SoCal FC
Engineering Design Document

Authors:
Rishika Verma
Shelly Mehta
Vandit Maheshwari
Puranjay Rajvanshi
Sangeeth Koratten
Aviral Upadhyay

Date: 12/02/2020

Table of Contents

Introduction

Background

Our project, SoCAL FC, is an attempt to create an artificially intelligent bot which will play the game of FIFA.

FIFA

FIFA is a series of association football simulation games developed and maintained by Electronic Arts (EA Sports). FIFA franchise is localized in 18 languages and made available in 51 countries making it the best-selling sports video game franchise in the world. The term FIFA stands for Fédération Internationale de Football Association. It was founded in 1904 to oversee international competition among the national associations of 8 countries in the beginning, and now contains 6 confederations and 211 national associations.

Soccer or Football is a highly tactical game with many rules associated with it. It's a team sport played with a spherical ball between two teams. Played by approximately 250 million players in over 200 countries, makes it the world's most popular sport. Explaining the game in depth can be a bit hectic but to get a basic understanding we say that the game is played on a rectangular field called a pitch with a goal at each end. Each team comprises of 11 players (excluding substitutes), one of which is a goalkeeper. Players are not allowed to touch the ball with hands or arms while it is in play except the goalkeeper. Other players mainly use their feet to strike or pass the ball hence the name "foot-ball". The objective of the game is to outscore your opponents by moving the ball past the opposing goal line. Although it may sound simple but all the above mentioned are accompanied by a set of rules and restriction making it more than a game of athletics and strength.

Purpose

Artificially intelligent bots in a game can be built by hand coding a bunch of rules that impart game intelligence. This approach successfully imitates the playing part but will not be able to imitate human like behavior. Thus, as football lovers we can easily tell apart a bot from a human.

Hence, the motivation behind the project. We want to create an artificially intelligent bot which can figure out the game by making it learn from humans playing the game.

Exploring this requires us to play FIFA in order to collect data of humans playing the game ahead of training the bot. While playing we record our in-game actions and decisions which in turn will allow us to train an end-to-end Deep Learning bot without hard coding the rules.

Goal

With growing AI in every sphere and bots being created to replicate every human action with the same or better performance level, the urge to do the same for the game FIFA arose. The intent is to create an artificial intelligent bot which can master behavior-cloning and perfectly replicate the actions performed by a player with a gaming controller or even perform better than a human would do. Performing better would mean choosing actions on the controller which would yield to better results like higher scores. The initial objective is for the bot to correctly identify articles on the field (like other players, the ball, the goalpost) and correctly choose actions depending upon the situation of the game (like when to pass the ball, when to shoot). With humans playing, there is always a chance of making mistakes or fumbles in a game like FIFA. The desire is to reduce those cases since chance of human error is out of the picture. The FIFA bot should eliminate the situations where human makes mistakes, thus increasing the accuracy of actions. This would improve the team's playing execution, thereby leading to increased number of goals and elevated chances to winning the game.

Prior Research

Behavioral cloning: It is the first and foremost approach to address the challenge of creating an artificial intelligent bot. Instead of providing a set of rules or actions by experts, we simply let the model train itself by imitating a human playing the game.
Behavioral cloning is a form of learning by imitation whose main purpose is to build a machine learning model or an agent which can imitate behavior of a human performing some action or complex skill such as playing soccer.
Hence, behavioral cloning is a method by which human sub cognitive skills can be captured and reproduced in a computer program. While in action, we record the decisions made by human in different situations and then present these as input to the learning bot. This results in a set of rules which are to be followed by the agent to reproduce the skilled behavior as closely as possible.

Object detection: It is a computer vision technology which is used to identify and locate numerous objects within an image or video. Object detection draws bounding boxes around these detected items, thus allowing us to locate and perform actions with those objects.
Object detection in our game helps us to track various objects such as players, ball, field, referee. This technique is vital to track the physical movement of the different characters in the game and to track other moving objects such as the ball while simulating the soccer game.
This approach uses a descriptive model of an object class that is rich enough to effectively model any of the possible shapes, poses, colors, and textures of an object but it is general enough to be transferred to a new class of objects thus, classify so many different objects on a soccer field.
We are using an example-based learning approach where a model of an object class is derived implicitly from a set of training images. This ensures that our model utilizes this algorithm which is specialized to a specific domain related to our surroundings.

Single shot detection: One way to imagine single shot detection is thinking advanced object detection algorithm. This technique takes a single shot to detect multiple objects present in an image using multibox. It provides significantly faster speed and higher accuracy while detecting multiple objects.
Higher accuracy for this algorithm is the result of using multiple boxes or filters with different sizes, and aspect ratio for object detection. It applies these filters to multiple feature maps not only in the beginning but also in the later stages of a network.
At the time of prediction, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Since, it completely eliminates proposal generation and subsequent pixel or feature resampling stages and encapsulates all computation in a single network, it is easy to train and integrate into the system.

Convolutional Neural Network (CNN): CNN is a machine learning algorithm used to implement image recognition and image classification. An image is given as input to CNN which in turn processes and then classifies it.
The model takes each input image and passes it through a series of convolutional layers with filters (Kernels), Pooling, fully connected layers and then apply SoftMax function to classify an object with some probabilistic values between 0 and 1.
Figure 1 represents the complete flow of CNN to process an input image and perform classification based on the values.

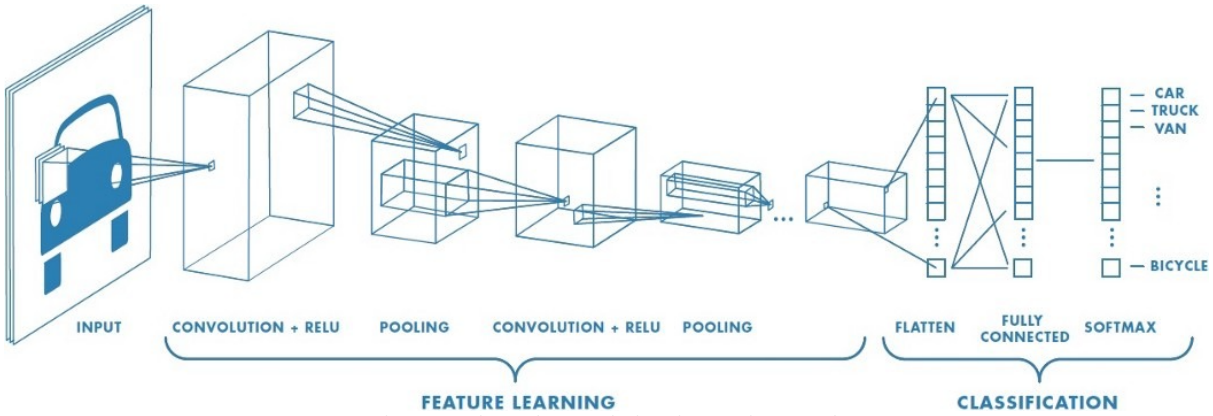


Figure 1: Flow of Convolutional Neural Network

Mobile net ssd: MobileNet are convolutional neural network architectures whose number of trainable parameters can be controlled by two hyperparameters – width and resolution multiplier.
It can be visualized as a cross trained model from single shot detection to MobileNet architecture. Although using MobileNet instead of single shot detection decreases average detection accuracy but it provides more speed relatively. It also introduces linear bottleneck layers in the network, reducing the input size in subsequent layers while preventing information loss on the side. Thus, using MobileNet architecture is key while designing a game playing intelligent agent keeping in mind the constrained hardware settings. Since, CPU performance mainly affects the performance of the game, MobileNet architecture provides us an edge over performance vs CPU with respect to performance vs GPU.

Long short-term memory(LSTM):
Long short-term memory is an artificial recurrent neural network architecture used in the field of artificial intelligence and deep learning. Unlike other feedforward neural networks, it has feedback connections which helps user to process not only single data points such as images but also entire sequences of data such as video, speech etc. The most common example where LSTM is applicable is speech and handwriting recognition.

Timeline/Milestones

The timeline of the project is depicted in Figure 2. The entire duration of 14 weeks was partitioned on the basis of time and resource requirement for each phase. The initial weeks were spent on formation of a team and working on ideas for the project.

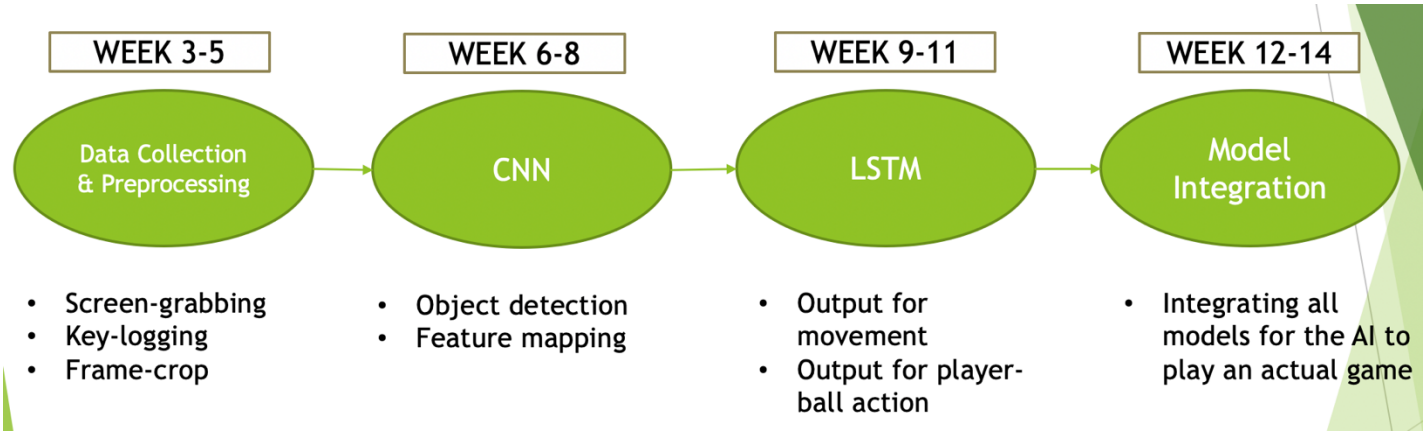


Figure 2: Timeline for the project

- **Week 1 – 2:** Team formation, brainstorming for ideas for the project and reading literature surveys.
- **Week 3 – 5:** Collecting input images as screenshots of while playing FIFA. Capturing key – logs for every action taken at the instance of the screen grab.
- **Week 6 – 8:** Implementing object detection on the input images to get bounding boxes on each object in each image. Getting feature maps for the images.
- **Week 9 – 11:** Training two parallel LSTM networks to obtain movement and action on the ball as output.
- **Week 12 – 14:** Integrating all the pieces of the model to make a working agent-based FIFA game.

Methodology

A. Feature Extraction using CNN:

Data Collection and Preprocessing

For the purpose of the project the model had to be trained on different images of the game environment. To fulfill this, the steps mentioned below were followed:

1. Setting up the environment: Downloaded FIFA18 Desktop version on the system.
2. Collect screenshots: Grabbed screenshots of the environment while playing lots of FIFA. The screenshots were stored in .jpg format.
3. Key logging: Key-press simulation is used to communicate the action (output) that needs to be taken.
4. Image cropping to obtain desired frame of the game window.

The dataset was split into 70%-30% for training and testing datasets.

```
def grab_screen(region=None):
    hwin = win32gui.GetDesktopWindow()

    if region:
        left, top, x2, y2 = region
        width = x2 - left + 1
        height = y2 - top + 1
    else:
        width = win32api.GetSystemMetrics(win32con.SM_CXVIRTUALSCREEN)
        height = win32api.GetSystemMetrics(win32con.SM_CYVIRTUALSCREEN)
        left = win32api.GetSystemMetrics(win32con.SM_XVIRTUALSCREEN)
        top = win32api.GetSystemMetrics(win32con.SM_YVIRTUALSCREEN)

    hwindc = win32gui.GetWindowDC(hwin)
    srcdc = win32ui.CreateDCFromHandle(hwindc)
    memdc = srcdc.CreateCompatibleDC()
    bmp = win32ui.CreateBitmap()
    bmp.CreateCompatibleBitmap(srcdc, width, height)
    memdc.SelectObject(bmp)
    memdc.BitBlt((0, 0), (width, height), srcdc, (left, top), win32con.SRCCOPY)

    signedIntsArray = bmp.GetBitmapBits(True)
    img = np.fromstring(signedIntsArray, dtype='uint8')
    img.shape = (height, width, 4)

    srcdc.DeleteDC()
    memdc.DeleteDC()
    win32gui.ReleaseDC(hwin, hwindc)
    win32gui.DeleteObject(bmp.GetHandle())

    return cv2.cvtColor(img, cv2.COLOR_BGRA2BGR)
```

Method to grab screen

Object Detection

The next step was detecting objects like players, ball or goalpost in each game window. The Object Detection API of TensorFlow was used to get the bounding boxes for each object on all the images.

1. PIP installed labelImg library in the TensorFlow folder from the terminal.
2. Start labelImg, pointing to the training images folder.
3. A File Explorer Dialog window should open, which points to the training images folder. (Figure 3)

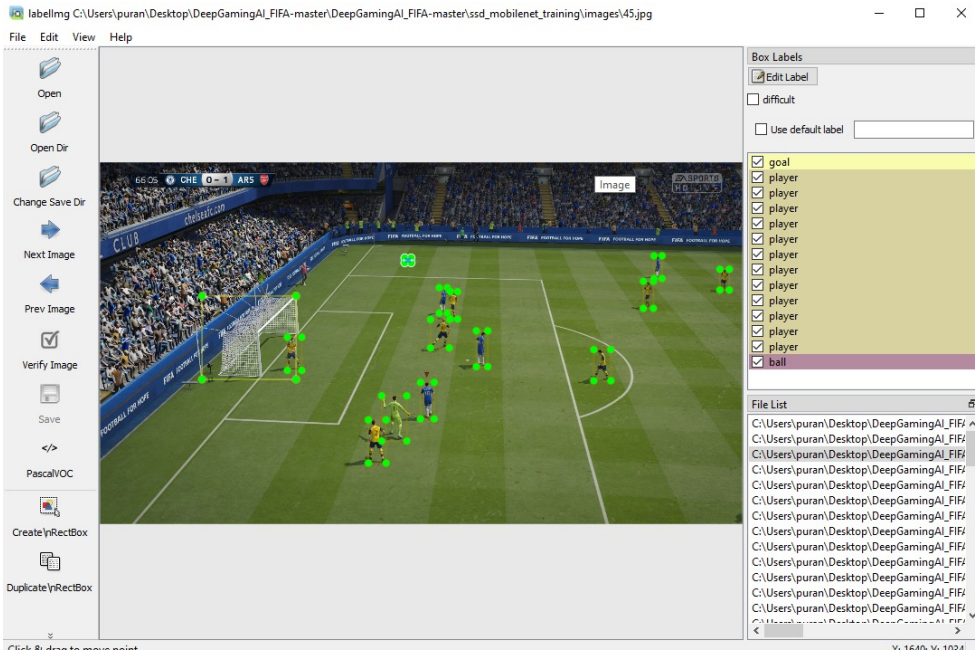


Figure 3: Drawing bounding boxes around objects

4. Click on “Select Folder” button and annotate the images.
5. Drag boxes around all the objects (players, ball, post) for annotating the positions.
6. Annotation will create *.xml files for each image inside the training images folder. Figure 4a shows the .xml file for the image in Figure 4b.

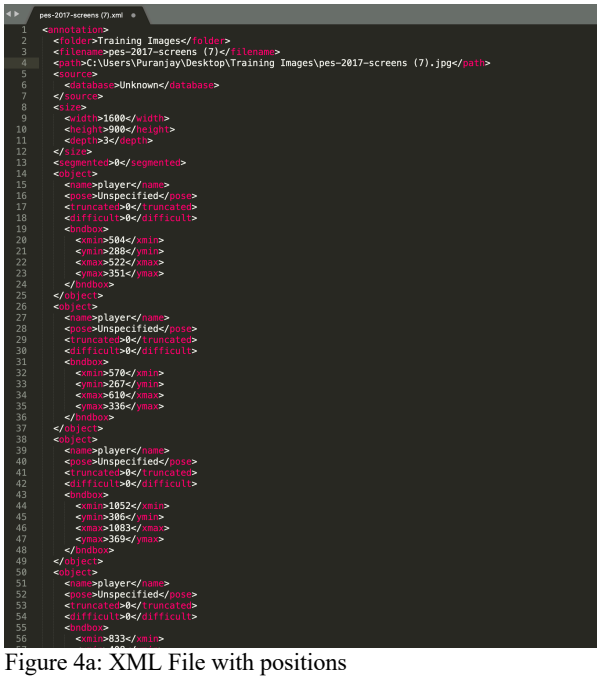


Figure 4a: XML File with positions

Figure 4b: Screenshot of game window

7. Converted all the .xml annotation files to .csv format.
8. Generated tfrecord files for each input image to be passed as input to the CNN model.

Transfer Learning (MobileNet)

Implemented Transfer Learning on MobileNet CNN. MobileNet is already trained and is a part of the Object Detection API.

1. The tfrecords obtained from the previous step is passed as input to the MobileNet model.
2. The output layer is updated to accommodate the output for the given training image dataset.
3. Hyperparameters were tuned to obtain better results on the dev dataset.
4. The CNN model will help make accurate bounding boxes for the test images.
5. This result of images with bounding boxes will be passed to the LSTM network to train on the action to be taken for the given game window.

```
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.compat.v1.GraphDef()
    with tf.compat.v2.io.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES,
                                                            use_display_name=True)
category_index = label_map_util.create_category_index(categories)
```

Loading the CNN model

```
with detection_graph.as_default():
    with tf.compat.v1.Session(graph=detection_graph) as sess:
        image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
        feature_vector = detection_graph.get_tensor_by_name(
            "FeatureExtractor/MobilenetV1/Conv2d_13_pointwise_2_Conv2d_5_3x3_s2_128/Relu6:0")
        for i in range(0, steps_of_history):
            screen = grab_screen(region=None)
            screen = screen[20:1000, :1910]
            image_np = cv2.resize(screen, (900, 400))
            image_np_expanded = np.expand_dims(image_np, axis=0)

            rep = sess.run([feature_vector], feed_dict={image_tensor: image_np_expanded})
            input_window[i, :] = np.array(rep).reshape(-1, 128)
```

Calling MobileNet SSD for inference

B. Action selection using LSTM:

Data Collection and Preprocessing

Time-series data is required for selecting the next action to be taken. Using the CNN model, 10 images at continuous time-steps for a given time instance are recorded and 128-bit feature vector of these images is passed to the LSTM model. Two parallel LSTMs are used to obtain the movement and action to be taken on the ball. Also, key-logging is done for which key-press simulation is used to communicate the action (output) that needs to be taken.

```
def key_check():
    keys = []
    for key in keyList:
        if wapi.GetAsyncKeyState(ord(key)):
            keys.append(key)
    if wapi.GetAsyncKeyState(wcon.VK_UP):
        keys.append('up')
    if wapi.GetAsyncKeyState(wcon.VK_DOWN):
        keys.append('down')
    if wapi.GetAsyncKeyState(wcon.VK_RIGHT):
        keys.append('right')
    if wapi.GetAsyncKeyState(wcon.VK_LEFT):
        keys.append('left')
    if wapi.GetAsyncKeyState(wcon.VK_SPACE):
        keys.append('space')

    return keys
```

Method for key-logging

The dataset was split into training and testing dataset in 70%-30% ratio.

Model building and training

Each LSTM model is built with 2 serially connected LSTM layers which are built of 256 units. The second LSTM layer in each model is connected to a fully-connected layer which has 5 outputs in both the models. 128-bit feature vector x 10 images are passed as input to both the parallel models for action selection. The outputs for the first LSTM model are relevant for the movement of the player: up, down, left, right or nothing. And the outputs for the second LSTM model is for the action the player takes on the ball: pass, shoot, cross, through or nothing. The hyperparameters for the model were set as: optimizer – SGD, epochs – 300 (400 before oversampling) and loss – categorical_crossentropy. Training the two models took about 2.5 days on CPU.

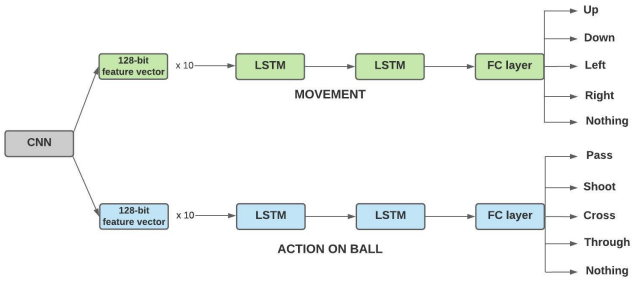


Figure 5: Two parallel LSTM models for action selection

```
def get_model_movement():
    # Network building
    model = tf.keras.Sequential([
        klearn.InputLayer(input_shape=[10, 128], name='net1_layer1'),
        klearn.LSTM(units=256, return_sequences=True, name='net1_layer2'),
        klearn.Dropout(0.6, name='net1_layer3'),
        klearn.LSTM(units=256, return_sequences=False, name='net1_layer4'),
        klearn.Dropout(0.6, name='net1_layer5'),
        klearn.Flatten(),
        klearn.Dense(5, activation='softmax', name='net1_layer6')
    ])
    opt = tf.keras.optimizers.SGD(learning_rate=0.001, clipvalue=5.0)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Model creation for next movement

```
with tf.Graph().as_default():
    # model_movement = tf.keras.models.load_model('./fifa_models2/model_movement')
    # opt = tf.keras.optimizers.SGD(learning_rate=0.001, clipvalue=5.0)
    # model_movement.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    model_movement = get_model_movement()
    model_movement.fit(trainX, trainY_movement, epochs=400, validation_split=0.35)
    model_movement.save('fifa_models2/model_movement')
```

Training the LSTM model for movement

```python
def get_model_action():
    # Network building
    model = tf.keras.Sequential([
        klearn.InputLayer(input_shape=[10, 128], name='net2_layer1'),
        klearn.LSTM(units=256, return_sequences=True, name='net2_layer2'),
        klearn.Dropout(0.6, name='net2_layer3'),
        klearn.LSTM(units=256, return_sequences=False, name='net2_layer4'),
        klearn.Dropout(0.6, name='net2_layer5'),
        klearn.Flatten(),
        klearn.Dense(5, activation='softmax', name='net1_layer6')
    ])
    opt = tf.keras.optimizers.SGD(learning_rate=0.001, clipvalue=5.0)
    model.compile(optimizer=opt, loss='categorical_crossentropy',metrics=['accuracy'])
    return model
```

Model creation for next action to be taken on the ball

```python
with tf.Graph().as_default():
    # model_action = tf.keras.models.load_model('./fifa_models2/model_action')
    # opt = tf.keras.optimizers.SGD(learning_rate=0.001, clipvalue=5.0)
    # model_action.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    model_action = get_model_action()
    model_action.fit(trainX, trainY_action, epochs=400, validation_split=0.35)
    model_action.save('fifa_models2/model_action')
```

Training the LSTM model for action on the ball

```python
def keys_to_output_movement(keys):
    """
    Convert keys to a ...multi-hot... array
    ['up - 0', 'down - 1', 'left - 2', 'right - 3', 'none - 4']
    """
    output = [0, 0, 0, 0, 0]

    if 'left' in keys:
        output[2] = 1
    elif 'up' in keys:
        output[0] = 1
    elif 'down' in keys:
        output[1] = 1
    elif 'right' in keys:
        output[3] = 1
    else:
        output[4] = 1

    return output
```

```python
def keys_to_output_action(keys):
    """
    Convert keys to a ...multi-hot... array
    ['shoot - 0 - space', 'pass - 1 - w', 'through - 2 - q', 'cross - 3 - f', 'none - 4']
    """
    output = [0, 0, 0, 0, 0]

    if 'space' in keys:
        output[0] = 1
    elif 'W' in keys:
        output[1] = 1
    elif 'Q' in keys:
        output[2] = 1
    elif 'F' in keys:
        output[3] = 1
    else:
        output[4] = 1

    return output
```

Methods for keys to output for movement and action on ball

```python
def PressKey(hexKeyCode):
    extra = ctypes.c_ulong(0)
    ii_ = Input_I()
    ii_.ki = KeyBdInput(0, hexKeyCode, 0x0008, 0, ctypes.pointer(extra))
    x = Input(ctypes.c_ulong(1), ii_)
    ctypes.windll.user32.SendInput(1, ctypes.pointer(x), ctypes.sizeof(x))


def ReleaseKey(hexKeyCode):
    extra = ctypes.c_ulong(0)
    ii_ = Input_I()
    ii_.ki = KeyBdInput(0, hexKeyCode, 0x0008 | 0x0002, 0, ctypes.pointer(extra))
    x = Input(ctypes.c_ulong(1), ii_)
    ctypes.windll.user32.SendInput(1, ctypes.pointer(x), ctypes.sizeof(x))
```

Methods for Press key and Release Key

Oversampling data

The players do not have any action on the ball for most of the game, so the data had to be oversampled for minority classes like pass, shoot, cross and through. Data for these classes was replicated in the training dataset. Also, more training data for the higher frequencies of these actions was added. This was done to improve the accuracy and make the players do more actions on the ball instead of just running in the field with the ball.

```
with tf.compat.v1.Session(graph=detection_graph).as_default() as sess:
    image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
    detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
    detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
    detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')
    num_detections = detection_graph.get_tensor_by_name('num_detections:0')
    feature_vector = detection_graph.get_tensor_by_name(
        "FeatureExtractor/MobilenetV1/Conv2d_13_pointwise_2_Conv2d_5_3x3_s2_128/Relu6:0")
    paused = True
    g1 = sess
    g2 = tf.Graph()
    model_movement = load_model('./fifa_models2/model_movement')
    model_action = load_model('./fifa_models2/model_action')
    while True:
        if not paused:
            screen = grab_screen(region=None)
            image_np = cv2.resize(screen, (900, 400))
            image_np_expanded = np.expand_dims(image_np, axis=0)

            with detection_graph.as_default():
                (rep) = sess.run([feature_vector], feed_dict={image_tensor: image_np_expanded})
                input_window[:-1, :] = input_window[1:, :]
                input_window[-1, :] = np.array(rep).reshape(-1, 128)
            Y_movement = model_movement.predict(input_window.reshape(-1, 10, 128))
            movement_index = np.argmax(Y_movement)
            Y_action = model_action.predict(input_window.reshape(-1, 10, 128))
            action_index = np.argmax(Y_action)
            print(action_index)

            if play == 1:
                take_action(movement_index, action_index)

            current_time = time.time()
            if current_time - last_time >= 1:
                print('{} frames per second'.format(frames_count))
                last_time = current_time
                frames_count = 0
            else:
                frames_count = frames_count + 1

        keys = key_check()
        if 'P' in keys:
            if paused:

                paused = False
                print('unpaused!')
                time.sleep(1)
            else:
                print('Pausing!')
                paused = True
                cv2.destroyAllWindows()
                time.sleep(1)
        elif 'O' in keys:
            print('Quitting!')
            cv2.destroyAllWindows()
            break
```

Complete workflow of the model

Additional learning: In addition to taking corners, penalties, free-kicks, throw-ins, the bot also learned to quick-skip the celebration sequences.

Results

A functional agent-based game of FIFA was developed. After running the CNN model, we obtained the following (Figure 6) localization loss between the predicted and ground-truth box parameters.

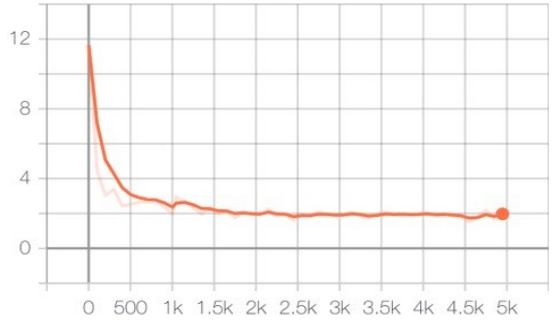
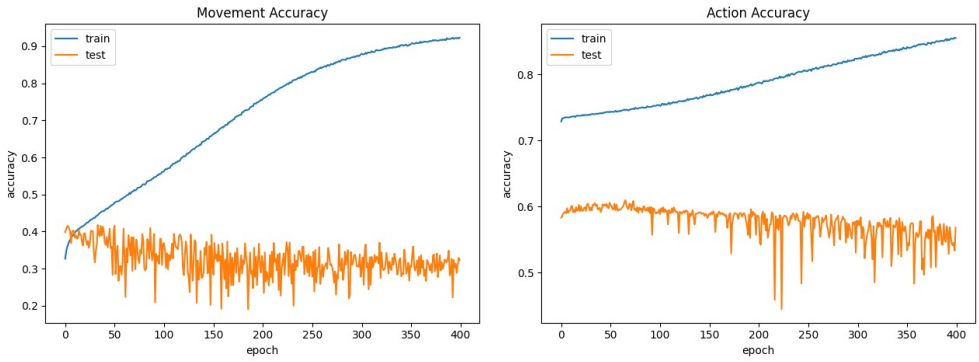
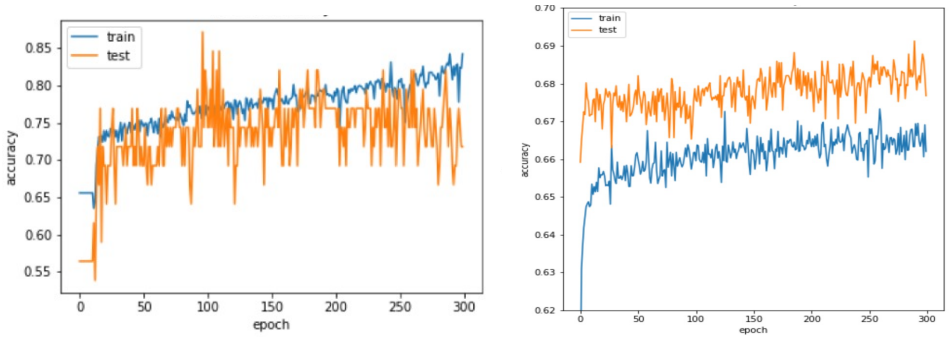


Figure 6: Localization Loss

The bot was made to play at different difficulty levels. At beginner level, the agent played exceptionally well but was unable to perform too well at the legendary level. The results for the LSTM models were stored as graphs before and after oversampling the dataset. But the effectiveness of the model was observed while actually playing the bot. There was vast improvement in the model performance for both movement and action.



Model accuracy before oversampling



Model accuracy after oversampling

Although, we have a running bot playing the entire game, the bot has certain limitations. The model has been trained to play the game while attacking from the left side only. We can build another similar model to train the bot to play from the right side as well. Better efficiency can also be obtained by training deeper models and using higher processing power.

References

1. Robert G. Abbott; Behavioural Cloning for Simulator Validation
2. Okihisa UTSUMI, Koichi MIURA, Ichiro IDE, Shuichi SAKAI, Hidehiko TANAKA; AN OBJECT DETECTION METHOD FOR DESCRIBING SOCCER GAMES FROM VIDEO
3. Juarez Monteiro_1, Nathan Gavenskiy1, Roger Granada_, Felipe Meneguzziz and Rodrigo Barrosz ; Augmented Behavioral Cloning from Observation
4. Wei Liu1, Dragomir Anguelov2, Dumitru Erhan3, Christian Szegedy3, Scott Reed4, Cheng-Yang Fu1, Alexander C. Berg1; SSD: Single Shot MultiBox Detector
5. Danilo Sorano1, Fabio Carrara2, Paolo Cintia,1Fabrizio Falchi2, and Luca Pappalardo2 ; Automatic Pass Annotation from Soccer Video Streams Based on Object Detection and LSTM
6. Ba, J.; Mnih, V.; and Kavukcuoglu, K. 2014. Multiple object recognition with visual attention. arXiv preprint arXiv:1412.7755
7. Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2016. End-to-end training of deep visuomotor policies. Journal of Machine Learning Research 17(39):1– 40.
8. Heess, N.; Wayne, G.; Silver, D.; Lillicrap, T.; Erez, T.; and Tassa, Y. 2015. Learning continuous control policies by stochastic value gradients. In Advances in Neural Information Processing Systems, 2944–2952
9. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602
10. Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484–489.
11. Hausknecht, M., and Stone, P. 2015. Deep recurrent q-learning for partially observable mdps. arXiv preprint arXiv:1507.06527
12. Foerster, J. N.; Assael, Y. M.; de Freitas, N.; and Whiteson, S. 2016. Learning to communicate to solve riddles with deep distributed recurrent q-networks. arXiv preprint arXiv:1602.02672.
13. J. Tao, J. Xu, L. Gong, Y. Li, C. Fan, and Z. Zhao, ‘‘NGUARD: A game bot detection framework for NetEase MMORPGs,’’ in Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, Jul. 2018, pp. 811–820.