AIP Assignment 1

Aryan Prasad

February 8, 2025

Q1 Local Binary Pattern (LBP)

Introduction

In this report, we implement a face recognition system using the Local Binary Pattern (LBP) method. LBP is a powerful texture descriptor that is widely used in facial recognition tasks. The objective of this project is to extract LBP features from both training and test images, compare the resulting histograms using Euclidean distance, and classify the test images based on the closest match to the training data. We aim to achieve classification accuracy and visually present the LBP maps and histograms for a selected test image.

Methodology

Data Loading

We use the Python Imaging Library (PIL) to load the training and test images in grayscale format. The images are loaded from specified folders, and the grayscale conversion is performed during the image loading step.

Local Binary Pattern (LBP) Application

The LBP method is applied using a 3x3 window with a radius of 1 and 8 neighboring pixels. The LBP value for each pixel is calculated by comparing its intensity with that of its neighbors, converting the comparison to binary form, and then generating a unique binary pattern. Then, this pattern is converted to a decimal value. The LBP map for each image is generated by applying this method to each pixel.

Histogram Calculation

For each LBP map, we calculate the histogram by counting the occurrences of each LBP value throughout the image. The histogram is normalized so that the sum of all bins equals 1, ensuring that the histogram represents the distribution of LBP patterns.

Euclidean Distance

We use Euclidean distance to measure similarity between histograms. The Euclidean distance is calculated between the histogram of each test image and all training images. The test image is classified based on the training image with the smallest distance.

Prediction and Accuracy

After computing the Euclidean distances between histograms, we predict the subject ID for each test image by finding the training image with the minimum distance. The model accuracy is then calculated by comparing the predicted subject ID to the true subject ID for each test image.

Results

The accuracy of the face recognition system was evaluated in the test data set. The classification accuracy achieved was approximately 63.33%.

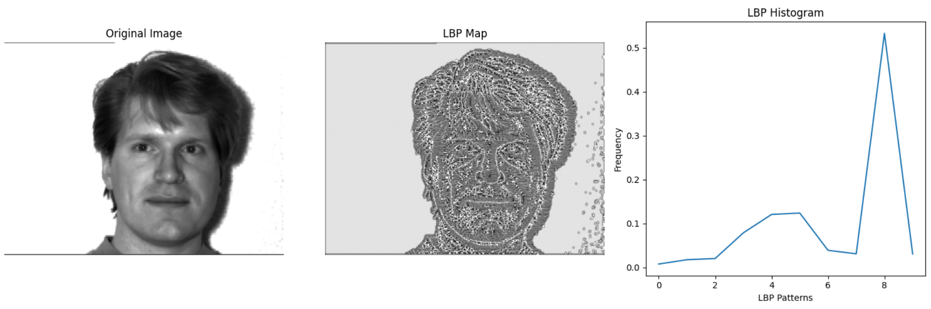

Figure 1: Example of a Test Image, its LBP Map, and Histogram

Analysis

From the results, we observe that the LBP method successfully captures the texture features of the face image. The LBP map clearly highlights the local texture patterns that are used for classification. The histogram provides information on the distribution of LBP values, which play a key role in distinguishing between different subjects.

The sharp peak in the LBP histogram indicates the dominance of certain local patterns, which are probably responsible for distinguishing this subject from others. Despite the accuracy achieved of 63.33%, the method may be sensitive to variations in lighting conditions, image resolution, and the complexity of facial features.

Conclusion

In this part, we successfully implemented a face recognition system using Local Binary Patterns. We computed the LBP features, generated histograms, and used Euclidean distance for classification. The accuracy achieved of **63.33%** demonstrates the effectiveness of LBP for face recognition in this scenario. However, further optimization and testing are needed for better performance, especially under varying illumination conditions.

2. Extracting Knowledge from Networks

(a)

In this task, we fine-tuned a ResNet50 model, pretrained on the ImageNet dataset, to classify images from the CIFAR-10 dataset. The ResNet50 model is a deep convolutional neural network with 50 layers. The model was pretrained on ImageNet, which contains 1,000 classes. We modify the last fully connected layer of the model to match the number of classes in CIFAR-10, which has 10 classes.

Methodology

Model Modification

The ResNet50 model was pretrained on the ImageNet dataset, so we removed its final fully connected layer (which was originally for 1,000 classes) and replaced it with a new fully connected layer designed to output 10 classes, corresponding to the CIFAR-10 dataset. The modified model architecture was as follows:

- Pretrained ResNet50 with the last layer replaced by a new fully connected layer for 10 classes.
- The pretrained layers were frozen to prevent further training, while the new layer was trained from scratch.

Data Preparation

The **CIFAR-10** data set consists of 60,000 images divided into 10 classes. The images were pre-processed as follows:

- The images were resized to **224x224** pixels to match the input size expected by ResNet50.
- The images were normalized using ImageNet statistics, with mean values '[0.485, 0.456, 0.406]' and standard deviation values '[0.229, 0.224, 0.225]'.

Training the Model

The model was fine-tuned for 20 epochs with a batch size of 32. We used the SGD optimizer with a learning rate of 0.001 and momentum 0.9, and the loss function used was Cross-Entropy Loss. The training process was as follows:

- We trained the model on the CIFAR-10 training set.
- The model was tested on the CIFAR-10 test set after each epoch to evaluate its performance.

Results

Training Results

The fine-tuned model achieved the following training accuracy after 20 epochs:

Epoch	Training Accuracy
1	67.9%
2	81.98%
3	87.42%
4	90.8%
5	93.41%
6	94.90%
7	96.04%
8	96.99%
9	97.52%
10	97.78%
11	98.29%
12	98.49%
13	98.59%
14	98.84%
15	99.10%
16	99.01%
17	98.93%
18	99.03%
19	99.33%
20	99.48%

Test Accuracy

After fine-tuning for 20 epochs, the model achieved a test accuracy of **85.22%** on the CIFAR-10 test set.

Conclusion

In this task, we successfully fine-tuned the ResNet50 model pre-training on ImageNet to classify images from the CIFAR-10 dataset. The model achieved an impressive **training accuracy** of **99.48%** and a **test accuracy** of **85.22%**. This shows that transferring knowledge from a large pre-trained model to a smaller dataset can significantly improve performance.

The results demonstrate the potential for fine-tuning pre-trained models for efficient learning on smaller datasets such as CIFAR-10.

(b)

In this part of the task, we built a simple **Convolutional Neural Network (CNN)** from scratch and trained it on the CIFAR-10 dataset. The network was evaluated on the test set to assess its performance.

Methodology

Dataset

The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. We utilized the training and test sets provided by CIFAR-10.

Data Preprocessing

The images were preprocessed as follows:

- The images were normalized to a range between 0 and 1 by dividing by 255.0.
- The images were reshaped and converted to the format $(32, 32, 3)$ to match the input shape expected by the model.
- Labels were one-hot encoded using TensorFlow's 'to_categorical' function to convert the categorical labels into a format suitable for classification.

Model Architecture

The custom CNN model was built with the following layers:

- 3 convolutional layers with ReLU activation, followed by max-pooling.
- Flatten layer to flatten the 3D feature maps into a 1D vector.
- Dense layer with 512 neurons and ReLU activation.
- Output layer with 10 neurons (corresponding to the 10 classes in CIFAR-10) and a softmax activation function.

The model was compiled with the Adam optimizer and categorical cross-entropy loss.

Training

The model was trained for 2 epochs with a batch size of 64. The accuracy was evaluated on the test set after each epoch.

Results

Training Results

The training results for the Student Model were as follows:

Epoch	**Validation Accuracy**
1	62.36%
2	68.13%

Test Accuracy

The final test accuracy on the CIFAR-10 test set after 2 epochs was **68.13**%.

Conclusion

We successfully built and trained a custom CNN model on the CIFAR-10 dataset. The model achieved a test accuracy of 68.13%, which shows a decent performance on the CIFAR-10 test set. While this is not as high as some state-of-the-art models, it demonstrates that even simple architectures can perform reasonably well on such datasets.

(c)

In this experiment, we applied Knowledge Distillation to train a small CNN model using a teacher model (ResNet50) pretrained on ImageNet. The ResNet50 model was frozen (its weights were not updated during training), and the student model was trained using the soft predictions (logits) from the teacher model. The distillation loss was calculated as the combination of Categorical Cross-Entropy Loss and KL Divergence Loss.

Methodology

- Teacher Model: ResNet50 pretrained on ImageNet.
- Student Model: A custom CNN with 3 convolutional layers, ReLU activation, and a fully connected layer, as described in part (b).
- Loss Function: The total loss is the combination of the Categorical Cross-Entropy Loss and the KL Divergence Loss between the teacher and student logits:

  $$\textbf{Total Loss} = \textbf{Cross-Entropy Loss} + \alpha \cdot \textbf{KL Divergence Loss}$$

  Where α is the weight for the KL Divergence term (set to 0.5).
- Training: The student model was trained for 10 epochs with a batch size of 64. The Adam optimizer was used with a learning rate of 0.001.

Results

The student model was trained for 5 epochs after applying knowledge distillation and achieved the following results:

Epoch	Training Accuracy	Validation Accuracy
1	19.16%	29.94%
2	29.09%	32.38%
3	31.93%	33.89%
4	33.76%	32.03%
5	34.58%	34.96%
6	35.94%	35.92%
7	35.74%	37.54%
8	37.23%	36.74%
9	37.45%	38.39%
10	38.38%	38.83%

The final test accuracy after 5 epochs of training with knowledge distillation was **70.95%**.

Conclusion

In this task, we applied Knowledge Distillation to a small CNN model by transferring knowledge from a pretrained ResNet50 model. The student model improved significantly, reaching **70.95**% test accuracy after distillation. This is an improvement over the **68.13**% test accuracy achieved in part (b) without distillation.

The results indicate that knowledge distillation is effective in transferring valuable information from a larger, pretrained model to a smaller model, enabling it to achieve better performance on the same task.