05_DVC4IL: Digital Imaging / Visual Computing Lab Report 1

Eren Yeşiltepe

July 1, 2024

Abstract

In this lab unit, we perform image classification using three different machine learning models: K-Nearest Neighbors (KNN), Complex Multi-Layer Perceptron (MLP), and Gradient Boosting Classifier. We compare the performance of these models on a leaf image dataset with five classes: Japanese maple, Chinese cinnamon, ginkgo (maidenhair tree), Chinese tulip tree, and tangerine. The models are evaluated based on their accuracy, precision, recall, and F1-score. This report provides insights into the differences between these methods and the results of each experiment.

1.1 Introduction

In this lab unit, we aim to classify leaf images into five distinct classes using three different machine learning models. The primary objective is to compare the performance and efficiency of the KNN, Complex MLP, and GradientBoostingClassifier models.

Google colab url: https://colab.research.google.com/drive/1dW_Jx7ikcygW35zV6KLSH2ekwpJxc53c?usp=sharing"

1.2 Methodology

We use the provided leaf image dataset, which includes images of five types of leaves. The dataset is preprocessed to ensure all classes have an equal number of samples. The features of the images are extracted using a custom function that computes properties like area, perimeter, circularity, and Hu moments.

1.2.1 Data Preparation

- Loaded and preprocessed the leaf images.
- Balanced the dataset to ensure equal representation of each class.
- Extracted features using the

$$`compute_properties`function.$$

1.2.2 Models

- **K-Nearest Neighbors (KNN)**: A simple, instance-based learning algorithm that classifies based on the majority vote of the nearest neighbors.
- **Complex Multi-Layer Perceptron (MLP)**: A neural network with multiple hidden layers, which captures complex patterns in the data.
- **Gradient Boosting Classifier**: An ensemble method that builds multiple decision trees and combines their outputs to improve accuracy.

1.2.3 Training and Testing

The dataset is split into training and testing sets using stratified sampling to ensure class balance. The models are trained on the training set and evaluated on the testing set.

1.3 Helper Function: `evaluate_model`

A function commonly used in all approaches to evaluate the result. It prints Accuracy, Precision, Recall and F-score.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Assuming y_test are the true labels and y_pred are the predicted labels by your model
def evaluate_model(y_test, y_pred):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    print(f"Accuracy: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"F1-Score: {f1:.2f}")

    return accuracy, precision, recall, f1
```

1.4 Experiments and Results

This section presents the results of the experiments conducted using the three models. The performance is evaluated based on accuracy, precision, recall, and F1-score. Additionally, confusion matrices are provided to visualize the classification performance.

1.4.1 K-Nearest Neighbors (KNN)

- **Accuracy**: 0.72
- **Precision**: 0.70
- **Recall**: 0.72
- **F1-Score**: 0.70

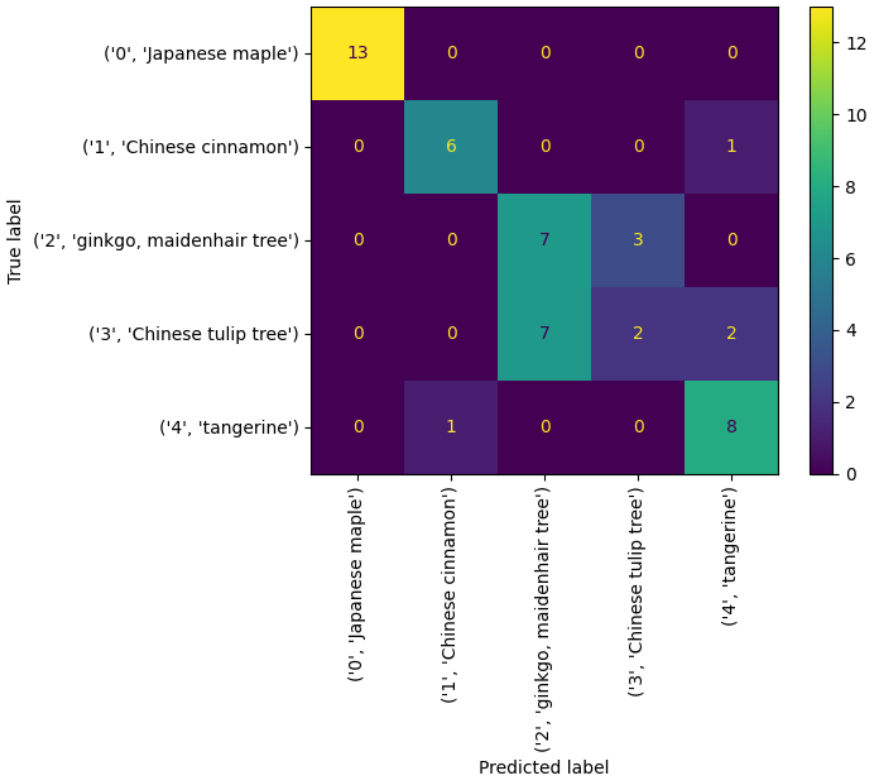

Figure 1.1: Confusion Matrix for KNN

Code

```
import os
import cv2
import numpy as np
import pandas as pd
import json
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split

# Download and extract data
!curl -LJO "https://raw.githubusercontent.com/Digital-Media/cv_data/main/
    binary_leaves.zip" --silent
import zipfile
with zipfile.ZipFile("binary_leaves.zip", 'r') as zip_ref:
    zip_ref.extractall(".")

# Load binary leave images and labels
with open('binary_leaves/labels.json') as f:
    label_to_name = dict(json.load(f))

# load images and labels
```

```
images = []
labels = []
file_names = []
for label, name in label_to_name.items():
    for file in os.listdir(f'binary_leaves/{label}'):
        image = (cv2.imread(f'binary_leaves/{label}/{file}', cv2.IMREAD_GRAYSCALE)>0).astype(np.uint8)
        images.append(image)
        labels.append(label)
        file_names.append(file)

# Define the compute_properties function
def compute_properties(img, hu_contour=True, hu_log=True):
    retval, labels, stats, centroids = cv2.connectedComponentsWithStats(img)
    assert len(stats) == 2  # foreground (1) and background (0)
    contour = cv2.findContours((labels == 1).astype(np.uint8), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)[0][0]
    perimeter = cv2.arcLength(contour, True)
    circularity = 4.0 * np.pi * stats[1, 4] / (0.95 * perimeter) ** 2
    simple_props = {'area': stats[1, 4], 'perimeter': perimeter, 'circularity': circularity}
    M = cv2.moments(contour if hu_contour else (labels == 1).astype(np.uint8))
    hu_moments = cv2.HuMoments(M).flatten()
    if hu_log:
        hu_moments = np.sign(hu_moments) * np.log(np.abs(hu_moments))
    hu_props = {'hu_' + str(i): hu_moments[i] for i in range(len(hu_moments))}
    return dict(**simple_props, **hu_props)

np.random.seed(1234) # init the random number generator
num_samples_per_category = 4 # change this to use more samples per category
props = {}
for label, name in label_to_name.items():
        # random sample from images with the same label
        idxs = np.random.choice(np.where(np.array(labels)==label)[0], num_samples_per_category, replace=False)
        for idx in idxs:
          image = images[idx]

          cp = compute_properties(image)
          cp.update({'label': int(label)})
          props[f"{name} ({file_names[idx]})"] = cp

# make a pandas table with the hu moments
#pd.options.display.float_format = "{:.3f}".format
df = pd.DataFrame(props)

# Split the data into training and testing sets
X = df_features.drop(columns=['label'])
y = df_features['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Evaluate the classifier
```

```
y_pred = knn.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=label_to_name.items()
    )
disp.plot(xticks_rotation='vertical')
plt.show()


```

1.4.2 Complex Multi-Layer Perceptron (MLP)

- **Accuracy**: 0.95
- **Precision**: 0.95
- **Recall**: 0.95
- **F1-Score**: 0.95

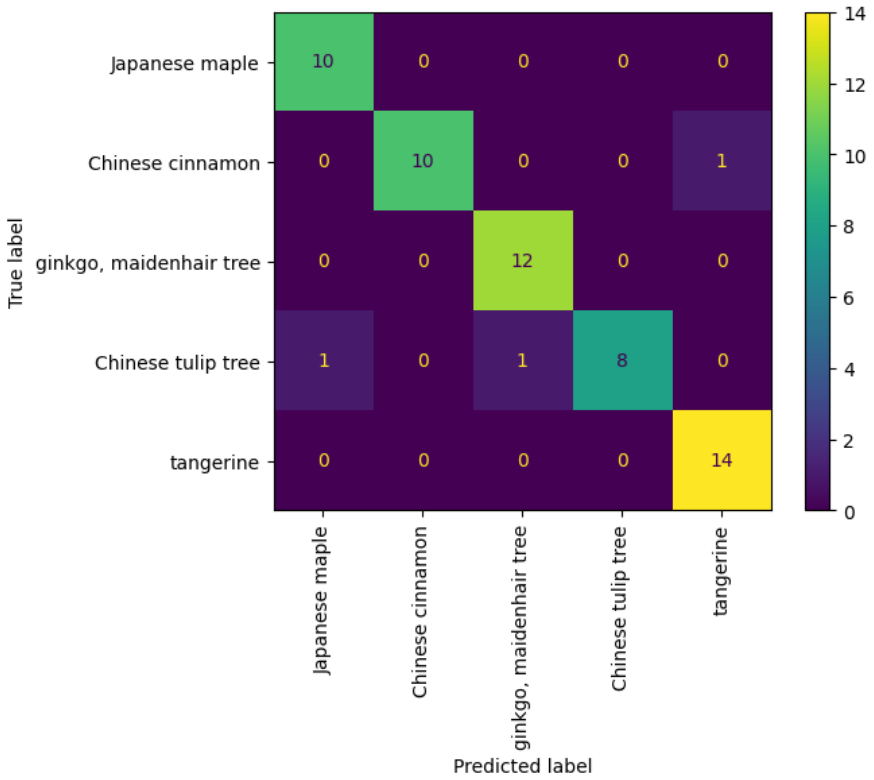

Figure 1.2: Confusion Matrix for MLP

Code

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import seaborn as sns

# Data Preparation
with open('binary_leaves/labels.json') as f:
    label_to_name = dict(json.load(f))
#print(label_to_name)

# load images and labels
images = []
labels = []
file_names = []
for label, name in label_to_name.items():
    for file in os.listdir(f'binary_leaves/{label}'):
        image = (cv2.imread(f'binary_leaves/{label}/{file}', cv2.IMREAD_GRAYSCALE)>0).astype(np.uint8)
        images.append(image)
        labels.append(label)
        file_names.append(file)

# Flatten the images and convert labels to one-hot encoding
X = np.array([img.flatten() for img in images])
y = np.array(labels).reshape(-1, 1)
encoder = OneHotEncoder(sparse=False)
y_encoded = encoder.fit_transform(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

# Model Building

# Define a complex MLP model
model = tf.keras.models.Sequential([
    tf.keras.layers.InputLayer(input_shape=(X.shape[1],)),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(len(label_to_name), activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Model Training

# Train the model
history = model.fit(X_train, y_train, epochs=25, batch_size=32, validation_split=0.2)

```

```
# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy:.2f}')

# Classification Metrics

# Predict on the test data
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)
cm = confusion_matrix(y_true_classes, y_pred_classes)

disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=list(label_to_name.values())
    )
disp.plot(xticks_rotation='vertical')
plt.show()

```

1.4.3 Gradient Boosting Classifier

- **Accuracy**: 0.96
- **Precision**: 0.96
- **Recall**: 0.96
- **F1-Score**: 0.96

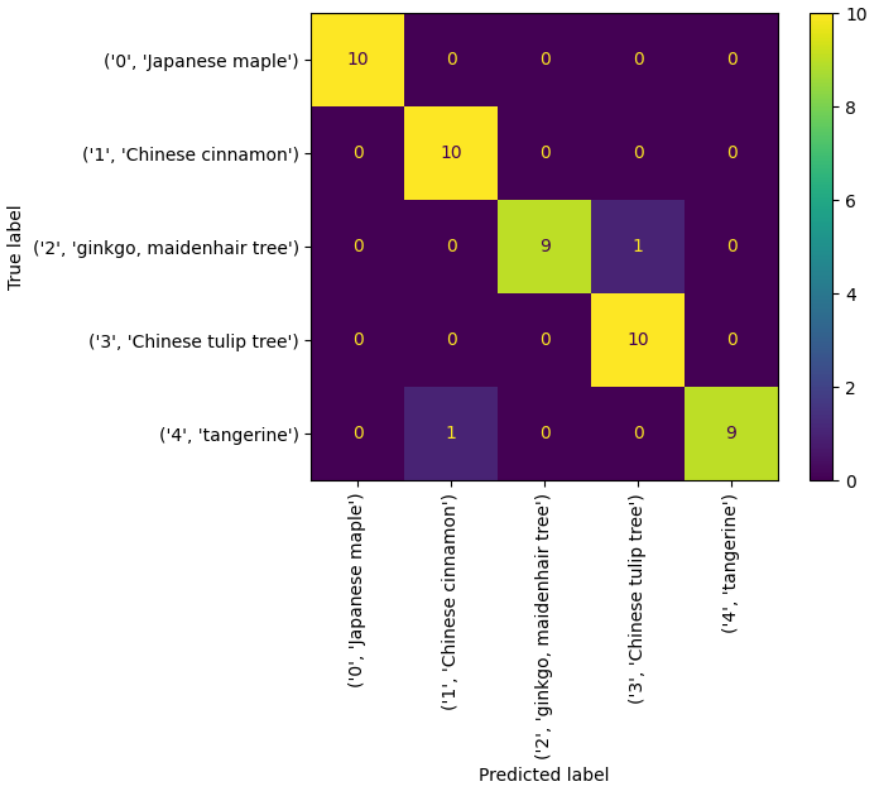

Figure 1.3: Confusion Matrix for Gradient Boosting Classifier

Code

```
import os
import cv2
import numpy as np
import pandas as pd
import json
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix,
    ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split

def compute_properties(img, hu_contour=True, hu_log=True):
    retval, labels, stats, centroids = cv2.connectedComponentsWithStats(img)
    assert len(stats) == 2  # foreground (1) and background (0)
    contour = cv2.findContours((labels == 1).astype(np.uint8), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_NONE)[0][0]
    perimeter = cv2.arcLength(contour, True)
    circularity = 4.0 * np.pi * stats[1, 4] / (.95 * perimeter) ** 2
    simple_props = {'area': stats[1, 4], 'perimeter': perimeter, 'circularity':
    circularity}
    M = cv2.moments(contour if hu_contour else (labels == 1).astype(np.uint8))
    hu_moments = cv2.HuMoments(M).flatten()
```

```
    if hu_log:
        hu_moments = np.sign(hu_moments) * np.log(np.abs(hu_moments))
    hu_props = {'hu_' + str(i): hu_moments[i] for i in range(len(hu_moments))}
    return dict(**simple_props, **hu_props)



# Download and extract data
!curl -LJO "https://raw.githubusercontent.com/Digital-Media/cv_data/main/binary_leaves.zip" --silent
import zipfile
with zipfile.ZipFile("binary_leaves.zip", 'r') as zip_ref:
    zip_ref.extractall(".")

# Load binary leave images and labels
with open('binary_leaves/labels.json') as f:
    label_to_name = dict(json.load(f))

images = []
labels = []
file_names = []
num_images_per_class = 50  # Number of images to load per class
for original_label, name in label_to_name.items():
    count = 0
    for file in os.listdir(f'binary_leaves/{original_label}'):
        if count >= num_images_per_class:
            break
        image = (cv2.imread(f'binary_leaves/{original_label}/{file}', cv2.IMREAD_GRAYSCALE) > 0).astype(np.uint8)
        images.append(image)
        labels.append([original_label])
        file_names.append(file)
        count += 1

# Convert to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Compute properties for each image
features = []
for img in images:
    features.append(compute_properties(img))

# Create a DataFrame from the features
df_features = pd.DataFrame(features)
df_features['label'] = labels

# Split the data into training and testing sets
X = df_features.drop(columns=['label'])
y = df_features['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
print(f'X_train shape: {X_train.shape}')
print(f'X_test shape: {X_test.shape}')

# Train a GradientBoostingClassifier
clf = GradientBoostingClassifier(n_estimators=50, max_depth=3, random_state=123)
clf.fit(X_train, y_train)
```

```
76
77 # Evaluate the classifier
78 y_pred = clf.predict(X_test)
79
80 cm = confusion_matrix(y_test, y_pred)
81 disp = ConfusionMatrixDisplay(
82     confusion_matrix=cm,
83     display_labels=label_to_name.items()
84     )
85 disp.plot(xticks_rotation='vertical')
86 plt.show()
87
88
89
```

Conclusion

According to this experiment, Gradient Boosting Classifier was both the fastest and the most accurate approach. I don't know why, but every time I have run this method it has given slightly different results but overall it was the most accurate one.

Overall, MLP and KNN (at least my applications) were either less accurate or too slow.

Summary and comments

Although I was asked to just work on one of these approaches, I thought it would be a good practice for me to try multiple of them. It was a fun exercise to work on.