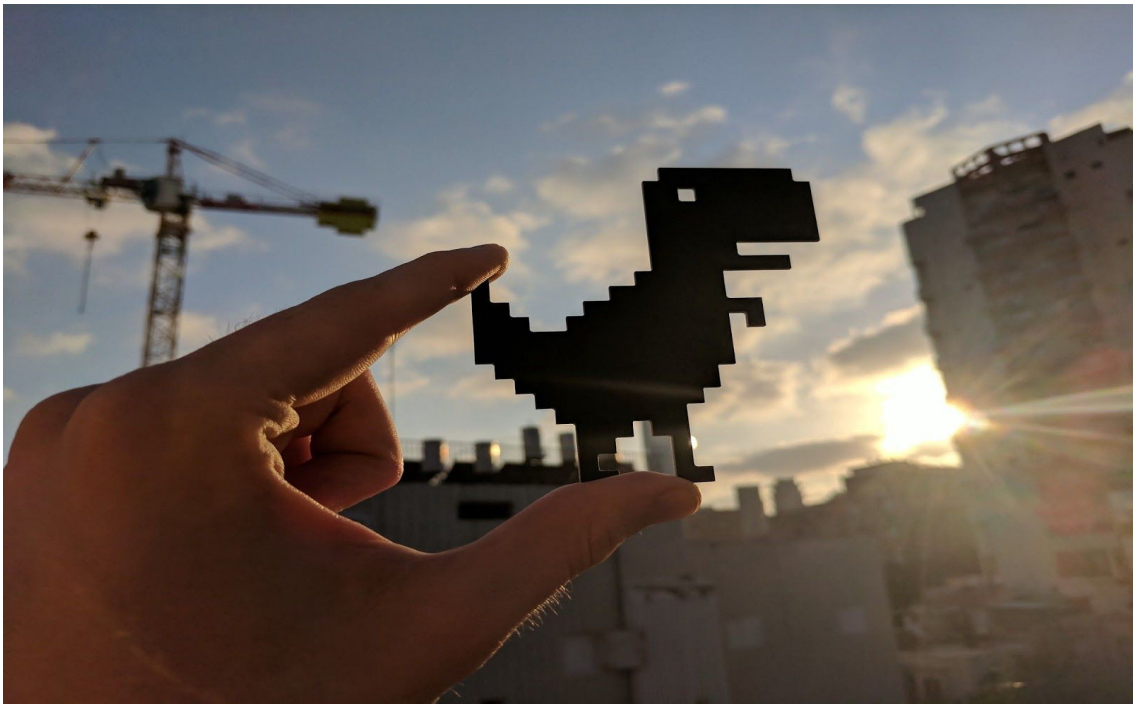
CPS 584 Final Project

Team Members:

1. Nubina Nitturu - 101659084
2. Ashish Parimi - 101659091

AI Bot for Chrome Dinosaur Game



1.Introduction:

The Chrome Dinosaur Game appeared in 2014 in a Chrome browser when a user tries to visit a website while disconnected from the Internet.The Chrome Dinosaur Game is a simple infinite runner, in which a player may have to jump over cacti, and duck underneath birds. Controls are basic: pushing the up Arrow translates into a jump, and the down arrow into ducking. The goal is to survive as long as possible.The set of features, given the nature of the game, are obtained by capturing the screen during gameplay to identify the instances of respective actions and using it to train a neural network which would predict the action of jump,duck and idle when provided with a play interface.

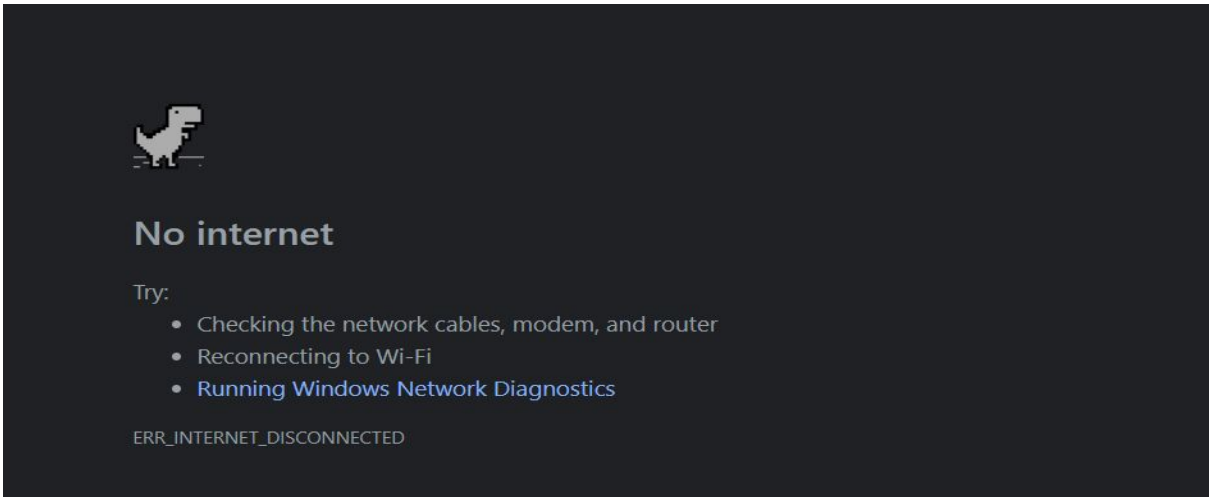


Figure 1.1 Chrome Page When Internet is Disconnected

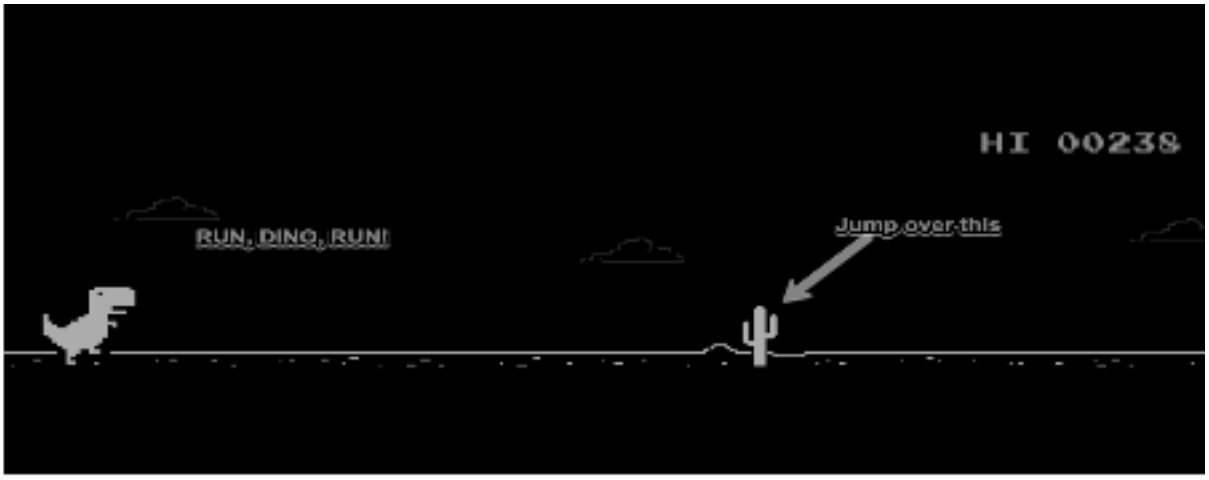


Figure 1.2 Dino is Idle and about to Jump

Figure 1.3 Game Over when Dino couldn't Jump

2.Project Description:

In this project, We propose a CNN model, for learning to control the game agent(Dino), in the classic game embedded in Chrome offline mode, directly from high-dimensional image input.We first create an interface to capture the images for particular actions like jump and duck.For pixel based feature extraction we crop the image to size to contain only the play environment .To these images we applied Edge detection algorithm for finding the boundaries of Dino and obstacles.This data is then used to train Convolutional Neural Network(CNN) to predict which action to take under given circumstances. These predictions are implemented using a network browser interface which results in the Dino being controlled by the Neural Network.

3.Purpose of the Project:

Learning human-level control policies directly from high-dimensional data is a challenge for controlling system design.In this project, We implement Convolutional Neural Network method to learn to control Chrome Offline Dinosaur Game directly from game screen input.

4.Source of Data:

- ➢ We collected the data by capturing the gameplay on our PC.
- ➢ Training Data: 2926 Images
- ➢ Test Data: 0.1% Training Data
- ➢ Classifications : 3(jump,duck,ideal)

5.Survey of Current Methods:

There are multiple approaches to this project:

1) Using Q-learning
2) CNN
3) Deep-Reinforcement learning
4) Directly implementing the game in Python

Currently Deep-Reinforcement Learning is widely used for creating game bots. In 2013 Google proposed the use of deep reinforcement learning on training agents to play the 2600 Atari games. Taking just the pixels and reward received from the game as inputs, they were able to reach human-expert performance in multiple Atari games. The agent is able to learn to play the game without knowing the underlying game logic. Further improvements involve prioritizing experience replay, more efficient training, and better stability when training.

6.Proposed Method:

Our proposed method is CNN as unlike other methods there is no need for creation of a game environment to collect data or train. Methods like Q-learning require modification of game parameters to get high scores.

7.Details of Implementation:

7.1 Built a two way interface between browser and Model

Selenium, a popular browser automation tool, is used to record and send actions to the browser during gameplay.

7.2 Captured screenshots and pre-process images

Selenium,mss and OpenCV gave best performance for screen capture and pre-processing of the images respectively, achieving a decent frame-rate of 40-50 fps.We captured screenshots of the play environment using mss per action based, with the help of *Keyboard* to detect the action on screen.These raw images were cropped to a size of 160X540 such that it only contains the obstacles. We used OpenCV's Canny Edge Detection [1], an open source computer vision tool for preprocessing the image.

Resulting Images:

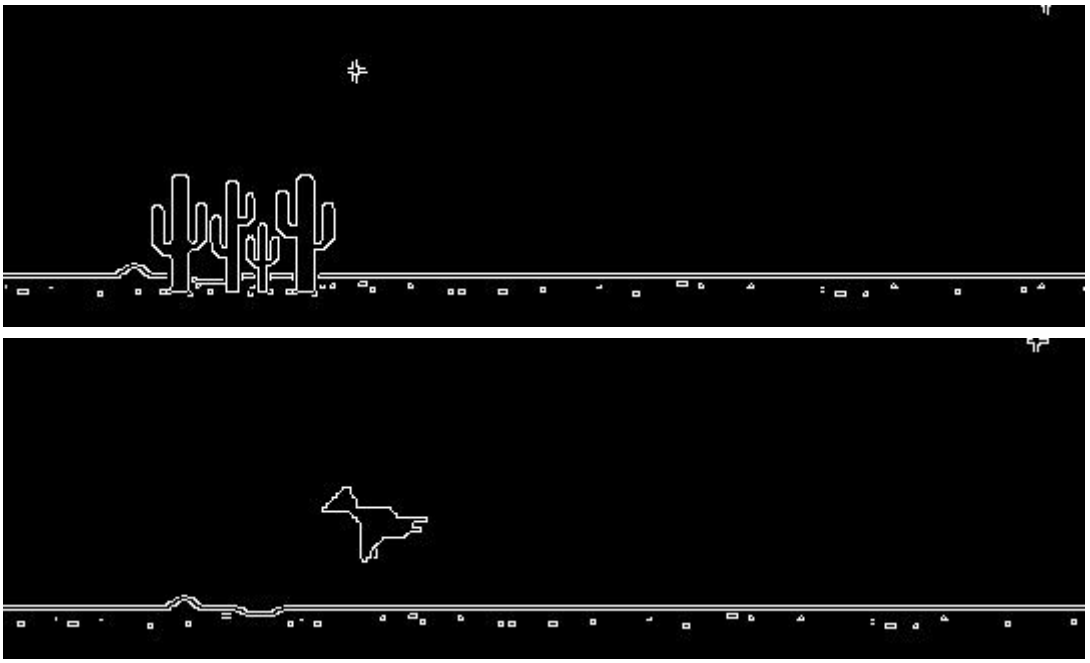


Figure 7.2.1 Images Extracted From Canvas after preprocessing

These images are further resized to 160X320 for faster training and accurate feature extraction.This Data is then Augmented using Tensorflow's ImageDataGenerator.
To Help with the training we have reduced the color dimension to Gray Scale while reading the images using ImageDataGenerator. The resulting image is of size 160X320 with a single channel.That makes our single input of dimensions 160X320X1.

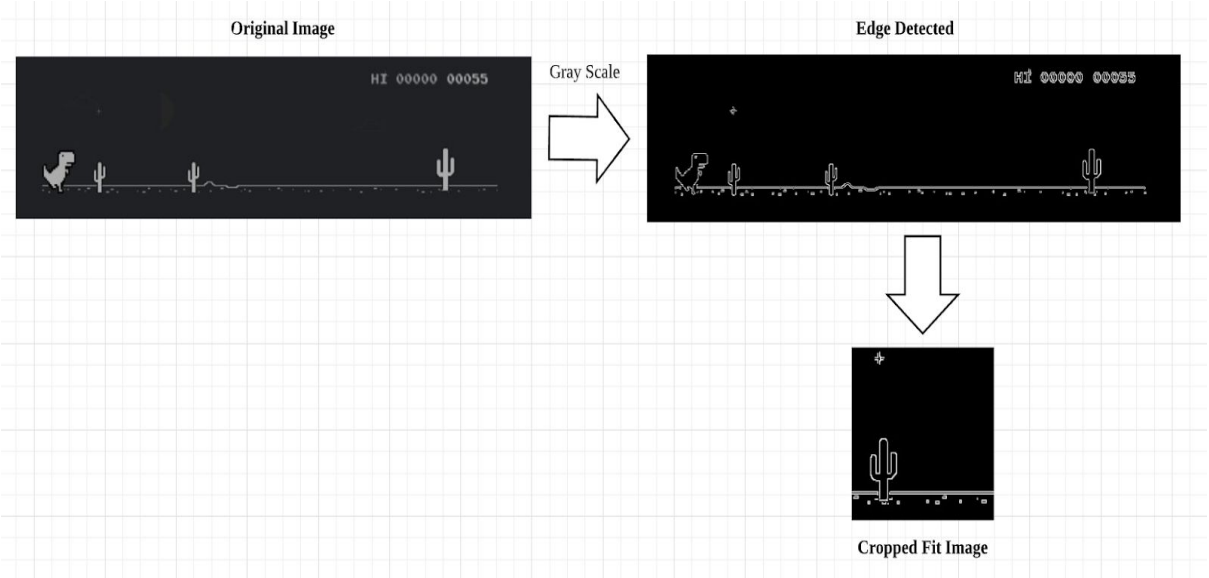


Figure 7.2.2 Image Processing

7.3 Model Architecture:

We used a series of three Convolution layers before flattening them to dense layers and output layers. Max Pooling layers significantly improve the processing of dense feature set.The CNN used contains convolution layers(with ReLU layers following them) and max pooling layer after each Convolution layer and two fully connected layers. The input image first goes through a convolution layer with 16 filters of size 3 × 3 , followed by a ReLU layer. Then a 1 × 2 max pooling is applied to the output of convolution. The tensor then goes through two convolution layers with 32 filters of size 3×3 and 128 filters of size 3×3.Then the tensor is flattened. Finally the flattened tensor goes through two fully connected layers with first dense layer with 256 units and Relu activation function and final output layer with 3 units and Softmax Activation Function.We used a dropout of 10% between the dense layers.

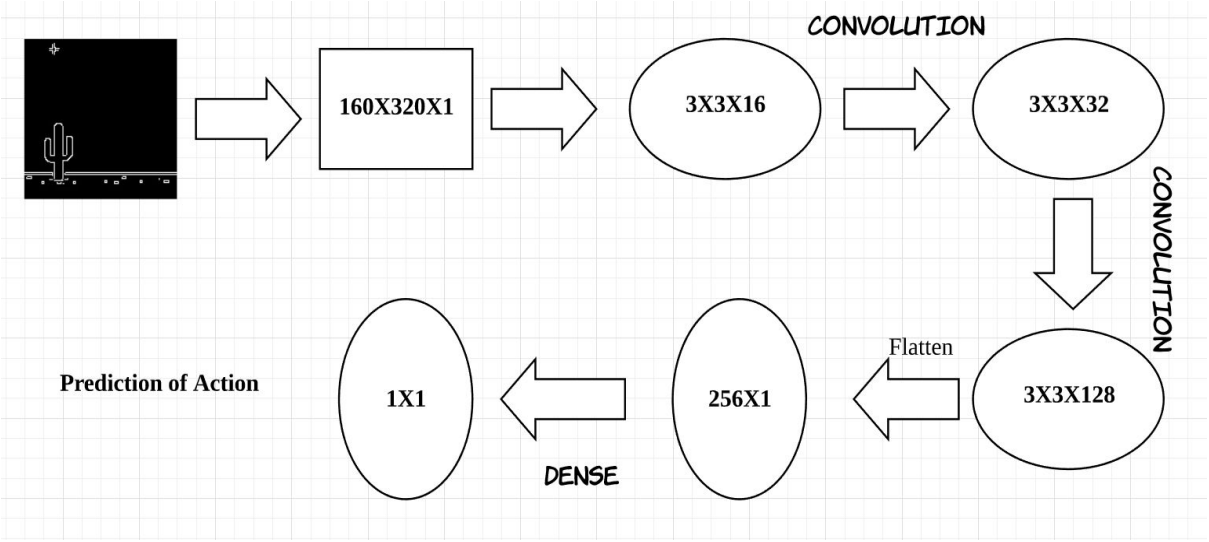


Figure 7.3. Model Architecture

We used early stopping callback and ModelCheckpoint to save the weights of the best epoch.

7.4 Implementation:

Selenium is used to create an interface to open a browser in offline mode to initiate the game.The screen of the play environment is then captured such that it contains only the obstacles using mss, pre-processed(Edge Detected and Re-Sized) and passed to the model for prediction. These predictions are then passed to the browser to perform actions using selenium.The capture window is adjusted after 38sec of start of the game to compensate for the increase in the speed.

8.Results:

We should be able to get good results by using this architecture. The GPU has significantly improved the results which can be validated with the improvement in the average score. The average score per 10 games stays well above 536.The highest score recorded was 1713 and the lowest was around 57.The speed of the Dino is proportional to the score, making it harder to detect and decide an action at higher speed.

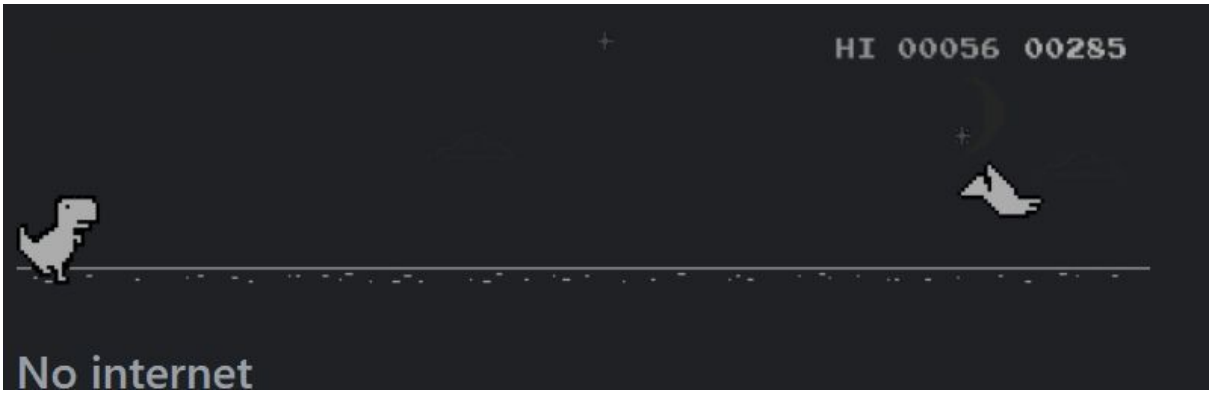


Figure 8 One of the Scores Captured from our Model

9.Conclusion and Future Work:

In our project, CNN Model couldn't handle velocity well enough. We notice velocity makes a great impact over the jumping position selection. Despite the compensation window changes in speed were not addressed as the game progressed . Several approaches were used to achieve this. Carefully designed CNN and carefully filtering the input data can successfully abstract the state and AI built upon them can improve its performance. However, CNN has its limits and can not outperform the human experts. Finally, specially designed training methods can help us overcome the training difficulties caused by the properties of our game, which further improves our AI's performance and helps achieve results.

10.Copy of Programs

GitHub Link: https://github.com/Ashish2Parimi/Chrome-Bot

References:

[1] Canny Edge Detection by John F. Canny
[2]https://www.youtube.com/watch?v=v9YwegSFyKI

[3]https://pythonspot.com/category/selenium/