1. Problem definition

1.1 Task definition

Our goal is to classify hand-written digits, using moments (centroid) of the image. We will use MNIST Dataset for this project. MNIST dataset contains 60,000 training examples. Each example is 28x28 pixel image, with each pixel having a value in range from 0 to 255. Our goal is to classify this image as which digit it represents (from 0 to 9).

Input: 28x28 pixel image

Output: digit label

1.2 Algorithm definition

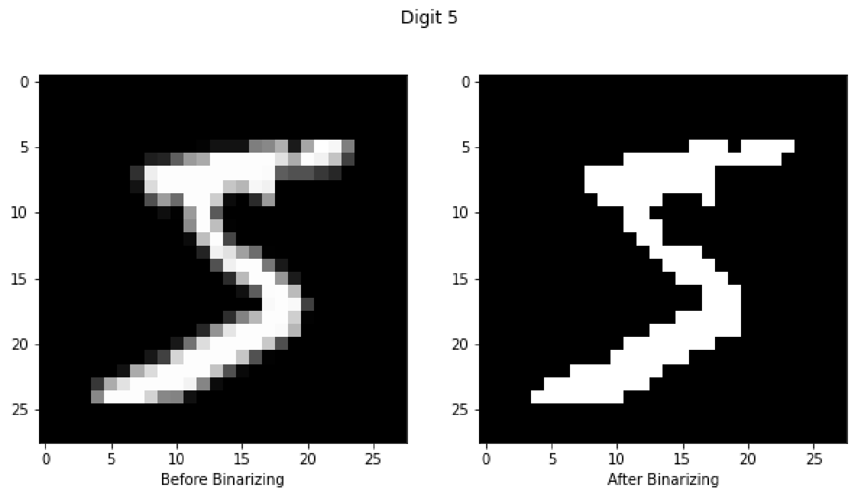
We will be using a simple KNN model to classify the digits, but our main goal is to extract useful features from the image, using the moments of the image.

2. Feature extraction

2.1 Image Preprocessing

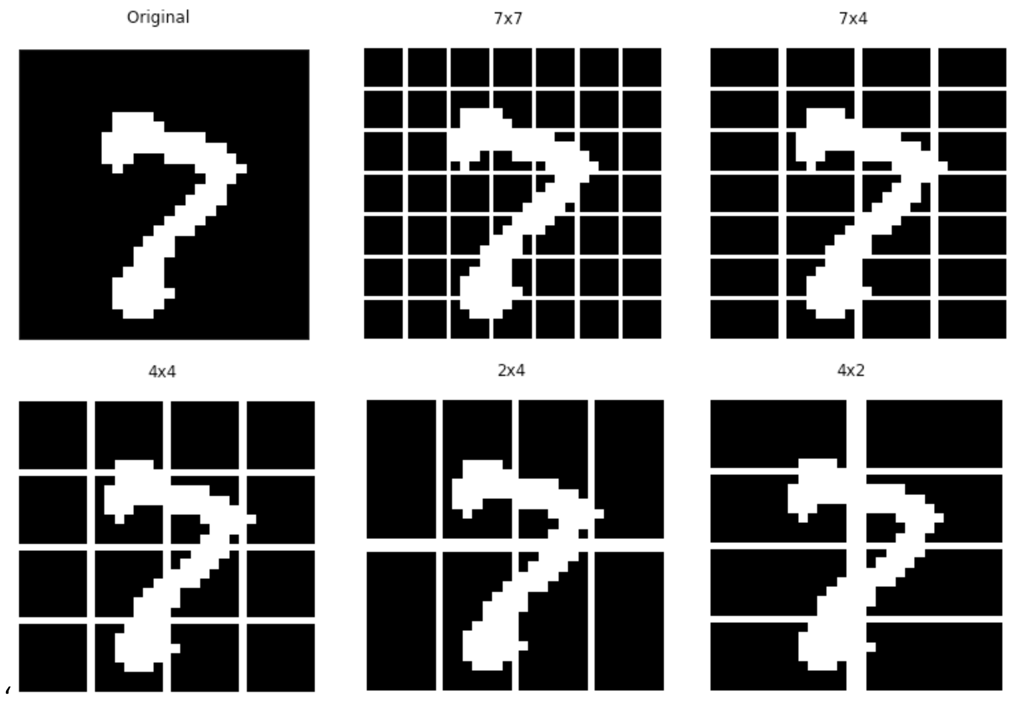
Each pixel in the image has a value in range from 0 to 255, We will binarize the image in order to make each pixel black or white, no in between.

For our goal, binarizing the image would not cost us a lot of lost information.



2.2 Splitting the image

Our next step is to split the image into smaller ones, we can define how we want to split he image by the size of the grid. (X×Y). Here’s an example with different grid size.

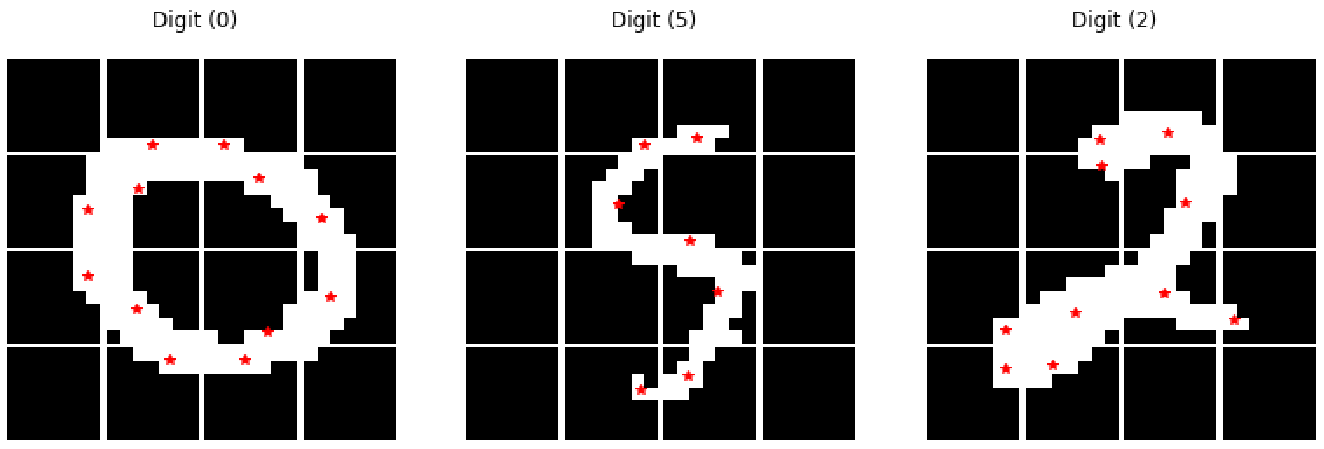


2.3 Calculating the moments (centroid)

After splitting the image into smaller parts, we will start calculating the centroid of each part, Informally, the centroid is the point at which a cutout of the shape could be perfectly balanced on the tip of a pin. Mathematically, for our purpose, we would define it as:

$$x_c = \frac{\sum_x \sum_y x f(x,y)}{\sum_x \sum_y f(x,y)} \qquad y_c = \frac{\sum_x \sum_y y f(x,y)}{\sum_x \sum_y f(x,y)}$$

Here is an example of the output (x, y) of calculating the centroid for different examples:



Note: The red star represents the centroid for each small image.

3. Building the model

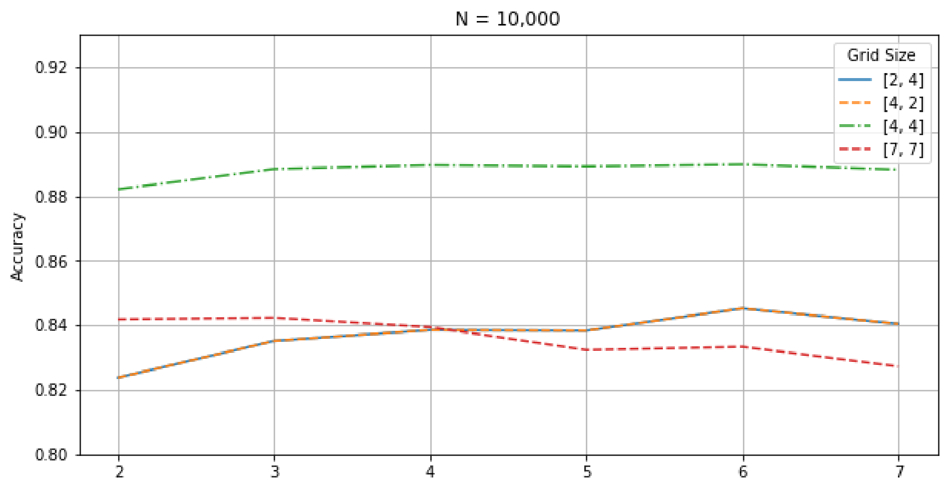
We will build our model using a KNN classifier, KNN stands for K-nearest neighbors, it basically looks for K nearest class and classifies the input based on the most nearest neighbors of the same class. We will fit our model with different sample size, different K, and also different grid size to see the impact of each of those on the model. We will be measuring the performance of our model by the accuracy (% of correctly classified images).

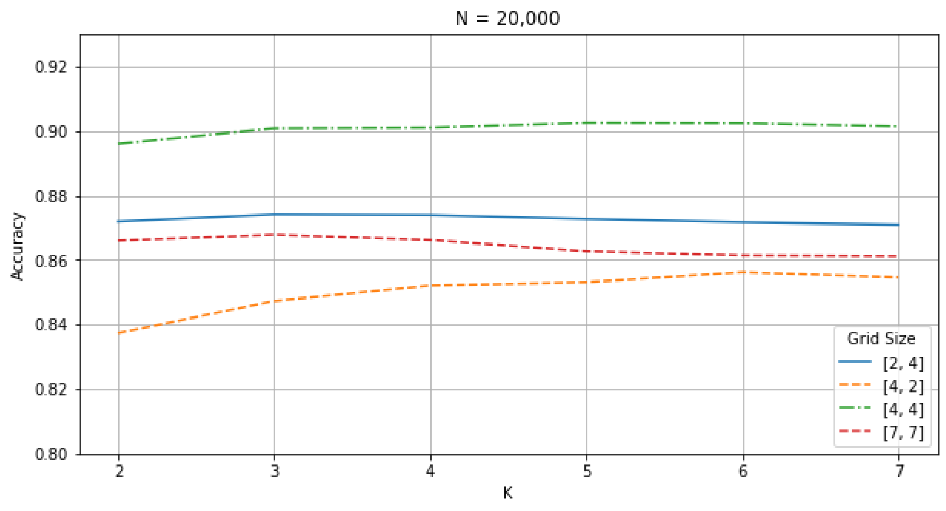
4. Model Performance Analysis

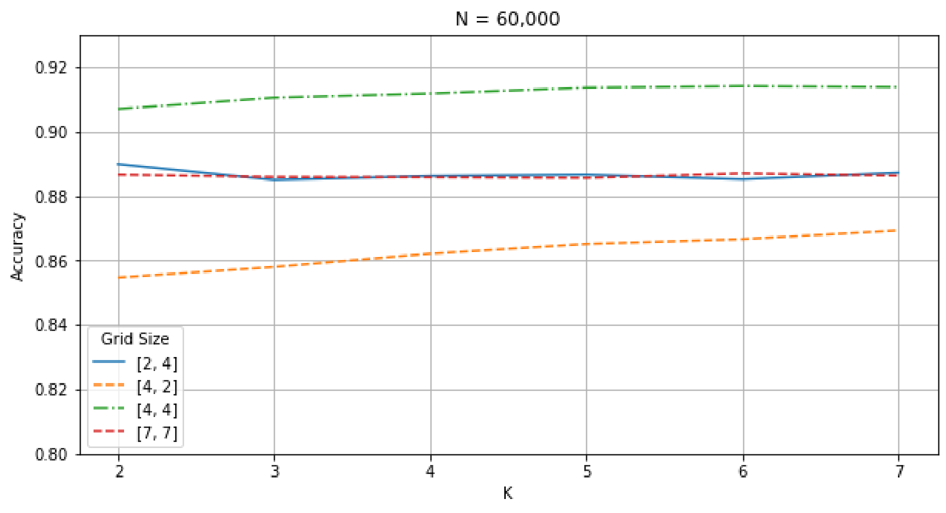
The model was tested on different K values [2 to 7], different N (number of examples) and also different grid size,

Here's a summary of the results.

N	Highest accuracy	Grid size	K
10000	88.98%	4 x 4	6
20000	90.25%	4 x 4	5
60000	91.41%	4 x 4	6







Note: for N less than 60000, results may be different depending on which data is used for training.

5. Conclusion

After multiple trials of different parameters for our pre-processing and model, we can easily notice that our model performs the best with a grid size of (4,4), no matters the sample size or K value. We were able to achieve the highest accuracy of course with training of sample size 60,000. But we were able to achieve an accuracy of 90.25% with only 20,000! (Although, I'm pretty sure this can even be pushed up further to 90.5% ~ depending on the sample data), Not a huge difference jumping from 20,000 to 60,000. Also, we notice that splitting the image into smaller parts yields back results for our model, for our data (28x28), a grid size of (4x4) seems to be the sweet spot for our purpose.