# Geometric Shape Detection

## Project Report

**Author: Anant Bhide**

# Introduction (Problem)

In this project, I am training a smart model that can classify a shape depicted in the input image into 6 categories, namely, None - no shape found (1), Circle (2), Triangle (3), Square (4), Pentagon (5), Hexagon (6). The primary task is multi-class image classification, where the goal is to identify the type of shape in the provided image. I will be using 3 ML algorithms, comparing them with each other and making logical analysis based on the results I get.

# Dataset

I am using the Geometric Shapes Dataset available online on the Hugging Face platform. It is a synthetic dataset containing images of various geometric shapes with superimposed random text. Each image features a random shape among the categories mentioned above on a randomly colored background, with a short string of random characters partially obscuring the shape.

The dataset contains 14.7K training sample, 2.1K validation samples and 4.2K testing samples. Each instance in the dataset consists of an image (50x50 pixels, RGB) and a label indicating the type of shape.

# Setup

I downloaded the data locally using python's dataset library. As this is a classification task, I will use these ML algorithms: logistic softmax regression, MLP classification and CNN.

(Note: Run the local_data.py script before running any other scripts, if needed. It will save the 2D geometry dataset I am using under a shapes dataset folder. I have attached the dataset with my submission)

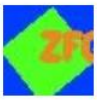
I have defined two important functions in load_data.py script: preprocess_image and prepare_split. These functions are used in every algorithm's script to preprocess the data, split it, train and then test. I convert the images to grayscale, resize, normalize and flatten them (except CNN) before inputting them to the ML model.

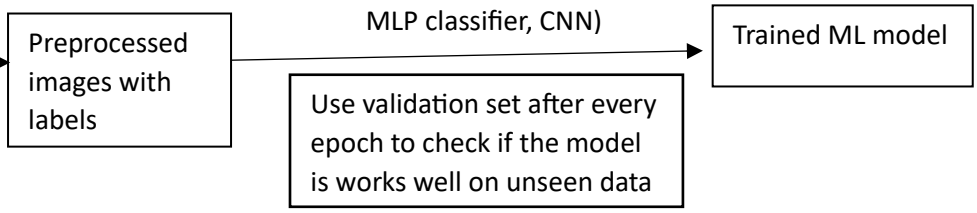
For every algorithm I used these lists to determine best pairs of hyperparameters, and picked one that gave me the highest validation accuracy.
batch size = [64, 128, 256, 512, 1024]; learning rates = [0.02, 0.01, 0.001, 0.0005, 0.0001]; regularizations = [0.001, 0.0005, 0.0001]

Once I have the trained model, I use the testing data on it to get the prediction classes and then use the numpy mean function to get the final accuracy: np.mean(predictions == labels).

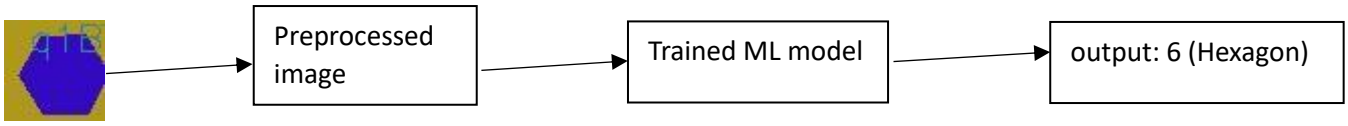
This is how the whole road-map of my project looks like -

Training process:

| Shape Name | Image |
|---|---|
| None |  |
| Circle |  |
| Triangle |  |
| Square |  |
| Pentagone |  |
| Hexagone |  |




After deployment:




I preprocess the image using the functions in load_data.py script and then pass it to our model. The output is a label number as follows:

| Label | Shape Name |
|---|---|
| 1 | None |
| 2 | Circle |
| 3 | Triangle |
| 4 | Square |
| 5 | Pentagone |
| 6 | Hexagone |

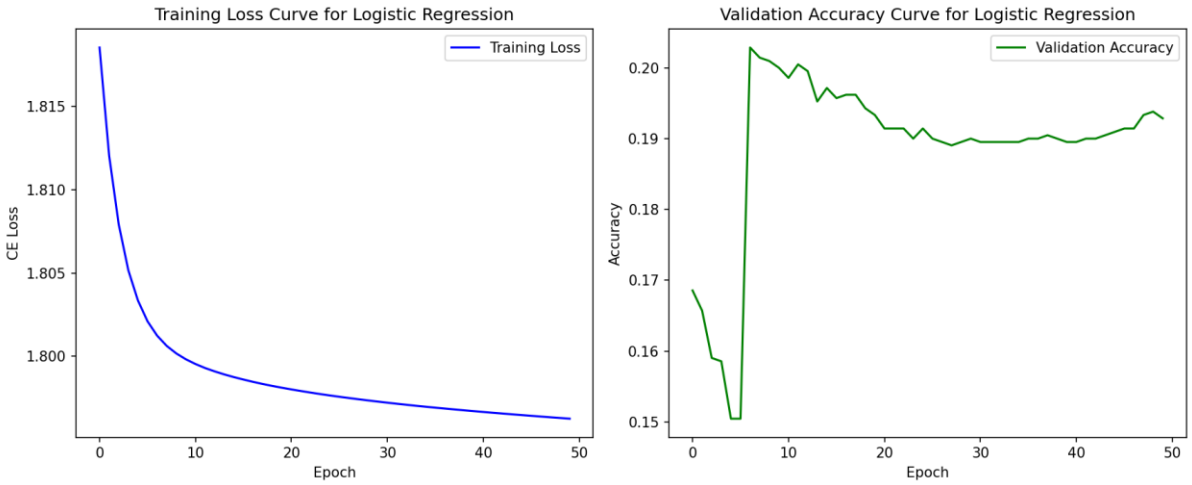
# Algorithms Used

## 1. Logistic Regression with Softmax

As this is a classification task, I decided to use the most straight forward approach: logistic regression. This algorithm has been implemented in logistic_regression.py script. I am using the cross-entropy loss with l2 regularization to prevent overfitting. I used the following settings for this algorithm:

learning rate = 0.0001
epochs = 50
batch size = 512
regularization = 0.001

The script gave me following results:




Best Epoch: 6

Corresponding Validation Accuracy: 0.20285714285714285

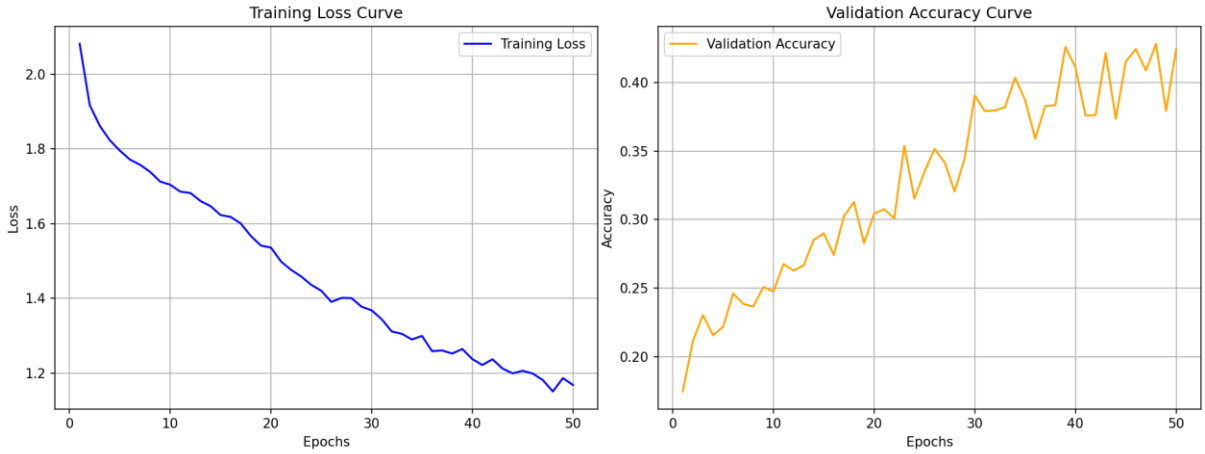
Logistic Softmax Regression Test Accuracy: 0.1990

Analysis: We can see that the multi-class logistic regression gave us around 20% accuracy which is very low for a classification task. This is probably because logistic regression is linear in nature and cannot really capture the complexity of the input image vector. As this could be the most basic approach, we can have accuracy of 20% as our baseline.

2. **MLP (multi-layer perceptron) classification**

My next approach was to induce some non-linearity so that the model can learn the images better: use a MLP classifier. This algorithm has been implemented in mlp_classifier.py script. In here the input vector is flattened into a 1D vector. Then I use 3 ReLU activation layers and softmax for the final output layer. Also, Adam's optimizer is used. For this task, these parameters seem to work the best:

learning rate = 0.001 (for Adam's optimizer)
epochs = 50
batch size = 512
regularization = 0.0001 (for dense layers)




Best Epoch: 48

Corresponding Validation Accuracy: 0.4281

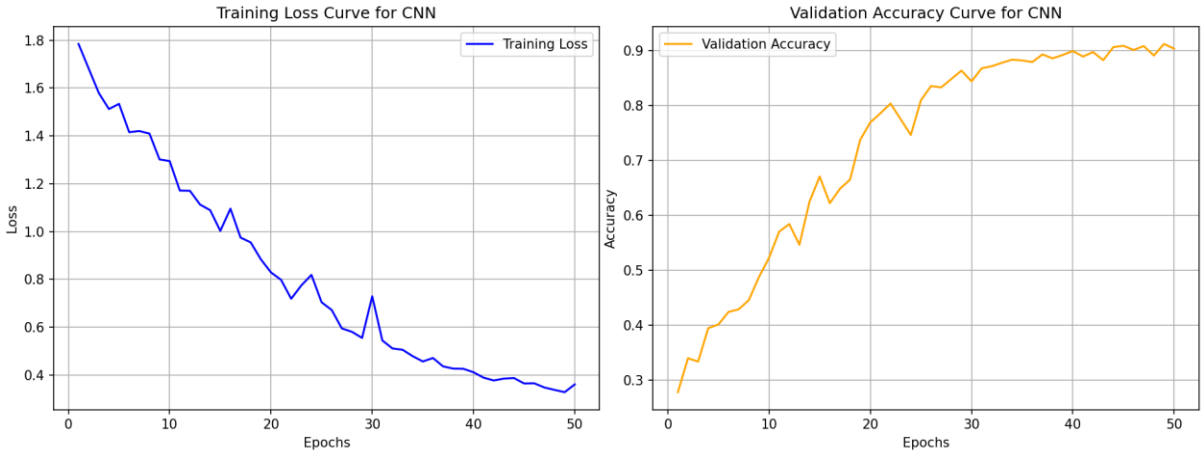
MLP classifier Test Accuracy: 0.4310

Analysis: It seems that MLP is giving us a better accuracy (43%) that logistic regression as it introduces non-linear functions. We got more than twice accuracy using MLP classifier compared to the baseline. Now the question is can we increase the accuracy more? In the both the approaches I have used so far, the input to our models is a flattened vector. Thus, both these ML models don't really consider the spatial relationships between the pixels of the images. Therefore, to better capture these relations we want a model that can directly take the image as the input, CNN!

3. **CNN (convolutional neural network)**

I am using a CNN as the ultimate tool that can directly take the preprocessed input image, instead of a vector. This will help model to learn spatial relationship between the pixels. The algorithm has been implemented in cnn.py script. In here, I use 4 convolution layers (kernel size: 3*3) and 2 max pooling layers (filter size: 2*2). I am using the following parameters for this task:

learning rate = 0.0005 (for Adam's optimizer)
epochs = 50
batch size = 256
no l2 regularization used here




Best Epoch: 49

Corresponding Validation Accuracy: 0.9124

CNN Test Accuracy: 0.9069

Analysis: As we can see from the above figures, CNN model better learns the structures in the images and thus trains better compared to other 2 approaches. This is because CNN takes the image as input and also contains non-linear activation layers which can understand the complex images (shapes even with texts).

Overall, CNN gave us the highest accuracy of 91% compared to other 2 methods.

# References / Acknowledgements

dataset: https://huggingface.co/datasets/0-ma/geometric-shapes/viewer

https://keras.io/examples/vision/mlp_image_classification/

https://keras.io/examples/vision/image_classification_from_scratch/

https://www.baeldung.com/cs/ml-relu-dropout-layers

https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-normalize-or-standardize-a-dataset-in-python.md

https://www.geeksforgeeks.org/categorical-cross-entropy-in-multi-class-classification/

I also referred to the code from coding assignment 2 and slides from class. TA Nicholas Rebstock helped me to pick MLP classifier as one of the algorithms.