Using Machine Learning to Interpret Dice Rolls

Hunter Wimsatt
Western Kentucky University, hunter.wimsatt565@topper.wku.edu

Aarohi Panzade
Western Kentucky University, aarohi.panzade528@topper.wku.edu

Kaaustaaub Shankar
Western Kentucky University, kaaustaaub.shankar610@topper.wku.edu

Warren Campbell
Civil Engineering, warren.campbell@wku.edu

Using Machine Learning to Interpret Dice Rolls

Hunter Wimsatt
Western Kentucky University and Carol Martin Gatton Academy of Mathematics and Science
hunter.wimsatt565@topper.wku.edu

Aarohi Panzade
Western Kentucky University and Carol Martin Gatton Academy of Mathematics and Science
aarohi.panzade528@topper.wku.edu

Kaaustaaub Shankar
Western Kentucky University and Carol Martin Gatton Academy of Mathematics and Science,
kaaustaaub.shankar610@topper.wku.edu

C. Warren Campbell
Kenneth E. and Irene S. Hall Professor of Civil Engineering
School of Engineering and Applied Sciences
Western Kentucky University
warren.campbell@wku.edu

Using Machine Learning to Interpret Dice Rolls

Abstract:

Gamers use polyhedral dice that come with 4 sides (D4), 6 sides (D6), 8 sides (D8), 10 sides (D10), 12 sides (D12) and 20 sides (D20). All dice are unfair, some more than others. The goal of this project was to develop a machine learning and computer vision solution for the interpretation of dice rolls. When combined with an automated dice roller it would facilitate the study of dice unfairness. In the machine learning literature, Convolutional Neural Networks (CNNs) are the preferred method of computer classification of images. The CNN convolves images with different weights and biases through multiple layers to produce an end array with odds of each number on the die. Using CNNs we were able to obtain interpretation accuracies ranging from 95.10% to 99.76%. Thousands of images were used for training, and thousands of separate images used for validation.

Introduction:

Dice are used very commonly throughout the world. There are 6 main types of dice; d4, which are 4 sided dice, d6, which are 6 sided dice, d8, d10, d12, and d20. The use of dice range from board games, to casinos, to research on dice, and even more. Board games, such as Dungeons and Dragons, use multiple different dice. Dungeons and Dragons uses one of each of the main types of dice with an extra d10 die for a total of 7 dice. Craps is a casino game where players wager on the outcome of a dice roll. Dice research has also been conducted to test dice unfairness by manually rolling a die and gathering data.

Our research project aims to make steps towards dice rolling automation in some of its uses by creating a program able to receive a dice image and classify the side that is facing up correctly at least 95% of the time by using a machine learning algorithm. This report goes through the methods, results, and discussions of our findings from our research and implementation.

A dataset from Kaggle with approximately 16,000 images of d4, d6, d8, d10, d12, and d20 dice was used to train our machine and validate the neural network. The images were taken from different angles and used different backgrounds (Lurig, 2018).

Convolution Neural Networks were also used in this project to analyze and sort the dice. It is a deep learning algorithm which was invented by Yann LeCun around the 1980s. It first takes in a certain type of input, in our case an image. It then multiplies it by different weights to convolve the image. If there are more features that must be detected, then more layers must be added to the network. More layers increase the amount of processing used for a higher accuracy.

Methods:

We used 4 steps to solve the above problem. First, we obtained pictures of multiple dice ranging from d4s to d20s made of different materials. We obtained our dataset from Kaggle. The link for this dataset is provided in the references. Every eighth image of the set was made a part of the validation set. The second step was to crop and grayscale the pictures. To make our AI more accurate, we eliminated irrelevant information from the image such as the background and the color of the dice. The third step was to manually sort the images into folders categorized by the number of sides on the dice. Finally, these folders were categorized by the number displayed on the dice. We then trained the computer on the training set and then interpreted the validation sets.

1. *Editing Dice Images (Cropping and Grayscaling)*

   The dice images for each different dice were shown from several different angles, with different colored dice and backgrounds. This gave a variation to our images so the machine learning algorithm could grow correctly and not base its training by one specific scenario. Some examples of the different angles and backgrounds of these dice are shown below in figures 1, 2, and 3.

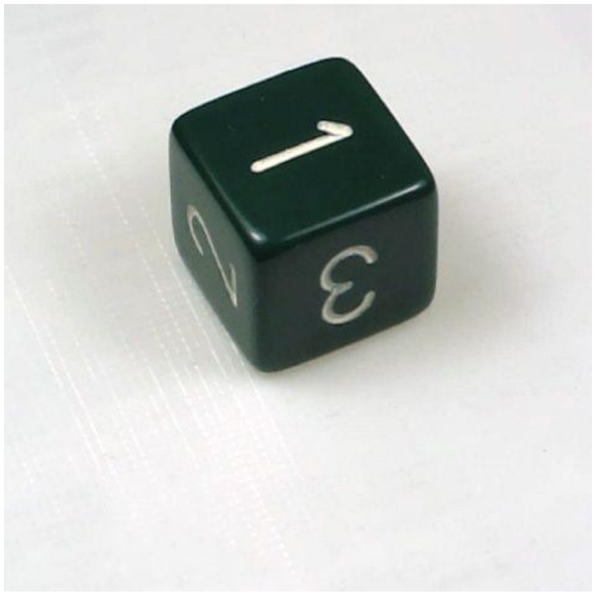


Figure 1. Original D6 Off Angle Dice Image

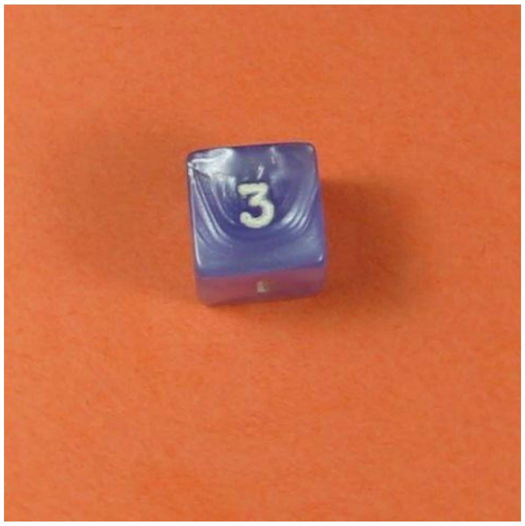


Figure 2. Original D6 Colored Background Image

This figure shows a blue dice image with white numbers and an orange colored background. The face up value is 3.

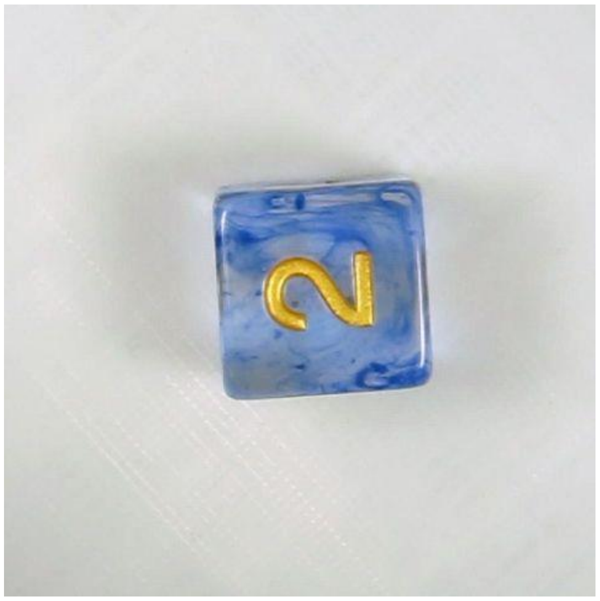


Figure 3. Original D6 Dice Image from Top

These figures have lots of color and angle variations between them which can be distracting to the neural network. From these figures, new dice images were created by cropping the backgrounds out and changing the dimensions of the image to 100 by 100. The background cropping was done with some code from stack overflow (Fraxel). Another program was created by us to change the dimensions of the image using the OpenCV module. Using these programs, we were able to create new, smaller images. Figures 4, 5, and 6 show these new images formed from figures 1, 2, and 3 respectively.

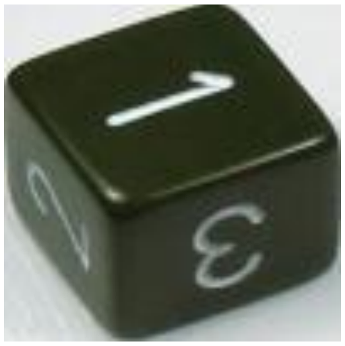


Figure 4. D6 Angled Image Cropped and Scaled to 100x100 Pixels

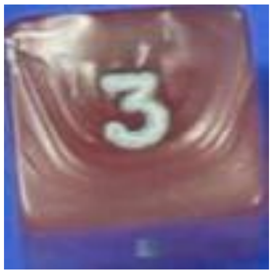


Figure 5. D6 Original Image Cropped and Converted to 100x100 Pixels

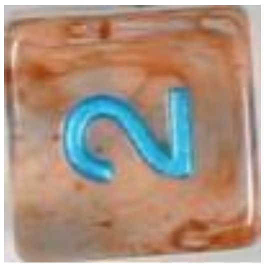


Figure 6. D6 Image Cropped and Reduced to 100x100 Pixels

Something to notice about these figures is that the color changed between each picture. This is because for the cropping, the picture was changed from RGB color to HSV color, and was never changed back. Once we have removed some of the irrelevant data like the background and changed the size to 100 by 100 to reduce the information even more to speed up the machine learning process, we changed each image to grayscale. The resulting images are shown in figures 7, 8, and 9, after converting to grayscale from figures 4, 5, and 6 respectively.

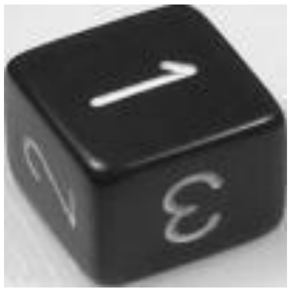


Figure 7: D6 Off Angled Image Cropped and Reduced to 100x100 Pixels and Converted to Grayscale

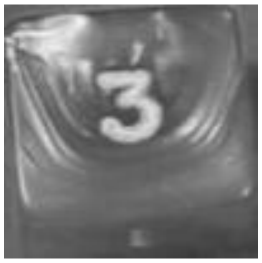


Figure 8. D6 Image Cropped, Reduced to 100x100 Pixels and Grayscaled

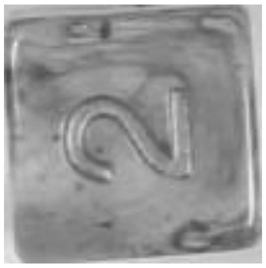


Figure 9. D6 Image Cropped, Reduced to 100x100 Pixels, and Grayscaled

The images were converted to grayscale using the .convert() function from the PIL module. Converting the images took a color array with a length of 3 and converted it to one value, giving the machine learning program only one value to look at later in the image for each pixel rather than 3. This theoretically would make the machine learning go three times as fast. These were the three major changes made to the images itself to make our machine learning program run the most efficiently.

Sorting Dice Images Manually Into Files

We downloaded each type of dice (d6, d8, d10, d12, and d20) folders as they were given on the Kaggle website. There were 2 folders given: training and validation. The training folder contained 5 separate folders, one for each type of die. These folders contained thousands of images from all angles. About every 8th image taken by the creator of the data set was placed into a validation folder. After downloading the Kaggle set and editing the images, we created folders for each die according to the number of sides; for example, in the d8ValidationSet folder, there were 8 folders created which represent various different images of each of the 8 sides. The folders were named “d8_1”, “d8_2”, “d8_3”, etc, all the way until “d8_8”. We transferred each image into the appropriate folder determined by the upsides of the dice.

Creating a Convolutional Neural Network

To create the convolutional neural network, we used 5 different types of layers in our model: Convolution layer, max pooling layer, flatten layer, dropout layer, and dense layer. The convolution layer convolves an input and passes it to the next layer. In our convolution layers, three by three filters were convolved with the original images. The max pooling layer clusters pixels into two by two matrices and creates one new pixel holding the highest value from the matrix. The flatten layer converts a matrix of pixels and puts them in a straight line. The dropout layer sets random input units to 0 to help reduce overfitting, that is, working well on the training set but performing poorly on the validation set. The dense layer uses a function to create new output values. The last dense layer has a softmax activation that converts the input to 20 outputs ranging from 0 to 1 where all the outputs add up to 1. These outputs can be interpreted as probabilities. The number with the highest probability is interpreted as the roll of that die. Adding more convolution layers increases the parameters which increases the run time of the machine learning program. Adding the dense layer with too big an input size can greatly increase the parameters which also increases the run time. In our machine learning programs, we had between 3-5 convolution layers and our dense layer was not run until the image had been convolved to a 9 by 9 pixel matrix or smaller.

Adjusting the Layers and Parameters

After our initial program for each type of dice was created, the program was overfitting the data. To correct this, we used two different methods. Either changes were made to the layer parameters, or regularizers were added to the program. Changing the parameters of the layer can increase the amount of computational power the program requires, but it allows the program to more accurately classify the top facing number on the die. Adding regularizers gives a penalty to the computer on layer parameters that is then added to the loss function to improve the program later. It ensures the computer is looking at the correct features of the image to classify the top facing number and maximizes the validation accuracy by taking a slight drop in the initial training accuracy growth.

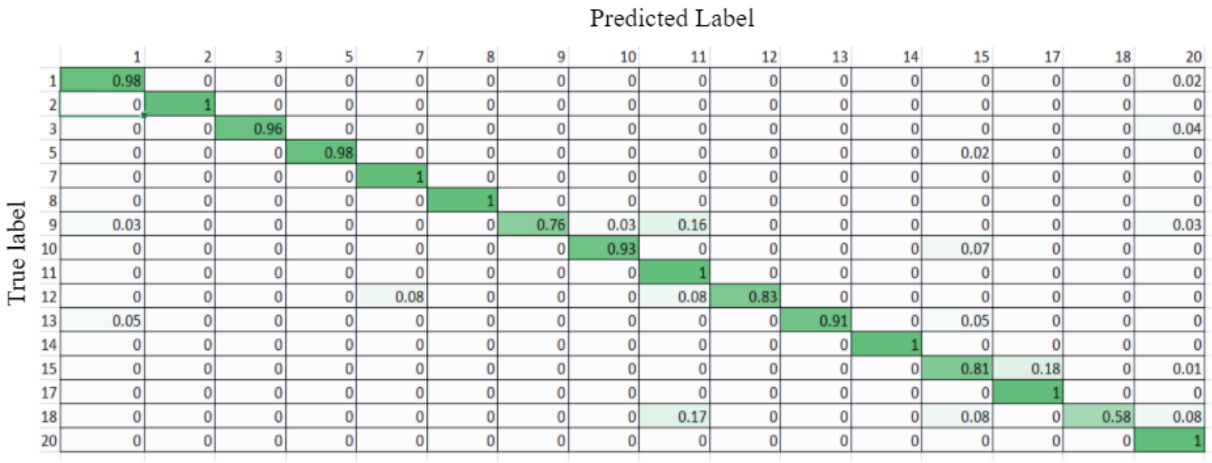
Results:

Table 1. Accuracies Across Various Types of Die

Dice Type	Training Accuracy	Validation Accuracy
d6	99.34%	99.76%
d8	99.75%	99.12%
d10	98.90%	95.12%
d12	97.22%	97.41%
d20	99.07%	95.40%

Table 1 displays the final training and validation accuracies for each dice respectively. There appears to be a correlation between the validation accuracies and the number of sides for each dice. This is very likely due to the machine learning program having more outputs to choose from when classifying the dice in the last layer. Despite this observation, the d10 dice had the lowest validation accuracy. This could be due to their irregular shape that causes many different numbers to show up in the cropped image, making it harder for the model to differentiate between which number is on the top face at different angles. Our goal to reach 95% accuracy for all dice was achieved as seen from table 1.

Using an online tensorboard program, we derived a confusion matrix to show specifically how our machine was doing. (Ashkitar) A confusion matrix for one of our several attempts for the d20 dice is shown in figure 10 below.



Predicted Label

True label	1	2	3	5	7	8	9	10	11	12	13	14	15	17	18	20
1	0.98	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.02
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.96	0	0	0	0	0	0	0	0	0	0	0	0	0.04
5	0	0	0	0.98	0	0	0	0	0	0	0	0	0.02	0	0	0
7	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
9	0.03	0	0	0	0	0	0.76	0.03	0.16	0	0	0	0	0	0	0.03
10	0	0	0	0	0	0	0	0.93	0	0	0	0	0.07	0	0	0
11	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
12	0	0	0	0	0.08	0	0	0	0.08	0.83	0	0	0	0	0	0
13	0.05	0	0	0	0	0	0	0	0	0	0.91	0	0.05	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0.81	0.18	0	0.01
17	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
18	0	0	0	0	0	0	0	0	0.17	0	0	0	0.08	0	0.58	0.08
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 10. Confusion Matrix of D20

A confusion matrix works by plotting the true dice labels vs the predicted dice labels. Unfortunately, this d20 dice confusion matrix is missing labels for 4, 6, 16, and 19 because they were not provided in the images in the dataset from Kaggle. Each cell shows the percentage of dice that were predicted to be that specific dice label out of the true dice labels. For example, our program predicted 98% of the dice with side 5 to be side 5, and 2% of the dice with side 5 to be side 15. Ideally, all true dice labels vs predicted dice labels should be alongside the diagonal stretching from the most top left to most bottom right cell. This is mostly the case, but some dice labels are not as successful. Labels with 9, 12, 15, and 18 did not have a 90% accuracy or higher in their predictions. There were fluctuations in which dice labels were inaccurate after each epoch, so it is likely that the program corrected other dice label predictions at the expense of those four dice labels.

<u>Conclusions:</u>

The above results proved a way to show that dice can be interpreted in many different ways. The way this was done was using convolution neural networks (CNN) which produce very high accuracies as shown previously. While in the preparation phase of the project, we were trying to figure out which form of identification such as edge detection through OpenCV would be the most accurate for identifying the number on the side of a dice; however we could not find a viable way to use it, so we found a method on stack overflow. For games such as “Memoir’44” and “Axis and Allies” that require extensive dice rolling, our AI will be able to speed up the time needed to identify the number on the dice side. Games will become much bigger and with a system that allows for a bigger reach of the audience, the casino industry would grow.
Some improvements that would increase the accuracy would be changing the cropping method of the dice. With the Kaggle dice set used, several images had a background on a wood top. These did not crop well and so they hurt the performance of the machine. A change that wasn’t initially noticed was the conversion of dice images from RGB to HSV. It is unknown whether or not this affected the results of our research. Another improvement could be having a consistent background set up ourselves which would make cropping easier. A big issue was some dice sides missing for the sets taken from Kaggle. Having all sides would provide a more accurate and true machine learning process.

<u>References</u>

Ashtikar, S. (2020, October 12). “Exploring Confusion Matrix Evolution on Tensorboard,” Medium, https://towardsdatascience.com/exploring-confusion-matrix-evolution-on-tensorboard-e66b39f4ac12.

Chollet, F. (2018). *Deep learning with Python*. Shelter Island, NY: Manning Publications.

Fraxel. (1961, February 01). “Trim whitespace using PIL,” Retrieved April 17, 2021, From https://stackoverflow.com/questions/10615901/trim-white

-space-using-pil

Geron, A. (n.d.). *Hands-on Machine Learning with Scikit-Learn, Keras, &TensorFlow* (2nd ed.). O'Reilly.

Lurig, M. (2018, June 21). Dice: D4, d6, d8, d10, d12, d20 images, Retrieved October 16, 2020, https://www.kaggle.com/ucffool/dice-d4-d6-d8-d10-d12-d20-images/