

# SISTEM PREDIKSI MAKANAN DAN ESTIMASI NILAI GIZI BERBASIS CITRA MENGGUNAKAN DEEP LEARNING

## Import Library

Blok ini berisi impor semua library yang diperlukan dalam proyek:

- `os` dan `random` : Untuk operasi file dan pemilihan acak.
- `torch` , `torch.nn` , `torch.nn.functional` : Untuk membangun dan menjalankan model deep learning dengan PyTorch.
- `torchvision.models` , `torchvision.transforms` : Untuk menggunakan model pretrained (ResNet50) dan preprocessing gambar.
- `PIL.Image` : Untuk membuka dan memproses gambar.
- `pandas` : Untuk membaca dan memanipulasi data nutrisi dari file `.csv` .
- `matplotlib.pyplot` : Untuk menampilkan gambar.
- `IPython.display` , `ipywidgets` : Untuk menampilkan hasil secara interaktif di Jupyter Notebook.

```
In [1]: import os
import random
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import models, transforms
from PIL import Image
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import display
import ipywidgets as widgets
```

## Sel 2: Definisikan Path dan Load Dataset Nutrisi

### Inisialisasi Path dan Load Data Nutrisi

- Menentukan direktori utama ( `BASE_DIR` ) sebagai acuan lokasi file.
- Menyusun path lengkap menuju:
  - `MODEL_PATH` : Lokasi file model hasil pelatihan ( `food101_model.pth` ).
  - `NUTRITION_PATH` : File CSV berisi data nilai gizi tiap makanan.
  - `IMAGE_PATH` : Folder tempat menyimpan gambar input.
- Terakhir, file `nutrition.csv` dibaca ke dalam DataFrame `nutrition_df` agar bisa digunakan untuk mengambil informasi kalori, protein, dll berdasarkan label makanan

yang diprediksi.

```
In [2]: # Direktori utama (ubah sesuai lokasi file)
BASE_DIR = '.'

# Lokasi model dan data nutrisi
MODEL_PATH = os.path.join(BASE_DIR, 'models', 'food101_model.pth')
NUTRITION_PATH = os.path.join(BASE_DIR, 'data', 'nutrition.csv')
IMAGE_PATH = os.path.join(BASE_DIR, 'static', 'images')

# Load data nutrisi ke DataFrame
nutrition_df = pd.read_csv(NUTRITION_PATH)
nutrition_df.head()
```

```
Out[2]:
```


	label	weight	calories	protein	carbohydrates	fats	fiber	sugars	sodium
0	apple_pie	80	240	2	36	10	2	16	120
1	apple_pie	100	300	3	45	12	2	20	150
2	apple_pie	120	360	4	54	14	3	24	180
3	apple_pie	150	450	5	68	18	3	30	225
4	apple_pie	200	600	6	90	24	4	40	300

## Sel 3: Load Model Food101 ResNet50

Fungsi `load_model()` melakukan hal berikut:

- Memanggil arsitektur **ResNet50** dari `torchvision.models`.
- Mengganti **fully connected layer** terakhir agar sesuai dengan jumlah kelas Food101 (101 kelas), ditambah dropout 0.4 untuk regularisasi.
- Memuat **model terlatih (pretrained)** dari file `food101_model.pth`.
- Mengatur model ke **mode evaluasi** (`model.eval()`) agar dropout dan batchnorm bekerja dengan benar saat inferensi.
- Mengambil `idx_to_class` untuk memetakan output indeks ke nama kelas makanan.
- Mengembalikan model dan pemetaan label.

Contoh output:

 "Model dimuat dengan 101 kelas."

```
In [3]: def load_model():
model = models.resnet50()
model.fc = nn.Sequential(
    nn.Dropout(0.4),
    nn.Linear(model.fc.in_features, 101)
)
checkpoint = torch.load(MODEL_PATH, map_location='cpu')
model.load_state_dict(checkpoint['model_state_dict'])
```

```

model.eval()
idx_to_class = checkpoint['idx_to_class']
return model, idx_to_class

model, idx_to_class = load_model()
print(f"Model dimuat dengan {len(idx_to_class)} kelas.")

```

Model dimuat dengan 101 kelas.



## Sel 4: Fungsi Preprocessing Gambar

Fungsi `preprocess_image(image_path)` melakukan pengolahan awal terhadap gambar sebelum dimasukkan ke model:

1. **Resize** – Ubah ukuran sisi terpendek gambar menjadi 256 piksel.
2. **CenterCrop** – Potong bagian tengah gambar menjadi 224x224 piksel.
3. **ToTensor** – Ubah gambar ke format tensor (dari [0–255] menjadi [0.0–1.0]).
4. **Normalize** – Sesuaikan nilai pixel berdasarkan mean dan std dataset ImageNet.
5. **Unsqueeze** – Tambahkan dimensi batch ( [1, 3, 224, 224] ) agar bisa diproses oleh model.

Fungsi ini mengembalikan:

- `tensor` : gambar dalam bentuk tensor untuk model.
- `image` : gambar asli (PIL) untuk ditampilkan ke user jika diperlukan.

```

In [4]: def preprocess_image(image_path):
        transform = transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406],
                                [0.229, 0.224, 0.225])
        ])
        image = Image.open(image_path).convert('RGB')
        tensor = transform(image).unsqueeze(0)
        return tensor, image

```



## Sel 5: Fungsi Prediksi Label dan Confidence

Fungsi `predict_label(tensor)` melakukan prediksi jenis makanan dari tensor gambar yang sudah diproses:

- Mematikan perhitungan gradien dengan `torch.no_grad()` agar proses prediksi lebih cepat dan hemat memori.
- Memasukkan tensor ke model untuk mendapatkan output logit.
- Mengubah output menjadi probabilitas menggunakan fungsi `softmax`.
- Mengambil label dengan probabilitas tertinggi ( `top_prob` ) dan indeksnya ( `top_idx` ).

- Mengonversi indeks menjadi nama kelas menggunakan `idx_to_class`.
- Mengembalikan:
  - `label` : nama makanan hasil prediksi.
  - `confidence` : nilai probabilitas keyakinan model terhadap prediksi tersebut.

```
In [5]: def predict_label(tensor):
        with torch.no_grad():
            output = model(tensor)
            prob = F.softmax(output, dim=1)
            top_prob, top_idx = torch.max(prob, 1)
            label = idx_to_class[top_idx.item()]
            confidence = top_prob.item()
        return label, confidence
```

## Sel 6: Fungsi Mendapatkan Berat Tersedia per Kategori

Fungsi `get_weight_options(categories)` bertujuan untuk:

- Menerima daftar kategori makanan ( `categories` ).
- Untuk tiap kategori, mencari nilai-nilai **berat** ( `weight` ) unik dari data nutrisi ( `nutrition_df` ) yang labelnya mengandung kategori tersebut.
- Mengumpulkan opsi berat tersebut dalam dictionary `weight_options` dengan kategori sebagai kunci.
- Mengurutkan daftar berat secara ascending agar mudah dipilih oleh pengguna.

Output:

- Dictionary berisi kategori sebagai key dan list berat yang mungkin sebagai value.

```
In [6]: def get_weight_options(categories):
        weight_options = {}
        for cat in categories:
            weights = nutrition_df[nutrition_df['label'].str.contains(cat, case=False)]
            weight_options[cat] = sorted(weights.tolist())
        return weight_options
```

## Sel 7: Fungsi Mendapatkan Data Nutrisi Berdasarkan Label dan Berat

Fungsi `get_nutrition(label, weight)` melakukan:

- Memfilter data nutrisi ( `nutrition_df` ) berdasarkan:
  - Label makanan yang mengandung `label` (case-insensitive).
  - Berat makanan sesuai input `weight`.
- Jika hasil filter kosong, fungsi mengembalikan `None`.

- Jika ada data, mengambil baris pertama ( `iloc[0]` ) dan mengembalikan dictionary berisi nilai nutrisi utama seperti:
  - Kalori
  - Protein
  - Karbohidrat
  - Lemak
  - Serat
  - Gula
  - Sodium

```
In [7]: def get_nutrition(label, weight):
    filtered = nutrition_df[nutrition_df['label'].str.contains(label, case=False)]
    filtered = filtered[filtered['weight'] == float(weight)]
    if filtered.empty:
        return None
    row = filtered.iloc[0]
    return {
        'Calories': row['calories'],
        'Protein (g)': row['protein'],
        'Carbohydrates (g)': row['carbohydrates'],
        'Fat (g)': row['fats'],
        'Fiber (g)': row['fiber'],
        'Sugars (g)': row['sugars'],
        'Sodium (mg)': row['sodium']
    }
```

## Sel 8: Siapkan Widget Interaktif untuk Pilihan Kategori dan Berat

- **Mendapatkan daftar kategori** makanan dari folder gambar di `IMAGE_PATH` (folder subdirektori dianggap kategori).
- Memanggil fungsi `get_weight_options(categories)` untuk mendapatkan opsi berat yang tersedia per kategori.
- Membuat widget dropdown interaktif:
  - `category_dropdown` : memilih kategori makanan.
  - `weight_dropdown` : memilih berat makanan (gram).
- Membuat fungsi `update_weight_options` yang:
  - Memperbarui opsi berat pada dropdown berat berdasarkan kategori yang dipilih.
- Menghubungkan fungsi update dengan event perubahan pilihan kategori ( `observe` ).
- Memanggil `update_weight_options()` sekali agar dropdown berat terisi saat awal.

Ini memungkinkan pengguna memilih kategori dan berat makanan secara interaktif di notebook.

```
In [8]: # Dapatkan daftar kategori berdasarkan folder gambar
categories = [cat for cat in os.listdir(IMAGE_PATH) if os.path.isdir(os.path.join(I
weight_options = get_weight_options(categories)
```

```

category_dropdown = widgets.Dropdown(options=categories, description='Kategori:')
weight_dropdown = widgets.Dropdown(description='Berat (g):')

def update_weight_options(change=None):
    selected_cat = category_dropdown.value
    weights = weight_options.get(selected_cat, [100])
    weight_dropdown.options = weights

category_dropdown.observe(update_weight_options, names='value')
update_weight_options()

```

## 9. Fungsi dan Tombol Prediksi

- Membuat tombol interaktif `predict_button` untuk memicu prediksi.
- Membuat `output` widget untuk menampilkan hasil secara interaktif.
- Fungsi `on_predict_clicked(b)` melakukan langkah berikut saat tombol ditekan:
  1. Mengambil kategori dan berat yang dipilih pengguna.
  2. Mengambil daftar gambar dari folder kategori tersebut.
  3. Memilih gambar secara acak untuk diprediksi.
  4. Melakukan preprocessing pada gambar.
  5. Memanggil model untuk prediksi label dan confidence.
  6. Mengambil data nutrisi berdasarkan label prediksi dan berat.
  7. Menampilkan nama gambar, hasil prediksi, tingkat kepercayaan, dan data nutrisi.
  8. Menampilkan gambar dengan judul prediksi dan confidence.
- Event handler tombol `predict_button` dihubungkan dengan fungsi `on_predict_clicked`.

Kode ini memungkinkan interaksi penuh dari pemilihan kategori dan berat hingga prediksi dan visualisasi hasil dalam notebook.

```

In [9]: predict_button = widgets.Button(description="Prediksi")
        output = widgets.Output()

def on_predict_clicked(b):
    with output:
        output.clear_output()
        category = category_dropdown.value
        weight = float(weight_dropdown.value)

        image_dir = os.path.join(IMAGE_PATH, category)
        image_files = [f for f in os.listdir(image_dir) if f.lower().endswith(('.jpg', '.png'))]
        if not image_files:
            print("Tidak ada gambar dalam kategori ini.")
            return

        image_name = random.choice(image_files)

```

```

image_path = os.path.join(image_dir, image_name)
tensor, image = preprocess_image(image_path)
label, confidence = predict_label(tensor)
nutrition = get_nutrition(label, weight)

print(f"🖼️ Gambar: {image_name}")
print(f"🔮 Prediksi: {label}")
print(f"🎯 Akurasi: {confidence*100:.2f}%")
print(f"🍽️ Nutrisi:")
if nutrition:
    for k, v in nutrition.items():
        print(f"    {k}: {v}")
else:
    print("    Data nutrisi tidak ditemukan untuk pilihan ini.")

plt.imshow(image)
plt.axis('off')
plt.title(f"Prediksi: {label} ({confidence*100:.1f}%")
plt.show()

predict_button.on_click(on_predict_clicked)

```

## 10. Tampilkan Widget Interaktif

Menampilkan widget dropdown kategori, dropdown berat, tombol prediksi, dan area output hasil prediksi secara berurutan dalam notebook agar user dapat berinteraksi dengan aplikasi deteksi nutrisi makanan.

```
In [10]: display(category_dropdown, weight_dropdown, predict_button, output)
```

```

Dropdown(description='Kategori:', options=('apple_pie', 'baby_back_ribs', 'baklava',
'beef_carpaccio', 'beef_t...
Dropdown(description='Berat (g):', options=(80, 100, 120, 150, 200), value=None)
Button(description='Prediksi', style=ButtonStyle())
Output()

```