Feature Extraction and Classification using CNN and GMM on CIFAR-10 Dataset

1. Introduction

This project aims to classify the CIFAR-10 dataset using a Gaussian Mixture Model (GMM). To enhance classification performance, feature extraction is performed using a Convolutional Neural Network (CNN). The extracted features are then analyzed and utilized by the GMM for classification.

---

2. Methodology

2.1 Dataset

- The **CIFAR-10 dataset** consists of **60,000 color images (32x32 pixels) in 10 classes**, with 6,000 images per class.
- It is divided into **50,000 training images** and **10,000 testing images**.
- Each image has three color channels: **Red, Green, and Blue (RGB)**.

2.2 Data Preprocessing

- The dataset is loaded using `cifar10.load_data()`.
- Normalization is performed by **scaling pixel values to [0,1]**.

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
```

---

3. Model Architecture

3.1 CNN Model for Feature Extraction

A **Convolutional Neural Network (CNN)** is implemented using the **Functional API** in TensorFlow. The CNN consists of:

- **Convolutional layers** to learn feature maps.
- **Batch Normalization** to stabilize training.
- **MaxPooling layers** to reduce spatial dimensions.
- **Flatten layer** to convert the feature map into a 1D vector.
- **Fully connected layers** for feature extraction.

- **Dropout** to reduce overfitting.
- **Softmax output layer** for classification.

```
input_layer = Input(shape=(32, 32, 3))
x = Conv2D(32, (3, 3), padding="same", activation="relu")(input_layer)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)
x = Conv2D(64, (3, 3), padding="same", activation="relu")(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)
x = Flatten()(x)
feature_layer = Dense(256, activation="relu")(x)  # Feature Extraction Layer
x = Dropout(0.5)(feature_layer)
output_layer = Dense(10, activation="softmax")(x)
cnn_model = Model(inputs=input_layer, outputs=output_layer)
```

---

4. Training the CNN Model

4.1 Compilation & Callbacks

- The model is compiled using the **Adam optimizer** and **Sparse Categorical Crossentropy** loss.
- Callbacks:
    - **EarlyStopping**: Stops training if validation loss doesn't improve for 3 consecutive epochs.
    - **ReduceLROnPlateau**: Reduces learning rate if validation loss plateaus.

```
cnn_model.compile(optimizer="adam", loss="sparse_categorical_crossentropy",
metrics=["accuracy"])
early_stopping = EarlyStopping(monitor="val_loss", patience=3,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.2, patience=2)
```

4.2 Training

- The model is trained for **15 epochs** with a **batch size of 128**.

```
history = cnn_model.fit(x_train, y_train, epochs=15, batch_size=128,
validation_data=(x_test, y_test), callbacks=[early_stopping, reduce_lr])
```

---

5. Feature Extraction

- A **Feature Extractor Model** is created using the trained CNN, outputting features from the dense layer.
- Features are extracted for both the training and test sets.

```
feature_extractor = Model(inputs=cnn_model.input, outputs=feature_layer)
train_features = feature_extractor.predict(x_train)
test_features = feature_extractor.predict(x_test)
```

---

6. PCA for Dimensionality Reduction

- **Principal Component Analysis (PCA)** reduces the feature dimensions to 50 components to simplify clustering.

```
pca = PCA(n_components=50)
train_pca = pca.fit_transform(train_features)
test_pca = pca.transform(test_features)
```

---

7. Clustering with Gaussian Mixture Model (GMM)

- **GMM is used to cluster** the feature representations into 10 groups (corresponding to the 10 CIFAR-10 classes).

```
gmm = GaussianMixture(n_components=10, covariance_type="full",
random_state=42)
gmm.fit(train_pca)
gmm_preds = gmm.predict(test_pca)
```

- **Cluster-to-label mapping** ensures proper alignment with CIFAR-10 labels.

```
def map_gmm_to_labels(gmm_preds, y_true):
    labels = np.zeros_like(gmm_preds)
    for i in range(10):
        mask = gmm_preds == i
        if np.sum(mask) > 0:
            labels[mask] = mode(y_true[mask], keepdims=True)[0][0]
    return labels.astype(int)
```

---

8. Model Evaluation

- Accuracy of GMM-based classification is computed.

```
gmm_labels = map_gmm_to_labels(gmm_preds, y_test)
accuracy = np.mean(gmm_labels == y_test.flatten())
print(f"GMM Classification Accuracy: {accuracy * 100:.2f}%")
```

- Confusion Matrix visualization:

```
cm = confusion_matrix(y_test.flatten(), gmm_labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap="Blues", values_format="d")
plt.title("Confusion Matrix")
plt.show()
```

- Cluster visualization:

```
def visualize_cluster_grid(x_test, gmm_preds, num_clusters=10,
samples_per_cluster=5):
    fig, axes = plt.subplots(num_clusters, samples_per_cluster, figsize=(15,
20))
    for cluster in range(num_clusters):
        cluster_indices = np.where(gmm_preds == cluster)[0]
        sample_indices = np.random.choice(cluster_indices,
min(samples_per_cluster, len(cluster_indices)), replace=False)
        for idx in range(samples_per_cluster):
            ax = axes[cluster, idx]
            if idx < len(sample_indices):
                ax.imshow(x_test[sample_indices[idx]])
                ax.set_xticks([])
                ax.set_yticks([])
    plt.tight_layout()
    plt.show()
```

---

9. Achievements & Observations

- Successfully **trained a CNN** for feature extraction.
- Used **PCA for dimensionality reduction**, improving computational efficiency.
- Implemented **GMM for clustering CIFAR-10 images** based on extracted features.
- Achieved **unsupervised classification accuracy of 62%**
- Visualized **clusters and classification results** for further analysis.

---

10. Conclusion

This project successfully demonstrated the classification of the CIFAR-10 dataset using a Gaussian Mixture Model (GMM) with feature extraction performed by a Convolutional Neural Network (CNN). The use of CNNs for feature extraction enhanced the representation of image data, enabling the GMM to achieve meaningful classification results.

CODE

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Conv2D,
    MaxPooling2D,
    Flatten,
    Dense,
    BatchNormalization,
    Dropout,
    Input,
)
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.decomposition import PCA
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt
from scipy.stats import mode
```

Loading and Preprocessing the Dataset

We first load the CIFAR-10 dataset and normalize it to the range [0,1].

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.
```

Building the CNN Model

A CNN is used to extract meaningful features from the images. We use a functional API approach to define the model.

```
input_layer = Input(shape=(32, 32, 3))
x = Conv2D(32, (3, 3), padding="same", activation="relu")(input_layer)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)
x = Conv2D(64, (3, 3), padding="same", activation="relu")(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)
x = Flatten()(x)
feature_layer = Dense(256, activation="relu")(x)  # Feature Extraction Layer
x = Dropout(0.5)(feature_layer)
output_layer = Dense(10, activation="softmax")(x)
```

Compiling and Training the Model

The model is compiled using Adam optimizer and trained using categorical cross-entropy loss.

```
cnn_model = Model(inputs=input_layer, outputs=output_layer)
# Training callbacks
early_stopping = EarlyStopping(monitor="val_loss", patience=3,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.2, patience=2
cnn_model.compile(optimizer="adam", loss="sparse_categorical_crossentropy",
metrics=["accuracy"])
history = cnn_model.fit(
    x_train,
    y_train,
    epochs=15,
    batch_size=128,
    validation_data=(x_test, y_test),
    callbacks=[early_stopping, reduce_lr],
)
```

Feature Extraction

We extract features from the trained CNN model using the feature layer before the final classification layer.

```
feature_extractor = Model(inputs=cnn_model.input, outputs=feature_layer)

# Extracting Features
train_features = feature_extractor.predict(x_train)
test_features = feature_extractor.predict(x_test)
print("Extracted feature shape:", train_features.shape)
```

Applying PCA for Dimensionality Reduction

To reduce the complexity of feature vectors, we apply PCA to reduce the dimensionality to 50 components.

```
pca = PCA(n_components=50)
train_pca = pca.fit_transform(train_features)
test_pca = pca.transform(test_features)
```

Clustering with Gaussian Mixture Model (GMM)

We apply a GMM to cluster the image features into 10 groups.

```
gmm = GaussianMixture(n_components=10, covariance_type="full", random_state=42)
gmm.fit(train_pca)
gmm_preds = gmm.predict(test_pca)
print("GMM predicted clusters:", np.unique(gmm_preds))
```

Mapping GMM Clusters to True Labels

```
def map_gmm_to_labels(gmm_preds, y_true):
    labels = np.zeros_like(gmm_preds)
    for i in range(10):  # 10 clusters
        mask = gmm_preds == i
        if np.sum(mask) > 0:
            labels[mask] = mode(y_true[mask], keepdims=True)[0][0]
    return labels.astype(int)

gmm_labels = map_gmm_to_labels(gmm_preds, y_test)

# Compute clustering accuracy
accuracy = np.mean(gmm_labels == y_test.flatten())
print(f"GMM Classification Accuracy: {accuracy * 100:.2f}%")
```

Visualization

```
def visualize_cluster_grid(x_test, gmm_preds, num_clusters=10,
samples_per_cluster=5):
    fig, axes = plt.subplots(num_clusters, samples_per_cluster, figsize=(15, 20))
    for cluster in range(num_clusters):
        cluster_indices = np.where(gmm_preds == cluster)[0]
        if len(cluster_indices) == 0:
            for idx in range(samples_per_cluster):
                axes[cluster, idx].axis("off")
            continue
        sample_indices = np.random.choice(
            cluster_indices, min(samples_per_cluster, len(cluster_indices)),
replace=False
        )
        for idx in range(samples_per_cluster):
            ax = axes[cluster, idx]
            if idx < len(sample_indices):
                ax.imshow(x_test[sample_indices[idx]])
                ax.set_xticks([])
                ax.set_yticks([])
            else:
                ax.axis("off")
            if idx == 0:
                ax.set_title(f"Cluster {cluster}", fontsize=12)
    plt.tight_layout()
    plt.savefig("cluster_grid.png", bbox_inches="tight", dpi=300)
    plt.show()

visualize_cluster_grid(x_test, gmm_preds)
```

Confusion Matrix

```
cm = confusion_matrix(y_test.flatten(), gmm_labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap="Blues", values_format="d")
plt.title("Confusion Matrix", fontsize=14)
plt.show()
```