
# Indian Institute of Information Technology Vadodara

Project Report

# Chess Pieces Classification

## CS 308 Machine Learning

**Course Instructor:**
**Dr. Jignesh Bhatt**

**Teaching Assistant:**
**Mrs. Swati Rai**

**Group Member:**
**Tapan Savani - 201952231**
**Manthan Ghasadiya - 201951065**
**Rushi Javiya - 201951074**
**Mohit Panchasara - 201952224**

CONTENTS

## I. Introduction

Deep Learning is just an incredible concept of what machines can do, and it is so powerful that engineers and business pioneers all focus on it. This extraordinary sort of calculation has far outperformed any past benchmarks for grouping pictures, text, and voice.

Deep learning is based on neural networks, similar to human brain cells. Earlier deep learning was not popular due to less processing power and a high need for data. But now, we are capable of power efficiency and more extensive data.

Neural networks work in the same way as human brain cells, which means we can recognize objects in the same way our human mind recognizes them. Which helps identify objects using Machines.

Our project is based on object detection using deep learning. It is based on multi-class classification for chess pieces using "CNN."

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the picture, and differentiate one from the other class.

CNNs, like neural networks, are made up of neurons with learning weights and biases. Each neuron gets several inputs, converts them to a weighted total, runs it through an activation fu²nction, and returns with an output.

## II. Pre-Processing

The dataset has been collected from Kaggle [1]. Dataset gathered is of variable-sized and colored images. The dataset has a total of 6 different classes possible. Knights, kings, queens, rooks, bishops, and pawns.

So as pre-processing steps, we have first resized it to $256 \times 256$ and then converted it to grayscale images.


Fig. 1: Pre-processing

## III. Model Development

We have created two different models for chess piece classification. One model is created using in-built libraries like Keras and OpenCV. In the second model, we write our code, i.e., only use of NumPy and matplotlib for applying convolution, different pooling, forward pass, backward pass, and activation layer. From the other pooling methods, we have used only max pooling to extract sharpen and smoothen features from images. You can see the structure of the model in figure 2 [2].

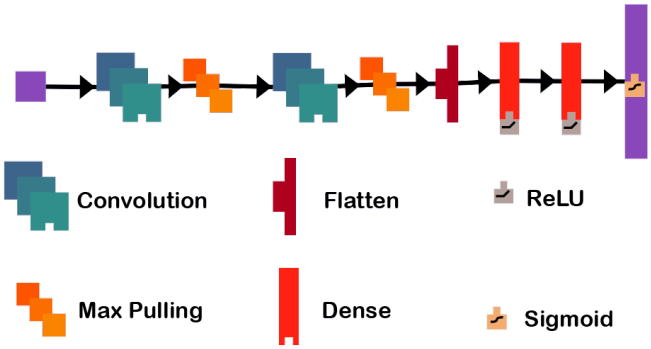


Fig. 2: Structure of classifier

Here, we first apply ten different filters to each image. After this convolution, we applied max pooling and one more layer of convolution-max pooling. So, from 1 image, we are getting 1*10*10 = 100 images. Now, we are applying to flatten layer to these images. We flat all the images.

In the neural network, we have two hidden layers. Each hidden layer has 512 neurons. We pass our flattened images to these hidden layers and get the output. This is a forward pass. After this, we apply a backward pass to update the weights of the hidden layer and learn from the data.

We also used activation in the forward pass and ReLU and Sigmoid. Here we have not used softmax for large input values; it makes 0. So, it was giving all 0 as output.

Sigmoid Formula:

$$\frac{1}{1+e^{x}} \tag{1}$$

ReLU Formula:

$$max(0, x) \tag{2}$$

## IV. Training

### A. Split into training and Testing

First, we divide all the dataset images into the training and testing parts. We have taken 80 pictures per class. And we have two convolution layers. In convolution, we have ten filters applied to each image. So, after the first convolution layer from a single image, we get ten images. So, the total images after the first convolution layer will be 800. And same ten filters were applied in the second convolution layer on these 800 images. So, the total number of images will be 8000 per class. And a total of 48000 images having six different classes.

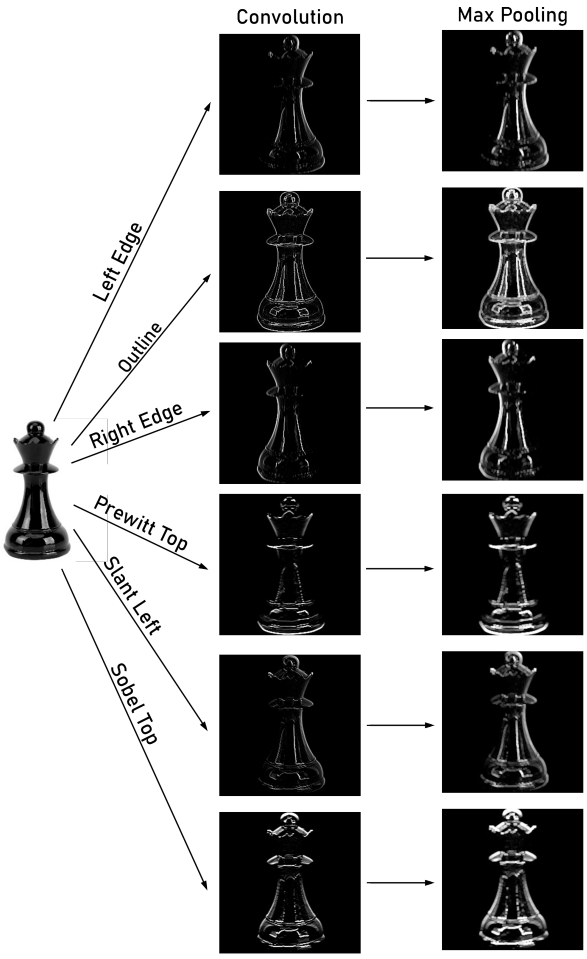


Fig. 3: First convolution and Pooling

After applying Max-Pool of kernel size 3 x 3 for two times, we get the dimension of the image reduced from 256 x 256 to 252 x 252.

Then 80% of these images, i.e., 38400 images, will go into the training part, and the remaining 9600 images for validation. This is necessary for the model to prevent it from over-fitting.

### *B. Flatten Images*

All the input images will be flattened into 1 Dimensional Array. Hence, initially, the image size was 252 x 252 pixels and then converted to 1 x (252*252) pixels. Each pixel value will be between 0 to 256 and stored in each array cell. So array for one image will be of size 1 x (252*252).

### *C. Input Formation*

By applying the convolutional layer, we got 38400 images for training. To form input matrix 'X' by concentrating all the images into the matrix. Therefore the dimensional of the final matrix is 38400 x (252*252).

This input array is passed directly into the neural network layer for predicting the output. Simultaneously original 'Y' is created, having index values corresponding to the class value of that particular image. This Y matrix has dimension 38400 x 1 and is used in backpropagation.

### *D. Training Model*

The X matrix will be passed into gradient descent function, which takes X, Y, no. of iterations/epochs, and learning rate alpha as input. Further, it is passed into the forward pass function, which multiplies the X with the corresponding weights of the layer.

Simultaneously applying the activation function on each value coming out of a neuron makes the function nonlinear, and this follows until the last layer is processed.

Then the output will be the input for the backpropagation function. The error is calculated by subtracting from the original labels and taking the dot product with the previous layer's output. Instead of the activation function, we use a derivative of the activation function.

Sigmoid Derivative:

$$x \times (1 - x) \tag{3}$$

ReLU Derivative:

$$1 \times (x > 0) \tag{4}$$

We use derivative because we want the differences in predicted and original output slopes. Then we adjust the weights according to the new predicted weights multiplied by learning rate alpha.

Here we are not using the SoftMax function because our model produced significantly higher values during the forward pass. Due to a lack of computational power, the SoftMax function returned only zero or NaN values. Hence we have taken sigmoid instead of SoftMax function.

## V. Testing

We have 9600 images to validate using our trained model in the testing phase. The updated weights will predict the output, and the accuracy is calculated. For accuracy, first, we are getting the outcome as a large matrix with six columns.

This indicates the six classes denoting the six types of chess pieces, namely 'King,' 'Queen,' 'Rook,' 'Knight,' 'Bishop,' 'Pawn.' Thus, we use the ArgMax function to calculate the most significant most value from the row, which denotes the highest tendency towards that class and return that class. We can estimate the accuracy after every epoch and calculate the Validating accuracy.

## VI. Results

### *A. Accuracy*

Successfully established the formation of convolutional neural network layers for detecting multi-class datasets of chess pieces. Using the above layers, we have built our model and tested it with images successfully. We have successfully classified images with a training accuracy of 82% and validation accuracy of 67%.

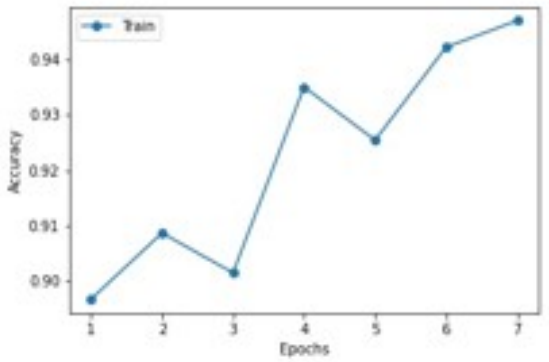


Fig. 4: Training Accuracy of Model using Libraries

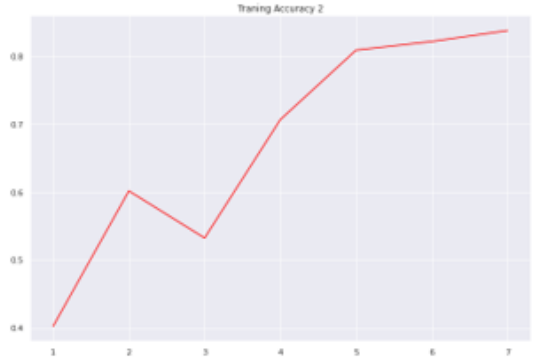


Fig. 5: Training Accuracy of Our Model

### B. *Saliency map*

Saliency maps are the images obtained after every neuron layer, i.e., the image after multiplying the updated weights with the input image. The following figures show how the saliency map extracts some patterns from the original image. The figure below shows that the image is the result of a pattern that is detected after multiplying 1st layer weights with the input test image.

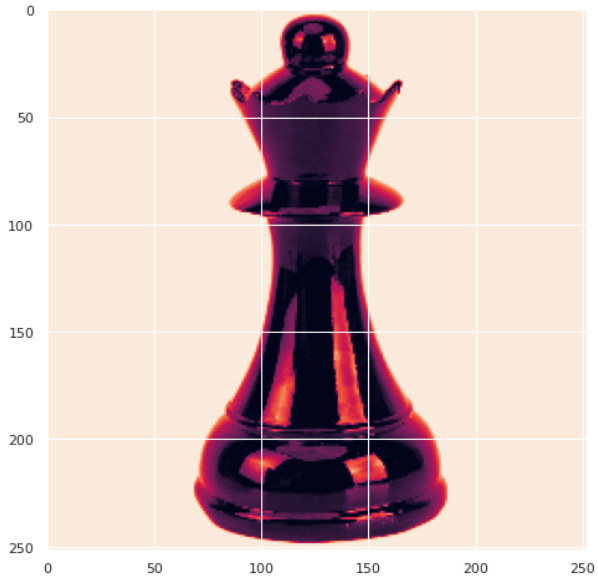

Fig. 6: Grayscaled Image (collab image)

This is the orignal graysaled image, which is tested using the trained model.

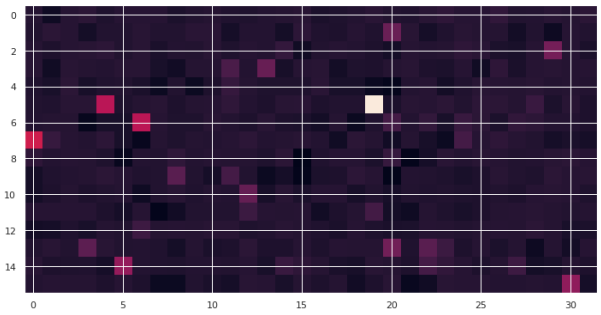

Fig. 7: Saliency map after 1st layer

We get here the resulting image after the 1st Hidden layer, i.e., the original grayscaled image is multiplied by the 1st layer trained Weight W1. We can see the edges are taken into a pattern (dark-colored). This will be the output to the next hidden layer, which detects some pattern from this image again. These saliency maps are very useful to detect some pattern or see the progress report of how the model is behaving to the input images.

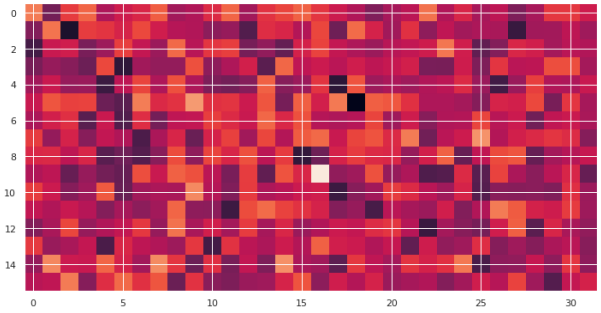

Fig. 8: Saliency map after 2nd layer

Here is the resulted image, which we get by multiplying 2nd layer weights W2 with 1st layer input.

## VII. Conclusion

### A. *Comparing two models*

In the below figure, we have compared both models. Here, the number of epochs, training images, and the number of neurons are higher in our model build without using the Keras library. But the accuracy is higher in the other model built with the Keras library. And there is also a big difference in the time taken by both the model for training.

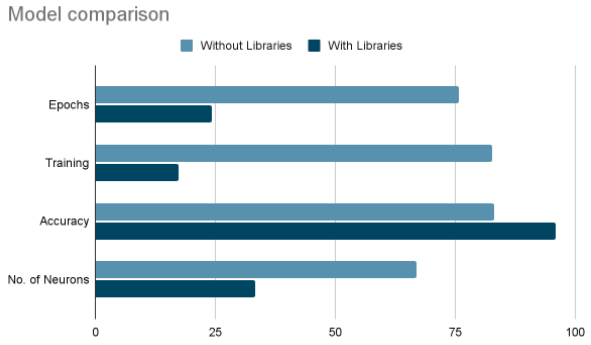


Fig. 9: Comparing Both Models

Model build with Keras takes less time in the training. As in the other model, we have written our functions for convolution, max pooling, forward pass, backward pass, and activation function. So, our function might not be the most optimal.

We can also conclude that using our pre-trained model, and we can easily adjust the number of neurons and the number of convolutional layers in our model. Also, we can get any of the updated weights during the training and can build saliency maps effortlessly.

This all cannot be done using some library functions. Comparatively, we get higher accuracy in the model made using libraries because it is optimized. To achieve higher accuracy in our model, we need to increase the number of layers and the number of epochs.

## VIII. Useful Links

- **Preprocessing**
- **Convolution + MaxPool**
- **2nd Convolution + MaxPool**
- **Code**
- **Presentation**

## References

[1] Kaggle. *Chessman image dataset*. URL: https://www.kaggle.com/datasets/niteshfre/chessman-image-dataset.

[2] MIT. *Math MIT*. URL: https://math.mit.edu/ennui/.