Hand Cricket Game Tracking

Members: Ashwin Suresh Babu (as14091), Varshitha Chennamsetti (vc2209), Zankar Murudkar (zm2117)

Project Description

In this project we employed palm detection and hand landmark classification to construct a hand based game called Hand Cricket. It is fairly similar to the game of cricket. First, the players will play rock, paper and scissors to decide who can make the choice to bat (accumulate points) or bowl (attempt to end the game). Then both the batsman (player) and the bowler (another player) will have to show a number with their fingers at the same time. For example, a player can show 3 fingers to signify 3 points/ runs. The batsman can keep adding this count he shows to his tally until the bowler shows the same count as the batsman. Then the players swap roles and they play again. If the other player exceeds the first player's count, then the other player wins. Otherwise, the other player may get out before matching and may not exceed the first player's tally which results in the first player winning. If both have the same count, the match would be a draw.

Dataset

The dataset was chosen depending of the requirement of parameters i.e. RGB images, RGB + depth, etc. We started with the dataset containing 30k+ hand images with 2D labels for landmarks.

The FreiHand dataset was chosen for training the neural network model. Freihand dataset contains 32560 right hand images with a green background along with 21 keypoint labels for each image and the rest of the training images contain data augmented images generated with different backgrounds. This dataset was used for the training, validation and testing of our Hand pose estimation model.

Figure 1: 21 Hand Keypoints

Methodology

The project will implement Hand Keypoints/Landmark detection with the use of heatmaps to determine the shape of the hand, positions of each finger and overall orientation of the hand.

A hand basically has 21 unique landmarks as illustrated in figure 1. We attempted to classify them based on a trained neural network model.

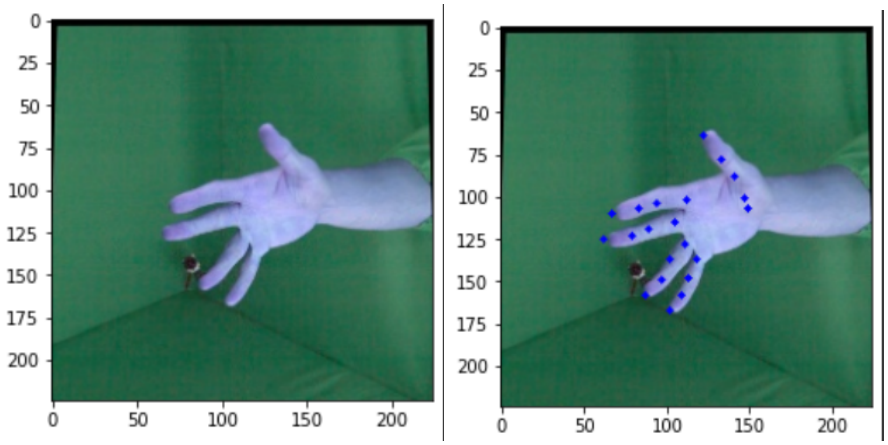

Figure 2: Image vs Image annotated with keypoints

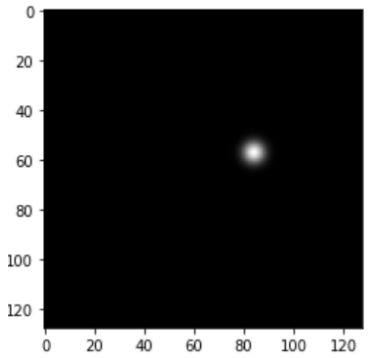

Figure 3: Heatmap created using one point

The entire process contains preprocessing, constructing the neural network, training and testing.

- **Pre-processing:** We initially considered a small number of image paths and landmarks which the system can support processing on. Additional information about the position of the landmarks is given in terms of the intrinsic camera matrix and the 3d coordinates of the landmarks. Since, we only need the 2d coordinates, those points are converted using the camera intrinsic matrix. For each image path in this small batch, we get the image, resize it to match the model input size and they are also normalized. Since, the heatmaps will be used instead of points for the training, heatmaps are generated by taking the 2d points and using gaussian blur at the place where there is a point. This is

done for each point to generate 21 heatmaps. Finally, we standardize the image set based on each channel value.

- **Model:** We used an encoder decoder neural network to retain the context of the image while transforming it to obtain the heatmaps. The output image would be the same size as the input with an additional dimension of heatmaps associated with each image. The pre-processed images from above will be trained with a Neural Network model. A custom model was built using the tensorflow module with the model architecture resembling the UNet. UNet is in the shape of 'U' and it is mainly used for encoding-decoding tasks like segmentation and localization. In our case, we not only need to find whether the landmark is there or not but also the area of the landmark so we utilize UNet. In UNet, there is first downsampling that's done like a normal CNN and then upsampling is done. The contracting or encoding path follows conv_layer1 (with ReLU activation and HeNormal initializer) -> conv_layer2 (with ReLU activation and HeNormal initializer) -> dropout(optional) -> max_pooling(optional) with the number of filters increasing up to 16 times. The expansive path just has a deconvolution layer with the corresponding downsampling output followed by the same set of the convolution layers as the contracting path with a decreasing number of filters by a multiplicative factor of 2 and the output from the corresponding upsampling layers is added. We used a slightly modified version of the IOULoss for the training to support continuous range values of the heatmap instead of the usual discrete values. IoU loss which is the Intersection over union loss used to calculate the amount of overlap between the predicted landmarks and the ground truth. We then added a SGD model with an initial learning rate of 0.1 with this IOULoss. We introduced a reduced learning rate scheduler using ReducedLROnPlateau which divides the learning rate by 2 during a suboptimal convergence. We finally fitted this small image and heatmap data as a numpy array and started the training.

- **Training:** We fitted the training data with the above configurations for 4 epochs. We then saved this model and loaded it for the next subsequent set of image and landmarks data. We repeated the preprocessing step and fit until we covered the entire training set of about 29600 images and the validation loss was bearable. We were able to bring down the validation loss to about 76 percent.

- **Testing:** Our test dataset was tested against the trained model to predict landmarks from heatmaps for the input image. We ran a series of images in which the user points to the number 3 with his fingers. We saw that it predicted correctly 4/9 times consistent with the loss.

The challenge was that the neural network was able to generalize the classification of the game, played by different orientation of the hand, only to a certain extent.

Once the key points for an input image(of a hand) are predicted accurately, we continue with the implementation of the Hand Cricket game rules. Using the landmarks, the shape of the hand is determined and score should be stored for the game to determine the winner. The entire user interface would be in a video where players play using their hands.

- **Game Interface:**

  Video Interface:
  A video interface was introduced to run the prediction model in real time over the frames captured from the video.
  We have used (openCV) cv2 videocapture function to capture video frames and introduce calculated values as well as other game elements in the displayed video.
  Since we did not need the entire area of frames captured for our hand keypoints detection, we used the cv2 rectangle function to establish a rectangular area in the frame which will be sent for further processing. It is important to show hand signs in the designated rectangular area.
  The sliced rectangle was then fed into the model as an image which in turn returned keypoint heatmaps for the input hand image. Acquired heatmaps were used to calculate the hand keypoint coordinates.

  Finger Count:
  Using the coordinates for each finger and its joints, we found the number of fingers displayed.
  The formula used is as follows:

  ```
  if(coordinates[finger-tip] < coordinates[knuckle joint])
  ```

  We calculated the number of fingers which satisfy the above condition.

  Tackling higher loss:
  The loss rate for the model is high which may affect the results for a certain number of frames. Using these results as the final result may affect the prediction and game values. To tackle this problem, we used a set of frames(between 20-70 frames). We took the result(finger count) which appeared for the maximum number of frames and passed it as the final result for further processing. For now we will be supporting only for the index, middle and ring finger of the right hand up to a maximum of 3.

  This number was sent back to the calling function and stored as a number played by the player.

  Game Logic:
  We have designed a Human vs Bot Hand Cricket game.
  The game starts with the player batting first(because humans are superior to bots).
  The finger count received by the model is compared against a random number(1-5) generated by the bot.
  If finger count == bot random number: the player is out and it is the bot's turn to bat.
  If finger count is not equal to the bot's random number: count is added to the player score.

  Same score accumulation rules apply to the bot as well.
  The contender with the highest accumulated score wins the game.

Two Player:

Game can accommodate 2 players to play against each other at the same time.

To accomplish this, we can establish a second rectangle zone to detect the second player's hand.

Finger count is calculated separately for the 2 rectangular hand regions and compared against each other.

Results

Final goal of the project is to estimate unique key points on the hand from a live video and process the data to result in estimated hand pose and delegated number of points for the game. The game may run for multiple iterations of keypoints estimation and score calculation. For example, if a user points 2 fingers, the score would be able to deduce it as 2 and would use the score for further comparison and calculation.

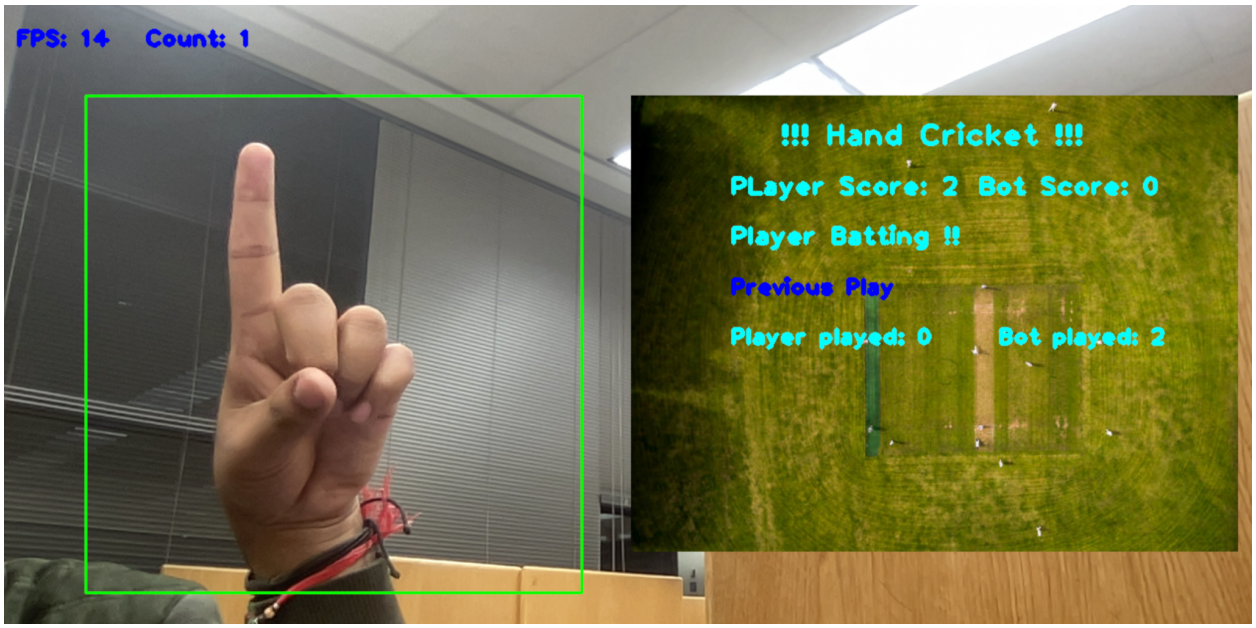

Figure 4 : Game interface

The validation loss achieved through training the Unet is 76 percent but through the results it can be seen that it didn't need to be as accurate in order to predict the hand gesture. This is because the Unet is only finding the landmarks. The landmarks need not be in a highly accurate position in order for us to recognize the hand gesture.

As it can be seen through the game interface, the video is captured and using those different frames' images, the gesture is classified based on an average of classification done by them. And it is also able to classify the hand gesture with a non-neutral background. On the left corner

of figure 4, you are able to see the current count. And on the right, the game score is being tracked.

In conclusion, the game interface and score tracking logic is built. The model gives very less validation loss in classifying the landmarks but that didn't stop the final gesture classifier in order to recognize the gesture in different environments.

Discussion

The main challenge that was observed was the training time it took. Even after using the tensorflow gpu support, the dataset preparation and model training took a large amount of time. Eventually only a few images were used at a time to train the model. Even Though, the training loss is very high, it is not doing a bad job classifying the hand gesture because our main goal is not to get the landmarks but to classify the gesture. So getting the high level features from the neural network and getting the classification as an average of all the frames is classifying the gesture accurately.

The strength of this proposed methodology is that we do not need a very small training loss to classify the gesture accurately. The weakness of this method is it is computationally expensive to even train 2000 images.

Future work

Because of the computational complexity of the network, the time taken to train is huge. With ample amount of time and computational power, the training can be fit to a lot of data and the test accuracy can be improved. Transfer learning using a trained model can also be explored to get a good test accuracy with very few images.

References

1. https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726
2. https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5
3. https://gogul.dev/software/hand-gesture-recognition-p1
4. https://towardsdatascience.com/gentle-introduction-to-2d-hand-pose-estimation-approach-explained-4348d6d79b11
5. https://www.blog.pythonlibrary.org/2021/02/23/drawing-shapes-on-images-with-python-and-pillow/