Classifying Polyhedral Dice

Neel Shah (1002241852)
Institute for Aerospace Studies, University of Toronto
Toronto, CA
neel.shah@mail.utoronto.ca

Jiaxin Chang (1002237867)
Institute for Aerospace Studies University of Toronto
Toronto, CA
jiaxin.chang@mail.utoronto.ca

Zehua Wu (1003217752)
Institute for Aerospace Studies University of Toronto
Toronto, CA
zehua.wu@mail.utoronto.ca

Abstract

Densenet121, Alexnet and Resnet18 were trained to classify d4, d6, d8, d10, d12 and d20 dice. The models were trained with logistic cross entropy loss and stochastic gradient descent over 25 epochs with a learning rate of 0.001, decaying by 0.1 every 7 epochs. Alexnet had an overall accuracy of 69%, Resnet18 had an accuracy of 96% and Densenet121 had an accuracy of 98%. Densenet121 was chosen as the final model as it had the best accuracy for the classification task.

1 Introduction

For this project, a dataset of various polyhedral dice was investigated. The goal was to train an image classification model to distinguish between d4 (tetrahedron), d6 (cube), d8 (octahedron), d10 (pentagonal trapezohedron), d12 (dodecahedron), and d20 (icosahedron) dice. This report will first introduce the problem and the dataset. Then, a brief literature review on relevant techniques and related papers will be presented. Next, the final model chosen to perform the classification will be explained. Lastly, the results and model performance will be discussed.

2 Problem Setup

The dataset studied is an image dataset containing pictures of various types of dice. A sample photo from each class of dice is shown in figure 1.

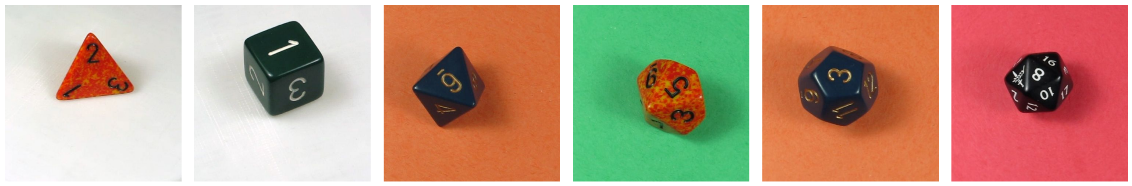
(a) d4 dice (b) d6 dice (c) d8 dice (d) d10 dice (e) d12 dice (f) d20 dice

Figure 1: Sample pictures from 6 types of dice being classified

The dataset was retrieved from the Kaggle dataset database found here [1]. Some basic statistics of the training set are:

- There are 14,284 images in the training set which consist of 6 different dice shapes, each with the following number of images
  - # of d4 images = 1947 = 13.6 % of entire data set
  - # of d6 images = 4046 = 28.3 % of entire data set
  - # of d8 images = 1532 = 10.7 % of entire data set
  - # of d10 images = 1722 = 12.1 % of entire data set
  - # of d12 images = 1691 = 11.8 % of entire data set
  - # of d20 images = 3346 = 23.4 % of entire data set

This is interesting as some classes are more represented than others. This discrepancy allows for two informative baselines to compare the trained model to. Firstly, a naive baseline can be constructed by randomly choosing a number between 0 and 5 and assigning the test case to the number chosen (if 0 represents d4, 1 represents d6 etc). This baseline is extremely naive because it does not account for the relative proportions of the dice in the training set. Accounting for this can be used to construct a weak baseline, where instead of choosing a random number between 1 and 6 the number is chosen to be proportional to the training dataset.

All the images for d4, d8, d10 and d12 are 480x480 pixels in the training and testing set. All of the images for d6 and d20 are 480x480 in the training set but some of the images in the test set are 1024 pixels on the long side. This means the model chosen must be able to handle pictures of varying sizes so a method like one used for MNIST (kNN) would probably not be suitable. Furthermore, the numbers visible are not always the same, meaning if the classifier simply identifies the dice by which numbers it can recognize in the picture the model will not be generalizable. To remedy this, the training set must be set up to contain images of dice with different colours, backgrounds and orientations (different numbers visible). This is achievable as the dataset was generated by taking videos of 5 different dice with 6 different backgrounds on a rotating platform. The video was then cropped to be 480x480 and exported as images. Additional cases were made by recording 5+ dice on wood surfaces (minimum 2) with handheld camera moving over them and exporting them as before. The d6 and d20 images with different dimensions were added to dataset as a robustness test. They were created similarly to the data described above but there was no attention payed to the lighting [1]. An example of the robust d6 test case is shown below.

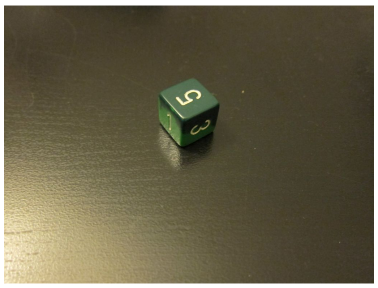

Figure 2: d6 dice robustness test case

3 Literature Review

There are many different approaches that were considered to solve the problem posed above. This section will review literature associated with both deep learning approaches and so called shallow learning approaches.

For starters, the problem described above can be approached similar to the MNIST data set which is typically handled using a kNN model. This has the benefit of no training time. Additionally, k is the only hyperparameter that requires validation, where as deep learning architectures can have many difficult to tune hyperparameters [2]. Lastly, using kNN would make the set up for the problem extremely straightforward as the feature vector would simply be the pixel brightness values. However, there are many important drawbacks to this method. Firstly, since this datset is a 480x480 grey scale photo, the feature vector will have 480x480 = 230,400 features which makes the model susceptible to the curse of dimensionality. Additionally, the robust test images included in the dataset make using kNN impossible because those images will have 480x1024 = 491,520 features. This mismatch in dimensions makes comparing distances impossible. For these reasons a kNN framework was not developed further.

Nowadays, leading approaches for image processing, image classification and computer vision competitions have been based on convolutional networks (CNNs) [3]. Khan et al. [5] reviewed and compared the state-of-art CNN models. They broadly categorized these models into seven classes, namely, spatial exploitation, depth, multi-path, width, feature-map exploitation, channel boosting and attention-based CNNs. In order to propose a model with good performance, three popular CNN architectures from different categories are selected and compared. Specifically, the three models selected are AlexNet from spatial exploitation-based CNNs, ResNet from depth-based CNNs and DenseNet from multi-path CNNs.

4 Model Description

AlexNet was considered as the first deep CNN architecture, which initiated the popularity and development of deep CNNs in the past few years [5]. The main attributes of AlexNet are that it uses ReLU, avoids overfitting with the Dropout technique and reduces the feature dimensions using overlap pooling [6]. However, it is more computationally expensive and less accurate than many models developed later. ResNet introduced the concept of residual learning in CNNs and devised an efficient method for training deep networks [5]. It is less expensive and more accurate [5]. DenseNet, or densely connected convolutional networks, introduced direct connections between layers with same feature-size map[4], which improves the flow of information through the network. Both ResNet and DenseNet alleviate the vanishing gradient problem, while DenseNet addresses the issue in ResNet, where many layers have little or no information contribution [5]. Based on the time each model was developed, it is expected that DenseNet is likely to have the highest accuracy and AlexNet to have the lowest accuracy among the three.

For this project, the pre-trained AlexNet, ResNet18 and DenseNet121 models in PyTorch have been used [8].

5 Results

5.1 Training

For training, the 14,284 training images were split into a training set and validation set with a training-validation split of 80-20 respectively. This was done stochastically and performed before each training iteration so that the models could be cross-validated on the training set. Each model was modified so that the last layer's output size was 6 to fit the number of classes of dice that we had. The loss was calculated using a logistic cross entropy loss function and the model parameters were optimized using stochastic gradient descent. Each model was trained for 25 epochs with an initial learning rate of 0.001 decaying by a factor of 0.1 every 7 epochs.

5.2 Testing

As described in Section 2, naive and weak baselines were created in order to evaluate each model's performance. The naive baseline was able to achieve an overall accuracy of 16% while the weak baseline was able to achieve an accuracy of 19%. Each model was then tested on the 2102 test images and their respective accuracies calculated. **Table 1** shows the accuracies of each model on the test images split by each class as well as an overall accuracy on the whole set.

Table 1: Prediction accuracies generated on the test images for all three models

Accuracy [%] \ Model	AlexNet	ResNet18	DenseNet121
D4	50	98	100
D6	72	99	100
D8	94	95	95
D10	93	99	100
D12	63	98	98
D20	56	96	100
Total	69	97	98

We can see that AlexNet had the worst overall performance of the three models while ResNet18 and DenseNet121 showed comparable results although DenseNet121 was marginally better in several categories. One thing to notice is that the both ResNet18 and DenseNet121 have lower performance in the d8 category. This is thought to be the result of the shape of the d8 dice being composed of 8 equilateral triangles which makes it very similar to the d4 dice which is also composed of equilateral triangles. We don't see a corresponding drop in accuracy in the d4 dice because the d4 must sit on one of the face while in the case of the d8, the dice sits slanted on one of the side faces. Under certain image angles, the d8 could be mistaken to be a d4 since the angle hides two of the d8 faces. The reverse is not possible since the d4 sits flush with the surface.

Although marginal, the performance increase from ResNet18 to DenseNet121 was the reason that DenseNet121 was chosen as the final classification model. The final model was tested on several images of dice which were not in either the training or test sets and was able to correctly identify all the images tested. Additionally, the model was tested on various non-dice polyhedron objects including a d4 rubik's cube (called a pyraminx) and a d12 rubik's cube(called a megaminx). The model was able to correctly identify the d4 but was unable to identify the d12.

6 Conclusion

In conclusion, three CNN models were trained and tested on a dataset of various polyhedral dice in order to create an image classification model capable of distinguishing between d4, d6, d8, d10, d12, and d20 dice. Each model was trained on the training image set which was split into a training and validation set of 80% and 20% respectively in order to cross-validate the models. Each model was then tested on the test set images and the accuracy of the classifications was calculated. It was found that the three models, AlexNet, ResNet18, and DenseNet121 were able to achieve an overall classification accuracy of 69%, 97%, and 98% respectively. Thus, DenseNet121 was chosen as the final model for image classification. The DenseNet model was able to perform well in testing on dice images outside the data set but failed to generalized in some cases when tested on non-dice polyhedral objects. Thus, in the future, training images of other non-dice polyhedron objects could be included to help generalize the models in their classification.

7 Github Repository

The code for this project can be found in the Github repository here: Project Github

8 Attributions

Each group member trained and tested one of the aforementioned models. Neel Shah performed training and testing on the AlexNet model, Jiaxin Chang performed testing and training on the ResNet18 model, and Zehua Wu performed training and testing on the DenseNet121 model. Additionally, each member contributed to two sections of this report with Neel Shah writing sections 1 and 2, Zehua Wu writing sections 3 and 4, and Jiaxin Chang writing sections 5 and 6.

References

[1] *Dice: d4, d6, d8, d10, d12, d20 Images*. URL: `https://www.kaggle.com/ucffool/dice-d4-d6-d8-d10-d12-d20-images`.

[2] R Grosse. *CSC2515: Machine Learning Lecture 1 - Introduction and Nearest Neighbours*. Sept. 2020.

[3] R Grosse. *CSC421: Lecture10 - Image Classification*. Sept. 2019.

[4] G. Huang et al. "Densely Connected Convolutional Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2261–2269. DOI: `10.1109/CVPR.2017.243`.

[5] Asifullah Khan et al. "A survey of the recent architectures of deep convolutional neural networks". In: *Artificial Intelligence Review* 53.8 (2020), pp. 5455–5516.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: `10.1145/3065386`.

[7] *Saving and Loading Models*. URL: `https://pytorch.org/tutorials/beginner/saving_loading_models.html`.

[8] *TorchVision.Models*. URL: `https://pytorch.org/docs/stable/torchvision/models.html`.

[9] *Transfer Learning For Computer Vision Tutorial*. URL: `https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html#training-the-model`.