Baseball Faces

Introduction

Humans have a great capability when it comes to analyzing the face and providing the raw judgement based on facial features and the expressions. But do Machines possess the same capabilities when it comes to analyzing the facial looks and features? At first, we had the same doubt on machines on this matter. But we are very hopeful at this point that if we apply the optimal machine learning algorithm to train the images against their features, we would definitely come up with the best predictive model. We need to predict the Trustworthy scores and Competency score of images by training our model on over 800 images. Each Image have the ratings ranging from 0 to 10. Each player has the ratings based on competency and trustworthiness of a player as per the survey received from people based on different interests.

Data

We had over 800 images with their Trustworthy scores and Competency scores which was rated by the students of MSA through a survey. The data set was unbiased as the dataset received equal ratings received as received by survey. Whole dataset was divided by using SkLearn library with train test split function, dividing the data set into train test data set. On applying this function, we receive the following output:

Train_x = The image data which have 640 images to be trained

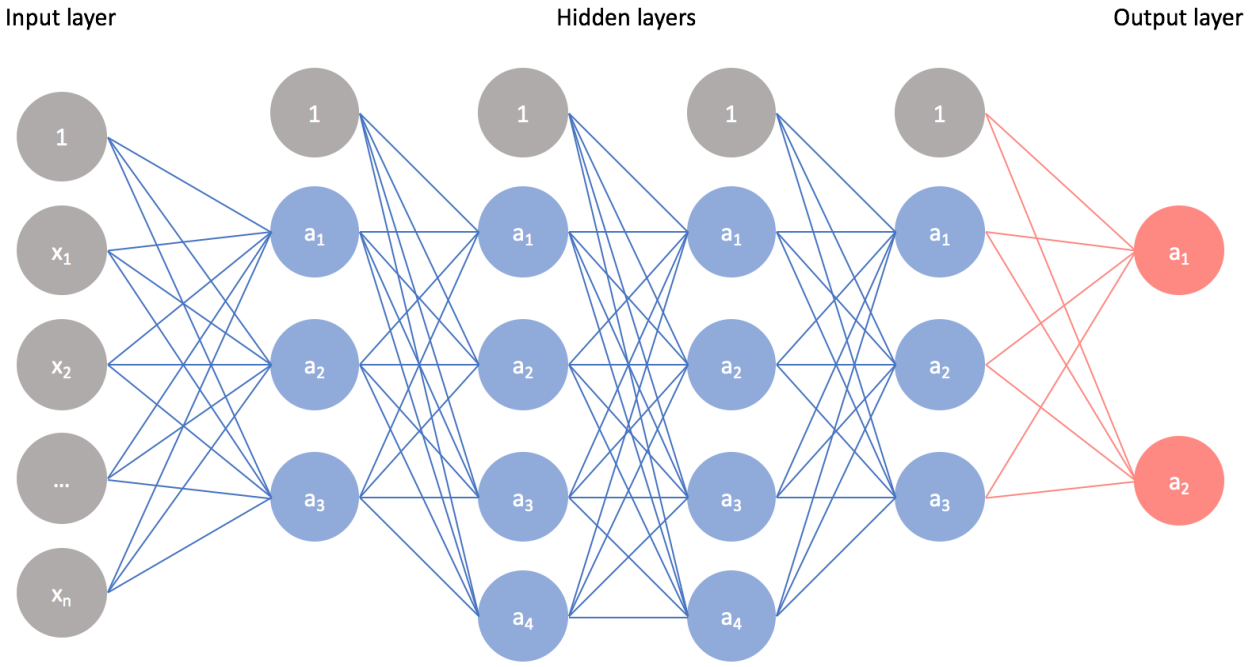
Train_y = Data set of ‘competency score’ and similarly for ‘Trustworthy score’

Valid_x = The image data which have 160 images to be tested according to the model fitted using 640 images

Valid_y= Data set of ‘competency score’ and similarly for ‘Trustworthy score’ for validating the trained model

- **Methodology**

We used CNN to classify our images based on the historical Trustworthy score and Competency scores.



A1 and a2 can be considered as the Trustworthy and competitive score respectively

Below are the steps we followed

1. *Imported all the training and testing images in a numpy array.*

2. *Split the dataset into training and validation dataset.*

3. *Building the model*

   The basic building block of a neural network is the *layer*. A layer extracts a representation from the data fed into it. Hopefully, a series of connected layers results in a representation that is meaningful for the problem at hand. Much of deep learning consists of chaining together simple layers. Most layers, like **tf.keras.layers.Dense,** have internal parameters which are adjusted ("learned") during training

   "convolutions" tf.keras.layers.Conv2D and MaxPooling2D - Network start with a pair of Conv/MaxPool. The first layer is a Conv2D filters (3,3) being applied to the input image, retaining the original image size by using padding, and creating 32 output (convoluted) images (so this layer creates 32 convoluted images of the same size as input). After that, the 32 outputs are reduced in size using a MaxPooling2D (2,2) with a stride of 2. The next Conv2D also has a (3,3) kernel, takes the 32 images as input and again creates 32 outputs which are again reduced in size by a MaxPooling2D layer.

   We follow the same step for few more times where we changed the output of the hidden layer to 128 which will be the output for our output layer.

output tf.keras.layers.Dense - A 128-neuron, followed by 11-node relu layer.

4. *Compile the model*

   Before the model is ready for training, it needs a few more settings. These are added during the model's compile step:

   Loss function - An algorithm for measuring how far the model's outputs are from the desired output. The goal of training is this measures loss.

   Optimizer - An algorithm for adjusting the inner parameters of the model in order to minimize loss.

   Metrics - Used to monitor the training and testing steps. The following example uses *accuracy*, the fraction of the images that are correctly classified.

5. *Train the model*

   Training is performed by calling the **model.fit** method. Feed the training data to the model using training dataset. The model learns to associate images and labels. The epochs=30 parameter limits training to 30 full iterations of the training dataset.

6. *Predicting the scores*

   model.predict_classes will predict the values by our trained model.

   Below is the output (top 10) we got for the test dataset that was provided:

Competency_Score	Trustworthy_Score
8	7
4	6
7	4
7	4
8	7
6	5
4	5
7	4
4	4
5	6

- ## Conclusion

  We gained the output from our model which were convincing enough and have the error rate of 3. That means the predicted values lies between the range of 3. The model could have performed better if we would have used the ratings as the numeric values. That can be

considered as the scope of improvement in future. From our initial assumption that machine can learn nearly same as humans and the results gained from our predictive model, we can conclude that we need a lot of research and trail and runs if we want to achieve the strengths same as that of human capacity.

- **References**:
  - Dr. Aghasi's Notes
  - www.towardsdatascience.com
  - https://github.com/
  - https://www.analyticsvidhya.com/
  - https://www.youtube.com/
  - https://stackoverflow.com/
  - https://www.geeksforgeeks.org/