

# Action Recognition using Transfer Learning and Majority Voting for CS:GO

**Abstract**—Presently online video games have become a progressively favorite source of recreation and Counter Strike: Global Offensive (CS: GO) is one of the top-listed online first-person shooting games. Numerous competitive games are arranged every year by Esports. Nonetheless, (i) No study has been conducted on video analysis and action recognition of CS: GO game-play which can play a substantial role in the gaming industry for prediction model (ii) No work has been done on the real-time application on the actions and results of a CS: GO match (iii) Game data of a match is usually available in the HLTV as a CSV formatted file however it does not have open access and HLTV tends to prevent users from taking data. This manuscript aims to develop a model for accurate prediction of 4 different actions and compare the performance among the five different transfer learning models with our self-developed deep neural network and identify the best-fitted model and also including majority voting later on, which is qualified to provide real time prediction and the result of this model aids to the construction of the automated system of gathering and processing more data alongside solving the issue of collecting data from HLTV.

**Index Terms**—Real Time Action Detection, Transfer Learning, Majority Voting, Performance Metrics, Automated Data Collection, Prediction Model.

## I. INTRODUCTION

Video games have become a significant part of the current century as a source of entertainment; thus, it owns a large portion of the entertainment market in the current world. This study focuses on a first-person shooting game named as Counter Strike: Global Offensive or shortly known ‘CS: GO’ where competition between two teams as attacking and defending roles takes place. While a match is taking place, different kinds of actions take place, kills, deaths, smoking, throwing a grenade, planting a bomb, getting injured, etc. can be shown as examples. The value of these parameters after a game varies from one player to another depending on their skills and experience. Reversely it can be said that, if the parameter values can be predicted accurately, not only the quality of a player in terms of skills and experience can be identified but also the winning and losing probability can also be determined even before the match have started. As the gaming market has grown substantially larger and still growing, as well as many huge tournaments and competitions are taking place all across the world, this kind of prediction that forecasts an individual’s and team’s attributes can be convenient and profitable for the related business industries.

Our work is comprised of five different transfer learning models VGG16, Xception, Inception\_v3, Inception\_ResNet\_v2 and ResNet152\_v2 and majority voting as the focal point, analyzing them separately, finding out the best-

fitted model for the prediction of four different actions of the CS: GO gameplay named as kill [Fig 1], death [Fig 2], smoke [Fig 3] and no action [Fig 4]. During performing our study, we have faced various obstacles such as (i) Data collecting and pre-processing was challenging as HLTV provides only dem formatted files. (ii) No such study on real-time action detection for CS: GO game has been conducted before (iii) Pre-processing of data for training purposes is tiresome and time-consuming. Particularly our contributions are listed below:

- Five different transfer learning models in addition to a self-developed deep neural network model have been used for video analysis.
- The model that has been developed in this study is capable of detecting actions in real time and provides a CSV formatted file for the prediction model simultaneously.
- A system API has been constructed that allows the automated collection and processing of data which can aid to the later training period and enhance our system’s performance.
- Through our constructed API, the frequency of any of the mentioned actions occurring in a video can be determined.

In this manuscript, a short presentation of literature review on preceding action recognition studies has been introduced in the Section II followed by a brief description of our approach, model’s and methods that have been used in the Section III, which is comprised of three sub-sections. The discussion of the system model has been presented in III-A, whereas III-B talks about the data acquisition and preparation techniques and lastly III-C illustrates a quick introduction of the used pre-trained model’s. The performance metrics and result have been demonstrated in the Section IV including confusion matrices for each of the models and finally, the study comes to an end with a conclusion part in Section V.

## II. RELATED WORK

There are numerous studies have been conducted to detect action in real time using CNN, LSTM or various pre-trained models. However, no work has been conducted on CS: GO. In this section, we discuss the previously available papers that conducted work to detect various actions from video footage.

Jeyanthi ET. Al. built a model using the Inception\_ResNet\_v2 pre-trained model and Long Short Term Model (LSTM) for the purpose of human action recognition, where the pre-trained model is used to extract the features and the LSTM is used to learn the order of the action [1]. Moreover, VGG16, ResNet152 and Inception\_v3 are also used

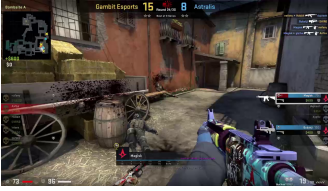


Fig. 1. Kill

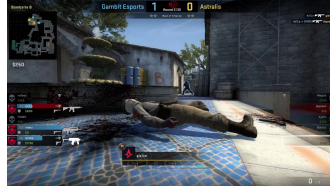


Fig. 2. Death



Fig. 3. Smoke

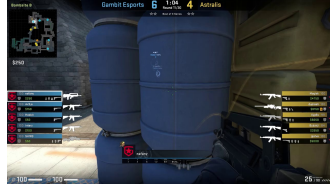


Fig. 4. No Action

as elements of comparison. In the end, the combination of LSTM and Inception\_ResNet\_v2 proves to be gaining the best accuracy, 92% and 91% on UCI 101 and HMDB 51 data sets respectively. Similar kind of work done by Yousry ET. Al. where he showed a three-fold technique which includes a pre-training stage where a generic dataset is used and then this pre-trained dataset is used to the aimed dataset [2]. Finally, LSTM is used combining five different model shapes. VGG16, Xception and DenseNet are used as the pre-trained models where VGG16 proves to be the most precise model with an accuracy of 93.48%.

A different approach is illustrated by Bux Sargano ET. Al. for action recognition using a hybrid model of Support Vector Machines and K-Nearest Neighbor (SVM-KNN) classifier along with the pre-trained CNNs and excludes the use of any sequential CNNs [3]. His proposed merged SVM and KNN model achieves an accuracy of 98.15%. Alexander ET. Al. suggested a strategy to put the good use of predictions from transform learning in action recognition into temporal localization [4]. By incorporating simple CNN with the pre-trained model can meet the challenges offered by two localization action datasets.

Tian ET. Al. developed a 3D convolutional neural network that can not only extract the spatial features but also the temporal features. Moreover, sub-data classification is integrated with the system to deal with the cases of insufficient datasets and this strategy was named as the internal transfer learning strategy [5]. This model attains 98.2% and 100% accuracy on KTH and Weizmann datasets respectively while integrating with ITL. The use of 3D convolutional neural networks can also be observed in Noha ET. Al.'s study [6]. More specifically Inflated 3D (I3D) has been implemented to recognize sign language. By incorporating optical flow images with the RGB images, higher accuracy is achieved on the ChaLearn249 Isolated Gesture Recognition dataset.

Unlike the previous studies, we offer an action detection and data collecting automation framework for CS: GO. In this study, our goal is to deal with CS: GO dem formatted data,

ensure real-time counting of detected actions, and gather data using Deep neural network and various pre-trained models.

### III. PROPOSED CS: GO REAL TIME ACTION RECOGNITION METHOD

From Section III we get a clear insight of our proposed model which is divided into three parts. In part III-A we talk about our system model and then we move on to part III-B to discuss our data acquisition and pre-processing and finally in part III-C we talk about our used pre-trained and our self-made deep neural network model.

#### A. System Model

Our proposed system model (as depicted in Fig. 5), which allows us to detect various actions from CS: GO is real time is detailed in this section.

We concentrated on gaining data for four main categories: kill, death, smoke and no action which is basically everything else except the first three classes. All of these data were collected by DEM formatted files from one of the leading CS: GO public-site in the world HLTV. Later those DEM files were converted to videos. We made sure that all of the videos had a high-definition resolution of 1280 x 720 pixels and that all of the movies had the same frame rate of 30. Each clip lasted an average of 2 to 5 seconds, signifying the only time frame when the action took place. Secondly, we manually classified all of the videos, which was a time-consuming process but quite effective in assigning the correct labels. We retrieved several frames from each clip after separating our dataset into train, test, and validation segments. Later we transformed those frames into arrays and finally normalized the values that we got from the images. After we are done with the pre-processing part we passed those values through a number of pre-trained models which returned an array of values for each image and later to get better solutions again we normalized those values. We reshaped those data into 1-dimensional shapes to pass them into our model. We attempted to generate outputs that were identical to our one-hot encoded label outputs using our model. In our next step, we compared all of the pre-trained models' scores to find out which pre-trained model performed better.

We have used VGG16, Xception, Inceptionv3, Inception\_ResNetv2 and ResNet152v2 as the transfer learning models to extract the detailed features from the input data and later developed a deep neural network model with five hidden layers to train the feature extracted data. In addition, we have performed majority voting on these five different approaches. Moreover, we have built an API after training our model which enabled us to predict random video frame by frame and later on the basis of our model's prediction we labeled those frames into different folders so that those frames later can be used to train our model more efficiently. In summary, we attempted to develop a system that allows us to combine existing solutions with our own to get superior outcomes. Fig 5 represents our system model.

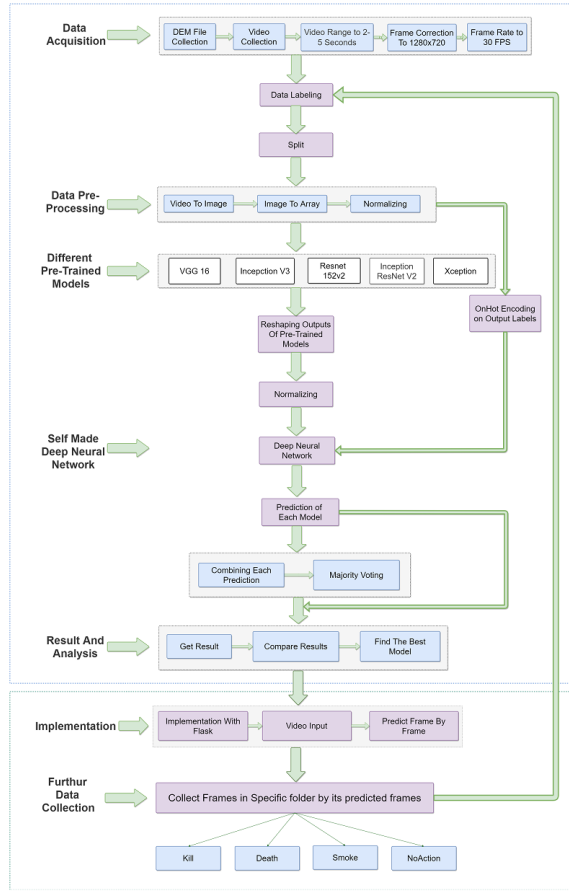


Fig. 5. System Model

### B. Data Acquisition and Preparation

In this section we discuss our data collection and data pre-processing.

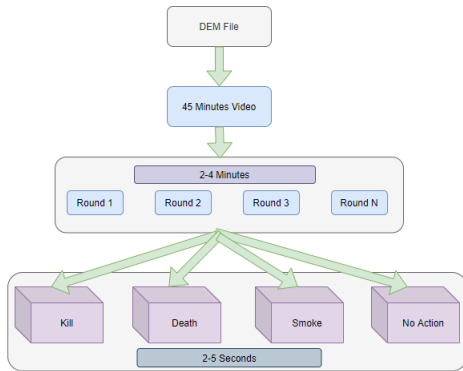


Fig. 6. Data Acquisition

As stated before we collected DEM formatted files which is a game file and only playable by CS: GO game. Each DEM file is about 45 minutes long which contains 30 rounds of game-play. We manually recorded each DEM file with a

Windows screen recorder. Later we cropped it into 30 different rounds to keep track of the rounds. And finally, we cropped individual actions with a duration of 2-5 seconds. Initially, we have collected more than 200 videos for each of the classes. Fig 6 represents our workflow for data acquisition. Videos are just a stack of frames. The term "video" refers to the capture or transmission of moving visual images. Videos, like pictures, have width, height, and depth. We must specify at what speed images will be presented in a clip as video is a representation of visual images. Its frame rate can be defined in fps (frames per second). The number of pictures presented in a video every second is measured in frames per second (fps). Dealing with all frames of a clip might not be the best answer for our problem even though we're trying to anticipate a real-time event, and dealing with each frame could lead to a false hypothesis, and it's a time-consuming procedure. As a result, we divided the frame rate by each frame number and picked only those frames that resulted in the remainder value being set to zero. The entire procedure was carried out with the help of openCV3. Our original frame size was (1280x720). After performing some analysis, we discovered that most pre-trained models use the shape (224,224,3) as input. We transformed our image into this shape during loading frames. We normalized all of the picture arrays after converting them to an array in order to extract additional information from them. We normalized our input shape after extracting features before passing it into the Deep Neural Network model. As previously stated, we have identified four categories of activities in our research which are Kill, death, smoke and no action. Initially, we manually applied labels to each clip. We saved the label with the frame during converting video to picture by adding action to the frame name. To produce output categories, we used a single hot encoding. Based on the labels we previously established, we produced a separate output row for each label and filled it with binary 0 and 1 values.

### C. Model Specification

Within the part, we discuss various pre-trained models that we have used and our self-developed deep neural network model.

**VGG-16 [7]:** It is one of the most commonly used pre-trained models which has more than 138 million parameters. Fig 7 depicts the fundamental VGG-16 structure. We have used the basic architecture for VGG-16 and received an output shape of 7\*7\*512.

**Inception V3 [8]:** The inception model is a very simple yet powerful architectural unit. The key idea behind this model is the inception block. The purpose of this model is to act as a multi-level feature extractor. It was also known as GoogLeNet. Fig 8 shows the architecture of Inception V3. Utilizing inception v3 as our pre-trained model our output shape was 5\*5\*2048

**Resnet 152v2 [9]:** Resnet 152v2 has over 60 million parameters. Fig 9 shows the architecture of Resnet 152v2. With the basic architecture of Resnet 152v2 and input shape as 224\*244\*3 we obtained our output shape as 7\*7\*2048.

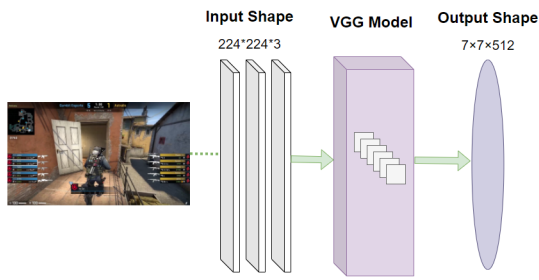


Fig. 7. Model Architecture with VGG-16

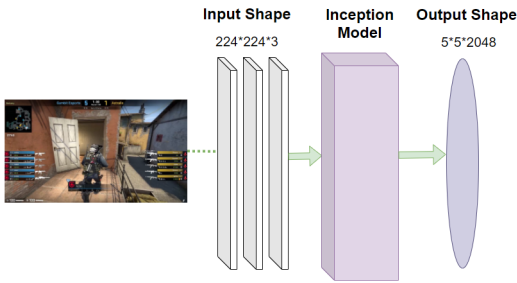


Fig. 8. Basic Model Architecture with Inception V3

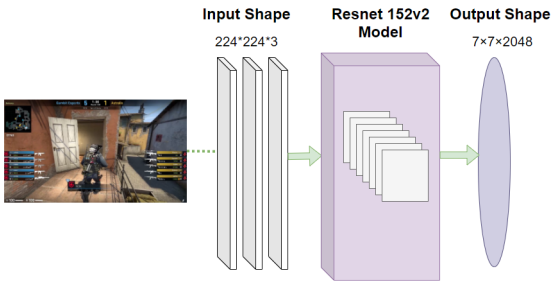


Fig. 9. Model Architecture With Resnet 152v2

**Inception ResNet V2 [10]:** Inception ResNet V2 is a CNN model that is built based on the Inception architecture family however it replaces the Inception architecture's filter concatenation stage with residual connections. Here our output shape was  $5 \times 5 \times 1536$ . Fig 10 depicts the architecture of Inception Resnet 152v2.

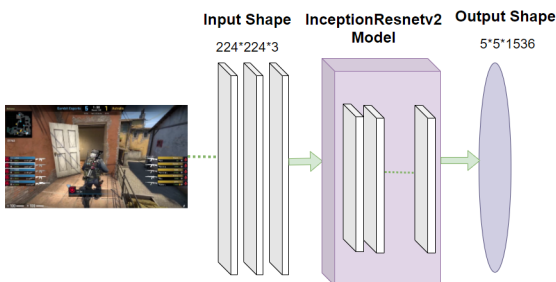


Fig. 10. Basic Model Architecture With Inception ResNet V2

**Xception [11]:** Xception model is also built based on the inception model which is a deep convolutional neural network. This CNN model is divided into three parts which are entry flow, middle flow and exit flow. We have duplicated the basic architecture and got an output shape of  $7 \times 7 \times 2048$ . Fig 11 depicts the architecture of Xception in our system.

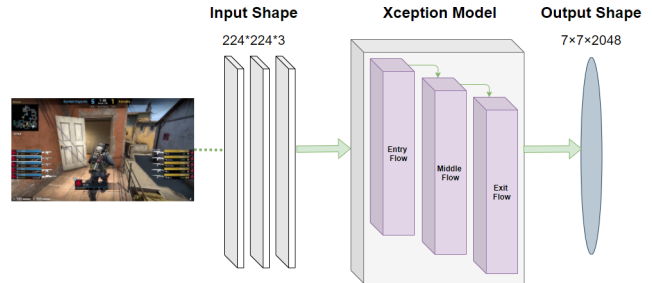


Fig. 11. Basic Model Architecture With Xception

**Self Made Deep Neural Network:** There are five fully connected layers in our Deep Neural Network model. This model's input layer uses the pre-trained model's normalized output value. We utilized four levels of dropout. Dropout is a technique to deal with overfitting. The fundamental idea behind dropout is to remove a unit from the neural network at random during training. We set the dropout rate to 0.5, which ensures that 50 percent of the neurons in that layer are reduced at random every epoch. It means if a fully connected layer had 1024 neurons, and the dropout rate is set at 0.5 then only 512 neurons would be trained in the next fully connected layer. The other 512 neurons are dropped at random. We used two different activation functions in our self-designed Deep Neural Network model. One of the activation functions is ReLU and the other one is softmax. In the output layer, we utilized the softmax activation function to transform actual value to probability. We got more accurate results by using the softmax activation algorithm.

#### D. API Implementation

Under this section we discuss about our API System which is divided into three main components depicted in fig 12.

We have implemented our best fitted algorithm with the aid of flask. Our API system is divided into three parts. In the first phase, we collect user input and then check to see if the FPS is 30. If our input data is not 30 frames per second, we convert to 30 FPS to ensure that we obtain one frame per second and then finally transform our video data into a sequence of frames. We cycle through all of the frames in the following component. As previously stated, in the final layer, which calculates the probability of each action, we utilized softmax as our activation function. We only keep the values in a list, if the probability of each frame's predicted action is more than 0.75. The rest of the frames are omitted. This probability checking ensure us that we can only have flawless predicted frames. In our final part, we cycle through



the predicted frame list ignoring all No Action values. Then we check if our previous predicted value is the same as the predicted value and change the predicted value to No Action if our condition is met. Otherwise, we increment our count for the predicted action. The purpose of this further checking is to ensure that we don't count the same action twice since actions might occasionally last longer than one second, which can be problematic when we're predicting each frame.

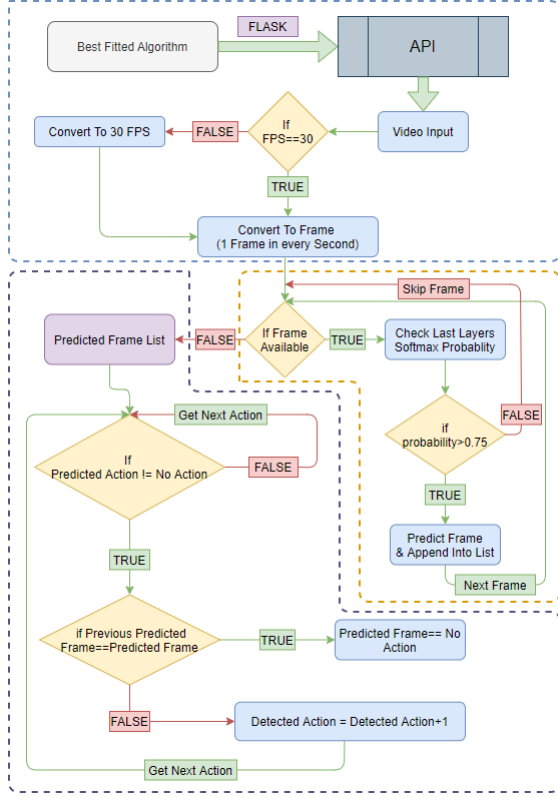


Fig. 12. API System Model

#### IV. PERFORMANCE EVALUATION

In this section, we analyze our overall performance which is divided into two parts. In part IV-A we compare and evaluate our proposed model's performance and in part IV-B we talk about our API's performance.

##### A. System Performance Evaluation

As mentioned earlier, five distinct pre-trained image classification models have been applied for feature extraction and a self-developed deep neural network has been implemented for training and later by combining all the predictions of the models we performed majority voting. Among our implemented individual models for extraction VGG-16 and Inception\_v3 performs better than the rest by providing 90.12% accuracy for both cases. ResNet152\_v2, Inception\_ResNet\_v2 and Xception have relatively less accuracy, but neither of them performed considerably poorly on the dataset and their accuracy value of 88.89%, 87.65% and 86.42% respectively

infers the precedent statement. Moreover, the later inclusion of majority voting exceeds the peak accuracy value of the individual pre-trained models and provides more precise prediction with 92.59% accuracy. Fig. 13 depicts the comparison of the described accuracy value among the different classifiers.

Model	Action	Precision	F-1 Score	Sensitivity	Accuracy
Xception	Kill	0.8421	0.8205	0.80	86.42
	Death	1.00	0.8889	0.80	
	Smoke	0.8695	0.9091	0.9523	
	No Action	0.7826	0.8372	0.90	
Inception_Resnet_V-2	Kill	1.00	0.8235	0.70	87.65
	Death	0.9047	0.9268	0.95	
	Smoke	0.8695	0.9091	0.9523	
	No Action	0.7826	0.8372	0.90	
ResNet_152v2	Kill	0.8421	0.8205	0.80	88.89
	Death	0.95	0.95	0.95	
	Smoke	0.9473	0.90	0.8571	
	No Action	0.8261	0.8837	0.95	
Inception_V-3	Kill	0.89	0.87	0.85	90.12
	Death	0.90	0.93	0.95	
	Smoke	0.87	0.91	0.95	
	No Action	0.94	0.89	0.85	
VGG-16	Kill	0.7826	0.8372	0.90	90.12
	Death	0.95	0.95	0.95	
	Smoke	0.95	0.9268	0.9047	
	No Action	0.9444	0.8947	0.85	
Majority Voting	-	-	-	-	92.59

TABLE I  
MODEL PERFORMANCE

Table I depicts the overall model performance with these accuracy values along with precision, f1 score and sensitivity values for each of the four actions of every pre-trained model. As illustrated from the table, on average the action 'death' achieves the finest performance values for every parameter and 1.00 in precision for Xception as well as 0.95 in all the parameter for VGG16 are some of the examples of that best performance. In addition, Fig 14 illustrates the confusion matrix of majority voting for the four classes. As it can be seen that, the detection of smoke through majority voting is the most accurate as all the 20 out of 20 data example of smoke is detected accurately.

##### B. API Performance Evaluation

Overall performance of our API was exceptional. We are able to correctly count the number of times an action has occurred using our API. As can be seen in Fig 15 our API successfully counts the number of actions in an input video that comprised four deaths and five kills. Here, 0 represents kill, 1 represents death, 2 represents no action, and 3 represents smoke. As there is no smoke in this video, none of the frames are identified as smoke. Initially, 0 appears six times and 1 appears five times, implying that six frames are predicted as kills and five frames are predicted as deaths which means kill count is six and death count is five. However, using our algorithm, this problem is resolved and our API can provide us with the proper count.

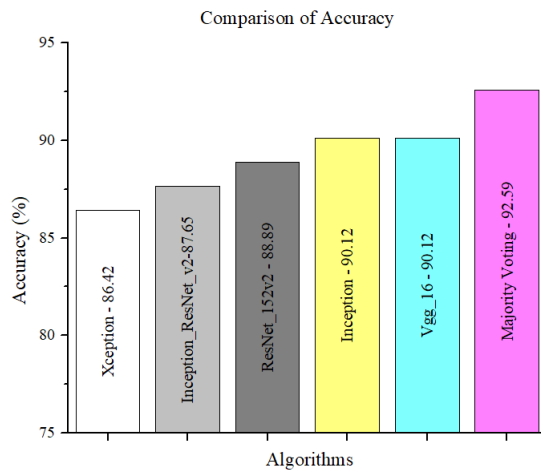


Fig. 13. Majority Voting : System Accuracy

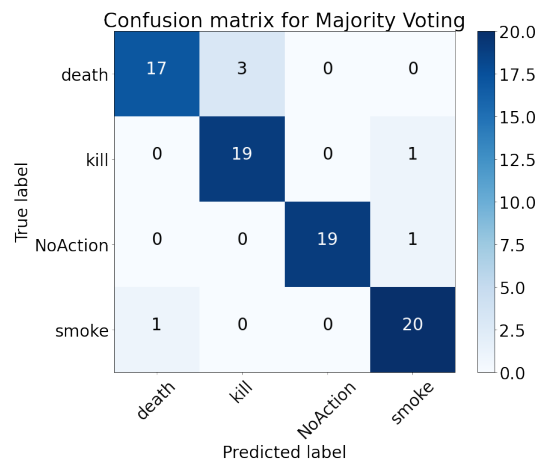


Fig. 14. Majority Voting : Confusion Matrix

[illegible]

Fig. 15. API Performance Evaluation

## V. CONCLUSION

As we can see from the performance evaluation section, the highest accuracy is gained by majority voting (95%), which infers the developed model as quite precise and reliable to predict the 4 different actions. Through our system we will

be able to detect various actions from 45 minutes competitive game-play of CS: GO game and upon detecting actions our system will generate data in real time in CSV format. Later these data can be used for a prediction model which is our one of the end goals of this study. Moreover, through API our system will be able to automate the data collection process that will improve the accuracy and other performance metrics after training on the increased data set. As mentioned in the manuscript, five transfer learning models have been used in the research, whereas there are many more transfer learning models with their own distinctive characteristics. In future, we are also expecting to work with these models and achieve variation in the performance metrics.

## REFERENCES

- [1] Suresh, A. Jeyanthi, and J. Visumathi. "Inception ResNet deep transfer learning model for human action recognition using LSTM." *Materials Today: Proceedings* (2020).
- [2] Abdulazeem, Yousry, et al. "Human Action Recognition based on Transfer Learning Approach." *IEEE Access* (2021).
- [3] Sargano, Allah Bux, et al. "Human action recognition using transfer learning with deep representations." 2017 International joint conference on neural networks (IJCNN). IEEE, 2017.
- [4] Iqbal, Ahsan, Alexander Richard, and Juergen Gall. "Enhancing temporal action localization with transfer learning from action recognition." *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*. 2019.
- [5] Wang, Tian, et al. "Internal transfer learning for improving performance in human action recognition for small datasets." *IEEE Access* 5 (2017): 17627-17633.
- [6] Sarhan, Noha, and Simone Frintrop. "Transfer Learning For Videos: From Action Recognition To Sign Language Recognition." 2020 IEEE International Conference on Image Processing (ICIP). IEEE, 2020.
- [7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
- [8] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* 2016 (pp. 2818-2826)
- [9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- [10] Szegedy, Christian, et al. "Inception-v4, inception-resnet and the impact of residual connections on learning." *Thirty-first AAAI conference on artificial intelligence*. 2017.
- [11] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1800-1807, 2017.