AI Capstone HW1 Report

109652030 周子翔

Dataset web link:

https://www.kaggle.com/datasets/kaggleashwin/vehicle-type-recognition

GitHub link:

https://github.com/sean41880/AI-Capstone

Brief statement of my research question:

In this project, I use three different model to classify four different kind of vehicles images. First I separate the data to train and test data. And then, I use the train data to train three different model. There are cnn, k-mean cluster, and svm. After that I compute the confusion matrix, AUC-ROC, and accuracy.

Documentation of my dataset:

The dataset is organized into four classes, each representing a different type of vehicle:

Car: Images of different car models and types, captured from various angles and under different lighting conditions.

Truck: Images of different types of trucks, including pickup trucks, delivery trucks, and heavy-duty trucks.

Bus: Images of buses used for public transportation, school buses, and other types of buses.

Motorcycle: Images of motorcycles and motorbikes.

Description of the Supervised and Unsupervised Methods Used

1. Supervised Learning Methods

I. Support Vector Machine (SVM)

- **Overview:**
  Support Vector Machine (SVM) is a supervised learning algorithm used

for classification tasks. It works by finding the hyperplane that best separates the data points of different classes in the feature space.

- **Implementation:**
  - **Feature Extraction:**
    HOG (Histogram of Oriented Gradients) features are extracted from images to capture the shape and structure information.
  - **Data Loading:**
    Training and validation data are loaded from specified directories.
  - **Model Training:**
    An SVM model with a linear kernel is trained using the combined training and validation data.
  - **Model Evaluation:**
    The trained model is evaluated on the test data to calculate accuracy, generate a confusion matrix, and print a classification report.
- **Libraries Used:**
  - `skimage.feature` for HOG feature extraction.
  - `sklearn.svm` for the SVM model.
  - `sklearn.metrics` for evaluation metrics.
  - `joblib` for saving and loading the model.

II. Convolutional Neural Network (CNN)

- **Overview:**
  Convolutional Neural Networks (CNNs) are deep learning models specifically designed for image classification tasks. They use convolutional layers to automatically learn spatial hierarchies of features from input images.
- **Implementation:**
  - **Data Loading:**
    Image data is loaded from the specified directory.
  - **Model Definition:**
    A CNN model is defined using MobileNetV2 as the base model, with additional fully connected layers for classification.
  - **Cross-Validation:**
    5-fold cross-validation is performed to evaluate the model. The

AUC-ROC score is calculated for each fold, and the best model based on AUC-ROC is saved.
    - **Model Evaluation:**
      The best model is evaluated on the test data to calculate accuracy, generate a confusion matrix, and print a classification report.
- **Libraries Used:**
    - `tensorflow.keras` for building and training the CNN model.
    - `sklearn.model_selection` for cross-validation.
    - `sklearn.metrics` for evaluation metrics.

2. Unsupervised Learning Methods

K-Means Clustering

- **Overview:**
  K-Means is an unsupervised learning algorithm used for clustering tasks. It partitions the data into K clusters by minimizing the variance within each cluster.
- **Implementation:**
    - **Feature Extraction:**
      HOG features are extracted from images to capture the shape and structure information.
    - **Data Loading:**
      Training data is loaded from the specified directory.
    - **Model Training:**
      A K-Means model is trained using the extracted HOG features.
    - **Model Evaluation:**
      The trained model is evaluated on the test data to calculate accuracy, generate a confusion matrix, and print a classification report.
- **Libraries Used:**
    - `skimage.feature` for HOG feature extraction.
    - `sklearn.cluster` for the K-Means model.
    - `sklearn.metrics` for evaluation metrics.
    - `joblib` for saving and loading the model.

3. Other Methods Used

Feature Extraction

- **HOG (Histogram of Oriented Gradients):**
  HOG is used to extract features from images by capturing the gradient structure that characterizes local object appearance and shape.

Data Resampling

- **Cross-Validation:**
  5-fold cross-validation is used to evaluate the CNN model. This technique helps in assessing the model's performance and robustness by training and validating it on different subsets of the data.

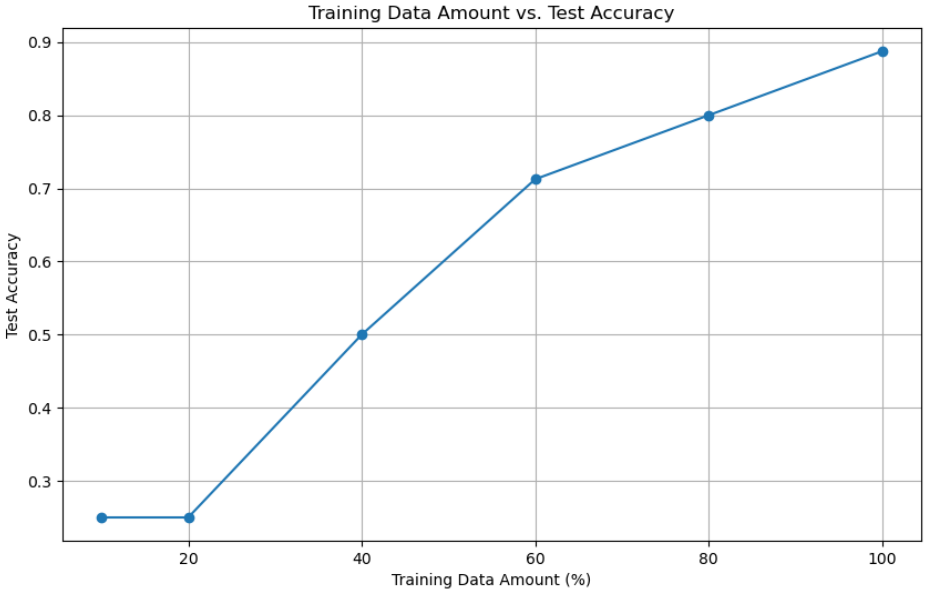
Dimensionality Reduction

- **Normalization:**
  `StandardScaler` is used to normalize the HOG features before training the K-Means model. This ensures that all features contribute equally to the clustering process.

Description of your experiments:

1. **Experiments on different aspects related to the datasets**
    1. **Data amount**

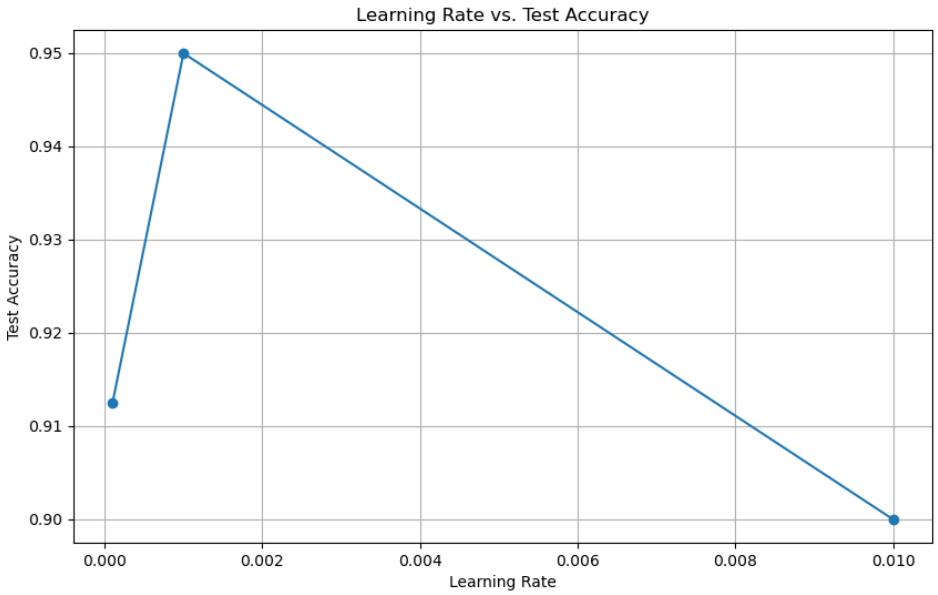
       In this part, a test the result affected by the amount of data. I try to use 20, 40, 60, 80, 100 percent of data to train the cnn model. The result tells that the more data I have the more accurate the model is.



2. Hyper-parameters

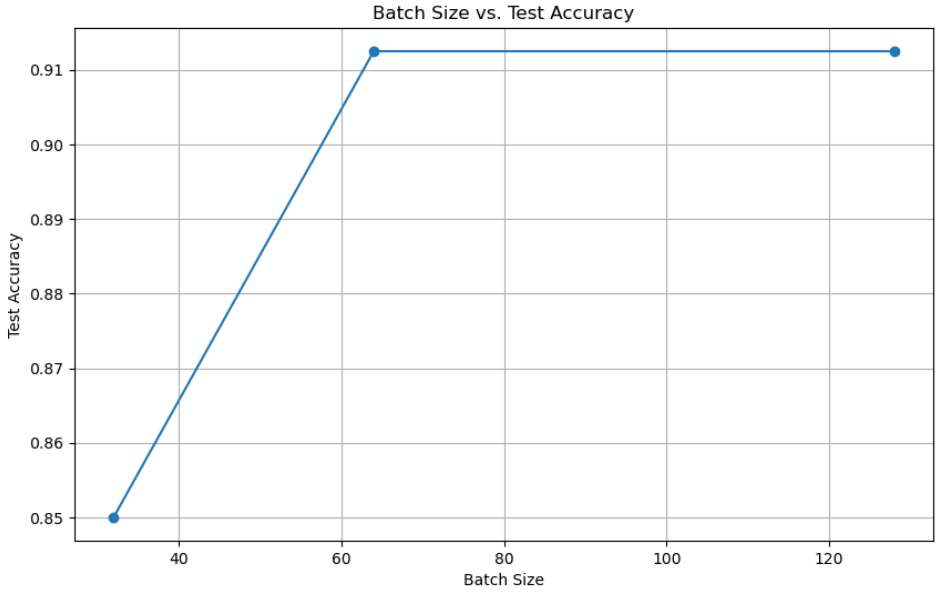
- **Learning rate**

  In this part, I test how the learning rate affects the results. I think that learning rate is strongly related to your own model. And need to find the best one by trying different learning rate.



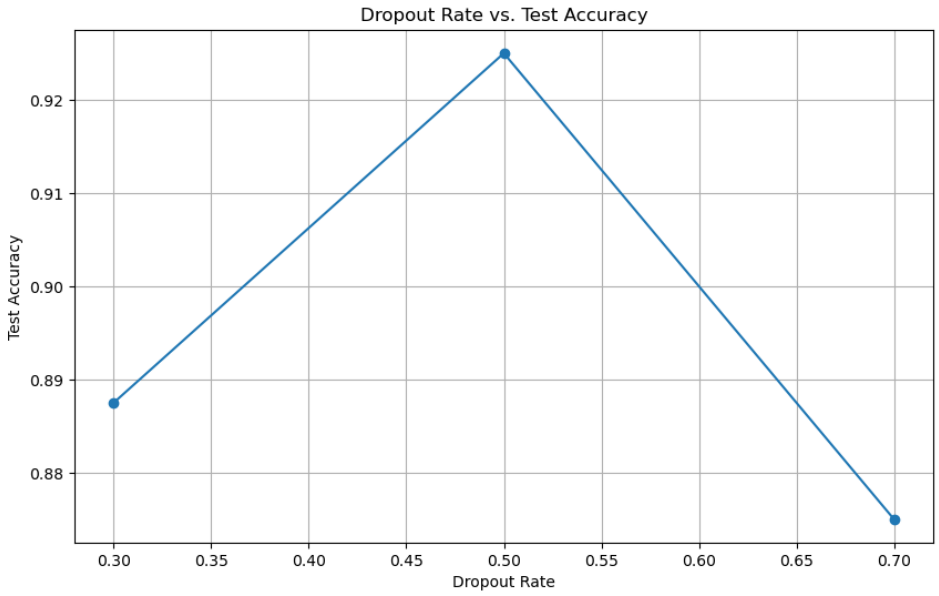
- **Batch size**

  In this part, I test how the batch size affects the results. In my case, bigger is better.



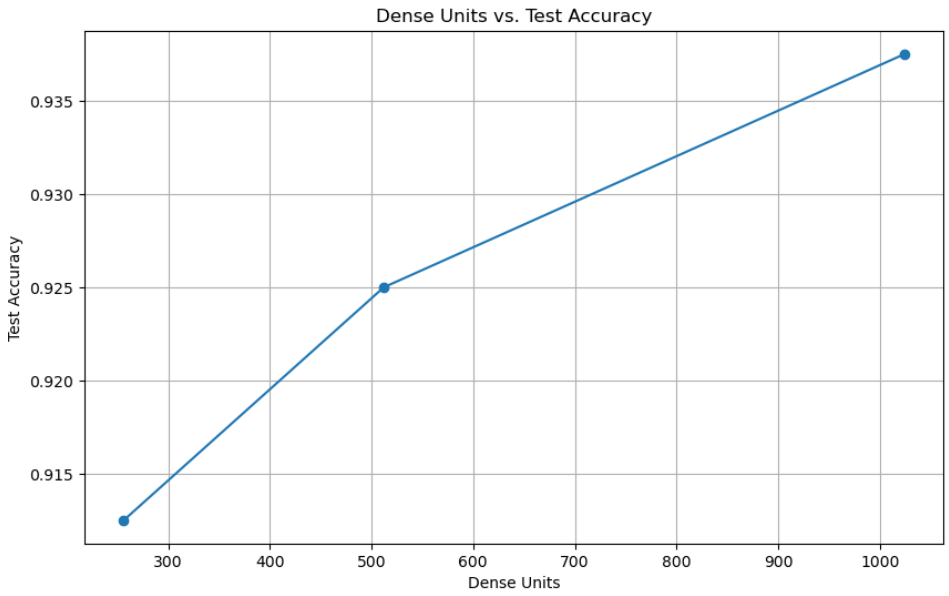
- **Dropout rate:**

  In this part, I test how the batch size affects the results. I think every model have its own best dropout rate.



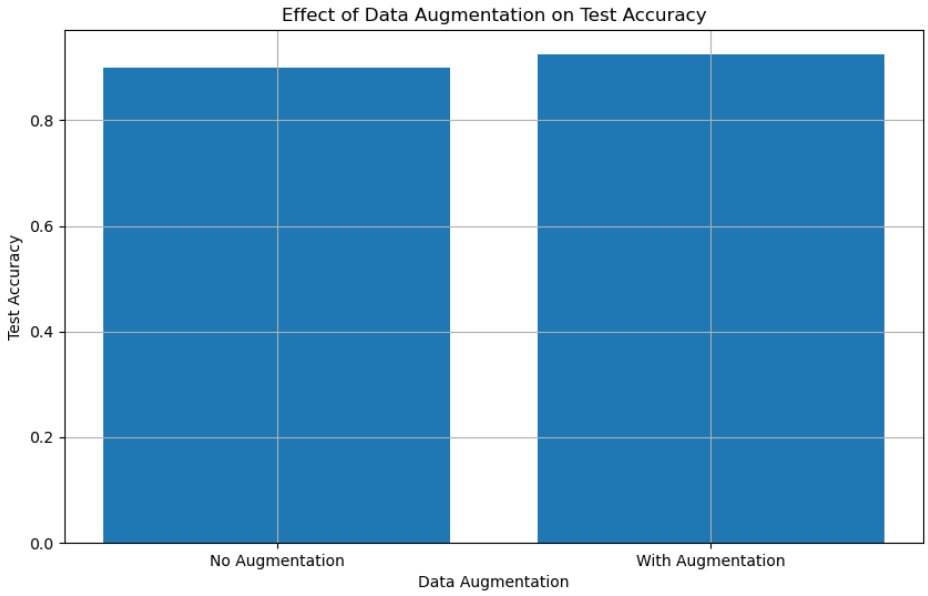
- **Dense units list**

  In this part, I test how the dense units list affects the results. In my case, higher is better.



3. **Data augmentation**

   In this part, I compare the results with and without data augmentation. The augmentation I did is shear, zoom, and Horizontal flip. The model with augmentation is better.



2. **Evaluate the performance of three models**

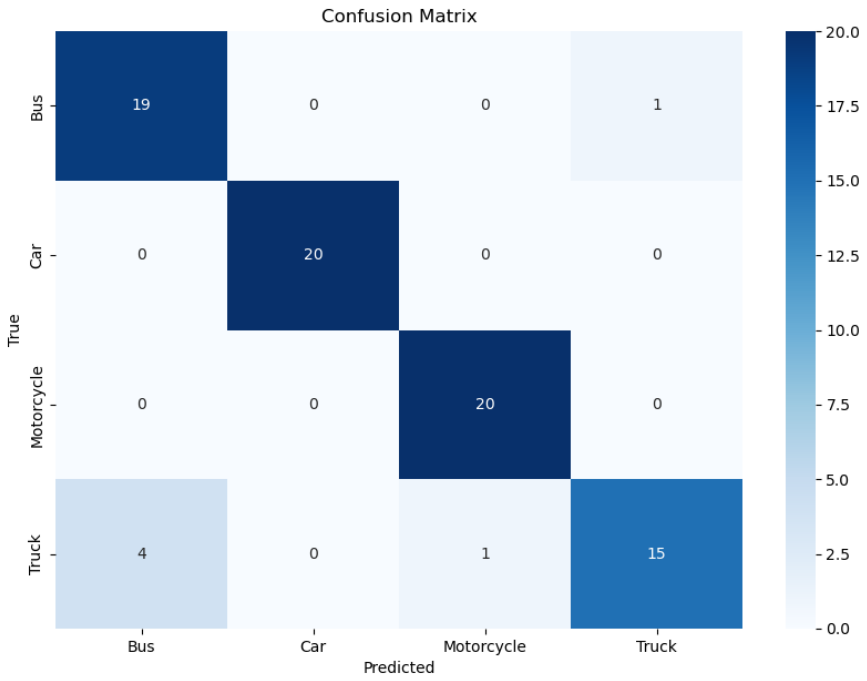
   I use cross validation and the AUC-ROC score to Evaluate the performance of two models and see which model is better to classify vehicle.

   - **CNN**

In the cnn model, I use cross validation to get the best model by the AUC-ROC score.

```
AUC-ROC scores: [0.44559733072916663, 0.47916666666666663, 0.4951985677083333, 0.519775390625, 0.4647216796875]
```

With the best model, I get 0.93 as the accuracy and this is the confusion matrix:



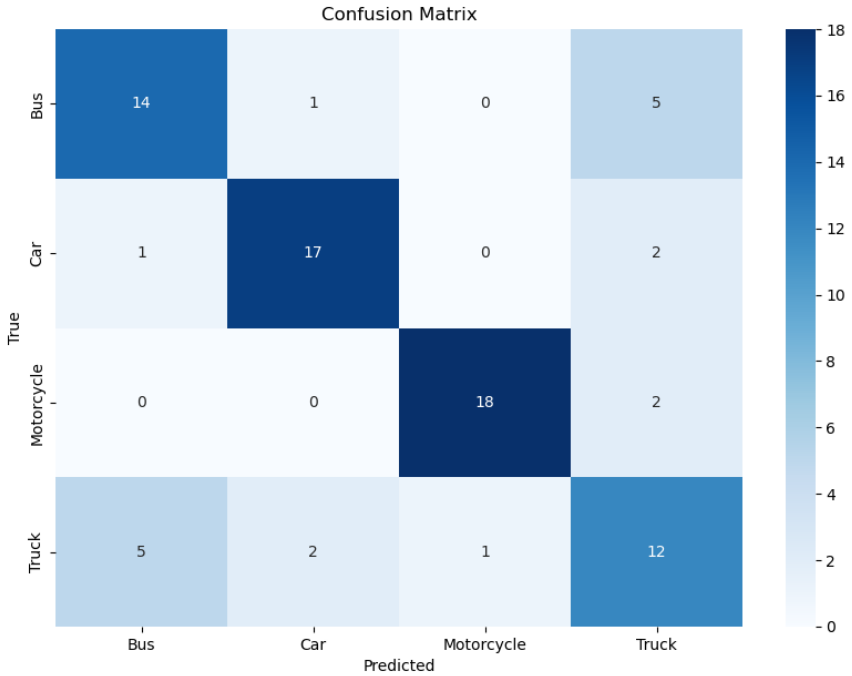
- **SVM model:**

  I also use cross validation to svm and this is the result:

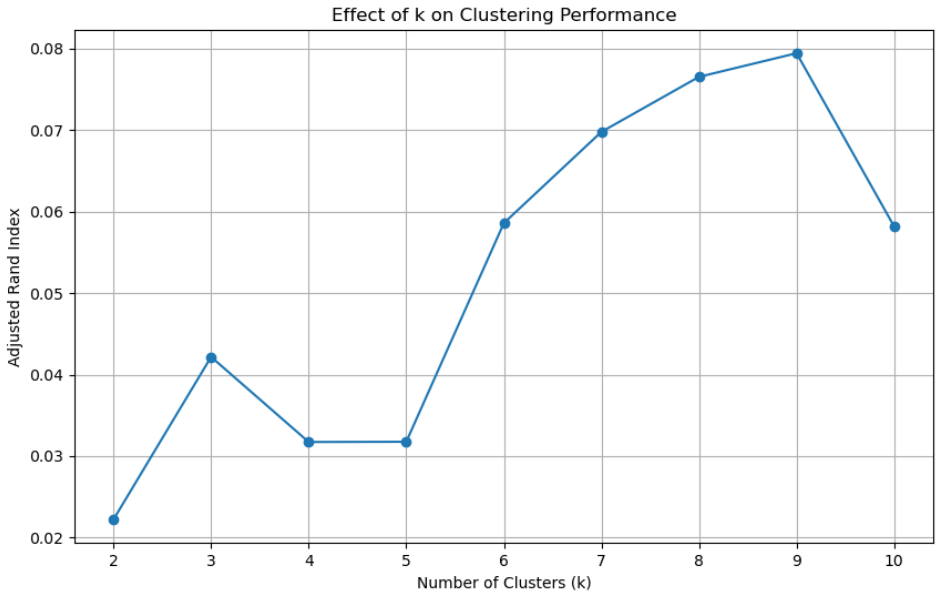
```
AUC-ROC scores: [0.9072265625, 0.9189453125, 0.9020182291666667, 0.9104817708333334, 0.9150390625000
Mean AUC-ROC: 0.91
```

Which can tell that svm did a better job in classification than cnn model.But the test accuracy is only 0.76. which tell the model may be overfitting.

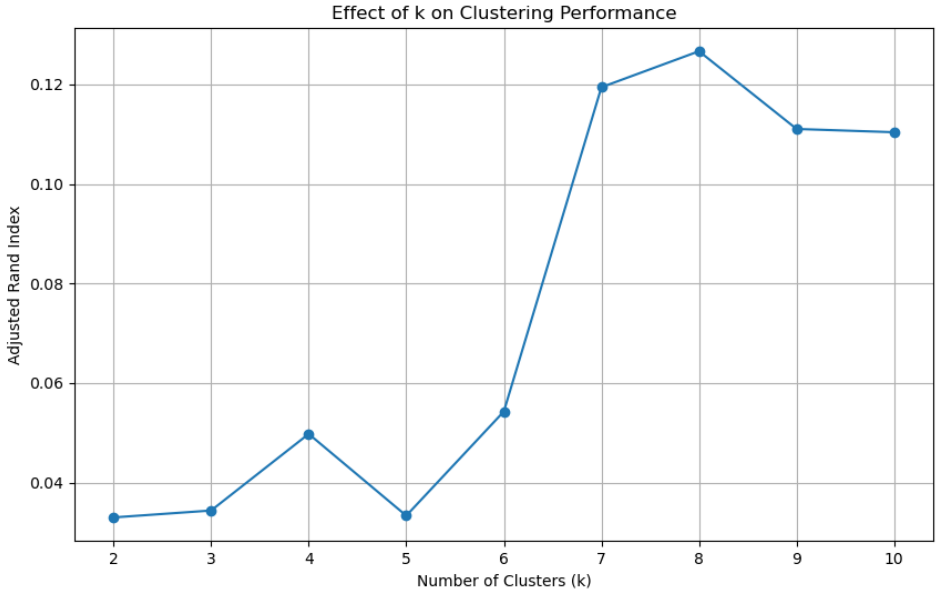
The confusion matrix:



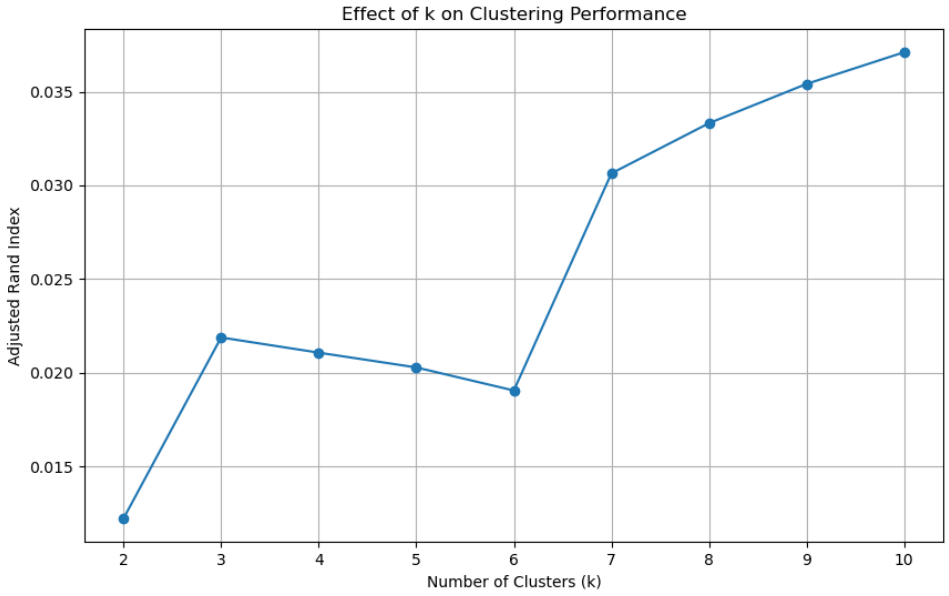
- **k-means cluster**
    1. without PCA



    2. With PCA:



3. With data augmentation and PCA:



The k-mean cluster don't perform good. After I add PCA, the performance is better a little bit but still bad. I think maybe its because the train data is not large enough. So I use augmentation to add more train data. After adding data augmentation, the result is still very bad.

But even though, I found that the more number of clusters, the better ARI score the model get.

Code:

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import MobileNetV2
from sklearn.metrics import accuracy_score
import numpy as np
import matplotlib.pyplot as plt
import os

# Define paths
dataset_dir = 'Dataset/train'
dataset_dir_test = 'Dataset/test'
class_labels = ['Bus', 'Car', 'Motorcycle', 'Truck']

# Image data generator
datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

datagen_without_augmentation = ImageDataGenerator(
    rescale=1./255
)

# Load data
def load_data(directory):
    data = []
    labels = []
    for label in class_labels:
        class_path = os.path.join(directory, label)
        for img_name in os.listdir(class_path):
            img_path = os.path.join(class_path, img_name)
            img = tf.keras.preprocessing.image.load_img(img_path, target_size=(150, 150))
            img_array = tf.keras.preprocessing.image.img_to_array(img)
            data.append(img_array)
            labels.append(class_labels.index(label))
    return np.array(data), np.array(labels)

X_train, y_train = load_data(dataset_dir)
X_test, y_test = load_data(dataset_dir_test)

# Convert labels to categorical
y_train_cat = tf.keras.utils.to_categorical(y_train, num_classes=4)
y_test_cat = tf.keras.utils.to_categorical(y_test, num_classes=4)

# Define the model with adjustable hyper-parameters
def create_model(learning_rate=0.001, dropout_rate=0.5, dense_units=512):
    base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
    base_model.trainable = False  # Freeze the pre-trained layers

    model = Sequential([
        base_model,
        Flatten(),
        Dense(dense_units, activation='relu'),
        Dropout(dropout_rate),
        Dense(4, activation='softmax')
    ])

    model.compile(optimizer=Adam(learning_rate=learning_rate), loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

```
import kagglehub
from sklearn.model_selection import train_test_split
import pandas as pd
import os
import shutil

'''
# Download the dataset
kagglehub.dataset_download("kaggleashwin/vehicle-type-recognition")
'''

dataset_path = 'Dataset'
categories = ['Bus', 'Car', 'Motorcycle', 'Truck']

# Create train and test directories
for category in categories:
    os.makedirs(os.path.join(dataset_path, 'train', category), exist_ok=True)
    os.makedirs(os.path.join(dataset_path, 'test', category), exist_ok=True)

# Split the dataset
for category in categories:
    category_path = os.path.join(dataset_path, category)
    images = os.listdir(category_path)
    train_images, test_images = train_test_split(images, test_size=0.2, random_state=3)

    for image in train_images:
        shutil.copy(os.path.join(category_path, image), os.path.join(dataset_path, 'train', category, image))

    for image in test_images:
        shutil.copy(os.path.join(category_path, image), os.path.join(dataset_path, 'test', category, image))

print("Dataset split into training and testing sets completed.")
```

```
from skimage.feature import hog
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score
from sklearn.model_selection import StratifiedKFold
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib

# Define paths and class labels
train_dir = 'Dataset/train'
class_labels = ['Bus', 'Car', 'Motorcycle', 'Truck']

# Function to extract HOG features
def extract_hog_features(img_path):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)  # Read image in grayscale
    img = cv2.resize(img, (150, 150))  # Resize image
    features, _ = hog(img, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=True)
    return features

# Load data from a directory
def load_data(directory):
    data = []
    labels = []
    for label in class_labels:
        class_path = os.path.join(directory, label)
        for img_name in os.listdir(class_path):
            img_path = os.path.join(class_path, img_name)
            features = extract_hog_features(img_path)
            data.append(features)
            labels.append(class_labels.index(label))
    return np.array(data), np.array(labels)

# Load training data
X, y = load_data(train_dir)

# Perform cross-validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
auc_scores = []
best_auc = 0
best_model = None

for fold, (train_index, val_index) in enumerate(skf.split(X, y)):
    X_train, X_val = X[train_index], X[val_index]
    y_train, y_val = y[train_index], y[val_index]

    # Train SVM model
    svm_model = SVC(kernel='linear', probability=True)
    svm_model.fit(X_train, y_train)

    # Predict probabilities
    y_val_pred_prob = svm_model.predict_proba(X_val)

    # Calculate AUC-ROC
    auc = roc_auc_score(y_val, y_val_pred_prob, multi_class='ovr')
    auc_scores.append(auc)

    # Save the model if it has the best AUC-ROC score
    if auc > best_auc:
        best_auc = auc
        best_model = svm_model
        joblib.dump(svm_model, f'best_svm_model_fold_{fold}.pkl')

# Print AUC-ROC scores
print(f"AUC-ROC scores: {auc_scores}")
print(f"Mean AUC-ROC: {np.mean(auc_scores):.2f}")

# Save the best model
if best_model:
    joblib.dump(best_model, 'best_vehicle_classification_svm_model.pkl')
    print("Best model saved as 'best_vehicle_classification_svm_model.pkl'")
```

```
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
from sklearn.decomposition import PCA
import cv2
import os
import numpy as np
import joblib
from skimage.feature import hog
import matplotlib.pyplot as plt
import imgaug.augmenters as iaa

# Define paths and class labels
train_dir = 'Dataset/train'
class_labels = ['Bus', 'Car', 'Motorcycle', 'Truck']

# Define data augmentation pipeline
augmentation_pipeline = iaa.Sequential([
    iaa.Fliplr(0.5),  # Horizontal flip
    iaa.Affine(rotate=(-20, 20)),  # Rotate
    iaa.Affine(scale=(0.8, 1.2)),  # Scale
    iaa.AdditiveGaussianNoise(scale=(0, 0.05*255)),  # Add Gaussian noise
    iaa.Multiply((0.8, 1.2))  # Change brightness
])

# Function to extract HOG features
def extract_hog_features(img):
    img = cv2.resize(img, (150, 150))  # Resize image
    features, _ = hog(img, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=True)
    return features

# Read images, apply data augmentation, and extract HOG features
data = []
true_labels = []

for label in class_labels:
    class_path = os.path.join(train_dir, label)
    for img_name in os.listdir(class_path):
        img_path = os.path.join(class_path, img_name)
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)  # Read image in grayscale
        if img is None:
            print(f"Warning: Unable to read image {img_path}")
            continue
        # Original image
        features = extract_hog_features(img)
        data.append(features)
        true_labels.append(class_labels.index(label))
        # Augmented images
        for _ in range(5):  # Generate 5 augmented images per original image
            augmented_img = augmentation_pipeline(image=img)
            features = extract_hog_features(augmented_img)
            data.append(features)
            true_labels.append(class_labels.index(label))

# Convert to NumPy array
data = np.array(data)
true_labels = np.array(true_labels)

# Normalize the data
data = (data - np.mean(data, axis=0)) / np.std(data, axis=0)

# Apply PCA for dimensionality reduction
pca = PCA(n_components=50)  # Adjust the number of components as needed
data_pca = pca.fit_transform(data)

# Define a range of k values to test
k_values = range(2, 11)  # Testing k values from 2 to 10
ari_scores = []

# Perform K-Means clustering for each k value and evaluate performance
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=48)
    predicted_labels = kmeans.fit_predict(data_pca)
    adjusted_rand = adjusted_rand_score(true_labels, predicted_labels)
    ari_scores.append(adjusted_rand)
    print(f"k: {k}, Adjusted Rand Index: {adjusted_rand:.2f}")

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(k_values, ari_scores, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Adjusted Rand Index')
plt.title('Effect of k on Clustering Performance')
plt.grid(True)
plt.savefig('/mnt/data/code_screenshot.jpg')
```

```
import kagglehub
from sklearn.model_selection import train_test_split
import pandas as pd
import os
import shutil

'''
# Download the dataset
kagglehub.dataset_download("kaggleashwin/vehicle-type-recognition")
'''

dataset_path = 'Dataset'
categories = ['Bus', 'Car', 'Motorcycle', 'Truck']

# Create train and test directories
for category in categories:
    os.makedirs(os.path.join(dataset_path, 'train', category), exist_ok=True)
    os.makedirs(os.path.join(dataset_path, 'test', category), exist_ok=True)

# Split the dataset
for category in categories:
    category_path = os.path.join(dataset_path, category)
    images = os.listdir(category_path)
    train_images, test_images = train_test_split(images, test_size=0.2, random_state=3)

    for image in train_images:
        shutil.copy(os.path.join(category_path, image), os.path.join(dataset_path, 'train', category, image))

    for image in test_images:
        shutil.copy(os.path.join(category_path, image), os.path.join(dataset_path, 'test', category, image))

print("Dataset split into training and testing sets completed.")

from skimage.feature import hog
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score
from sklearn.model_selection import StratifiedKFold
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib

# Define paths and class labels
train_dir = 'Dataset/train'
class_labels = ['Bus', 'Car', 'Motorcycle', 'Truck']

# Function to extract HOG features
def extract_hog_features(img_path):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)  # Read image in grayscale
    img = cv2.resize(img, (150, 150))  # Resize image
    features, _ = hog(img, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=True)
    return features

# Load data from a directory
def load_data(directory):
    data = []
    labels = []
    for label in class_labels:
        class_path = os.path.join(directory, label)
        for img_name in os.listdir(class_path):
            img_path = os.path.join(class_path, img_name)
            features = extract_hog_features(img_path)
            data.append(features)
            labels.append(class_labels.index(label))
    return np.array(data), np.array(labels)

# Load training data
X, y = load_data(train_dir)

# Perform cross-validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
auc_scores = []
best_auc = 0
best_model = None

for fold, (train_index, val_index) in enumerate(skf.split(X, y)):
    X_train, X_val = X[train_index], X[val_index]
    y_train, y_val = y[train_index], y[val_index]

    # Train SVM model
    svm_model = SVC(kernel='linear', probability=True)
    svm_model.fit(X_train, y_train)

    # Predict probabilities
    y_val_pred_prob = svm_model.predict_proba(X_val)

    # Calculate AUC-ROC
    auc = roc_auc_score(y_val, y_val_pred_prob, multi_class='ovr')
    auc_scores.append(auc)

    # Save the model if it has the best AUC-ROC score
    if auc > best_auc:
        best_auc = auc
        best_model = svm_model
        joblib.dump(svm_model, f'best_svm_model_fold_{fold}.pkl')

# Print AUC-ROC scores
print(f"AUC-ROC scores: {auc_scores}")
print(f"Mean AUC-ROC: {np.mean(auc_scores):.2f}")

# Save the best model
if best_model:
    joblib.dump(best_model, 'best_vehicle_classification_svm_model.pkl')
    print("Best model saved as 'best_vehicle_classification_svm_model.pkl'")

from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
from sklearn.decomposition import PCA
import cv2
import os
import numpy as np
import joblib
from skimage.feature import hog
import matplotlib.pyplot as plt
import imgaug.augmenters as iaa

# Define paths and class labels
train_dir = 'Dataset/train'
class_labels = ['Bus', 'Car', 'Motorcycle', 'Truck']

# Define data augmentation pipeline
augmentation_pipeline = iaa.Sequential([
    iaa.Fliplr(0.5),  # Horizontal flip
    iaa.Affine(rotate=(-20, 20)),  # Rotate
    iaa.Affine(scale={0.8, 1.2}),  # Scale
    iaa.AdditiveGaussianNoise(scale=(0, 0.05*255)),  # Add Gaussian noise
    iaa.Multiply((0.8, 1.2))  # Change brightness
])

# Function to extract HOG features
def extract_hog_features(img):
    img = cv2.resize(img, (150, 150))  # Resize image
    features, _ = hog(img, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=True)
    return features

# Read images, apply data augmentation, and extract HOG features
data = []
true_labels = []

for label in class_labels:
    class_path = os.path.join(train_dir, label)
    for img_name in os.listdir(class_path):
        img_path = os.path.join(class_path, img_name)
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)  # Read image in grayscale
        if img is None:
            print(f"Warning: Unable to read image {img_path}")
            continue
        # Original image
        features = extract_hog_features(img)
        data.append(features)
        true_labels.append(class_labels.index(label))
        # Augmented images
        for _ in range(5):  # Generate 5 augmented images per original image
            augmented_img = augmentation_pipeline(image=img)
            features = extract_hog_features(augmented_img)
            data.append(features)
            true_labels.append(class_labels.index(label))

# Convert to NumPy array
data = np.array(data)
true_labels = np.array(true_labels)

# Normalize the data
data = (data - np.mean(data, axis=0)) / np.std(data, axis=0)

# Apply PCA for dimensionality reduction
pca = PCA(n_components=50)  # Adjust the number of components as needed
data_pca = pca.fit_transform(data)

# Define a range of k values to test
k_values = range(2, 11)  # Testing k values from 2 to 10
ari_scores = []

# Perform K-Means clustering for each k value and evaluate performance
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=48)
    predicted_labels = kmeans.fit_predict(data_pca)
    adjusted_rand = adjusted_rand_score(true_labels, predicted_labels)
    ari_scores.append(adjusted_rand)
    print(f"k: {k}, Adjusted Rand Index: {adjusted_rand:.2f}")

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(k_values, ari_scores, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Adjusted Rand Index')
plt.title('Effect of k on Clustering Performance')
plt.grid(True)
plt.savefig('/mnt/data/code screenshot.jpg')
```