**Report on Final Project**

On

**‘Build a Human action recognition model’**

Submitted by

**Team Leads –**

Sairaj Bhise

Kanishk Mehta

Prahaladh Chandrahasan

Guided by

**Yasin Shah (CEO)**

**Project Planning –**

***Phase I: Analysis Stage***

***Team Lead: Kanishk Mehta***

1. Configuring the deep learning environment for the project.

2. Exploring the Data

3. Understanding the data for analysis

4. Data Pre-processing and filtration

5. Data Extraction

***Phase II: Modelling and Evaluation Stage***

***Team Lead: Prahaladh Chandrahasan***

6. Data Preparation for the modelling.

7. Building the dynamic models.

8. Optimizing the model and Evaluation

9. Model testing and validation.

***Phase III: Project Deployment Phase***

***Team Lead: Sairaj Bhise***

10. Assembling the Model System

11. Preparing the deployment environment.

12. Deployment of the final Project.

**Project Overview**

Models are being made where in, 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.

**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.

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

**Metrics**

Once the model has been trained on the training data, its performance will be evaluated using the test 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 – <https://deepmind.com/research/open-source/kinetics>

→ The dataset has 700 labels where each of the label signifies a particular human action. There are approximately 6.5L youtube videos labelled accordingly and these videos are clipped to 10 seconds.

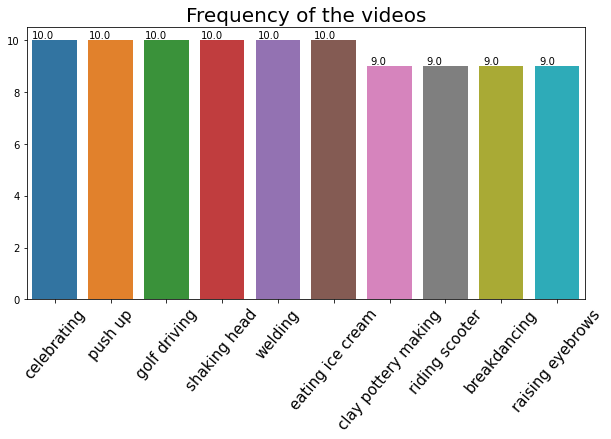
→ The final dataset contains 100 videos – 10 videos for each of the 10 categories after sampling.

→ The videos were captured at a frame rate of 5 fps and each frame was down-sampled to the resolution of 128 x 128 pixels.

→ While loading the data the categories are mapped with integer labels.

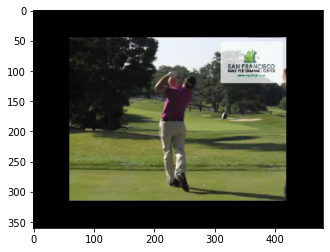
|  |  |
| --- | --- |
| **Categories/Labels** |  |
| Break Dancing | 0 |
| Celebrating | 1 |
| Clay Pottery Making | 2 |
| Eating the ice cream | 3 |
| Golf Driving | 4 |
| Push up | 5 |
| Raising the eyebrows | 6 |
| Riding Scooter | 7 |
| Shaking head | 8 |
| Welding | 9 |

Distribution of videos



**Exploratory Visualization**

Below is a single frame of a sample video of golf driving



It can be observed that the spatial dimensions of the video (width x height) are 360 x 480 pixels. Also, on loading a single video into a NumPy array in python, the shape of the array obtained was – (1, 20, 360, 480, 3)

This indicates that:

* There is 1 video
* The video has 20 frames
* The spatial dimension of the video is 360 x 480 (width x height) pixels
* Each frame has 3 channels – Red(R), Green(G) and Blue(B)

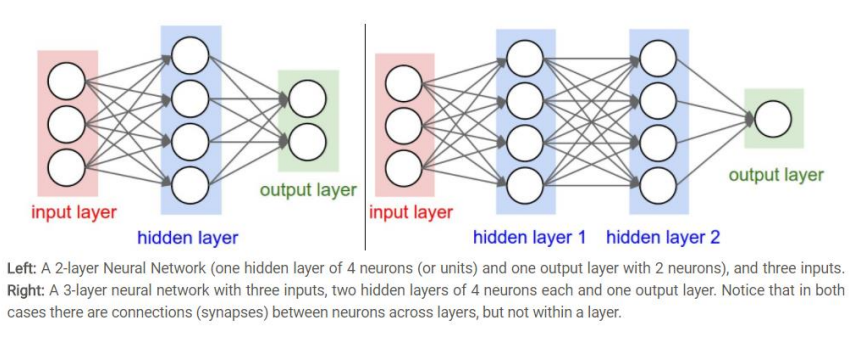
A similar methodology would be used for reading in the entire dataset.

**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.

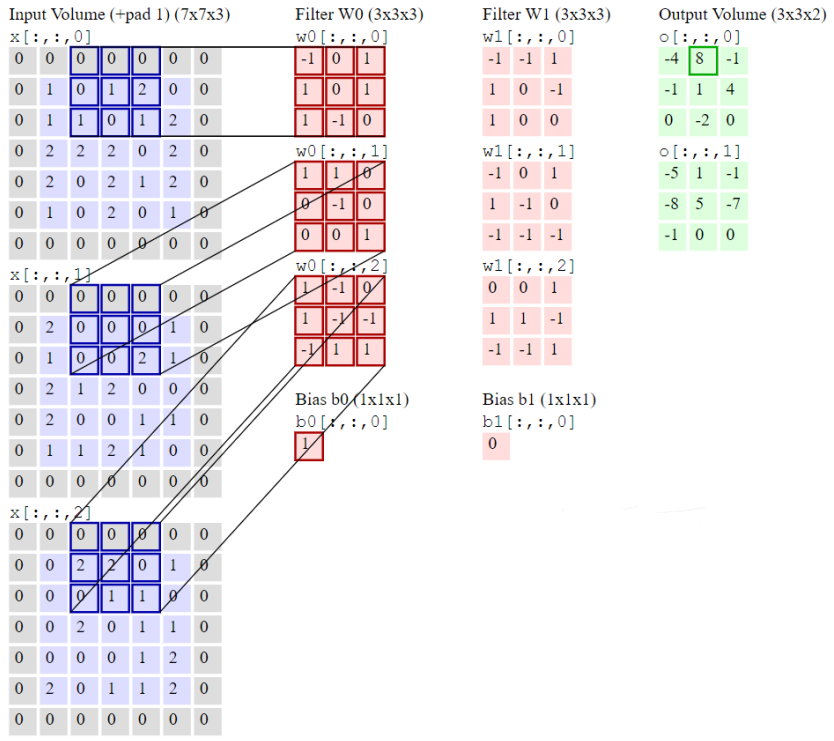
**Traditional Neural Networks**

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.



**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 feature 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.

A brief procedure:

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

Model Parameters:

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

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

**Methodology**

**Data pre-processing**

* Reading in the video frame-by-frame.
* The videos were captured at a frame rate of 5 fps. This means that for each second of the video, there will be 5 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 initial 20 frames (i.e., first 8 seconds of the video). Or we can use middle frames.
* 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.
* In the end, 15 frames at 5fps was extracted for the modelling as some of the videos gave the exception errors.
* 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 range 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:

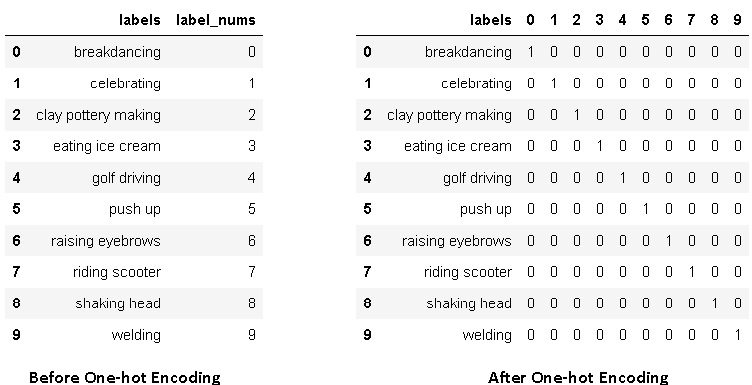
* Min-max Normalization – Get the values of the pixels in a given range (say 0 to 1)
* Z-score Normalization – This basically determines the number of standard deviations from the mean a data point is.

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.

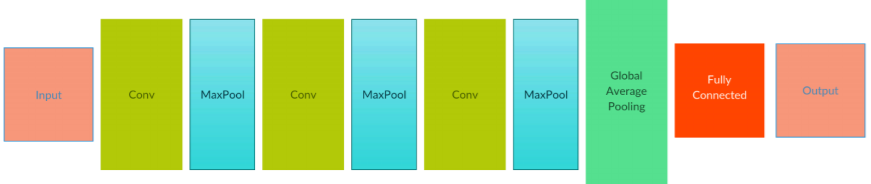
****

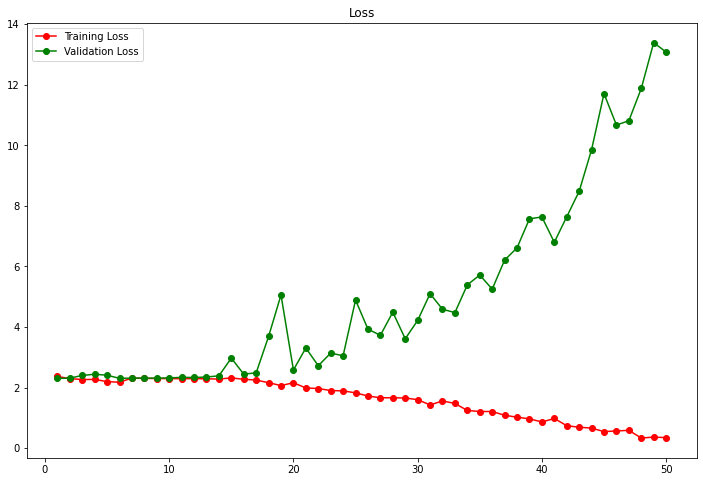
**Implementation**

One of the most important part of the project was to load the video dataset and perform the necessary pre-processing steps. So, 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

**Model 1**

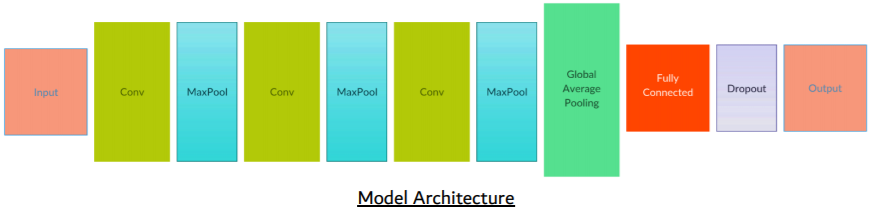
****

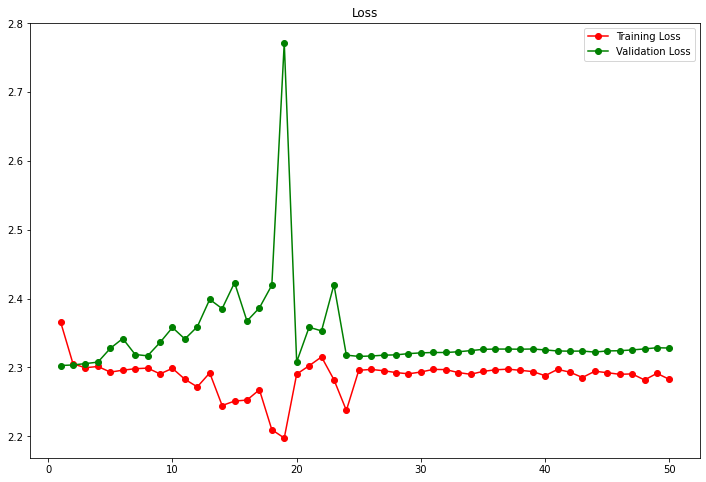
****

Accuracy on train data: 15.00%

Accuracy on test data: 5.26%

**Model 2**

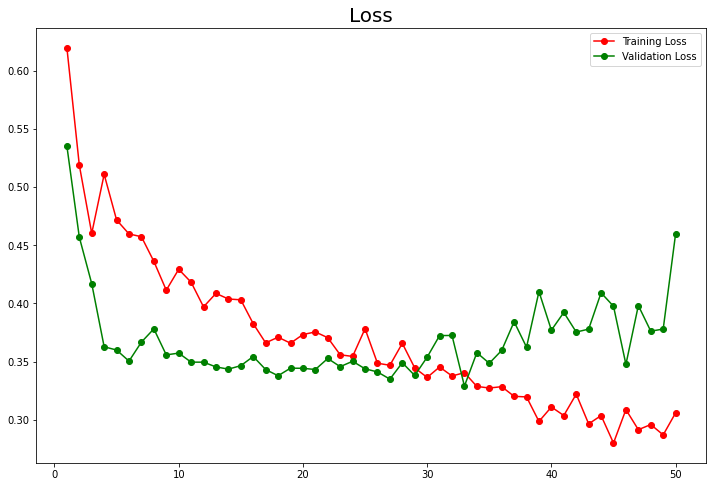
****

****

Accuracy on train data: 11.67%

Accuracy on test data: 5.26%

**Model 3**

****

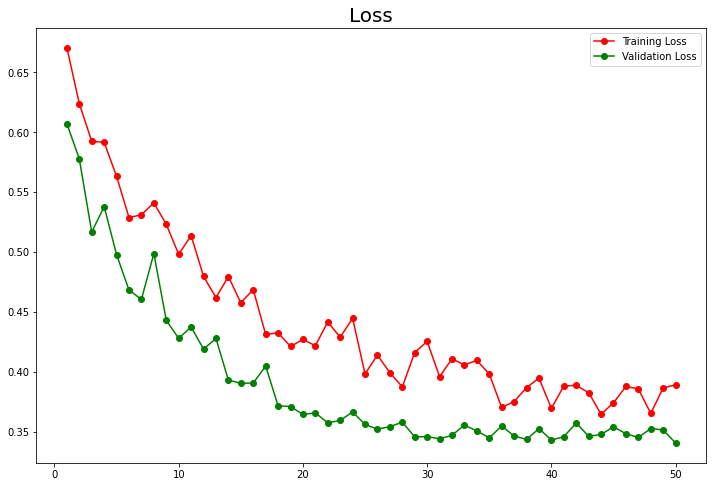
Accuracy on train data: 41.67%

Accuracy on test data: 10.53%

**Results**

**Model 4**

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. The final clearly tells that if more is given to the model then our learning rate wil l improve more and accuracy can be increased.

****

Accuracy on train data: 28.33%

Accuracy on test data: 10.53%

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.