**Indian Institute of Technology Indore**
**Discipline of Computer Science and Engineering**
**Minor Project in the course “Computational Intelligence”**
**Spring 2022-2023**

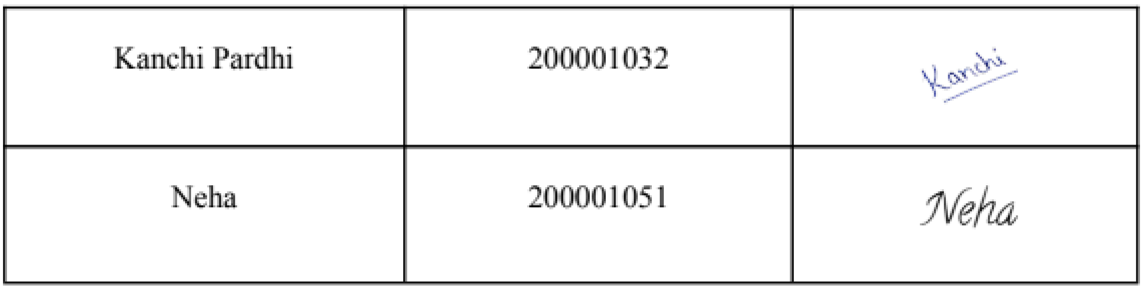
# Title: Dynamic Image Generation For Action Recognition

# Project Report

# Team Members

Kanchi Pardhi (200001032)
Neha (200001051)

Under the Supervision of
Dr. Aruna Tiwari Professor,
CSE



| | | |
|---|---|---|
| Kanchi Pardhi | 200001032 | Kanchi |
| Neha | 200001051 | Neha |

# INDEX

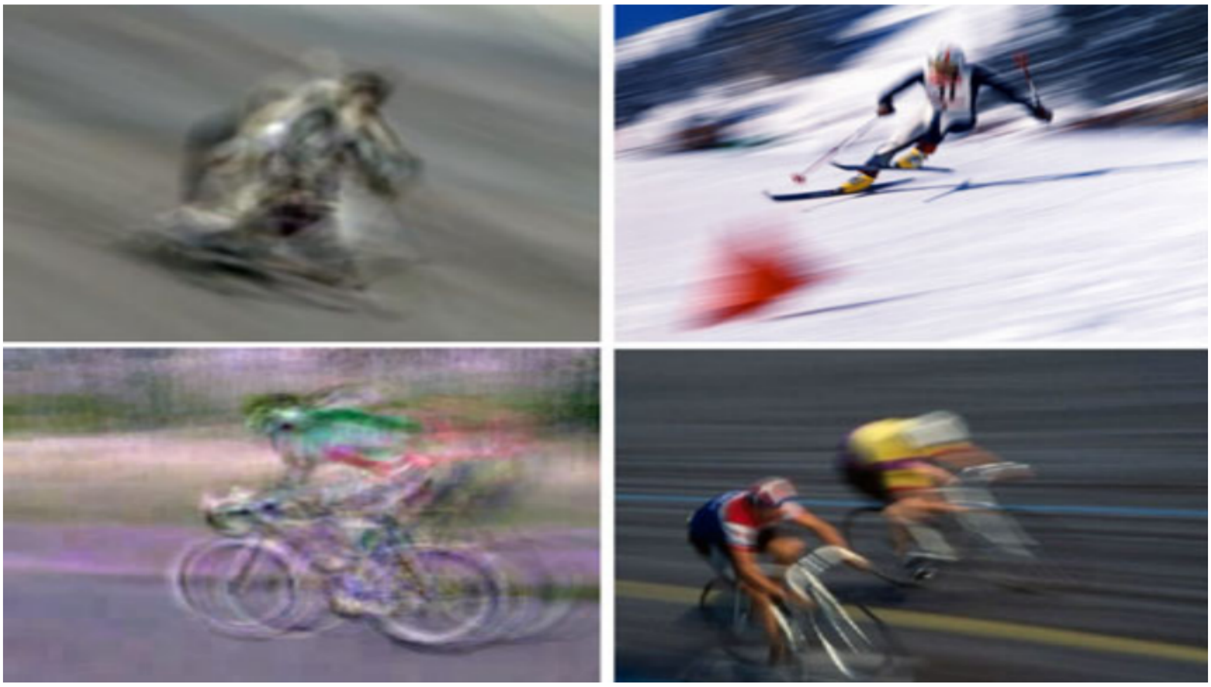
## 1. Problem Statement

Videos outnumber still images by a huge margin in terms of the amount of visual data that is now available. Therefore, it is crucial to precisely and thoroughly comprehend the content of videos. Videos are made up of a series of still pictures. The challenge of comprehending video content is the main concern. This is accomplished by viewing films as a stack of frames, after which discriminative models are developed that extract the data required to address certain issues like action recognition.

In this project, we provide a new method for representing videos that, like the preceding instances, encodes the data in a generic and content-independent way, producing a long-lasting, reliable motion representation useful for both action recognition and other video analysis tasks. This fresh depiction simplifies a single image, which we refer to as the dynamic image, that combines the motion information present in each frame of a movie.

## 2. Dynamic Image Generation

Dynamic image generation from video refers to the process of extracting still images or frames from a video and then generating new images based on those frames.
It is a RGB still image that summarizes, in a compressed format, the gist of a whole video (sub)sequence
They explain some form of movement.



## 3. Implementation

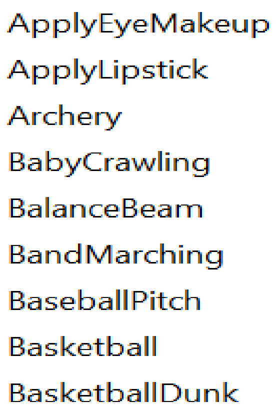
We need to develop a dynamic image for a video that will be utilised for action recognition later on. We will build dynamic images from a video by combining a certain number of frames. For example, if we have a 25-frame video, we will generate 1 dynamic image using these frames. Now, using that dynamic image, the video will then be classified based on human activity, i.e. action recognition. (This will be handled in the testing part).

## 4. Frames Combination for Image Generation

We have taken the mean of all the frames for a video.
We also tried some other methods but the mean was giving good results hence proceeded with that.

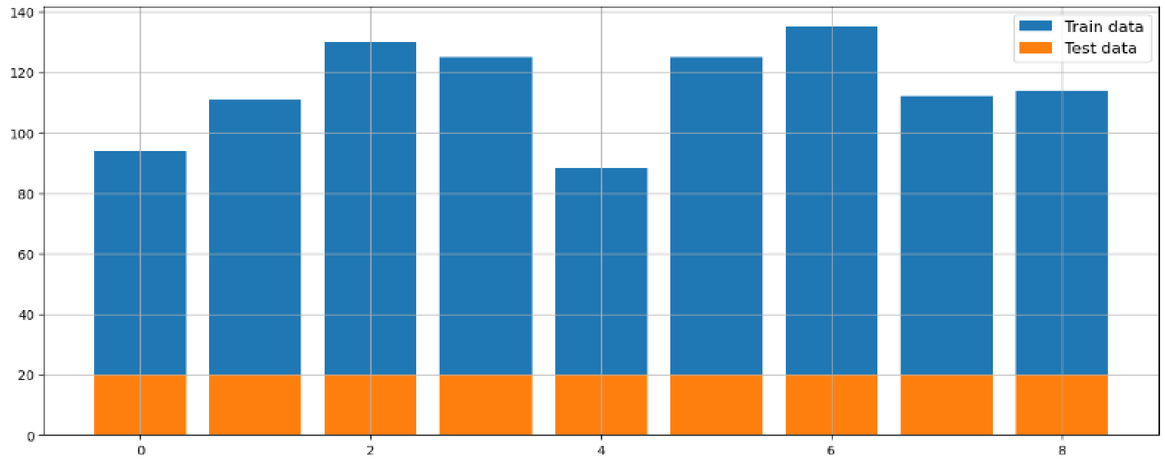
## 5. Data Set

We have used UCF101 (subset of its classes) datasets.
In UCF-101 we have 13, 320 clips distributed in 101 action classes.
For our project we have taken its 9 classes for the classification purpose.



```
ApplyEyeMakeup
ApplyLipstick
Archery
BabyCrawling
BalanceBeam
BandMarching
BaseballPitch
Basketball
BasketballDunk
```

9 classes with 1214 images in total



## 6. Data Preprocessing

We need to convert video clips into frames

Some basic steps that we are going to implement for obtaining frames :

- Load the video clip: The first step is to load the video clip using a suitable library such as OpenCV or FFmpeg. These libraries provide various functions for working with video files.
- Extract the frames: Once the video clip is loaded, the next step is to extract the frames from the clip. This can be done by looping through the video frames and saving each frame as an image file.
- Preprocess the frames: After extracting the frames, you may need to preprocess them to improve the quality of the images. This could include tasks such as resizing, cropping, and color correction.

- Save the frames: Once the frames have been extracted and preprocessed, they need to be saved in a suitable format such as JPEG or PNG.
- Use the frames for analysis: Once the frames have been saved, you can use them for various analysis tasks such as object detection, image classification, or image segmentation.

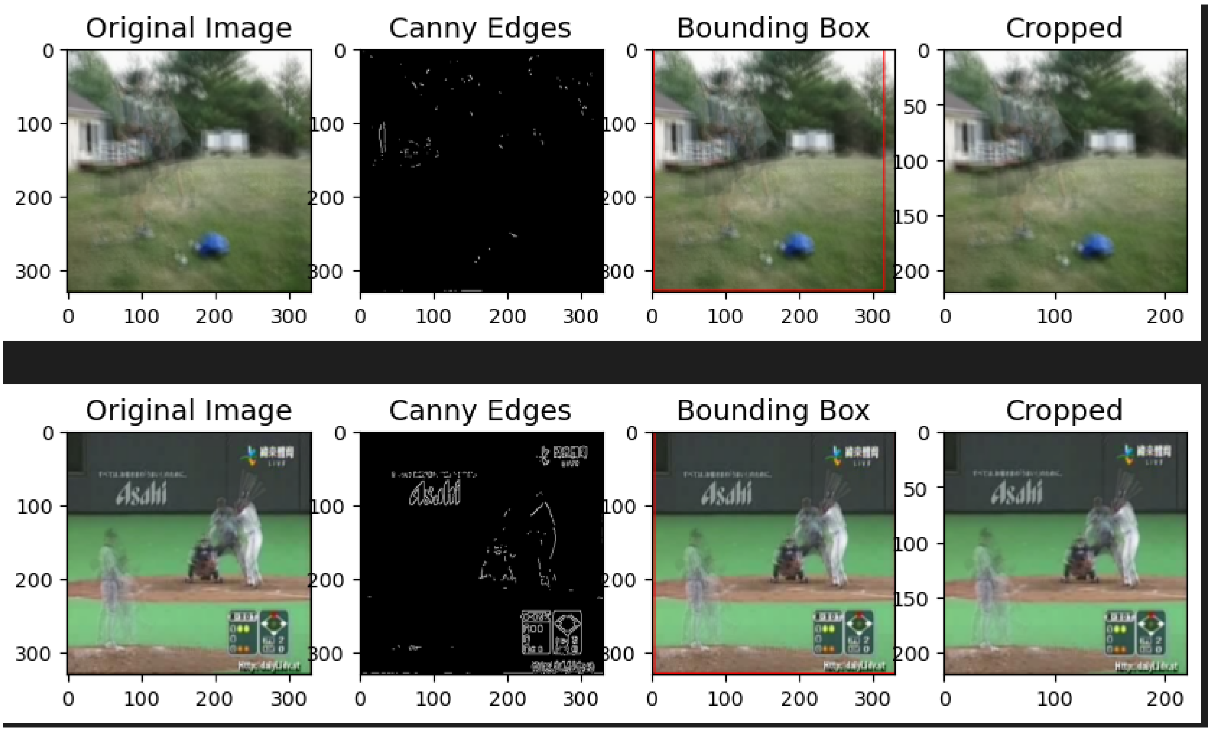
Used Open CV libraries video capture function to extract the frames from the video at certain intervals.

The image read by cv2 was in BGR format, we have to convert it into RGB format. Without this option images will be shown in blue hue because matplotlib uses RGB to display image

Resizing all the images to the same dimensions ( 330 x 330).

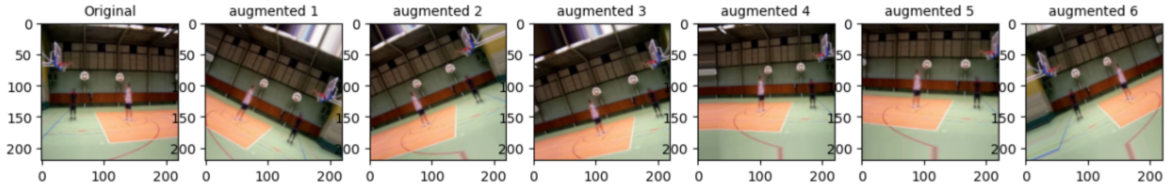
**Some of the dynamic images generated with their labels on the top**

Cropping edges of images using Canny algorithm.
Now the images are ready for training and testing.



## 7. Data Augmentation

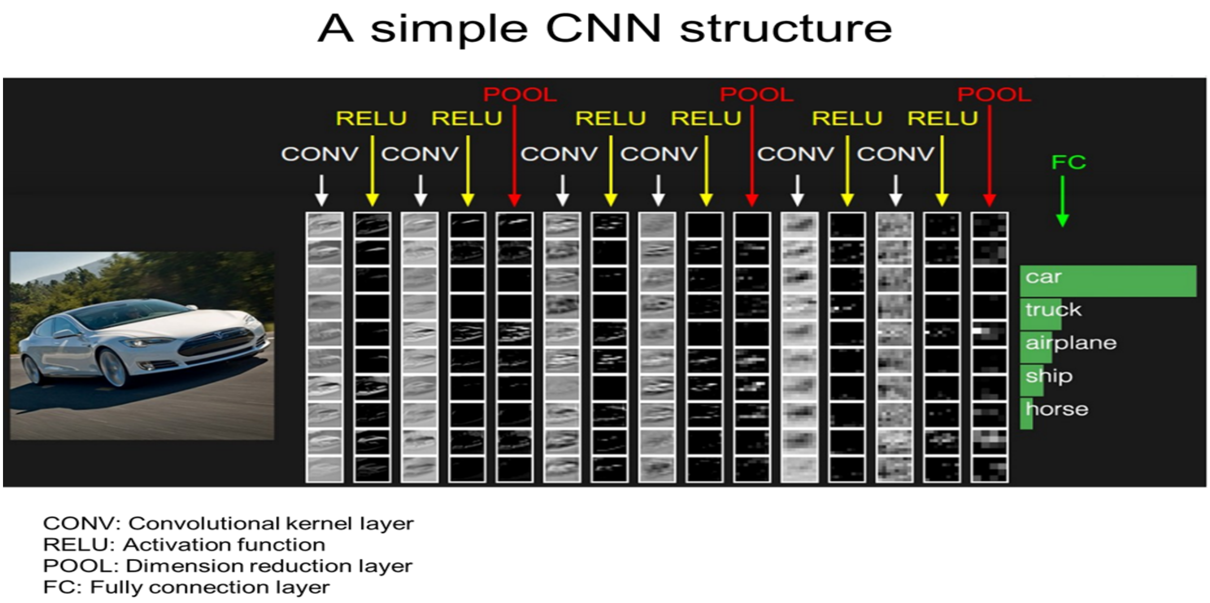
Image augmentation is a creation of additional training data based on existing images, for example translation, rotation, flips and zoom. Using ImageDataGenerator class from Keras library create additional images of each class.



## 8. Convolutional Neural Network

- CNNs are fully connected feed forward neural networks.
- CNNs are trained to identify the parameters like edges of objects in any image.

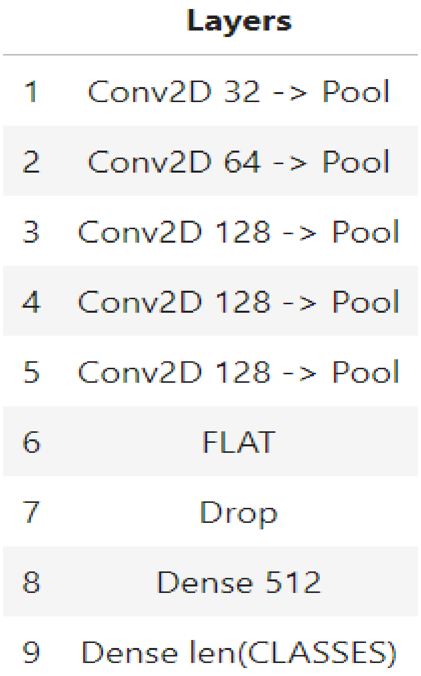
- CNNs are very effective in reducing the number of parameters without losing on the quality of models. Images have high dimensionality (as each pixel is considered as a feature) which suits the above described abilities of CNNs
- CNN is mainly used to extract features from the images.
- It contains a set of kernels (filters), parameters which are to be learned throughout the trainings.
- Each filter convolves with the image and feature maps are created.



A simple CNN structure

CONV: Convolutional kernel layer
RELU: Activation function
POOL: Dimension reduction layer
FC: Fully connection layer

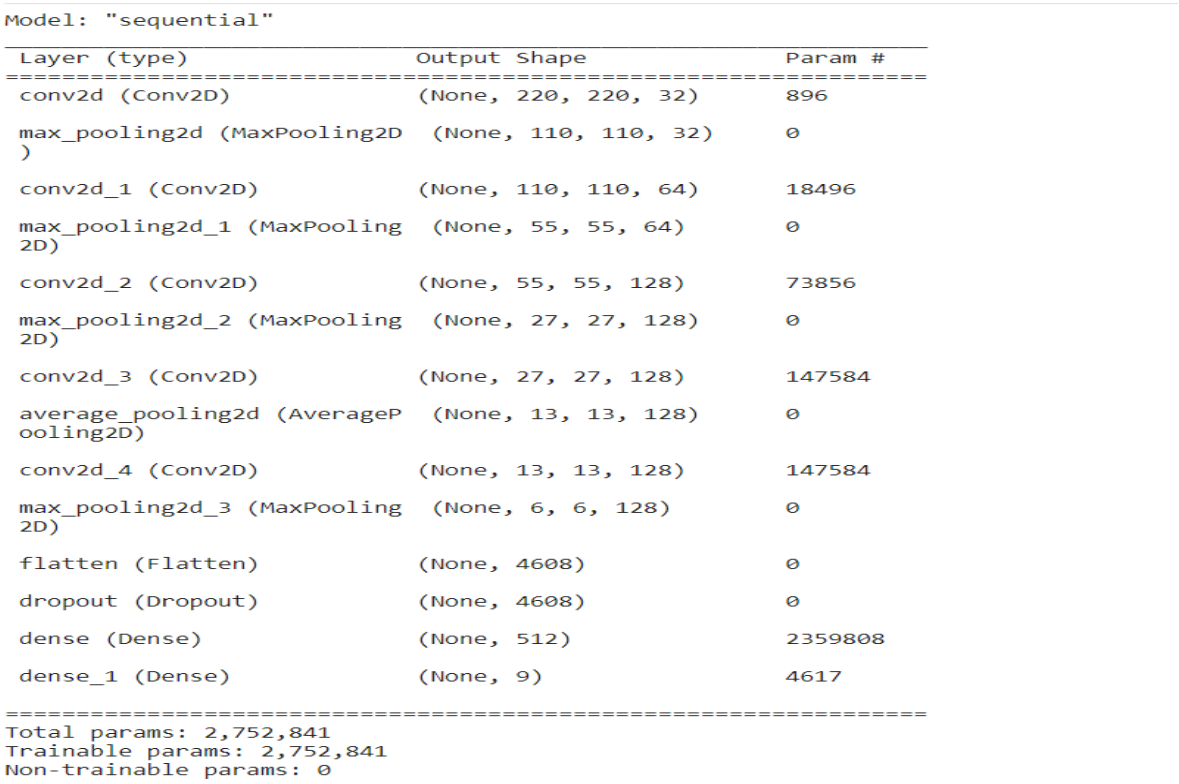
Architecture of the CNN Model used



|   | Layers |
|---|---|
| 1 | Conv2D 32 -> Pool |
| 2 | Conv2D 64 -> Pool |
| 3 | Conv2D 128 -> Pool |
| 4 | Conv2D 128 -> Pool |
| 5 | Conv2D 128 -> Pool |
| 6 | FLAT |
| 7 | Drop |
| 8 | Dense 512 |
| 9 | Dense len(CLASSES) |

- Conv2D with hyperparameters mentioned above: Conv2D(kernel_size, (filters, filters), input_shape=(img_w, img_h, 3)) with activation function for each layer as a Rectified Linear Unit (ReLU): Activation('relu').
- MaxPooling2D layer to reduce the spatial size of the incoming features; 2D input space: MaxPooling2D(pool_size=(max_pool, max_pool)).
- Do the same increasing the kernel size: 32 -> 64 -> 128 -> 128 -> 128
- Flatten the input: transform the multidimensional vector into a single dimensional vector: Flatten().
- Add dropout layer which randomly sets a certain fraction of its input to 0 and helps to reduce overfitting: Dropout(0.5)
- Add fully connected layer with 512 nodes and activation function relu: Dense(512), Activation('relu').
- Provide last fully connected layer which specifies the number of classes . Softmax activation function outputs a vector that represents the probability distributions of a list of potential outcomes: Dense(activation='softmax').

## 9. Model Summary



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 220, 220, 32)      896

 max_pooling2d (MaxPooling2D  (None, 110, 110, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 110, 110, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 55, 55, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 55, 55, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 27, 27, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 27, 27, 128)       147584

 average_pooling2d (AverageP  (None, 13, 13, 128)      0
 ooling2D)

 conv2d_4 (Conv2D)           (None, 13, 13, 128)       147584

 max_pooling2d_3 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 flatten (Flatten)           (None, 4608)              0

 dropout (Dropout)           (None, 4608)              0

 dense (Dense)               (None, 512)               2359808

 dense_1 (Dense)             (None, 9)                 4617

=================================================================
Total params: 2,752,841
Trainable params: 2,752,841
Non-trainable params: 0
```

## 10. Optimizers

Optimizer that implements the Adam algorithm.
Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.
Some of its arguments are :
learning_rate : default to 0.001
epsilon : defaults to 1e-7
beta_1: the exponential decay rate for the 1st moment estimates, defaults to 0.999.
beta_2: the exponential decay rate for the 2nd moment estimates, defaults to 0.999.
The first moment is mean, and the second moment is variance.

Overall, this CNN has 5 convolutional layers with increasing filter sizes and 4 max pooling layers to reduce the spatial dimensions of the features. The final dense layers are used for classification. The Dropout layer is added to prevent overfitting.

## 11. Loss function

It measures how different the predicted outputs of a machine learning model are from the true outputs. The goal of a deep learning model is to minimize the value of the loss function, as this indicates that the model is making more accurate predictions.
Here used the loss function “sparse_categorical_crossentropy”.
Sparse categorical cross-entropy is a loss function used in multiclass classification problems where the target variable is represented as integer labels. It measures the difference between the predicted probability distribution over the classes and the actual integer labels. The loss is calculated by taking the negative log of the predicted probability for the correct class label. During training, the model updates its weights in the direction that minimizes the sparse categorical cross-entropy loss function.

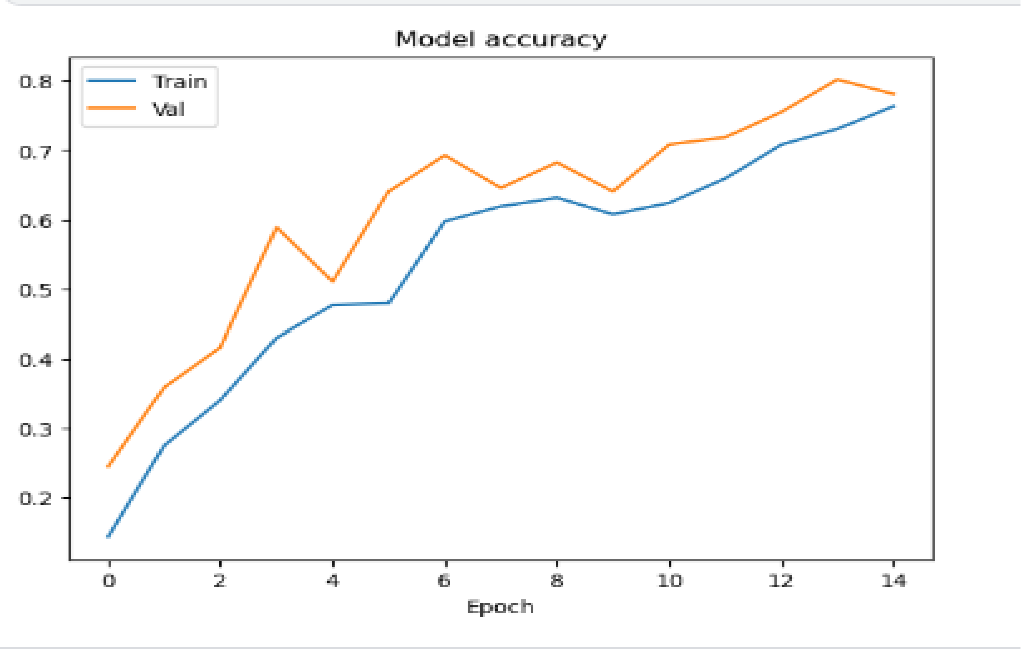
## 12. Performance Metrics

Accuracy: The percentage of correctly classified samples out of the total number of samples.

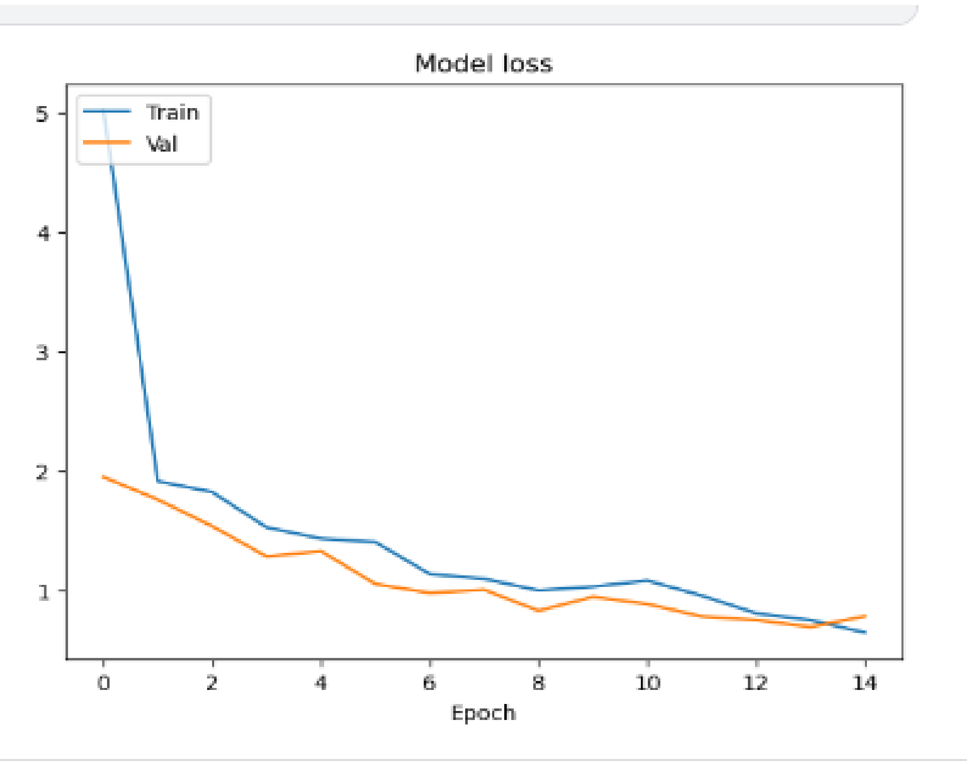
Confusion matrix: A table that summarizes the classification results. It shows the number of true positives, false positives, true negatives, and false negatives.

## 13. Results

**Model Accuracy on training  and validation dataset**



### Model loss on training and validation dataset

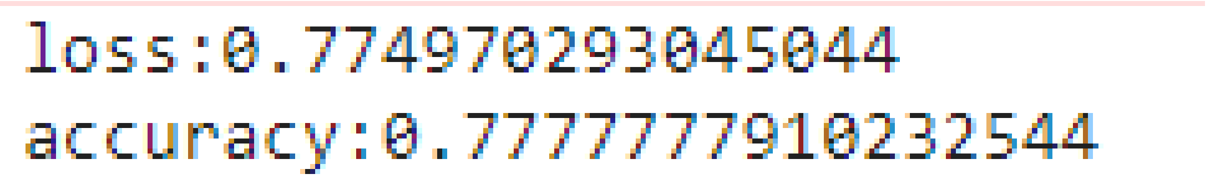


### Loss and Accuracy on training dataset

After 15 epochs…………
loss: 0.6511

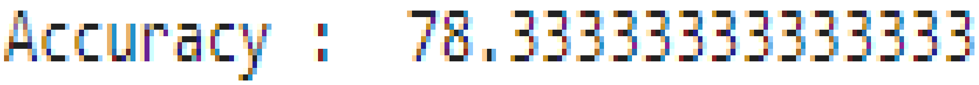
accuracy: 0.7635
val_loss: 0.7868
val_accuracy: 0.7812

**Loss and Accuracy on validation dataset**



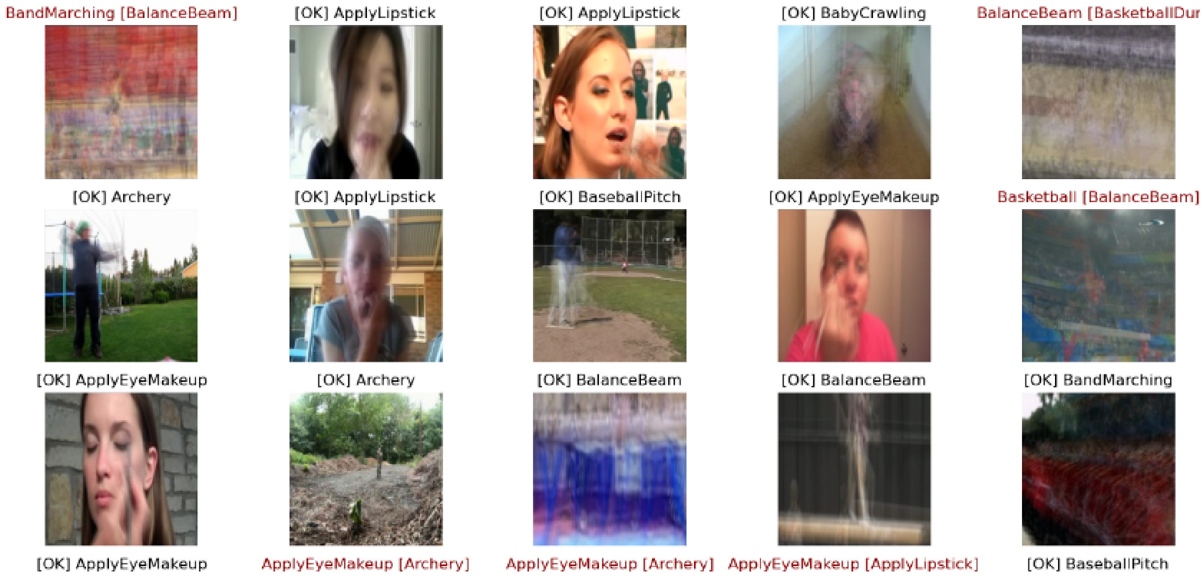
```
loss:0.774970293045044
accuracy:0.777777910232544
```

**Test data Accuracy**



```
Accuracy :  78.33333333333333
```

Some of the images from the test folder with a label, class which model predicted, and actual class.



# 13. References

- **Bilen, H., Fernando, B., Gavves, E., & Vedaldi, A. (2017). Action recognition with dynamic image networks. *IEEE transactions on pattern analysis and machine intelligence*, *40*(12), 2799-2813.**
- **Bilen, H., Fernando, B., Gavves, E., Vedaldi, A., & Gould, S. (2016). Dynamic image networks for action recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3034-3042).**