**A DEEP LEARNING APPROACH TO CONTROL A GAME BASED ON HAND GESTURES USING A LOW-RESOLUTION CAMERA**

***ABSTRACT***

*Though Deep learning and Neural Network is used as a potential tool to comprehend a considerable lot of this present reality learning and trans-formation issues, the research projects revealing the simple facts of how to simulate a game application using human gestures with reduced latency and low-resolution camera are very scarce. This project has in its objective to present the details of design and implementation of gesture-controlled game application using deep learning and neural network. A simple third-party game application is taken into consideration and the keyboard inputs used to control the game are controlled through hand gestures with a low-resolution camera with reduced latency. This project initially unfolds with much focus on building an efficient deep learning model using Keras and TensorFlow that could recognize and validate the hand gestures which are fed as inputs through a low-resolution camera. Next, the corresponding keyboard key is predicted for the corresponding input and the keyboard is simulated in real time when the game application is run. Keywords – Deep Learning, Neural Networks, Gesture control, Tensor-Flow, Keras.*

**1) INTRODUCTION**

With the rapid development of computer vision technology, gesture recognition based on vision has low cost and convenient remote interaction, so it is the focus of gesture recognition. Gesture Recognition is a functioning field of exploration with applications, for example, programmed acknowledgment of gesture-based communication, connection of people and robots or for better approaches for controlling computer games. We will expect to order various pictures of hand signals, which implies that the PC should "learn" the highlights of each motion and group them effectively. For instance, in the event that it is offered a picture of a hand doing a go-ahead signal, the yield of the model should be "the hand is completing an approval motion". Vision-based gesture recognition methods can be roughly divided into two types: one is to use Kinect, Leap motion and other depth cameras or neural networks to obtain image depth information, such as position information of gestures; the other one is to split the gesture from the background by traditional methods and then extract the apparent image characteristics of the gesture itself to perform gesture recognition. We consider a simple interactive game application and the keyboard inputs used to control the game are controlled through hand gestures with a low-resolution camera with reduced latency.

**Problem Definition**

Gesture based controls is a thriving domain in gaming environment, but the devices use a high-resolution camera to capture gesture and use it for controlling the system. Utilizing portions of the human body as input has the advantage of being consistently accessible as the client is not needed to carry any optional gadget. Significantly, appropriating portions of the human body for motion-based connection has been appeared to improve client experience and commitment. While body as information is not altogether new, existing methodologies which influence PC vision, wearable, and sensors (Kinect, wii) here and there experience the ill effects of precision challenges, are not versatile and can be challenging to coordinate with 3rd party software. There are several challenges in gesture recognition based on the video streams:

1. The background is complex and easily affected by light.

2. There will be a large number of redundant gesture pictures.

3. The multi-view of the human hand and the inconsistent shape size.

The 3D Convolutional Neural Network (CNN) Long Short-Term Memory (LSTM) approach with appropriate morphological operations is used to remember the direction of motion of hands thus removing the ambiguity of certain hand gestures while the CNN model predicts the hand gesture of the user. The proposed methodology does not get affected with the objects of same color and removes the need for an external device, yet achieving the same accuracy. We propose a way to control the system and yield better results using the system (a low resolution) camera by following a deep learning-based approach. We intend to use deep learning to enhance the low-quality image and then use it to perform the right action based on the gesture demonstrated by the user, with a low latency.

**2) LITERATURE SURVEY**

In this section we have analyzed the existing approaches to solve this problem and have described our inferences from them.

**2.1 Data Capturing**

The data in real time without compromising in its quality is one of the major challenges in real time gesture recognition problems. (Molchanov, Gupta, Kim, and Kautz, Molchanov et al.) proposed an algorithm for drivers’ hand gesture recognition (Neethu et al., 2020) from challenging depth and intensity data where the data was captured with the Microsoft Kinect device and have a resolution of 115 × 250 pixels.

**2.2 Computational complexity and Optimization**

Maintaining the accuracy of prediction while reducing the latency is challenging, thus demanding reduction in computational complexity and optimization. By using Long-term Recurrent Convolutional Networks (LRCN) classifier where multiple frames are sampled from the video sequence reduce computational complexity (John et al., 2016). Gesture recognition based on gyroscope and accelerometer sensors (Kim et al., 2018) and combining data from gyroscope, four convolution and Gated Recurrent Unit (GRU) layers to remove external devices (Roccetti et al., 2012) are key for optimization of the model.

**2.3 Pickling and Flask API**

Python Pickling is used to save the deep learning model and Sci-kit Learn is used for pipe-lining the pre-process, classification. Since consumers of machine learning model rarely use python and use other technology stacks a flask API will make cross-Platform integration easier.(Yaganteeswarudu, 2020) When the API is called with appropriate parameters, it will invoke the corresponding model and the classification result be sent as a JavaScript Object Notation (JSON) file

**2.4 Summary**

As we had observed from the past works, the gesture recognition has either been done with the help of external devices like motor sensors, accelerometer and gyroscope or using a high costing depth camera, like Kinect. In one of the works, there wasn’t enough data considered. There seems to be scope to decrease the computational complexity in parallel to increase the gesture recognition accuracy. This paper aims to present a novel idea using low resolution camera with reduced latency. This is achieved by using gulpio package which prevents GPU starving during training of the deep learning model.

**2.5 Data Set Source of data set:**

https://20bn.com/datasets/jester/v1 (Jester Dataset). The dataset around 27 types of hand gestures. The dataset contains 148,092 small videos of various gestures. Each of this small clip has 37 to 40 frames, that sums up to 5479404 frames. The 20BN-JESTER dataset is an ample collection of labeled video clips that show humans performing pre-defined hand gestures in front of a laptop camera or webcam. The dataset was created by a large number of crowd workers. It allows for training robust machine learning models to recognize human hand gestures.

**2.6 Software/Tools Requirements**

Keras Framework – an open-source library built over TensorFlow used for deep learning applications

Gulp io - Python Package used for Binary storage format for deep learning on videos

TensorFlow – a open-source library used for machine learning applications

OpenCV - To get image frames from the video

GPU - For training the model

**3) PROPOSED SYSTEM**

**Specification of the Stages**

The process of Building a Gesture Recognition System using Deep Learning has the following stages.

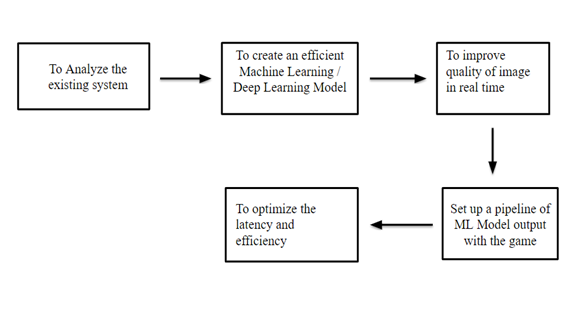
1. Challenges with training neural networks on massive video dataset

2. The development of a network architecture that allows for classifications of video clips solely with RGB input frames

3. Iterations important to make the neural network run continuously on embedded devices.

4. Lastly, the discovery of interesting and playful gesture-based applications.

**Model Diagram**



**3.1 System Analysis**

In this Chapter a detailed description of our system is given below along with an architecture diagram.

**3.1.1 System requirement analysis**

• An API to expose the model to other programming dialects

• A Deep learning model to predict the hand gestures

• Change the model underway without influencing the existing server

**Functional Requirements:**

• API with version control

• API to serve different models to diverse types of clients

• Generate report and stats

• User feedback report for management

• The software should be compatible with cross-platform access

**Non-Functional Requirements:**

• Logging the data for future analytics purpose

• Cost-effective: keeping the modules which are effective and necessary

**3.1.2 Description of modules**

Here we describe the modules responsible for various functionalities.

**Importing the data**

Jester dataset has 148,092 small videos of various gestures. Each of this small clip has 37 to 40 frames, that sums up to 5479404 frames. This dataset has CSV file with columns as filename and its corresponding gesture. While using our personal computer, often we will not have the performance needed to work on all this data. To simplify we will have to import only those gestures corresponding to the game that we have chosen.

**Preprocessing of the video dataset**

Preprocessing is one of the most important steps in our project. We will do a multistaged preprocessing.

Stage 1 : RGB to Gray scale conversion. Convert all the frames from color images to black and white.

Stage 2 : Unifying Frames. The videos do not have the same number of frames, here we try to unify. For each gesture we try to duplicate last frame if video is shorter than necessary, else if there are more frames, then sample starting offset.

Stage 3 : Resize the frames. Resize all the frames to a fixed height and width.

Stage 4 : Normalization. Normalize each frame. Data normalization is a crucial step which ensures that each input parameter (pixel, in this case) has a similar data distribution. This makes convergence faster while training the network.

**Model Construction and Compilation**

This module consists of the following steps:

1.Construct a custom model, where we use 3D convolutional layer, Max pooling layer, Convolutional LSTM layer (to learn from the sequence of frames), flatten layer, dropout layer and dense layer. Then, we will have to do hyper-parameter tuning, as this will directly impact the performance of the model.

2.Compile the model using loss as sparse categorical cross entropy, then go about trying various optimizers for our model.

3.Train the model with the training data and the appropriate parameters.

4.Use the validation data to analyze the performance of the model and make the further modifications to make the right model.

5.Save the weights and architecture of this trained model to be used in our driver application.

**Deployment of the model**

This module consists of the following steps:

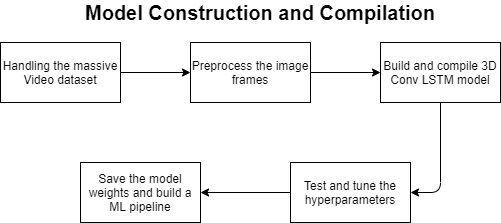
1.Make a web application using flask which will capture users’ gesture and use the model to recognize the gesture and control the game.

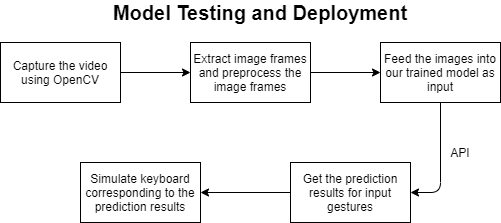
2.Load the weight from the previously saved data into this newly created model.

3.Capture the video through user’s low-resolution camera and preprocess the frames.

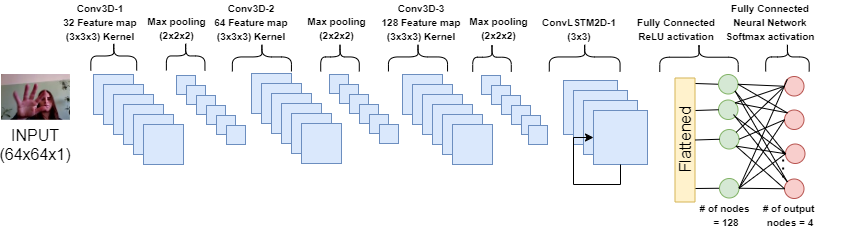
4.Predict the gesture of the current action.

5.The predicted output will be mapped to a keyboard action, which in turn gets reflected on the game





**Architecture Diagram**



There are two 3D convolutional layers. Each convolutional layer is followed by a max-pooling layer. There is one convolutional LSTM layer. There is one Flatten Layer. There are two fully connected layers with 128 and 4 nodes respectively, following the flatten layer. There is a SoftMax activation layer at the end to predict the class probabilities for 4 gesture classes.

**4) IMPLEMENTATION AND TESTING**

This chapter describes steps of implementation and testing in detail.

**4.1 Data Capturing**

The low-resolution camera captures the video continuously and image frames are extracted using OpenCV python library. We aim to prevent using high resolution camera for image capturing. The hand gestures of the users are captured through the video feed and then the captured image frames are preprocessed as required. We also perform morphological operations so as to make the captured image frames suitable for passing as inputs to the deep learning model. The image frames captured are then resized to 64 x 64.

**4.2 Loading Data from video dataset**

Jester dataset has 148,092 small videos of various gestures. The 20BN-JESTER dataset is a huge assortment of named video cuts that show people performing pre-characterized hand motions before a PC camera or webcam. The dataset was made by an enormous number of group laborers. It considers preparing hearty AI models to perceive human hand motions. Each of this small clip has 37 to 40 frames, that sums up to 5479404 frames. This dataset has CSV file with columns as filename and its corresponding gesture. In the model training phase, first the frames are extracted from the video dataset at the rate of 10 frames per second using OpenCV.

**4.3 Labelling the image frames**

In the training phase, after the frames were extracted from the video files in the training data, those frames are stored in a folder. During frame extraction, the count of number of frames obtained for each hand washing step is monitored and it is used to label the image frames. A Comma Separated Values (CSV) file is used to store the name of the image frame and its corresponding step(label).

**4.4 Split tarin and test data**

After the frames were labelled, they are split into training and validation data. In our dataset, the number of frames for each step is different. So, splitting the entire frames in a specific ratio will result in poor training of the neural network model. So, the frames in each step are split in the ratio 75:25 and they are grouped into training data and testing data.

**4.5 Preprocessing**

Preprocessing is one of the most important steps in our project. We will do a multistage preprocessing.

Stage 1 : RGB to Gray scale conversion. Convert all the frames from color images to black and white.

Stage 2 : Unifying Frames. The videos do not have the same number of frames, here we try to unify. For each gesture we try to duplicate last frame if video is shorter than necessary, else if there are more frames, then sample starting offset.

Stage 3 : Resize the frames. Resize all the frames to a fixed height and width. The image frames in the training and validation data are then resized into 64 X 64 which is the required input size for the first layer of 3D Conv LSTM model.

Stage 4 : Normalization. Normalize each frame. Data normalization is a key step which ensures that each input parameter (pixel, in this case) has a similar data distribution. It accepts NumPy array as input and it gives the normalized values as output. This makes convergence faster while training the network.

**4.6 Deep Learning model**

There are three 3D convolutional layers. Each convolutional layer is followed by a maxpooling layer. There is one convolutional LSTM layer. There is one Flatten Layer. There are two fully connected layers with 128 and 4 nodes respectively, following the flatten layer. There is a SoftMax activation layer at the end to predict the class probabilities for 4 gesture classes.

**4.7 PyAutoGUI**

A basic third-party game application is mulled over and the console inputs used to control the game are controlled through hand gestures. The corresponding keyboard key to the users’ hand gestures is predicted by the deep learning model and the keyboard is simulated in real time when the game application is run with the help of PyAutoGUI. PyAutoGUI is a cross-platform Graphical User Interface (GUI) automation Python module for people used to automatically control the mouse and console. PyAutoGUI lets your Python scripts control the mouse and console to automate associations with different applications. The API is intended to be as basic as could reasonably be expected. The PyAutoGUI module has functions for simulating mouse movements, button clicks, and scrolling the mouse wheel. With GUI automation, your programs can do anything that a human user sitting at the computer can do. PyAutoGUI has several features:

• Moving the mouse and clicking or typing in the windows of different applications.

• Sending keystrokes to applications.

• Take screen captures, and given a picture, discover it on the screen.

• Find an application’s window, and move, resize, expand, limit, or close it.

• Show message boxes for client connection while your GUI automation content runs.

**4.8 Serve through Tensorflow**

Deploying the model is one of the last phases of any AI extend and can be precarious. Most of the ventures stop in the wake of building and testing it locally however it is not the end. It ought to be accessible for the end-clients with the goal that they can utilize it. TensorFlow Serving is an adaptable, superior serving framework for deep learning models, intended for creation conditions. TensorFlow Serving makes it simple to send new calculations and examinations, same server architecture and APIs.

TensorFlow Serving furnishes out-of-the-crate combination with TensorFlow models, however can be handily reached out to serve different sorts of models and information. Tensorflow serving gives answers for the accompanying post deployment challenges.

• If an individual needs to change the model underway without influencing the current server.

• If an individual needs to move back to past stable version.

• If an individual needs to test numerous models or serve various models to various kinds of customers.

Now we can make solicitations to the server and get the outcome. Commands are as follows,

• tensorflow\_model\_server –rest\_api\_port = 9000 –model\_base\_path = "path to model folder" –model\_name=demo

• rest\_api\_port = Port number to send requests

• model\_base\_path = Saved model directory path

• model\_name = You can give any name here (we will use the same name while making the requests)

**5) RESULTS**

Initially the 20BN Jester dataset was explored. Jester dataset has 148,092 small videos of various gestures. Each of this small clip has 37 to 40 frames, that sums up to5479404 frames. This dataset has CSV file with columns as filename and its corresponding gesture. But we need only four gestures namely Swiping-Left, Swiping-Right, Stop Sign and Thum Up. So, we extract all the image frames corresponding to these 4 gestures from the dataset containing 27 types of gestures.

Next, we performed a multi-staged preprocessing which involved RGB to Grayscale conversion, unifying the number of frames in each folder, normalization of image frames and conversion of image frames into NumPy arrays. At last, the CONV3D LSTM model was built and the model was fit on the training data and compiled.

**Discussion and Comparison to existing works**

We've seen independent game production evolve over the past decade from a marginal, smoldering corner of the market to a raging wildfire relentlessly spreading across the globe. In favor of digital downloads, we have seen retail launches plummet. We've looked at how the demographics of gaming have wildly changed to include almost anyone we meet. This has all influenced the production of games in different ways. The nature of game development teams has changed; the types of games being made have changed; the ways in which games are manufactured, promoted, and monetized have changed; and relationships have changed between developers and gamers. In such an environment many prefer not to play video games only because it is difficult to learn the controls. We are proposing a gesture-based simple control using which a user will be easily able to control any game of his liking.

We take on an approach to use 3D Convolutional LSTM model to be able to classify user action to a meaningful input for the game. Our approach brings forward a model wherein the accuracy established is 80% (approx.). Existing works in this field seems to pose the problem of high latency, though holding a better accuracy. The increasing popularity of video games has led to an interest in researching the impact of latency on users of these systems and the effectiveness of methods to optimize for this latency. Latency can have a significant impact on the online computer gaming experience of the end user. Latency causes the user's commands to be delayed so that by the time the command is received and executed by the server the game state has changed [7]. Our work has taken both accuracy and latency into consideration. We capture the gesture performed by the user and pass it on to the already trained and deployed (as mentioned in previous section) multi class classifier 3D ConvLSTM model, which then gives out a label as an output. We then use pyautogui to map this label to a meaningful keyboard action which in turn gets applied on the selected region. We are able to achieve low latency by not embedding our gesture recognition module along with the game control module, the game module recognizes as if the input is direct from the keyboard. Having the control module working separately and in parallel to the game module allows us to make predictions and convert it to keyboard action work with a minimal latency.

**6) Conclusion**

We have proposed a model to simulate a game application using human gestures with reduced latency and low-resolution camera. This model aims to reduce the latency without having the need for a high-resolution camera with help of efficient pre-processing and deep learning-based model architecture. The gulpio package is playing an important role in overcoming the starved GPU problem by caching the image frames during the training phase of the model. The LSTM layers present in the model help the model to remember and distinguish ambiguous gestures like swipe-left and swipe-right. So, we have provided a cost-efficient and time-efficient solution to control any application through human gestures.

**7) Problems of using low resolution camera**

The current arrangement proposed in "A productive strategy for human hand signal identification and acknowledgment utilizing profound learning convolutional neural organizations" utilizes a cycle stream that comprises of hand locale of premium division utilizing veil picture, fingers division, standardization of divided finger picture and finger acknowledgment utilizing CNN classifier. This methodology utilizes CNN to identify signals dependent on number of fingers as opposed to the gesture. In "Hand Gesture Recognition With 3D Convolutional Neural Networks", they propose a calculation for driver’s hand motion acknowledgment from testing profundity and force information utilizing 3D convolutional neural networks. Their answer consolidates data from various spatial scales for the last prediction.

**7.1 To handle low resolution image frames**

We propose an answer for utilize all the edges of a video to pass it into a CNN which has convLSTM layer, it is like a LSTM layer, yet the information changes and intermittent changes are both convolutional. And furthermore, we mean to utilize conv3d layer for spatial convolution over volumes. Along these lines, permitting us to anticipate the signal as well as the heading of development (left to right, start to finish), which thus makes the motion acknowledgment framework more robust.

**REFERENCES**

1. John, V., Boyali, A., Mita, S., Imanishi, M., and Sanma., N. (2016). “Deep learningbased fast hand gesture recognition using representative frames.” International Conference on Digital Image Computing: Techniques and Applications (DICTA), 1–8.

2. Kim, J.-H., H., G.-S., K., B.-G., and Dogra, D. P. (2018). “Deepgesture: Deep learningbased gesture recognition scheme using motion sensors.

3. Molchanov, P., Gupta, S., Kim, K., and Kautz, J. “Hand gesture recognition with 3d convolutional neural networks.

4. Neethu, P., Suguna, R., and Divya, S. (2020). “An efficient method for human hand gesture detection and recognition using deep learning convolutional neural networks.” Soft Computing (2020), 1–10.

5. Roccetti, M., Marfia, G., and Semeraro, A. (2012). “Playing into the wild: A gesturebased interface for gaming in public spaces.” Journal of Visual Communication and Image Representation, 23, 426–440.

6. Yaganteeswarudu, A. (2020). “Multi disease prediction model by using machine learning and flask api.” 2020 5th International Conference on Communication and Electronics Systems (ICCES). 1242–1246.

7. Ian McNeil, Matthew Beyler. https://web.cs.wpi.edu/~claypool/mqp/latency-comp/imcneil-mbeyler-mqpreport.pdf