|  |  |
| --- | --- |
| **Name of Student:** | **Class:** |
| **Semester/Year:** | **Roll No:** |
| **Date of Performance:** | **Date of Submission:** |
| **Examined By:** | **Experiment No: Mini Project** |

**ASSIGNMENT – Mini Project**

**AIM: Mini Project Implementation**

**TITLE: “Feature Extraction using Zernike Moments”**

**OBJECTIVES:**

1. To understand the concept of feature extraction and its importance in machine learning.
2. To understand the concept of Zernike moments and their mathematical properties.
3. To implement feature extraction using Zernike moments in a programming language, such as Python.
4. To apply the feature extraction technique to a dataset of images and analyze the results.
5. To compare the performance of Zernike moment feature extraction with other feature extraction techniques.

**APPRATUS:**

* Python Programming Knowledge.
* NLP Algorithms.
* Computer with appropriate software for data analysis and programming (e.g., Python)
* Dataset for classification tasks (e.g., MNIST dataset)
* A software library for image processing and analysis, such as OpenCV
* Jupyter Notebook.

**ABSTRACT:**

Zernike moments possess valuable mathematical properties that make them highly suitable as image features for shape classification tasks. Their rotational invariance ensures that the descriptors remain consistent regardless of the orientation of the shape. Furthermore, with additional techniques, Zernike moments can be made scale and translational invariant, enhancing their robustness for different image scales and positions. However, the correct application of Zernike moments requires careful consideration of various factors. This paper explores several techniques to optimize the usage of Zernike moments as shape descriptors, and the experimental results have demonstrated exceptional accuracy, reaching up to 100% in certain cases. These findings highlight the effectiveness and potential of Zernike moments in shape classification problems.

**TABLE OF CONTENTS:**

|  |  |  |
| --- | --- | --- |
| Sr. No. | Content | Page No. |
| **1** | **Abstract** | **2** |
| **2** | **Introduction** | **2** |
| **3** | **Implementation** | **3** |
| **4** | **Result and Analysis** | **7** |
| **5** | **Conclusion** | **8** |
| **6** | **References** | **9** |

**INTRODUCTON:**

Moments have been used in image processing and classification type problems since Hu introduced them in his groundbreaking publication on moment invariants [4]. Hu used geometric moments and showed that they can be made to be translation and scale invariant. Since then more powerful moment techniques have been developed. A notable example is Teague’s work on Zernike Moments (ZM); he was the first to use the Zernike polynomials (ZP) as basis functions for the moments [6]. ZM’s have been used in a multitude of applications with great success and some with 99% classification accuracy [1]. The use of ZP’s as a basis function is theoretically beneficial because they are orthogonal polynomials which allows for maximum separation of data points, given that it reduces information redundancy between the moments. Their orthogonal properties make them simpler to use during the reconstruction process as well. Furthermore, the magnitude of ZM’s are rotationally invariant, which is crucial for certain image processing applications, such as classifying shapes that are not aligned. The breakdown of this paper is as follows. Section 2 introduces standard geometric moments and their application in achieving rotational and translational invariance of the images. Section 3 defines Zernike polynomials and Zernike moments. Section 4 explains the rotational invariant properties of the magnitudes of Zernike moments. Section 5 discusses how to achieve reconstruction from Zernike moments. Furthermore, it describes how to use these reconstructions to apply weights on these moments so as to make the moments which capture the most amount of information about the image contribute the most amount of ’votes’ for the classification. Section 6 gives a description of the experiments used to test the un-supervised classification accuracy of Zernike moment feature vectors. In Section 7 we will present our conclusion of our results, and our justification for why these results should be used in applicable image classification problems. Finally, In Section 8 a brief overview of future research is presented.

**IMPLEMENTATION:**

Let’s go ahead and introduce our two images. The first image, our reference image, can be seen below:



Here we can see a [Game Boy ROM cartridge](http://en.wikipedia.org/wiki/ROM_cartridge) of game, Pokemon.

Given this reference image of the Pokemon game cartridge, our goal is to detect it in the following image filled with distractor objects:

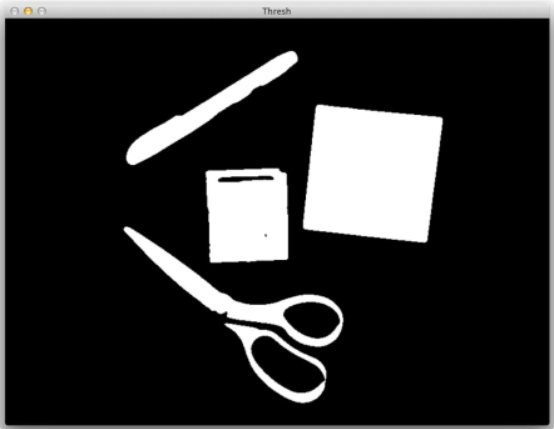


In this image we can see that we have the Pokemon game cartridge. But also also have a bunch of other distractor objects, such as scissors, a highlighter, and a sticky note.

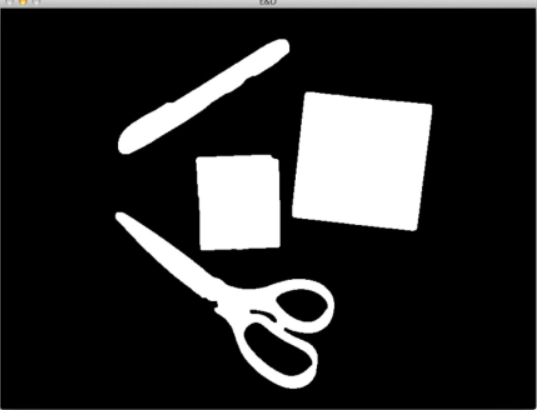
The first thing we’ll do here is define a **describe\_shapes**  function. This function will be applied to the Pokemon game cartridge in the reference image along with every object in the second image. Given that this section will be called so many times, I thought it best to create a dedicated function to describing the shape regions.

* The *describe\_shapes*function will take a single argument, an image  that contains the objects and shapes we want to quantify using Zernike Moments.

We’ll need to apply a little bit of pre-processing to our image  before we can extract our shape features, so we’ll convert the image to grayscale, blur it to remove high frequency noise and allow us to focus on the structural aspects of the image, followed by thresholding to segment the objects from the background.



We’ll also perform a series of dilations and erosions to close gaps between any parts of objects.

****

Now that the [**Morphological Operations**](https://gurus.pyimagesearch.com/lessons/morphological-operations/) have been applied, we can then detect the contours of each of the objects in the image.

We start looping over each of the individual contours. And then for each of these contours, we allocate memory for a ***mask***followed by drawing the contoured region on the mask.

Now that we have just the contoured region drawn on the mask, we can then compute the bounding box and extract the ROI (from the mask  image). Again, this step mentioned above to ensure that **only** the current shape was drawn on the mask  and **nothing** else.

Given the ROI of the shape we can now extract Zernike Moments by making a call to the **mahotas.features.zernike\_moments**  function. This function accepts 3 parameters: the image/ROI that we want to quantify, the radius of the region, and the degree of the polynomial.

\*\* **Tuning the radius**can be non-trivial based on the sizes of the objects in the image. If the sizes of the objects vary dramatically, as they do in our distractor image above, the value of this radius is often not immediately obvious.

To solve this problem, we have 2 solutions :

* The first solution is to simply **resize the ROI** to a known size. This ensures that we will be able to hardcode the radius value and that our images are described in a consistent manner.
  + The **problem**with this approach is that resizing to a fixed size **can destroy the aspect ratio** of the shape. This is especially troublesome if we are trying to quantify the shape of a region — by throwing away the aspect ratio, we throw away information regarding the dimensions of the shape, which is completely counterintuitive to our goal!

The other solution is what this example utilizes — we can dynamically compute the radius of the Zernike Moments simply by computing the **cv2.minEnclosingCircle**  function (a [**simple contour property**](https://gurus.pyimagesearch.com/topic/simple-contour-properties/)). This method will return the minimum size radius that can be used to enclose the entire object. By applying this method, we can ensure the radius of the moments covers the entire ROI.

* We also supply degree=8  to the zernike\_moments  function, which is the default degree of the polynomial. In most cases you’ll need to tune this value until it obtains adequate results.

Finally, we can return a **2-tuple** from our describe\_shapes  function consisting of (1) the contours of the objects/shapes in the image, followed by (2) the Zernike Moments feature vectors corresponding to each shape.

So now that we have the describe\_shapes  function defined, let’s see how we can use it to recognize the Pokemon game cartridge in an image:

We load our reference image of the Pokemon game cartridge from disk and describe shape of the game cartridge.

We load the distractor image from disk and quantify all of the shapes in the image using Zernike Moments.

To detect the actual game cartridge, we use SciPy’s [distance](http://docs.scipy.org/doc/scipy/reference/spatial.distance.html) sub-module to compute the Euclidean distance between all pairs of the gameFeatures  and shapeFeatures.

* Notice how we have only one row, which is the Zernike Moments associated with the game cartridge features. We also have multiple columns — one for each of the shapes in the shapeFeatures  list.

The index of the column that minimizes this distance is thus our Pokemon game cartridge. How can we be sure??

Zernike Moments are used to quantify the shape of an object. If we assume shapes that have similar feature vectors also have similar visual contents, then the shape that minimizes the distance between Zernike Moments must be our reference image!

We’ll start by looping over the contours in our distractor image. If the index of the current contour does not match the index of the contour with minimum distance in our distance matrix D, then we draw a rotated bounding box surrounding it in ***red***.

We then  then draw a **green** rotated bounding box surrounding our identified game cartridge region, followed by displaying the text “FOUND!” directly above the bounding box.

**Result and Analysis:**

****

On the left we have our original reference image of the Pokemon game cartridge. And on the right we have our distractor image containing not only the Pokemon game cartridge, but other objects and shapes meant to “confuse” our algorithm.

However, by applying Zernike Moments features we were able to discard the distractor shapes and detect the actual game cartridge shape in the image.

**CONCLUSION:**

In conclusion, when describing multiple shapes in an image using Zernike Moments, it is important to follow a few key steps. First, extract the region of interest (ROI) for each object in the image. Then, apply Zernike Moments to each ROI to capture the shape characteristics.

When utilizing Zernike Moments, it is crucial to consider the radius and degree parameters. The radius determines the size of the disc onto which the shape is mapped. It is essential to choose a radius that is sufficiently large to include all relevant pixels of the shape. If any pixels fall outside the radius, they will be disregarded. Therefore, careful attention must be paid to ensure an appropriate radius is selected before extracting Zernike Moments.

Additionally, the degree parameter directly affects the dimensionality of the resulting feature vector. Higher degree values lead to larger and potentially more distinctive feature vectors. However, it is important to consider the computational cost as the number of degrees increases.

When setting the radius and degree parameters, it is recommended to prioritize the radius selection. Reflecting on the size of the objects in the dataset will help determine an adequate radius that captures their shapes effectively. Once the radius is established, the degree parameter can be fine-tuned accordingly. Starting with a value of degree=8 is often a reasonable initial choice, with adjustments made as needed based on the desired dimensionality of the feature vector.

By following these guidelines, one can effectively describe shapes in an image using Zernike Moments, taking into account the appropriate radius and degree parameters to capture the desired shape characteristics.

**REFERENCES:**

[1] Khotanzad A. and Hong Y. H. Invariant image recognition by zernike moments. IEEE, 12(5):489 – 497, 1990.

[2] Suk T. Flusser J. and Zitova B. Moments and Moment Invariants in Pattern Recognition. Wiley and Sons Ltd., 2009.

[3] Fan X.X. Fu B., Liu J. and Quan Y. A hybrid algorithm of fast and accurate computing zernike moments. IEEE, 2007.

[4] Hu M. K. Visual pattern recognition by moment invariants. IRE Transactions on Information Theory, 8(2):179 – 187, 1962.

[5] Yu M. Feature-Weighted Hierarchical Sparse Learning for Semantic Concept Identification.

[6] Teague M. R. Image analysis via the general theory of moments. Optical Society of America, 70(8):920 – 930, 1979.

[7] Prokip R.J. and Reeves A.P. A survey of moment-based techniques for unoccluded object representation and recognition. CVGIP: Graphical Models and Image Processing, 54(5):438 – 460, 1991.

[8] Witlin R.S. The witlin center for advanced eyecare, 2008.