# Notebook

November 22, 2024

## 1 Import libraries

```
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

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

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

```
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)
```

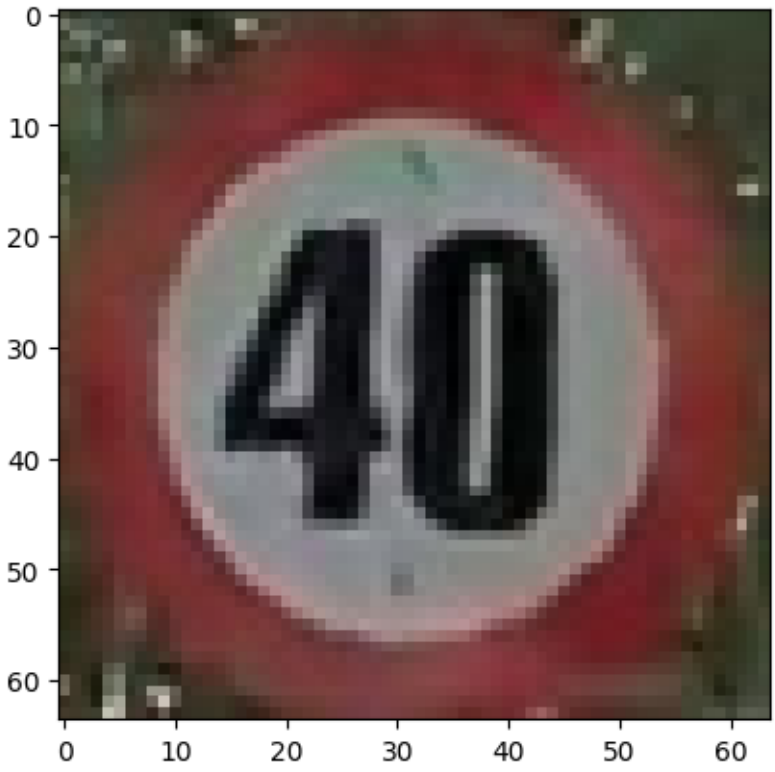
```
label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(train_labels)
test_labels_encoded = label_encoder.transform(test_labels)
```

```
joblib.dump(train_images, project_dir + '\joblib\\train_images.joblib')
joblib.dump(test_images, project_dir + '\joblib\\test_images.joblib')
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')
```

```
['d:\\ASUS\\Deploy-Traffic-Sign-Classification-through-
Images\\joblib\\label_encoder.joblib']
```

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

```
<matplotlib.image.AxesImage at 0x20aa4a5ce90>
```




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

```
<matplotlib.image.AxesImage at 0x20aa49db7d0>
```


## 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')
```

```
Sharpening Images: 100%|      | 1415/1415 [00:00<00:00, 1929.53it/s]
Extracting Color Features: 100%|      | 1415/1415 [00:00<00:00,
14334.57it/s]
Extracting HOG Features: 100%|      | 1415/1415 [00:05<00:00, 253.05it/s]
Combining Features: 1415it [00:00, 30002.78it/s]

['d:\\ASUS\\Deploy-Traffic-Sign-Classification-through-
Images\\joblib\\train_features.joblib']
```

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

```
Sharpening Images: 100%|      | 150/150 [00:00<00:00, 2158.15it/s]
Extracting Color Features: 100%|      | 150/150 [00:00<00:00, 11513.75it/s]
Extracting HOG Features: 100%|      | 150/150 [00:00<00:00, 294.02it/s]
Combining Features: 150it [00:00, ?it/s]

['d:\\ASUS\\Deploy-Traffic-Sign-Classification-through-
Images\\joblib\\test_features.joblib']
```

## 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 Load Best Model

```
# knn_model = joblib.load_model(project_dir + '\\joblib\\best_knn_model.joblib')
# svm_model = joblib.load_model(project_dir + '\\joblib\\best_svm_model.joblib')

# y_pred_knn = knn_model.predict(test_features)
# y_pred_svm = svm_model.predict(test_features)
```

## 6 Gridsearch KNN

```
knn_model = KNeighborsClassifier()
knn_model.fit(train_features, train_labels_encoded)
y_pred_knn = knn_model.predict(test_features)
```

```
# param_grid = {
#     'n_neighbors': [3, 4, 5, 6, 7, 10],
#     'weights': ['uniform', 'distance'],
#     'leaf_size': [5, 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='f1_macro',
#     verbose=3
# )
```

```
# grid_search_knn.fit(train_features, train_labels_encoded)
```

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


# y_pred_knn = best_knn.predict(test_features)

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

## 7 Gridsearch SVM

```
svm_model = SVC()
svm_model.fit(train_features, train_labels_encoded)
y_pred_svm = svm_model.predict(test_features)
```

```
# param_grid = {
#     'C': [0.1, 0.2, 0.3, 0.4],
#     'kernel': ['rbf', 'linear', 'poly', 'sigmoid'],
#     'gamma': ['scale', 'auto', 0.1, 0.01, 0.001],
#     'degree': [2, 3, 4],
# }

# svm_model = SVC()

# grid_search_svm = GridSearchCV(
#     estimator=svm_model,
#     param_grid=param_grid,
#     cv=3,
#     scoring='f1_macro',
#     verbose=3,
# )

# grid_search_svm.fit(train_features, train_labels_encoded)
```

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

# y_pred_svm = best_svm.predict(test_features)

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

## 8 Predict on test images for KNN

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

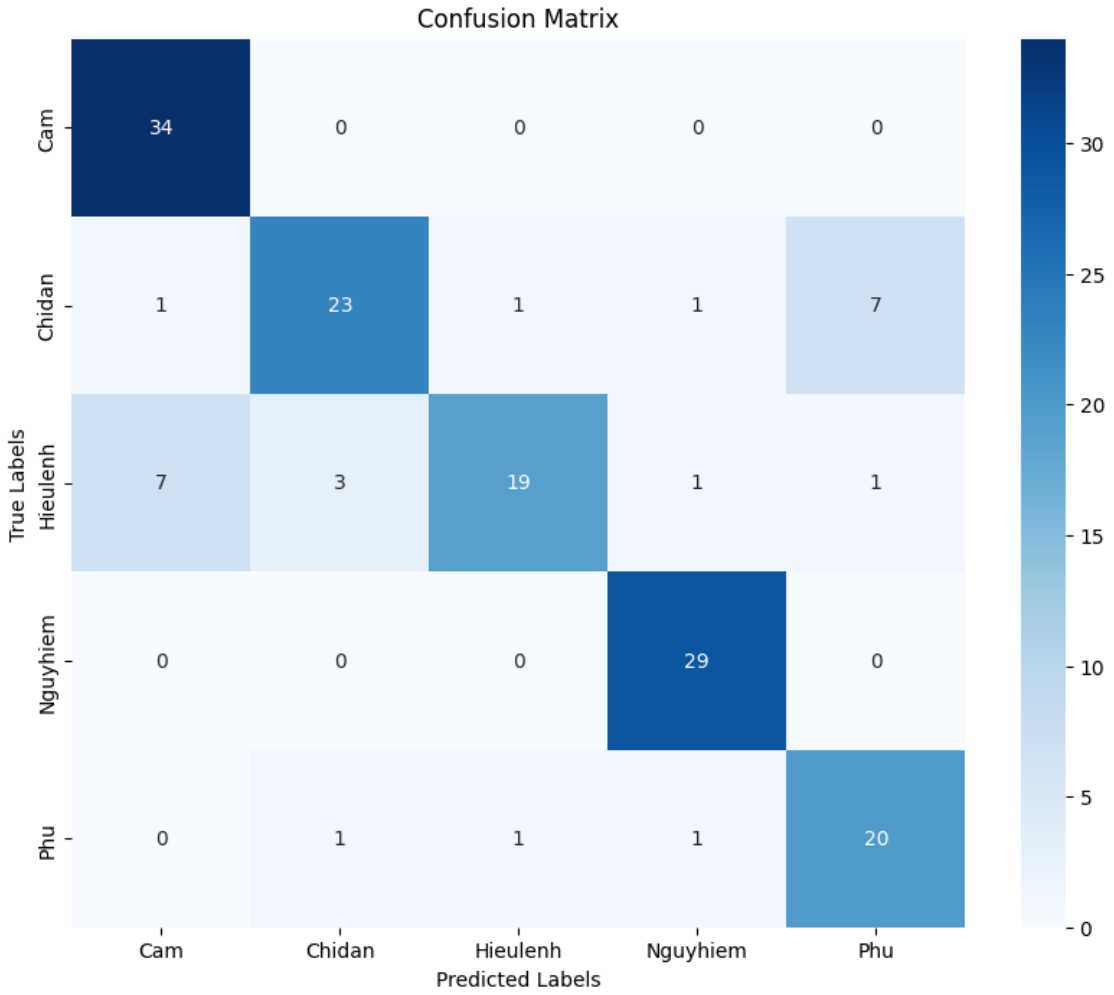
```
              precision    recall  f1-score   support

         Cam       0.81      1.00      0.89        34
      Chidan       0.85      0.70      0.77        33
    Hieulenh       0.90      0.61      0.73        31
    Nguyhiem       0.91      1.00      0.95        29
         Phu       0.71      0.87      0.78        23

    accuracy                           0.83       150
   macro avg       0.84      0.84      0.83       150
weighted avg       0.84      0.83      0.83       150
```

```
heatmap_label_knn = confusion_matrix(test_labels_encoded, y_pred_knn)

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()
```

## 9 Predict on test images for SVM

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

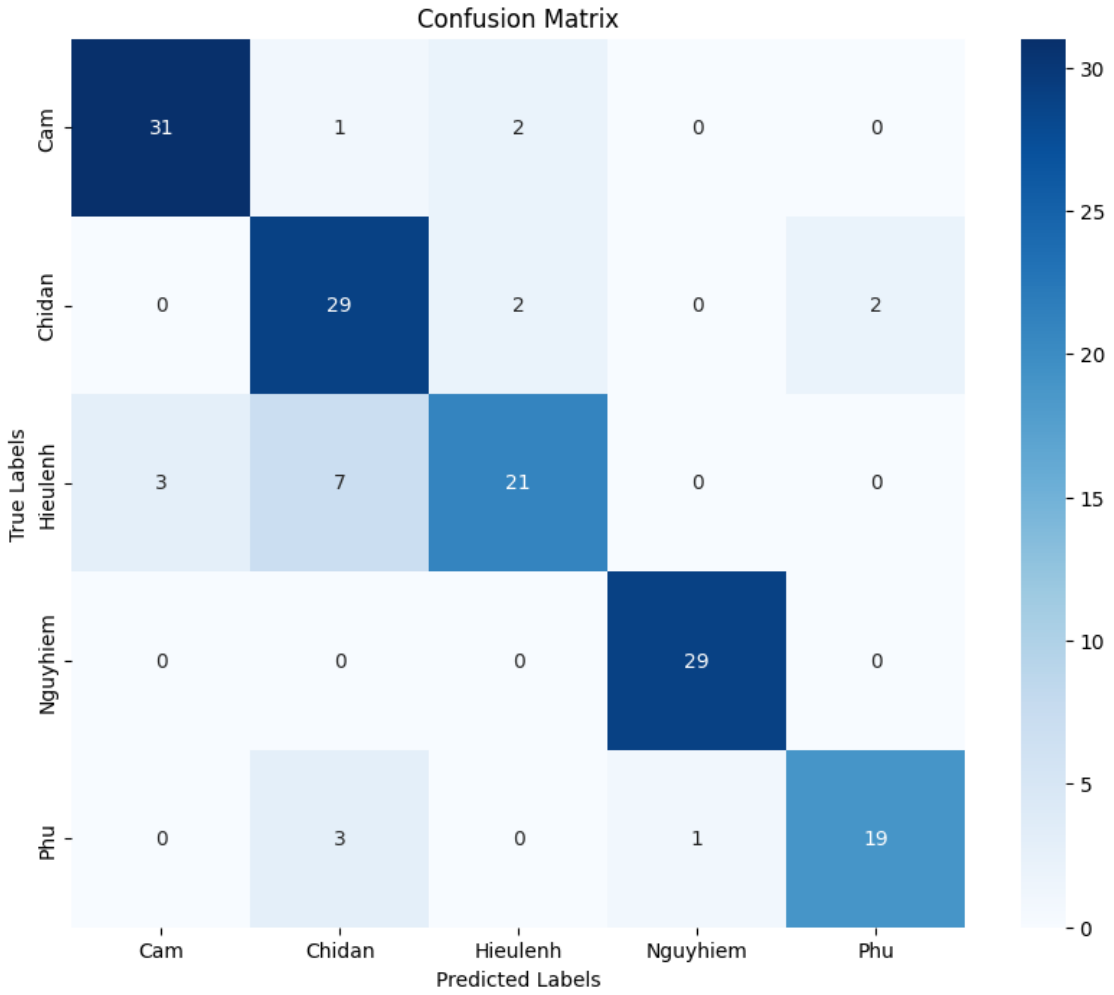
```
              precision    recall  f1-score   support

         Cam       0.91      0.91      0.91        34
      Chidan       0.72      0.88      0.79        33
    Hieulenh       0.84      0.68      0.75        31
    Nguyhiem       0.97      1.00      0.98        29
         Phu       0.90      0.83      0.86        23

    accuracy                           0.86       150
   macro avg       0.87      0.86      0.86       150
weighted avg       0.87      0.86      0.86       150
```

```
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()
```

## 10 Save grid search results

```
def export_notebook_to_pdf(notebook_path, project_dir):
    results_dir = os.path.join(project_dir)
    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)
```