Multi-Layer Perceptron Algorithm for Image Classification of Simple Shapes

I. Introduction

Machine learning algorithms are used to classify images in many industries, whether it be road detection in autonomous driving, diagnosing medical conditions in medical imaging, or facial recognition for security services. The capabilities of machine learning services for image classification vary depending both on the hardware of the machines performing the calculations and the software or code that is being performed. Learned from this introduction to machine learning, multi-layer perceptron (MLP) models, while capable of image classification, are best utilized on simple, small images. Accordingly, this project examines the image classification capabilities of neural networks by attempting to create an algorithm that will classify 4 different simple shapes (square, rectangle, circle, triangle).

The idea behind selecting this problem is to create a fully-automated activity for children to learn how to draw and identify different shapes. During the COVID-19 pandemic, parents have struggled to both entertain their children while working from home and to ensure their children are getting the education they would otherwise get in pre-pandemic times at their regular schools. The solution of an app providing a fun, interactive environment to both teach and practice drawing shapes will provide a first cut solution to keep children occupied and learning at the same time.

This report consists of eight key sections:

(1) Introduction
(2) Description of the Data
(3) Experimental Setup
(4) Description of the Algorithms Used
(5) Results
(6) Summary and Conclusions
(7) References
(8) Appendices

The code generated for this code consists of two parts: the generation of the MLP classification model and the interface through which a user can generate a shape and request the classification of said shape. The code used to produce this analysis can be found here: https://github.com/sacohen11/Final-Project-Group2.

II. Description of the Data

The data used within this report to generate the first MLP model is all self-generated JPG files. These JPG files are formed from:

- Digitally-drawn shapes via a digital trackpad and simple recording software,
- Replicated and augmented drawn shapes in Adobe Photoshop, and
- Shapes generated from the patch and line libraries of MatPlotLib within Python.

All images are generated on an 80-by-80 pixel canvas upon which the shape is drawn or generated. The proportion of digital, augmented, and Python-generated images in the original model are 30-10-60, respectively. Examples of each digitally drawn and Python-generated shape (circle, rectangle, square, and triangle) can be seen in *Appendix A*.

Following the generation of the MLP classification model, users can generate new images. When the model correctly classifies the generated shape (confirmed by the user), the model is unaffected; however, if the model improperly classifies a generated shape, the interface prompts the user to identify the correct classification, and the new image is saved into the image archive of the appropriate classification. In this way, future models also contain originally-misclassified, user-generated shapes in order to improve the performance of the model.

III. Experimental Set-Up

The training set of image shapes may not contain the same amount of images from each shape classification. For this reason, prior to model training, the image dataset is augmented with additional image shapes equalizing the number of shape images between all shapes. Additional augmented images are formed by performing one of the following manipulations to a given image within the dataset:

- Adding “salt-and-pepper” noise to the image (randomly converting pixels to black or white),
- Rotating the image 90 degrees,
- Rotating the image 180 degrees,
- Flipping the image about the Y-axis, or
- Translating the image within the canvas space.

In addition to image augmentation, image pixel values are thresholded so all pixel values are either white or black (no grey) and images are cropped and expanded such that the minimum square border (equal height and width) encompassing the shape, however drawn, becomes the new full image. That is, the position of the shape within the image canvas canvas is removed and only the orientation of the shape about the x or y-axis is conserved. Examples of image cropping of a triangle shape can be seen in *Appendix B*.

IV. Description of the Algorithms Used

This project examined the Multi-layer Perceptron (MLP) classifier model. As each image is an 80-by-80 grey-scale pixel image, when flattened and prepared for model use, each data point (image) has 6400 features corresponding to each pixel within the image.

Following data augmentation and image cropping, the images are fed into an MLP Classifier object as part of SKLearn library in Python. The inputs into this classifier object include:

· Activation Function:	a = f(x) = max(0, x)
· Single Hidden Layer:	100 neurons
· Solver:	Stochastic Gradient Descent
· Alpha:	0.0001

With 100 neurons within the single hidden layer in use, the MLP model is a 6400-100-1 MLP.

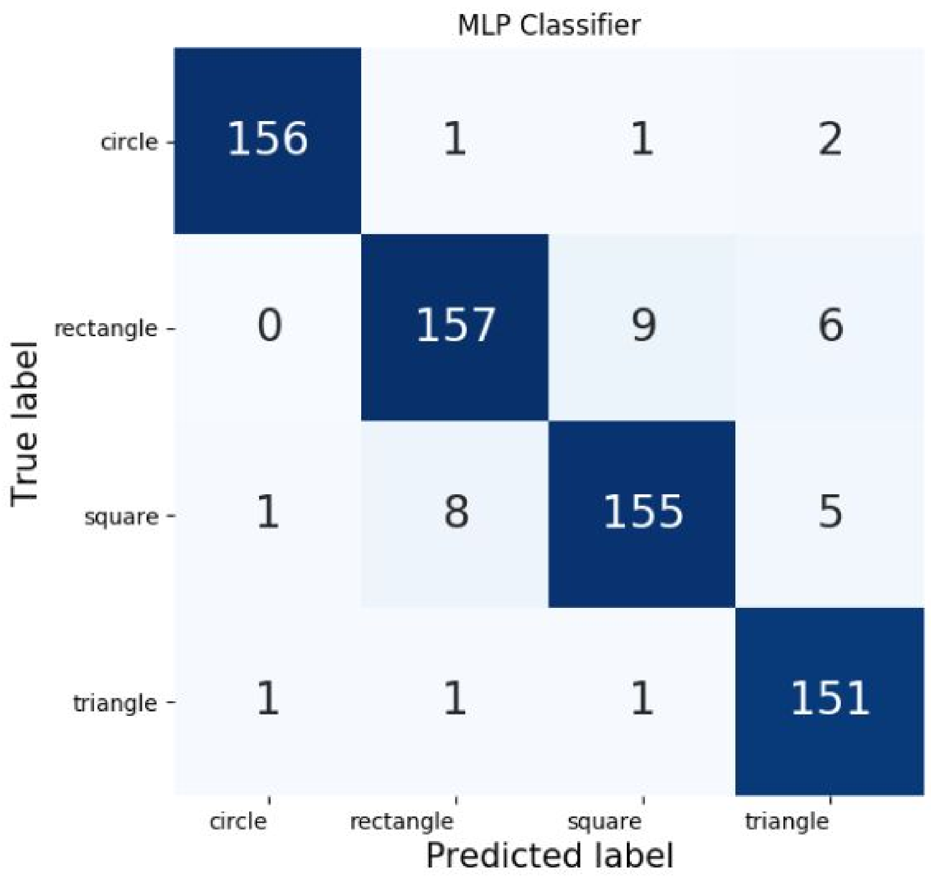
To judge the results and success of each model, the team decided to use accuracy as its measure. Accuracy is defined as:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Number\ of\ Total\ Predictions}\ X\ 100\%$$

The team also viewed confusion matrices to identify for which classifications were the models predicting better or worse.

V. Results

The accuracy for the multi-layer perceptron classification algorithm, as described above, is 94.5%. The confusion matrix is shown in Table 1 below.


MLP Classifier

True label \ Predicted label	circle	rectangle	square	triangle
circle	156	1	1	2
rectangle	0	157	9	6
square	1	8	155	5
triangle	1	1	1	151

Table 1: Confusion Matrix

The accuracy is very high. In the majority of cases, the MLP Classifier model is correctly predicting the shape drawn.

Looking at the confusion matrix, it seems there are two areas in which the model can improve. First, the model classified rectangles as squares nine times and squares as rectangles eight times. Misclassifications between squares and rectangles were expected, given the similarities between the two shapes and the hand-drawn quality of the images. Lastly, the model classified rectangles as triangles six times and squares as triangles five times. These misclassifications are occurring due to variations in the angle of squares and rectangles. For instance, if a square is rotated to a diamond-shape, the MLP model is often classifying that rotated square as a triangle. This makes sense, as the model recognizes the traditional “point” of a triangle in the rotated square. Looking at the case of misclassified rectangles, if a rectangle is drawn with two long sides and two short sides, the model often doesn’t recognize one of the short sides and misclassifies it as a triangle.

To improve these misclassifications, users can draw “odd” squares and rectangles and add them to the image database. If enough images are added that were not previously recognized by the model, then the model will learn the new images and model accuracy will improve.

VI. Summary and Conclusions

In summary, the model performs well at classifying simple shapes. With a 94.5% accuracy level, the model can be used to teach children basic shapes with a high degree of certainty that the model will not be teaching children incorrectly. In addition, the feature of the application in which users can add an image to the database if it was classified incorrectly will allow the model to continue improving in accuracy, improving its education value for children.

For future research, the team plans to build an IOS/Android compatible mobile application for children to use on mobile devices and tablets. Most mobile apps that educate children on colors, shapes, animals, and more don’t actually allow children to draw the item of classification. Instead, children choose the category that they see in a picture. By allowing children to draw the item and the computer to classify it, we are reversing the traditional learning method and allowing children to take a more active part in their learning. In addition, the team would like the application to be able to detect more shapes, possibly with edge detection techniques. Lastly, the team would like to develop a mechanism to double-check the user-input classification. For example, if a user draws a circle and the machine classifies it as a square, the user will “re-classify” the item as a circle and add it to the database. The team would like to develop an algorithm that determines the probability that the user’s classification is the correct one. There are many avenues for the team to take this project and scale it to something new.

VII. References

Research References

OpenCV-Python Tutorials. (n.d.). Retrieved April 28, 2020, from https://docs.opencv.org/trunk/d6/d00/tutorial_py_root.html

Code References

Bhatnagar, T. B. T. (2018). Resize an image without distortion OpenCV. Retrieved April 28, 2020, from https://stackoverflow.com/questions/44650888/resize-an-image-without-distortion-opencv

Python cv2.getRotationMatrix2D() Examples. (n.d.). Retrieved April 28, 2020, from https://www.programcreek.com/python/example/89459/cv2.getRotationMatrix2D

SanchitSanchit 2. (2014). How to add noise (Gaussian/salt and pepper etc) to image in Python with OpenCV. Retrieved from https://stackoverflow.com/questions/22937589/how-to-add-noise-gaussian-salt-and-pepper-etc-to-image-in-python-with-opencv

Translation of image. (n.d.). Retrieved April 28, 2020, from http://wiki.lofarolabs.com/index.php/Translation_of_image

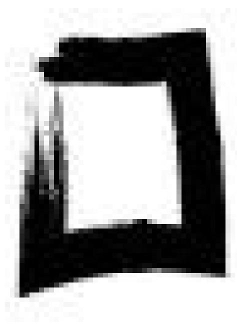
imneonizer. Find and crop objects from images using OpenCV and Python, from: https://github.com/imneonizer/Find-and-crop-objects-From-images-using-OpenCV-and-Python/blob/master/crop_objects.py

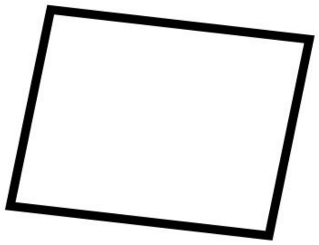
Fitzpatrick, M. (n.d.). *QPainter and Bitmap Graphics*. https://www.learnpyqt.com/courses/custom-widgets/bitmap-graphics

VIII. Appendix A

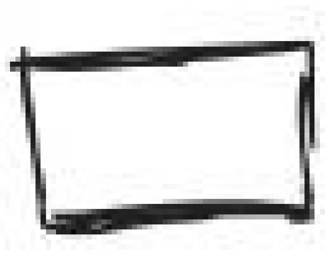
Example Images of Each Shape and Generation Algorithm

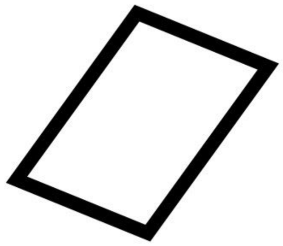
Square Images: Digitally-Drawn (left) and Python-Generated (right)



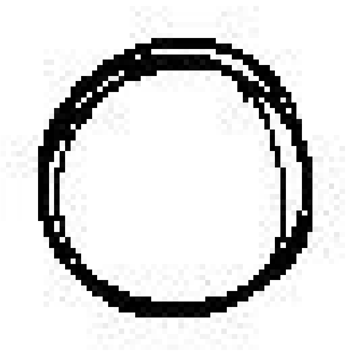
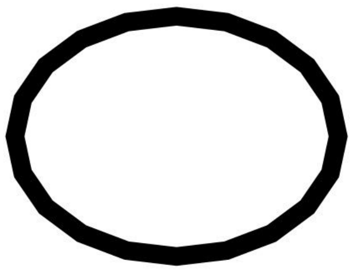


Rectangle Images: Digitally-Drawn (left) and Python-Generated (right)

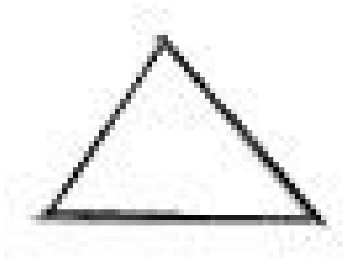
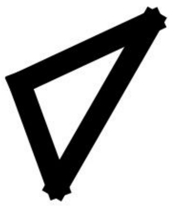




Circle Images: Digitally-Drawn (left) and Python-Generated (right)

Triangle Images: Digitally-Drawn (left) and Python-Generated (right)

IX. Appendix B

Example Image of Image Cropping of a Triangle Shape3