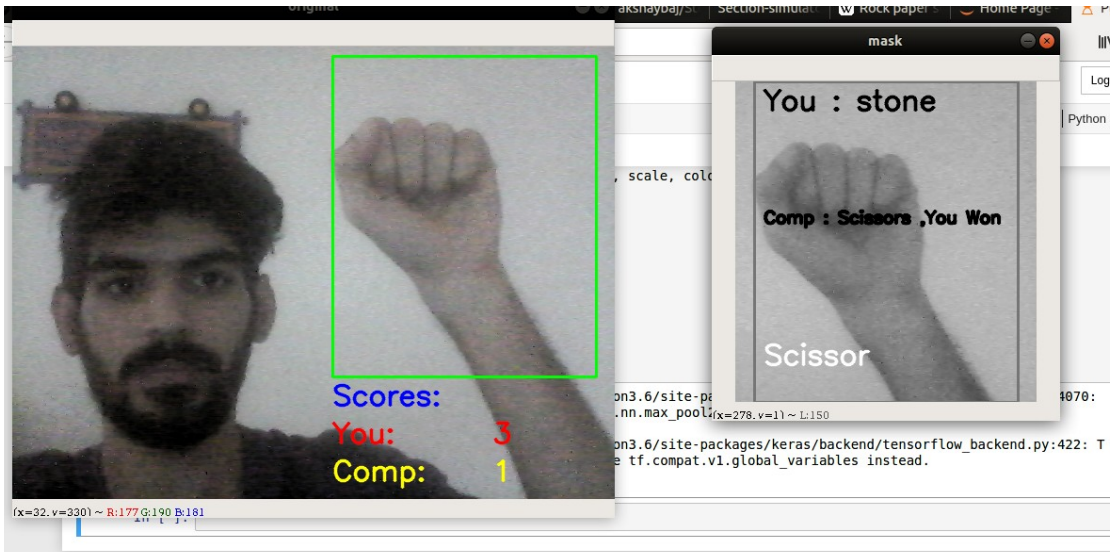
Rock Paper Scissors Game Using CNN
Akshay Bajpai , 193079002 , EE769

Introduction -

Rock paper scissors (also known by other permutations such as scissors paper rock, or ro-sham-bo) is a hand game usually played between two people, in which each player simultaneously forms one of three shapes with an outstretched hand. These shapes are "rock" (a closed fist), "paper" (a flat hand), and "scissors" (a fist with the index finger and middle finger extended, forming a V). **(Source : Wikipedia)**

The aim of building this small project was to eliminate the requirement of a second player and enabling a user to play the game with the computer using his webcam. The game has only two possible outcomes which are either a draw or a win for one and loss for the other player . As the problem involves recognition of images on webcam in real time, a CNN model was trained to detect the shapes of rock,paper or scissors. The final outcome screenshot is given below , where we can see that the user is gesturing one of the three options using his hand and the computer generates a move randomly. As per the rules of the game the winner is decided and the score gets incremented accordingly.

Concepts applied – OpenCV , CNN , Transfer Learning



Challenges faced -

There were two major challenges faced while building this project and both of them were situational (Lockdown owing to Covid-19). But in order to overcome these two challenges, I was able to learn and apply transfer learning .The first challenge was the lack of availability of good internet connection so as to download good data set for training. The second one being low specifications of my laptop which would have increased the training time considerably if the model would have been trained from scratch.

- ✔ To overcome the second challenge, we searched for pre-trained models for detecting hand gestures . One such model was found on github . (https://github.com/filipefborba/HandRecognition) . The goal of the project in this repository was to train a Machine Learning algorithm capable of classification images of different hand gestures, such as a fist, palm, showing the thumb, and others. This was

chosen as the starting point . The dense layers of these models were removed and new dense layers were added for training.

- ✔ To overcome the first challenge, initially I took images of my own hand gesturing rock,paper and scissors on the webcam . A total of 3000 images (1000 of each category) were taken and stored as the training data set. The problem with this method was the lack of variation in the data set as the images were of the hand of the same person.

Algorithm -

Below is the summary of the pre-trained model downloaded from github. Initially we removed only the dense layers of the model and added new dense layers which were trainable. Upon training with the data set, the accuracy achieved was very low (close to 32%) . The possible reason behind the low accuracy may be that the initial data set on which our downloaded model was trained did not have any image of rock (gesture of rock using hand) or scissors . Thus the initial convolutional layers did not capture any relevant features which might help to detect these two cases.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 58, 158, 32)       832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 29, 79, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 27, 77, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 38, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 11, 36, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 18, 64)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 5760)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               737408
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 794,954
Trainable params: 794,954
Non-trainable params: 0
```

To overcome this, the last convolutional layer in our pre-trained model was also converted to “trainable”. Once this was done , we achieved a much higher accuracy (close to 94%) . Below is the screenshot of the model we used for training.

The first screenshot of the model retrieved from the pre-trained model **(model_1)**. As we can see , the dense layers have been removed and the total trainable parameters are 36,928 which are the total parameters in the last convolutional layer.

The second screenshot is of the model_1 plus the new dense layers which are all trainable. (**sequential_1)**

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1_input (InputLayer)  (None, 120, 320, 1)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 58, 158, 32)       832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 29, 79, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 27, 77, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 38, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 11, 36, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 18, 64)         0
=================================================================
Total params: 56,256
Trainable params: 36,928
Non-trainable params: 19,328
```

Fig : Pre trained model with only last layer trainable

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model_1 (Model)              (None, 5, 18, 64)         56256
_________________________________________________________________
flatten_1 (Flatten)          (None, 5760)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                57610
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 33
=================================================================
Total params: 113,899
Trainable params: 94,571
Non-trainable params: 19,328
_________________________________________________________________
```

Fig : Final model used for training after adding dense layers

All the input images to the model were resized to be of shape (120,320) as this was set by our pre-trained model. There was not much image per-processing performed apart from converting the images from RGB to Grayscale and resizing them.
Activation used for the dense layers were sigmoid and softmax for the final output layer as it was a multi class problem.

Training Model-

The model was compiled using ‘adam’ optimizer and ‘sparse_categorical_crossentropy’ loss function was optimized. To improve upon the accuracy , two new data sets for rock,paper,scissors was downloaded from kaggle. Below are the links for the same.

1. https://www.kaggle.com/drgfreeman/rockpaperscissors
2. https://www.kaggle.com/sanikamal/rock-paper-scissors-dataset

The files within the data set had to be renamed and stored in the ‘data’ directory . This was done using a python code as the name of the files for each category had to be systematic. The naming convention of file followed was – [label]_frame_[img_number] . Here, label corresponds to the label of the image (0 - scissors , 1- stone , 2- paper) and img_number is the image number of that particular label. The naming convention is important as the label of the image is extracted from the file name in the python code so as to create the data set (X) and the corresponding labels (y) for training the model.
After adding these two downloaded data set, the total images for training were 11008 (roughly 3500 for each category) . The data set was split into training and testing with 9907 images for training and 1101 images set aside for testing. The model was trained again, this time achieving a higher accuracy of close to 97% and also capturing more variation.

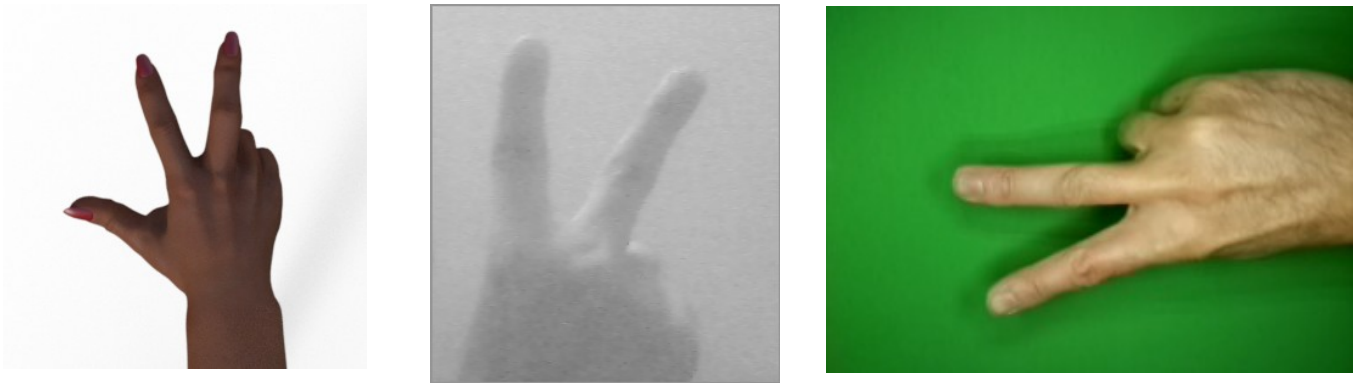


Fig: Images from “Scissors” data set – Label 0

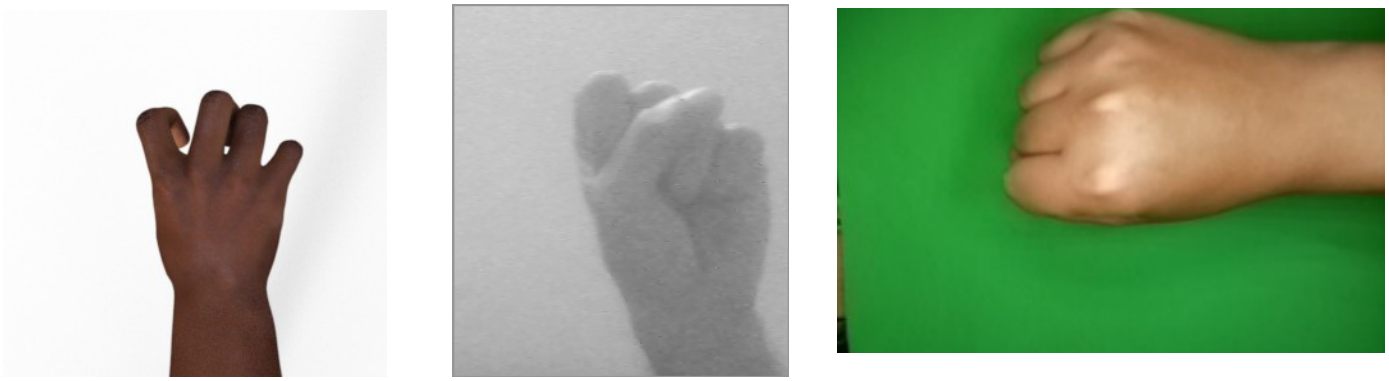


Fig: Images from “Stone” data set – Label 1

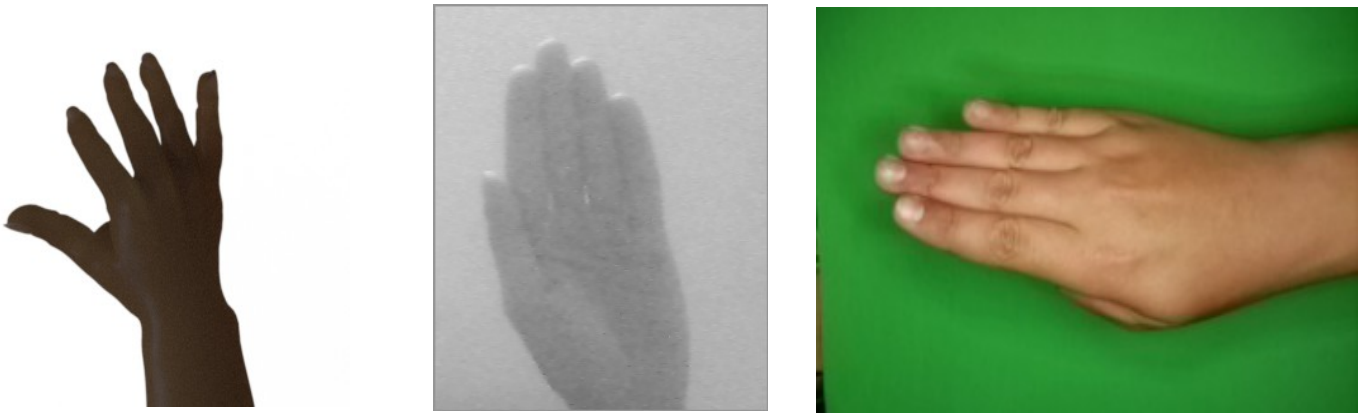
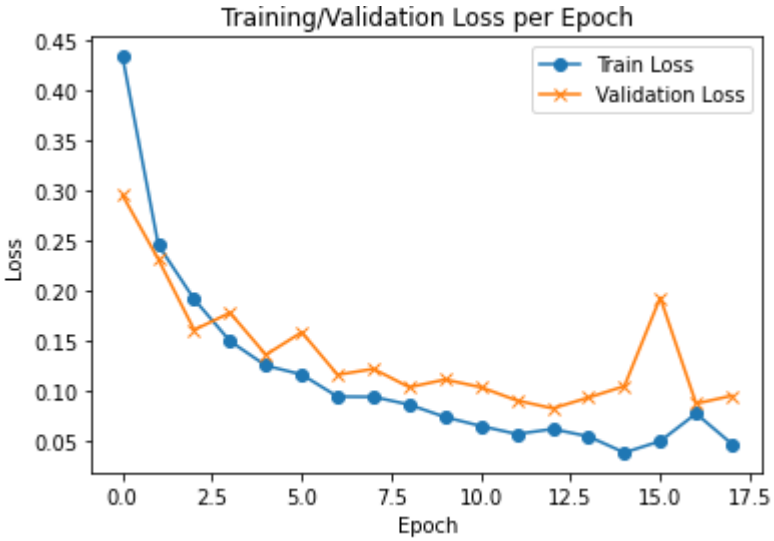
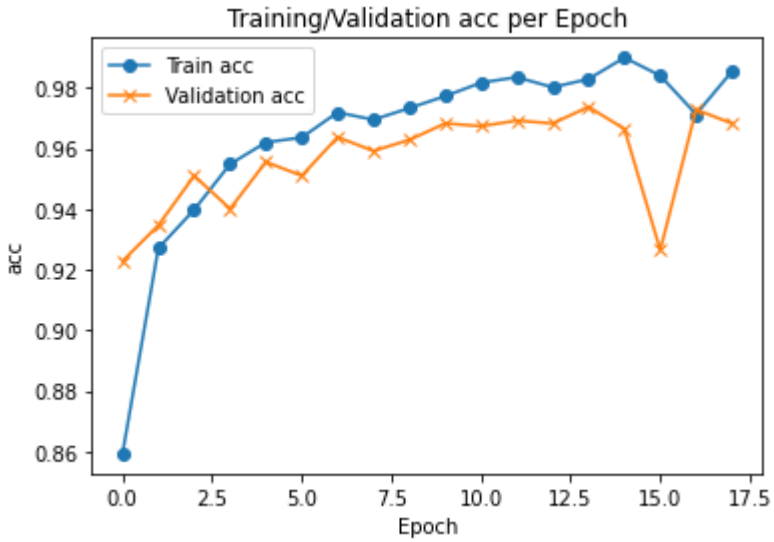


Fig: Images from “Paper” data set – Label 2

Evaluation of the Model -

Post training the model was evaluated using the accuracy metrics, confusion matrix and the classification report. Also a plot of training/validation accuracy and loss with the number of epochs was drawn to visualize the process of training.





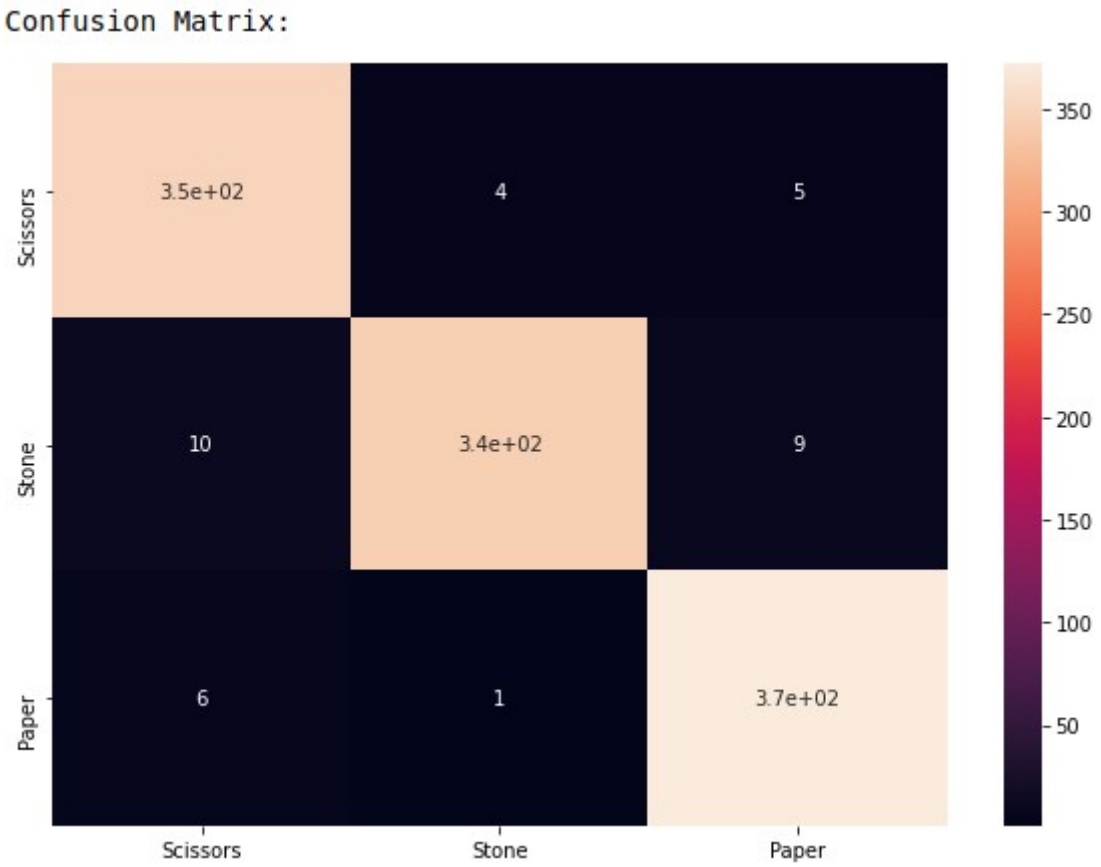
Below are the classification report and confusion matrix to understand the accuracy of our model better -

```
Accuracy score on validation set :  0.9682107175295186

Claasification report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.97       359
           1       0.99      0.95      0.97       363
           2       0.96      0.98      0.97       379

    accuracy                           0.97      1101
   macro avg       0.97      0.97      0.97      1101
weighted avg       0.97      0.97      0.97      1101
```



Places for improvement -

1. The accuracy and generalization of the model can be further improved either by adding more data to the data set or by using image augmentation techniques.
2. As of now , the user needs to place his hand in the specific location within the video capture. This can be eliminated if we can use image segmentation technique and detect user's hand anywhere within the video capture area.
3. The game gets a bit slow as of now as to capture the image correctly the user needs to place his hand in the region of interest a bit earlier than the computer . This can be improved if we apply more image pre-processing techniques so that even blurry frames can be processed and detected correctly.

Github Repository - https://github.com/akshaybaj/Stone_Paper_Scissors