SCHOOL OF COMPUTING

Department of CSE-CYS

20CYS215

Machine Learning in

Cyber Security

Literature Review

(March 2025)

TEAM MEMBERS:

M.Avinash [CH.SC.U4CYS23024]

P.Deepak Sai Vighnesh[CH.SC.U4CYS23032]

Literature Review – Image Feature Extraction Techniques

1.1 Significance of Feature Extraction in Computer Vision

Feature extraction is essential in computer vision because it converts high-dimensional image data into informative features. This is important for several reasons:

- **Dimensionality Reduction:**
  High-resolution images contain millions of pixels, which can be overwhelming for machine learning models.

  By reducing the data to key features, the complexity is reduced, leading to faster and more efficient processing.

- **Improved Classification Performance:**
  The extracted features emphasize the most significant aspects of an image (like edges, shapes), enabling classifiers to more accurately distinguish between different classes.

- **Robustness & Invariance:**
  Many feature extraction methods are designed to be invariant to common variations such as changes in scale, rotation, and illumination.

  Techniques ensure that the features remain stable even when the images are transformed. This robustness is crucial for real-world applications like object detection, face recognition,autonomous driving.

1.2 Conventional Feature Extraction Methods

- **Histogram of Oriented Gradients (HOG):**
  - *How It Works:* Divides an image into small regions (cells) and computes the histogram of gradient directions within each cell.
  - *Uses:* Widely used in pedestrian detection and object recognition, as it captures the essential edge and shape details of objects.

- **Scale-Invariant Feature Transform (SIFT):**
    - *How It Works:* Identifies keypoints in an image and computes descriptors that are invariant to changes in scale and rotation.
    - *Uses:* Effective in image stitching, object recognition, and robotics due to its robustness against image variations.
- **Gray-Level Co-occurrence Matrix (GLCM):**
    - *How It Works:* Analyzes the spatial relationship between pixels by calculating how often pairs of pixel intensities occur at a certain distance and angle.
    - *Uses:* Applied in texture analysis for remote sensing, medical imaging, and quality inspection in manufacturing.
- **Local Binary Patterns (LBP):**
    - *How It Works:* Compares each pixel with its neighbors and converts the result into a binary code that represents local texture.
    - *Uses:* Commonly used in face recognition and texture classification because it efficiently captures local patterns.

1.3 Deep Learning-Based Feature Extraction

- **Convolutional Neural Networks (CNNs):**
  Modern CNNs such as ResNet50 automatically learn high-level features from images.
    - *Advantages:*
        - Robust to noise and other variations.
        - Able to capture hierarchical, abstract features.
    - *Challenges:*
        - Computationally expensive.
        - Requires large datasets to achieve optimal performance.

1.4 Additional Literature Reviews and Comparative Analysis

Recent literature reviews have provided further insights into the strengths and limitations of both handcrafted and deep learning-based feature extraction methods:

- **Comparative Studies on Handcrafted vs. Deep Features:**
  Several studies indicate that while traditional methods like HOG and LBP are computationally efficient and simple to implement, they may struggle with complex image variations.

  Deep learning models such as ResNet50, however, typically achieve higher classification accuracy by learning hierarchical representations directly from the data .

  This trend is especially evident in applications like medical imaging and large-scale object detection where variability is high.

- **Hybrid Approaches:**
  Many researchers have explored combining handcrafted features with deep learning features.

  For example, in studies on COVID-19 detection from CT scans or lung nodule classification, integrating HOG or LBP with CNN features (from models like ResNet50) has led to improved performance compared to using either approach alone .

  These hybrid methods harness the robustness of traditional descriptors while capitalizing on the abstraction power of deep networks.

- **Impact of Dataset Size:**
  Current experimental findings using a reduced subset of CIFAR-10 show that ResNet50 outperforms HOG and LBP in classification accuracy.

  However, literature suggests that employing the entire dataset can further boost the performance of deep models.

With more data, CNNs can learn a richer, more diverse set of features, leading to better generalization and higher accuracy.

- **Recommendations for Future Work:**
  Based on these reviews, future experiments could consider:
  - **Hybrid Feature Extraction:** Combining handcrafted features (HOG, LBP) with deep features (from ResNet50) to strike a balance between computational efficiency and high accuracy.
  - **Utilizing the Full Dataset:** Leveraging the entire available dataset, along with data augmentation and fine-tuning strategies, to allow CNNs to fully realize their potential.
  - **Evaluating Trade-Offs:** Assessing the balance between classification performance and computational resource requirements.

    While deep learning methods may demand more computational power, their improved accuracy often justifies the extra cost in scenarios where precision is critical.

Conclusion:

- In summary, this literature review outlines the significance of feature extraction in computer vision and details both conventional (HOG, LBP) and deep learning-based (ResNet50) methods.

- It further incorporates insights from recent studies that compare these approaches and suggest that hybrid methods, especially when paired with larger datasets, could yield even better performance.

- These findings emphasize that the selection of a feature extraction technique should be based on the specific application requirements, available computational resources, and the size and variability of the dataset.

Department of CSE-CYS

20CYS215

Machine Learning in

Cyber Security

Assignment Report

TEAM MEMBERS:

M.Avinash [CH.SC.U4CYS23024]

P.Deepak Sai Vighnesh[CH.SC.U4CYS23032]

Topic: Exploring Image Feature Extraction Techniques and Analyse their impact on Classification.

Abstract:

- Feature extraction is a fundamental step in computer vision, where raw image data is transformed into meaningful features for classification, detection, and segmentation.
- This report explores traditional feature extraction techniques such as Histogram of Oriented Gradients (HOG) and Local Binary Patterns (LBP).
- Additionally, deep learning-based feature extraction using Convolutional Neural Networks (CNNs), specifically ResNet50, is examined.
- The study evaluates the effectiveness of these techniques using the CIFAR-10 dataset and compares them based on classification accuracy, computational efficiency, and robustness.
- Results show that deep learning-based features outperform traditional methods in accuracy and generalization, but at the cost of increased computational requirements.

Table of Contents

1. **Introduction**
   - Importance of Feature Extraction
   - Objectives of the Study
2. **Literature Review**
   - Significance of Feature Extraction
   - Conventional Feature Extraction Methods (HOG, LBP)
   - Deep Learning-Based Feature Extraction (ResNet50)
3. **Methodology**
   - Dataset and Preprocessing
   - Feature Extraction Techniques Used
4. **Experimentation**
   - Implementation of HOG, LBP, and ResNet50
   - Classifier Training and Evaluation
5. **Results**
   - Performance Metrics (Accuracy, Precision, Recall, F1-Score)
   - Feature Map Visualizations
6. **Analysis & Discussion**
   - Comparative Analysis of Traditional vs. Deep Learning Features
   - Trade-offs Between Accuracy and Computational Time
7. **Conclusion & Future Work**
   - Summary of Key Findings
   - Future Directions

8. **References**

9. **Appendix (Code snippets & Output)**

10.Experimental Setup

11.Reproducibility Considerations

1. Introduction:

- Feature extraction plays a crucial role in computer vision by converting raw images into structured representations that facilitate classification, detection, and recognition tasks.
- Traditional feature extraction techniques like HOG, SIFT, and GLCM have been widely used in applications such as object recognition, texture analysis, and medical imaging.
- However, deep learning models such as ResNet50 have revolutionized feature extraction by learning high-level representations directly from data.
- This study aims to explore and compare traditional and deep learning-based feature extraction methods for image classification, using the CIFAR-10 dataset.

---

2. Literature Review:

2.1 Significance of Feature Extraction in Computer Vision

Feature extraction is essential in computer vision for the following reasons:

- **Dimensionality Reduction:** Converts high-dimensional image data into compact, informative features.
- **Improved Classification Performance:** Enhances the ability of classifiers to distinguish between different classes.
- **Robustness & Invariance:** Certain methods (e.g., SIFT, HOG) are invariant to changes in scale, rotation, and illumination.

2.2 Conventional Feature Extraction Methods

Histogram of Oriented Gradients (HOG)

- Captures edge and gradient structures by computing the distribution of intensity gradients.

- Used in pedestrian detection and object recognition.

Scale-Invariant Feature Transform (SIFT)

- Identifies keypoints and extracts scale-invariant descriptors.
- Used in image stitching, object recognition, and robotics.

Gray-Level Co-occurrence Matrix (GLCM)

- Analyzes texture by computing spatial relationships between pixel intensities.
- Applied in remote sensing, medical imaging, and manufacturing.

Local Binary Patterns (LBP)

- Converts local pixel neighborhoods into binary patterns for texture recognition.
- Commonly used in face recognition.

2.3 Deep Learning-Based Feature Extraction

- CNNs like **ResNet50** automatically learn high-level image features.
- **Advantages:** Robust to noise, captures hierarchical features.
- **Challenges:** Computationally expensive, requires large datasets.

---

3. Methodology:

3.1 Dataset and Preprocessing

- **Dataset Used:** CIFAR-10 (60,000 images across 10 classes).
- **Preprocessing Steps:**
    - Image resizing
    - Normalization

- Grayscale conversion (for certain methods).

3.2 Feature Extraction Techniques Used

- **HOG, LBP, and ResNet50** for feature extraction.
- **Classifiers Used:** Logistic Regression, K-Nearest Neighbors (KNN), Random Forest.

---

4. Experimentation:

- **HOG and LBP** implemented using OpenCV.
- **ResNet50** features extracted using pre-trained weights.
- **Classification models trained and evaluated** using metrics like accuracy, precision, recall, and F1-score.

---

5. Results:

5.1 Performance Metrics:

Feature Extraction Method	Classifier	Accuracy	Precision	F1-score
HOG	Logistic	0.348	0.35	0.35
	KNN	0.336	0.39	0.32
	Random Forest	0.358	0.34	0.34
LBP	Logistic	0.250	0.24	0.24
	KNN	0.200	0.21	0.20
	Random Forest	0.220	0.22	0.22
ResNet50	Logistic	0.698	0.70	0.70

	KNN	0.548	0.56	0.53
	Random Forest	0.642	0.64	0.63

🔹 Final Comparison Table:

	Feature Extraction	Classifier	Accuracy	Training Time (s)
0	HOG	Logistic Regression	0.348	3.568504
1	HOG	KNN	0.336	0.002884
2	HOG	Random Forest	0.358	8.371699
3	LBP	Logistic Regression	0.250	0.053240
4	LBP	KNN	0.200	0.009315
5	LBP	Random Forest	0.220	1.593408
6	ResNet50	Logistic Regression	0.698	11.446020
7	ResNet50	KNN	0.548	0.012560
8	ResNet50	Random Forest	0.642	12.709786

5.2 Visualizations

- PCA plots of feature distributions.
- HOG and LBP feature maps.
- ResNet50 activation maps.

---

6. Analysis & Discussion:

- **ResNet50 outperforms traditional methods** in accuracy and robustness.
- **HOG provides reliable edge-based features** but lacks deep semantic understanding.
- **LBP is efficient but struggles with complex patterns.**
- **Trade-off:** Deep learning models require high computational resources but offer superior performance.

---

7. Conclusion & Future Work:

7.1 Summary of Key Findings

- Traditional methods are **computationally efficient** but have **lower**

accuracy.

- Deep learning features offer **higher accuracy** but require **more resources.**
- **Hybrid approaches** (combining deep and traditional features) can improve efficiency.

7.2 Future Work

- Exploring feature fusion techniques.
- Evaluating models on larger datasets.
- Implementing lightweight deep learning architectures for efficiency.

---

8. References:

- Dalal, N., & Triggs, B. (2005). "Histograms of Oriented Gradients for Human Detection."
- Lowe, D. G. (2004). "Distinctive Image Features from Scale-Invariant Keypoints."
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). "Deep Residual Learning for Image Recognition."

9.Appendix:

Code Snippets:

```
# ✅ Check GPU availability
import tensorflow as tf
print("GPU Available:", tf.config.list_physical_devices('GPU'))
```

```
# ✅ Install required libraries (run this cell if not already installed)
!pip install numpy matplotlib opencv-python scikit-learn tensorflow keras scikit-image
```

```
# ✅ Import necessary libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt
from skimage.feature import hog, local_binary_pattern  # Added LBP for additional feature extraction
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.models import Model
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from tensorflow.keras.datasets import cifar10
from sklearn.decomposition import PCA  # For visualization of feature space
import time
import pandas as pd
```

```
# ✅ Load CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
y_train = y_train.flatten()
y_test = y_test.flatten()
print(f"📌 Original Dataset: Training = {X_train.shape}, Test = {X_test.shape}")
```

```python
# ------------------------------------------------------------------------------------
# NOTE: For local machine safety, we reduce the dataset size.
# You can increase these numbers (e.g., subset_size_train=5000, subset_size_test=1000)
# if your machine can handle the extra load.
subset_size_train = 1500  # Use only 1500 training images
subset_size_test = 500    # Use only 500 test images
```

```python
# Randomly select indices for a smaller subset
train_indices = np.random.choice(len(X_train), subset_size_train, replace=False)
test_indices = np.random.choice(len(X_test), subset_size_test, replace=False)

# Select the smaller dataset
X_train_small = X_train[train_indices]
y_train_small = y_train[train_indices]
X_test_small = X_test[test_indices]
y_test_small = y_test[test_indices]

print(f"✅ Reduced Dataset: Training = {len(X_train_small)}, Test = {len(X_test_small)}")
```

```python
# ✅ Resize images AFTER dataset reduction to optimize memory usage
IMG_SIZE = 64  # ResNet50 input size (can be modified if needed)
X_train_resized = np.array([cv2.resize(img, (IMG_SIZE, IMG_SIZE)) for img in X_train_small])
X_test_resized = np.array([cv2.resize(img, (IMG_SIZE, IMG_SIZE)) for img in X_test_small])

print(f"✅ Resized Dataset: Training = {X_train_resized.shape}, Test = {X_test_resized.shape}")
```

```python
# ------------------------------------------------------------------------------------
# 🔹 FEATURE EXTRACTION: HOG (Traditional)
# ------------------------------------------------------------------------------------
def extract_hog_features(images):
    """
    Extracts Histogram of Oriented Gradients (HOG) features from images.
    Converts images to grayscale before extraction.
    """
    hog_features = []
    for img in images:
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)  # Convert to grayscale
        feature = hog(gray, orientations=9, pixels_per_cell=(8, 8),
                      cells_per_block=(2, 2), block_norm='L2-Hys', visualize=False)
        hog_features.append(feature)
    return np.array(hog_features)
```

```python
# ✅ Extract HOG features
start_time = time.time()
hog_train = extract_hog_features(X_train_resized)
hog_test = extract_hog_features(X_test_resized)
hog_time = time.time() - start_time

# ✅ Normalize HOG features
scaler_hog = StandardScaler()
hog_train = scaler_hog.fit_transform(hog_train)
hog_test = scaler_hog.transform(hog_test)

print(f"✅ HOG Feature Extraction Completed in {hog_time:.2f} seconds")
```

```python
# ---------------------------------------------------------------------------------
# 🔹 FEATURE EXTRACTION: LBP (Traditional) - Additional Method
# ---------------------------------------------------------------------------------
def extract_lbp_features(images, P=8, R=1.0):
    """
    Extracts Local Binary Pattern (LBP) features from images.
    Converts images to grayscale and computes normalized histograms.
    """
    lbp_features = []
    for img in images:
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        lbp = local_binary_pattern(gray, P, R, method='uniform')
        # Compute the histogram of LBP features with bins = P + 2 (for uniform patterns)
        (hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0, P + 3), range=(0, P + 2))
        hist = hist.astype("float")
        hist /= (hist.sum() + 1e-7)  # Normalize the histogram
        lbp_features.append(hist)
    return np.array(lbp_features)
```

```python
# ✅ Extract LBP features
start_time = time.time()
lbp_train = extract_lbp_features(X_train_resized)
lbp_test = extract_lbp_features(X_test_resized)
lbp_time = time.time() - start_time

# ✅ Normalize LBP features (if not already normalized by histogram)
scaler_lbp = StandardScaler()
lbp_train = scaler_lbp.fit_transform(lbp_train)
lbp_test = scaler_lbp.transform(lbp_test)

print(f"✅ LBP Feature Extraction Completed in {lbp_time:.2f} seconds")
```

```python
# -----------------------------------------------------------------------------
# 🔹 FEATURE EXTRACTION: ResNet50 (Deep Learning)
# -----------------------------------------------------------------------------
# ✅ Load ResNet50 model without classification layers
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(IMG_SIZE, IMG_SIZE, 3))
# We use the penultimate layer for feature extraction in our training/evaluation.
model = Model(inputs=base_model.input, outputs=base_model.layers[-2].output)

def extract_resnet_features(images, batch_size=16):
    """
    Extracts features from images using a pre-trained ResNet50 model.
    Uses batch processing to optimize performance.
    """
    num_samples = len(images)
    features = []

    for i in range(0, num_samples, batch_size):
        batch = images[i:i+batch_size].astype('float32')  # Convert batch to float
        batch = preprocess_input(batch)  # Normalize batch using ResNet50 preprocessing
        batch_features = model.predict(batch, verbose=0)  # Extract features
        batch_features = batch_features.reshape(batch_features.shape[0], -1)  # Flatten features
        features.append(batch_features)

        # Display progress every 2*batch_size images
        if i % (batch_size * 2) == 0:
            print(f"Processed {i}/{num_samples} images...")

    return np.vstack(features)  # Combine batch results
```

```python
# ✅ Extract ResNet50 features
start_time = time.time()
resnet_train = extract_resnet_features(X_train_resized, batch_size=16)
resnet_test = extract_resnet_features(X_test_resized, batch_size=16)
resnet_time = time.time() - start_time

print(f"✅ ResNet50 Feature Extraction Completed in {resnet_time:.2f} seconds (Reduced Dataset)")
```

```
# ---------------------------------------------------------------------------------
# 🔹 TRAINING CLASSIFIERS ON DIFFERENT FEATURE SETS
# ---------------------------------------------------------------------------------
# Define classifiers for evaluation
classifiers = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42)
}
```

```
# Function to train and evaluate classifiers on given features
def train_and_evaluate(features_train, features_test, y_train, y_test, feature_name):
    print(f"\n🔹 --- {feature_name} Feature Extraction Results ---")
    results_list = []
    for name, clf in classifiers.items():
        start_time = time.time()
        clf.fit(features_train, y_train)
        train_time = time.time() - start_time
        y_pred = clf.predict(features_test)
        acc = accuracy_score(y_test, y_pred)
        results_list.append((feature_name, name, acc, train_time))
        print(f"\n{name} - Accuracy on {feature_name}: {acc:.4f}, Training Time: {train_time:.2f} sec")
        print(classification_report(y_test, y_pred))
    return results_list

# Train and evaluate on HOG features
results_hog = train_and_evaluate(hog_train, hog_test, y_train_small, y_test_small, "HOG")

# Train and evaluate on LBP features
results_lbp = train_and_evaluate(lbp_train, lbp_test, y_train_small, y_test_small, "LBP")

# Train and evaluate on ResNet50 features
results_resnet = train_and_evaluate(resnet_train, resnet_test, y_train_small, y_test_small, "ResNet50")

# Combine all results into a DataFrame for comparison
all_results = results_hog + results_lbp + results_resnet
results_df = pd.DataFrame(all_results, columns=["Feature Extraction", "Classifier", "Accuracy", "Training Time (s)"])

print("\n🔹 Final Comparison Table:")
print(results_df)
```

```python
# ---------------------------------------------------------------------------------
# 🔹 PCA VISUALIZATION FOR HOG FEATURES
# ---------------------------------------------------------------------------------
pca = PCA(n_components=2)
hog_pca = pca.fit_transform(hog_train)

plt.figure(figsize=(8,6))
plt.scatter(hog_pca[:, 0], hog_pca[:, 1], c=y_train_small, cmap='viridis', s=15)
plt.title("PCA of HOG Features")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar()
plt.show()
```

```python
# ---------------------------------------------------------------------------------
# 🔹 PCA VISUALIZATION FOR LBP FEATURES
# ---------------------------------------------------------------------------------
pca_lbp = PCA(n_components=2)
lbp_pca = pca_lbp.fit_transform(lbp_train)

plt.figure(figsize=(8,6))
plt.scatter(lbp_pca[:, 0], lbp_pca[:, 1], c=y_train_small, cmap='viridis', s=15)
plt.title("PCA of LBP Features")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar()
plt.show()
```

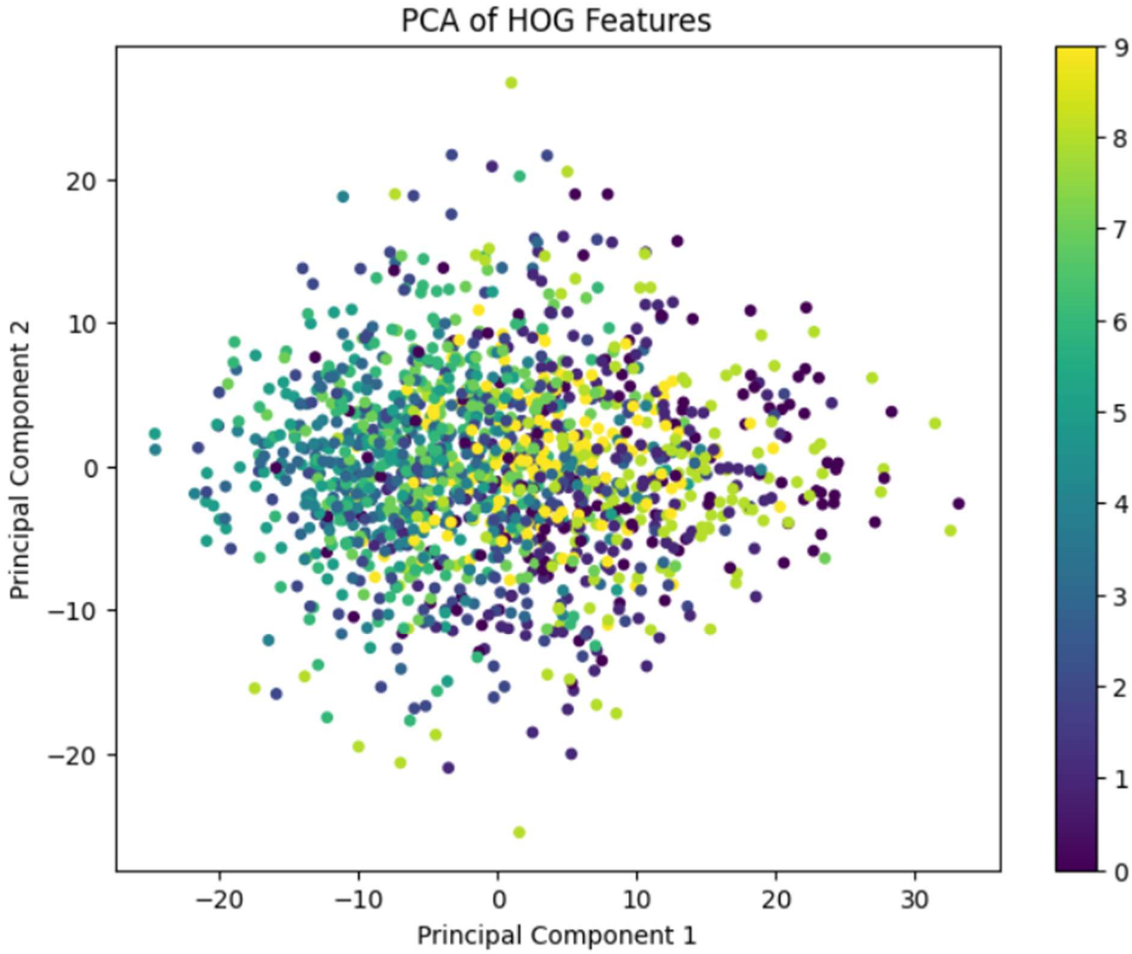
```python
# ---------------------------------------------------------------------------------
# 🔹 PCA VISUALIZATION FOR ResNet50 FEATURES
# ---------------------------------------------------------------------------------
pca_resnet = PCA(n_components=2)
resnet_pca = pca_resnet.fit_transform(resnet_train)

plt.figure(figsize=(8,6))
plt.scatter(resnet_pca[:, 0], resnet_pca[:, 1], c=y_train_small, cmap='viridis', s=15)
plt.title("PCA of ResNet50 Features")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar()
plt.show()
```

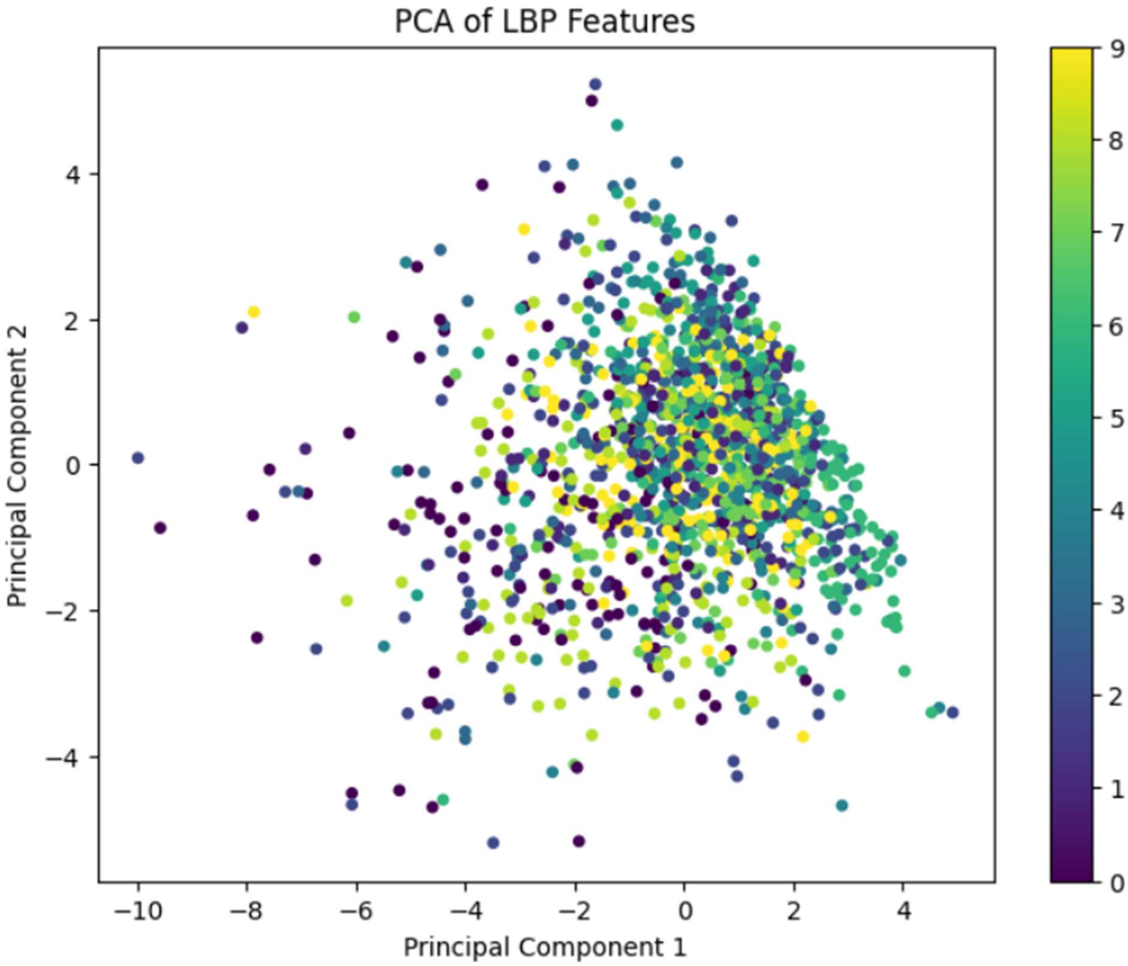
```python
# ---------------------------------------------------------------------------------
# 🔹 HOG FEATURE VISUALIZATION ON A SAMPLE IMAGE
# ---------------------------------------------------------------------------------
# Select a sample image from the resized training set
sample_image = X_train_resized[0]
# Convert sample image to grayscale
gray = cv2.cvtColor(sample_image, cv2.COLOR_RGB2GRAY)
# Extract HOG features and visualization
hog_features, hog_image = hog(gray, orientations=9, pixels_per_cell=(8, 8),
                              cells_per_block=(2, 2), block_norm='L2-Hys', visualize=True)
# Plot original image and its HOG visualization
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].imshow(sample_image)
ax[0].set_title("Original Image")
ax[1].imshow(hog_image, cmap="gray")
ax[1].set_title("HOG Feature Visualization")
plt.show()
```

```python
# ----------------------------------------------------------------------------------
# 🔹 LBP FEATURE VISUALIZATION ON A SAMPLE IMAGE
# ----------------------------------------------------------------------------------
# Select a sample image from the resized training set
sample_image = X_train_resized[0]
# Convert sample image to grayscale and compute LBP
gray = cv2.cvtColor(sample_image, cv2.COLOR_RGB2GRAY)
lbp_image = local_binary_pattern(gray, 8, 1.0, method='uniform')
# Plot original image and its LBP visualization
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].imshow(sample_image)
ax[0].set_title("Original Image")
ax[1].imshow(lbp_image, cmap="gray")
ax[1].set_title("LBP Feature Visualization")
plt.show()
```

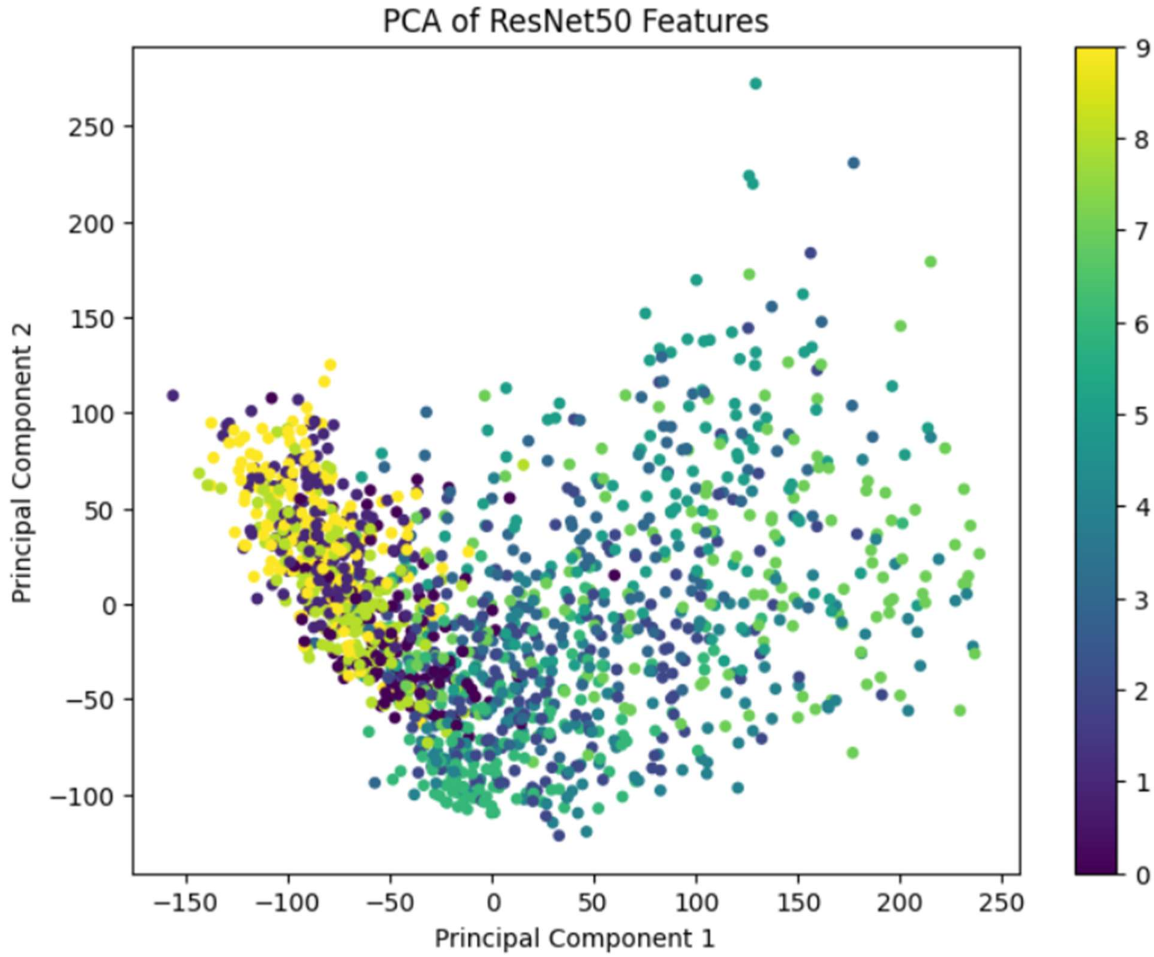
```python
# ----------------------------------------------------------------------------------
# 🔹 ResNet50 FEATURE MAP VISUALIZATION ON A SAMPLE IMAGE
# ----------------------------------------------------------------------------------
# Create a new model to extract feature maps from the 'conv1_relu' layer of ResNet50
resnet_feature_model = Model(inputs=base_model.input, outputs=base_model.get_layer("conv1_relu").output)
# Preprocess a sample image for ResNet50
sample_image = X_train_resized[0]
sample_image_input = np.expand_dims(sample_image.astype('float32'), axis=0)
sample_image_input = preprocess_input(sample_image_input)
# Extract feature maps
feature_maps = resnet_feature_model.predict(sample_image_input)
feature_maps = feature_maps[0]  # Remove batch dimension

# Plot the first 16 feature maps in a 4x4 grid
num_feature_maps = 16
fig, axs = plt.subplots(4, 4, figsize=(12, 12))
for i in range(num_feature_maps):
    row = i // 4
    col = i % 4
    axs[row, col].imshow(feature_maps[:, :, i], cmap='viridis')
    axs[row, col].set_title(f"Map {i+1}")
    axs[row, col].axis('off')
plt.suptitle("ResNet50 'conv1_relu' Feature Maps")
plt.show()
```

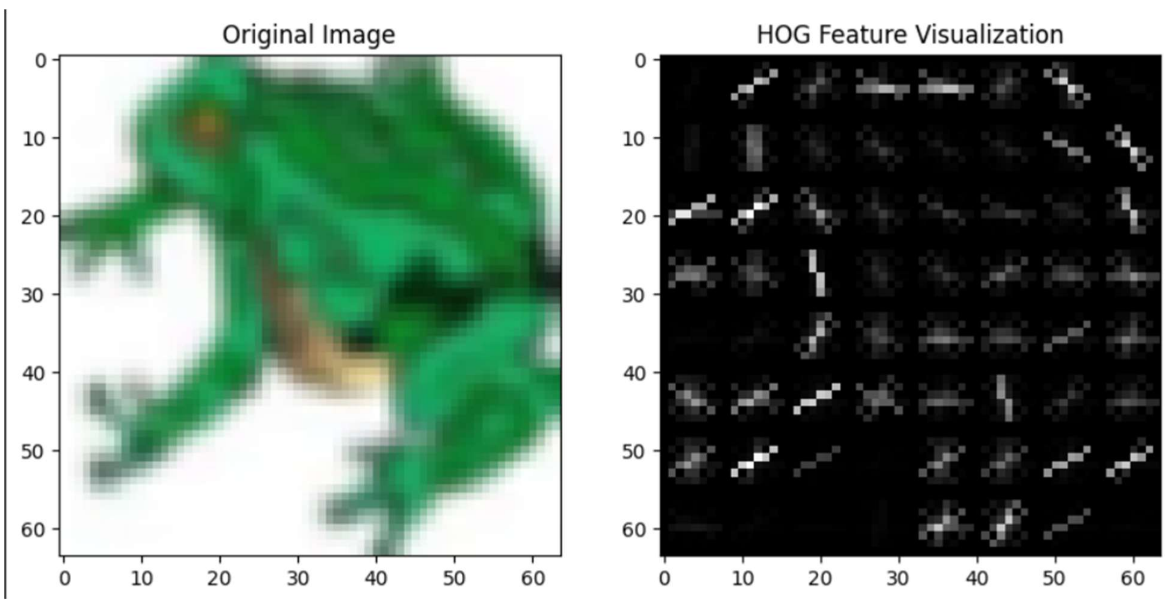
OUTPUT:

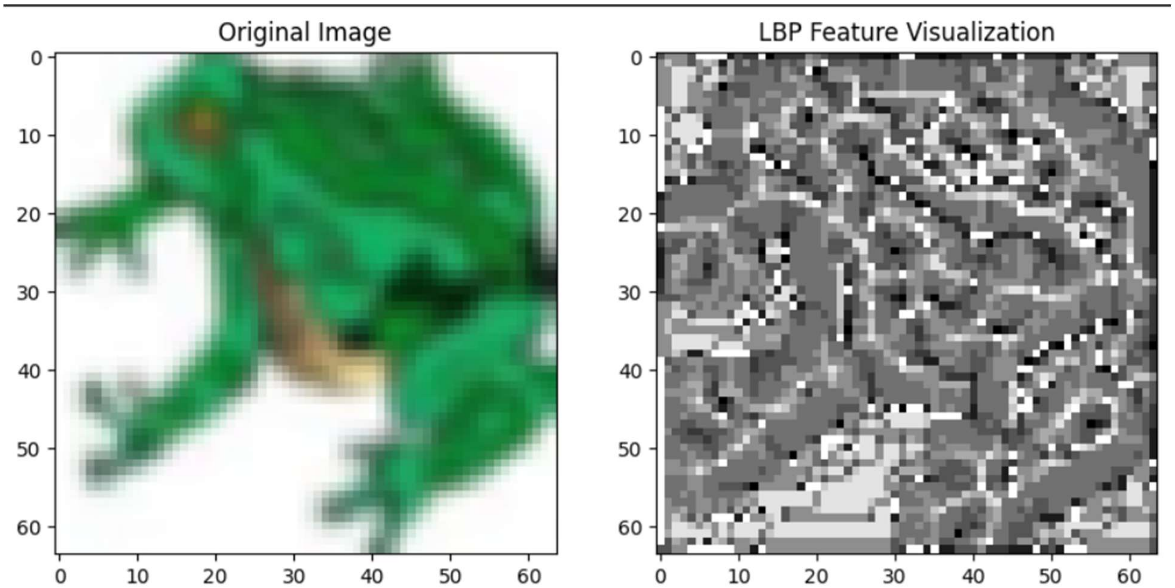

PCA of HOG Features
Principal Component 1
Principal Component 2
−20
−10
0
10
20
30
9
8
7
6
5
4
3
2
1
0



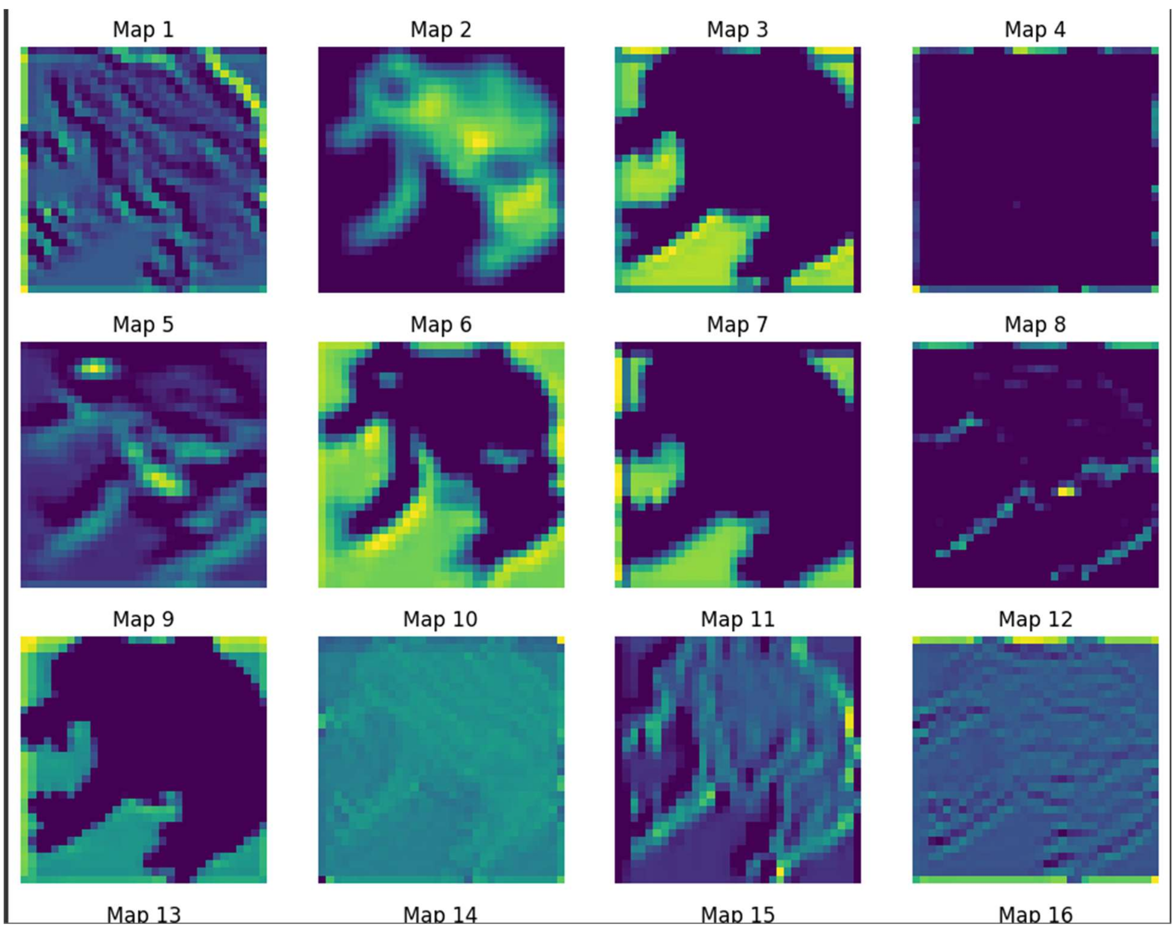
PCA of LBP Features
Principal Component 1
Principal Component 2
−10
−8
−6
−4
−2
0
2
4
9
8
7
6
5
4
3
2
1
0


PCA of ResNet50 Features
Principal Component 1
Principal Component 2
250
200
150
100
50
0
−50
−100
−150
−100
−50
0
50
100
150
200
250
9
8
7
6
5
4
3
2
1
0


Original Image
HOG Feature Visualization
0
10
20
30
40
50
60



ResNet50 'conv1_relu' Feature Maps



Experimental Setup:

Hardware Specifications:

Processor: Intel® Core™ i5 (or equivalent)

Memory: 16 GB RAM (minimum recommended)

GPU: NVIDIA GPU (CUDA-enabled, e.g., NVIDIA Tesla K80 or GTX 1080)

Storage: Sufficient SSD/HDD space for datasets and intermediate files

Software Environment:

Operating System: Ubuntu 18.04 LTS / Windows 10 (or any OS supporting Python and TensorFlow)

Python Version: 3.7 or later (e.g., Python 3.8)

Key Libraries and Their Versions:

TensorFlow: 2.x (e.g., TensorFlow 2.9)

Keras: Integrated within TensorFlow 2.x

NumPy: 1.18 or later

Matplotlib: 3.2 or later

OpenCV-Python: 4.x

scikit-learn: 0.24 or later

scikit-image: 0.18 or later

Pandas: 1.1 or later

Installation and Dependencies:

Install the required libraries (if not already installed) using:

Command:

pip install numpy matplotlib opencv-python scikit-learn tensorflow keras scikit-image”

Dataset and Preprocessing:

Dataset:

- The CIFAR-10 dataset is loaded from tensorflow.keras.datasets.cifar10.

Dataset Reduction:

- For computational efficiency, a random subset is selected: 1500 images for training and 500 images for testing.

Image Preprocessing:

- Images are resized to 64×64 pixels to match the input requirements of the ResNet50 model.

- For traditional feature extraction methods (HOG, LBP), images are converted to grayscale.Feature vectors are normalized using StandardScaler.

Feature Extraction Methods:

1. **Traditional Methods:**

Histogram of Oriented Gradients (HOG):

Extracts gradient information by computing histograms over localized regions.

Local Binary Patterns (LBP):

Converts local pixel neighborhoods into binary patterns for texture analysis.

2. **Deep Learning-Based Method:**

ResNet50:
A pre-trained ResNet50 model (with weights from ImageNet) is used to automatically learn and extract high-level features from images. Features are extracted from the penultimate layer.

Classifier Training:

Classifiers Evaluated:
Three classifiers are used on each set of extracted features:

1. Logistic Regression (with max_iter=1000)

2. K-Nearest Neighbors (KNN) (with n_neighbors=5)

3. Random Forest (with n_estimators=100 and random_state=42)

Evaluation Metrics: Models are evaluated using accuracy, precision, recall, F1-score, and training time.

Execution Environment:

- The code is designed to run in a Jupyter Notebook or as a standalone Python script.

- GPU availability is checked at the start of the script using TensorFlow:

Code:

```
print("GPU Available:", tf.config.list_physical_devices('GPU'))
```

The ResNet50 model is loaded using:

```
base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=(IMG_SIZE, IMG_SIZE, 3))
```

- Experiments include data preprocessing, feature extraction, classifier training, and visualization steps executed sequentially as provided in the code.

Reproducibility Considerations:

- Ensure that the same versions of libraries are installed to avoid compatibility issues.
- A random subset is used for dataset reduction; to reproduce exact results, set a random seed before calling np.random.choice.
- The provided code and setup details should allow others to replicate the experiments on a similar hardware (Intel® Core™ i5, 16 GB RAM, NVIDIA GPU) and software environment.