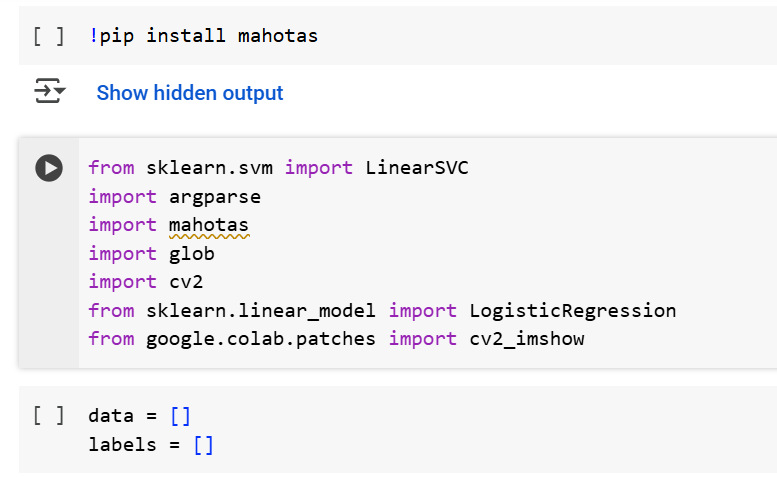
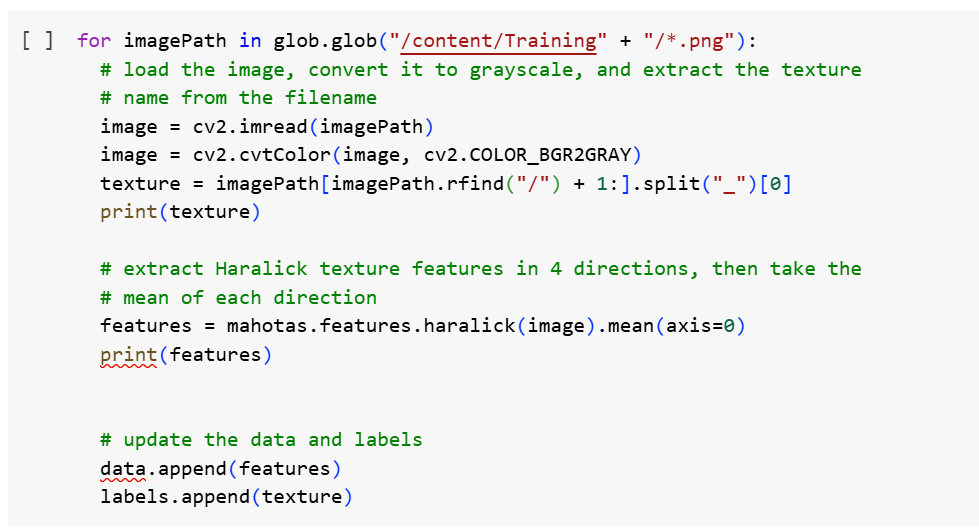
**Image Processing basis Texture**

**Code:**

****

****

****

**Details:**

**Glob Library**

Glob library in python is used to pick files with pathnames that match a specified pattern,

**It's a part of Python's standard library, so no additional installation is needed.**

It is used for pattern matching with wildcard characters to match filenames and paths.

* + \*: Matches zero or more characters.
  + ?: Matches any single character.
  + []: Matches any single character within the brackets.
  + [!...]: Matches any single character not within the brackets.



**CV2 Library**

CV2 library in python is used for image processing. It is used to load the image and then perform action on the image data.

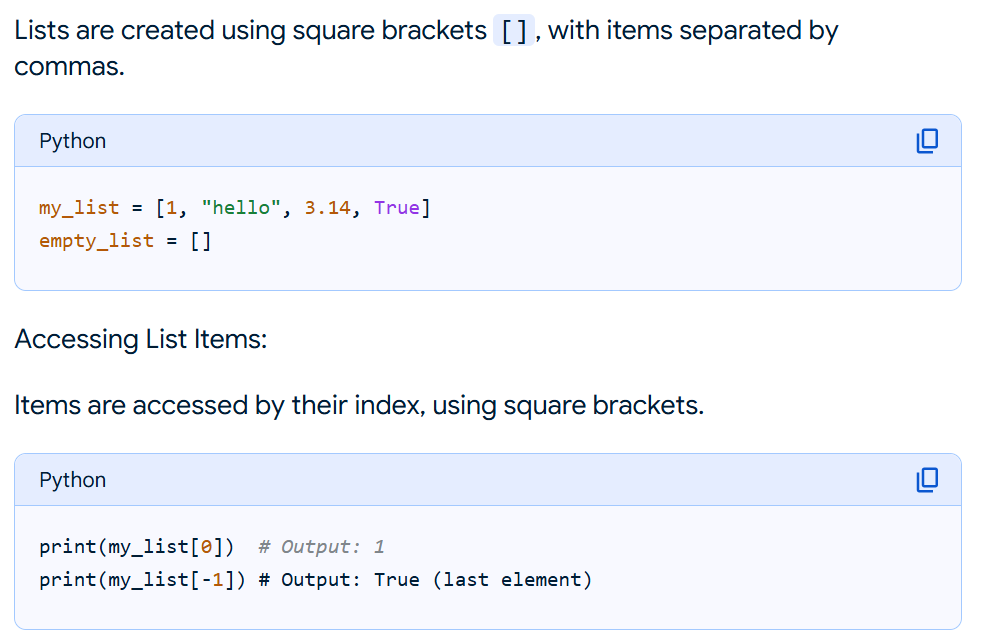


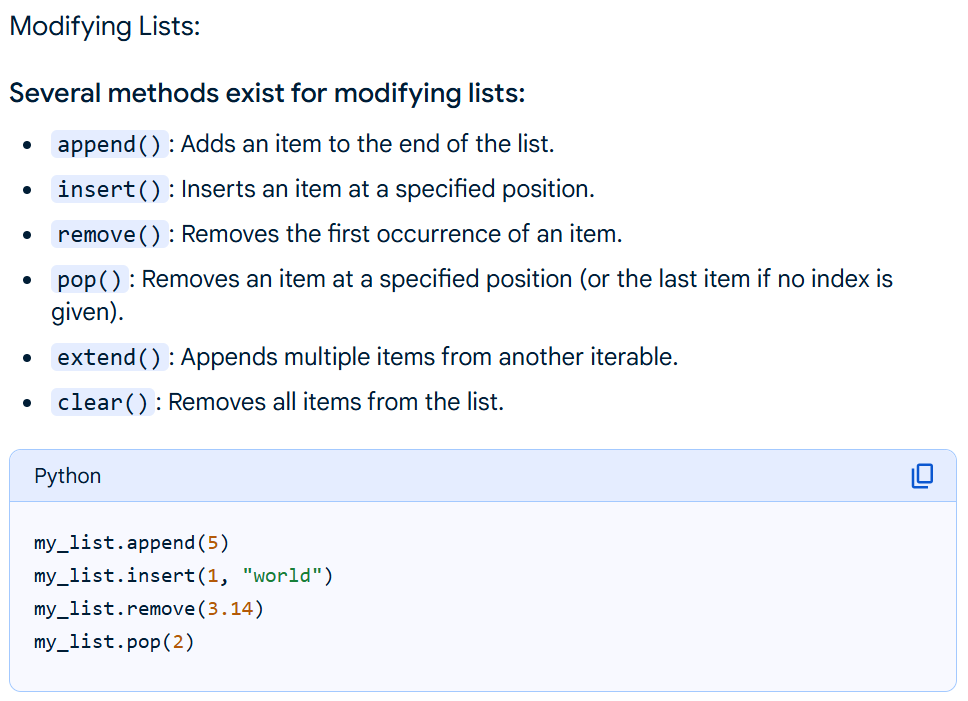
**Data Structures**

Data Structures in the world of Python is managed in 3 forms of collections or arrays of Data.

1. List – [ ] (Square brackets)
2. Tumple
3. Dictionary

**List**

****

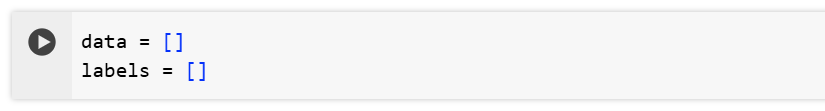
****

**HARALICK TEXTURE – Training**

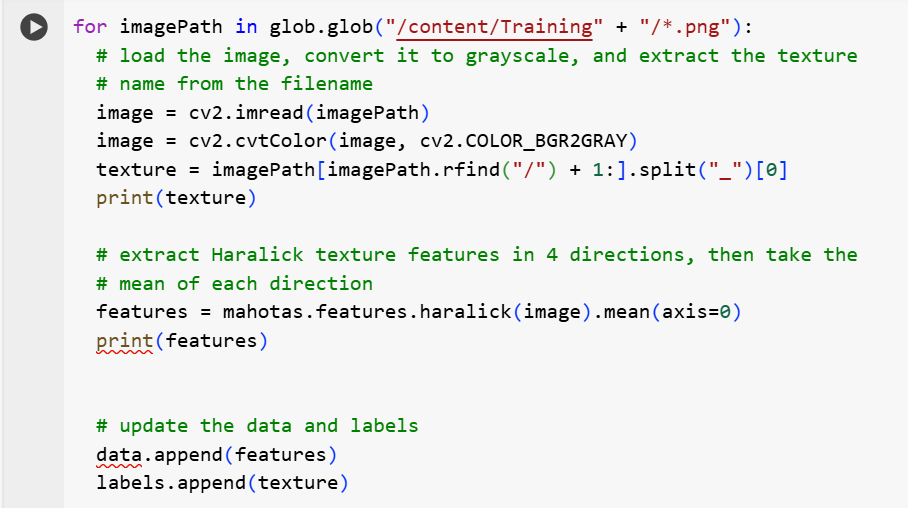
****

Step 1

Two List data structures are created:



Step 2



Each ***imagepath*** in the folder Training is pick one by one using Glob.

Image is picked from path and added to the variable ***image***

Same image is then coverted to GREY-SCALE and restored into the same variable ***image***

The portion of the imagename of the image is filtered from complete pathname sorted in variable ***imagepath*** and stored in variable ***texture*** as ID for identification and later appended into the **List variable *Labels[ ]***

**Now extraction of the Feature**

**# extract Haralick texture features in 4 directions,**

**then take the mean of each direction**

**features = mahotas.features.haralick(image).mean(axis=0)**

**Understanding Haralick**

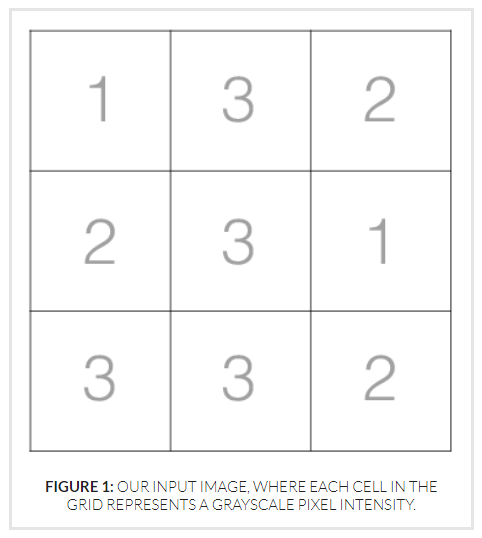
**What are Haralick texture features?**

Haralick features are used to describe the texture of an image. Texture refers to the appearance, consistency, or “feeling” of a surface. Examples of textures include “rough” vs. “soft.” Potential applications of Haralick features include determining if a road is paved vs. gravel.

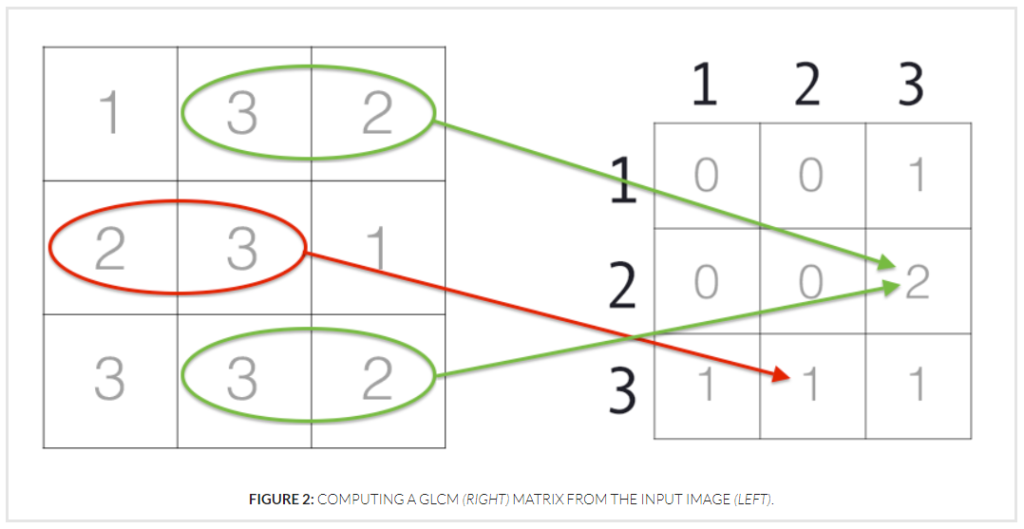
**How do Haralick texture features work?**

Haralick texture features are computed using the Gray-Level Co-occurrence Matrix (GLCM). This matrix characterizes texture by *recording how often pairs of adjacent pixels with specific values* occur in an image.

To understand how the GLCM works, take a look at the following figure, where the values in each block represent the pixel intensity of a grayscale (single channel) image:



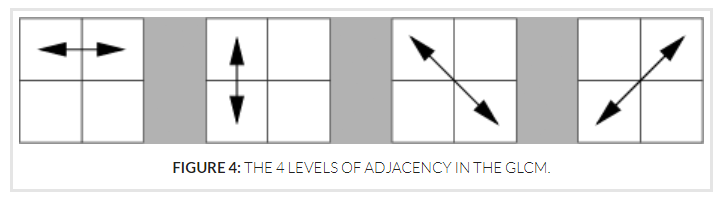
Then, to construct our GLCM, we *look at pairs of adjacent pixels* and record the number of times these two values appear next to each other:



Here we can see the pixel intensities *2 and 3* appear next to each other *once*, so they have an entry of *1* in the GLCM. The pixel intensities *3 and 2*; however, appear next to each other *twice*, so they have a corresponding entry of *2*in the GLCM.

However, we are not just limited to recording the number of times a pixel value appears to the *left or right*of a given pixel.

We *can actually specify four different****directions of adjacency***: *left to right*, *top to bottom*, *top-left to bottom-right*, and *top-right to bottom-left*:



This leaves us with a total of 4 GLCM matrices that we can use to compute Haralick features.

Now that we have these 4 GLCMs, we can *compute our Haralick features for each of the GLCMs*. Again, these values are simply statistics computed from the GLCM used to characterize and represent contrast, correlation, dissimilarity, entropy, homogeneity, and other desirable statistical properties.

After computing these statistics for each of these GLCMs we are left with 4 feature vectors (one row per direction), each of 13-dimensionality (or 14-dimensionality, depending on whether or not you computed the final 14th statistic). [ 4 x 13 matrix ]

Finally, we take the ***average of these directions*** to form a final feature vector of 13-dimensionality. This averaging is performed in an attempt to make the feature vector more robust in changes in rotation. We’ll demonstrate this averaging in the next section of this article. [ 1 x 13 matrix ]

**Where are Haralick texture features implemented?**

Haralick texture features are implemented in the ***mahotas***Python package. You can also compute the GLCM explicitly using the scikit-image library.

|  |  |
| --- | --- |
| 1  2  3 | import mahotas  gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  features = mahotas.features.haralick(gray).mean(axis=0) |

# Logistic Regression Basics (Machine Learning)

Logistic Regression is a supervised machine learning algorithm used for classification problems.

Unlike linear regression which predicts continuous values **Logistic Regression** predicts the probability that an input belongs to a specific class.

It is used for binary classification where the output can be one of two possible categories such as Yes/No, True/False or 0/1. It uses sigmoid function to convert inputs into a probability value between 0 and 1.

**Types of Logistic Regression**

Logistic regression can be classified into three main types based on the nature of the dependent variable:

1. **Binomial Logistic Regression**:

This type is used when the dependent variable has only two possible categories.

Examples include Yes/No, Pass/Fail or 0/1. It is the most common form of logistic regression and is used for binary classification problems.

1. **Multinomial Logistic Regression**:

This is used when the dependent variable has three or more possible categories **that are not ordered**.

For example, classifying animals into categories like "cat," "dog" or "sheep."

1. **Ordinal Logistic Regression**:

This type applies when the dependent variable has three or more categories **with a natural order or ranking**.

Examples include ratings like "low," "medium" and "high." It takes the order of the categories into account when modeling.

## Assumptions of Logistic Regression

Understanding the assumptions behind logistic regression is important to ensure the model is applied correctly, main assumptions are:

1. **Independent observations**: Each data point is assumed to be independent of the others means there should be no correlation or dependence between the input samples.
2. **Binary dependent variables**: It takes the assumption that the dependent variable must be binary, means it can take only two values.
3. **Linearity relationship between independent variables and log odds**: The model assumes a linear relationship between the independent variables and the log odds of the dependent variable which means the predictors affect the log odds in a linear way.
4. **No outliers**: The dataset should not contain extreme outliers as they can distort the estimation of the logistic regression coefficients.
5. **Large sample size**: It requires a sufficiently large sample size to produce reliable and stable results.



## **In the **first step** we are training the program by adding the created list to our model of LogisticRegression (with data for type of texture from images and the name of each type of Texture)**

## **Data contains the numeric number values for Textures and Labels contains the associated names.**

## **In the **second step:****

## **We are looping through the supplied images for predictions.**

## **Each image is picked from the supplied path.**

## **Converted to gray scale**

## **Feature value is picked against the image**

## **LogisticRegession is requested to predict the type of Texture**

## **The supplied texture is then written on the image with putText function**

## ****Understanding our code in details:****

features = mahotas.features.haralick(gray).mean(axis=0)

## ****pred = model.predict(features.reshape(1, -1))[0]****

These two lines of code performs a prediction using a machine learning model. Let's break it down:

* **model.predict(features.reshape(1, -1))**:

This part calls the predict method of a trained machine learning model (represented by the variable model). This method is designed to take input data and generate predictions based on what the model has learned.

* features.reshape(1, -1):

The input data, stored in a variable named features in the previous line of code, is being reshaped before being passed to the predict method. Let's look closer:

* + - reshape() is a method used to change the dimensions of an array (in this case, likely a NumPy array).
    - (1, -1): This tuple specifies the new shape. The 1 indicates that the first dimension of the reshaped array should have a size of 1 (i.e., a single row). The -1 is a placeholder that tells NumPy to automatically calculate the size of the second dimension based on the original size of the features array. This effectively transforms the input into a 2D array with one row. This is a common requirement for many machine learning models, which expect inputs to be in a specific format.
* [0]:

Finally, the [0] at the end of the line extracts the first element from the result of model.predict(...). This suggests that the model is expected to return a prediction as a list or array, and you are interested in the first (and likely only) prediction.

In summary:

This line takes a set of features, reshapes it into a format suitable for the model, uses the model to make a prediction, and then extracts the single predicted value. The result is stored in the variable pred.

## ****With this, all the images are categorised one by one.****

## **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***