Chess Playing Bot : Report 1

Aditya C, Gayatri T, Sahaj S, Shobuj P

November 2020

1 Summary

Currently, we are at the end of the Learning Phase. We plan on implementing the Speech Recognizer (Using CNN's) by this week and shall then start working on the Circuitry and the Robot Design. A few prototype designs will be presented to Prof. Gangadharan (CSD) by the end of November or first week of December[hopefully] and get the required parts 3D Printed (other than those commercially available).

From the Speech Recognition POV, our GUI works well in combination with the Microphone and the Arduino helps prompt the user to Speak. We shall add a few more features like Hints and Reverse previous moves once the Basic Model is completed.

2 The Graphical User Interface

We used the python-chess library along with the Stockfish chess engine. The GUI for the same was obtained from a Terminal Chess project by Nick Zuber. This code was modified to incorporate our needs i.e. responding to voice command and storing values in a file.
To modify the code such that it responded to voice commands, the libraries Pyaudio and Speech-Recognition were used. For speech recognition, we used Google web speech API. The accuracy of Google API is pretty good although it sometimes does give some errors. We intend to build a speech recognizer. However, the Google API will be used for now.
The results of the speech recognizer (the chess moves) are stored in a file. The coordinates of the moves are then sent to Arduino which will be used to control the motors of the mechanical system.

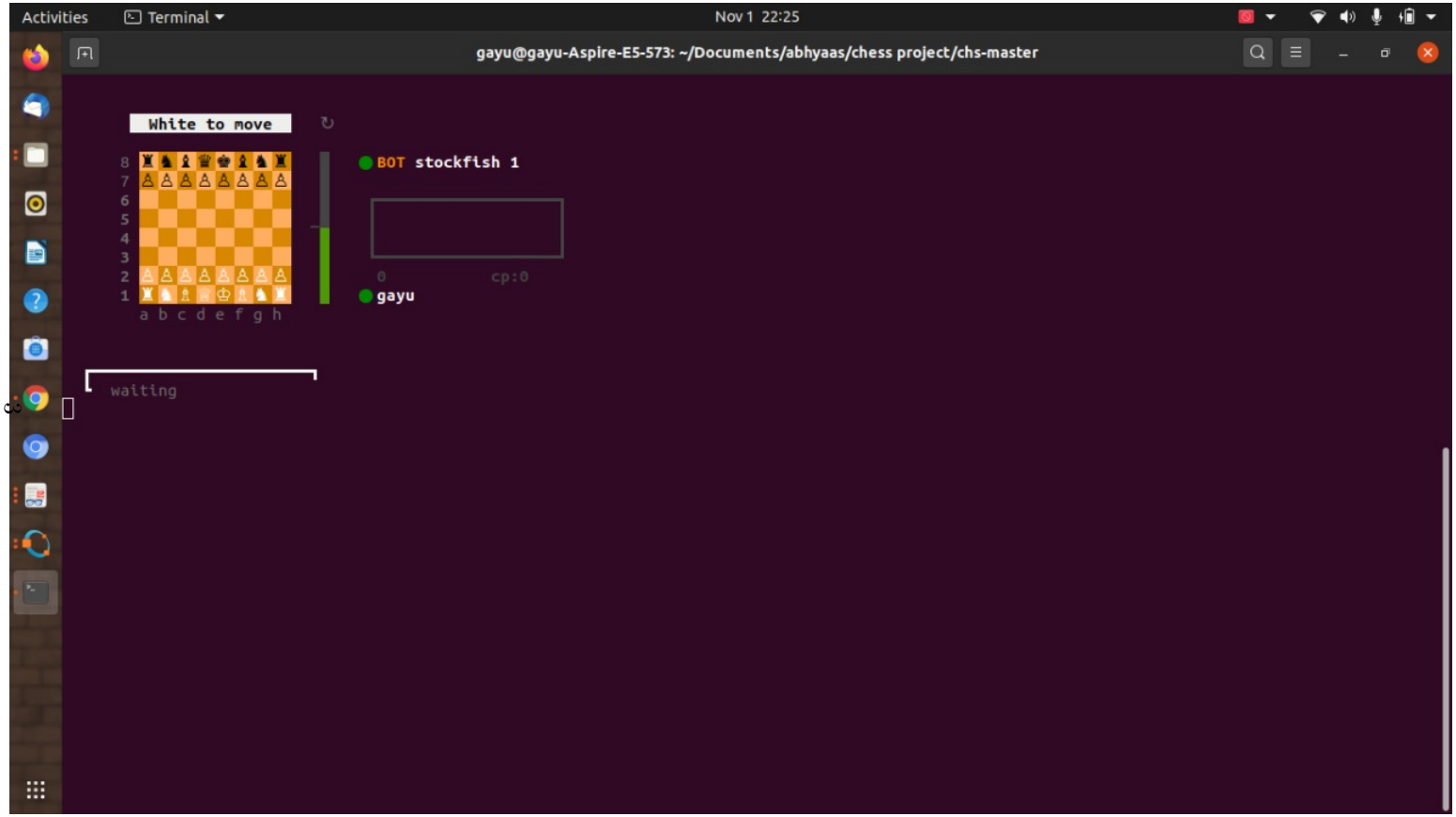

Figure 1: Chess Bot : GUI

3 Learning Phase

3.1 Signal Processing

Basics of Signal processing was covered for speech processing, which included Fourier Transforms and different methods of calculating them which were as follows:

Fourier Series Any periodic signal can be represented as a (possibly infinite) sum of sines and cosines of various frequencies. The coefficients of the sum were calculated using orthogonality of sinusoidal functions. They can also be represented as a series of complex exponential functions functions of which are themselves sums of sines and cosines. We can also construct a signal given a Fourier Series.

Continuous Time Fourier Transform For a non-periodic function, we can say that it's frequency approaches zero, and take the limit of the sum of the Fourier series, which gives us the continuous time Fourier transform. The Fourier coefficients will be represented as a continuous function It was a transform integral with the integrating factor $exp^{-i\omega t}$, and we can similarly conduct an inverse transform to give us the original function.

Discrete Time Fourier Transform For a discrete signal, we can construct the discrete Fourier transform which uses the properties of the Nth roots of unity and the periodicity of 'mod N' to derive its properties like circular convolutions.

Fast- Fourier Transform FFT is an algorithm for computing DFTs in a fast and efficient way by pre-computing certain sub-expressions. This can be understood by means of matrix factorization. This reduces the computational complexity from O($4N^2$) to O($2Nlog_2N$). We can use this algorithm to compute the Fourier transform of images.

All these transforms have some basic common properties like Parseval's theorem and convolution properties. We also covered basic filter responses like the Butterworth and Bessel characteristic as well as basics of systems.

3.2 Convolutional Neural Networks

Introduction The convolutional neural network is a specialized type of neural network model designed for working with two-dimensional image data, although they can be used with one-dimensional and three-dimensional data. Central to the convolutional neural network is the convolutional layer that gives the network its name. This layer performs an operation called a "convolution".Technically, the convolution as described in the use of convolutional neural networks is actually a "cross-correlation". Nevertheless, in deep learning, it is referred to as a "convolution" operation.In the context of a convolutional

neural network, a convolution is a linear operation that involves the multiplication of a set of weights with the input, much like a traditional neural network. Given that the technique was designed for two-dimensional input, the multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel. The filter is smaller than the input data and the type of multiplication applied between a filter-sized patch of the input and the filter is a dot product. A dot product is the element-wise multiplication between the filter-sized patch of the input and filter, which is then summed, always resulting in a single value. Because it results in a single value, the operation is often referred to as the scalar product.Using a filter smaller than the input is intentional as it allows the same filter (set of weights) to be multiplied by the input array multiple times at different points on the input. Specifically, the filter is applied systematically to each overlapping part or filter-sized patch of the input data, left to right, top to bottom.This systematic application of the same filter across an image is a powerful idea. If the filter is designed to detect a specific type of feature in the input, then the application of that filter systematically across the entire input image allows the filter an opportunity to discover that feature anywhere in the image. This capability is commonly referred to as translation invariance, e.g. the general interest in whether the feature is present rather than where it was present. The output from multiplying the filter with the input array one time is a single value. As the filter is applied multiple times to the input array, the result is a two-dimensional array of output values that represent a filtering of the input. As such, the two-dimensional output array from this operation is called a "feature map". Once a feature map is created, we can pass each value in the feature map through a nonlinearity, such as a ReLU, much like we do for the outputs of a fully connected layer.

Stride The filter is moved across the image left to right, top to bottom, with a one-pixel column change on the horizontal movements, then a one-pixel row change on the vertical movements. The amount of movement between applications of the filter to the input image is referred to as the stride, and it is almost always symmetrical in height and width dimensions. The default stride or strides in two dimensions is (1,1) for the height and the width movement, performed when needed. And this default works well in most cases. The stride can be changed, which has an effect both on how the filter is applied to the image and, in turn, the size of the resulting feature map. For example, the stride can be changed to (2,2). This has the effect of moving the filter two pixels right for each horizontal movement of the filter and two pixels down for each vertical movement of the filter when creating the feature map

Padding By default, a filter starts at the left of the image with the left-hand side of the filter sitting on the far left pixels of the image. The filter is then stepped across the image one column at a time until the right-hand side of the filter is sitting on the far right pixels of the image.An alternative approach to applying a filter to an image is to ensure that each pixel in the image is given

an opportunity to be at the center of the filter. By default, this is not the case, as the pixels on the edge of the input are only ever exposed to the edge of the filter. By starting the filter outside the frame of the image, it gives the pixels on the border of the image more of an opportunity for interacting with the filter, more of an opportunity for features to be detected by the filter, and in turn, an output feature map that has the same shape as the input image. For example, in the case of applying a 3×3 filter to the 8×8 input image, we can add a border of one pixel around the outside of the image. This has the effect of artificially creating a 10×10 input image. When the 3×3 filter is applied, it results in an 8×8 feature map. The added pixel values could have the value zero value that has no effect with the dot product operation when the filter is applied. The addition of pixels to the edge of the image is called padding.

Pooling Layer A limitation of the feature map output of convolutional layers is that they record the precise position of features in the input. This means that small movements in the position of the feature in the input image will result in a different feature map. This can happen with re-cropping, rotation, shifting, and other minor changes to the input image.A common approach to addressing this problem from signal processing is called down sampling. This is where a lower resolution version of an input signal is created that still contains the large or important structural elements, without the fine detail that may not be as useful to the task.Down sampling can be achieved with convolutional layers by changing the stride of the convolution across the image. A more robust and common approach is to use a pooling layer.A pooling layer is a new layer added after the convolutional layer. Specifically, after a nonlinearity (e.g. ReLU) has been applied to the feature maps output by a convolutional layer. The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps.The pooling operation is specified, rather than learned. Two common functions used in the pooling operation are:

- **Average Pooling**: Calculate the average value for each patch on the feature map.
- **Maximum Pooling (or Max Pooling)**: Calculate the maximum value for each patch of the feature map.

The result of using a pooling layer and creating down sampled or pooled feature maps is a summarized version of the features detected in the input. They are useful as small changes in the location of the feature in the input detected by the convolutional layer will result in a pooled feature map with the feature in the same location. This capability added by pooling is called the model's invariance to local translation.
