LAMPIRAN

1. Script Ubah Data dari Gambar Menjadi Vector 1 Dimensi

```
import os
import numpy as np
import cv2
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import joblib
import matplotlib.pyplot as plt

def extract_color_histogram(image, bins=(32, 32, 32)):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist_h = cv2.calcHist([hsv], [0], None, [bins[0]], [0, 256])
    hist_s = cv2.calcHist([hsv], [1], None, [bins[1]], [0, 256])
    hist_v = cv2.calcHist([hsv], [2], None, [bins[2]], [0, 256])
    hist_h = cv2.normalize(hist_h, hist_h).flatten()
    hist_s = cv2.normalize(hist_s, hist_s).flatten()
    hist_v = cv2.normalize(hist_v, hist_v).flatten()
    features = np.concatenate([hist_h, hist_s, hist_v])
    return features

def process_image(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_COLOR)
    if image is None:
        raise ValueError(f"Image at {image_path} could not be read.")
    image = cv2.resize(image, (64, 64))
    return extract_color_histogram(image)

def load_dataset(folder_paths, labels):
    data = []
    target = []
    for folder_path, label in zip(folder_paths, labels):
        for filename in os.listdir(folder_path):
            file_path = os.path.join(folder_path, filename)
            if os.path.isfile(file_path):
                try:
                    image_vector = process_image(file_path)
                    data.append(image_vector)
                    target.append(label)
                except ValueError as e:
                    print(e)
    return np.array(data), np.array(target)

# Load dataset
base_path = "/Users/macbook/Downloads/AplikasiAkbar"
folders = ["KulitPisang", "KulitTelur", "Nasi"]
```

```
labels = [0, 1, 2]

folder_paths = [os.path.join(base_path, folder) for folder in folders]
data, target = load_dataset(folder_paths, labels)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, random_state=42)

# Train model
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Test model and calculate accuracy
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Save model
model_file_path = os.path.join(base_path, "random_forest_model_histogram3.joblib")
joblib.dump(clf, model_file_path)
print(f"Model telah disimpan ke file {model_file_path}.")

# Plot accuracy
categories = ["Kulit Pisang", "Kulit Telur", "Nasi"]
correct_predictions = [sum((y_pred == i) & (y_test == i)) for i in range(len(categories))]
total_per_category = [sum(y_test == i) for i in range(len(categories))]
accuracy_per_category = [cp / tp if tp > 0 else 0 for cp, tp in zip(correct_predictions,
total_per_category)]

plt.figure(figsize=(8, 6))
plt.bar(categories, accuracy_per_category, color='skyblue')
plt.xlabel("Category")
plt.ylabel("Accuracy")
plt.title("Accuracy Per Category")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

2. Script Pembuatan Model .Joblib

```
import os
import numpy as np
import cv2
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import joblib

def extract_color_histogram(image, bins=(32, 32, 32)):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist_h = cv2.calcHist([hsv], [0], None, [bins[0]], [0, 256])
    hist_s = cv2.calcHist([hsv], [1], None, [bins[1]], [0, 256])
    hist_v = cv2.calcHist([hsv], [2], None, [bins[2]], [0, 256])
    hist_h = cv2.normalize(hist_h, hist_h).flatten()
    hist_s = cv2.normalize(hist_s, hist_s).flatten()
    hist_v = cv2.normalize(hist_v, hist_v).flatten()
    features = np.concatenate([hist_h, hist_s, hist_v])
    return features

def process_image(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_COLOR)
    if image is None:
        raise ValueError(f"Image at {image_path} could not be read.")
    image = cv2.resize(image, (64, 64))
    return extract_color_histogram(image)

def load_dataset(folder_paths, labels):
    data = []
    target = []
    for folder_path, label in zip(folder_paths, labels):
        for filename in os.listdir(folder_path):
            file_path = os.path.join(folder_path, filename)
            if os.path.isfile(file_path):
                try:
                    image_vector = process_image(file_path)
                    data.append(image_vector)
                    target.append(label)
                except ValueError as e:
                    print(e)
    return np.array(data), np.array(target)

base_path = "/Users/macbook/Downloads/AplikasiAkbar"
folders = ["KulitPisang", "KulitTelur", "Nasi"]
labels = [0, 1, 2]
```

```
folder_paths = [os.path.join(base_path, folder) for folder in folders]

data, target = load_dataset(folder_paths, labels)

X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, random_state=42)

clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

model_file_path = os.path.join(base_path, "random_forest_model_histogram3.joblib")
joblib.dump(clf, model_file_path)

print(f"Model telah disimpan ke file {model_file_path}.")
```

3. Script Pembuatan Confusion Matrix

```
import os
import numpy as np
import cv2
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import joblib
import matplotlib.pyplot as plt
import seaborn as sns

def extract_color_histogram(image, bins=(32, 32, 32)):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist_h = cv2.calcHist([hsv], [0], None, [bins[0]], [0, 256])
    hist_s = cv2.calcHist([hsv], [1], None, [bins[1]], [0, 256])
    hist_v = cv2.calcHist([hsv], [2], None, [bins[2]], [0, 256])
    hist_h = cv2.normalize(hist_h, hist_h).flatten()
    hist_s = cv2.normalize(hist_s, hist_s).flatten()
    hist_v = cv2.normalize(hist_v, hist_v).flatten()
    features = np.concatenate([hist_h, hist_s, hist_v])
    return features

def process_image(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_COLOR)
    if image is None:
        raise ValueError(f"Image at {image_path} could not be read.")
    image = cv2.resize(image, (64, 64))
    return extract_color_histogram(image)

def load_dataset(folder_paths, labels):
    data = []
    target = []
```

```
    for folder_path, label in zip(folder_paths, labels):
        for filename in os.listdir(folder_path):
            file_path = os.path.join(folder_path, filename)
            if os.path.isfile(file_path):
                try:
                    image_vector = process_image(file_path)
                    data.append(image_vector)
                    target.append(label)
                except ValueError as e:
                    print(e)
    return np.array(data), np.array(target)

# Load dataset
base_path = "/Users/macbook/Downloads/AplikasiAkbar"
folders = ["ORGANIK", "ANORGANIK"]  # Removed 'SAMPAH_ORGANIK_DAN_ANORGANIK'
labels = [0, 1]  # Only two classes now

folder_paths = [os.path.join(base_path, folder) for folder in folders]
data, target = load_dataset(folder_paths, labels)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, random_state=42)

# Train model
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Test model and calculate accuracy
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Save model
model_file_path = os.path.join(base_path, "random_forest_model_histogram2.joblib")  # Updated
model file name
joblib.dump(clf, model_file_path)
print(f"Model telah disimpan ke file {model_file_path}.")

# Plot accuracy
categories = ["Sampah Organik", "Sampah Anorganik"]  # Adjusted to only two categories
correct_predictions = [sum((y_pred == i) & (y_test == i)) for i in range(len(categories))]
total_per_category = [sum(y_test == i) for i in range(len(categories))]
accuracy_per_category = [cp / tp if tp > 0 else 0 for cp, tp in zip(correct_predictions,
total_per_category)]
```

```
plt.figure(figsize=(8, 6))
plt.bar(categories, accuracy_per_category, color='skyblue')
plt.xlabel("Kategori")
plt.ylabel("Akurasi")
plt.title("Akurasi Per Kategori")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# Limit confusion matrix values to a maximum of 10
cm = np.minimum(cm, 10)

# Plot confusion matrix as a heatmap with vmax=10
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=categories,
yticklabels=categories, vmax=10)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

4. Script Mengubah Format Model Joblib ke format TF.js

```
import os
import numpy as np
import cv2
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, Dense
import tensorflowjs as tfjs  # Untuk konversi ke TensorFlow.js

def extract_color_histogram(image, bins=(32, 32, 32)):
    # Konversi gambar ke ruang warna HSV
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # Menghitung histogram untuk masing-masing channel HSV
    hist_h = cv2.calcHist([hsv], [0], None, [bins[0]], [0, 256])
    hist_s = cv2.calcHist([hsv], [1], None, [bins[1]], [0, 256])
    hist_v = cv2.calcHist([hsv], [2], None, [bins[2]], [0, 256])
    # Normalisasi histogram dan flatten
    hist_h = cv2.normalize(hist_h, hist_h).flatten()
    hist_s = cv2.normalize(hist_s, hist_s).flatten()
    hist_v = cv2.normalize(hist_v, hist_v).flatten()
    # Menggabungkan fitur histogram
    features = np.concatenate([hist_h, hist_s, hist_v])
    return features

def process_image(image_path):
    # Membaca gambar
    image = cv2.imread(image_path, cv2.IMREAD_COLOR)
    if image is None:
        raise ValueError(f"Image at {image_path} could not be read.")
    # Mengubah ukuran gambar
    image = cv2.resize(image, (64, 64))
    # Ekstraksi fitur histogram
    return extract_color_histogram(image)

def load_dataset(folder_paths, labels):
    data = []
    target = []
    valid_extensions = ('.jpg', '.jpeg', '.png', '.bmp', '.tiff', '.tif')
    for folder_path, label in zip(folder_paths, labels):
        for filename in os.listdir(folder_path):
            # Mengecek ekstensi file
            if filename.lower().endswith(valid_extensions):
                file_path = os.path.join(folder_path, filename)
                if os.path.isfile(file_path):
                    try:
```

```
                image_vector = process_image(file_path)
                data.append(image_vector)
                target.append(label)
            except ValueError as e:
                print(e)
        else:
            print(f"File {filename} bukan gambar, dilewati.")
    return np.array(data), np.array(target)

# Path ke dataset
base_path = "/Users/macbook/Downloads/AplikasiAkbar"
folders = ["KulitPisang", "KulitTelur", "Nasi"]
labels = [0, 1, 2]

folder_paths = [os.path.join(base_path, folder) for folder in folders]

# Memuat dataset
data, target = load_dataset(folder_paths, labels)

# Split data menjadi training dan testing
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, random_state=42)

# Definisikan model Neural Network menggunakan Keras
def create_model(input_shape, num_classes):
    model = Sequential()
    model.add(InputLayer(input_shape=input_shape))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    # Kompilasi model
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

input_shape = (X_train.shape[1],)  # Jumlah fitur dari histogram warna
num_classes = len(set(y_train))  # Jumlah kelas

# Inisialisasi dan latih model
model = create_model(input_shape, num_classes)
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Simpan model Keras dalam format .keras (format Keras terbaru)
keras_model_path = os.path.join(base_path, "keras_model.keras")
model.save(keras_model_path)
print(f"Model Keras telah disimpan ke file {keras_model_path}.")
```

```
# Konversi model Keras ke TensorFlow.js
tensorflow_js_model_path = os.path.join(base_path, "tensorflow_model_js")
tfjs.converters.save_keras_model(model, tensorflow_js_model_path)
print(f"Model TensorFlow.js telah disimpan ke folder {tensorflow_js_model_path}.")
```

5. Script Halaman Utama (UI)

```
<!DOCTYPE html>
<html lang="id">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Aplikasi Deteksi Sampah Organik</title>
  <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap"
 rel="stylesheet">
  <style>
    body {
      font-family: 'Roboto', sans-serif;
      background-color: #e0f7fa;
      margin: 0;
      padding: 0;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      height: 100vh;
      text-align: center;
    }
    h1 {
      margin-bottom: 30px;
      color: #ff5722;
      font-size: 24px;
    }
    .button-container {
      display: flex;
      flex-direction: column;
      gap: 15px;
    }
    button {
      padding: 15px;
      font-size: 18px;
      color: white;
      background-color: #007bff;
      border: none;
      border-radius: 5px;
      cursor: pointer;
```

```
            transition: background-color 0.3s, transform 0.3s;
            width: 200px;
            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);
        }
        button:hover {
            background-color: #0056b3;
            transform: translateY(-2px);
        }
        @media (max-width: 600px) {
            h1 {
                font-size: 20px;
            }
            button {
                width: 100%;
            }
        }
    </style>
</head>
<body>
    <h1>Aplikasi Deteksi Sampah</h1>
    <div class="button-container">
        <form action="realtime.php" method="post">
            <button type="submit">Test Realtime</button>
        </form>
        <form action="test_gambar.php" method="post">
            <button type="submit">Test Gambar</button>
        </form>
        <form action="logout.php" method="post">
            <button type="submit" onclick="return confirmLogout()">Keluar</button>
        </form>
    </div>

    <script>
        function confirmLogout() {
            return confirm("Anda yakin ingin keluar?");
        }
    </script>
</body>
</html>
```

6. Script Tes *Realtime*

```
<!DOCTYPE html>
<html lang="id">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Prediksi Real-Time</title>
    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap"
rel="stylesheet">
    <style>
        body {
            font-family: 'Roboto', sans-serif;
            background-color: #e0f7fa;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            height: 100vh;
            text-align: center;
        }
        h1 {
            margin-bottom: 30px;
            color: #ff5722;
            font-size: 24px;
        }
        #webcam-container {
            margin-top: 20px;
        }
        #prediction-container {
            margin-top: 20px;
            font-size: 18px;
            font-weight: bold;
            color: #333;
        }
        #back-button {
            margin-top: 30px;
            padding: 10px 20px;
            font-size: 16px;
            color: white;
            background-color: #007bff;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            transition: background-color 0.3s;
```

```
        }
        #back-button:hover {
            background-color: #0056b3;
        }
        @media (max-width: 600px) {
            h1 {
                font-size: 20px;
            }
            #back-button {
                width: 100%;
            }
        }
    </style>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest/dist/tf.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/@teachablemachine/image@latest/dist/teachablemachine-image.min.js"></script>
</head>
<body>
    <h1>Prediksi Real-Time</h1>
    <div id="webcam-container"></div>
    <div id="prediction-container">Prediksi: -</div>
    <button id="back-button" onclick="goBack()">Kembali ke Menu Utama</button>

    <script>
        const URL = "./models/";
        let model, webcam, maxPredictions;

        async function init() {
            const modelURL = URL + "model.json";
            const metadataURL = URL + "metadata.json";

            // Memuat model dan metadata
            model = await tmImage.load(modelURL, metadataURL);
            maxPredictions = model.getTotalClasses();

            // Mengatur webcam tanpa mirroring
            const flip = false; // Tidak melakukan mirroring
            webcam = new tmImage.Webcam(400, 300, flip);

            try {
                await webcam.setup({ facingMode: "environment" }); // Menggunakan kamera belakang
                await webcam.play();
                window.requestAnimationFrame(loop);

                // Menambahkan canvas webcam ke DOM
                document.getElementById("webcam-container").appendChild(webcam.canvas);
```

```
        } catch (error) {
            alert("Gagal mengakses kamera. Pastikan izin telah diberikan dan gunakan HTTPS.");
            console.error("Kesalahan webcam:", error);
        }
    }

    async function loop() {
        webcam.update();

        // Menggunakan setTimeout untuk mengurangi beban prediksi
        setTimeout(async () => {
            await predict();
            window.requestAnimationFrame(loop);
        }, 100); // Interval prediksi (100ms untuk mengurangi delay)
    }

    async function predict() {
        const predictions = await model.predict(webcam.canvas);
        const threshold = 0.5; // Ambang batas 50%

        // Filter prediksi berdasarkan probabilitas
        const validPredictions = predictions.filter(pred => pred.probability > threshold);

        // Urutkan prediksi berdasarkan probabilitas (dari tinggi ke rendah)
        validPredictions.sort((a, b) => b.probability - a.probability);

        // Ambil hingga dua prediksi tertinggi
        const topPredictions = validPredictions.slice(0, 2);

        let predictionsText = "Prediksi: ";

        if (topPredictions.length > 0) {
            predictionsText += topPredictions.map(pred => pred.className).join(" dan ");
        } else {
            predictionsText += "Tidak ada objek yang terdeteksi.";
        }

        document.getElementById("prediction-container").innerHTML = predictionsText;
    }

    function goBack() {
        window.location.href = "index.php";
    }

    window.onload = init;
</script>
```

```
</body>
</html>
```

7. Script Tes Gambar

```
<!DOCTYPE html>
<html lang="id">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Test Gambar</title>
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest/dist/tf.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/@teachablemachine/image@latest/dist/teachablemachine-
image.min.js"></script>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      padding: 20px;
      background-color: #f0f8ff;
      color: #333;
    }
    h1 {
      color: #ff4500;
    }
    input[type="file"] {
      margin-top: 20px;
      padding: 10px;
      border: 2px solid #ff4500;
      border-radius: 5px;
      background-color: #fff;
      color: #333;
      width: 80%;
      max-width: 300px;
    }
    button {
      margin-top: 20px;
      padding: 10px 20px;
      font-size: 16px;
      background-color: #007bff;
      color: white;
      border: none;
      border-radius: 5px;
      cursor: pointer;
      transition: background-color 0.3s ease;
    }
    button:hover {
```

```
            background-color: #0056b3;
        }
        img {
            max-width: 300px;
            height: auto;
            margin-top: 20px;
            display: block;
            margin-left: auto;
            margin-right: auto;
        }
        #result {
            margin-top: 20px;
            font-size: 20px;
            font-weight: bold;
            color: #ff4500;
        }
    </style>
</head>
<body>
    <h1>Test Gambar</h1>
    <input type="file" id="imageUpload" accept="image/*" />
    <button type="button" onclick="predictImage()">Prediksi Gambar</button>
    <div id="result"></div>
    <img id="selectedImage" src="" alt="" style="display:none;"/>

    <button onclick="goBack()">Kembali ke Menu Utama</button>

    <script>
        const URL = "./models/";
        let model;

        // Muat model saat halaman dimuat
        window.onload = async () => {
            const modelURL = URL + "model.json";
            const metadataURL = URL + "metadata.json";
            model = await tmImage.load(modelURL, metadataURL);
        };

        async function predictImage() {
            const imageUpload = document.getElementById('imageUpload');
            const image = imageUpload.files[0];
            const imgElement = document.getElementById('selectedImage');

            if (image) {
                const reader = new FileReader();
                reader.onload = async (event) => {
```

```
                imgElement.src = event.target.result;
                imgElement.style.display = 'block';

                // Pastikan ukuran gambar seragam dengan data pelatihan
                const processedImage = new Image();
                processedImage.onload = async () => {
                    const predictions = await model.predict(processedImage);
                    displayPrediction(predictions);
                };
                processedImage.src = event.target.result;
            };
            reader.readAsDataURL(image);
        } else {
            alert("Silakan pilih gambar terlebih dahulu.");
        }
    }

    function displayPrediction(predictions) {
        const resultContainer = document.getElementById('result');
        resultContainer.innerHTML = '';

        // Ambang batas probabilitas minimum
        const threshold = 0.5;

        // Filter prediksi yang valid
        const validPredictions = predictions.filter(pred => pred.probability > threshold);

        if (validPredictions.length > 0) {
            let predictionsText = "Prediksi:<br>";
            validPredictions.forEach(pred => {
                predictionsText += `${pred.className} (${(pred.probability * 100).toFixed(2)}%)<br>`;
            });
            resultContainer.innerHTML = predictionsText;
        } else {
            resultContainer.innerHTML = "Tidak ada prediksi yang valid.";
        }
    }

    function goBack() {
        window.location.href = "index.php";
    }
</script>
</body>
</html>
```

8. **Script Logout**

```
<?php
```

```
// Mulai sesi
session_start();

// Hapus semua data sesi
session_unset();

// Hancurkan sesi
session_destroy();

// Arahkan pengguna kembali ke halaman utama atau login
header("Location: index.php"); // Ganti dengan URL halaman utama Anda
exit();
?>
```