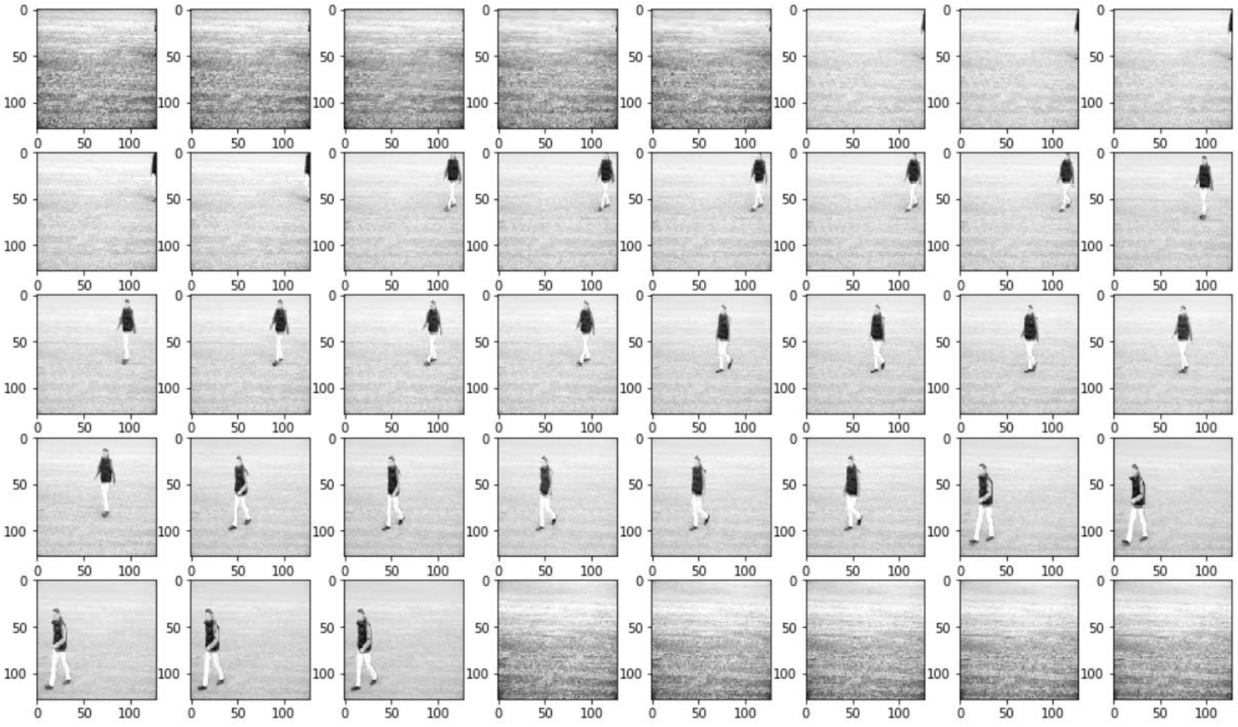
Also, on loading a single video into a NumPy array in python,

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.



The frames of a sample video of ‘Walking’

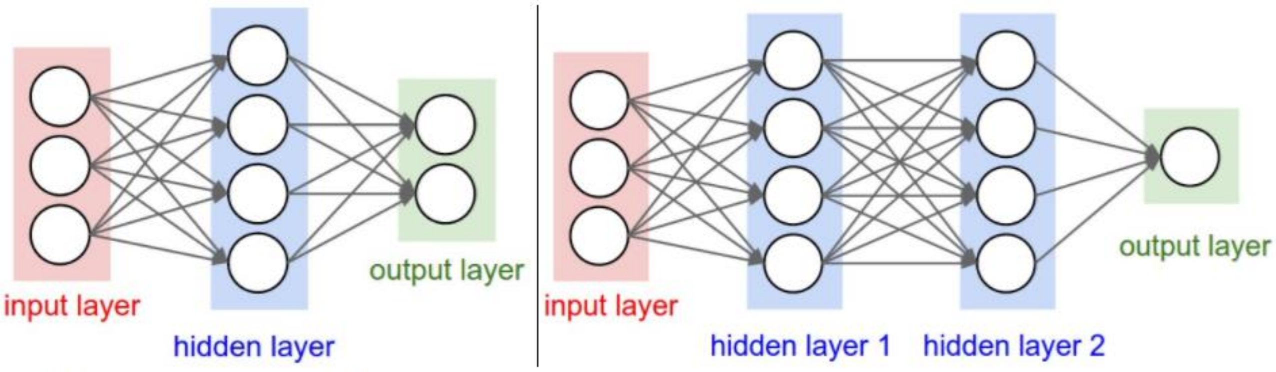
The above visual shows that in the videos of activities involving the movement of the whole body, a lot of frames in the video might be empty (no person performing any action). Also, if the person is moving very slowly, then most of the frames would be redundant.
It would be a major challenge for the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

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.

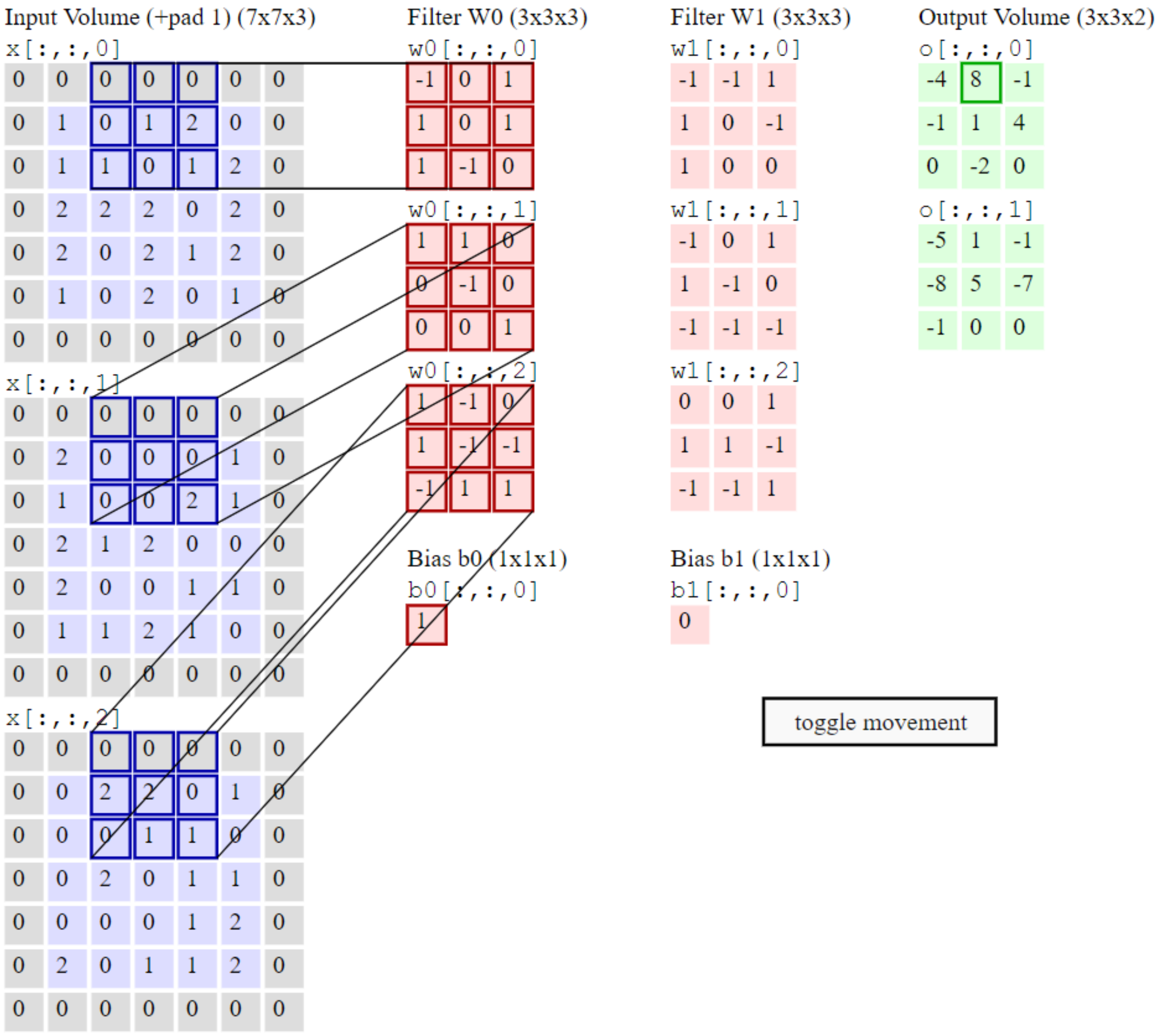


Left: A 2-layer Neural Network (one hidden layer of 4 neurons (or units) and one output layer with 2 neurons), and three inputs. **Right:** A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) 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.

<u>Model Parameters</u>

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`).

Benchmark

The existing models use the notion of local features in space-time to capture and describe local events in a video. The general idea is to describe such events is to define several types of image descriptors over local spatio-temporal neighbourhoods and evaluate these descriptors in the context of recognizing human activities. These points have stable locations in space-time and provide a potential basis for part-based representations of complex motions in video.

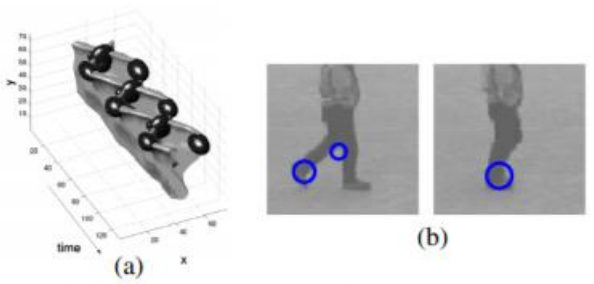
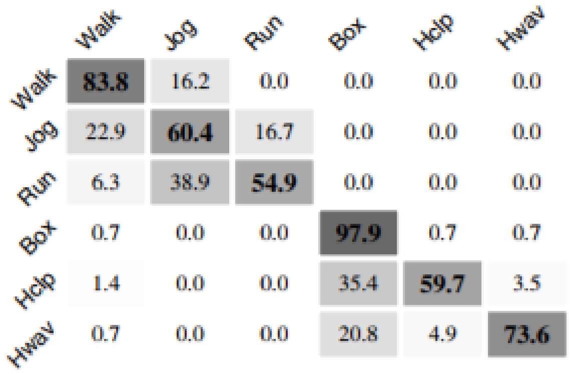


Figure 1. Local space-time features detected for a walking pattern: (a) 3-D plot of a spatio-temporal leg motion (up side down) and corresponding features (in black); (b) Features overlaid on selected frames of a sequence.



	Walk	Jog	Run	Box	Hclp	Hwav
Walk	**83.8**	16.2	0.0	0.0	0.0	0.0
Jog	22.9	**60.4**	16.7	0.0	0.0	0.0
Run	6.3	38.9	**54.9**	0.0	0.0	0.0
Box	0.7	0.0	0.0	**97.9**	0.7	0.7
Hclp	1.4	0.0	0.0	35.4	**59.7**	3.5
Hwav	0.7	0.0	0.0	20.8	4.9	**73.6**

Confusion matrix of Benchmark model

The benchmark model was able to achieve an overall recognition rate of 80-85%. But, in order to compare the benchmark model with the proposed model, the **confusion matrix** of the benchmark model will be analysed with the confusion matrix of the proposed model.

Reference

Recognizing Human Actions: A Local SVM Approach

Methodology

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.

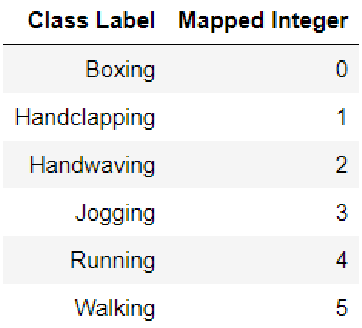
  Different strategies would be used for frame extraction like:
  - Extracting a **fixed number of frames from the total frames** in the video – say only the first 200 frames (i.e., first 8 seconds of the video).
  - Extracting a **fixed number of frames each second** from the video – say we need only 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>)`

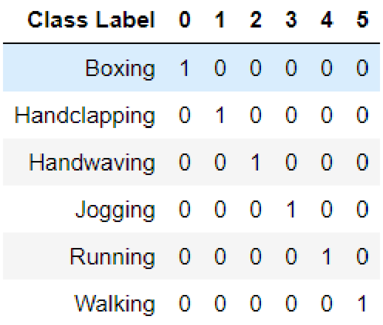
- `'channels'` can have the value 1 (grayscale) or 3 (RGB)
- `'number of frames'` - the extracted frames (will have to be the same for each video)

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

Implementation

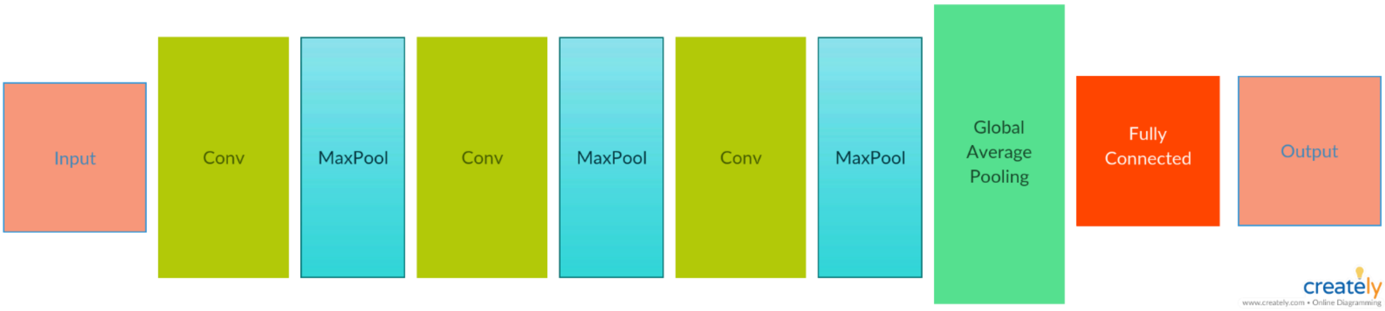
One of the most important part of the project was to load the video dataset and perform the necessary pre-processing steps. So, I developed a class (`Videos`) that had a function called (`read_videos()`) that can be used to for reading and processing videos. Creating this was very challenging as I concentrated on generalizing this function for any kind of videos (not specific to this project). The usage of helper class can be found here – Link
I have used NumPy (wherever) for storage and processing of the videos (much faster than in-built python lists with a ton of extra functionalities).

The neural network was implemented using Keras. For a detailed documentation, refer the following –

- Convolutional Layer (3D)
- MaxPooling Layer (3D)
- Dense Layer

Refinement

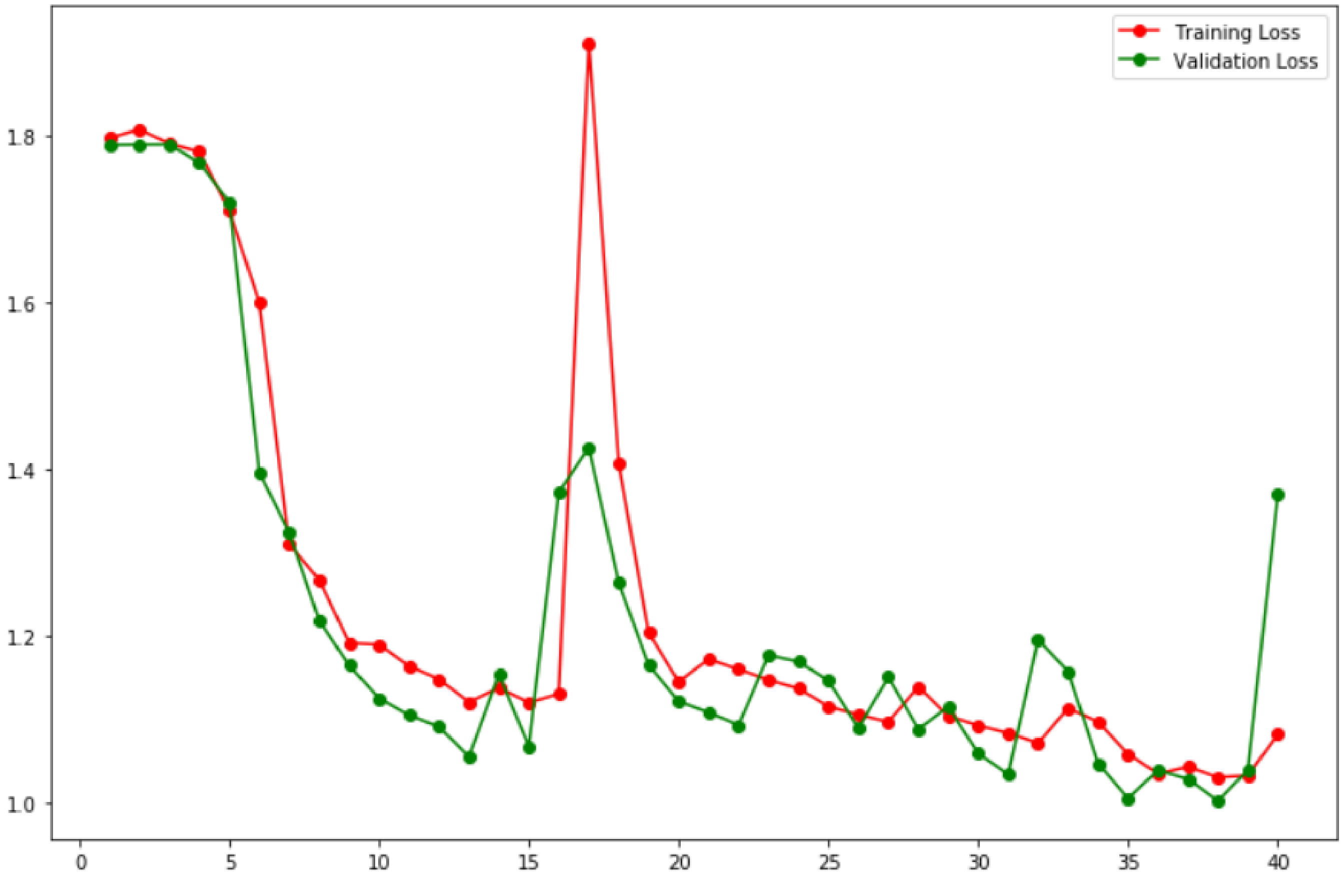
Model-1



Model Architecture

The model was trained on the training data for 40 epochs. The weights of the model which gave the best performance on the validation data were loaded. The model was then tested on the test data.

The model gave an accuracy of 37% on the test data.



Learning Curve of Model-1

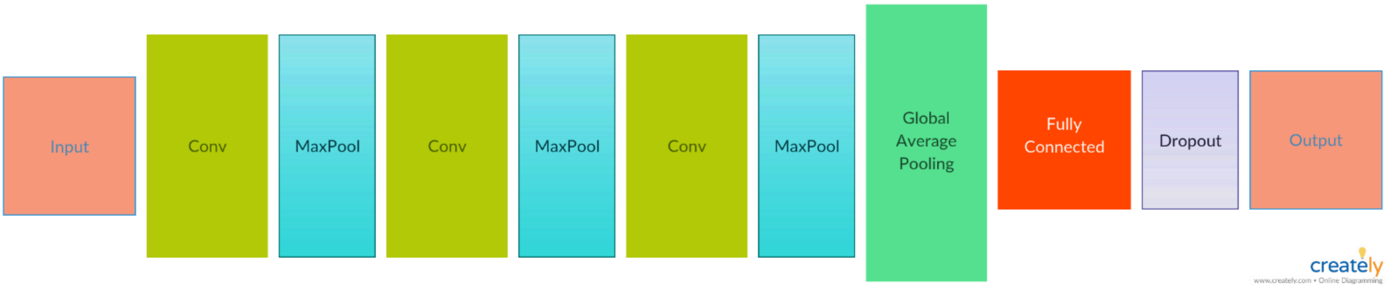
As we can observe from the learning curve, during the first 15 epochs, the training and validation loss both decrease steeply. But after that, the model starts to overfit a little. Here, overfitting means that although the model performed better on the training data, but its performance on the validation data got degraded. *This usually happens when our model is too complex for the data and it starts to memorize the training data*. We can see that during the last 5 epochs, there is a huge difference between the training and validation loss.

Model-2

In order to prevent overfitting, there is a method called **Dropout**. What this does is that at each epoch, a fraction of the neurons (of the layer to which dropout is applied) are deactivated. This forces the network to use and update the weights of the remaining neurons. The dropout is applied to the fully-connected layers.

Note: Dropout is used only while model training, and not during the testing.

In this model, I've added a dropout for the hidden layer (with 32 neurons) in the fully-connected layers. Rest of the model is same as *Model-1*.

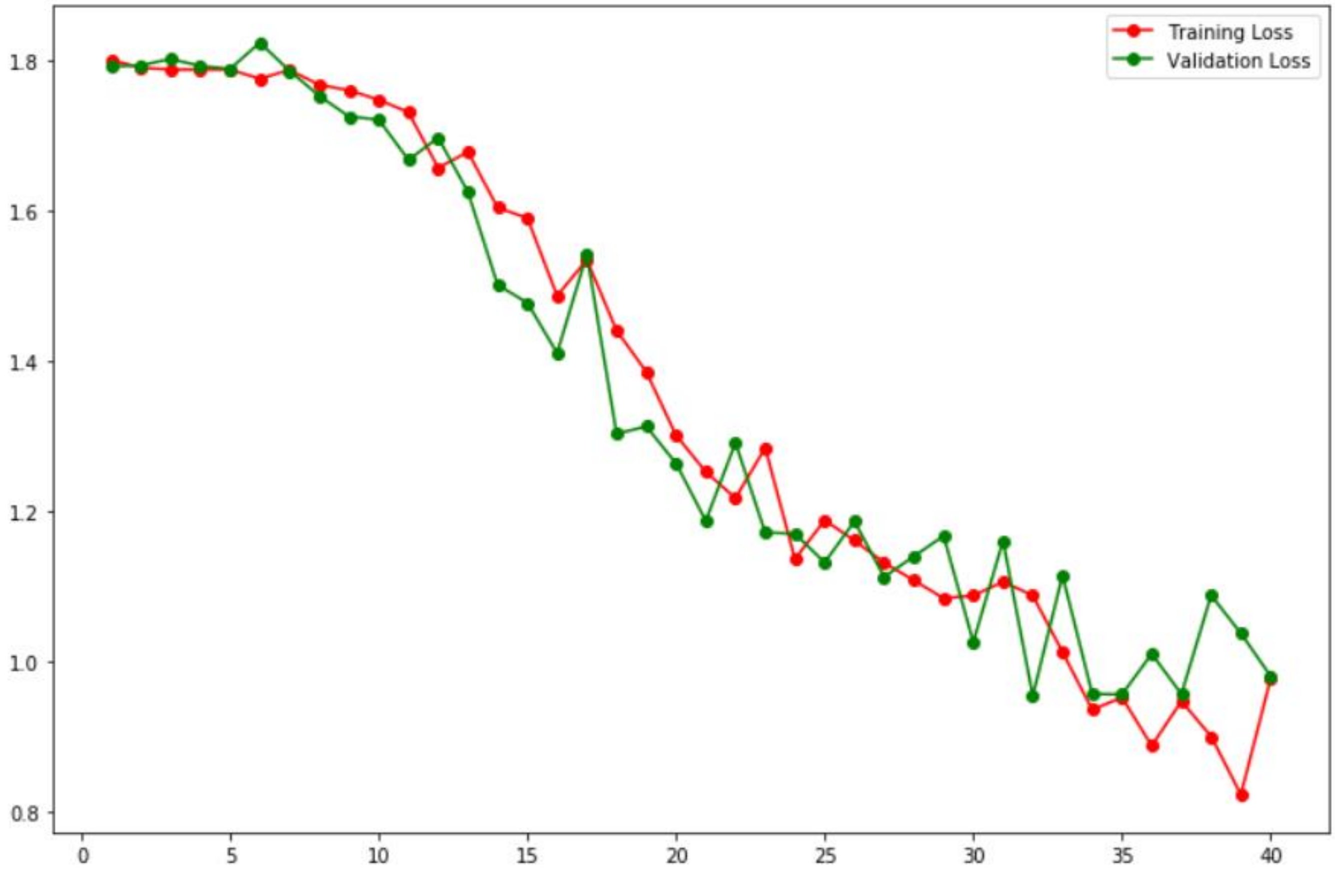


Model Architecture

The model was trained on the training data for 40 epochs. The weights of the model which gave the best performance on the validation data were loaded. The model was then tested on the test data.

The model gave an accuracy of 58.5% on the test data.

So, just by adding a dropout layer, the *model's accuracy increased by almost 22%*. This shows that the dropout prevented our model from overfitting. We can see this further in the learning curve of this model.



Learning Curve of Model-2

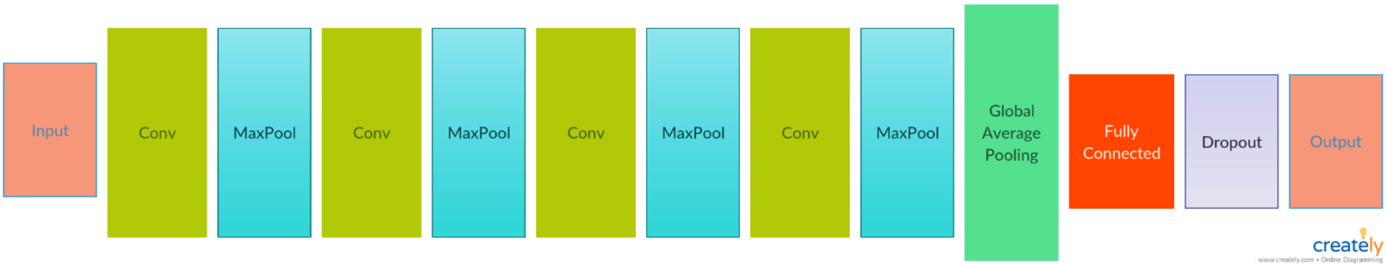
There is a *gradual decrease in both the training and validation loss*. Also, the difference between the training and validation loss is not very large, suggesting that our model is no longer overfitting the training data.

Model-3

Till now, 200 frames for each video were extracted and given as the input to the models. But the approach to extract these frames is not very appropriate. What was being done is that from each video, **200 contiguous frames (8 seconds)** were being extracted. We know that the human body performs these activities (running, boxing etc.) with a certain speed. Within one second, the human body does not make much of a movement. Therefore, we do not need to collect every frame for each second of video that we are capturing. A different approach could be used, where only a certain number of frames are extracted for each second.

Now, we would be extracting only 5 frames per second (first 5 frames for each second). So, suppose we have a video of 10 seconds, we will get 10 x 5 = 50 frames. There is also a maximum limit on the number of frames that should be extracted from each video. I have set this value to 40. So, these 40 frames will be selected from the front of the extracted frames.

The range of normalized pixels has also been changed from [0, 1] to [-1, 1]. This is because the mean of the pixels would then be 0, which would *help the model converge faster*.



Model Architecture

The model was trained on the training data for 40 epochs. The weights of the model which gave the best performance on the validation data were loaded. The model was then tested on the test data.

The model gave an accuracy of 64.5% on the test data. This model gave a higher accuracy than the previous models, despite using 5 times lesser data for training.

In this model, another pair of convolutional and max pooling layer was added. This made the output of the final convolutional layer to have a depth of 1024 (earlier it was 256).

Also, this model used **NADAM** as the optimizer (instead of **ADAM**). In Keras, the default values of learning rate for ADAM optimizer is set to 0.001. For NADAM, the default value of learning rate is 0.002 and there is a scheduled decay of learning rate.

Using NADAM as the optimizer gave better results than ADAM. Also, at the end of 40 epochs, the model did not overfit when the optimizer used was NADAM, but in case of ADAM, the model showed some signs of overfitting.

The whole project can be viewed as a jupyter notebook here - Link

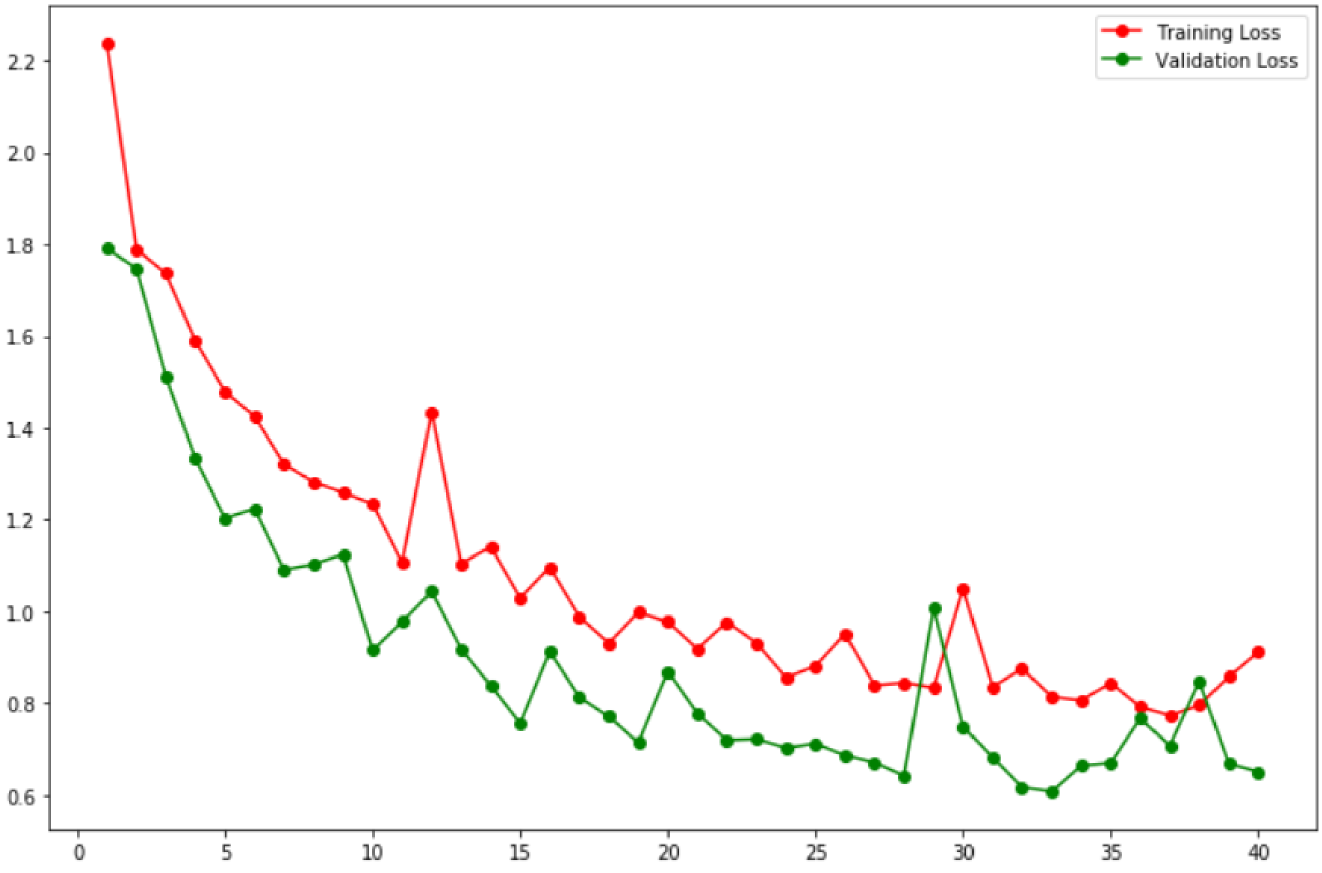
References:

1. Optimizing Gradient Descent
2. Optimizers in Keras

Results

Model Evaluation and Validation

The last model (Model-3) gave the highest accuracy on the test data (64.5%). 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.

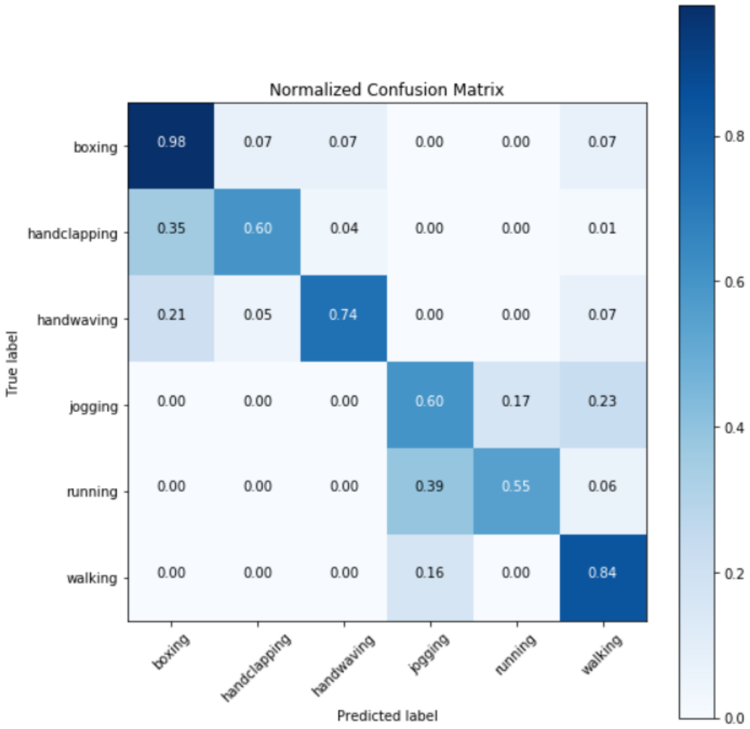
Since the input data had been reduced while pre-processing (by 5 times), the previous models would have taken considerably less time for training. It's because of this reduction in the data, that a more complex model was constructed and used. If we had the same data, this model would have taken a very 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:

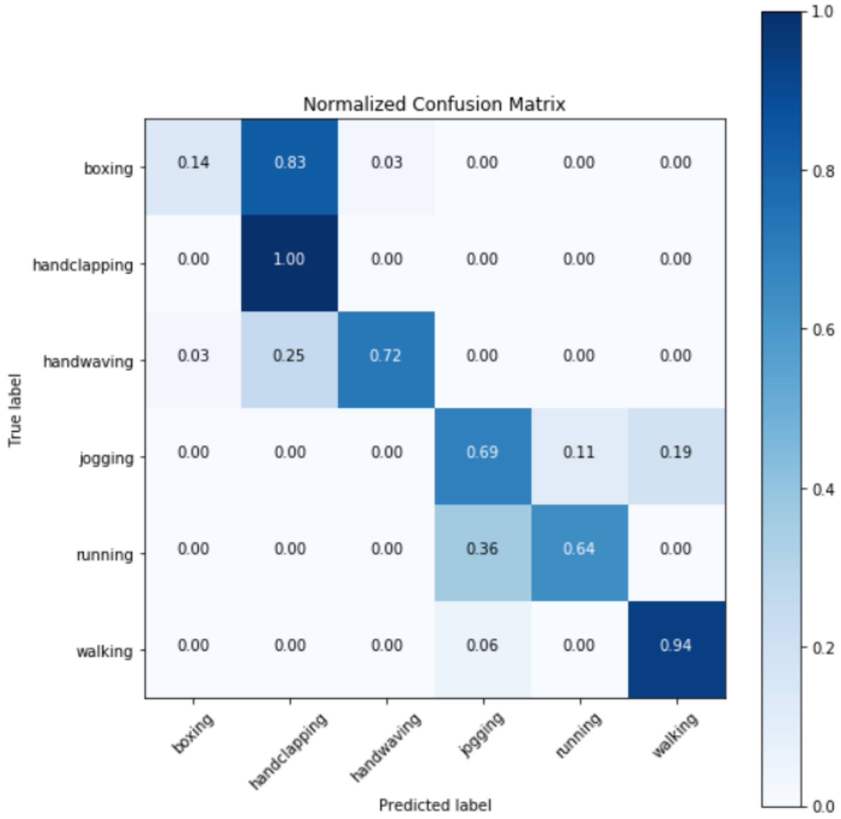
- The depth of the vector obtained by the last convolutional layer is 1024.
- A Global Average Pooling layer (GAP) then takes the average value from each of these 1024 dimensions and gives a 1-dimensional vector representing the entire video.
- The GAP is followed by a fully-connected layer containing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

Justification

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.



Benchmark Confusion Matrix (Normalized)



Proposed Model Confusion Matrix (Normalized)

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.

Comparison

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 the proposed model predicted 'walking' 94% of the time.

Similar results were obtained for –

ACTION	BENCHMARK MODEL	PROPOSED MODEL
WALKING	0.84	0.94
JOGGING	0.60	0.69
HANDWAVING	0.74	0.72
HANDCLAPPING	0.60	1.00
RUNNING	0.55	0.64

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

In fact, for 'handclapping' the proposed model gave 100% accurate results (i.e., the model successfully distinguished handclapping from all other activities and had "no confusion" when giving a prediction about handclapping).

But, for the remaining action - Boxing:

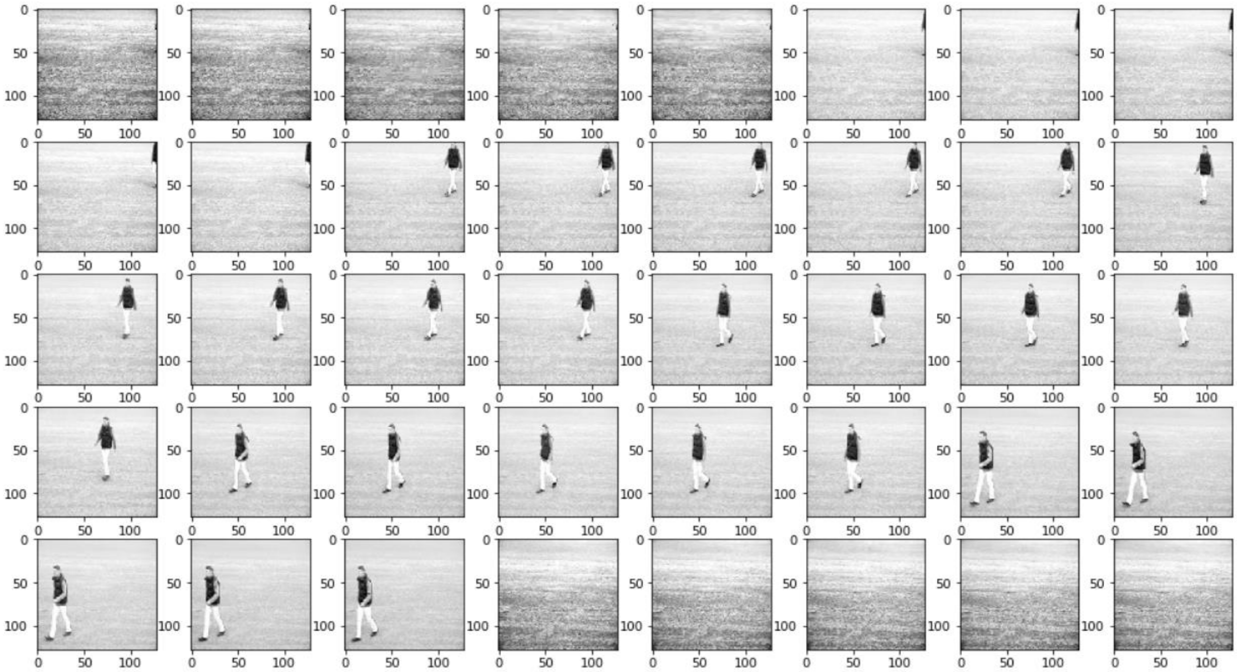
ACTION	BENCHMARK MODEL	PROPOSED MODEL
BOXING	0.98	0.14

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 the proposed model. Our model confused 'Boxing' with 'Handclapping' 83% of the time.

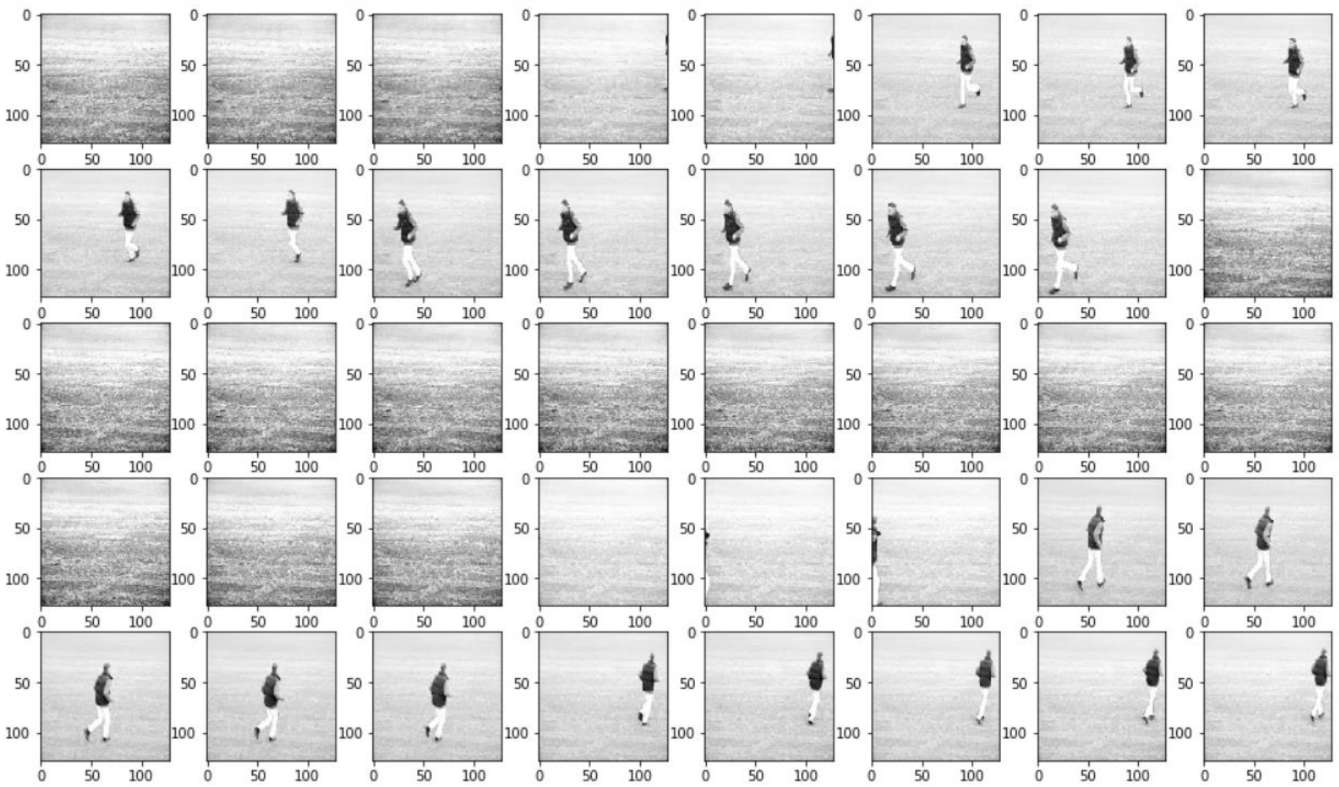
- When the actual labels are – boxing, handclapping or handwaving, the proposed model never gave the prediction as jogging, running or walking (or vice-versa). These are indicated by the cells having the values as 0. **This means that the proposed model was able to perfectly differentiate the actions that involved movement of hands from those that involved movement of legs.**
- The proposed model was able to distinguish 'running' from 'jogging' better then 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 the proposed model performed extremely poor)*.
- If 'Boxing' is kept aside, the proposed model has a much lower false positive rate than the benchmark model.

Conclusion

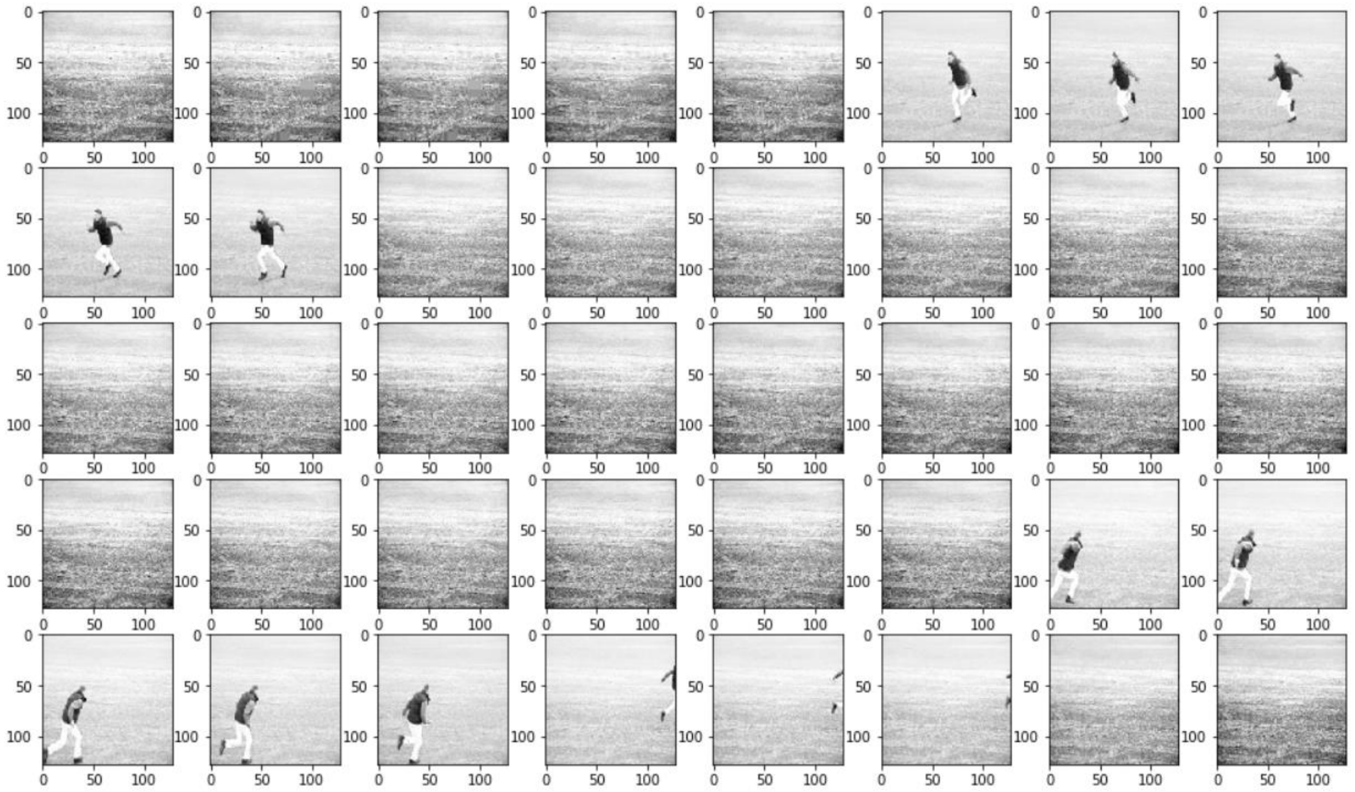
Free-Form Visualization



The frames of a processed video of 'Walking'



The frames of a processed video of ‘Jogging’



The frames of a processed video of ‘Running’

In case of ‘walking’, since the movement of the person is slow, he is captured in most of the frames.
As we increase the speed of movement of the person, more and more frames captured are empty **– they do not describe any action**.
We can see that in case of ‘running’, only a handful of frames show a person running, rest all the frames are empty. This is one major problem with the data that we have used.
Suppose we give our model a video that has all empty frames (no human performing any action). The model would still predict it as ‘running’, as the frames of the video resemble that of ‘running’ with a high probability.

This is also the reason why our model gave a lower false positive rate for the actions involving the hands (boxing, handclapping and handwaving) than those involving the legs (or movement of the entire body).
One possible way to solve the problem could be to calculate the average intensity of frames (with and without a human in the frame). Then, while selecting the frames, discard those with the average intensity similar to the average intensity of frame without any human.

Reflection

The project can be summarized in the following steps:

1. A problem domain was chosen and relevant datasets were searched for.
2. The datasets were downloaded and stored in a specific format.
3. A benchmark model was selected to compare the proposed model with.
4. A model was constructed and was trained on the training data. The model (parameters) that performed the best on the validation data was chosen finally.
5. This model was evaluated using the test data and cross-verified with the benchmark.
6. Finally, some problems related to the approach used were addressed.

In particular, the pre-processing of the data was the most interesting according to me. It’s because instead of creating programs that would work specifically for this project and data, I tried to create some generalized helper functions, that can be used for processing datasets from any similar problem domain (related to videos).

Constructing the model with different structure and hyperparameters and choosing the best was the most challenging task that I had to face.

Improvement

1. 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.

2. 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.

3. 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.

References

1. What is Deep Learning?
2. Metrics to evaluate ML algorithms
3. Intro to Neural Networks
4. Convolutional Neural Networks
5. Confusion Matrix Terminology