KNN - facial expression recognition

Kylian Varganyi

13 October 2024

1 Introduction

Facial expression recognition plays a crucial role in computer vision. One effective approach for feature extraction from images is using Local Binary Patterns (LBP), which capture texture by comparing each pixel to its neighboring pixels. In this report, we applied LBP to extract features from the FER2013 dataset and classified emotions using the K-Nearest Neighbors (KNN) algorithm.

2 Local Binary Patterns (LBP)

Local Binary Patterns (LBP) is a straightforward but highly effective texture descriptor. It works by comparing each pixel to its surrounding neighbors and creating a binary pattern based on these comparisons. This pattern is then transformed into a histogram that captures the texture of the image. The LBP technique is sensitive to texture variation and controlled by two parameters: the radius and the number of points sampled around each pixel. For this project, I tried different radius and number of points. Finally, the parameters that give the best value were :

- radius = 8
- number of points = $8 \times radius$

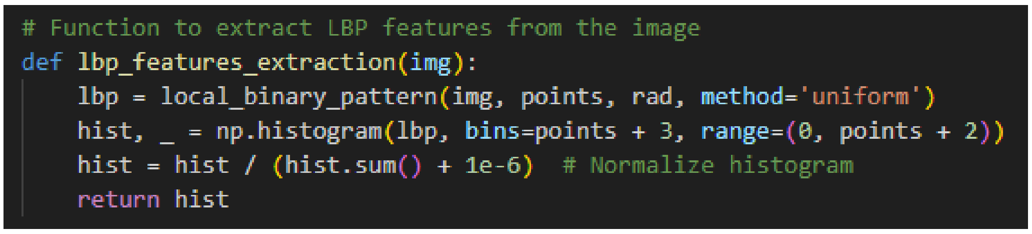
3 Code Description

3.1 Loading images

The first step is to load the images from the dataset. Each emotion is stored in its own directory, and images are read in grayscale and resized to 48x48 pixels. Histogram equalization is applied to enhance the contrast and ensuring more consistent feature extraction across different images.

3.2 Extraction of features with LBP

The second step is to used the Local Binary Pattern (LBP) technique in order to extract texture features from each image. LBP creates histograms of local texture patterns, which are then normalized. These histograms serve as the feature vectors for classification.



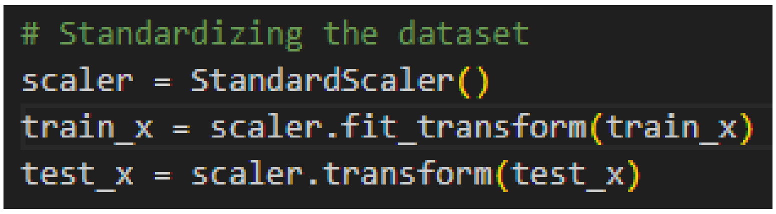
```python
# Function to extract LBP features from the image
def lbp_features_extraction(img):
    lbp = local_binary_pattern(img, points, rad, method='uniform')
    hist, _ = np.histogram(lbp, bins=points + 3, range=(0, points + 2))
    hist = hist / (hist.sum() + 1e-6)  # Normalize histogram
    return hist
```

Figure 1: LBP Features Extraction

I tried different LBP methods and the best was **uniform**. Other methods (ror, default) gave lower accuracy.

3.3 Features standardization

The extracted LBP features need to be standardized to ensure optimal performance of the KNN classifier. This step normalizes the feature vectors (zero mean and unit variance), which helps the model converge more effectively.



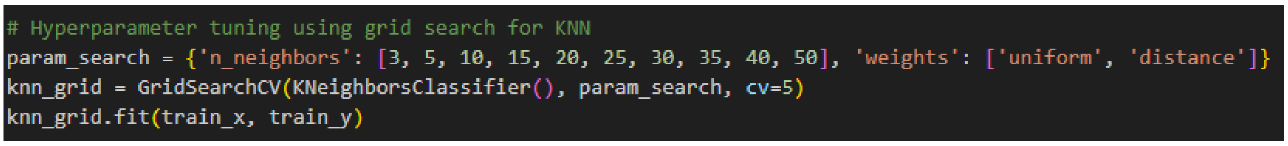
```
# Standardizing the dataset
scaler = StandardScaler()
train_x = scaler.fit_transform(train_x)
test_x = scaler.transform(test_x)
```

Figure 2: Features standardization

I used **StandardScaler** to standardize LBP features. It ensures that each feature has a mean of 0 and a standard deviation of 1, allowing the KNN classifier to treat all features equally and improving the model's performance.

3.4 Optimisation of hyperparameters of KNN

A grid search is employed to find the optimal hyperparameters for the KNN classifier. Specifically, the model tests different values for the number of neighbors and the weighting strategy. This ensures that the classifier is fine-tuned for the best possible accuracy.



```
# Hyperparameter tuning using grid search for KNN
param_search = {'n_neighbors': [3, 5, 10, 15, 20, 25, 30, 35, 40, 50], 'weights': ['uniform', 'distance']}
knn_grid = GridSearchCV(KNeighborsClassifier(), param_search, cv=5)
knn_grid.fit(train_x, train_y)
```

Figure 3: Automation of best hyperparameters

The first line of the Figure 3 defines the two hyperparameters to be tuned: the number of neighbors (with various values) and the weight function to use ('uniform' or 'distance').

The third line applies the grid search to the training data to train the model on each combination of hyperparameters to find the best configuration.

3.5 Evaluation of the model : Classification Report

After training, the model is evaluated on the test dataset. The predictions are compared to the true labels to generate a classification report that includes precision, recall, and F1-score for each emotion, as shown in Table 1.

Table 1: Classification Report

Emotion	Precision	Recall	F1-score	Support
Angry	0.29	0.16	0.21	958
Disgust	1.00	0.27	0.43	111
Fear	0.35	0.19	0.24	1024
Happy	0.32	0.70	0.43	1774
Neutral	0.33	0.20	0.25	1233
Sad	0.30	0.20	0.24	1247
Surprise	0.60	0.40	0.48	831
Accuracy			0.34	
Macro avg	0.45	0.30	0.33	7178
Weighted avg	0.36	0.34	0.32	7178

- **Precision:** The ratio of true positive predictions to the total number of positive predictions. It indicates the quality of the positive class predictions.
- **Recall:** The ratio of true positive predictions to the total number of actual positives. It reflects the model's ability to find all relevant instances.
- **F1-score:** The harmonic mean of precision and recall. It provides a balance between the two metrics, particularly useful when the class distribution is uneven.
- **Support:** The number of actual occurrences of each class in the specified dataset. It represents the quantity of true instances for each emotion.

3.6 Evaluation of the model : Confusion Matrix

Additionally, a confusion matrix is created to visualize the classification errors across different emotions. It's a table used to evaluate the performance of a classification model by comparing predicted labels with true labels.

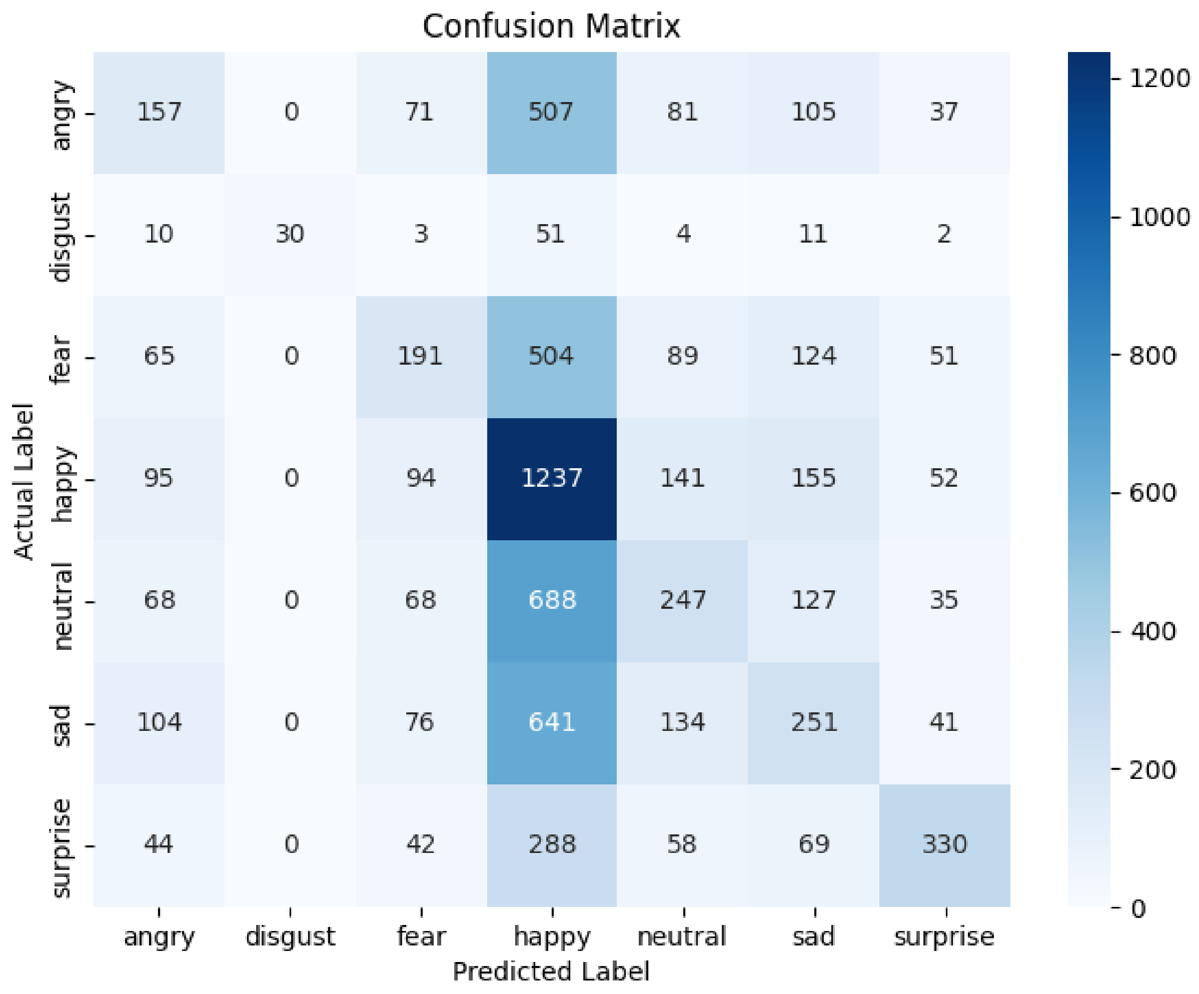


Figure 4: Confusion matrix of the model

3.6.1 Read the Confusion Matrix

The confusion matrix is created from rows and colums. Rows represent the **true labels** (what is actually present). Columns represent the **predicted labels** (what the model predicted). Each cell in the matrix contains the count of observations for the corresponding true and predicted labels.

3.6.2 Results Interpretation

1. **True positives** are values along the diagonal that indicate the number of correctly classified instances for each emotion.

2. **False positives** corresponds to the sum of each column, excluding the diagonal. It shows how many instances of other emotions were incorrectly classified as this class.

3. **False negatives** are the sum of each row, excluding the diagonal. It indicates how many instances of this class were incorrectly classified as other emotions.

3.6.3 Deduction from the Confusion Matrix

Emotions with high values on the diagonal are well classified. Conversely, emotions with high values off the diagonal indicate confusion with other emotions and may require improvements. For example, the "happy" class has a high number of true positives (1237), indicating good classification, while the "angry" class shows confusion with "happy".