Costum CNN

```
import os, sys, json, glob, itertools, random
from dataclasses import dataclass
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers, callbacks
from tensorflow.keras.preprocessing import image_dataset_from_directory

from sklearn.metrics import confusion_matrix, classification_report
from PIL import Image

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
os.environ["TF_FORCE_GPU_ALLOW_GROWTH"] = "true"
os.environ["TF_ENABLE_ONEDNN_OPTS"] = "0"
os.environ["TF_DETERMINISTIC_OPS"] = "1"

import sys
import tensorflow as tf

print("Python:", sys.version)
print("TensorFlow:", tf._version_)
print("Num GPUs Available:", len(tf.config.list_physical_devices('GPU')))
print("Physical GPUs:", tf.config.list_physical_devices('GPU'))

# === 1. CONFIG ===
@dataclass
class Config:
```

```
  data_dir: str = "/home/jupyter-230712427/Projek UAS PMDPM_A_Pingouin/budaya_nusantara_foods/dataset"
  predict_dir: str = "/home/jupyter-230712427/Projek UAS PMDPM_A_Pingouin/budaya_nusantara_foods/predict_samples"
  img_size: int = 128
  batch_size: int = 8
  seed: int = 42
  epochs: int = 10
  models_dir: str = "/home/jupyter-230712427/Projek UAS PMDPM_A_Pingouin/budaya_nusantara_foods/models"

cfg = Config()
os.makedirs(cfg.models_dir, exist_ok=True)
cfg
```

Link Dataset Drive : https://drive.google.com/drive/folders/14htYGHZ86IXBaylLl8XvF1_r4HuDp8AC?usp=drive_link

```
# === 2. DATA LOADING ===
def load_datasets(data_dir: str, img_size: int, batch_size: int, seed: int=42):
  data_dir = Path(data_dir)
  train_dir = data_dir / "train"
  val_dir   = data_dir / "val"
  test_dir = data_dir / "test"

  print(f" Memuat dataset dari: {data_dir.resolve()}")
  print(f" Train: {train_dir}")
  print(f" Val : {val_dir}")
  print(f" Test : {test_dir}\n")

  train_ds = image_dataset_from_directory(
    train_dir, seed=seed, image_size=(img_size, img_size),
```

```
    batch_size=batch_size, shuffle=True)
  val_ds = image_dataset_from_directory(
    val_dir, seed=seed, image_size=(img_size, img_size),
    batch_size=batch_size, shuffle=False)
  test_ds = image_dataset_from_directory(
    test_dir, seed=seed, image_size=(img_size, img_size),
    batch_size=batch_size, shuffle=False)

  class_names = train_ds.class_names
  print(f" Kelas terdeteksi: {class_names}")
  return train_ds, val_ds, test_ds, class_names



train_ds, val_ds, test_ds, class_names = load_datasets(
  cfg.data_dir, cfg.img_size, cfg.batch_size, cfg.seed
)


num_classes = len(class_names)
print(f"\nJumlah kelas: {num_classes}")

# === 3. DATA VISUALIZATION ===
def show_samples(dataset, class_names, n_images=9, title="Sample Images",
denormalize=False):
  plt.figure(figsize=(10,10))
  shown = 0
  for images, labels in dataset.take(1):
    for i in range(min(n_images, images.shape[0])):
      ax = plt.subplot(int(np.ceil(n_images/3)), 3, i+1)
      img = images[i].numpy()
      if denormalize:
        img = np.clip(img*255.0, 0, 255).astype("uint8")
      plt.imshow(img.astype("uint8"))
```

```
        plt.title(class_names[int(labels[i])])
        plt.axis("off")
        shown += 1
        if shown >= n_images:
          break
    plt.suptitle(title)
    plt.tight_layout()
    plt.show()

show_samples(train_ds, class_names, n_images=9, title="Raw Train Samples")

# === 4. DATA PREPARATION ===
AUTOTUNE = tf.data.AUTOTUNE
normalization_layer = layers.Rescaling(1./255)

def prepare(ds, shuffle=False):
    ds = ds.map(lambda x, y: (normalization_layer(x), y), num_parallel_calls=AUTOTUNE)
    if shuffle:
      ds = ds.shuffle(1024, seed=cfg.seed)
    ds = ds.prefetch(AUTOTUNE)
    return ds

train_ds_prep = prepare(train_ds, shuffle=True)
val_ds_prep   = prepare(val_ds, shuffle=False)
test_ds_prep = prepare(test_ds, shuffle=False)

show_samples(train_ds_prep, class_names, n_images=9, title="Normalized Train Samples",
denormalize=True)

# === 5. MODEL ARCHITECTURES ===
input_shape = (cfg.img_size, cfg.img_size, 3)

# Custom CNN
```

```
def build_custom_cnn(num_classes):
  inputs = layers.Input(shape=input_shape)
  x = layers.Conv2D(32, 3, padding='same', activation='relu')(inputs)
  x = layers.MaxPooling2D()(x)
  x = layers.Conv2D(64, 3, padding='same', activation='relu')(x)
  x = layers.MaxPooling2D()(x)
  x = layers.Conv2D(128, 3, padding='same', activation='relu')(x)
  x = layers.MaxPooling2D()(x)
  x = layers.Flatten()(x)
  x = layers.Dense(128, activation='relu')(x)
  x = layers.Dropout(0.4)(x)
  outputs = layers.Dense(num_classes, activation='softmax')(x)
  model = models.Model(inputs, outputs, name="CustomCNN")
  model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
  return model

models_builders = {
  "CustomCNN": build_custom_cnn,
}



for name, fn in models_builders.items():
  m = fn(num_classes)
  print("\n", name)
  m.summary()

# === 6. TRAINING ===
histories = {}
val_acc_records = {}
early_stop = callbacks.EarlyStopping(patience=3, restore_best_weights=True,
monitor='val_accuracy')
```

```
for name, builder in models_builders.items():
    print(f"\n===== Training {name} =====")
    tf.keras.backend.clear_session()

    model = builder(num_classes)
    history = model.fit(
        train_ds_prep,
        validation_data=val_ds_prep,
        epochs=cfg.epochs,
        callbacks=[early_stop],
        verbose=1
    )

    # Simpan hasil training
    histories[name] = history.history
    save_path = os.path.join(cfg.models_dir, f"{name}.h5")
    model.save(save_path)
    print(f" Model disimpan ke: {save_path}")
    val_acc_records[name] = float(np.max(history.history['val_accuracy']))

# === Simpan metrik training ===
rows = []
for name, hist in histories.items():
    for i in range(len(hist['accuracy'])):
        rows.append({
            "model": name,
            "epoch": i+1,
            "train_accuracy": hist['accuracy'][i],
            "val_accuracy": hist['val_accuracy'][i],
            "train_loss": hist['loss'][i],
```

```
        "val_loss": hist['val_loss'][i]
      })

hist_df = pd.DataFrame(rows)
csv_path = os.path.join(cfg.models_dir, f"training_metrics_{name}.csv")
hist_df.to_csv(csv_path, index=False)

print(f"\n Log disimpan ke: {csv_path}")
print("\n Validation Accuracy:")
for n, a in val_acc_records.items():
  print(f"{n:12s}: {a:.4f}")

# === 6b. GRAFIK METRIK ===
def plot_metric(df, metric, title=None):
  plt.figure(figsize=(8,5))
  for name in df['model'].unique():
    x = df[df['model']==name]['epoch']
    y = df[df['model']==name][metric]
    plt.plot(x, y, label=name)
  plt.xlabel("Epoch")
  plt.ylabel(metric.replace("_"," ").title())
  plt.title(title or f"Training Curves – {metric}")
  plt.legend()
  plt.grid(True)
  plt.show()

plot_metric(hist_df, "train_accuracy", "Train Accuracy")
plot_metric(hist_df, "val_accuracy", "Validation Accuracy")
plot_metric(hist_df, "train_loss", "Train Loss")
plot_metric(hist_df, "val_loss", "Validation Loss")

# === 7. EVALUATION (CustomCNN Only) ===
```

```
# Path model CustomCNN
model_path = os.path.join(cfg.models_dir, "CustomCNN.h5")
print(" Evaluating model:", model_path)

# Load model
model = tf.keras.models.load_model(model_path)

# Kumpulkan y_true dan y_pred dari test set
y_true, y_pred = [], []
for images, labels in test_ds_prep:
    probs = model.predict(images, verbose=0)
    preds = np.argmax(probs, axis=1)
    y_true.extend(labels.numpy().tolist())
    y_pred.extend(preds.tolist())

y_true = np.array(y_true)
y_pred = np.array(y_pred)

# Akurasi test
acc = (y_true == y_pred).mean()
print(f"\n Test Accuracy (CustomCNN): {acc:.4f}\n")

# Laporan klasifikasi
print(classification_report(y_true, y_pred, target_names=class_names))

# === Confusion Matrix ===
cm = confusion_matrix(y_true, y_pred, labels=list(range(num_classes)))

def plot_confusion_matrix(cm, classes, normalize=True, title='Confusion Matrix
(CustomCNN)'):
```

```
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1, keepdims=True)
    plt.figure(figsize=(6,5))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title(title)
    plt.colorbar()
    ticks = np.arange(len(classes))
    plt.xticks(ticks, classes, rotation=45, ha='right')
    plt.yticks(ticks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.show()

plot_confusion_matrix(cm, class_names, normalize=True)

# === 7b. VISUAL CONTOH PREDIKSI (test set) ===
def show_predictions(dataset, model, class_names, n_images=9):
    plt.figure(figsize=(10,10))
    count = 0
    for images, labels in dataset.take(1):
        probs = model.predict(images, verbose=0)
        preds = np.argmax(probs, axis=1)
        for i in range(min(n_images, images.shape[0])):
            ax = plt.subplot(int(np.ceil(n_images/3)), 3, i+1)
            img = np.clip(images[i].numpy()*255.0, 0, 255).astype("uint8")
```

```
        plt.imshow(img)
        true_label = class_names[int(labels[i])]
        pred_label = class_names[int(preds[i])]
        conf = np.max(probs[i])
        title = f"T:{true_label}\nP:{pred_label} ({conf:.2f})"
        color = "green" if true_label==pred_label else "red"
        plt.title(title, color=color)
        plt.axis("off")
        count += 1
        if count >= n_images:
          break
    plt.suptitle("Sample Predictions (Test)")
    plt.tight_layout()
    plt.show()

show_predictions(test_ds_prep, best_model, class_names, n_images=9)

# === 8. PREDIKSI 10 GAMBAR PER KELAS (predict_samples/<kelas>) ===
def load_predict_images(predict_dir, class_names, img_size=cfg.img_size, per_class=10):
    records = []
    predict_dir = Path(predict_dir)
    for cls in class_names:
      cls_dir = predict_dir / cls
      if not cls_dir.exists():
        print(f"[WARN] Folder tidak ditemukan: {cls_dir}")
        continue
      files = sorted([p for p in cls_dir.glob("*") if p.suffix.lower() in
                {".jpg",".jpeg",".png",".bmp",".webp"}])[:per_class]
      for fp in files:
        img = Image.open(fp).convert("RGB").resize((img_size, img_size))
        arr = np.array(img).astype("float32")/255.0
        records.append((cls, str(fp), arr))
```

```
    return records

def predict_records(model, records, class_names):
    results = []
    for true_cls, path, arr in records:
        x = np.expand_dims(arr, axis=0)
        prob = model.predict(x, verbose=0)[0]
        pred_idx = int(np.argmax(prob))
        pred_cls = class_names[pred_idx]
        conf = float(np.max(prob))
        results.append({
            "true_class": true_cls,
            "image_path": path,
            "pred_class": pred_cls,
            "confidence": conf
        })
    return pd.DataFrame(results)

records = load_predict_images(cfg.predict_dir, class_names, cfg.img_size, per_class=10)
pred_df = predict_records(best_model, records, class_names)
print(" Contoh hasil prediksi:")
display(pred_df.head(20))

# === 8b. VISUALISASI PREDIKSI PER KELAS (maks 10 gambar) ===
for cls in class_names:
    subset = [r for r in records if r[0]==cls][:10]
    if not subset:
        continue
    plt.figure(figsize=(12,6))
    for i, (true_cls, path, arr) in enumerate(subset):
        ax = plt.subplot(2,5,i+1)
        prob = best_model.predict(np.expand_dims(arr,0), verbose=0)[0]
```

```
        pred_idx = int(np.argmax(prob))
        conf = float(np.max(prob))
        pred_cls = class_names[pred_idx]
        plt.imshow((arr*255).astype("uint8"))
        title = f"T:{true_cls}\nP:{pred_cls} ({conf:.2f})"
        color = "green" if true_cls==pred_cls else "red"
        plt.title(title, color=color, fontsize=9)
        plt.axis("off")
    plt.suptitle(f" Predictions (CostumCNN) – Class '{cls}' (max 10 images)")
    plt.tight_layout()
    plt.show()

# === 9. EXPORT BEST MODEL + CATATAN VAL ACC ===
best_export_path = os.path.join(cfg.models_dir, "BestModel_CostumCNN_Pingouin.h5")
tf.keras.models.save_model(best_model, best_export_path)

print(" Best model exported to:", best_export_path)
```

AlexNet

```
import os, sys, json, glob, itertools, random
from dataclasses import dataclass
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers, callbacks
from tensorflow.keras.preprocessing import image_dataset_from_directory

from sklearn.metrics import confusion_matrix, classification_report
from PIL import Image

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
os.environ["TF_FORCE_GPU_ALLOW_GROWTH"] = "true"
os.environ["TF_ENABLE_ONEDNN_OPTS"] = "0"
os.environ["TF_DETERMINISTIC_OPS"] = "1"

import sys
import tensorflow as tf

print("Python:", sys.version)
print("TensorFlow:", tf._version_)
print("Num GPUs Available:", len(tf.config.list_physical_devices('GPU')))
print("Physical GPUs:", tf.config.list_physical_devices('GPU'))

# === 1. CONFIG ===
@dataclass
class Config:
```

```
  data_dir: str = "/home/jupyter-230712427/Projek UAS PMDPM_A_Pingouin/budaya_nusantara_foods/dataset"
  predict_dir: str = "/home/jupyter-230712427/Projek UAS PMDPM_A_Pingouin/budaya_nusantara_foods/predict_samples"
  img_size: int = 128
  batch_size: int = 8
  seed: int = 42
  epochs: int = 10
  models_dir: str = "/home/jupyter-230712427/Projek UAS PMDPM_A_Pingouin/budaya_nusantara_foods/models"

cfg = Config()
os.makedirs(cfg.models_dir, exist_ok=True)
cfg
```

Link Dataset Drive : https://drive.google.com/drive/folders/14htYGHZ86IXBaylLl8XvF1_r4HuDp8AC?usp=drive_link

```
# === 2. DATA LOADING ===
def load_datasets(data_dir: str, img_size: int, batch_size: int, seed: int=42):
  data_dir = Path(data_dir)
  train_dir = data_dir / "train"
  val_dir   = data_dir / "val"
  test_dir = data_dir / "test"

  print(f" Memuat dataset dari: {data_dir.resolve()}")
  print(f" Train: {train_dir}")
  print(f" Val : {val_dir}")
  print(f" Test : {test_dir}\n")

  train_ds = image_dataset_from_directory(
    train_dir, seed=seed, image_size=(img_size, img_size),
```

```
    batch_size=batch_size, shuffle=True)
  val_ds = image_dataset_from_directory(
    val_dir, seed=seed, image_size=(img_size, img_size),
    batch_size=batch_size, shuffle=False)
  test_ds = image_dataset_from_directory(
    test_dir, seed=seed, image_size=(img_size, img_size),
    batch_size=batch_size, shuffle=False)

  class_names = train_ds.class_names
  print(f" Kelas terdeteksi: {class_names}")
  return train_ds, val_ds, test_ds, class_names



train_ds, val_ds, test_ds, class_names = load_datasets(
  cfg.data_dir, cfg.img_size, cfg.batch_size, cfg.seed
)


num_classes = len(class_names)
print(f"\nJumlah kelas: {num_classes}")

# === 3. DATA VISUALIZATION ===
def show_samples(dataset, class_names, n_images=9, title="Sample Images",
denormalize=False):
  plt.figure(figsize=(10,10))
  shown = 0
  for images, labels in dataset.take(1):
    for i in range(min(n_images, images.shape[0])):
      ax = plt.subplot(int(np.ceil(n_images/3)), 3, i+1)
      img = images[i].numpy()
      if denormalize:
        img = np.clip(img*255.0, 0, 255).astype("uint8")
      plt.imshow(img.astype("uint8"))
```

```
        plt.title(class_names[int(labels[i])])
        plt.axis("off")
        shown += 1
        if shown >= n_images:
          break
    plt.suptitle(title)
    plt.tight_layout()
    plt.show()

show_samples(train_ds, class_names, n_images=9, title="Raw Train Samples")

# === 4. DATA PREPARATION ===
AUTOTUNE = tf.data.AUTOTUNE
normalization_layer = layers.Rescaling(1./255)

def prepare(ds, shuffle=False):
    ds = ds.map(lambda x, y: (normalization_layer(x), y), num_parallel_calls=AUTOTUNE)
    if shuffle:
      ds = ds.shuffle(1024, seed=cfg.seed)
    ds = ds.prefetch(AUTOTUNE)
    return ds

train_ds_prep = prepare(train_ds, shuffle=True)
val_ds_prep   = prepare(val_ds, shuffle=False)
test_ds_prep = prepare(test_ds, shuffle=False)

show_samples(train_ds_prep, class_names, n_images=9, title="Normalized Train Samples",
denormalize=True)

# === 5. MODEL ARCHITECTURES ===
input_shape = (cfg.img_size, cfg.img_size, 3)

# AlexNet
```

```
def build_alexnet(num_classes):
  inputs = layers.Input(shape=input_shape)
  x = layers.Conv2D(64, 11, strides=4, activation='relu')(inputs)
  x = layers.MaxPooling2D(pool_size=3, strides=2)(x)
  x = layers.Conv2D(128, 5, padding='same', activation='relu')(x)
  x = layers.MaxPooling2D(pool_size=3, strides=2)(x)
  x = layers.Conv2D(192, 3, padding='same', activation='relu')(x)
  x = layers.Conv2D(128, 3, padding='same', activation='relu')(x)
  x = layers.MaxPooling2D(pool_size=3, strides=2)(x)
  x = layers.Flatten()(x)
  x = layers.Dense(512, activation='relu')(x)
  x = layers.Dropout(0.5)(x)
  outputs = layers.Dense(num_classes, activation='softmax')(x)
  model = models.Model(inputs, outputs, name="AlexNet_Lite")
  model.compile(optimizer=optimizers.Adam(1e-4),
          loss='sparse_categorical_crossentropy',
          metrics=['accuracy'])
  return model

models_builders = {
  "AlexNet_Lite": build_alexnet,
}



for name, fn in models_builders.items():
  m = fn(num_classes)
  print("\n", name)
  m.summary()
# === 6. TRAINING ===
histories = {}
val_acc_records = {}
```

```
early_stop = callbacks.EarlyStopping(patience=3, restore_best_weights=True,
monitor='val_accuracy')


for name, builder in models_builders.items():
  print(f"\n===== Training {name} =====")
  tf.keras.backend.clear_session()

  model = builder(num_classes)
  history = model.fit(
    train_ds_prep,
    validation_data=val_ds_prep,
    epochs=cfg.epochs,
    callbacks=[early_stop],
    verbose=1
  )

  # Simpan hasil training
  histories[name] = history.history
  save_path = os.path.join(cfg.models_dir, f"{name}.h5")
  model.save(save_path)
  print(f" Model disimpan ke: {save_path}")
  val_acc_records[name] = float(np.max(history.history['val_accuracy']))

# === Simpan metrik training ===
rows = []
for name, hist in histories.items():
  for i in range(len(hist['accuracy'])):
    rows.append({
      "model": name,
      "epoch": i+1,
      "train_accuracy": hist['accuracy'][i],
```

```
        "val_accuracy": hist['val_accuracy'][i],
        "train_loss": hist['loss'][i],
        "val_loss": hist['val_loss'][i]
      })

hist_df = pd.DataFrame(rows)
csv_path = os.path.join(cfg.models_dir, "training_metrics_AlexNet.csv")
hist_df.to_csv(csv_path, index=False)

print(f"\n Log disimpan ke: {csv_path}")
print("\n Validation Accuracy:")
for n, a in val_acc_records.items():
  print(f"{n:12s}: {a:.4f}")

# === 6b. GRAFIK METRIK ===
def plot_metric(df, metric, title=None):
  plt.figure(figsize=(8,5))
  for name in df['model'].unique():
    x = df[df['model']==name]['epoch']
    y = df[df['model']==name][metric]
    plt.plot(x, y, label=name)
  plt.xlabel("Epoch")
  plt.ylabel(metric.replace("_"," ").title())
  plt.title(title or f"Training Curves – {metric}")
  plt.legend()
  plt.grid(True)
  plt.show()

plot_metric(hist_df, "train_accuracy", "Train Accuracy")
plot_metric(hist_df, "val_accuracy", "Validation Accuracy")
plot_metric(hist_df, "train_loss", "Train Loss")
```

```
plot_metric(hist_df, "val_loss", "Validation Loss")

# === 7. EVALUATION (AlexNet_Lite Only) ===

# Tentukan path model AlexNet_Lite
model_path = os.path.join(cfg.models_dir, "AlexNet_Lite.h5")
print(" Evaluating model:", model_path)

# Load model
alexnet_model = tf.keras.models.load_model(model_path)

# Kumpulkan label asli C prediksi dari test set
y_true, y_pred = [], []
for images, labels in test_ds_prep:
  probs = alexnet_model.predict(images, verbose=0)
  preds = np.argmax(probs, axis=1)
  y_true.extend(labels.numpy().tolist())
  y_pred.extend(preds.tolist())

# Ubah ke numpy array
y_true = np.array(y_true)
y_pred = np.array(y_pred)

# Hitung akurasi
acc = (y_true == y_pred).mean()
print(f"\n Test Accuracy (AlexNet_Lite): {acc:.4f}\n")

# Laporan klasifikasi
print(classification_report(y_true, y_pred, target_names=class_names))

# === Confusion Matrix ===
cm = confusion_matrix(y_true, y_pred, labels=list(range(num_classes)))
```

```
def plot_confusion_matrix(cm, classes, normalize=True, title='Confusion Matrix (AlexNet_Lite)'):
  if normalize:
    cm = cm.astype('float') / cm.sum(axis=1, keepdims=True)
  plt.figure(figsize=(6,5))
  plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
  plt.title(title)
  plt.colorbar()
  ticks = np.arange(len(classes))
  plt.xticks(ticks, classes, rotation=45, ha='right')
  plt.yticks(ticks, classes)
  fmt = '.2f' if normalize else 'd'
  thresh = cm.max() / 2.
  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
        horizontalalignment="center",
        color="white" if cm[i, j] > thresh else "black")
  plt.ylabel('True label')
  plt.xlabel('Predicted label')
  plt.tight_layout()
  plt.show()

# Tampilkan confusion matrix
plot_confusion_matrix(cm, class_names, normalize=True)

# === 7b. VISUAL CONTOH PREDIKSI (AlexNet_Lite – Test Set) ===
def show_predictions_alexnet(dataset, model, class_names, n_images=9):
  """
  Menampilkan contoh hasil prediksi dari model AlexNet_Lite pada test set.
  Gambar dengan judul hijau = prediksi benar, merah = salah.
  """
  plt.figure(figsize=(10, 10))
```

```
count = 0

# Ambil 1 batch dari test dataset
for images, labels in dataset.take(1):
  probs = model.predict(images, verbose=0)
  preds = np.argmax(probs, axis=1)

  for i in range(min(n_images, images.shape[0])):
    ax = plt.subplot(int(np.ceil(n_images / 3)), 3, i + 1)
    img = np.clip(images[i].numpy() * 255.0, 0, 255).astype("uint8")

    plt.imshow(img)
    true_label = class_names[int(labels[i])]
    pred_label = class_names[int(preds[i])]
    conf = np.max(probs[i])

    # Warna hijau jika benar, merah jika salah
    color = "green" if true_label == pred_label else "red"
    title = f"T:{true_label}\nP:{pred_label} ({conf:.2f})"
    plt.title(title, color=color, fontsize=10)
    plt.axis("off")

    count += 1
    if count >= n_images:
      break

plt.suptitle(" Sample Predictions (AlexNet_Lite – Test Set)", fontsize=14)
plt.tight_layout()
plt.show()
```

```
# === Jalankan visualisasi untuk AlexNet_Lite ===
show_predictions_alexnet(test_ds_prep, alexnet_model, class_names, n_images=9)
# === 8. PREDIKSI 10 GAMBAR PER KELAS (predict_samples/<kelas>) – AlexNet_Lite ===

def load_predict_images(predict_dir, class_names, img_size=cfg.img_size, per_class=10):
    """
    Memuat maksimal 10 gambar dari setiap kelas di folder predict_samples/<kelas>
    untuk diuji dengan model AlexNet_Lite.
    """
    records = []
    predict_dir = Path(predict_dir)

    for cls in class_names:
        cls_dir = predict_dir / cls
        if not cls_dir.exists():
            print(f"[¡    ] Folder tidak ditemukan: {cls_dir}")
            continue

        # Ambil maksimal 10 gambar (format umum)
        files = sorted([
            p for p in cls_dir.glob("*")
            if p.suffix.lower() in {".jpg", ".jpeg", ".png", ".bmp", ".webp"}
        ])[:per_class]

        for fp in files:
            img = Image.open(fp).convert("RGB").resize((img_size, img_size))
            arr = np.array(img).astype("float32") / 255.0
            records.append((cls, str(fp), arr))

    return records
```

```
def predict_records_alexnet(model, records, class_names):

    results = []
    for true_cls, path, arr in records:
        x = np.expand_dims(arr, axis=0)
        prob = model.predict(x, verbose=0)[0]
        pred_idx = int(np.argmax(prob))
        pred_cls = class_names[pred_idx]
        conf = float(np.max(prob))

        results.append({
            "true_class": true_cls,
            "image_path": path,
            "pred_class": pred_cls,
            "confidence": conf
        })

    return pd.DataFrame(results)


# === Jalankan Prediksi pada Folder predict_samples ===
records = load_predict_images(cfg.predict_dir, class_names, cfg.img_size, per_class=10)
pred_df = predict_records_alexnet(alexnet_model, records, class_names)

print("Contoh hasil prediksi AlexNet_Lite:")
display(pred_df.head(20))

# === 8b. VISUALISASI PREDIKSI PER KELAS (maks 10 gambar) ===
for cls in class_names:
    subset = [r for r in records if r[0]==cls][:10]
    if not subset:
```

```
    continue
plt.figure(figsize=(12,6))
for i, (true_cls, path, arr) in enumerate(subset):
    ax = plt.subplot(2,5,i+1)
    prob = best_model.predict(np.expand_dims(arr,0), verbose=0)[0]
    pred_idx = int(np.argmax(prob))
    conf = float(np.max(prob))
    pred_cls = class_names[pred_idx]
    plt.imshow((arr*255).astype("uint8"))
    title = f"T:{true_cls}\nP:{pred_cls} ({conf:.2f})"
    color = "green" if true_cls==pred_cls else "red"
    plt.title(title, color=color, fontsize=9)
    plt.axis("off")
plt.suptitle(f" Predictions (AlexNet) – Class '{cls}' (max 10 images)")
plt.tight_layout()
plt.show()
```

VGG-16

```
import os, sys, json, glob, itertools, random
from dataclasses import dataclass
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers, callbacks
from tensorflow.keras.preprocessing import image_dataset_from_directory

from sklearn.metrics import confusion_matrix, classification_report
from PIL import Image

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
os.environ["TF_FORCE_GPU_ALLOW_GROWTH"] = "true"
os.environ["TF_ENABLE_ONEDNN_OPTS"] = "0"
os.environ["TF_DETERMINISTIC_OPS"] = "1"

import sys
import tensorflow as tf

print("Python:", sys.version)
print("TensorFlow:", tf._version_)
print("Num GPUs Available:", len(tf.config.list_physical_devices('GPU')))
print("Physical GPUs:", tf.config.list_physical_devices('GPU'))

# === 1. CONFIG ===
@dataclass
class Config:
```

```
  data_dir: str = "/home/jupyter-230712427/Projek UAS PMDPM_A_Pingouin/budaya_nusantara_foods/dataset"
  predict_dir: str = "/home/jupyter-230712427/Projek UAS PMDPM_A_Pingouin/budaya_nusantara_foods/predict_samples"
  img_size: int = 128
  batch_size: int = 8
  seed: int = 42
  epochs: int = 10
  models_dir: str = "/home/jupyter-230712427/Projek UAS PMDPM_A_Pingouin/budaya_nusantara_foods/models"

cfg = Config()
os.makedirs(cfg.models_dir, exist_ok=True)
cfg
```

Link Dataset Drive : https://drive.google.com/drive/folders/14htYGHZ86IXBaylLl8XvF1_r4HuDp8AC?usp=drive_link

```
# === 2. DATA LOADING ===
def load_datasets(data_dir: str, img_size: int, batch_size: int, seed: int=42):
  data_dir = Path(data_dir)
  train_dir = data_dir / "train"
  val_dir   = data_dir / "val"
  test_dir = data_dir / "test"

  print(f" Memuat dataset dari: {data_dir.resolve()}")
  print(f" Train: {train_dir}")
  print(f" Val : {val_dir}")
  print(f" Test : {test_dir}\n")

  train_ds = image_dataset_from_directory(
    train_dir, seed=seed, image_size=(img_size, img_size),
```

```
    batch_size=batch_size, shuffle=True)
  val_ds = image_dataset_from_directory(
    val_dir, seed=seed, image_size=(img_size, img_size),
    batch_size=batch_size, shuffle=False)
  test_ds = image_dataset_from_directory(
    test_dir, seed=seed, image_size=(img_size, img_size),
    batch_size=batch_size, shuffle=False)

  class_names = train_ds.class_names
  print(f" Kelas terdeteksi: {class_names}")
  return train_ds, val_ds, test_ds, class_names



train_ds, val_ds, test_ds, class_names = load_datasets(
  cfg.data_dir, cfg.img_size, cfg.batch_size, cfg.seed
)


num_classes = len(class_names)
print(f"\nJumlah kelas: {num_classes}")

# === 3. DATA VISUALIZATION ===
def show_samples(dataset, class_names, n_images=9, title="Sample Images",
denormalize=False):
  plt.figure(figsize=(10,10))
  shown = 0
  for images, labels in dataset.take(1):
    for i in range(min(n_images, images.shape[0])):
      ax = plt.subplot(int(np.ceil(n_images/3)), 3, i+1)
      img = images[i].numpy()
      if denormalize:
        img = np.clip(img*255.0, 0, 255).astype("uint8")
      plt.imshow(img.astype("uint8"))
```

```
        plt.title(class_names[int(labels[i])])
        plt.axis("off")
        shown += 1
        if shown >= n_images:
          break
    plt.suptitle(title)
    plt.tight_layout()
    plt.show()

show_samples(train_ds, class_names, n_images=9, title="Raw Train Samples")

# === 4. DATA PREPARATION ===
AUTOTUNE = tf.data.AUTOTUNE
normalization_layer = layers.Rescaling(1./255)

def prepare(ds, shuffle=False):
    ds = ds.map(lambda x, y: (normalization_layer(x), y), num_parallel_calls=AUTOTUNE)
    if shuffle:
      ds = ds.shuffle(1024, seed=cfg.seed)
    ds = ds.prefetch(AUTOTUNE)
    return ds

train_ds_prep = prepare(train_ds, shuffle=True)
val_ds_prep   = prepare(val_ds, shuffle=False)
test_ds_prep = prepare(test_ds, shuffle=False)

show_samples(train_ds_prep, class_names, n_images=9, title="Normalized Train Samples",
denormalize=True)

# === 5. MODEL ARCHITECTURES ===
input_shape = (cfg.img_size, cfg.img_size, 3)

# VGG-16
```

```
def build_vgg16_like(num_classes):
  def block(x, f):
    x = layers.Conv2D(f, 3, padding='same', activation='relu')(x)
    x = layers.Conv2D(f, 3, padding='same', activation='relu')(x)
    return layers.MaxPooling2D(2)(x)
  inputs = layers.Input(shape=input_shape)
  x = block(inputs, 32)
  x = block(x, 64)
  x = block(x, 128)
  x = layers.Flatten()(x)
  x = layers.Dense(256, activation='relu')(x)
  x = layers.Dropout(0.5)(x)
  outputs = layers.Dense(num_classes, activation='softmax')(x)
  model = models.Model(inputs, outputs, name="VGG16_Lite")
  model.compile(optimizer=optimizers.Adam(1e-4),
          loss='sparse_categorical_crossentropy',
          metrics=['accuracy'])
  return model

models_builders = {
  "VGG16_Lite": build_vgg16_like,
}



for name, fn in models_builders.items():
  m = fn(num_classes)
  print("\n", name)
  m.summary()
# === 6. TRAINING ===
histories = {}
val_acc_records = {}
```

```
early_stop = callbacks.EarlyStopping(patience=3, restore_best_weights=True,
monitor='val_accuracy')


for name, builder in models_builders.items():
  print(f"\n===== Training {name} =====")
  tf.keras.backend.clear_session()

  model = builder(num_classes)
  history = model.fit(
    train_ds_prep,
    validation_data=val_ds_prep,
    epochs=cfg.epochs,
    callbacks=[early_stop],
    verbose=1
  )

  # Simpan hasil training
  histories[name] = history.history
  save_path = os.path.join(cfg.models_dir, f"{name}.h5")
  model.save(save_path)
  print(f" Model disimpan ke: {save_path}")
  val_acc_records[name] = float(np.max(history.history['val_accuracy']))

# === Simpan metrik training ===
rows = []
for name, hist in histories.items():
  for i in range(len(hist['accuracy'])):
    rows.append({
      "model": name,
      "epoch": i+1,
      "train_accuracy": hist['accuracy'][i],
```

```
        "val_accuracy": hist['val_accuracy'][i],
        "train_loss": hist['loss'][i],
        "val_loss": hist['val_loss'][i]
      })

hist_df = pd.DataFrame(rows)
csv_path = os.path.join(cfg.models_dir, "training_metrics_VGG16.csv")
hist_df.to_csv(csv_path, index=False)

print(f"\n Log disimpan ke: {csv_path}")
print("\n Validation Accuracy:")
for n, a in val_acc_records.items():
  print(f"{n:12s}: {a:.4f}")

# === 6b. GRAFIK METRIK ===
def plot_metric(df, metric, title=None):
  plt.figure(figsize=(8,5))
  for name in df['model'].unique():
    x = df[df['model']==name]['epoch']
    y = df[df['model']==name][metric]
    plt.plot(x, y, label=name)
  plt.xlabel("Epoch")
  plt.ylabel(metric.replace("_"," ").title())
  plt.title(title or f"Training Curves – {metric}")
  plt.legend()
  plt.grid(True)
  plt.show()

plot_metric(hist_df, "train_accuracy", "Train Accuracy")
plot_metric(hist_df, "val_accuracy", "Validation Accuracy")
plot_metric(hist_df, "train_loss", "Train Loss")
```

```
plot_metric(hist_df, "val_loss", "Validation Loss")

# === 7. EVALUATION (VGG16_Lite Only) ===

# Tentukan path model VGG16_Lite
model_path = os.path.join(cfg.models_dir, "VGG16_Lite.h5")
print("Evaluating model:", model_path)

# Load model
vgg16_model = tf.keras.models.load_model(model_path)

# Kumpulkan label asli C prediksi dari test set
y_true, y_pred = [], []
for images, labels in test_ds_prep:
    probs = vgg16_model.predict(images, verbose=0)
    preds = np.argmax(probs, axis=1)
    y_true.extend(labels.numpy().tolist())
    y_pred.extend(preds.tolist())

# Ubah ke numpy array
y_true = np.array(y_true)
y_pred = np.array(y_pred)

# Hitung akurasi
acc = (y_true == y_pred).mean()
print(f"\n ✅ Test Accuracy (VGG16_Lite): {acc:.4f}\n")

# Laporan klasifikasi lengkap
print(classification_report(y_true, y_pred, target_names=class_names))

# === Confusion Matrix ===
```

```
cm = confusion_matrix(y_true, y_pred, labels=list(range(num_classes)))

def plot_confusion_matrix(cm, classes, normalize=True, title='Confusion Matrix (VGG16_Lite)'):
  if normalize:
    cm = cm.astype('float') / cm.sum(axis=1, keepdims=True)
  plt.figure(figsize=(6,5))
  plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
  plt.title(title)
  plt.colorbar()
  ticks = np.arange(len(classes))
  plt.xticks(ticks, classes, rotation=45, ha='right')
  plt.yticks(ticks, classes)
  fmt = '.2f' if normalize else 'd'
  thresh = cm.max() / 2.
  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
        horizontalalignment="center",
        color="white" if cm[i, j] > thresh else "black")
  plt.ylabel('True Label')
  plt.xlabel('Predicted Label')
  plt.tight_layout()
  plt.show()

plot_confusion_matrix(cm, class_names, normalize=True)

# === 7b. VISUAL CONTOH PREDIKSI (VGG16_Lite – Test Set) ===
def show_predictions_vgg16(dataset, model, class_names, n_images=9):
  plt.figure(figsize=(10,10))
  count = 0
  for images, labels in dataset.take(1):
    # Prediksi probabilitas
```

```
    probs = model.predict(images, verbose=0)
    preds = np.argmax(probs, axis=1)

    for i in range(min(n_images, images.shape[0])):
      ax = plt.subplot(int(np.ceil(n_images/3)), 3, i+1)

      # Denormalisasi gambar untuk ditampilkan
      img = np.clip(images[i].numpy() * 255.0, 0, 255).astype("uint8")
      plt.imshow(img)

      # Ambil label sebenarnya dan hasil prediksi
      true_label = class_names[int(labels[i])]
      pred_label = class_names[int(preds[i])]
      conf = np.max(probs[i])

      # Warna hijau jika benar, merah jika salah
      color = "green" if true_label == pred_label else "red"
      plt.title(f"T:{true_label}\nP:{pred_label} ({conf:.2f})", color=color)
      plt.axis("off")

      count += 1
      if count >= n_images:
        break

  plt.suptitle(" Sample Predictions (VGG16_Lite – Test Set)")
  plt.tight_layout()
  plt.show()


# Jalankan visualisasi
```

```
show_predictions_vgg16(test_ds_prep, vgg16_model, class_names, n_images=9)

# === 8. PREDIKSI 10 GAMBAR PER KELAS (VGG16_Lite – predict_samples/<kelas>) ===
def load_predict_images(predict_dir, class_names, img_size=cfg.img_size, per_class=10):
    """
    Memuat maksimal 10 gambar per kelas dari folder predict_samples/<kelas>
    dan mengubahnya menjadi array (float32, 0-1).
    """
    records = []
    predict_dir = Path(predict_dir)
    for cls in class_names:
        cls_dir = predict_dir / cls
        if not cls_dir.exists():
            print(f"[WARN] Folder tidak ditemukan: {cls_dir}")
            continue

        # Ambil maksimal 10 gambar pertama dengan ekstensi valid
        files = sorted([p for p in cls_dir.glob("*")
                        if p.suffix.lower() in {".jpg", ".jpeg", ".png", ".bmp", ".webp"}])[:per_class]
        for fp in files:
            img = Image.open(fp).convert("RGB").resize((img_size, img_size))
            arr = np.array(img).astype("float32") / 255.0
            records.append((cls, str(fp), arr))
    return records


def predict_records_vgg16(model, records, class_names):
    """
    Melakukan prediksi untuk seluruh record gambar pada VGG16_Lite
    """
    results = []
    for true_cls, path, arr in records:
```

```
        x = np.expand_dims(arr, axis=0)
        prob = model.predict(x, verbose=0)[0]
        pred_idx = int(np.argmax(prob))
        pred_cls = class_names[pred_idx]
        conf = float(np.max(prob))
        results.append({
            "true_class": true_cls,
            "image_path": path,
            "pred_class": pred_cls,
            "confidence": conf
        })
    return pd.DataFrame(results)


# === Jalankan prediksi dengan model VGG16_Lite ===
records = load_predict_images(cfg.predict_dir, class_names, cfg.img_size, per_class=10)
pred_df_vgg16 = predict_records_vgg16(vgg16_model, records, class_names)

print(" Contoh hasil prediksi (VGG16_Lite):")
display(pred_df_vgg16.head(20))
# === 8b. VISUALISASI PREDIKSI PER KELAS (VGG16_Lite – maks 10 gambar) ===
for cls in class_names:
    subset = [r for r in records if r[0] == cls][:10]
    if not subset:
        continue

    plt.figure(figsize=(12, 6))
    for i, (true_cls, path, arr) in enumerate(subset):
        ax = plt.subplot(2, 5, i + 1)

        # Prediksi dengan model VGG16_Lite
```

```
    prob = vgg16_model.predict(np.expand_dims(arr, 0), verbose=0)[0]
    pred_idx = int(np.argmax(prob))
    conf = float(np.max(prob))
    pred_cls = class_names[pred_idx]

    # Tampilkan gambar
    plt.imshow((arr * 255).astype("uint8"))

    # Buat judul prediksi
    title = f"T:{true_cls}\nP:{pred_cls} ({conf:.2f})"
    color = "green" if true_cls == pred_cls else "red"
    plt.title(title, color=color, fontsize=9)
    plt.axis("off")

plt.suptitle(f" Predictions (VGG16_Lite) – Class '{cls}' (max 10 images)")
plt.tight_layout()
plt.show()
```

MainStreamlit.py

```
import streamlit as st
import numpy as np
import pandas as pd
from pathlib import
Path
from PIL import Image
import plotly.express as
px

# === IMPORT PINTAR
(HIBRID) ===
try:
    import
tflite_runtime.interpret
er as tflite
    print("Berhasil impor
'tflite_runtime' (mode
server/deploy)")
except ImportError:
    import tensorflow as
tf
    tflite = tf.lite
    print("Gagal impor
'tflite_runtime',
menggunakan 'tf.lite'
(mode lokal)")
# === SELESAI IMPORT
HIBRID ===



# === FUNGSI UNTUK
MEMUAT CSS ===
```

```
def
load_css(file_name):
    try:
        with
open(file_name) as f:

st.markdown(f"<style>{f
.read()}</style>",
unsafe_allow_html=Tru
e)
    except
FileNotFoundError:
        st.error(f"File CSS
'{file_name}' tidak
ditemukan.")



# === KONFIGURASI
APLIKASI ===
st.set_page_config(
    page_title="🍜
Klasifikasi Makanan
Nusantara",
    page_icon="🍜",
    layout="wide",

initial_sidebar_state="e
xpanded",
)

# === CSS FILE ===
load_css("style.css")
```

```
# === NAMA KELAS DAN
EMOJI ===
def get_class_names():
    return ["gudeg",
"papeda", "pempek",
"rendang"]


def
get_food_emoji(pred_cl
ass):
    emojis = {"gudeg":
"🍛", "papeda": "🥣",
"pempek": "🐟",
"rendang": "🍖"}
    return
emojis.get(pred_class,
"❓")


# === FUNGSI MODEL
(TFLITE) ===
MODEL_PATH =
"BestModel_CostumCNN.
tflite"


@st.cache_resource
def load_model():
    """Memuat TFLite
```

```
interpreter dan
mengalokasikan
tensor."""
    try:
        interpreter =
tflite.Interpreter(model
_path=MODEL_PATH)

interpreter.allocate_ten
sors()
        input_details =
interpreter.get_input_d
etails()
        output_details =
interpreter.get_output_
details()
        return interpreter,
input_details,
output_details
    except Exception as
e:
        st.error(f"Gagal
memuat model TFLite:
{e}")
        st.error(f"Pastikan
file '{MODEL_PATH}' ada
di folder yang sama.")
        return None,
None, None



def
preprocess_image(imag
```

```
e, target_size=(128,
128)):
    """Pre-processing
gambar agar sesuai
dengan input model."""
    if image.mode !=
"RGB":
        image =
image.convert("RGB")
    img =
image.resize(target_siz
e)
    img_array =
np.array(img)
    img_array =
img_array / 255.0
    img_array =
np.expand_dims(img_ar
ray, axis=0)
    img_array =
img_array.astype(np.flo
at32)
    return img_array



# === UI APLIKASI ===
# --- HEADER ---
st.markdown(
    "<h1 class='main-
header'>🍜 Klasifikasi
Makanan
Nusantara</h1>",
```

```
unsafe_allow_html=Tru
e,
)
st.markdown(
    "<p class='sub-
header'>Unggah gambar
makanan Anda 📸 dan
biarkan AI kami
menebaknya!</p>",

unsafe_allow_html=Tru
e,
)

# --- SIDEBAR ---
with st.sidebar:
    st.title("Tentang
Proyek")
    st.markdown(
        """
        Ini adalah
prototipe aplikasi web
untuk Ujian Tengah
Semester (UTS) .

        - **Model:**
Custom CNN (TFLite)
        - **Tujuan:**
Klasifikasi 4 Makanan
Nusantara.
        - **Author:** Beny,
```

```
Denis, Renaldi
        """
    )
    st.divider()
    st.info("Akurasi
model mungkin tidak
100%.")

# --- MAIN CONTENT ---
col1, col2 =
st.columns([1, 1])
prob = None
interpreter,
input_details,
output_details =
load_model()

with col1:
    st.header("1. Unggah
Gambar Anda 🖼️")
    uploaded_file =
st.file_uploader(
        "Pilih file
gambar...",
        type=["jpg",
"jpeg", "png"],

label_visibility="collaps
ed",
    )

    with st.expander("ℹ️
```

```
Info Kelas yang
Dilatih"):
        st.write(
            """
            Model ini dilatih
untuk mengenali 4 kelas
makanan:
            - 🧿 **gudeg**
            - 🥣 **papeda**
            - 🐟 **pempek**
            - 🍖
**rendang**
            """
        )

    if uploaded_file is
not None:
        image =
Image.open(uploaded_fi
le)
        st.image(image,
caption="Gambar yang
Diunggah",
use_column_width=True
)

with col2:
    st.header("2. Hasil
Prediksi AI 🧠")

    if uploaded_file is
not None:
```

```
        if interpreter:
            with
st.spinner("AI sedang
berpikir... 🤖"):

                input_shape
=
input_details[0]["shape"
]
                target_size =
(input_shape[1],
input_shape[2])

                img_array =
preprocess_image(imag
e,
target_size=target_size)


interpreter.set_tensor(i
nput_details[0]["index"]
, img_array)

interpreter.invoke()
                prob =
interpreter.get_tensor(
output_details[0]["index
"])[0]

                pred_index =
np.argmax(prob)
                class_names
= get_class_names()
```

```
            if len(prob)
!= len(class_names):
                st.error(
                    f"Error:
Model output
{len(prob)} kelas, tapi
daftar kelas punya
{len(class_names)}."
                )
                prob =
None
            else:
                pred_class
=
class_names[pred_index
]
                confidence
= np.max(prob) * 100
                emoji =
get_food_emoji(pred_cl
ass)

                st.metric(

label="Hasil Tebakan
AI:",

value=f"{emoji}
{pred_class.capitalize()}
",

delta=f"Keyakinan
```

```
{confidence:.2f}%",

delta_color="normal",
                )

                if
confidence > 80:

st.success("🎯 Prediksi
sangat yakin!")
                elif
confidence > 60:

st.info("👍 Prediksi
cukup yakin")
                elif
confidence > 40:

st.warning("⚠️ Prediksi
kurang yakin")
                else:

st.error("❌ Prediksi
sangat rendah, coba
gambar lain")
        else:
            st.error("Model
tidak dapat dimuat. Cek
log.")
    else:
        st.info("Silakan
unggah gambar di
```

```
sebelah kiri untuk
melihat hasil prediksi.")



# === GRAFIK
PROBABILITAS ===
if prob is not None:
    st.divider()
    st.subheader(" 📊
Distribusi Probabilitas")

    class_names =
get_class_names()
    prob_data =
pd.DataFrame(
        {"Kelas":
class_names,
"Probabilitas": prob *
100}

).sort_values("Probabilit
as", ascending=False)

    fig = px.bar(
        prob_data,
        x="Kelas",
        y="Probabilitas",

color="Probabilitas",

color_continuous_scale=
"RdYlGn",
```

```
        text_auto=".2f",
        title="Keyakinan
Model untuk Setiap
Kelas (%)",
    )

fig.update_layout(yaxis
_title="Probabilitas (%)")
    st.plotly_chart(fig,
use_container_width=Tr
ue)
```