Predict.py

import os

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

import cv2

from PIL import Image

from gradcam\_multi import generate\_gradcam\_explanations

from config import Config

def load\_and\_preprocess(image\_path):

    original\_img = tf.keras.utils.load\_img(image\_path, target\_size=(224, 224))

    img\_array = tf.keras.utils.img\_to\_array(original\_img)

    img\_array = np.expand\_dims(img\_array, axis=0) / 255.0

    processed\_tensor = tf.convert\_to\_tensor(img\_array, dtype=tf.float32)

    return original\_img, processed\_tensor

def shrink\_image(image, scale=0.85):

    """Shrink image without changing figure size or spacing."""

    if isinstance(image, Image.Image):

        image = np.array(image)  # Convert PIL to NumPy

    h, w = image.shape[:2]

    new\_w, new\_h = int(w \* scale), int(h \* scale)

    return cv2.resize(image, (new\_w, new\_h), interpolation=cv2.INTER\_AREA)

def interpret\_predictions(preds):

    # 1) Organic / Inorganic

    is\_organic\_prob = preds['is\_organic'][0][0]

    label = "Organic" if is\_organic\_prob > 0.5 else "Inorganic"

    confidence = is\_organic\_prob if is\_organic\_prob > 0.5 else 1 - is\_organic\_prob

    confidence\_pct = confidence \* 100

    # 2) Quality grade

    qg\_probs = preds['quality\_grade'][0]

    qg\_idx = np.argmax(qg\_probs)

    quality\_map = {0: "Bad", 1: "Medium", 2: "Good"}

    quality\_label = quality\_map.get(qg\_idx, "Unknown")

    quality\_conf\_pct = qg\_probs[qg\_idx] \* 100

    # 3) Size

    size\_probs = preds['size'][0]

    size\_idx = np.argmax(size\_probs)

    size\_map = {0: "Small", 1: "Medium", 2: "Big"}

    size\_label = size\_map.get(size\_idx, "Unknown")

    size\_conf\_pct = size\_probs[size\_idx] \* 100

    # 4) Shininess

    shiny\_prob = preds['shininess'][0][0]

    shiny\_label = "Shiny" if shiny\_prob > 0.5 else "Dull"

    shiny\_conf\_pct = shiny\_prob \* 100 if shiny\_label == "Shiny" else (1 - shiny\_prob) \* 100

    # 5) Darkspots

    darkspot\_prob = preds['darkspots'][0][0]

    darkspot\_label = "Yes" if darkspot\_prob > 0.5 else "No"

    darkspot\_conf\_pct = darkspot\_prob \* 100 if darkspot\_label == "Yes" else (1 - darkspot\_prob) \* 100

    # 6) Shape irregularity

    shape\_probs = preds['shape\_irregularity'][0]

    shape\_idx = np.argmax(shape\_probs)

    shape\_map = {0: "Normal", 1: "Some irregularity", 2: "Lots of irregularity"}

    shape\_label = shape\_map.get(shape\_idx, "Unknown")

    shape\_conf\_pct = shape\_probs[shape\_idx] \* 100

    # Bold headings + bold labels

    pred\_text = (

        r"$\mathbf{PREDICTION}$" + "\n"

        + r"$\mathbf{Label:}$" + f" {label} ({confidence\_pct:.1f}%)\n"

        + r"$\mathbf{Quality:}$" + f" {quality\_label} ({quality\_conf\_pct:.1f}%)\n"

        + r"$\mathbf{Size:}$" + f" {size\_label} ({size\_conf\_pct:.1f}%)\n"

        + r"$\mathbf{Shine:}$" + f" {shiny\_label} ({shiny\_conf\_pct:.1f}%)\n"

        + r"$\mathbf{Dark\ Spots:}$" + f" {darkspot\_label} ({darkspot\_conf\_pct:.1f}%)\n"

        + r"$\mathbf{Shape:}$" + f" {shape\_label} ({shape\_conf\_pct:.1f}%)"

    )

    classification\_notes = []

    if shape\_idx == 2:

        classification\_notes.append("Shape asymmetry detected")

    if shiny\_label == "Dull":

        classification\_notes.append("Dull shine")

    if darkspot\_label == "Yes":

        classification\_notes.append("Dark spots detected")

    if len(classification\_notes) == 0:

        classification\_notes.append("No obvious defects")

    quality\_notes = []

    if quality\_label == "Good":

        quality\_notes.append("Smooth, glossy surface")

    elif quality\_label == "Medium":

        quality\_notes.append("Some wrinkling visible")

    else:

        quality\_notes.append("Poor surface quality")

        quality\_notes.append("Possibly rotten or damaged")

    class\_text = r"$\mathbf{CLASSIFICATION}$" + "\n" + "\n".join(f"- {line}" for line in classification\_notes)

    quality\_text = r"$\mathbf{QUALITY}$" + "\n" + "\n".join(f"- {line}" for line in quality\_notes)

    return pred\_text, class\_text, quality\_text

def plot\_results(original\_img, class\_overlay, quality\_overlay, pred\_text, class\_text, quality\_text):

    fig, axs = plt.subplots(1, 3, figsize=(18, 6))

    axs[0].imshow(shrink\_image(original\_img, scale=0.85))

    axs[0].axis('off')

    axs[0].set\_title(r"$\mathbf{Original\ Image}$", fontsize=14)

    axs[0].text(0.3, -0.08, pred\_text,

                transform=axs[0].transAxes,

                fontsize=10, va='top', ha='center', wrap=True)

    axs[1].imshow(shrink\_image(class\_overlay, scale=0.85))

    axs[1].axis('off')

    axs[1].set\_title(r"$\mathbf{Classification\ Grad\text{-}CAM}$", fontsize=14)

    axs[1].text(0.3, -0.08, class\_text,

                transform=axs[1].transAxes,

                fontsize=10, va='top', ha='center', wrap=True)

    axs[2].imshow(shrink\_image(quality\_overlay, scale=0.85))

    axs[2].axis('off')

    axs[2].set\_title(r"$\mathbf{Quality\ Grad\text{-}CAM}$", fontsize=14)

    axs[2].text(0.3, -0.08, quality\_text,

                transform=axs[2].transAxes,

                fontsize=10, va='top', ha='center', wrap=True)

    plt.tight\_layout()

    plt.show()

def predict\_and\_explain(image\_path, model):

    original\_img, processed\_tensor = load\_and\_preprocess(image\_path)

    preds = model(processed\_tensor)

    preds = {k: v.numpy() for k, v in preds.items()}

    # Select heads for Grad-CAM visualization

    class\_head = 'is\_organic'

    quality\_head = 'quality\_grade'

    class\_overlay, quality\_overlay, classification\_text, quality\_text = generate\_gradcam\_explanations(

    model, processed\_tensor, class\_head, quality\_head

    )

    pred\_text, class\_text, quality\_text = interpret\_predictions(preds)

    return original\_img, class\_overlay, quality\_overlay, pred\_text, class\_text, quality\_text

if \_\_name\_\_ == "\_\_main\_\_":

    test\_image\_path = os.path.join(Config.TEST\_IMAGES\_DIR, "test3.jpg")

    model = tf.keras.models.load\_model(os.path.join(Config.MODEL\_SAVE\_PATH, "model\_final.keras"))

    print("Model loaded.")

    print(f"Processing: {test\_image\_path}")

    results = predict\_and\_explain(test\_image\_path, model)

    plot\_results(\*results)

gradcam\_multi.py

import tensorflow as tf

import numpy as np

import cv2

def compute\_gradcam(model, processed\_tensor, head\_name):

    model\_input = model.input

    last\_conv\_layer = model.get\_layer('Conv\_1')

    last\_conv\_output = last\_conv\_layer.output

    head\_output = model.get\_layer(head\_name).output

    grad\_model = tf.keras.Model(inputs=model\_input, outputs=[last\_conv\_output, head\_output])

    with tf.GradientTape() as tape:

        conv\_outputs, predictions = grad\_model(processed\_tensor)

        tape.watch(conv\_outputs)

        if len(predictions.shape) == 2:

            class\_idx = tf.argmax(predictions[0])

            target = predictions[:, class\_idx]

        else:

            target = predictions[:, 0]

    grads = tape.gradient(target, conv\_outputs)

    pooled\_grads = tf.reduce\_mean(grads, axis=(0, 1, 2))

    conv\_outputs = conv\_outputs[0]

    heatmap = tf.reduce\_sum(tf.multiply(pooled\_grads, conv\_outputs), axis=-1)

    heatmap = tf.maximum(heatmap, 0) / (tf.reduce\_max(heatmap) + 1e-10)

    return heatmap.numpy()

def overlay\_heatmap(heatmap, image, alpha=0.4, colormap=cv2.COLORMAP\_JET):

    heatmap = cv2.resize(heatmap, (image.shape[1], image.shape[0]))

    heatmap = np.uint8(255 \* heatmap)

    heatmap\_color = cv2.applyColorMap(heatmap, colormap)

    overlayed\_img = heatmap\_color \* alpha + image

    overlayed\_img = np.clip(overlayed\_img, 0, 255).astype(np.uint8)

    return overlayed\_img

def generate\_gradcam\_explanations(model, processed\_tensor, class\_head, quality\_head):

    class\_heatmap = compute\_gradcam(model, processed\_tensor, class\_head)

    quality\_heatmap = compute\_gradcam(model, processed\_tensor, quality\_head)

    # Convert processed tensor (normalized) back to uint8 image for overlay

    img\_for\_overlay = (processed\_tensor[0].numpy() \* 255).astype(np.uint8)

    class\_overlay = overlay\_heatmap(class\_heatmap, img\_for\_overlay)

    quality\_overlay = overlay\_heatmap(quality\_heatmap, img\_for\_overlay)

    classification\_text = f"Grad-CAM for head: {class\_head}"

    quality\_text = f"Grad-CAM for head: {quality\_head}"

    return class\_overlay, quality\_overlay, classification\_text, quality\_text