MODELING

backend_classification > test_classification.py > ImageClassifierTester > __init__

```python
import os
import cv2
import numpy as np
import pickle

import sys
sys.path.append('C:/Users/ACER/Documents/kuliah/Code_penerapan_AI/backend_classification')
from FeatureExtractor_GLCM import GLCMFeatureExtractor

class ImageClassifierTester:
    def __init__(self, model_dir, feature_dir, feature_type):
        self.model_dir = model_dir
        self.feature_dir = feature_dir
        self.feature_type = feature_type
        self.data = None
        self.labels = None
        self.classifier = None
        self.feature_extractors = {
            "histogram": self.extract_histogram,
            "glcm": self.extract_glcm
        }

    def extract_histogram(self, image):
        hist = cv2.calcHist([image], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
        cv2.normalize(hist, hist)
        hist = hist.flatten()
        # Flatten and reshape histogram to 1-dimensional array
        hist = hist.reshape(1, -1)
        return hist

    def extract_glcm(self, image):
        feature_extractor = GLCMFeatureExtractor()
        glcm_features = feature_extractor.compute_glcm_features(image)
        return glcm_features

    def load_data(self):
        self.data = np.load(os.path.join(self.feature_dir, 'data.npy'))
        self.labels = np.load(os.path.join(self.feature_dir, 'labels.npy'))
```

backend_classification > test_classification.py > ImageClassifierTester > __init__

```python
class ImageClassifierTester:
    def load_classifier(self, classifier_type):
        model_file = os.path.join(self.model_dir, f'{classifier_type}_model.pkl')
        with open(model_file, 'rb') as f:
            self.classifier = pickle.load(f)

    def read_image(self, test_image_path):
        image = cv2.imread(test_image_path)
        return image

    def process_image(self, image):
        image = image
        return image

    def test_classifier(self, test_image_path):
        image = self.read_image(test_image_path)
        image = self.process_image(image)
        features = self.feature_extractors[self.feature_type](image)
        features = features.reshape(1, -1)

        prediction = self.classifier.predict(features)
        return prediction[0], features, image


if __name__ == "__main__":
    MODEL_DIR = 'C:/Users/ACER/Documents/kuliah/Code_penerapan_AI/backend_classification/model'
    FEATURE_DIR = 'C:/Users/ACER/Documents/kuliah/Code_penerapan_AI/backend_classification/fitur'
    FEATURE_TYPE = 'histogram'  # choose from 'histogram', 'glcm', or 'histogram_glcm'
    CLASSIFIER_TYPE = "naive_bayes"  # "mlp", "naive_bayes"

    TEST_IMAGE_PATH = 'C:/Users/ACER/Documents/kuliah/Code_penerapan_AI/backend_classification/nyobanyoba/Mangkok/Mangkok001.jpg'
    # Create an instance of ImageClassifierTester
    tester = ImageClassifierTester(MODEL_DIR, FEATURE_DIR, FEATURE_TYPE)
    tester.load_data()
    tester.load_classifier(CLASSIFIER_TYPE)

    # Test the classifier on the test image
    prediction = tester.test_classifier(TEST_IMAGE_PATH)
    print("Prediction:", prediction)
```

Ln 17, Col 31 Spaces: 4 UTF-8 CRLF Python 3.12.4 64-bit Windows (windows-x64)

main.dart M | aplikasi.ipynb | apiserver.py M | test_classification.py | pubspec.yaml | train_classification.py M | FeatureExtractor_GLCM.py

backend_classification > train_classification.py > ...

```python
import os
import cv2
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import pickle
import warnings
from pyswarm import pso  # Instal
                         (class) GLCMFeatureExtractor
from FeatureExtractor_GLCM import GLCMFeatureExtractor
warnings.filterwarnings("ignore")

class ImageClassifier:
    def __init__(self, dataset_dir, model_dir, feature_dir, feature_type):
        self.dataset_dir = dataset_dir
        self.model_dir = model_dir
        self.feature_dir = feature_dir
        self.feature_type = feature_type
        self.data = []
        self.labels = []
        self.feature_extractors = {
            "histogram": self.extract_histogram,
            "glcm": self.extract_glcm
        }
        self.classifiers = {
            "mlp": self.train_mlp_with_pso,
            "naive_bayes": self.train_naive_bayes
        }

    def extract_histogram(self, image):
        hist = cv2.calcHist([image], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
        cv2.normalize(hist, hist)
        hist = hist.flatten()
        hist = hist.reshape(1, -1)
        return hist
```

backend_classification > train_classification.py > ...

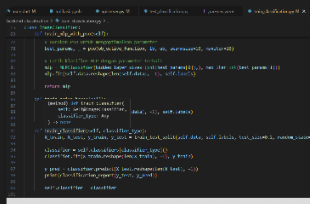
```python
class ImageClassifier:

    def extract_glcm(self, image):
        feature_extractor = GLCMFeatureExtractor()
        glcm_features = feature_extractor.compute_glcm_features(image)
        return glcm_features

    def load_data(self):
        for folder in os.listdir(self.dataset_dir):
            folder_path = os.path.join(self.dataset_dir, folder)

            if os.path.isdir(folder_path):
                for file in os.listdir(folder_path):
                    file_path = os.path.join(folder_path, file)

                    if file.endswith('.jpg'):
                        image = cv2.imread(file_path)
                        features = self.feature_extractors[self.feature_type](image)

                        self.data.append(features)
                        self.labels.append(folder)

        self.data = np.array(self.data)
        self.labels = np.array(self.labels)

    def train_mlp_with_pso(self):
        def objective_function(params):
            hidden_layer_size = int(params[0])
            max_iter = int(params[1])

            mlp = MLPClassifier(hidden_layer_sizes=(hidden_layer_size,), max_iter=max_iter)
            mlp.fit(self.data.reshape(len(self.data), -1), self.labels)

            y_pred = mlp.predict(self.data.reshape(len(self.data), -1))
            return -np.mean(y_pred == self.labels)  # Minimalkan negatif akurasi

        lb = [50, 100]  # Batas bawah untuk parameter (contoh: hidden_layer_size, max_iter)
        ub = [200, 500]  # Batas atas untuk parameter
```

main.dart M | aplikasi.ipynb | apiserver.py M | test_classification.py | pubspec.yaml | train_classification.py M | FeatureExtractor_GLCM.py

backend_classification > train_classification.py > ...

```python
class ImageClassifier:
    def train_mlp_with_pso(self):
        # Gunakan PSO untuk mengoptimalkan parameter
        best_params, _ = pso(objective_function, lb, ub, swarmsize=10, maxiter=10)

        # Latih klasifier MLP dengan parameter terbaik
        mlp = MLPClassifier(hidden_layer_sizes=(int(best_params[0]),), max_iter=int(best_params[1]))
        mlp.fit(self.data.reshape(len(self.data), -1), self.labels)

        return mlp

    def train_naive_bayes(self):
        self.classifier.fit(self.data.reshape(len(self.data), -1), self.labels)

    def train_classifier(self, classifier_type):
        X_train, X_test, y_train, y_test = train_test_split(self.data, self.labels, test_size=0.2, random_state=42)

        classifier = self.classifiers[classifier_type]()
        classifier.fit(X_train.reshape(len(X_train), -1), y_train)

        y_pred = classifier.predict(X_test.reshape(len(X_test), -1))
        print(classification_report(y_test, y_pred))

        self.classifier = classifier

    def save_classifier(self, classifier_type):
        np.save(os.path.join(self.feature_dir, 'data.npy'), self.data)
        np.save(os.path.join(self.feature_dir, 'labels.npy'), self.labels)

        classifier = self.classifier

        with open(os.path.join(self.model_dir, f'{classifier_type}_model.pkl'), 'wb') as f:
            pickle.dump(classifier, f)


if __name__ == "__main__":
    DATASET_DIR = 'C:/Users/ACER/Documents/kuliah/Code_penerapan_AI/backend_classification/nyobanyoba'
    MODEL_DIR = 'C:/Users/ACER/Documents/kuliah/Code_penerapan_AI/backend_classification/model'
    FEATURE_DIR = 'C:/Users/ACER/Documents/kuliah/Code_penerapan_AI/backend_classification/fitur'
    FEATURE_TYPE = 'histogram'  # pilih 'histogram', 'glcm', atau 'histogram_glcm'
    CLASSIFIER_TYPE = "naive_bayes"  # "mlp", "naive_bayes"

    # Buat instans ImageClassifier dan latih klasifier yang dipilih
    classifier = ImageClassifier(DATASET_DIR, MODEL_DIR, FEATURE_DIR, FEATURE_TYPE)
    classifier.load_data()
    classifier.train_classifier(CLASSIFIER_TYPE)
    classifier.save_classifier(CLASSIFIER_TYPE)
```

(method) def train_classifier(
    self: Self@ImageClassifier,
    classifier_type: Any
) -> None



Snipping Tool

DEPLOYMENT

API SERVER MENGGUNAKAN FLASK

main.dart M | aplikasi.ipynb | apiserver.py M | test_classification.py | pubspec.yaml | train_classification.py M | FeatureExtractor_GLCM.py

server > apiserver.py > ...

```python
from flask import Flask, jsonify, request
from werkzeug.utils import secure_filename
import os
import cv2
import sys
sys.path.append('C:/Users/ACER/Documents/kuliah/Code_penerapan_AI')

from backend_classification.test_classification import ImageClassifierTester

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'C:/Users/ACER/Documents/kuliah/Code_penerapan_AI/server/uploads_images'  # Sesuaikan dengan lokasi penyimpanan Anda
app.config['PREPROCESS_FOLDER'] = 'C:/Users/ACER/Documents/kuliah/Code_penerapan_AI/server/preprocess_images'  # Folder untuk hasil pengolahan citra

@app.route('/upload', methods=['POST'])
def upload_file():
    file = request.files['image']
    filename = secure_filename(file.filename)
    file_extension = os.path.splitext(filename)[1].lower()

    if file_extension != '.jpg' and file_extension != '.jpeg' and file_extension != '.png':
        return jsonify({'error': 'Citra harus dalam format JPG.'})
    else:
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(file_path)
        # Proses citra menggunakan OpenCV atau library lain
        image = cv2.imread(file_path)
        # Tambahkan kode pemrosesan citra di sini
        # processed_image = process_image(image)  # Asumsi process_image adalah fungsi Anda untuk memproses citra
        prediction, features, image = processed_image(file_path)
        # Simpan citra yang telah diproses
        processed_filename = 'processed_' + filename
        processed_file_path = os.path.join(app.config['PREPROCESS_FOLDER'], processed_filename)
        cv2.imwrite(processed_file_path, image)

        return jsonify(prediction)  # Mengirimkan hasil prediksi ke klien dalam format JSON
```

```python
def processed_image(file_path):
    MODEL_DIR = 'C:/Users/ACER/Documents/kuliah/Code_penerapan_AI/backend_classification/model'
    FEATURE_DIR = 'C:/Users/ACER/Documents/kuliah/Code_penerapan_AI/backend_classification/fitur'
    FEATURE_TYPE = 'histogram'  # choose from 'histogram', 'glcm', or 'histogram_glcm'
    CLASSIFIER_TYPE = "naive_bayes"  # "mlp", "naive_bayes"

    TEST_IMAGE_PATH = file_path

    # Create an instance of ImageClassifierTester
    tester = ImageClassifierTester(MODEL_DIR, FEATURE_DIR, FEATURE_TYPE)
    tester.load_data()
    tester.load_classifier(CLASSIFIER_TYPE)

    # Test the classifier on the test image
    prediction, features, image = tester.test_classifier(TEST_IMAGE_PATH)
    print("Prediction:", prediction)
    return prediction, features, image

if __name__ == '__main__':
    app.run(host='192.168.1.3', port=5000)
```