Noise-Resistant LBP Descriptor for Texture Recognition - A Supervised Approach

Rishideep Chatterjee
Dept. of Comp. Sc. and Eng.
RCC Institute of Information Technology
Kolkata, India
rishi1a2@gmail.com

***Abstract*—Texture recognition is an essential task in many fields, including computer vision, medical imaging, and material science. The Local Binary Pattern (LBP) descriptor has proved to be an effective tool for texture recognition. However, it has been conclusively proved that the LBP descriptor is sensitive to image noise. Noise in an image can degrade the visual quality and result in reduced recognition accuracy, leading to faults in our measurements, calculations, and analysis. This paper proposes a noise-resistant LBP descriptor for texture recognition through a supervised learning approach. The methodology consists of four steps: (1) Preprocessing the image, (2) Extracting LBP features, (3) Applying a noise-resistant thresholding technique, and (4) Classifying the texture pattern using a machine learning algorithm, in this case, a Linear Support Vector Machine (SVM).**

***Index Terms*—local binary pattern, texture recognition, machine learning, image processing, thresholding**

I. Introduction

An image texture is a set of metrics calculated during image processing, designed to compute the perceived texture of an image. It gives us information about the spatial arrangement of color or intensities in an image or a region of an image. This is primarily done to substantiate our perception with quantifiable measures. By definition, Texture recognition is the task of identifying such an image texture. This task is critical in many domains, such as computer vision, medical imaging, material sciences, and often in space sciences. Many texture recognition methods have been proposed in the literature, including statistical methods, filter-based methods, and transform-based methods. In recent years, local feature-based methods have become popular due to their effectiveness, computational efficiency, and increased level of accuracy.

The local binary pattern (LBP) descriptor is a widely used 'local feature-based' method for texture recognition. LBP was first introduced by Ojala et al. in 1996 [1]. The LBP descriptor encodes the local texture information by comparing the intensities of a pixel and its neighbors in a circular neighborhood. The LBP descriptor has many advantages, including its simplicity, accuracy, efficiency, and robustness to geometric and photometric distortions. However, the LBP descriptor is sensitive to image noise, which can result in degraded recognition accuracy.

Image noise refers to random variations in pixel brightness or color values in an image that do not represent the true features of the scene being captured. It can occur due to various factors, such as the sensitivity of the camera sensor, low light conditions, high temperature, or electromagnetic interference. Noise in an image can affect the visual quality and recognition accuracy, leading to faults in our eventual analysis.

To address this limitation, different methods have been proposed to make the LBP descriptor more robust to noise. For example, Guo et al. proposed a noise-resistant LBP descriptor by using a non-linear filter to smooth the LBP image before feature extraction [2]. However, this method has a high computational cost and may lead to counterproductive over-smoothing of the LBP image.

In this paper, a noise-resistant LBP descriptor for robust texture recognition is proposed. The goal is to minimize the computational cost and enhance efficiency. The method combines the LBP descriptor with a noise-resistant thresholding technique to improve recognition accuracy while maintaining said efficiency. The remainder of this paper is organized as follows. Section II provides a brief literature review in the context of existing local-binary pattern descriptors. Section III describes the proposed method in detail, while Section IV presents the experimental results and analysis. Finally, Section V concludes the paper.

II. Literature Review

Texture recognition is a fundamental task in computer vision with applications ranging from image classification to object detection and segmentation. The Local Binary Pattern (LBP) descriptor has been a widely adopted method for texture analysis due to its simplicity, computational efficiency, and effectiveness. The topic, therefore, has been extensively studied and there is a wealth of literature available on it.

The paper by Guoying Zhao and Matti Pietikainen [3] addresses dynamic texture (DT) recognition, an extension of texture analysis into the temporal domain. The authors propose a novel approach using Volume Local Binary Patterns (VLBP) to model DTs, which combine motion and appearance.

The use of grayscaling is also quite common. T. Ojala, M. Pietikainen, and T. Maenpaa in their paper [4], talk about a multiresolution approach to gray-scale and rotation invariant texture classification based on local binary patterns. They identify "uniform" LBP patterns as crucial texture features and

derive a generalized operator for rotation invariance. The work of Zhenhua Guo, Lei Zhang, and David Zhang [5] employs a local region's center pixel and a Local Difference Sign-Magnitude Transform (LDSMT) to represent texture, through global thresholding.

In a nonparametric approach, M. Topi, O. Timo, P. Matti, and S. Maricor [6] show that a carefully chosen subset of patterns derived from LBP offers an efficient texture description, outperforming the entire LBP histogram in classification accuracy. The work by M. Topi, P. Matti, and O. Timo [7] proposes a multi-predicate version of the LBP, making it useful for textures with multiple scales.

The usage of the Enhanced Fisher Model (EFM) is highlighted in a paper by Sugata Banerji, Abhishek Verma, and Chengjun Liu [8]. Four novel color Local Binary Pattern (LBP) descriptors are proposed in the work and the classification relies heavily on the nearest neighbor classification rule (EFM-NN). A study on a fusion approach based on uniform local quinary pattern (LQP) and a rotation invariant local quinary pattern was conducted by L Nanni, A Lumini and S Brahnam [9]. In another paper, N Kazak, M Koc [10] introduce the use of symmetric two spirals LBP to measure grayscale differences between the center pixel and its neighbors.

However, it is well-known that LBP is sensitive to noise and variations in lighting conditions, which can hinder its performance in real-world applications. The use of a noise-resistant Local Binary Pattern descriptor in this paper, aims to fill this research gap.

III. Methodology

The proposed noise-resistant LBP descriptor consists of four steps: (1) preprocessing the image, (2) extracting LBP features, (3) applying a noise-resistant thresholding technique, and (4) classifying the texture pattern using a machine learning algorithm.

The initial stage involves a critical step known as preprocessing, aimed at improving the overall quality of the image while removing any unwanted aberrations that may be present. These aberrations often lead to undesirable variations or irregularities in pixel values that can obscure the underlying information in the image. To tackle this issue, various filtering techniques are employed, with two common choices being the Gaussian filter and the Median filter. In the specific context of this scenario, a systematic approach is adopted in order to enhance the image.

Initially, the image is converted into grayscale. This conversion simplifies the image by representing it solely in terms of grayscale values, eliminating color information. Grayscale images are often easier to process, and they retain vital information regarding brightness and contrast, which is essential for many image analysis tasks, in this case, extracting the lbp features.

After the grayscale conversion, the next step is to employ a median filter as a filtering tool. The median filter is a powerful non-linear filter that plays a pivotal role in noise reduction, particularly for mitigating a type of noise known as "salt-and-pepper noise." This type of noise appears as sporadic, isolated pixels with extreme brightness values, similar to grains of salt and pepper scattered across the image. The median filter operates by replacing each pixel in the image with the median value of its neighboring pixels, within a specified window or kernel size.

In place to the median filter, another widely used tool for image preprocessing and noise reduction is the Gaussian filter. Like the median filter, the Gaussian filter is a fundamental component of image processing that contributes to enhancing image quality and making subsequent feature extraction more reliable.

The Gaussian filter, as we know, derives its name from the Gaussian distribution. Thus, this filter works by applying a weighted average to each pixel in the image, taking into account the values of its neighboring pixels within a specified kernel or window. However, unlike the median filter, which uses the median value, the Gaussian filter uses a weighted average that places more emphasis on nearby pixels and less on distant ones.

The effectiveness of median filter over the Gaussian filter lies in its ability to effectively suppress salt-and-pepper noise while preserving the essential structural details of the image. By selecting the median value within the local neighborhood, it ensures that extreme outliers, that is, the noisy pixels are replaced with values that are more representative of the underlying image content. This filtering process helps to smoothen the image, making it more visually pleasing and suitable for subsequent image analysis or recognition tasks. In texture recognition, preserving texture details is essential for achieving high discriminative power. The median filter's noise reduction capabilities (without significant texture loss) contribute to better feature vectors for distinguishing between different texture patterns.

Although the choice between Median and Gaussian filtering for LBP feature extraction isn't universal, when the primary concern is noise reduction while preserving texture information, the Median filter is often the preferred choice for LBP-based texture analysis.

Algorithm 1: Converting digital image to grayscale

Input : resized image-list
Output: grayscale images

```
for img ∈ images do
    gray ←
      cv2.cvtColor(img, cv2.COLOR_BGR2GRAY);
      gray_images.append(gray);
end
```

Algorithm 1 takes a list of color images as input and aims to convert them into grayscale images, which contain only intensity or brightness information, devoid of color data. The algorithm employs the OpenCV library's

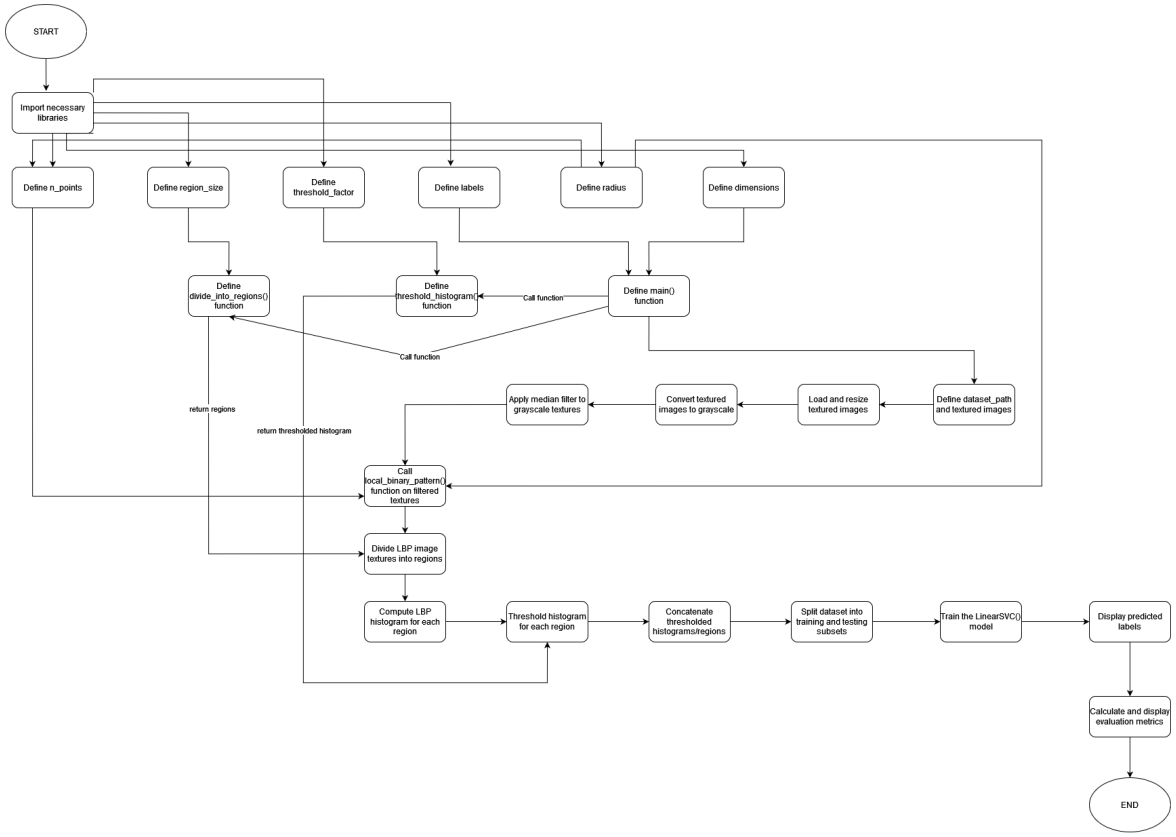


Fig. 1: Flow Diagram of Methodology

Algorithm 2: Using denoising filter - Median Blur

Input : grayscale image-list
Output: median blurred images

1 **for** $filtered \in filtered\_images$ **do**
2 $\quad filtered \leftarrow cv2.medianBlur(gray, 5)$;
3 $\quad filtered\_images.append(filtered)$;
4 **end**

Algorithm 3: calculate lbp features

Input : filtered image-list
Output: lbp feature map

1 **for** *filtered* $\in filtered\_images$ **do**
2 $\quad lbp \leftarrow$ $local\_binary\_pattern(filtered, n\_points, radius)$;
3 $\quad lbp\_images.append(lbp)$;
4 **end**
5

`cv2.cvtColor` function to perform this conversion for each image in the input list. It iterates through the images, converts them to grayscale using the specified color conversion code (`cv2.COLOR_BGR2GRAY`), and accumulates the resulting grayscale images in a separate list. One can also use the `cv2.IMREAD_GRAYSCALE` method to convert the color images to corresponding grayscale.

Algorithm 2 takes a list of grayscale images and applies a median blur filter to reduce aberrations. It iterates through each image, applies the filter, and stores the smooth versions in a new list. Inside the loop, Line 2 applies the median blur filter to the grayscale image denoted as `gray`. The `cv2.medianBlur` function is called, taking two arguments: the input grayscale image (`gray`) and the kernel size (`5`). The kernel size determines the size of the neighborhood used for median filtering, affecting the extent of noise reduction.

The second step is to extract the LBP features from the preprocessed image. This is typically achieved by

a. Comparing the intensity of each pixel with its neighboring pixels, and

b. Encoding the result as a 'binary pattern'

This process is repeated for each pixel in the image, and the resulting patterns are used for the classification task, using an SVM classifier. First, the LBP features are computed for each preprocessed image to capture texture information. The scikit-image library is used to achieve this. The method 'local_binary_pattern' is defined as follows: -
@params:

`filtered`: The filtered grayscale image as input.

`n_points`: The number of sampling points to be used for LBP calculation. It is set to `8 * radius` as a common

choice.

`radius`: The radius parameter defines the distance from the central pixel to the surrounding sampling points.

`method='uniform'`: This specifies the LBP method to use, with the 'uniform' method being a common choice that simplifies LBP patterns.

The LBP function returns a 2D array, known as the LBP feature map, where each pixel in the map represents the LBP code of the corresponding pixel in the input image. The LBP feature map is added to the `lbp_images` list, which stores these feature maps for all filtered images. Algorithm 3 illustrates the function discussed above.

We now move onto the next step in this research, that is, the region-based thresholding. Dividing each LBP feature map or, each LBP image into non-overlapping regions using the `divide_into_regions` function defined in Algorithm 4, each region is processed to compute the LBP histogram. The LBP histogram counts the occurrence of each LBP code in the region. The LBP code is a binary code that encodes the local texture information of a pixel and its neighbors in a circular neighborhood. It is computed by comparing the intensity value of each neighbor with the intensity value of the center pixel. If the neighbor's intensity is greater than or equal to the center pixel's intensity, the corresponding bit in the LBP code is set to 1; else, it is set to 0.

To achieve this, the code calls the `divide_into_regions` function described in Algorithm 4, for each LBP feature map. The `divide_into_regions` function:

- Accepts the LBP feature map (`lbp`) and a specified region size (`region_size`) as input.
- Divides the LBP feature map into smaller regions of the specified size without overlap.
- Returns a list of these regions.

Algorithm 4: Method *divide_into_regions*

Input : lbp feature map
Output: list of non-overlapping regions

$height, width \leftarrow$ _lbp.shape;
for $i \in range(0, height, \_region\_size$ **do**
  for $j \in range(0, width, \_region\_size$ **do**
    region $\leftarrow \_lbp[i : i + \_region\_size, j : j + \_region\_size]regions.append(region)$**end**
  end
  return $regions$

For each region within the list returned by `divide_into_regions`, the code calculates a histogram of LBP values. These histograms are represented as arrays. The histograms are then thresholded using the `threshold_histogram` function illustrated in Algorithm 5. This function normalizes the histogram to have a unit L1 norm and applies a threshold to make it binary. As discussed, the threshold is determined by multiplying the specified `threshold_factor` by the mean of the normalized histogram. Values below this threshold are set to 0, and values equal to or above the threshold are set to 1.

Algorithm 5: thresholding histograms

Input : list of non-overlapping histograms
Output: thresholded histograms

$threshold \leftarrow$ _threshold_factor * np.mean(_histogram) $\_histogram[\_histogram < threshold] \leftarrow 0$;
$\_histogram[\_histogram >= threshold] \leftarrow 1$;
return $\_histogram$

The primary purpose of normalizing a histogram is to ensure that its values represent relative frequencies or probabilities, making it more interpretable and invariant to the scale of the data. In this specific context, the function is used to normalize the LBP histogram for a region of an image. Normalization, being a common practice in image processing and feature extraction tasks, enhances the robustness and comparability of features, ensuring that the LBP histograms are transformed into normalized representations that are both scale-invariant and more suitable for subsequent texture analysis and classification.

After thresholding each region's histogram, the binary histograms are collected in a list to form a thresholded representation of the entire LBP feature map. The thresholded histograms of LBP values for each region are then concatenated to create a single feature vector representing the entire image. Algorithm 6 summarizes the above.

Algorithm 6: fetching thresholded images

Input : thresholded histograms
Output: thresholded images

for *lbp* $\in$ *lbp_images* **do**
  $regions \leftarrow$ *divide_into_regions*(*lbp*, *region_size*);
  for *region* $\in$ *regions* **do**
    $histogram, \_ \leftarrow$ *np.histogram*(*region*, *bins* = *np.arange*(0, 10), *density* = *True*);
    $histogram \leftarrow$ *threshold_histogram*(*histogram*, *threshold_factor)*
    thresholded_regions.append(*histogram*);
  end
  thresholded_image $\leftarrow$ *np.concatenate*(*thresholded_regions*);
  thresholded_images.append(*thresholded_image*);
end
thresholded_images $\leftarrow$ *np.array*(*thresholded_images*);

For each LBP feature map (`lbp`), the code divides it into non-overlapping regions using the `divide_into_regions` function. The regions are defined by the `region_size` parameter. This step aims to segment the LBP feature map into smaller areas to capture local texture patterns. Within the loop for each LBP feature map, there is an inner loop that iterates over the regions obtained in the previous step. Each `region` variable represents a specific segment of the LBP feature map.

Inside the inner loop, a histogram of LBP values is computed for the current `region`. This histogram is created using the `np.histogram` function. The `np.histogram` function takes the `region` as input and specifies the number of bins using `bins=np.arange(0, 10)`, which creates 10 equally spaced bins. The `density=True` parameter normalizes the histogram, ensuring that it represents relative frequencies. After calculating the histogram for the current region, it is passed to the `threshold_histogram` function for thresholding. As discussed, the `threshold_histogram` function normalizes the histogram and applies a threshold based on the `threshold_factor`. Values in the histogram that are below the threshold are set to 0, while values equal to or above the threshold are set to 1, as we already discussed.

The thresholded histograms for all regions within the current LBP feature map are collected in a list called `thresholded_regions`. After processing all regions within the current LBP feature map, the code concatenates the thresholded histograms from all regions into a single 1D array called `thresholded_image`. This array represents the thresholded LBP feature map for that image. The `thresholded_image` is appended to the `thresholded_images` list, which accumulates the thresholded representations of LBP feature maps for all images in the dataset. After processing all LBP feature maps in the outer loop, the `thresholded_images` list contains the thresholded representations of LBP feature maps for all images in the dataset. Finally, the code converts `thresholded_images` into a NumPy array using `np.array(thresholded_images)` to facilitate classification.

The next set of tasks revolve around training and testing the model. The Brodatz textured dataset is used for this purpose. Labels are assigned to the images and the assigned labels are stored in an array called `labels`. Each image file has a filename that includes information about its category or class.

Once the images are labeled, it's crucial to split the dataset into training and testing sets. This separation is vital for assessing how well the trained model generalizes to unseen data and avoids overfitting, where a model memorizes the training data but performs poorly on new, unseen examples.

```
X_train, X_test, y_train, y_test =
train_test_split(thresholded_images, labels,
test\_size=0.15)
model = LinearSVC()
model.fit(X\_train, y\_train)
```

In the provided code, the dataset is split into two subsets using the `train_test_split` function from scikit-learn. By convention, one portion is designated as the training set, while the other serves as the testing set. The training set is used to train the machine learning model, allowing it to learn the relationships between the extracted texture features and their corresponding labels. The testing set is used to evaluate the model's performance on unseen data. This step simulates real-world scenarios where the model encounters new textures it hasn't seen during training.

Next step is to train the model, in this case, using an Linear Support Vector Machine. Let us see why a LinearSVC is specifically employed over any other classifying algorithm, for this texture recognition task.

LinearSVCs are well-suited for problems where the decision boundary is linear, or can be effectively approximated as linear. In texture recognition, LBP features often capture linearly separable patterns within images. Considering a classification task where the goal is to distinguish between "wood" and "brick" textures, LinearSVC can effectively learn a linear decision boundary between these two classes if they exhibit linearly separable characteristics in the LBP feature space.

Now, making predictions on the testing data with `model.predict`, the model's accuracy, precision, recall, and F1-score is calculated to evaluate performance, using scikit-learn's metrics functions. The discussion is continued in the next section.

IV. Results

So far, we have looked at the motivation, idea, and methodology behind the noise-resistant LBP-based texture recognition model, involving the following key steps:

1) Feature Extraction: Local Binary Pattern (LBP) descriptors were computed for each image to capture the texture patterns.
2) Model Training: A Linear Support Vector Machine (LinearSVC) classifier was trained on a subset of the dataset. And finally,
3) Model Evaluation: The trained model was evaluated on the remaining subset of the Brodatz dataset to measure its classification performance.

In this section, let us look at the results of the texture recognition experiments using Local Binary Pattern (LBP) descriptors and Linear Support Vector Machine (LinearSVC) classification. This study aimed to assess the effectiveness of this approach in accurately classifying different texture patterns. The Brodatz dataset used in the training process and experiments comprises a diverse collection of texture images, each belonging to a specific texture class. The textures have descriptions that include "Brodatz - Straw (D15)," "Brodatz - Plastic bubbles (D112)," etc., measure 512x512 in pixels, and 256 KB in size. Figure 2 illustrates a couple of examples of textures. The dataset was preprocessed to ensure uniform dimensions and quality across all images.

To gauge the efficacy of the texture recognition model, a range of performance metrics were employed, including accu-

racy, precision, recall, and F1-score. These metrics provide comprehensive insights into the model's ability to classify textures accurately, minimize false positives, avoid false negatives, and achieve a balanced performance.

Table I displays the results of all 10 runs of the model. It can be clearly observed that the model achieves an average accuracy of 60.034%. A reduction in test size and a consequent increase in the size of the training subset led to a substantial increment in the accuracy after the first three runs. The Precision metric recorded an average of 65.416%, and Recall score was found to be around 61.902% across the 10 runs of the model. The average F1 Score at 63.596%, indicates a balanced trade-off between Precision and Recall, often desirable in such classification and recognition tasks.

V. Conclusion

In this study, we discussed a possible, robust solution for texture recognition, a fundamental practice in computer vision with applications spanning diverse domains, from image analysis to industrial automation. This research focused on harnessing the power of Local Binary Pattern (LBP) descriptors in conjunction with Linear Support Vector Machine (LinearSVC) classification to achieve accurate and robust texture recognition. In this section, we reflect on the idea, the methodologies explored, the insights gained, and the implications of the findings.

At the outset of the research, the primary objectives were clear: to develop an effective texture recognition system and to evaluate its performance rigorously. The exploration began with the foundation of feature extraction through LBP descriptors. LBP, a widely recognized and versatile texture analysis technique, was employed to capture intricate texture patterns within images. We discussed the theory behind LBP, its robustness to various transformations, and its ability to encode spatial relationships among pixel intensities effectively. Emphasizing on the adaptability of LBP, we looked at its ability to accommodate variations in texture scale, rotation, and illumination. This adaptability is a critical asset in real-world applications where textures may exhibit substantial variations.

With feature extraction in place, we turned our attention to classification, a pivotal component of any texture recognition system. Linear Support Vector Machine (LinearSVC) emerged as the classifier of choice, elucidating the reasons behind this selection. The linear nature of LinearSVC aligns harmoniously with the discriminative power of LBP descriptors, especially in scenarios where texture patterns exhibit linearly separable characteristics. We discussed LinearSVC's computational efficiency, robustness to high-dimensional feature spaces, and regularization capabilities as attributes that significantly contribute to its suitability for the texture recognition task.

To validate the effectiveness of this approach, a series of meticulously designed experiments were conducted. A diverse dataset comprising various texture classes was obtained, each item representing distinct visual patterns. The dataset's quality was ensured through preprocessing steps that standardized image dimensions and reduced noise and aberrations. Experiments involved comprehensive data splitting into training and testing sets, a crucial step in assessing model generalization. The models were rigorously trained on a subset of the data and subsequently evaluated on an independent testing dataset.

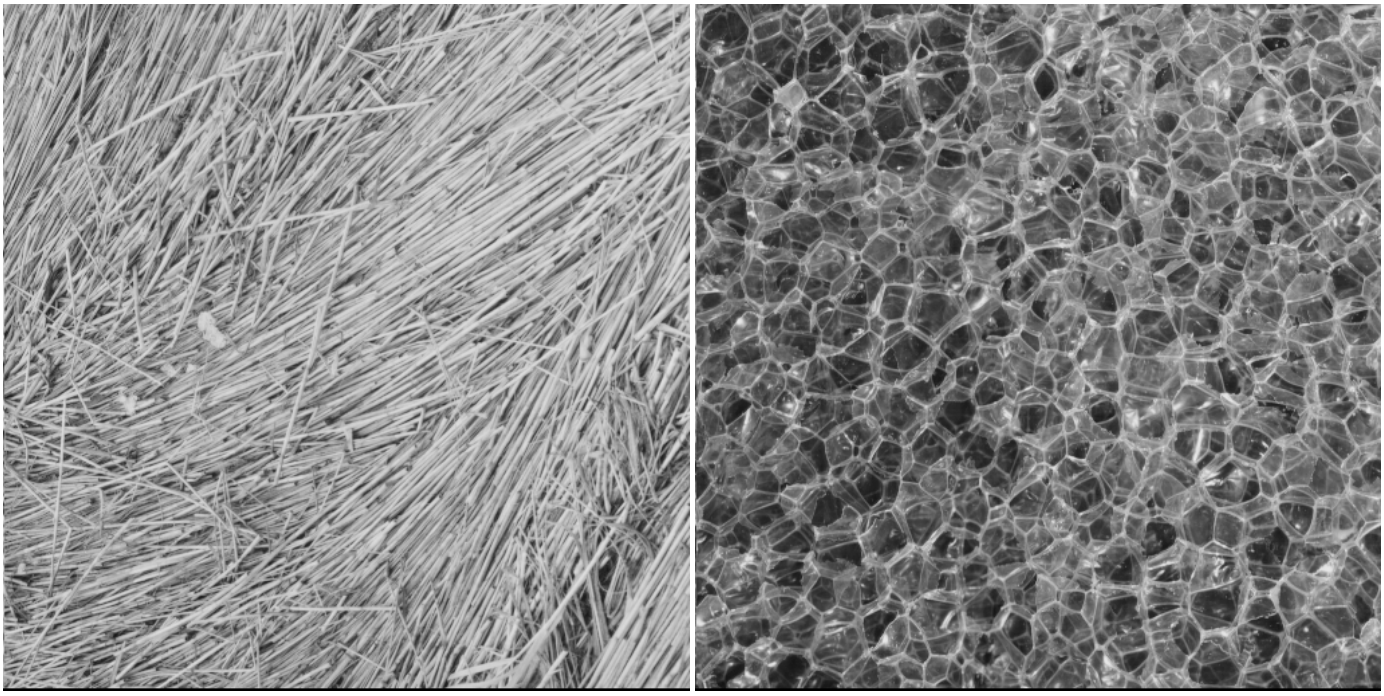
In the pursuit of a comprehensive evaluation, a suite of performance metrics were employed, each designed to shed light on different facets of the model's capabilities. Accuracy provided an overarching measure of model's correctness in predictions. Precision, recall, and the F1-score complemented accuracy, offering insights into the model's ability to make accurate positive predictions, correctly identify relevant instances, and balance precision and recall in scenarios with class imbalances. These metrics provide a holistic perspective on the model's performance.

Experimental results revealed compelling insights into the efficacy of the LBP-based texture recognition model. The table showcasing model performance metrics for multiple runs underscored the reliability of the approach. Our discussions delved into the nuances of these results, elucidating their implications in various contexts. We looked at the model's average accuracy, which demonstrated its proficiency in distinguishing between diverse texture classes. Additionally, the precision, recall, and F1-score values were gathered and a table was compiled, emphasizing their significance in applications where minimizing false positives or false negatives holds importance.

While this research has yielded promising results, the journeyis far from complete. Several avenues for future exploration beckon:

Deep Learning Integration: The integration of deep learning techniques, such as Convolutional Neural Networks (CNNs), could enhance the LBP-based texture recognition model's performance, especially when dealing with complex and diverse texture patterns.
Semantic Segmentation: Extending this approach to semantic segmentation, where texture patterns within images are delineated and classified, opens doors to applications in object detection and scene understanding.
Transfer Learning: Leveraging transfer learning by pretraining on large-scale textured image datasets can boost the Local Binary Pattern Descriptor's ability to recognize textures with limited training data.
Multimodal Data Fusion: Combining texture information with other modalities, such as color or depth data, can further enhance the discriminative power of this system.

This research underscores the significance of feature extraction techniques like LBP in encoding intricate texture information and the importance of robust classification algorithms like LinearSVC in making accurate predictions. As we look to the future, the integration of advanced techniques and the pursuit of interdisciplinary collaborations hold the promise of advancing LBP-centered texture recognition further and broadening its impact across multiple domains.



(a) Brodatz - Straw (D15)

(b) Brodatz - Plastic bubbles (D112)

Fig. 2: Different Brodatz textures used

Run	Accuracy	Precision	Recall	F1 Score
1	52.98	57.29	53.11	55.12
2	53.71	58.35	53.74	55.95
3	53.12	58.74	54.48	56.52
4	65.63	69.98	66.12	67.99
5	60.63	67.83	66.93	67.37
6	61.85	65.31	61.85	63.53
7	65.32	71.47	67.68	69.52
8	62.81	68.38	66.56	67.45
9	62.72	69.06	62.79	65.77
10	61.57	67.75	65.76	66.74

TABLE I: Model Performance Metrics for 10 Runs

REFERENCES

[1] M. Pietikäinen, "Local binary patterns," *Scholarpedia*, vol. 5, no. 3, p. 9775, 2010.

[2] T. Song, H. Li, F. Meng, Q. Wu, and J. Cai, "Letrist: Locally encoded transform feature histogram for rotation-invariant texture classification," *IEEE Transactions on circuits and systems for video technology*, vol. 28, no. 7, pp. 1565–1579, 2017.

[3] G. Zhao and M. Pietikainen, "Dynamic texture recognition using local binary patterns with an application to facial expressions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 915–928, 2007.

[4] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[5] Z. Guo, L. Zhang, and D. Zhang, "A completed modeling of local binary pattern operator for texture classification," *IEEE Transactions on Image Processing*, vol. 19, no. 6, pp. 1657–1663, 2010.

[6] M. Topi, O. Timo, P. Matti, and S. Maricor, "Robust texture classification by subsets of local binary patterns," in *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, vol. 3, 2000, pp. 935–938 vol.3.

[7] M. Topi, P. Matti, and O. Timo, "Texture classification by multi-predicate local binary pattern operators," in *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, vol. 3, 2000, pp. 939–942 vol.3.

[8] S. Banerji, A. Verma, and C. Liu, "Novel color lbp descriptors for scene and image texture classification," in *15th international conference on image processing, computer vision, and pattern recognition, Las Vegas, Nevada*, 2011, pp. 537–543.

[9] L. Nanni, A. Lumini, and S. Brahnam, "Survey on lbp based texture descriptors for image classification," *Expert Systems with Applications*, vol. 39, no. 3, pp. 3634–3641, 2012.

[10] N. Kazak and M. Koc, "Some variants of spiral lbp in texture recognition," *IET Image Processing*, vol. 12, no. 8, pp. 1388–1393, 2018.