Notebook

November 18, 2024

1 Import libraries

```python
import os
import sys
import cv2
import math
import json
import joblib
import nbformat
import numpy as np
import pandas as pd
import seaborn as sns
from tqdm import tqdm
from sklearn.svm import SVC
from datetime import datetime
import matplotlib.pyplot as plt
from nbconvert.exporters import PDFExporter
from skimage.feature import hog as skimage_hog
from sklearn.preprocessing import LabelEncoder
from IPython.display import display, Javascript
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
from scipy.spatial.distance import cityblock, cosine, correlation, sqeuclidean
```

2 Load data

```python
project_dir = os.getcwd()
project_dir = os.path.dirname(project_dir)
```

```python
width = 64
height = 64
```

```python
data_dir = project_dir + "\\data"

train_path = os.path.join(data_dir, "train")
```

```
test_path = os.path.join(data_dir, "test")

train_images = []
test_images = []

train_labels = []
test_labels = []

for path in (train_path, test_path):
    if (path.split('\\')[-1] == "train"):
        for dir in os.listdir(path):
            label_path = os.path.join(path, dir)
            label = dir.split('\\')[-1]
            for image in os.listdir(label_path):
                image_path = os.path.join(label_path, image)
                image = cv2.imread(image_path)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                image = cv2.resize(image, (width, height))
                train_images.append(image)
                train_labels.append(label)
    else:
        for dir in os.listdir(path):
            label_path = os.path.join(path, dir)
            label = dir.split('\\')[-1]
            for image in os.listdir(label_path):
                image_path = os.path.join(label_path, image)
                image = cv2.imread(image_path)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                image = cv2.resize(image, (width, height))
                test_images.append(image)
                test_labels.append(label)
```

```
label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(train_labels)
test_labels_encoded = label_encoder.transform(test_labels)
```

```
joblib.dump(train_labels_encoded, project_dir + '\joblib\\train_labels_encoded.
 ↪joblib')
joblib.dump(test_labels_encoded, project_dir + '\joblib\\test_labels_encoded.
 ↪joblib')
joblib.dump(label_encoder, project_dir + '\joblib\\label_encoder.joblib')
```

```
plt.imshow(test_images[0])
```

```
plt.imshow(train_images[1])
```

3 Extract features

```
def blur_image(image):
    blurred_image = cv2.medianBlur(image, 5)
    return blurred_image
```

```
def color_histogram(image):
    # image = cv2.cvtColor(image, cv2.COLOR_RGB2LUV)
    row, column, channel = image.shape[:3]
    size = row * column
    feature = []
    for k in range(channel):
        histogram = np.squeeze(cv2.calcHist([image], [k], None, [32], [0, 256]))
        histogram = histogram / size
        feature.extend(histogram)
    return feature
```

```
def hog(image):
    # image = cv2.cvtColor(image, cv2.COLOR_RGB2LUV)
    hog_features = skimage_hog(image, orientations=9, pixels_per_cell=(8, 8),␣
 ↪cells_per_block=(2, 2), visualize=False, block_norm='L2-Hys',␣
 ↪transform_sqrt=True, channel_axis=2)
    return hog_features
```

```
def extract_features(images):
    blurred_images = [blur_image(image) for image in tqdm(images,␣
 ↪desc="Sharpening Images")]
    color_features = [color_histogram(image) for image in tqdm(blurred_images,␣
 ↪desc="Extracting Color Features")]
    hog_features = [hog(image) for image in tqdm(blurred_images,␣
 ↪desc="Extracting HOG Features")]
    combined_features = [np.concatenate((color_feature, hog_feature))
                         for color_feature, hog_feature in␣
 ↪tqdm(zip(color_features, hog_features), desc="Combining Features")]

    return combined_features
```

```
train_features = extract_features(train_images)
joblib.dump(train_features, project_dir + '\joblib\\train_features.joblib')
```

```
test_features = extract_features(test_images)
joblib.dump(test_features, project_dir + '\joblib\\test_features.joblib')
```

```
X_train, X_val, y_train, y_val = train_test_split(train_features,␣
 ↪train_labels_encoded, test_size=0.2, random_state=42)
```

4 Distance metrics KNN

```
def chi_square_distance(x, y):
    return cv2.compareHist(np.array(x, dtype=np.float32), np.array(y, dtype=np.
 ↪float32), cv2.HISTCMP_CHISQR)

def bhattacharyya_distance(x, y):
    return cv2.compareHist(np.array(x, dtype=np.float32), np.array(y, dtype=np.
 ↪float32), cv2.HISTCMP_BHATTACHARYYA)

def intersection_distance(x, y):
    return 1 - cv2.compareHist(np.array(x, dtype=np.float32), np.array(y,␣
 ↪dtype=np.float32), cv2.HISTCMP_INTERSECT)
```

5 Gridsearch KNN

```
param_grid = {
    'n_neighbors': [3, 4, 5, 6, 7],
    'weights': ['uniform', 'distance'],
    'leaf_size': [10, 20, 30, 40, 50],
    'metric': [
        cityblock,
        cosine,
        correlation,
        sqeuclidean,
        chi_square_distance,
        bhattacharyya_distance,
        intersection_distance
    ]
}

knn_model = KNeighborsClassifier()
grid_search_knn = GridSearchCV(
    knn_model,
    param_grid,
    cv=3,
    scoring='accuracy',
    verbose=3
)

grid_search_knn.fit(X_train, y_train)
```

```
best_knn = grid_search_knn.best_estimator_
print(f"Best Params: {grid_search_knn.best_params_}")

best_knn.fit(train_features, train_labels_encoded)
```

```
y_pred_knn = best_knn.predict(test_features)

joblib.dump(best_knn, project_dir + '\joblib\\best_knn_model.joblib')
```

6 Gridsearch SVM

```
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['rbf', 'linear', 'poly', 'sigmoid'],
    'gamma': ['scale', 'auto', 0.1, 0.01, 0.001],
    'class_weight': ['balanced', None],
    'degree': [2, 3, 4],
}

svm_model = SVC(random_state=42)

grid_search_svm = GridSearchCV(
    estimator=svm_model,
    param_grid=param_grid,
    cv=3,
    scoring='accuracy',
    verbose=3,
)

grid_search_svm.fit(X_train, y_train)
```

```
best_svm = grid_search_svm.best_estimator_
# Get the best parameters and score
print("Best parameters:", grid_search_svm.best_params_)
best_svm.fit(train_features, train_labels_encoded)

y_pred_svm = best_svm.predict(test_features)

joblib.dump(best_svm, project_dir + '\joblib\\best_svm_model.joblib')
```

7 Predict on test images for KNN

```
report_knn = classification_report(test_labels_encoded, y_pred_svm,␣
  ↪target_names=label_encoder.classes_)
print(report_knn)
```

```
heatmap_label_knn = confusion_matrix(test_labels_encoded, y_pred_svm)

plt.figure(figsize=(10, 8))
```

```
sns.heatmap(heatmap_label_knn, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```
n_columns = 10
n_rows = math.ceil(len(test_images) / n_columns)

fig, axes = plt.subplots(n_rows, n_columns, figsize=(30, n_rows * 3))

for idx, (image, true_label, pred_label) in enumerate(zip(test_images,␣
 ↪test_labels_encoded, y_pred_knn)):
    row = idx // n_columns
    col = idx % n_columns

    axes[row, col].imshow(image)
    axes[row, col].set_title(f'True: {label_encoder.classes_[true_label]}\nPred:
 ↪ {label_encoder.classes_[pred_label]}')
    axes[row, col].axis('off')

for ax in axes.flat:
    if not ax.has_data():
        ax.axis('off')

plt.tight_layout()
plt.show()
```

8 Predict on test images for SVM

```
report_svm = classification_report(test_labels_encoded, y_pred_svm,␣
 ↪target_names=label_encoder.classes_)
print(report_svm)
```

```
heatmap_label_svm = confusion_matrix(test_labels_encoded, y_pred_svm)

plt.figure(figsize=(10, 8))
sns.heatmap(heatmap_label_svm, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```
n_columns = 10
n_rows = math.ceil(len(test_images) / n_columns)

fig, axes = plt.subplots(n_rows, n_columns, figsize=(30, n_rows * 3))

for idx, (image, true_label, pred_label) in enumerate(zip(test_images,␣
 ↪test_labels_encoded, y_pred_svm)):
    row = idx // n_columns
    col = idx % n_columns

    axes[row, col].imshow(image)
    axes[row, col].set_title(f'True: {label_encoder.classes_[true_label]}\nPred:
 ↪ {label_encoder.classes_[pred_label]}')
    axes[row, col].axis('off')

for ax in axes.flat:
    if not ax.has_data():
        ax.axis('off')

plt.tight_layout()
plt.show()
```

```
## Predict on test images for SVM
report_svm = classification_report(test_labels_encoded, y_pred_svm,␣
 ↪target_names=label_encoder.classes_)
print(report_svm)
heatmap_label_svm = confusion_matrix(test_labels_encoded, y_pred_svm)

plt.figure(figsize=(10, 8))
sns.heatmap(heatmap_label_svm, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
n_columns = 10
n_rows = math.ceil(len(test_images) / n_columns)

fig, axes = plt.subplots(n_rows, n_columns, figsize=(30, n_rows * 3))

for idx, (image, true_label, pred_label) in enumerate(zip(test_images,␣
 ↪test_labels_encoded, y_pred_svm)):
    row = idx // n_columns
    col = idx % n_columns

    axes[row, col].imshow(image)
    axes[row, col].set_title(f'True: {label_encoder.classes_[true_label]}\nPred:
 ↪ {label_encoder.classes_[pred_label]}')
    axes[row, col].axis('off')

for ax in axes.flat:
    if not ax.has_data():
        ax.axis('off')

plt.tight_layout()
plt.show()
```

9 Save grid search results

```
def save_grid_search_results_with_timestamp(grid_search, image_size,␣
 ↪project_dir):
    results_dir = os.path.join(project_dir)
    os.makedirs(results_dir, exist_ok=True)
    current_time = datetime.now().strftime('%Y-%m-%d_%H_%M_%S')
    file_path = os.path.join(results_dir, f'grid_search_results_{current_time}.
 ↪txt')

    # Lấy tất cả kết quả từ cv_results_
    cv_results = grid_search.cv_results_

    # Chuẩn bị nội dung ghi vào file
    results = [
        f"Date and Time: {current_time}",
        f"Image Size: {image_size}",
        f"Best Parameters: {grid_search.best_params_}",
        f"Best Score: {grid_search.best_score_}",
        "Detailed Grid Search Results:"
    ]

    # Lấy các tham số của mỗi lần huấn luyện
    param_keys = [key for key in cv_results if key.startswith("param_")]
    score_keys = [key for key in cv_results if key.startswith("mean") or key.
 ↪startswith("std")]

    # In từng kết quả từ cv_results_
    for i in range(len(cv_results['params'])):
        result = [f"Result {i + 1}:"]

        # Thêm các tham số của lần chạy hiện tại
        for key in param_keys:
            result.append(f"  {key}: {cv_results[key][i]}")
        # Thêm điểm số của lần chạy hiện tại
        for key in score_keys:
            result.append(f"  {key}: {cv_results[key][i]}")

        results.extend(result)

    # Ghi nội dung vào file .txt
    with open(file_path, 'w') as f:
        f.write("\n".join(results))

    print(f'Results saved to {file_path}')
```

```
image_size = (width, height)
save_grid_search_results_with_timestamp(grid_search_knn, image_size,␣
 ↪project_dir + '\grid_search_results\knn')
```

```
image_size = (width, height)
save_grid_search_results_with_timestamp(grid_search_svm, image_size,␣
 ↪project_dir + '\grid_search_results\svm')
```

```
def export_notebook_to_pdf(notebook_path, project_dir):
    # Tự động lưu notebook hiện tại

    display(Javascript('IPython.notebook.save_notebook()'))

    # Đợi một chút để đảm bảo notebook được lưu xong
    import time
    time.sleep(2)  # đợi 2 giây

    # Tạo thư mục kết quả
    results_dir = os.path.join(project_dir, "results")
    os.makedirs(results_dir, exist_ok=True)

    # Đọc notebook
    with open(notebook_path, 'r', encoding='utf-8') as f:
        nb = nbformat.read(f, as_version=4)

    # Cấu hình PDF exporter
    pdf_exporter = PDFExporter()
    pdf_exporter.exclude_input_prompt = True
    pdf_exporter.exclude_output_prompt = True

    # Thêm template và style cơ bản
    pdf_exporter.template_name = 'classic'

    # Chuyển đổi sang PDF
    pdf_data, resources = pdf_exporter.from_notebook_node(nb)
    # Tạo tên file với timestamp
    current_time = datetime.now().strftime('%Y-%m-%d_%H_%M_%S')
    pdf_file = os.path.join(results_dir, f"notebook_export_{current_time}.pdf")

    # Lưu file PDF
    with open(pdf_file, 'wb') as f:
        f.write(pdf_data)

    print(f"Đã xuất file PDF thành công: {pdf_file}")
    return pdf_file
```

```
# project_dir = os.path.dirname(project_dir)
notebook_path = project_dir + "\\model\\main.ipynb"
proj_dir = project_dir + "\\grid_search_results"

export_notebook_to_pdf(notebook_path, proj_dir)
```