**HUMAN TASK RECOGNITION**

**J Component Project Report for the course**

**DATA MINING CSE3019**

**BY**

AVULA ROHITHA 17BLC1007

K. MADHU SWAPNIKA 17BLC1024

AARTHI G 17BLC1166

A project report submitted to

**Dr. ANUSHA K**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

in partial fulfilment of the requirements for the course of



**Vellore Institute of Technology (VIT) CHENNAI**

**Vandalur – Kelambakkam Road**

**Chennai – 600127**

**June 2020**

**BONAFIDE CERTIFICATE**

Certified that this project report entitled “**HUMAN POSTURE RECOGNITION”** is a bonafide work of **AVULA ROHITHA (17BLC1007), K. MADHU SWAPNIKA (17BLC1024)** and **AARTHI G (17BLC1166)** who carried out the Project work under my supervision and guidance.

**Dr. ANUSHA K**

Associate Professor

School of Computer Science and Engineering (SCOPE),

VIT University, Chennai

Chennai – 600 127.

**ABSTRACT**

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 Data Mining techniques for Video Recognition - given a set of labelled videos, train a model so that it can give a label/prediction for video. Here, the label might represent what is being performed in the video, or what the video is about.

**ACKNOWLEDGEMENT**

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Anusha K,** Associate Professor, School of Computer Science and Engineering, for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Sivasubramanian A** Dean of the School of Electronics Engineering, VIT Chennai, for extending the facilities of the School towards our project and for her unstinting support.

We express our thanks to our Programme Chair**, Dr. Thiripurasundari D for** her support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

**Table of Contents**

**Chapter No. Title Page No.**

**Abstract 3**

**1**  **Introduction** **6**

1.1 Objectives **7**

1.2 Scope

**2** **Design/ Implementation** **7-17**

2.1 Introduction **7**

2.2 Data Exploration **7-8**

2.3 Exploratory Visualization **9**

2.4 Algorithms and Techniques **9-17**

2.4.1 Traditional neural network **9-10**

2.4.2 Convolutional neural network **10-11**

2.4.3 Conv 3D **11-13**

2.4.4 Pooling Layers  **13**

2.4.5 Max Pooling 3D **13-14**

2.4.6 Global Average pooling 3D **14**

2.4.7 Batch Normalization **14-16**

2.4.8 Activation **17**

2.5 Classification Algorithms **17-22**

2.5.1 Logistic Regression **17-18**

2.5.2 K-Means Clustering **18-19**

2.5.3 Support Vector Machine **19**

2.6 Model Parameters **19-20**

2.7 Benchmark **20**

2.8 Methodology **20-21**

2.9 Implementation **22**

2.10 Refinement **22-25**

2.10.1 Model 1 **22-23**

2.10.2 Model 2 **23-24**

2.9.3 Model 3 **25**

**3** **Result and Analysis/Testing** **26-46**

3.1 Result  **26-46**

3.2 Analysis **46**

**4 Conclusion and future Enhancement 47**

**5 Appendix 48-52**

**6 References 52**

**Bio- Data 53**

**CHAPTER 1**

**INTRODUCTION AND STATEMENT OF THE PROBLEM**

* 1. **OBJECTIVE**

Human activity recognition (HAR) aims to recognize activities from a series of observations on the actions of subjects and the environmental conditions. The vision-based HAR research is the basis of many applications including video surveillance, health care, and human-computer interaction (HCI). This review highlights the advances of state-of-the-art activity recognition approaches, especially for the activity representation and classification methods. For the representation methods, we sort out a chronological research trajectory from global representations to local representations, and recent depth-based representations. For the classification methods, we conform to the categorization of template-based methods, discriminative models, and generative models and review several prevalent methods. Next, representative and available datasets are introduced. Aiming to provide an overview of those methods and a convenient way of comparing them, we classify existing literatures with a detailed taxonomy including representation and classification methods, as well as the datasets they used. Finally, we investigate the directions for future research.

* 1. **STATEMENT OF THE PROBLEM**

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 videos, 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.

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

* 1. **SCOPE**

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.

**CHAPTER 2**

**DESIGN/IMPLEMENTATION**

**2.1 INTRODUCTION**

Research in Human Activity Recognition is in massive demand due to its applications in Health care domain, Computer Vision, Household safety and Robot Learning. Human Activity Recognition (HAR) aims to identify the actions carried out by a person given a set of observations of him/her and the surrounding environment.

Human activity recognition (HAR) is a widely studied computer vision problem. Applications of HAR include video surveillance, health care, and human-computer interaction. As the imaging technique advances and the camera device upgrades, novel approaches for HAR constantly emerge. This review aims to provide a comprehensive introduction to the video-based human activity recognition, giving an overview of various approaches as well as their evolutions by covering both the representative classical literatures and the state-of-the-art approaches.

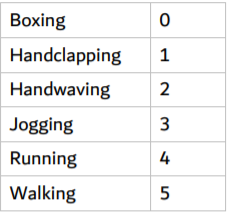
## 2.2 DATA EXPLORATION

* 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.

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

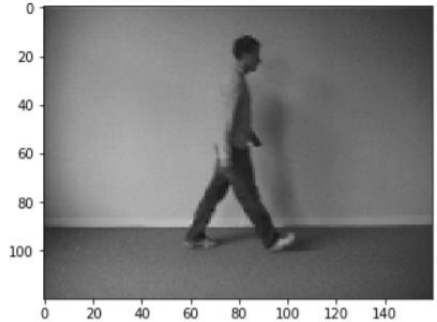


There is a total of 6 categories - boxing, handclapping, handwaving, jogging, running and walking. While loading the data, we convert these text labels into integers according to the following mapping:



**2.3 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:

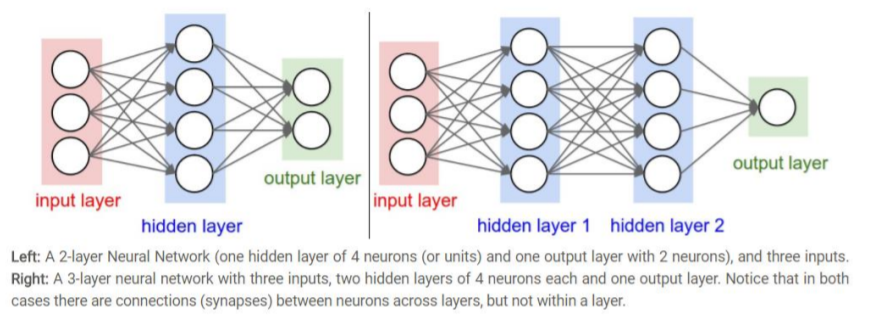
* There is 1 video
* The video has 515 frames
* The spatial dimension of the video is 160 x 120 (width x height) pixels
* Each frame has 3 channels – Red(R), Green(G) and Blue(B) A similar methodology would be used for reading in the entire dataset.

**2.4 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.

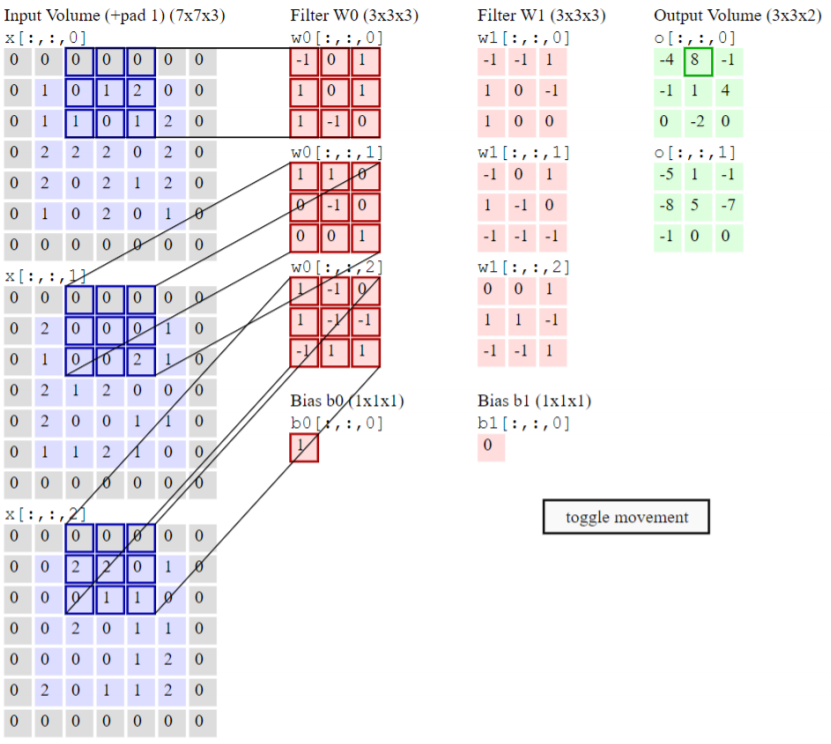
**2.4.1 TRADITIONAL NEURAL NETWORKS:**

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.



**2.4.2 CONVOLUTION 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 (MaxPooling 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.



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.

**2.4.3 CONV 3D:**

3D convolution layer (e.g. spatial convolution over volumes).

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use\_bias is True, a bias vector is created and added to the outputs. Finally, if activation is not None, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword argument input\_shape (tuple of integers, does not include the sample axis), e.g. input\_shape=(128, 128, 128, 1) for 128x128x128 volumes with a single channel, in data\_format="channels\_last".

**Arguments**

* **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
* **kernel\_size:** An integer or tuple/list of 3 integers, specifying the depth, height and width of the 3D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
* **strides**: An integer or tuple/list of 3 integers, specifying the strides of the convolution along each spatial dimension. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value! = 1 is incompatible with specifying any dilation\_rate value! = 1.
* **padding:** one of "valid" or "same" (case-insensitive).
* **data\_format:** A string, one of channels\_last (default) or channels\_first. The ordering of the dimensions in the inputs. channels\_last corresponds to inputs with shape (batch\_size, spatial\_dim1, spatial\_dim2, spatial\_dim3, channels) while channels\_first corresponds to inputs with shape (batch\_size, channels, spatial\_dim1, spatial\_dim2, spatial\_dim3). It defaults to the image\_data\_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels\_last".
* **dilation\_rate:** an integer or tuple/list of 3 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation\_rate value != 1 is incompatible with specifying any stride value != 1.
* **activation:** Activation function to use. If you don't specify anything, no activation is applied ( see keras.activations).
* **use\_bias:** Boolean, whether the layer uses a bias vector.
* **kernel\_initializer:** Initializer for the kernel weights matrix ( see keras.initializers).
* **bias\_initializer:** Initializer for the bias vector ( see keras.initializers).
* **kernel\_regularizer:** Regularizer function applied to the kernel weights matrix ( see keras.regularizers).
* **bias\_regularizer:** Regularizer function applied to the bias vector ( see keras.regularizers).
* **activity\_regularizer:** Regularizer function applied to the output of the layer (its "activation") ( see keras.regularizers).
* **kernel\_constraint:** Constraint function applied to the kernel matrix ( see keras.constraints).
* **bias\_constraint:** Constraint function applied to the bias vector ( see keras.constraints).

**Input shape**

* **5D tensor with shape:** (batch\_size, channels, conv\_dim1, conv\_dim2, conv\_dim3) if data\_format='channels\_first' or 5D tensor with shape: (batch\_size, conv\_dim1, conv\_dim2, conv\_dim3, channels) if data\_format='channels\_last'.

**Output shape**

* **5D tensor with shape:** (batch\_size, filters, new\_conv\_dim1, new\_conv\_dim2, new\_conv\_dim3) if data\_format='channels\_first' or 5D tensor with shape: (batch\_size, new\_conv\_dim1,new\_conv\_dim2,new\_conv\_dim3,filters)
* **If data\_format='channels:**\_last' new\_conv\_dim1, new\_conv\_dim2 and new\_conv\_dim3 values might have changed due to padding.

**Returns**

A tensor of rank 5 representing activation(conv3d(inputs, kernel) + bias).

**Raises**

* **ValueError:** if padding is "causal".
* **ValueError:** when both strides > 1 and dilation\_rate > 1.

**2.4.4 POOLING LAYERS:**

After the data is activated, it is sent through a [pooling layer](http://deeplearning.stanford.edu/tutorial/supervised/Pooling/). Pooling "downsamples" an image, meaning that it takes the information which represents the image and compresses it, making it smaller. The pooling process makes the network more flexible and more adept at recognizing objects/images based on the relevant features.

When we look at an image, we typically aren't concerned with all the information in the background of the image, only the features we care about, such as people or animals.

Similarly, a pooling layer in a CNN will abstract away the unnecessary parts of the image, keeping only the parts of the image it thinks are relevant, as controlled by the specified size of the pooling layer.

Because it has to make decisions about the most relevant parts of the image, the hope is that the network will learn only the parts of the image that truly represent the object in question. This helps prevent [overfitting](https://en.wikipedia.org/wiki/Overfitting), where the network learns aspects of the training case too well and fails to generalize to new data.

There are various ways to pool values, but [max pooling](https://computersciencewiki.org/index.php/Max-pooling_/_Pooling) is most commonly used. Max pooling obtains the maximum value of the pixels within a single filter (within a single spot in the image). This drops 3/4ths of information, assuming 2 x 2 filters are being used.

The maximum values of the pixels are used in order to account for possible image distortions, and the parameters/size of the image are reduced in order to control for overfitting. There are other pooling types such as average pooling or sum pooling, but these aren't used as frequently because max pooling tends to yield better accuracy.

**2.4.5 MAX POOLING 3D**

Max pooling operation for 3D data (spatial or spatio-temporal).

**Arguments:**

* **pool\_size:** Tuple of 3 integers, factors by which to downscale (dim1, dim2, dim3). (2, 2, 2) will halve the size of the 3D input in each dimension.
* **strides:** tuple of 3 integers, or None. Strides values.
* **padding:** One of "valid" or "same" (case-insensitive).
* **data\_format:** A string, one of channels\_last (default) or channels\_first. The ordering of the dimensions in the inputs. channels\_last corresponds to inputs with shape (batch, spatial\_dim1, spatial\_dim2, spatial\_dim3, channels) while channels\_first corresponds to inputs with shape (batch, channels, spatial\_dim1, spatial\_dim2, spatial\_dim3). It defaults to the image\_data\_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels\_last".

**Input shape:**

* **If data\_format='channels\_last':** 5D tensor with shape: (batch\_size, spatial\_dim1, spatial\_dim2, spatial\_dim3, channels)
* **If data\_format='channels\_first':** 5D tensor with shape: (batch\_size, channels, spatial\_dim1, spatial\_dim2, spatial\_dim3)

**Output shape:**

* **If data\_format='channels\_last':** 5D tensor with shape: (batch\_size, pooled\_dim1, pooled\_dim2, pooled\_dim3, channels)
* **If data\_format='channels\_first':** 5D tensor with shape: (batch\_size, channels, pooled\_dim1, pooled\_dim2, pooled\_dim3)

**2.4.6 GLOBAL AVERAGE POOLING 3D:**

Global Average pooling operation for 3D data.

**Arguments**

* **data\_format:** A string, one of channels\_last (default) or channels\_first. The ordering of the dimensions in the inputs. channels\_last corresponds to inputs with shape (batch, spatial\_dim1, spatial\_dim2, spatial\_dim3, channels) while channels\_first corresponds to inputs with shape (batch, channels, spatial\_dim1, spatial\_dim2, spatial\_dim3). It defaults to the image\_data\_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels\_last".

**Input shape**

* **If data\_format='channels\_last':** 5D tensor with shape: (batch\_size, spatial\_dim1, spatial\_dim2, spatial\_dim3, channels)
* **If data\_format='channels\_first':** 5D tensor with shape: (batch\_size, channels, spatial\_dim1, spatial\_dim2, spatial\_dim3)

**Output shape**

2D tensor with shape (batch\_size, channels).

**2.4.6 BATCH NORMALIZATION:**

Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

Batch normalization differs from other layers in several key aspects:

1) Adding BatchNormalization with training=True to a model causes the result of one example to depend on the contents of all other examples in a minibatch. Be careful when padding batches or masking examples, as these can change the minibatch statistics and affect other examples.

2) Updates to the weights (moving statistics) are based on the forward pass of a model rather than the result of gradient computations.

3) When performing inference using a model containing batch normalization, it is generally (though not always) desirable to use accumulated statistics rather than mini-batch statistics. This is accomplished by passing training=False when calling the model, or using model.predict.

**Arguments:**

* **axis:** Integer, the axis that should be normalized (typically the features axis). For instance, after a Conv2D layer with data\_format="channels\_first", set axis=1 in BatchNormalization.
* **momentum:** Momentum for the moving average.
* **epsilon:** Small float added to variance to avoid dividing by zero.
* **center:** If True, add offset of beta to normalized tensor. If False, beta is ignored.
* **scale:** If True, multiply by gamma. If False, gamma is not used. When the next layer is linear (also e.g. nn.relu), this can be disabled since the scaling will be done by the next layer.
* **beta\_initializer:** Initializer for the beta weight.
* **gamma\_initializer:** Initializer for the gamma weight.
* **moving\_mean\_initializer:** Initializer for the moving mean.
* **moving\_variance\_initializer:** Initializer for the moving variance.
* **beta\_regularizer:** Optional regularizer for the beta weight.
* **gamma\_regularizer:** Optional regularizer for the gamma weight.
* **beta\_constraint:** Optional constraint for the beta weight.
* **gamma\_constraint:** Optional constraint for the gamma weight.
* **renorm:**This adds extra variables during training. The inference is the same for either value of this parameter.
* **renorm\_clipping:** A dictionary that may map keys 'rmax', 'rmin', 'dmax' to scalar Tensors used to clip the renorm correction. The correction (r, d) is used as corrected\_value = normalized\_value \* r + d, with r clipped to [rmin, rmax], and d to [-dmax, dmax]. Missing rmax, rmin, dmax are set to inf, 0, inf, respectively.
* **renorm\_momentum:** Momentum used to update the moving means and standard deviations with renorm. Unlike momentum, this affects training and should be neither too small (which would add noise) nor too large (which would give stale estimates). Note that momentum is still applied to get the means and variances for inference.
* **fused:** if True, use a faster, fused implementation, or raise a ValueError if the fused implementation cannot be used. If None, use the faster implementation if possible. If False, do not used the fused implementation.
* **trainable:** Boolean, if True the variables will be marked as trainable.
* **virtual\_batch\_size:** An int. By default, virtual\_batch\_size is None, which means batch normalization is performed across the whole batch. When virtual\_batch\_size is not None, instead perform "Ghost Batch Normalization", which creates virtual sub-batches which are each normalized separately (with shared gamma, beta, and moving statistics). Must divide the actual batch size during execution.
* **adjustment:** A function taking the Tensor containing the (dynamic) shape of the input tensor and returning a pair (scale, bias) to apply to the normalized values (before gamma and beta), only during training. For example, if axis==-1, adjustment = lambda shape: ( tf.random.uniform(shape[-1:], 0.93, 1.07), tf.random.uniform(shape[-1:], -0.1, 0.1)) will scale the normalized value by up to 7% up or down, then shift the result by up to 0.1 (with independent scaling and bias for each feature but shared across all examples), and finally apply gamma and/or beta. If None, no adjustment is applied. Cannot be specified if virtual\_batch\_size is specified.

**Call arguments:**

* **inputs:** Input tensor (of any rank).
* **training:** Python boolean indicating whether the layer should behave in training mode or in inference mode.
* **training=**True: The layer will normalize its inputs using the mean and variance of the current batch of inputs.
* **training=**False: The layer will normalize its inputs using the mean and variance of its moving statistics, learned during training.

**Input shape:**

Arbitrary. Use the keyword argument input\_shape (tuple of integers, does not include the samples axis) when using this layer as the first layer in a model.

**Output shape:**

Same shape as input.

About setting layer.trainable = False on a `BatchNormalization layer:

The meaning of setting layer.trainable = False is to freeze the layer, i.e. its internal state will not change during training: its trainable weights will not be updated during fit() or train\_on\_batch(), and its state updates will not be run.

Usually, this does not necessarily mean that the layer is run in inference mode (which is normally controlled by the training argument that can be passed when calling a layer). "Frozen state" and "inference mode" are two separate concepts.

However, in the case of the BatchNormalization layer, setting trainable = False on the layer means that the layer will be subsequently run in inference mode (meaning that it will use the moving mean and the moving variance to normalize the current batch, rather than using the mean and variance of the current batch).

This behavior has been introduced in TensorFlow 2.0, in order to enable layer.trainable = False to produce the most commonly expected behavior in the convnet fine-tuning use case.

**2.4.8 ACTIVATION FUNCTION:**

After the feature map of the image has been created, the values that represent the image are passed through an activation function or activation layer. The activation function takes values that represent the image, which are in a linear form (i.e. just a list of numbers) thanks to the convolutional layer, and increases their non-linearity since images themselves are non-linear.

The typical activation function used to accomplish this is a **Rectified Linear Unit** (ReLU), although there are some other activation functions that are occasionally used.

**Arguments:**

* **x**: Input tensor or variable.
* **alpha:** A float that governs the slope for values lower than the threshold.
* **max\_value:** A float that sets the saturation threshold (the largest value the function will return).
* **threshold:** A float giving the threshold value of the activation function below which values will be damped or set to zero.
* **Returns:**A Tensor representing the input tensor, transformed by the relu activation function. Tensor will be of the same shape and dtype of input x.

**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.

**2.5 CLASSIFICATION ALGORITHMS:**

**2.5.1 LOGISTIC REGRESSION:**

Logistic regression produces a logistic curve, which is limited to values between 0 and Logistic regression is similar to a linear regression, but the curve is constructed using the natural logarithm of the “odds” of the target variable, rather than the probability. Moreover, the predictors do not have to be normally distributed or have equal variance in each group.

In the logistic regression the constant (b0) moves the curve left and right and the slope (b1) defines the steepness of the curve. By simple transformation, the logistic regression equation can be written in terms of an odds ratio.

p / (1 - p) = exp (b0 + b1x)

Finally, taking the natural log of both sides, we can write the equation in terms of log-odds (logit) which is a linear function of the predictors. The coefficient (b1) is the amount the logit (log-odds) changes with a one-unit change in x.

ln( p / ( 1-p )) = b0 + b1x

As mentioned before, logistic regression can handle any number of numerical and/or categorical variables.

p = 1 / ( 1 + e -(b0 + b1x1 + b2x2 + … + bpxp)

There are several analogies between linear regression and logistic regression. Just as ordinary least square regression is the method used to estimate coefficients for the best fit line in linear regression, logistic regression uses [maximum likelihood estimation](https://saedsayad.com/further_readings.htm) (MLE) to obtain the model coefficients that relate predictors to the target. After this initial function is estimated, the process is repeated until LL (Log Likelihood) does not change significantly.

ꞵ1 = ꞵ0 + [XTWX]-1.XT ( y - µ )

ꞵ is a vector of the logistic regression coefficients.

W is a square matrix of order N with elements niπi (1 – πi ) on the diagonal and zeros everywhere else.

µ is a vector of length N with elements µi = niπi.

**2.5.2 K-MEANS CLUSTERING:**

K-means clustering is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster. These centroids should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other.

The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early groupage is done. At this point we need to re-calculate k new centroids as barycenters of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop we may notice that the k centroids change their location step by step until no more changes are done.

In other words, centroids do not move any more. Finally, this algorithm aims at minimizing an objective function, in this case a squared error function. The objective function

J = ∑∑ || Xi(j) – Cj ||2 ,

where || Xi(j) – Cj ||2 is a chosen distance measure between a data point Xi(j) and the cluster centre Cj, is an indicator of the distance of the n data points from their respective cluster centres.

The algorithm is composed of the following steps:

1) Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.

2) Assign each object to the group that has the closest centroid.

3) When all objects have been assigned, recalculate the positions of the K centroids.

4) Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

Although it can be proved that the procedure will always terminate, the k-means algorithm does not necessarily find the most optimal configuration, corresponding to the global objective function minimum. The algorithm is also significantly sensitive to the initial randomly selected cluster centres. The k-means algorithm can be run multiple times to reduce this effect.

**2.5.3 SUPPORT VECTOR MACHINE:**

SVM is one of the most popular, versatile supervised machine learning algorithms. It is used for both classification and regression task. Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3. Hyperplaneis line divides data point into two classes, written as:

y = a \* x + b

a \* x + b – y = 0

Let vector X = (x,y) and W = (a,-1) then in vector form, hyperplane is

W . X + b = 0

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM. Length of vectors are also called as norms. It tells how far vectors are from the origin. Length of vector x ( x1 , x2 , x3 ) is calculated as:

|| x || = √ ( x12 + x22 + x32 )

**2.6 MODEL PARAMETERS:**

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

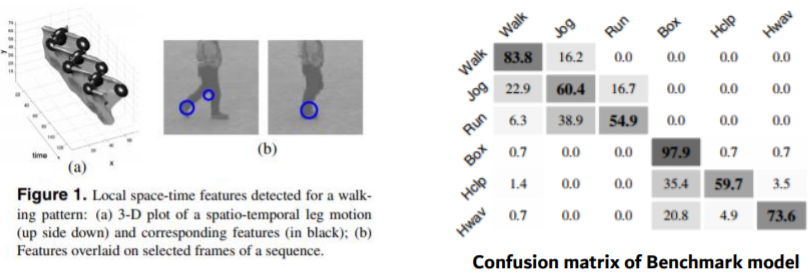
* **filters -** This is the number of feature maps required as the output of that convolutional layer.
* **kernel\_size -** The size of the window that will get convolved along all the axes of the input data to produce a single feature map.
* **strides -** The number of pixels by which the convolutional 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).

**2.7 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.

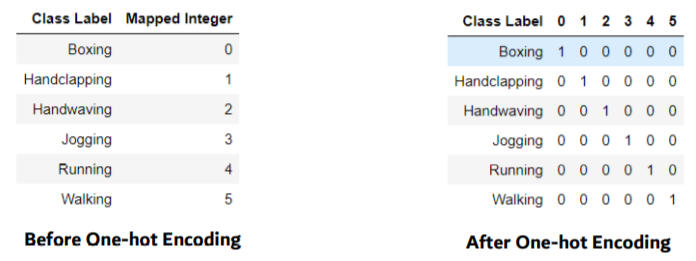


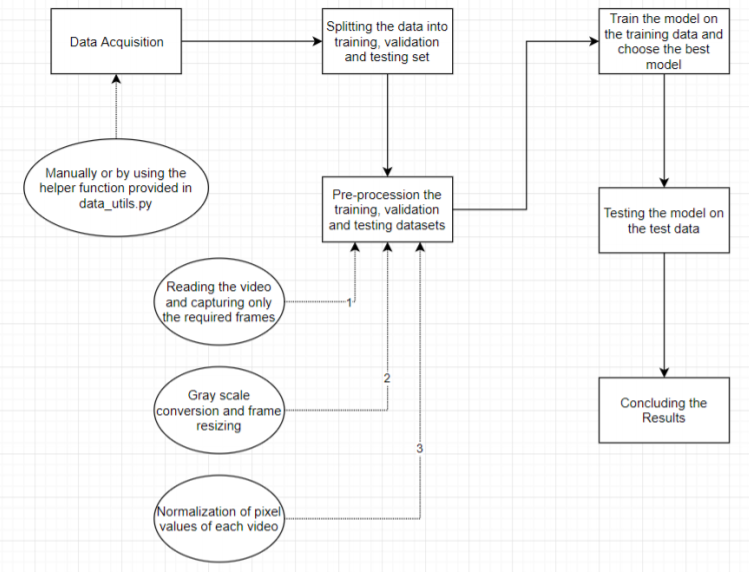
The benchmark model was able to achieve an overall recognition rate of 80-85%. But, in order to compare the benchmark model with the proposed model, the confusion matrix of the benchmark model will be analysed with the confusion matrix of the proposed model.

**2.8 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: o 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). o 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: o Min-max Normalization – Get the values of the pixels in a given range (say 0 to 1) o 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 – (, , , , ) - ‘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.



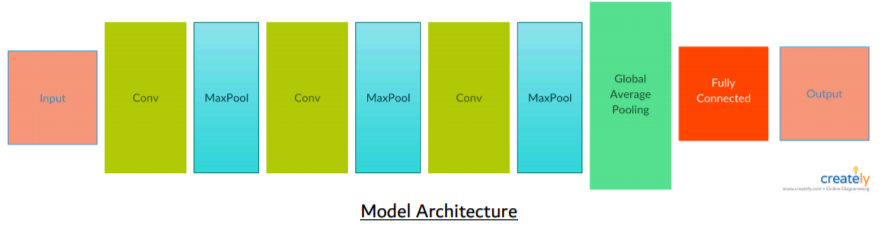


**2.9 IMPLEMENTATION:**

One of the most important part of the project was to load the video dataset and perform the necessary pre-processing steps. So, We 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 we concentrated on generalizing this function for any kind of videos (not specific to this project. We have used NumPy (wherever) for storage and processing of the videos (much faster than in-built python lists with a ton of extra functionalities).

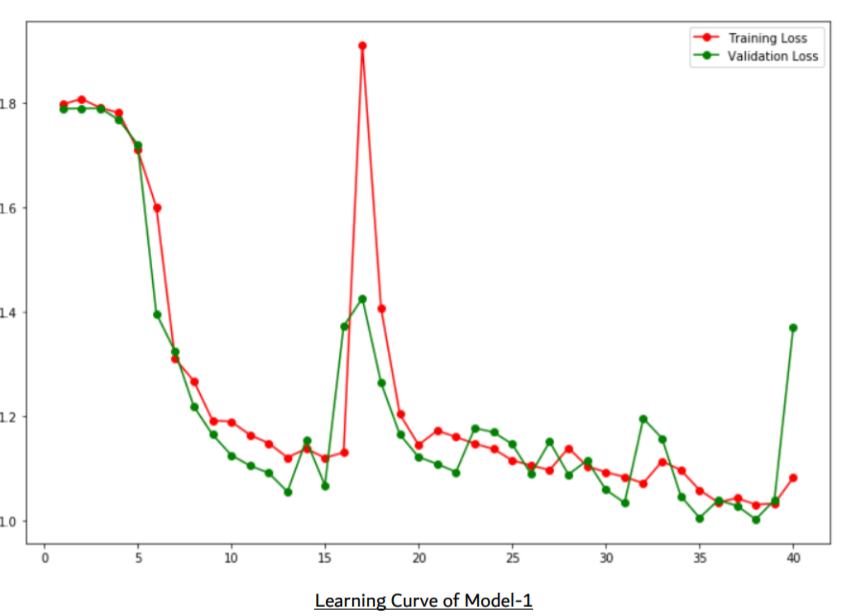
**2.10 REFINEMENT:**

**2.10.1 MODEL-1:**



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.**

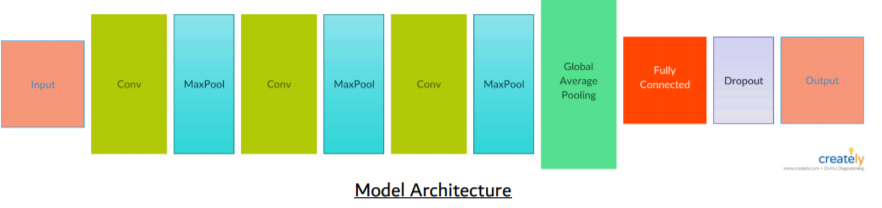


As we can observe from the learning curve, during the first 15 epochs, the training and validation loss both decrease steeply. But after that, the model starts to overfit a little. Here, overfitting means that although the model performed better on the training data, but its performance on the validation 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.

**2.10.2 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.

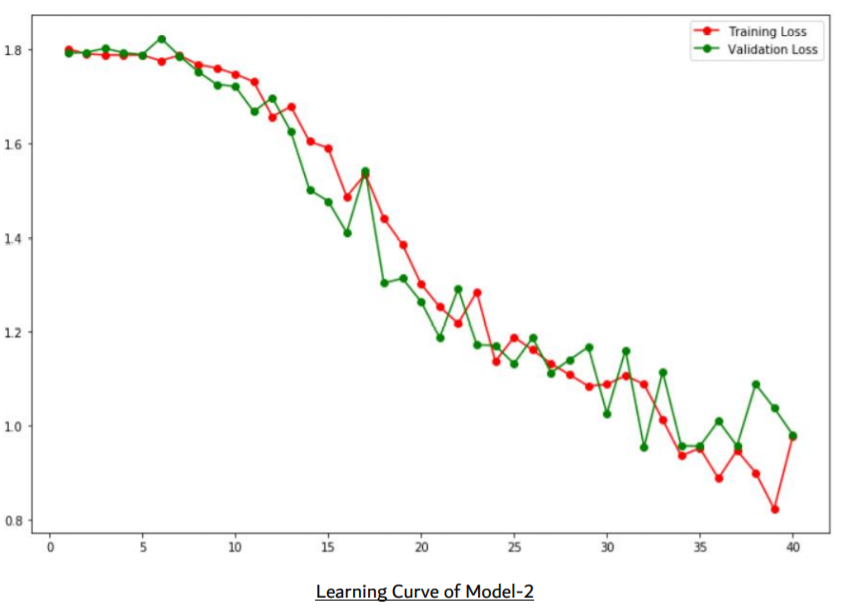
Note: Dropout is used only while model training, and not during the testing. In this model, I've added a dropout for the hidden layer (with 32 neurons) in the fully-connected layers. Rest of the model is same as Model-1.



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.



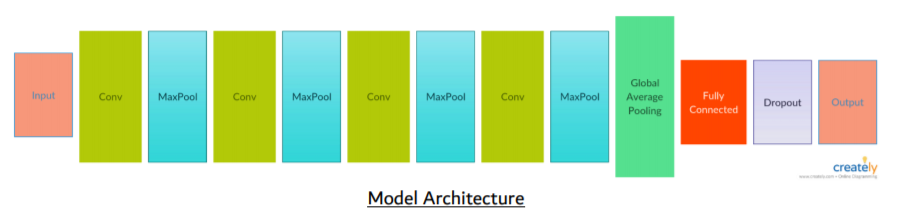
There is a \*gradual decrease in both the training and validation loss\*. Also, the difference between the training and validation loss is not very large, suggesting that our model is no longer overfitting the training data.

**2.10.3 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.



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.

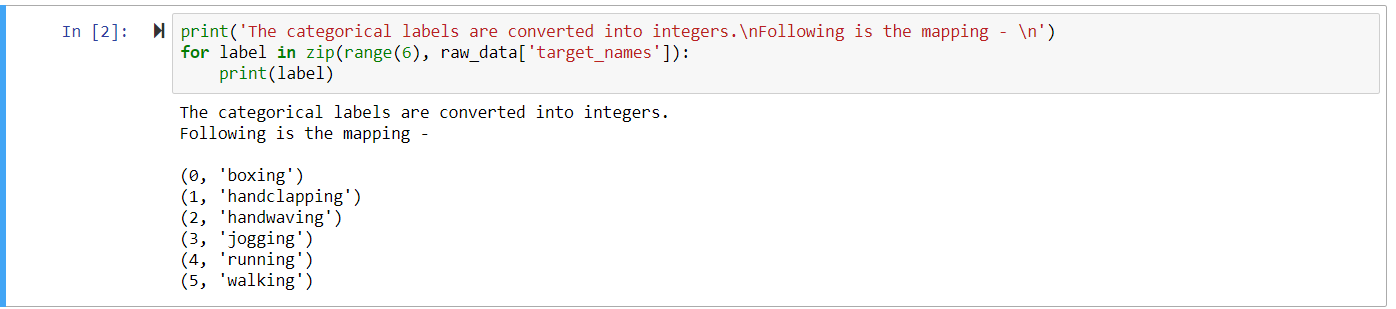
**CHAPTER 3**

**RESULTS AND ANALYSIS**

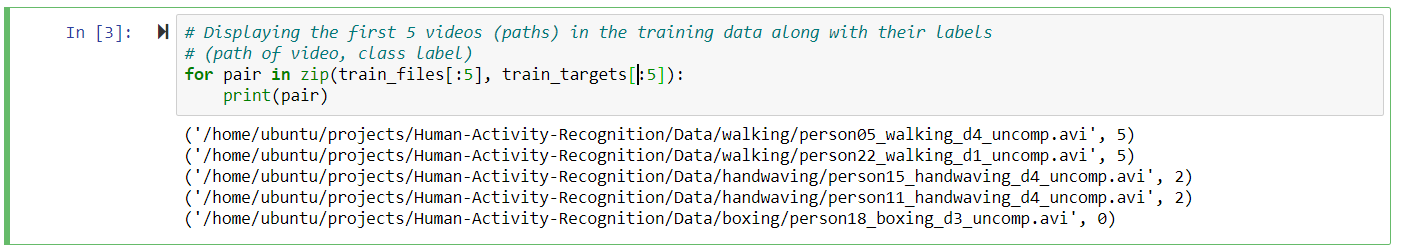
**3.1 RESULT**



**DEFINING CLASS LABELS:**



**EACH VIDEO IS ASSOCIATED WITH ITS CLASS LABEL:**



**EXPLORING VIDEO SET**

There is a helper class called Videos in utils.py for reading the videos into numpy ndarrays.

The class provides some additional functionalities like:

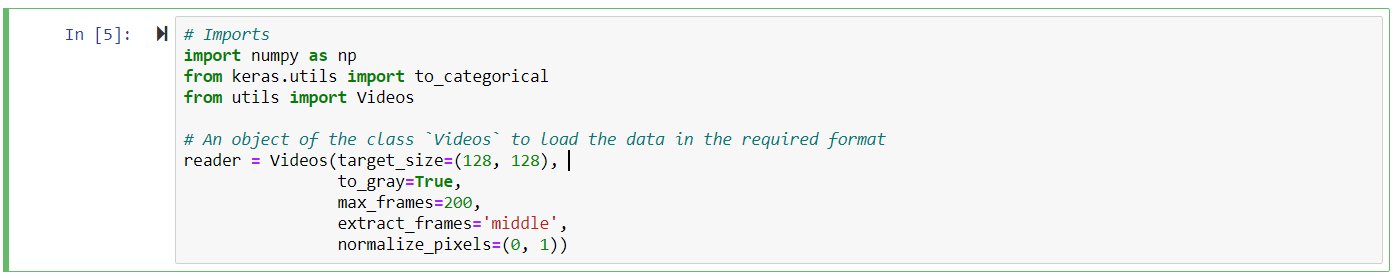
* Setting the target size for each frame of a video
* Conversion each frame to gray scale
* Various options to extract a subset of frames from each video
* Normalizing the pixel values of each video

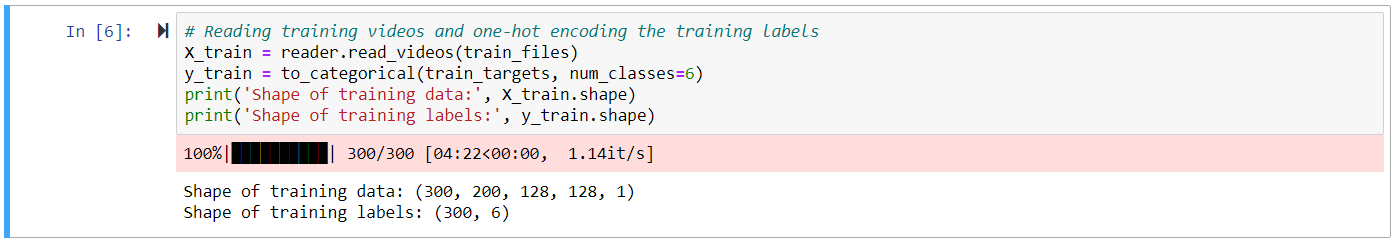
We will use this to load out training, validation and test data.

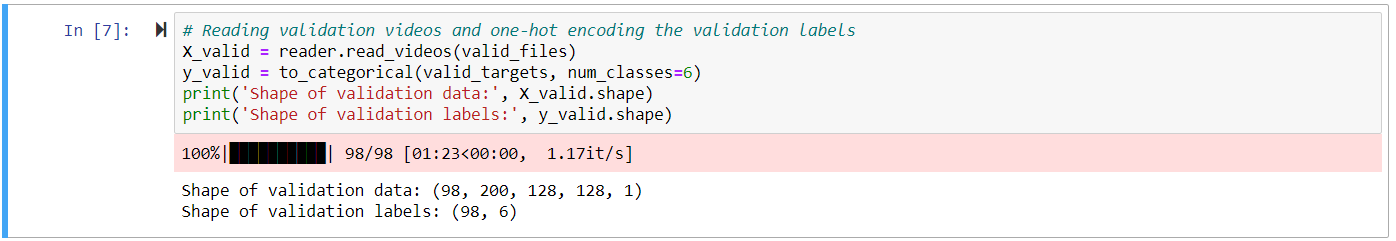


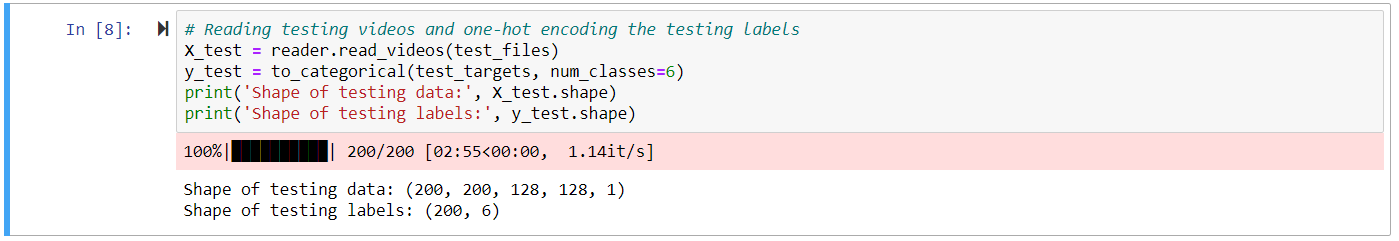


**PREPROCESSING THE DATA:**





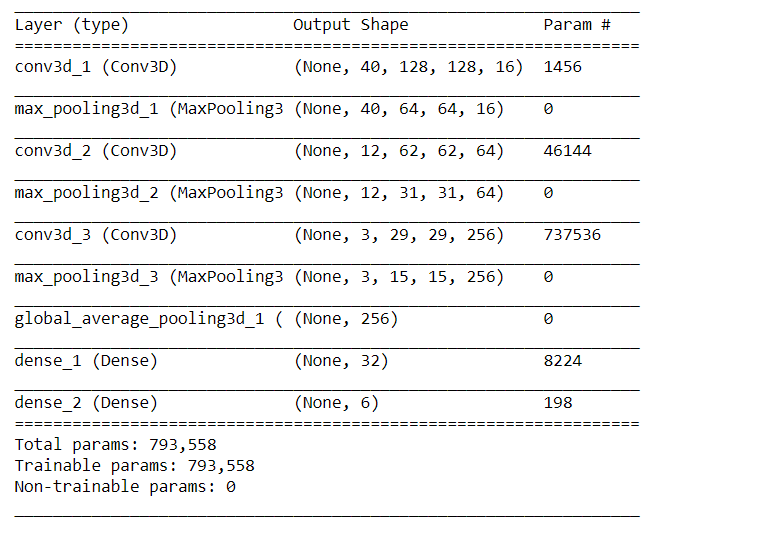






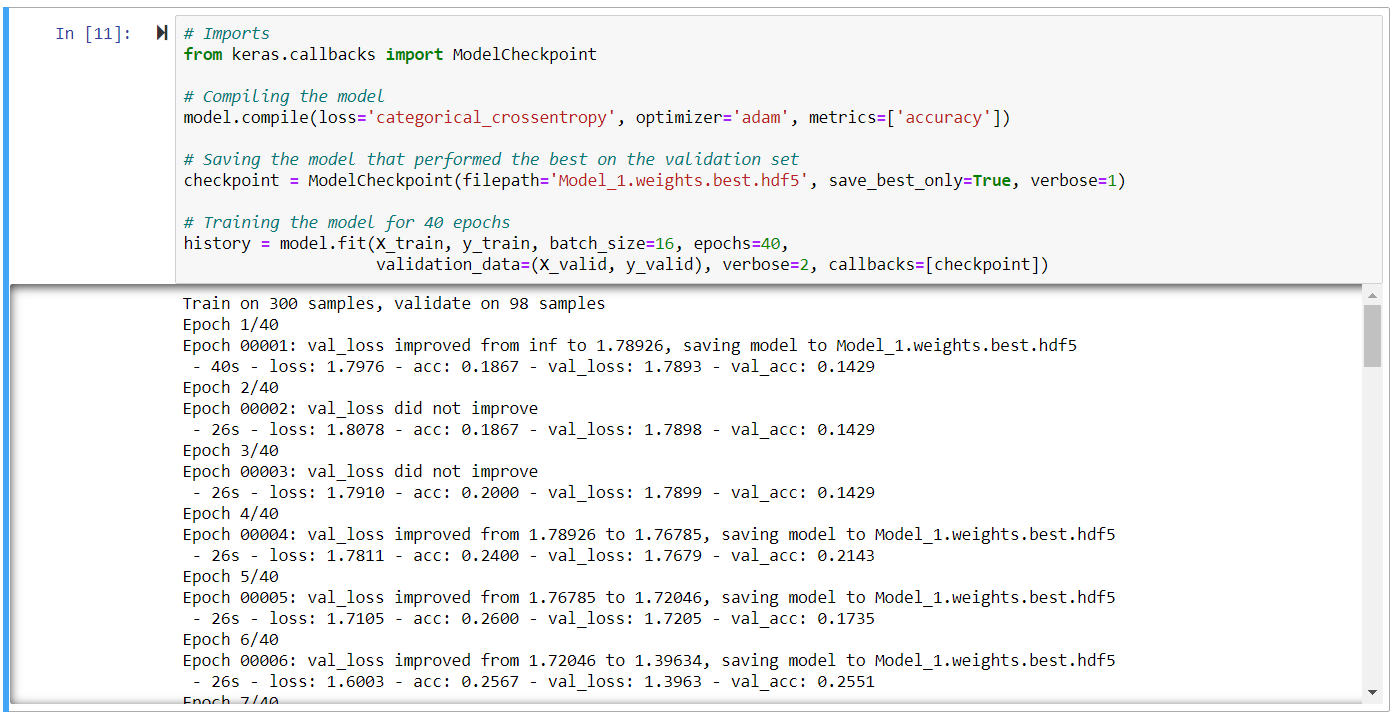
**MODEL 1 CONSTRUCTION:**

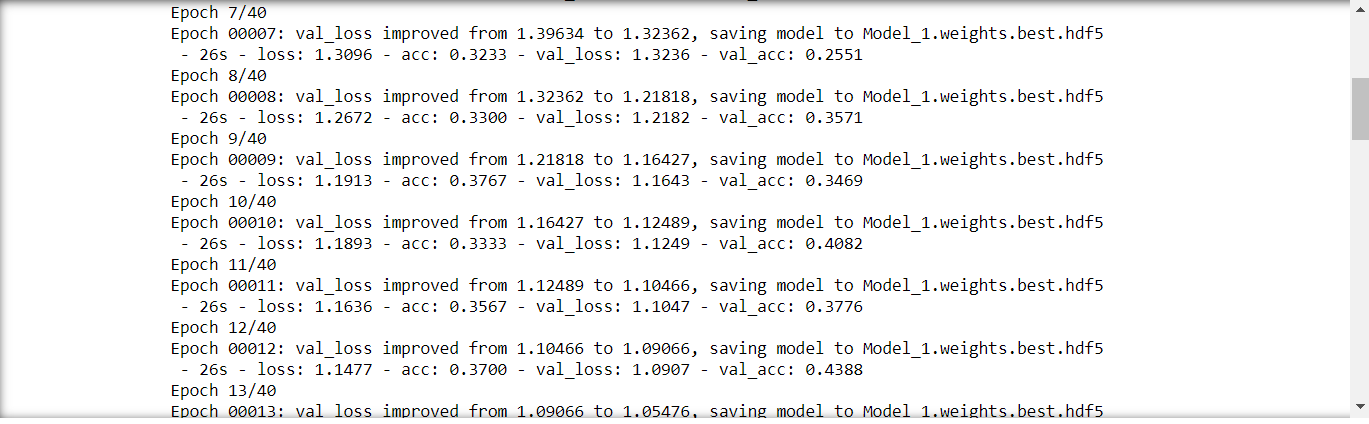


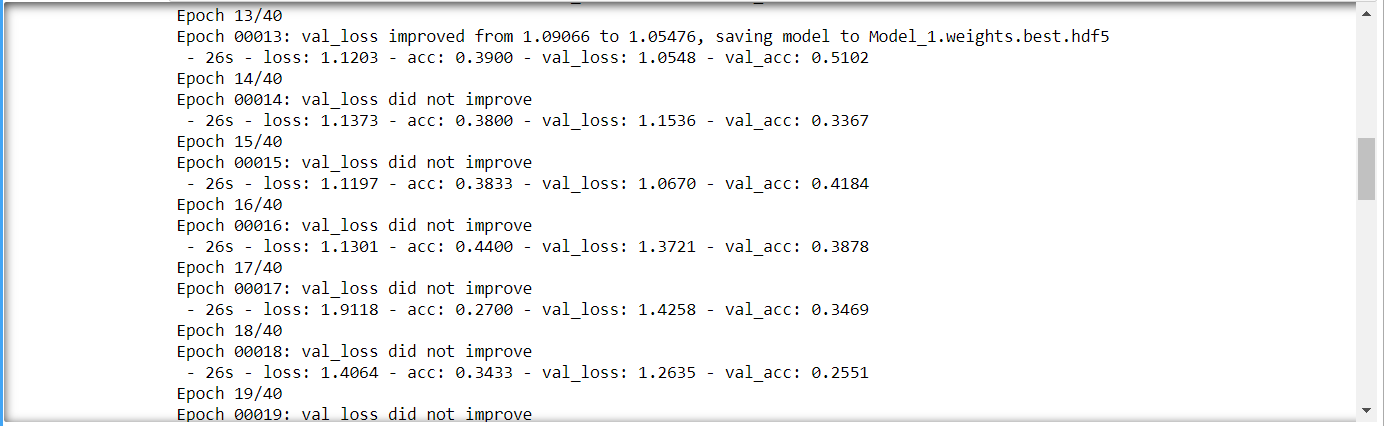


**TRAINING MODEL 1:**

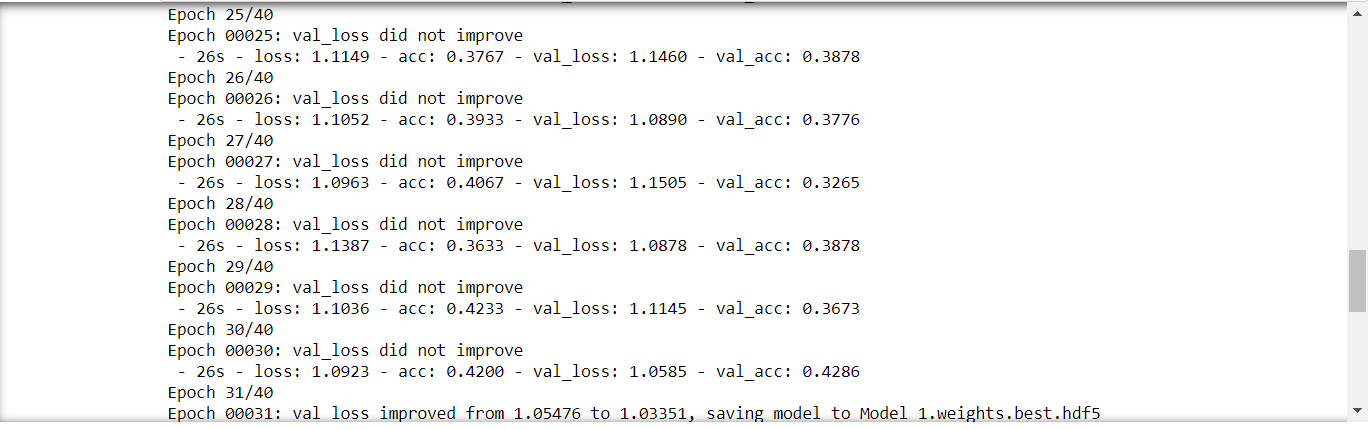
Here, the model (Model-1) is being trained on the training data. For each epoch (iteration), the model is being validated using the validation data. The model is trained for 40 epochs. Also, the model (model's weights) that performed the best on the validation set is being saved in a file (Model\_1.weights.best.hdf5).

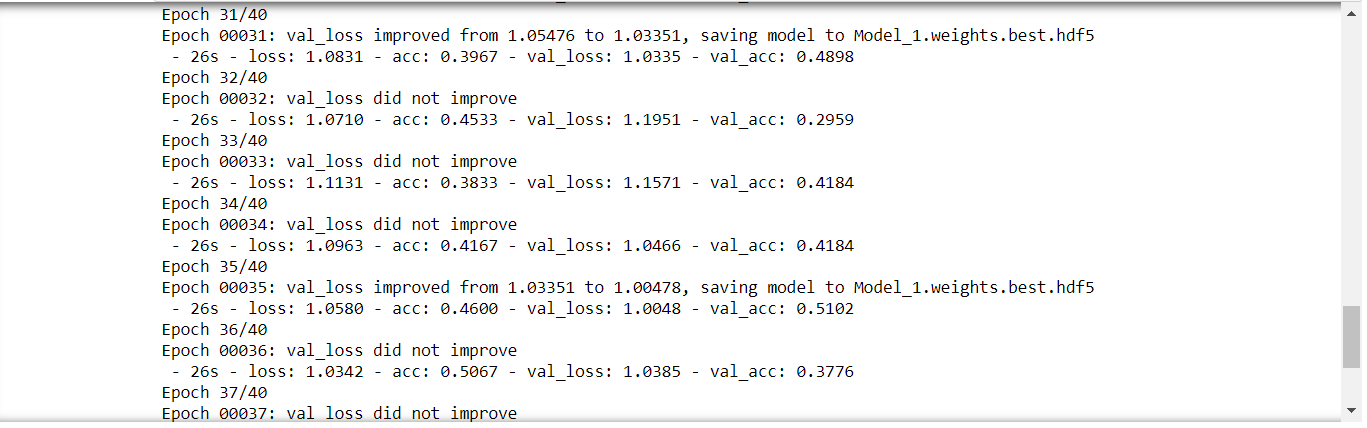


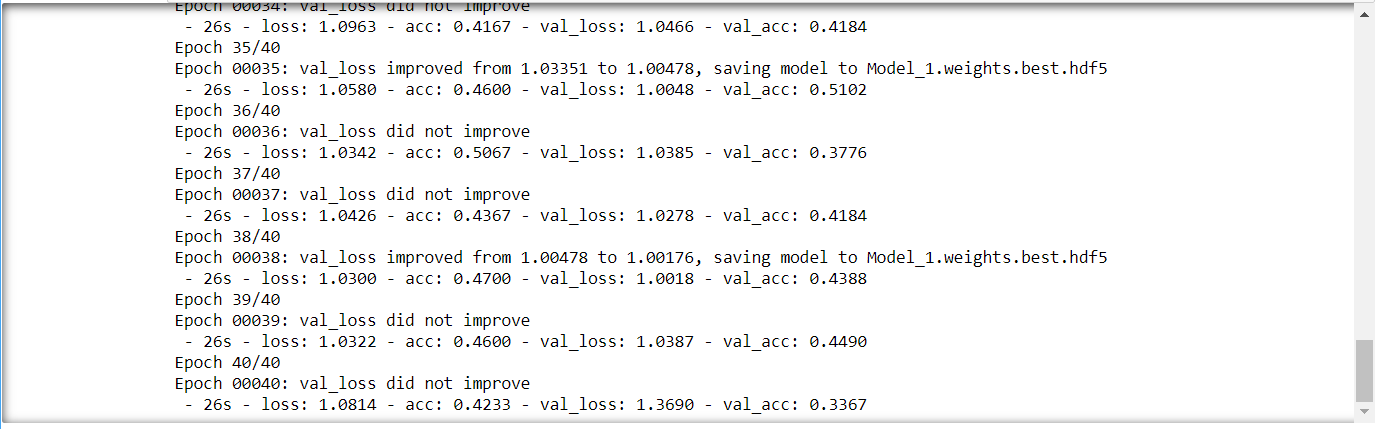






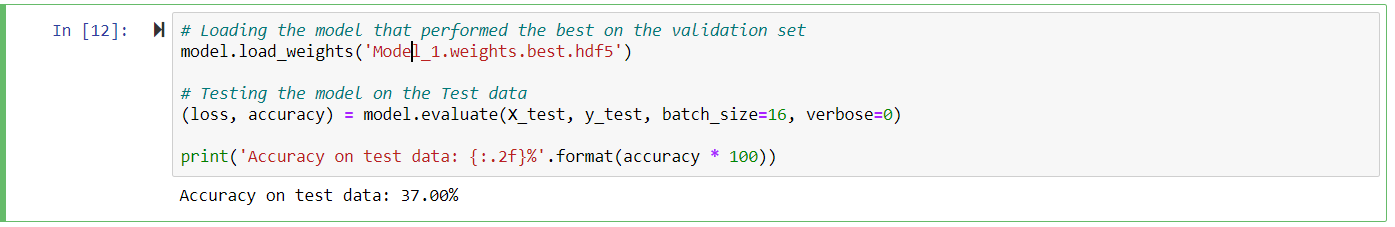




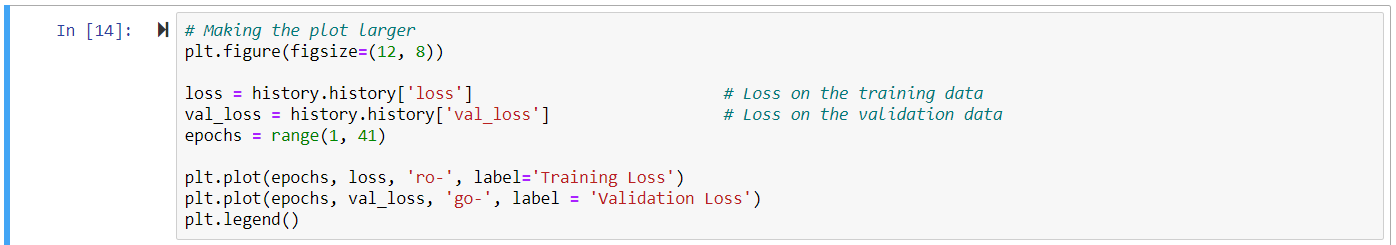


**EVALUATING MODEL 1:**

The best model weights are loaded and then the model is evaluated on the test data. I have used accuracy as the metric to test the model's performance on the test data.



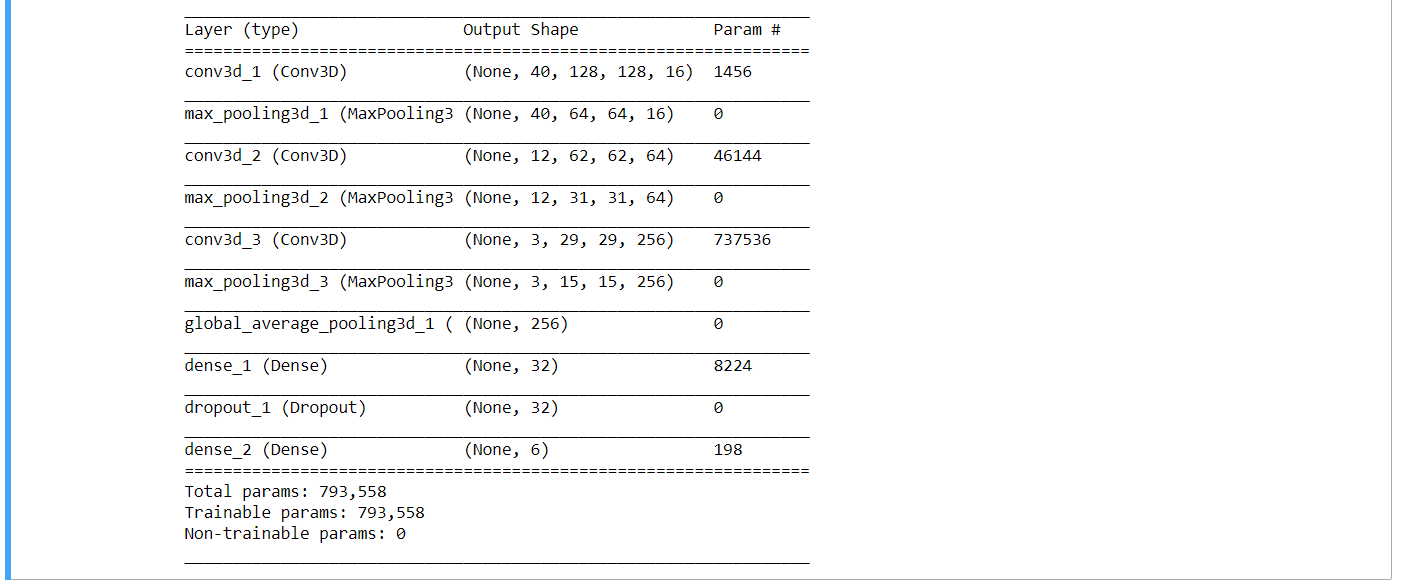
**TRAINING LOSS VS VALIDATION LOSS:**



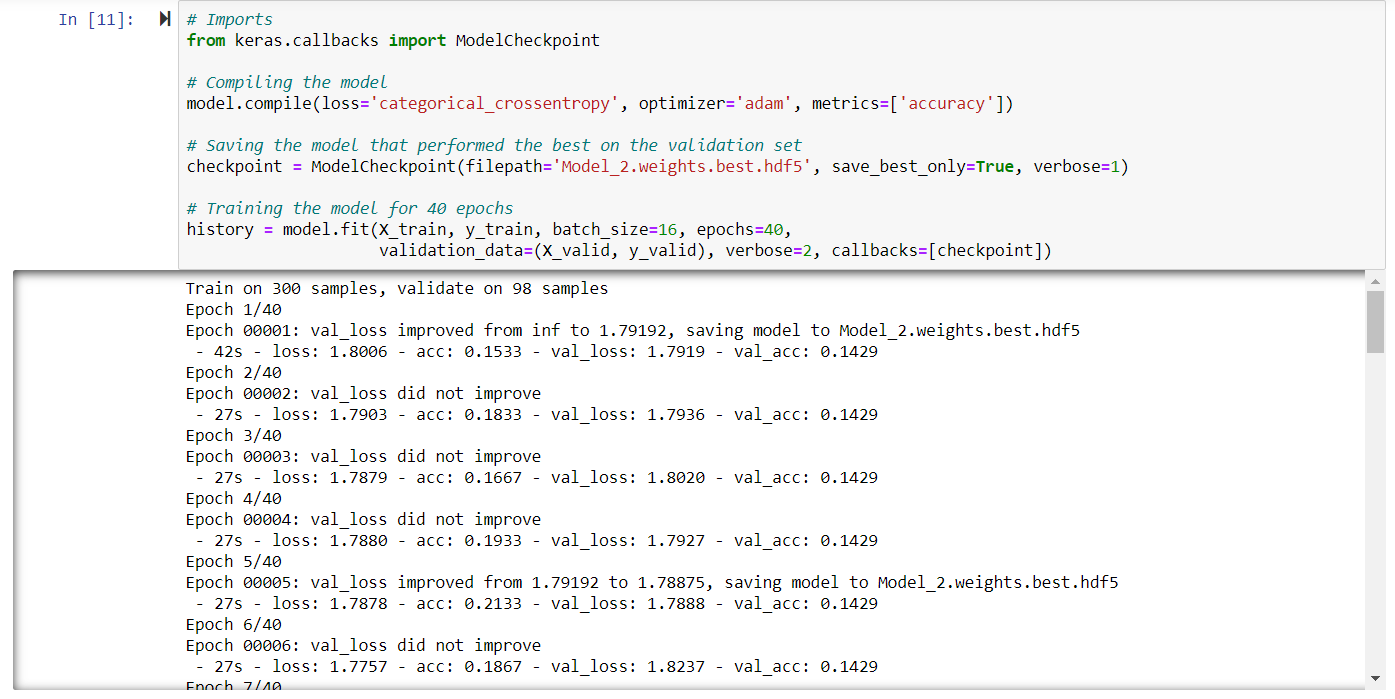


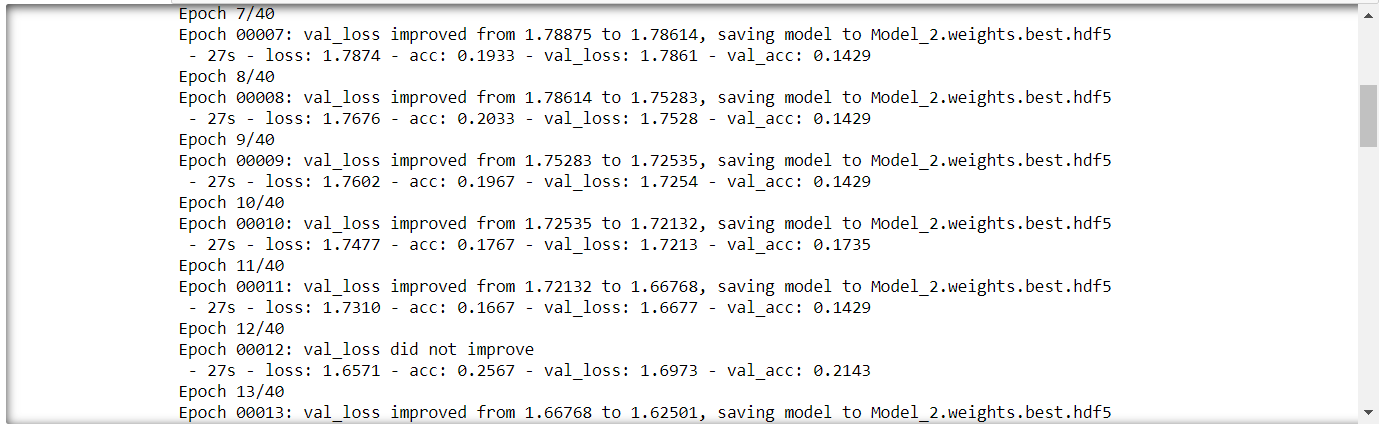
**MODEL 2:**

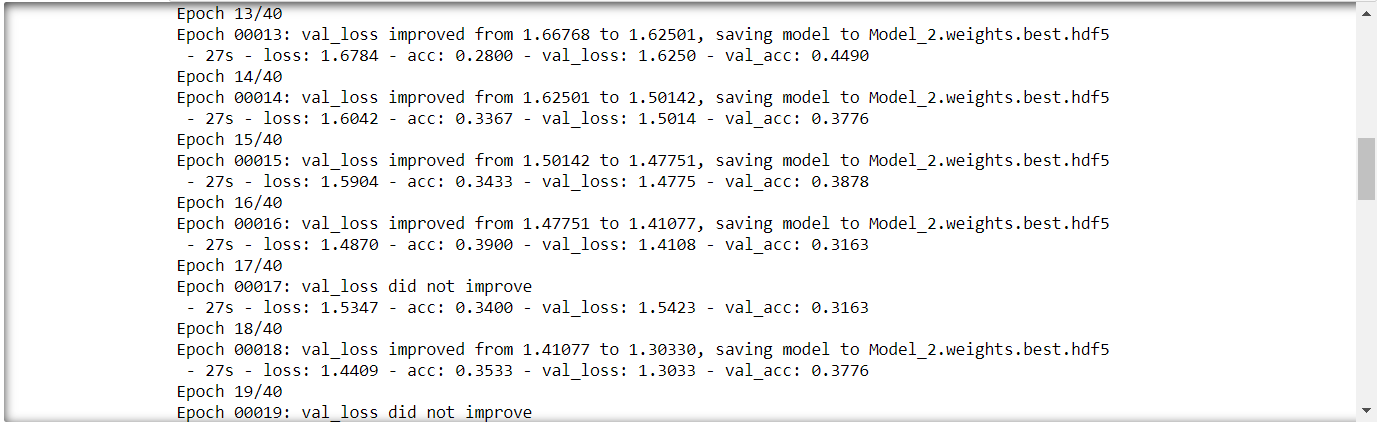


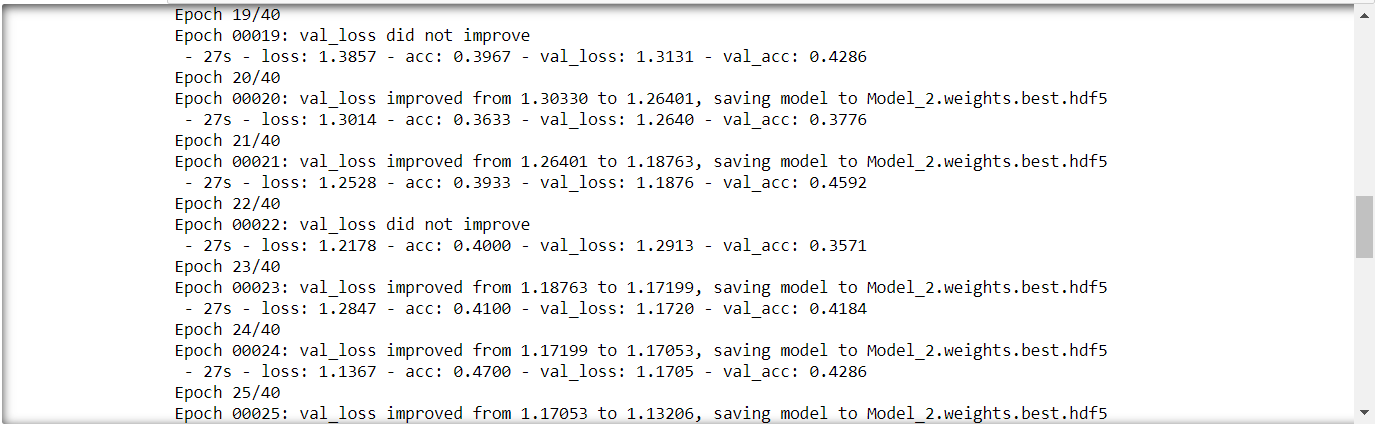


**TRAINING MODEL 2:**

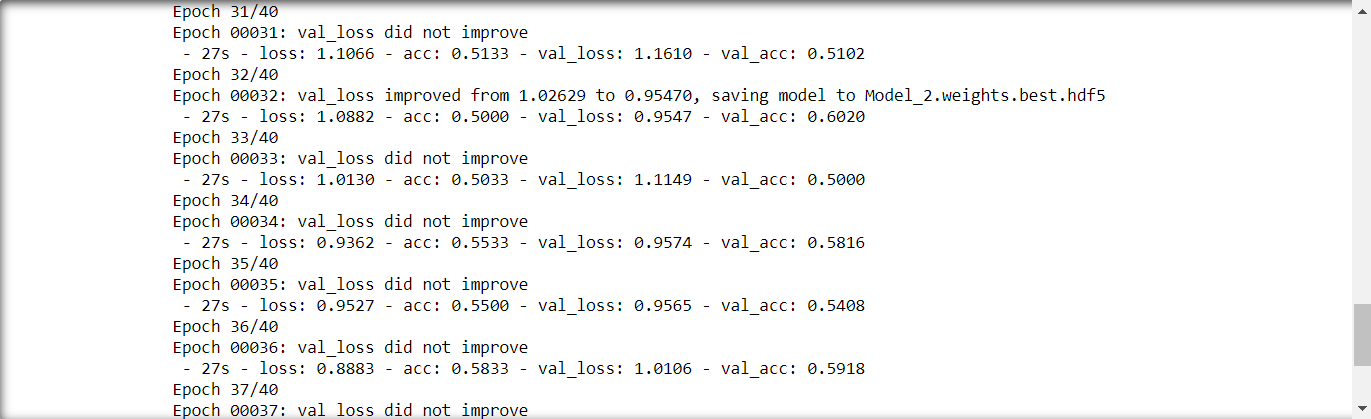


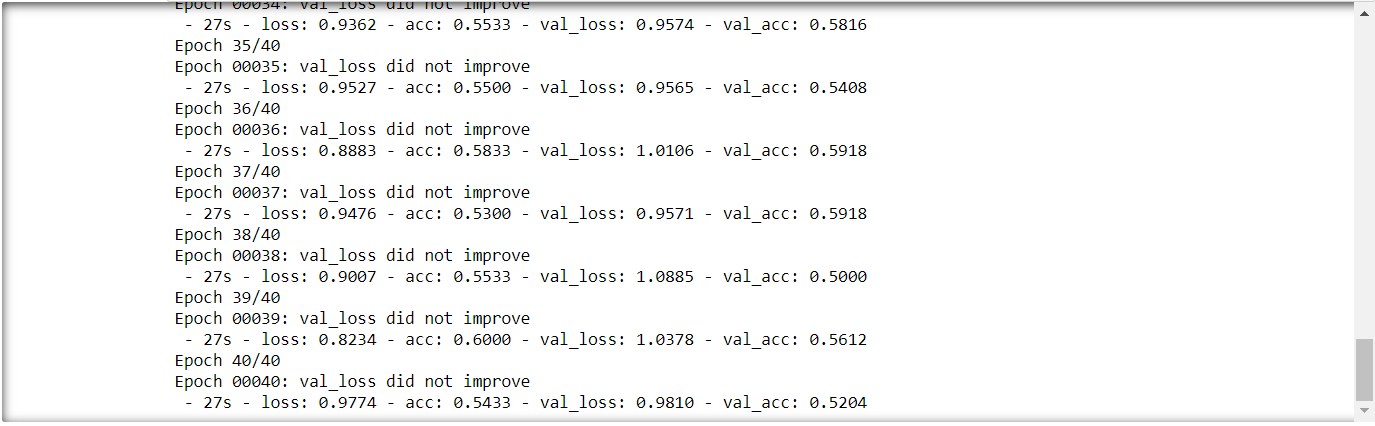




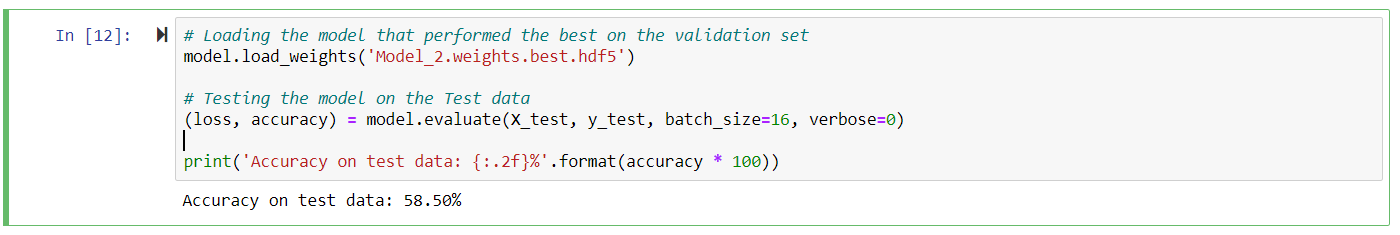




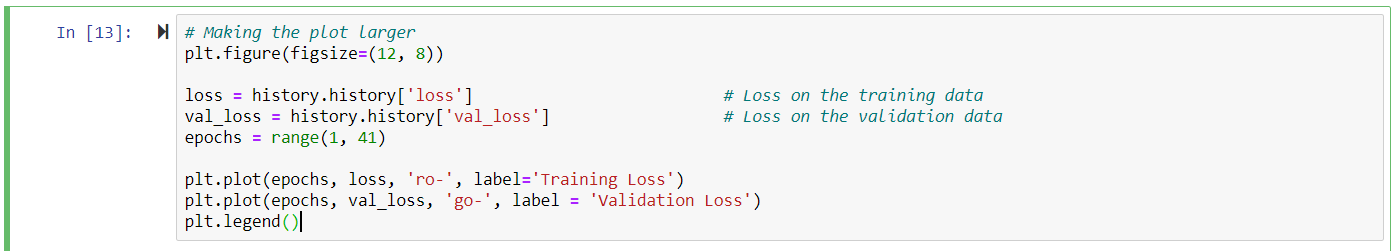


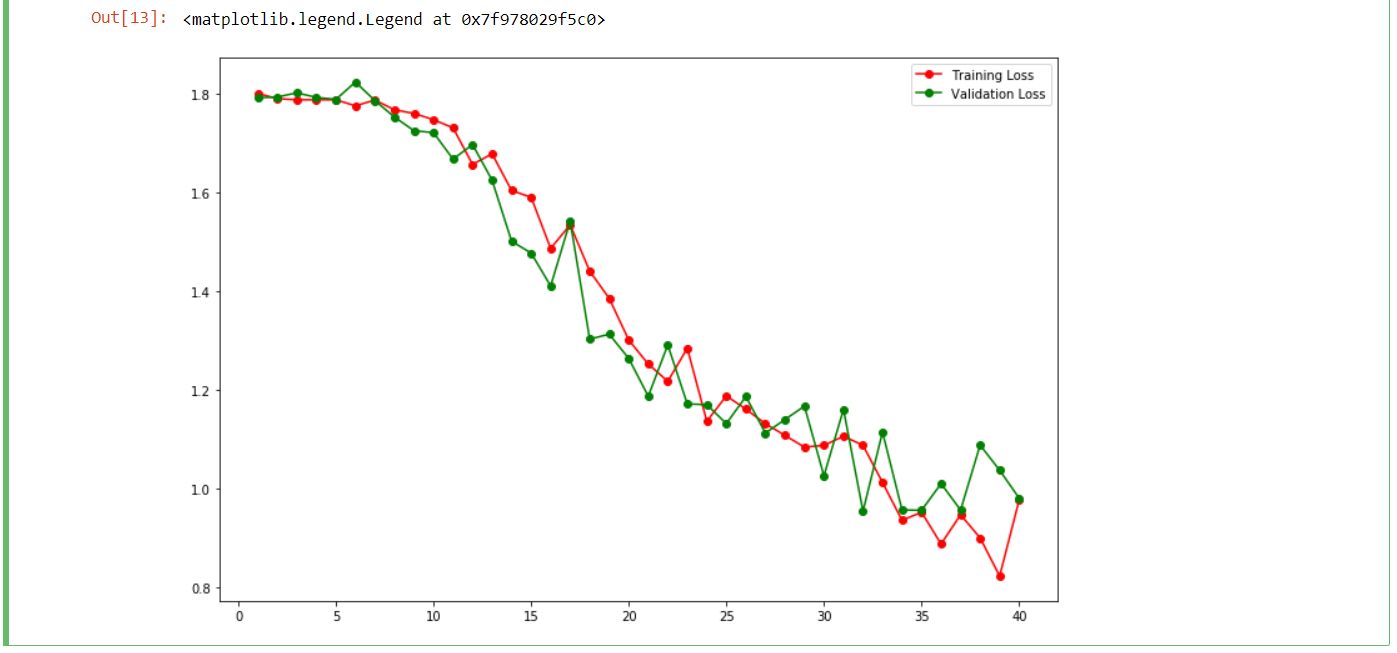


**EVALUATING MODEL 2:**



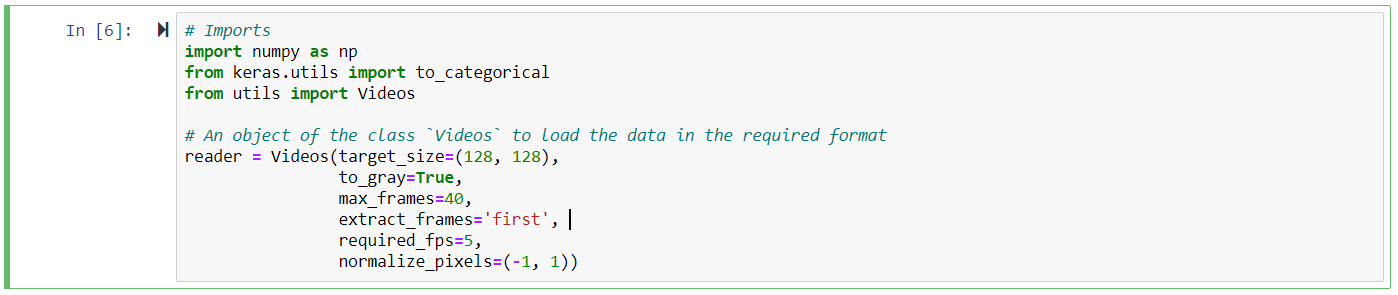
**LEARNING CURVE:**

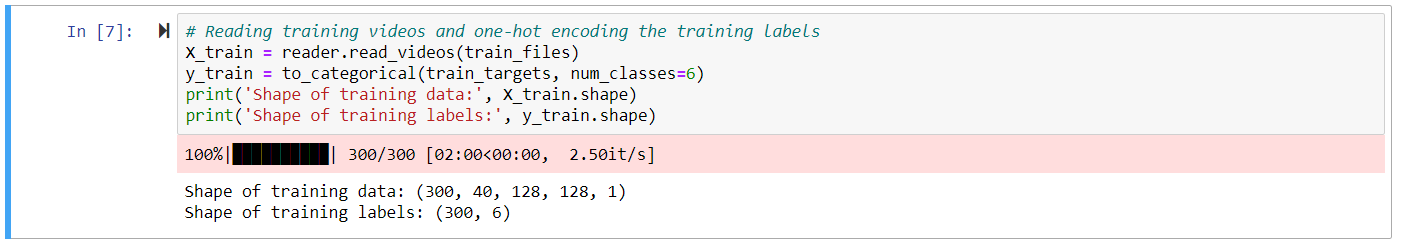


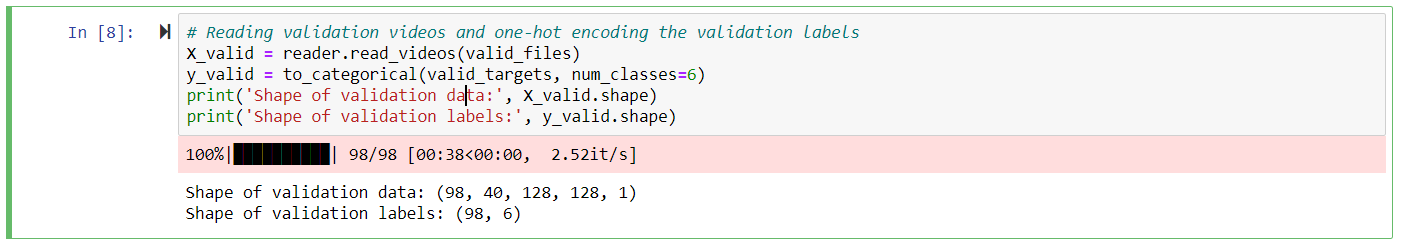


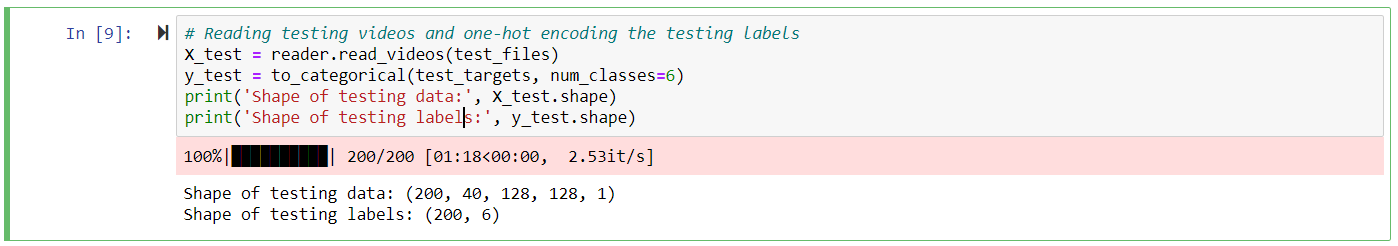
There is a gradual decrease in both the training and validation loss. Also, The difference between the training and validation loss is not very large, suggesting that our model is no longer overfitting the training data.

**FURTHER PROCESSING THE DATA:**



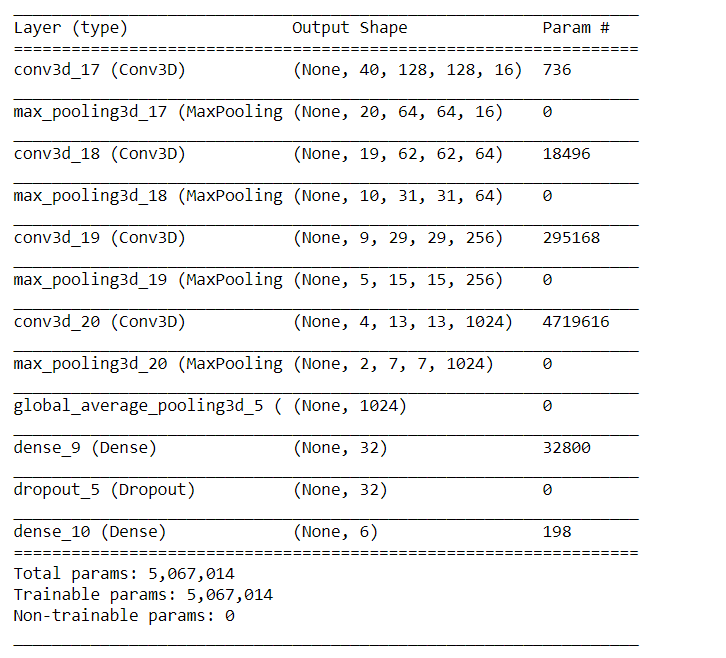




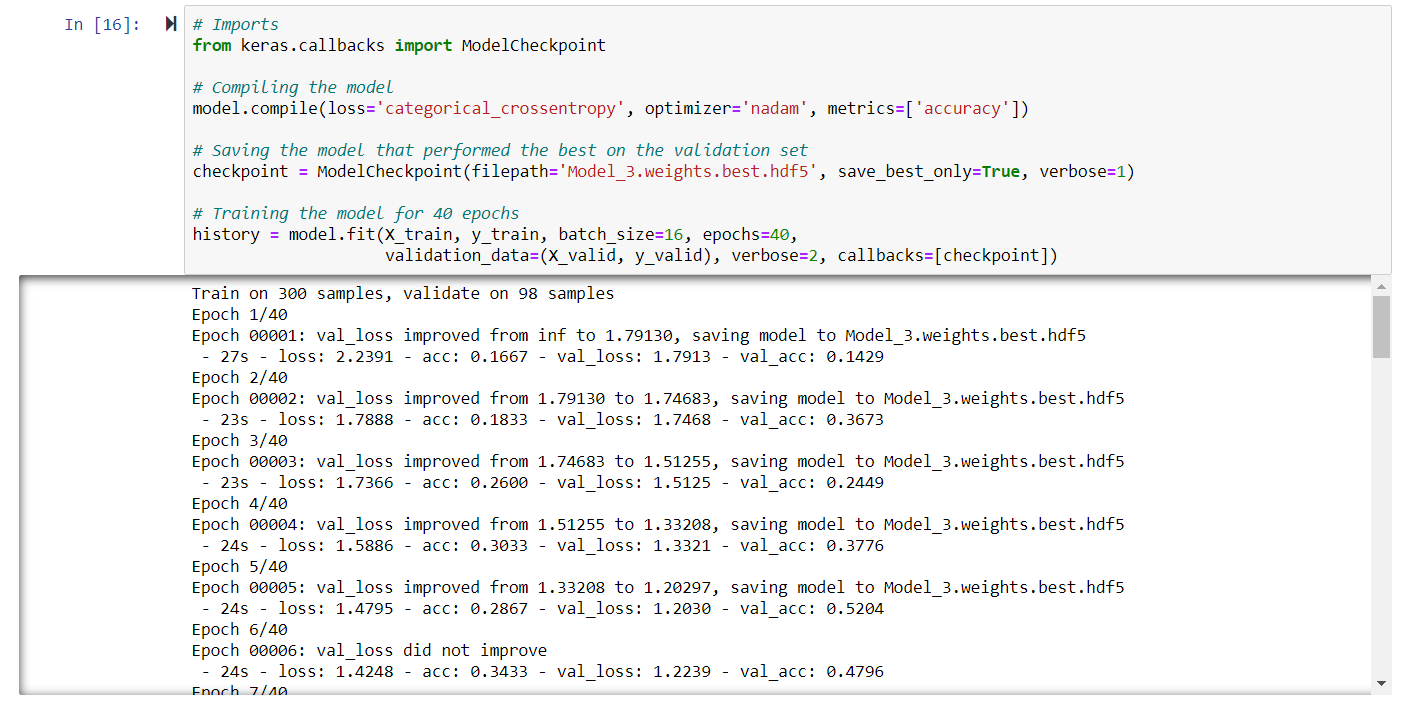


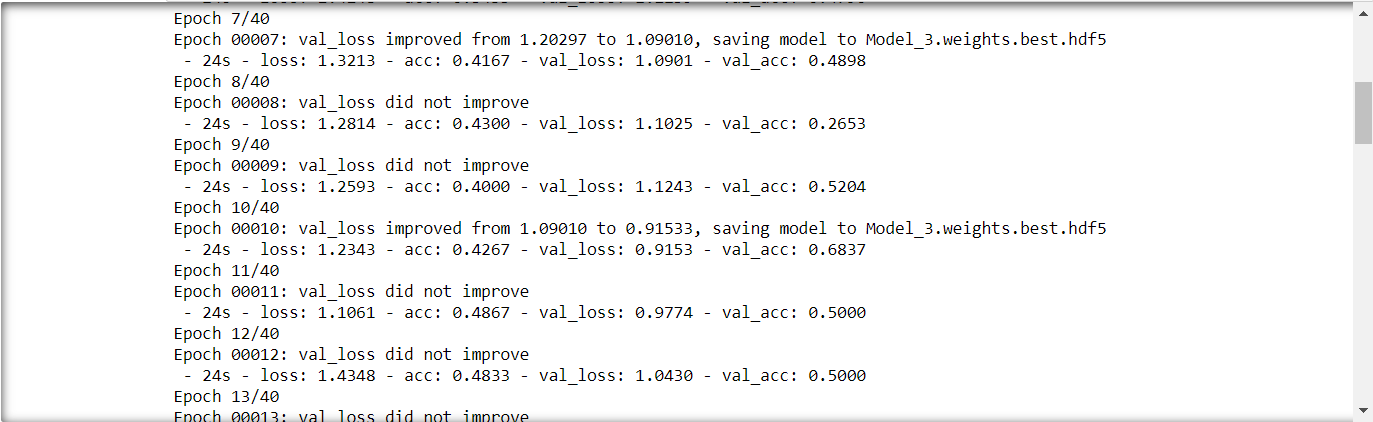
**MODEL 3:**

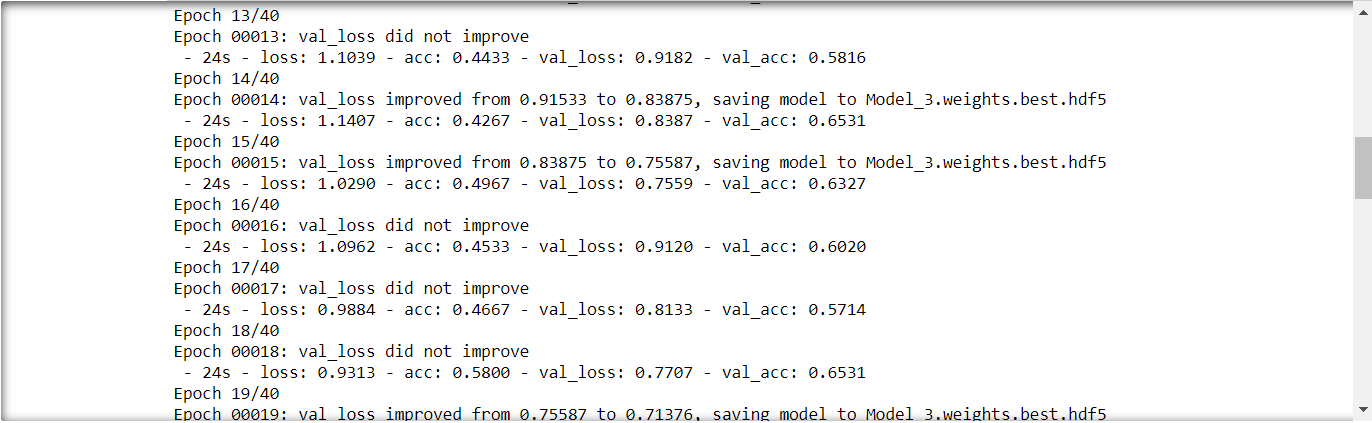


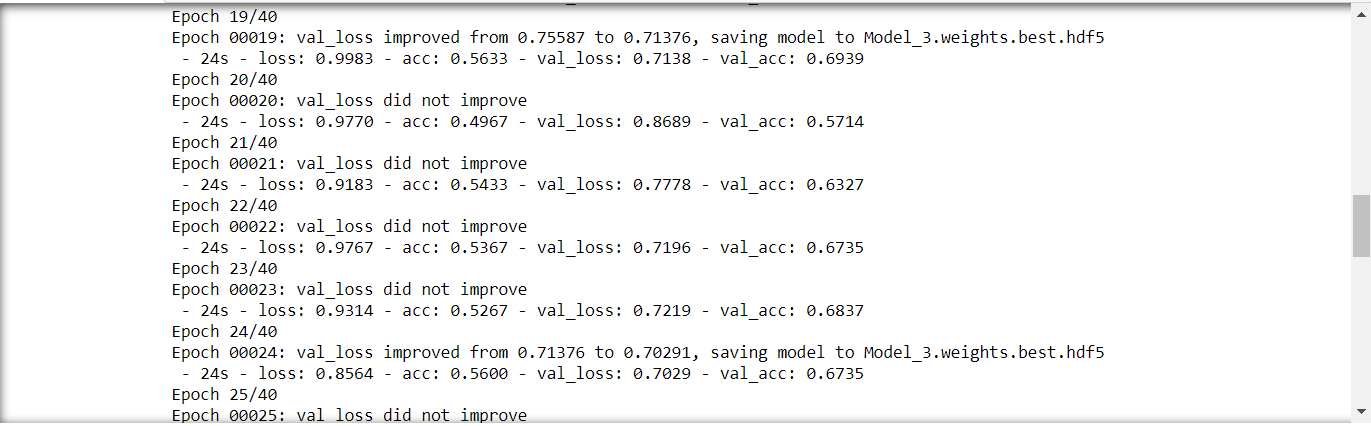


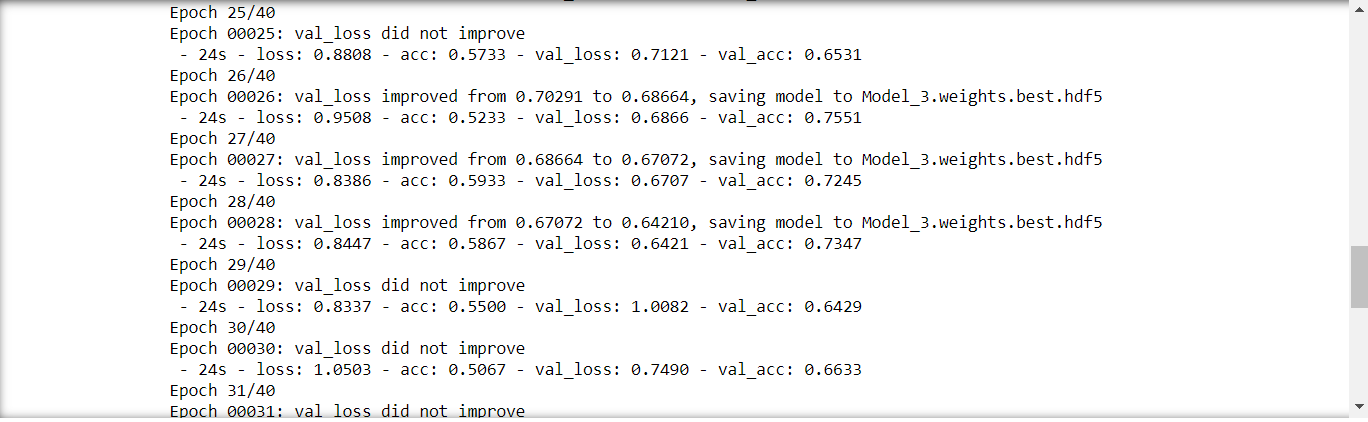
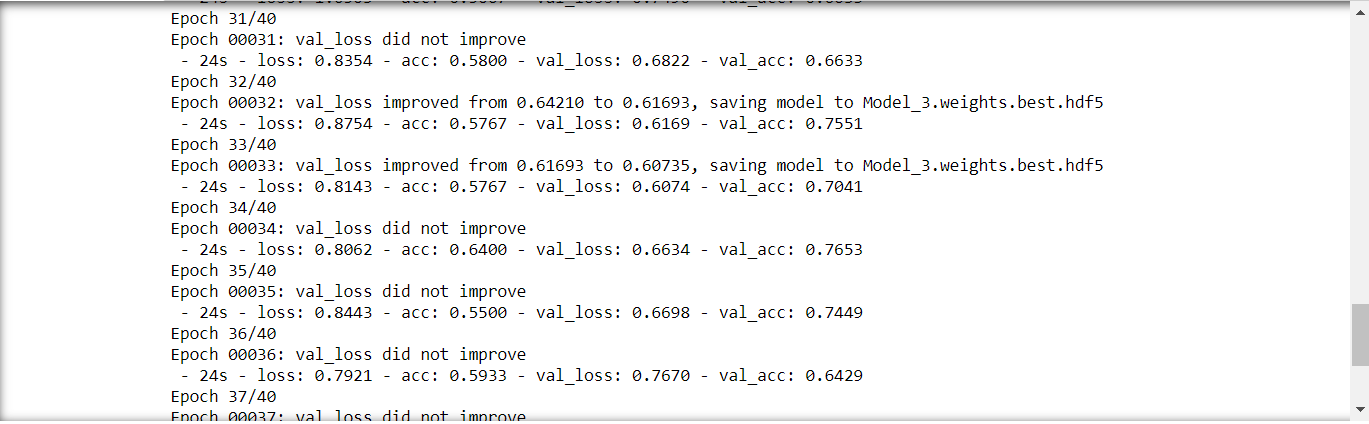
**TRAINING MODEL 3:**

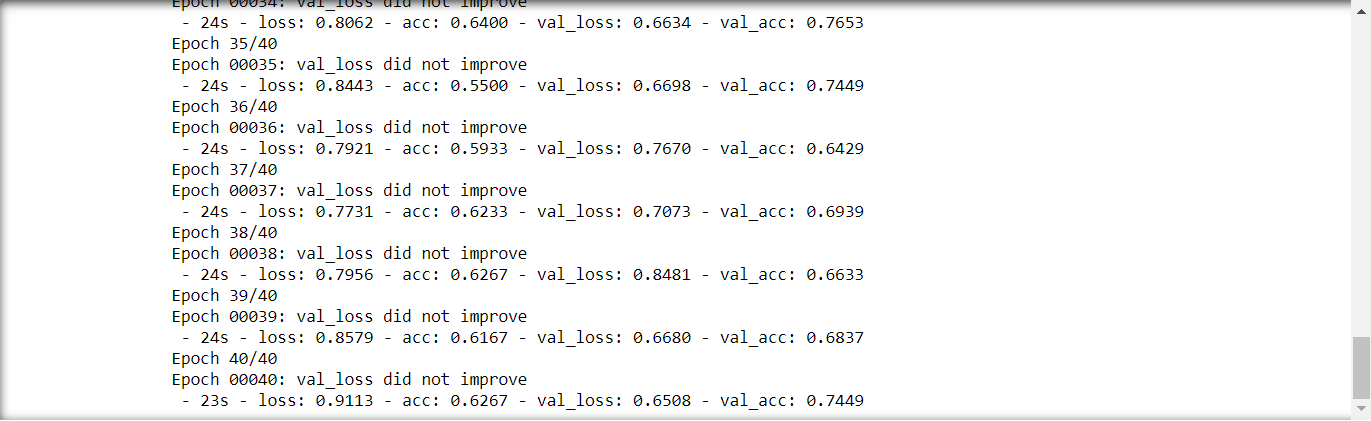




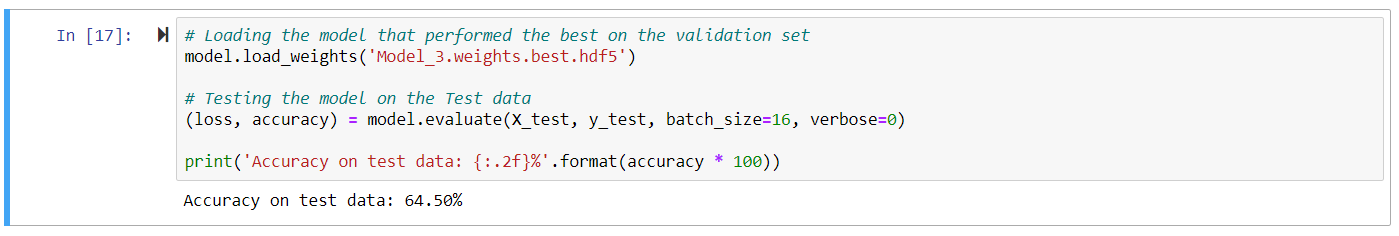




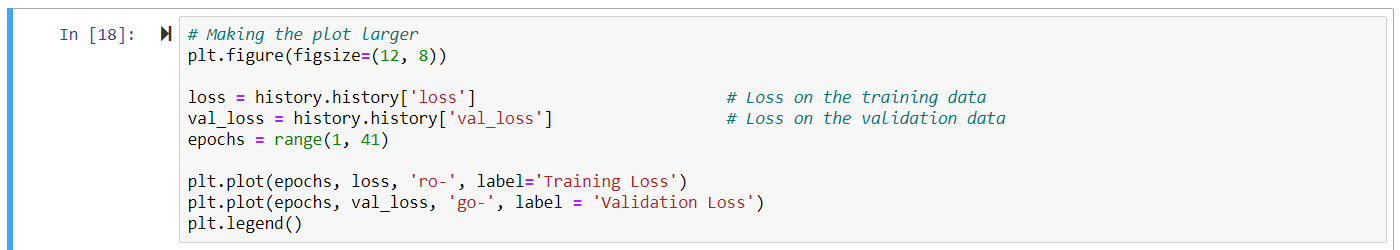
 

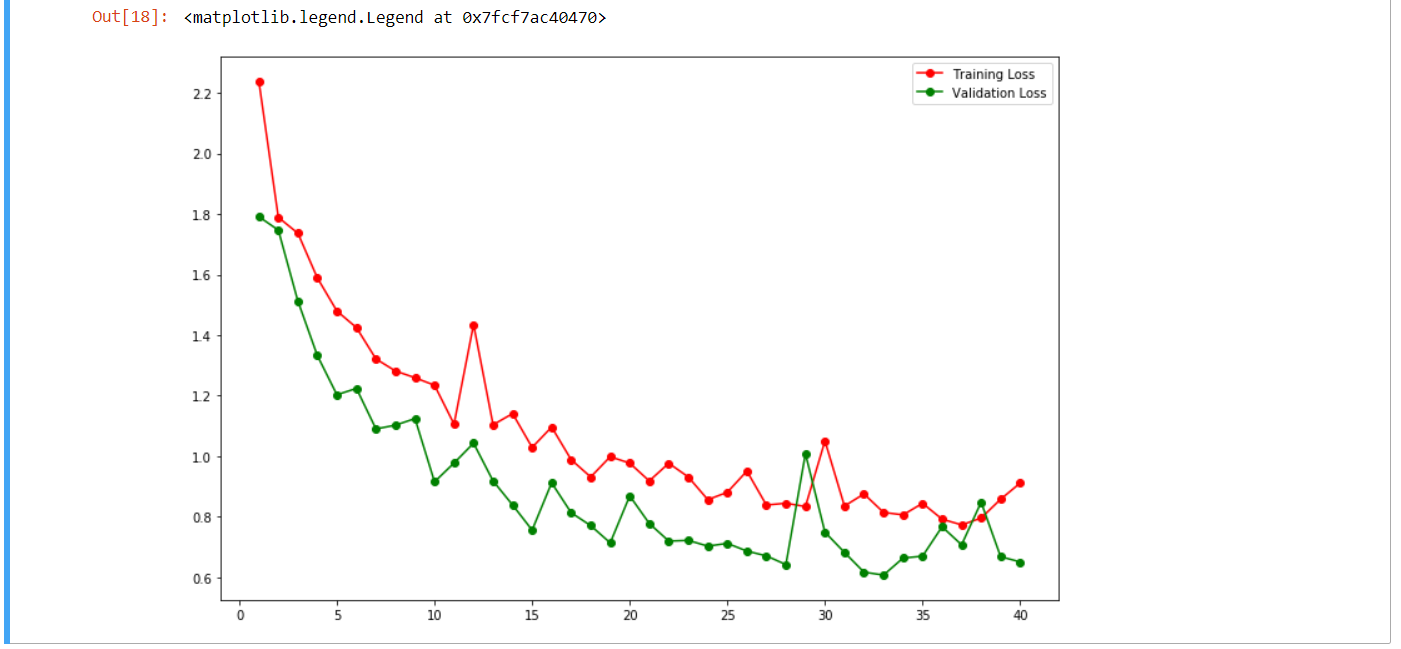


**EVALUATING MODEL 3:**

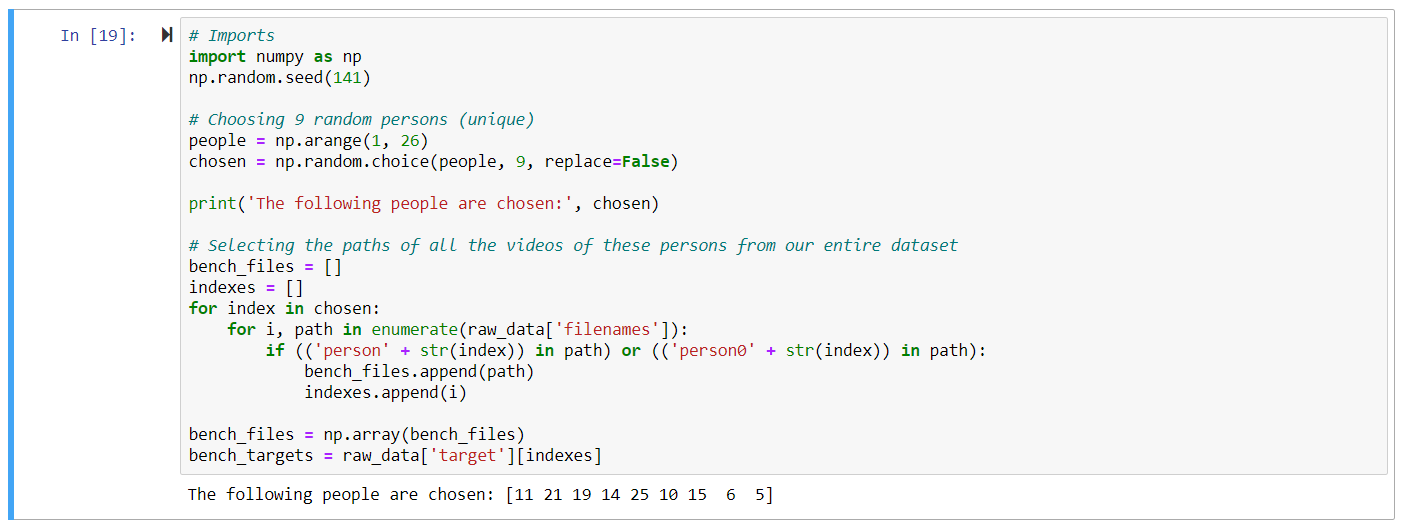


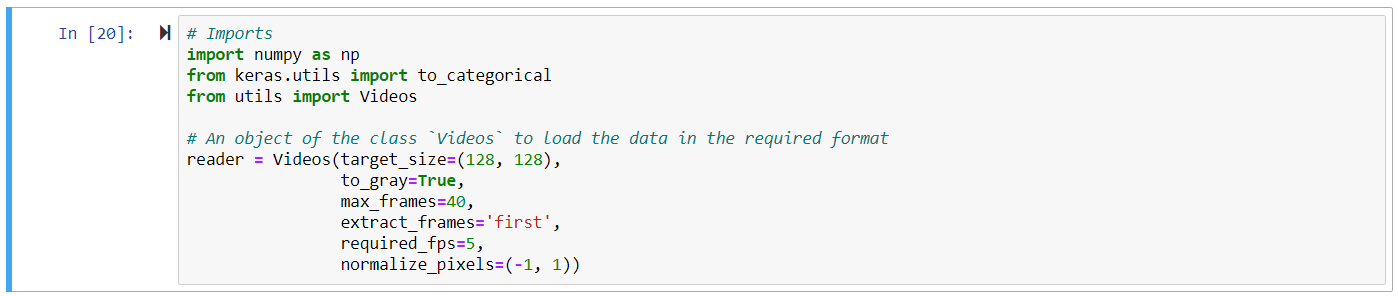
**LEARNING CURVE:**

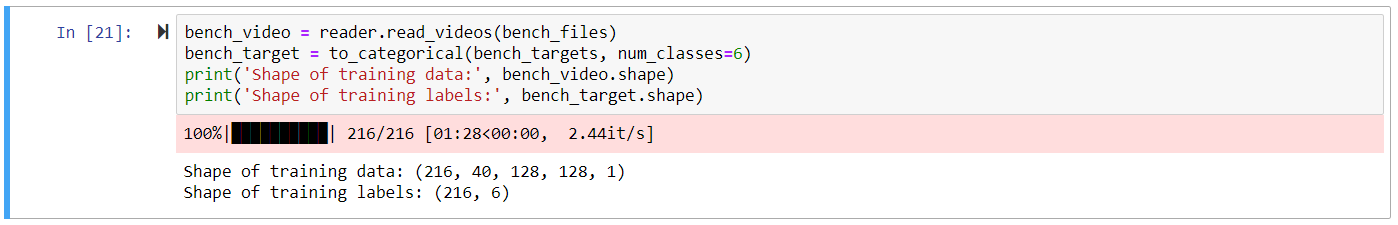


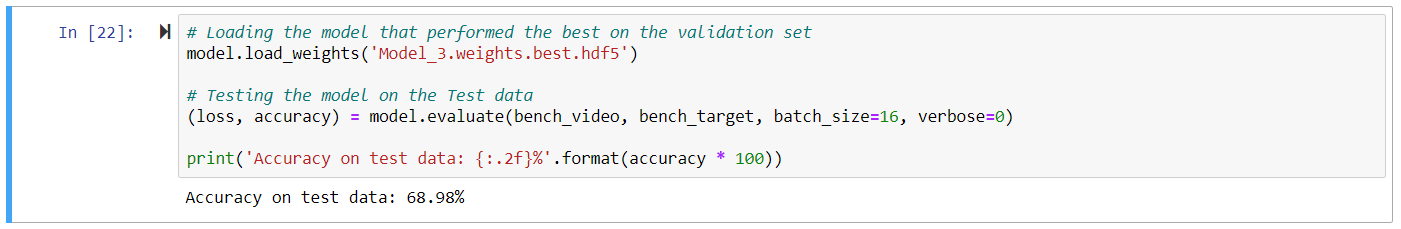


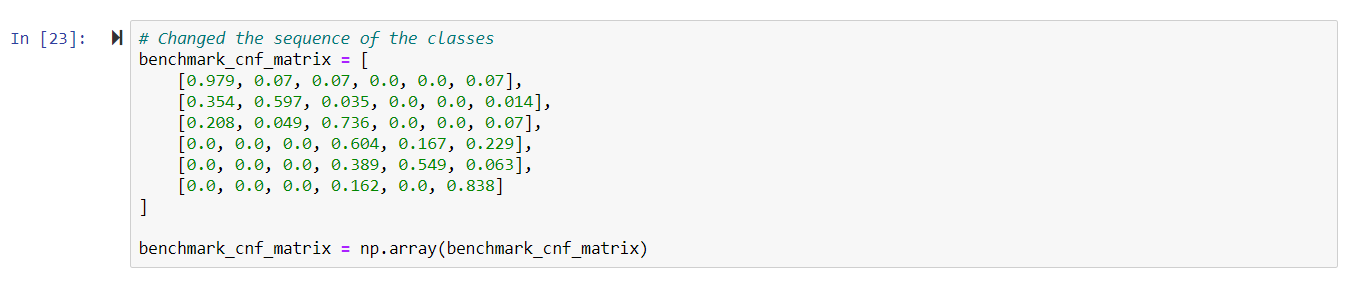
**BENCHMARK MODEL:**





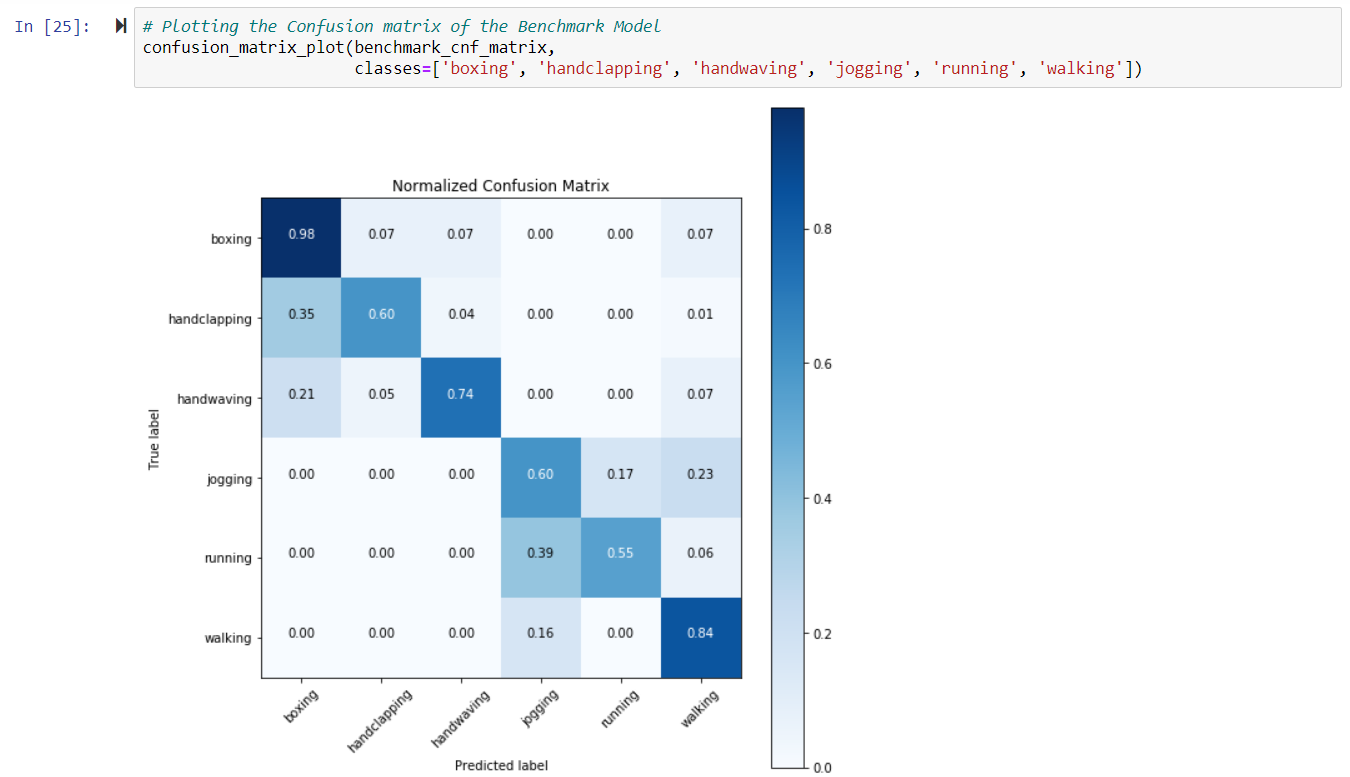




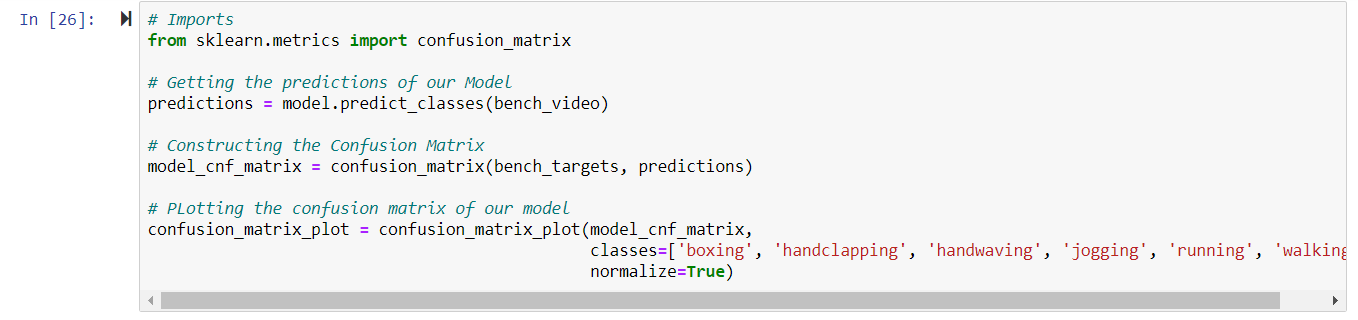


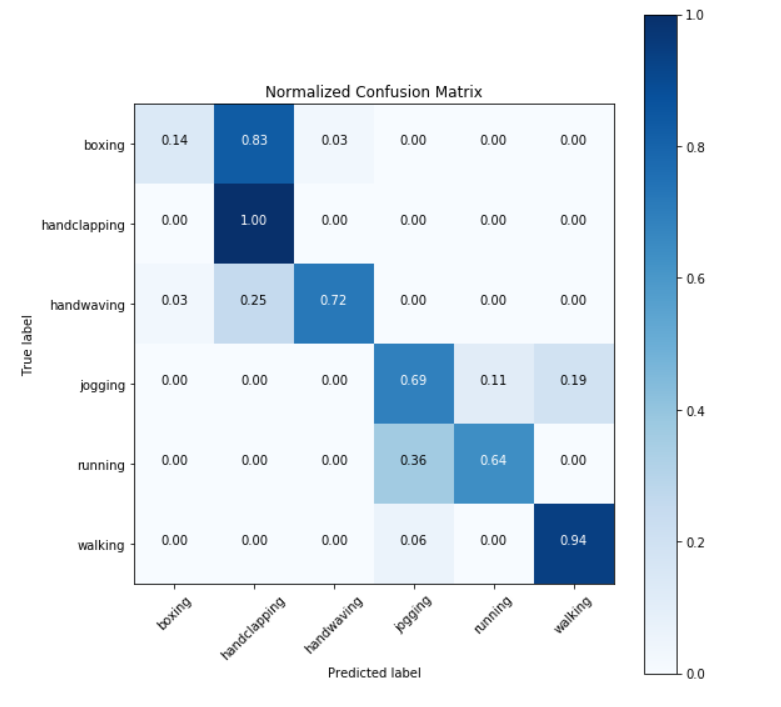


**CONFUSION MATRIX FOR BENCH MARK MODEL:**

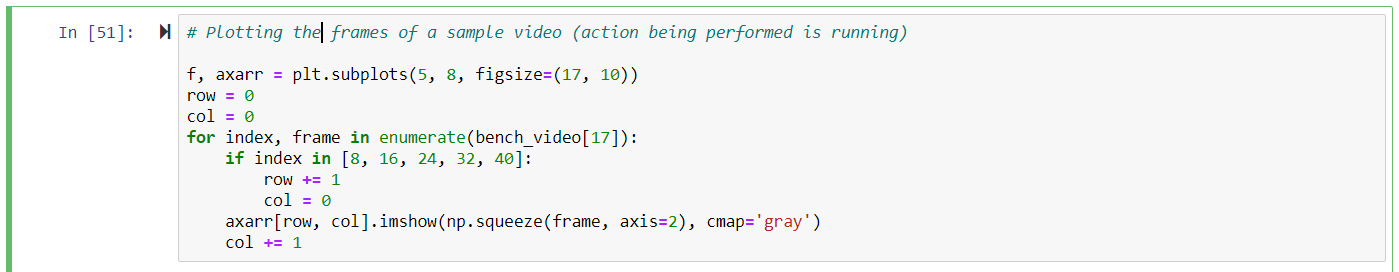


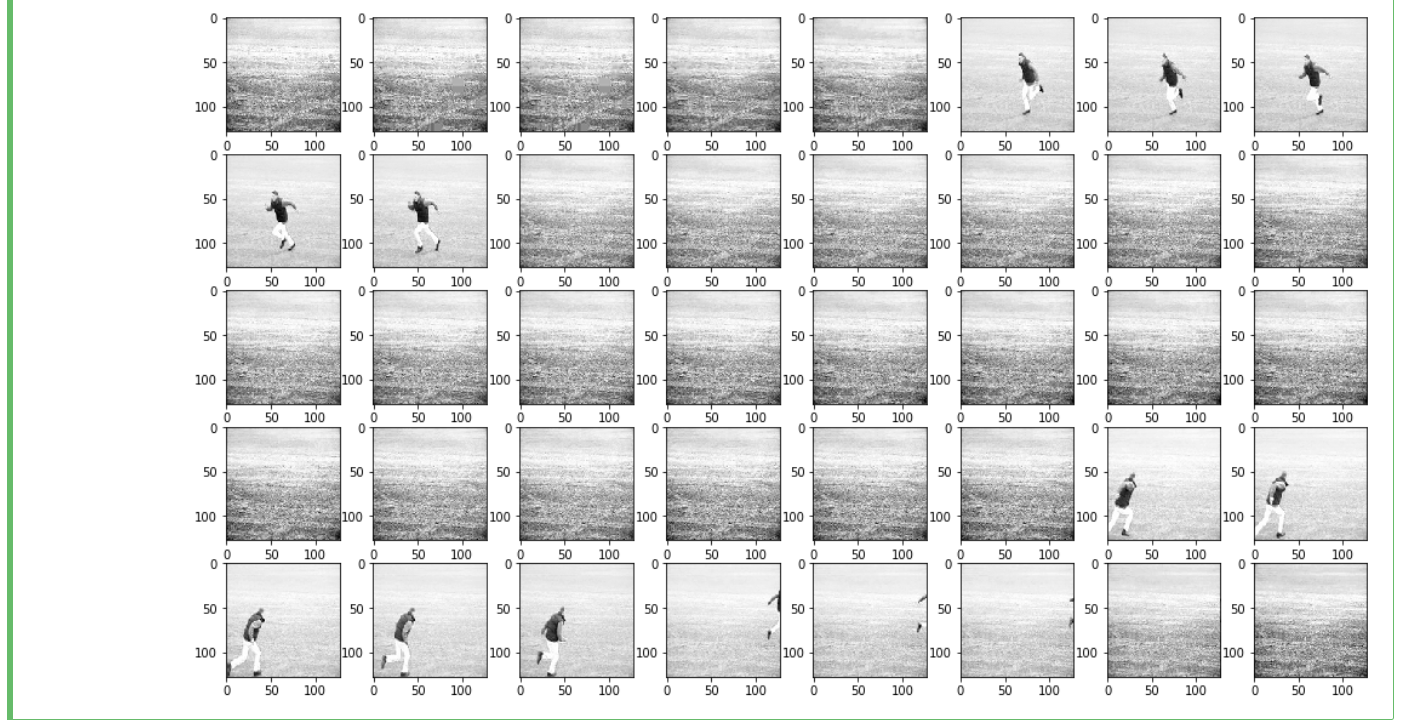
**CONFUSION MATRIX FOR MODEL 3:**



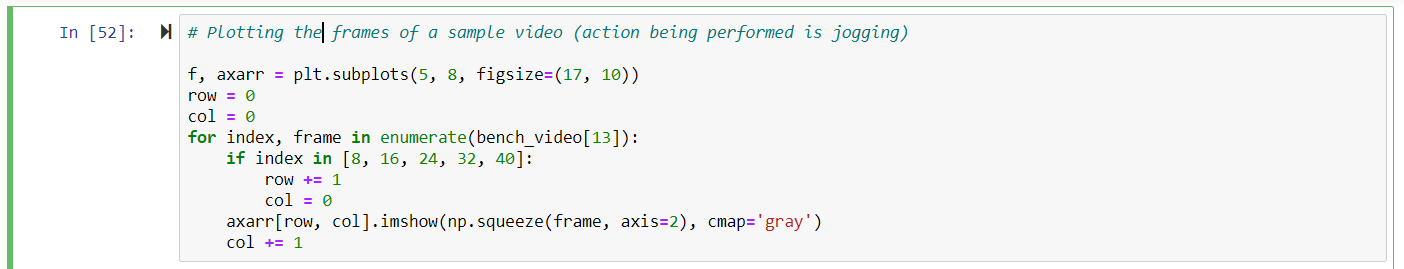


**SEQUENCE OF FRAMES OF RUNNING ACTION BEING DETECTED:**





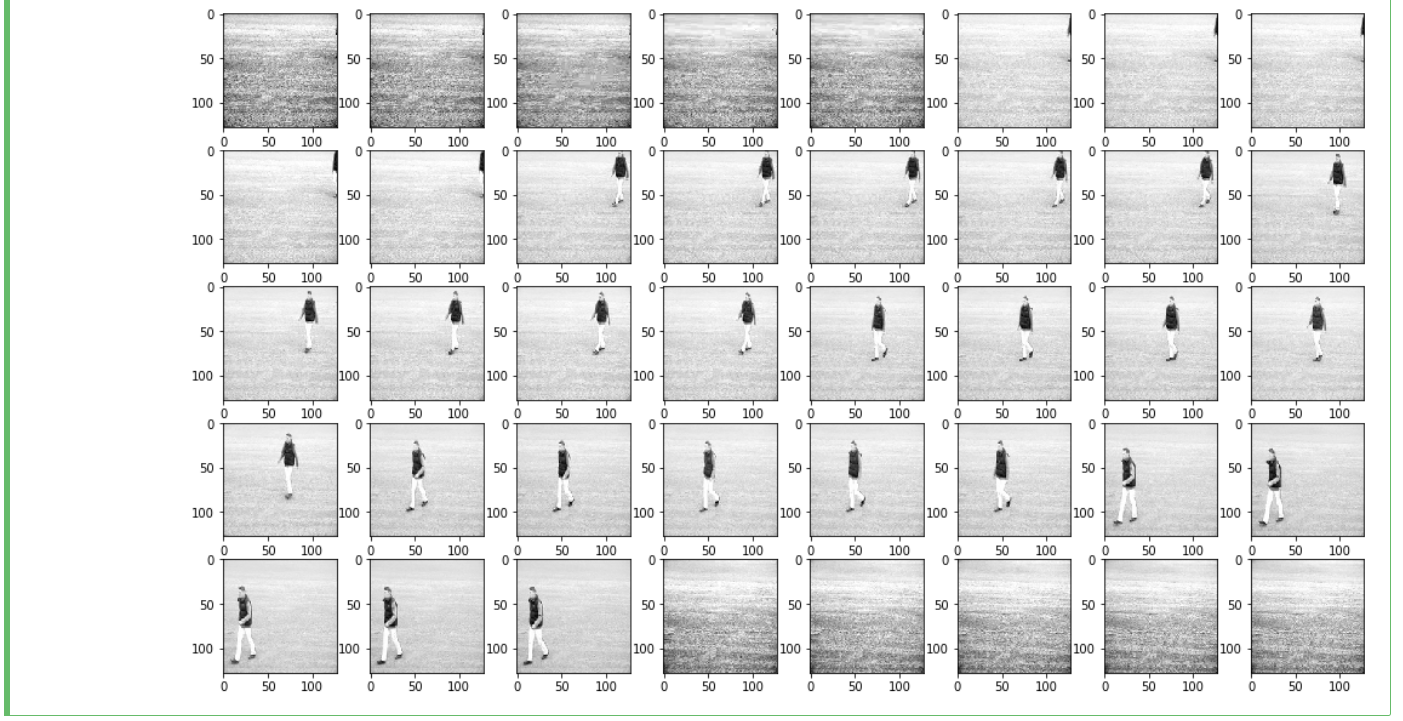
**SEQUENCE OF FRAMES OF JOGGING ACTION BEING DETECTED:**





**SEQUENCE OF FRAMES OF WALKING ACTION BEING DETECTED:**





**3.2 ANALYSIS:**

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.

Hence Model 3 is used for identification human postures.

**CHAPTER 4**

**CONCLUSION AND SCOPE FOR FUTURE WORK**

**4.1 CONCLUSION**

Human activity recognition remains to be an important problem in Neural networks. HAR is the basis for many applications such as video surveillance, health care, and human-computer interaction. Methodologies and technologies have made tremendous development in the past decades and have kept developing up to date. However, challenges still exist when facing realistic sceneries, in addition to the inherent intraclass variation and interclass similarity problem.

**4.2 FUTURE WORK**

1. 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.

1. 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.

**APPENDIX**

**CODE:**

**data\_utils.py**

import urllib

import os

from keras.utils.data\_utils import \_extract\_archive

from tqdm import tqdm

class TqdmUpTo(tqdm):

def update\_to(self, b=1, bsize=1, tsize=None):

"""

Parameters:

b (int): optional

Number of blocks transferred so far [default: 1]

bsize (int): optional

Size of each block (in tqdm units) [default: 1]

tsize (int): optional

Total size (in tqdm units)

If [default: None] remains unchanged

"""

if tsize is not None:

self.total = tsize

self.update(b \* bsize - self.n) # will also set self.n = b \* bsize

def download\_files():

"""

Downloads the 6 zip files in the current directory and extracts them within a directory called 'Data'

The format mentioned in the file `Directory Structure for Data.txt` is maintained.

If a file is already present, it is not downloaded again.

"""

base\_link = r'http://www.nada.kth.se/cvap/actions/'

base\_file = os.getcwd() + r'/'

files = ['walking', 'jogging', 'running',

'boxing', 'handwaving', 'handclapping']

for file in files:

link = base\_link + file + '.zip'

file\_name = base\_file + file + '.zip'

if not os.path.exists(file\_name):

with TqdmUpTo(unit='B', unit\_scale=True, miniters=1, desc=link.split('/')[-1]) as t:

urllib.request.urlretrieve(

link, file\_name, reporthook=t.update\_to, data=None)

success = \_extract\_archive(file\_name, path=r'Data/' + file)

if success:

print('-----------------------------{}------'.format('-' \* len(file)))

print('| Successfully extracted --> {}.zip |'.format(file))

print('-----------------------------{}------'.format('-' \* len(file)))

os.remove(file\_name)

else:

print('\nUnsuccessful extraction --> {}.zip\n'.format(file))

**utils.py**

import numpy as np

from skvideo.io import FFmpegReader, ffprobe

from skvideo.utils import rgb2gray

from PIL import Image

from keras.preprocessing import image

from tqdm import tqdm

class Videos(object):

def \_\_init\_\_(self, target\_size=None, to\_gray=True, max\_frames=None,

extract\_frames='middle', required\_fps=None,

normalize\_pixels=None):

"""

Initializing the config variables

Parameters:

target\_size (tuple): (New\_Width, New\_Height), Default 'None'

A tuple denoting the target width and height of each frame in each of the video

to\_gray (boolean): Default 'True'

If True, then each frame will be converted to gray scale. Otherwise, not.

max\_frames (int): Default 'None'

The maximum number of frames to return for each video.

Extra frames are removed based on the value of 'extract\_frames'.

extract\_frames (str): {'first', 'middle', 'last'}, Default 'middle'

'first': Extract the first 'N' frames

'last': Extract the last 'N' frames

'middle': Extract 'N' frames from the middle

Remove ((total\_frames - max\_frames) // 2) frames from the beginning as well as the end

required\_fps (int): Default 'None'

Capture 'N' frame(s) per second from the video.

Only the first 'N' frame(s) for each second in the video are captured.

normalize\_pixels (tuple/str): Default 'None'

If 'None', the pixels will not be normalized.

If a tuple - (New\_min, New\_max) is passed, Min-max Normalization will be used.

If the value is 'z-score', then Z-score Normalization will be used.

For each pixel p, z\_score = (p - mean) / std

"""

self.target\_size = target\_size

self.to\_gray = to\_gray

self.max\_frames = max\_frames

self.extract\_frames = extract\_frames

self.required\_fps = required\_fps

self.normalize\_pixels = normalize\_pixels

self.fps = None

def read\_videos(self, paths):

"""

Parameters:

paths (list): Required

A list of paths of the videos to be read

Returns:

Numpy.ndarray

A 5-d tensor with shape (<No. of Videos>, <No. of frames>, <height>, <width>, <channels>)

"""

list\_of\_videos = [

self.\_read\_video(path) for path in tqdm(paths)

]

tensor = np.vstack(list\_of\_videos)

if self.normalize\_pixels != None:

# Pixels are normalized for each video individually

if (type(self.normalize\_pixels) == tuple) and (len(self.normalize\_pixels) == 2):

base = self.normalize\_pixels[0]

r = self.normalize\_pixels[1] - base

min\_ = np.min(tensor, axis=(1, 2, 3), keepdims=True)

max\_ = np.max(tensor, axis=(1, 2, 3), keepdims=True)

return ((tensor.astype('float32') - min\_) / (max\_ - min\_)) \* r + base

elif self.normalize\_pixels == 'z-score':

mean = np.mean(tensor, axis=(1, 2, 3), keepdims=True)

std = np.std(tensor, axis=(1, 2, 3), keepdims=True)

return (tensor.astype('float32') - mean) / std

else:

raise ValueError('Invalid value of \'normalize\_pixels\'')

return tensor

def get\_frame\_count(self, paths):

"""

Can be used to determine the value of `max\_frames`

Parameters:

paths (list): Required

A list of paths of the videos to be read

Returns:

dict (python dictionary)

For each video, the total number of frames in that video is stored in the dictionary.

"""

frame\_count = {}

for path in paths:

cap = FFmpegReader(filename=path)

frame\_count[path] = cap.inputframenum

cap.close()

return frame\_count

def \_read\_video(self, path):

"""

Parameters:

path (str): Required

Path of the video to be read

Returns:

Numpy.ndarray

A 5-d tensor with shape (1, <No. of frames>, <height>, <width>, <channels>)

"""

cap = FFmpegReader(filename=path)

list\_of\_frames = []

self.fps = int(cap.inputfps) # Frame Rate

for index, frame in enumerate(cap.nextFrame()):

capture\_frame = True

if self.required\_fps != None:

is\_valid = range(self.required\_fps)

capture\_frame = (index % self.fps) in is\_valid

if capture\_frame:

if self.target\_size is not None:

temp\_image = image.array\_to\_img(frame)

frame = image.img\_to\_array(

temp\_image.resize(

self.target\_size,

Image.ANTIALIAS)).astype('uint8')

# Shape of each frame -> (<height>, <width>, 3)

list\_of\_frames.append(frame)

temp\_video = np.stack(list\_of\_frames)

cap.close()

if self.to\_gray:

temp\_video = rgb2gray(temp\_video)

if self.max\_frames is not None:

temp\_video = self.\_process\_video(video=temp\_video)

return np.expand\_dims(temp\_video, axis=0)

def \_process\_video(self, video):

"""

Parameters:

video (Numpy.ndarray):

Shape = (<No. of frames>, <height>, <width>, <channels>)

Video whose frames are to be extracted

Returns:

Numpy.ndarray

A tensor (processed video) with shape (<`max\_frames`>, <height>, <width>, <channels>)

"""

total\_frames = video.shape[0]

if self.max\_frames <= total\_frames:

if self.extract\_frames == 'first':

video = video[:self.max\_frames]

elif self.extract\_frames == 'last':

video = video[(total\_frames - self.max\_frames):]

elif self.extract\_frames == 'middle':

# No. of frames to remove from the front

front = ((total\_frames - self.max\_frames) // 2) + 1

video = video[front:(front + self.max\_frames)]

else:

raise ValueError('Invalid value of \'extract\_frames\'')

else:

raise IndexError(

'Required number of frames is greater than the total number of frames in the video')

return video

**REFERENCES:**

[1]<https://www.pyimagesearch.com/2019/11/25/human-activity-recognition-with-opencv-and-deep-learning/>

[2] <https://machinelearningmastery.com/how-to-load-and-explore-a-standard-human-activity-recognition-problem/>

[3] <https://medium.com/datadriveninvestor/a-guide-to-human-activity-recognition-f11e4637dc4e>

[4] <https://www.kaggle.com/machinoai/human-activity-recognition-using-lstm>

[5] <http://aqibsaeed.github.io/2016-11-04-human-activity-recognition-cnn/>

**BIODATA**

