**Shape Classification Using Convolutional Neural Networks on 2D Geometric Shapes**

Izabella Kocsis and Sam Strickler

May 2, 2025

# Abstract

Image classification using Convolutional Neural Networks (CNNs) is widely recognized for its effectiveness in a variety of applications. One promising area where CNNs are utilized is the recognition of 2D geometric shapes. This relatively simple task is a small but crucial part of the CNNs’ role in image analysis. The ability to correctly classify 2D shapes can be built upon and utilized in more complex analysis. By developing a custom CNN and improving the accuracy in 2D geometric shape classification, we take a step closer to improvement in related areas of image analysis.

# Introduction

In recent years, deep learning has revolutionized the field of computer vision, allowing computers to discover and classify visual patterns with increasing accuracy. A fundamental application of computer vision is shape recognition, which serves as a foundation for more complex tasks like object detection, autonomous navigation, and other forms of visual pattern analysis. This project aims to develop and train a convolutional neural network to classify 17 different types of 2D geometric shapes. These shapes include circles, triangles, stars, and various polygons using the publicly available 2D Geometric Shapes dataset from Kaggle.

The goal of this project was to build a simple yet effective image classification pipeline using PyTorch. To do this, we used a CNN to extract and learn features from the dataset’s synthetically generated images. By completing this project, we aim to reinforce our understanding of deep learning principles like data preprocessing, model architecture design, training and evaluation, and performance optimization. Our hope is that this project will serve as a gateway into understanding more complex applied computer vision problems that we may encounter in the future.

# Related Work

One related work focused on classifying 2D shapes to then solve geometry math word problems. Although the broader aim was to support problem solving of geometric shapes with geometric properties, the underlying approach is insightful. The researchers implemented a two-layer hierarchical classification model: the first layer handled general classification into broad categories while the second layer sorted further into specific subtypes. Convolutional neural network architectures were implemented at each level, demonstrating strong performance. The key takeaway from this work is the effective use of CNNs for 2D shape classification, achieving accurate results [1].

The implementation of CNNs for image classification has proven to be highly effective. While many researchers design custom CNNs for their relevant research, there are also well-known and widely used CNN architectures. For example, AlexNet is a deep CNN utilizing rectified linear units (ReLUs) and dropout which improved performance on challenging datasets [2]. ResNet introduced residual connections which enhanced the baseline VGG architecture allowing it to win classification competitions in 2015 [3]. Based on the findings from popular CNN architectures like those mentioned above, a custom CNN can be designed for 2D geometric shape recognition.

In designing our model architecture, we also drew inspiration from Assignment 2, which introduced convolutional neural networks through an MNIST digit classification task, and the textbook Understanding Deep Learning. These resources provided foundational insights into CNN layer design, kernel sizing, number of filters, activation functions, and pooling operations, which guided the structure of our shape classification network. We adapted these principles to suit our multi-class image classification task involving the 2D geometric shapes.

# Methodology and Work

Although the images contained shapes rendered in varying colors, we retained the original RGB color channels rather than converting the data to grayscale. This choice preserved the full dimensionality of the input (3 channels per image), ensuring that the network could learn from the raw pixel distribution. However, since the classification task was focused purely on shape rather than color, the network learned to prioritize spatial and structural features over color differences. We resized all images to 64x64 pixels to reduce training time and computational load without sacrificing the model’s ability to detect key shape features. This stuck a balance between efficiency and performance.

Our final CNN architecture consists of three convolutional blocks, each with 2d convolutional layers followed by a ReLU activation function and a max pooling layer. We used 32, 64, and 128 filters in our three convolutional layers to progressively capture features of increasing complexity, starting with simple edges and progressing to detailed shape characteristics. We chose a kernel size of 3x3 since we found that it was a common standard that balances detail and computational cost, which was important in this project. We used a stride of 1 to preserve spatial resolution in the early layers of the network, making sure that no important features were skipped. We also set the max pooling layers to a size of 2x2 and stride 2 to reduce the size of feature maps, also helping with the number of computations and to prevent overfitting. We found that increasing the number of filters at each layer allowed the network to learn richer, more abstract features critical for distinguishing between shapes such as decagons and circles. Before tuning this, the model struggled to differentiate between high dimensional polygons and circles. The use of ReLU introduced non-linearity, allowing the network to model more complex decision boundaries. While dropout layers were initially included to explore regularization, they were excluded from the final model as we observed no signs of overfitting in training or validation performance. In other words, we did not notice the model performing well on the training data and not generalize well to the test data in any of our test runs. The final layer applies a SoftMax activation function, converting raw output logits into a probability distribution over the 17 shape classes, suitable for multi-class classification.

For training, we employed the Adam optimizer with a learning rate of 0.001, due to its adaptive learning rate capabilities and strong performance in practice. After tuning other hyperparameters, we did not feel the need to experiment with different learning rates. Cross-entropy loss was used to quantify the discrepancy between the predicted and true labels. The model was trained using mini batches of size 32 over 12 and 13 epochs in different runs. We found that the model naturally performed better with more training data and learned enough without overfitting with 13 epochs. Throughout the training process, we recorded accuracy and loss on both the training and validation sets to monitor learning progress. These metrics were visualized to better understand convergence behavior and generalization.

Following training, the model was evaluated on an unseen test set. We compared training and test performance to assess the model’s ability to generalize to unseen data and ensure that it had not overfit the training set. Final evaluation included a visual inspection of predictions and confidence levels to interpret the model’s strengths and limitations.

# Dataset

The 2D Geometric Shapes is a publicly available dataset that models synthetic geometric figures. Here, 2D images of geometric shapes are generated with random position and orientation, the images are of size 224 x 224. There are seventeen different figures which include figures like circles, ovals, triangles, rhombuses, and decagons. There are a total of 850,000 images, so 50,000 images per shape. The background is white, and the color of the shape is randomly assigned from a predefined set of colors. The data is organized in folders, where each folder’s name is the label for the images within.

# Evaluation

To evaluate the effectiveness of the CNN we tracked the accuracy and loss values for both training and testing conditions. The accuracy is calculated as the number of correct predictions over the total number of predictions. Since the dataset is relatively simple, an accuracy above 90% was expected, which we obtained. The loss on both the training and testing sets measured and assessed how well the CNN learned. The training and test loss values decreased steadily over time. The validation loss specifically plateaued at a low value of 0.0289, near zero, indicating that the model has learned successfully. Based on the capability of well-known CNNs like AlexNet, ResNet, and considering the 2D geometric shape dataset is quite simple, we achieved our goal in that our unique CNN is accurate for A graph with different colored lines

AI-generated content may be incorrect.at least 90% of all test classifications.

# Timeline

The project was completed over the course of three weeks and the timeline was divided into three phases. The first week was the initial implementation of the CNN for 2D shape classification. The data set was imported and prepared, the model designed, and training began. The second week required fine-tuning of our model to achieve the desired accuracy, including hyperparameter tuning and testing regularization techniques to improve accuracy on the test set. The third week required a few additional adjustments and a comprehensive report on the project and findings. This included our presentation. The timeline allowed for sufficient time and planning to complete all aspects of the project: implementation, optimization, and final reporting.

# Contributions

This project’s development was evenly split across both of us. We initially hunted for the dataset together and selected the one used to train our model. Sam implemented the necessary modules, data imports, and split the dataset 80/20 for training and testing purposes. When we started evaluating possible approaches for our model, we worked together to implement the convolutional network’s class. We debugged the implementation together, with each of us trying different filter sizes in our respective notebooks. We often used copies of the same notebook to test out all changes including dropout, l2 regularization, and dataset size. We also implemented the training and validation section of our report together, utilizing Gemini within collab to debug any errors in our code. Izabella then handled the data visualization of our training and testing statistics, as well as the visualization of our model’s final predictions.

# Conclusion

The accurate classification of 2D geometric shapes through the implementation of a convolutional neural network was the primary goal of this project. We aimed to accomplish a high accuracy in classifying 2D shapes across 17 different classes, and we did with a validation accuracy of 99.12% and a final loss value of 0.0289. Our specific target benchmark was that the CNN should produce a test accuracy that exceeds 90%, with the model almost always correctly classifying the shapes. By increasing the number of filters and using a subset of 200,000 images for training and testing, we found that these tweaks influenced our model’s performance the most over techniques that target overfitting, such as dropout and regularization. We searched Kaggle’s website to see if anyone had posted their loss or accuracy values from their respective models but could not find any.

# References

Dataset: <https://www.kaggle.com/datasets/khalidboussaroual/2d-geometric-shapes-17-shapes>

1. Boob, A., Radke, M. Leveraging two-level deep learning classifiers for 2D shape recognition to automatically solve geometry math word problems. Pattern Analysis & Applications, 2024.
2. Krizhevsky, A., Sutskever, I., & Hinton, G. E. ImageNet classification with deep convolutional neural networks. Advances in neural information processing systems, 2012.
3. He, K., Zhang, X., Ren, S., & Sun, J. Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition, 2015.
4. Dey, S. Understanding Deep Learning: Application in PyTorch. Independently published, 2021.