Data Augmentation

ICS3206 - Course Project (Jupyter Notebook #1)

Name: Andrea Filiberto Lucas
ID No: 0279704L

Importing Necessary Libraries

Essential libraries are imported in this section. ANSI escape codes are defined to enhance the readability of console messages with colored text. A mapping between PyPI package names and their corresponding import names is established to facilitate automated package management. Functions are provided to check for the presence of required packages and install any that are missing using `pip`. Once all dependencies are ensured, necessary modules such as OpenCV, NumPy, Pillow, Matplotlib, IPyWidgets, and IPython are imported to support image processing, concurrent execution, data manipulation, and visualization tasks.

```
############################
#         IMPORTS          #
############################

import importlib
import subprocess
import sys

# ANSI escape codes for colored text (optional, for better
readability)
GREEN = "\033[92m"
RED = "\033[91m"
RESET = "\033[0m"

# Mapping of PyPI package names to their import names
# Format: 'package-name': 'import-name'
# For submodules, use the top-level package name as the import name
packages = {
    'opencv-python': 'cv2',
    'numpy': 'numpy',
    'Pillow': 'PIL',
    'matplotlib': 'matplotlib',
    'ipywidgets': 'ipywidgets',  # Added ipywidgets
    'IPython': 'IPython',        # Added IPython
    'tqdm': 'tqdm',              # Added tqdm
}
```

```
def install_package(package_name: str) -> None:
    """
    Install a package using pip.

    Args:
        package_name (str): The name of the package to install.
    """
    try:
        print(f"Installing package: {package_name}")
        # Use subprocess to run the pip install command
        subprocess.check_call([sys.executable, "-m", "pip", "install",
package_name])
        print(f"{GREEN}Successfully installed {package_name}.{RESET}")
    except subprocess.CalledProcessError as e:
        # Print the error message in red
        print(f"{RED}Failed to install package {package_name}. Error:
{e}{RESET}")
        sys.exit(1)

def check_and_install_packages(packages_dict: dict) -> None:
    """
    Check if the packages are installed, and install them if they are
missing.

    Args:
        packages_dict (dict): A dictionary mapping package names to
their import names.
    """
    for package, import_name in packages_dict.items():
        try:
            # Attempt to import the package
            importlib.import_module(import_name.split('.')[0])
            print(f"Package '{package}' is already installed.")
        except ImportError:
            print(f"Package '{package}' is not installed. Installing
now...")
            install_package(package)

# Check and install the required packages
check_and_install_packages(packages)

# Now, import the packages after ensuring they are installed
import os
import cv2
import numpy as np
from PIL import Image, ImageEnhance
from matplotlib import pyplot as plt
from pathlib import Path
import random
```

```
# Additional Imports
import ipywidgets as widgets
from IPython.display import display, clear_output
from tqdm import tqdm  # Import tqdm for progress bar

# Print the success message in green
print(f"{GREEN}All required packages are installed and imported
successfully.{RESET}")
```

```
Package 'opencv-python' is already installed.
Package 'numpy' is already installed.
Package 'Pillow' is already installed.
Package 'matplotlib' is already installed.
Package 'ipywidgets' is already installed.
Package 'IPython' is already installed.
Package 'tqdm' is already installed.
All required packages are installed and imported successfully.
```

Directory Selection

This section enables the selection of input and output directories using an interactive dropdown widget. Users can choose a specific constellation directory or select "All" to include all available directories. Upon selection, the script verifies the existence and non-emptiness of the chosen input directories and creates the corresponding output directories if they do not already exist.

```
############################
#     DIRECTORY SELECTION   #
############################

# List of available directory options with "All" as the first option
available_dirs = [
    "All",
    "CoronaBorealis", "Lacerta", "LeoMinor", "Lepus", "Norma",
    "Pisces", "Sextans", "Virgo"
]

# Create a Dropdown widget for directory selection
dir_dropdown = widgets.Dropdown(
    options=available_dirs,
    description='Select Directory:',
    disabled=False,
    layout=widgets.Layout(width='300px'),
    style={'description_width': '102px'}
)

# Create an Output widget to display messages
output = widgets.Output()
```

```
# Initialize global variables
input_dir = []
output_dir = []

def on_dir_change(change):
    """
    Callback function to handle directory selection changes.
    Sets the input and output directories, performs checks,
    and displays relevant messages.
    """
    global input_dir, output_dir
    if change['type'] == 'change' and change['name'] == 'value':
        selected_dir = change['new']

        with output:
            clear_output()

            if selected_dir == "All":
                # Set input_dir and output_dir to include all
directories
                input_dir = [Path(f'../8CD/{dir_name}') for dir_name
in available_dirs if dir_name != "All"]
                output_dir = [Path(f'../8CD/{dir_name}_augmented') for
dir_name in available_dirs if dir_name != "All"]

                # Check if all input directories exist and are not
empty
                missing_dirs = [dir_path for dir_path in input_dir if
not dir_path.exists()]
                empty_dirs = [dir_path for dir_path in input_dir if
not any(dir_path.iterdir())]

                if missing_dirs:
                    for dir_path in missing_dirs:
                        print(f"{RED}The input directory '{dir_path}'
does not exist.{RESET}")
                    return

                if empty_dirs:
                    for dir_path in empty_dirs:
                        print(f"{RED}The input directory '{dir_path}'
is empty.{RESET}")
                    return

                # Create all output directories if they do not exist
                for out_dir in output_dir:
                    out_dir.mkdir(parents=True, exist_ok=True)

                # Display the valid directories
```

```
                print(f"{GREEN}Valid Input Directories:
{[str(dir_path) for dir_path in input_dir]}{RESET}")
                print(f"{GREEN}Valid Output Directories:
{[str(dir_path) for dir_path in output_dir]}{RESET}")

            else:
                # Set input_dir and output_dir for a single directory
                input_dir = [Path(f'../8CD/{selected_dir}')]
                output_dir =
[Path(f'../8CD/{selected_dir}_augmented')]

                input_path = input_dir[0]
                output_path = output_dir[0]

                # Check if the input directory exists
                if not input_path.exists():
                    print(f"{RED}The input directory '{input_path}'
does not exist.{RESET}")
                    return

                # Check if the input directory contains any files
                if not any(input_path.iterdir()):
                    print(f"{RED}The input directory '{input_path}' is
empty.{RESET}")
                    return

                # Create the output directory if it does not exist
                output_path.mkdir(parents=True, exist_ok=True)

                # Display the valid directories
                print(f"{GREEN}Valid Input Directory: {input_path}
{RESET}")
                print(f"{GREEN}Valid Output Directory: {output_path}
{RESET}")

# Attach the callback to the Dropdown
dir_dropdown.observe(on_dir_change)

# Display the Dropdown and Output widgets
display(dir_dropdown, output)

# Set a default selection to the first directory (not "All")
default_dir = available_dirs[1]  # Index 1 since "All" is at index 0
dir_dropdown.value = default_dir  # This will trigger the callback and
display directories
```

{"model_id":"88881eb5eae040499db1c89ec6acc5d3","version_major":2,"vers
ion_minor":0}

{"model_id":"9cc5e7d1226c4a46a0afc0723f3683a9","version_major":2,"version_minor":0}

Brightness and Contrast Adjustment

This function adjusts the brightness and contrast of an input image. It accepts an image in the form of a NumPy array or a PIL Image and applies specified brightness and contrast levels within predefined bounds. The adjustments are performed using the `ImageEnhance` module from Pillow, ensuring that the output image maintains visual consistency.

```
def apply_brightness_contrast(img, brightness=0, contrast=0):
    # Ensure the input is a valid image (either NumPy array or PIL Image)
    if isinstance(img, np.ndarray):
        img = Image.fromarray(img)  # Convert NumPy array to PIL Image if needed
    elif not isinstance(img, Image.Image):
        raise TypeError("Input image must be a NumPy array or a PIL Image - apply_brightness_contrast")

    # Ensure brightness and contrast are within reasonable bounds
    brightness = max(-100, min(brightness, 100))
    contrast = max(-100, min(contrast, 100))
    # Apply brightness and contrast adjustments
    enhancer_b = ImageEnhance.Brightness(img)
    img = enhancer_b.enhance(1 + brightness / 100.0)
    enhancer_c = ImageEnhance.Contrast(img)
    img = enhancer_c.enhance(1 + contrast / 100.0)

    return np.array(img)  # Return the result as a NumPy array
```

Noise Addition

This function introduces Gaussian noise to an input image to simulate real-world imperfections and enhance data variability. It accepts an image as a NumPy array and applies noise with specified mean and standard deviation values. The function ensures that the input is a valid NumPy array, generates the Gaussian noise, and adds it to the image while maintaining pixel values within the valid 8-bit range.

```
def add_noise(img, mean=0, stddev=25):
    # Ensure the input image is a NumPy array
    if not isinstance(img, np.ndarray):
        raise TypeError("Input image must be a NumPy array - add_noise")

    # Generate Gaussian noise
    noise = np.random.normal(mean, stddev, img.shape).astype(np.float32)
```

```
    # Add the noise to the image
    noisy_img = cv2.add(img.astype(np.float32), noise)
    # Clip values to ensure they stay within valid 8-bit range [0,
255]
    noisy_img = np.clip(noisy_img, 0, 255).astype(np.uint8)

    return noisy_img
```

Blur Application

This function applies Gaussian blur to an input image to smooth out details and reduce noise. It accepts an image as a NumPy array and a kernel size (`ksize`) which must be a positive odd integer. The function verifies that the kernel size meets the required conditions and then utilizes OpenCV's `GaussianBlur` method to perform the blurring operation.

```
def apply_blur(img, ksize=5):
    # Ensure ksize is a positive odd integer
    if not isinstance(ksize, int) or ksize <= 0 or ksize % 2 == 0:
        raise ValueError("Kernel size (ksize) must be a positive odd
integer - apply_blur")

    # Apply Gaussian blur with the given kernel size
    blurred_img = cv2.GaussianBlur(img, (ksize, ksize), 0)

    return blurred_img
```

Skew Transformation

This function applies a skew transformation to an input image, introducing perspective distortion to simulate real-world angular variations. It accepts an image as a NumPy array and parameters for skew factor and skew angle. The function validates the input to ensure it is a valid RGB or RGBA image, defines source and destination points for the affine transformation, and computes the transformation matrix using OpenCV's `getAffineTransform` method.

```
def apply_skew(img, skew_factor=0.1, skew_angle=0.33):
    # Ensure the input is a valid image (NumPy array)
    if not isinstance(img, np.ndarray):
        raise TypeError("Input image must be a NumPy array.")

    if img.ndim != 3 or img.shape[2] not in [3, 4]:  # Checking for
RGB or RGBA images
        raise ValueError("Input image must be a 3-channel (RGB) or 4-
channel (RGBA) image.")

    rows, cols, _ = img.shape

    # Define source and destination points for the affine
transformation
```

```
    pts1 = np.float32([[0, 0], [cols, 0], [0, rows]])
    pts2 = np.float32([
        [cols * skew_factor, rows * skew_angle],  # Skewing the top-
left corner
        [cols * (1 - skew_factor), rows * (skew_angle - 0.08)],  #
Skewing the top-right corner
        [cols * skew_factor, rows * (1 - skew_angle)]  # Skewing the
bottom-left corner
    ])

    # Get the affine transformation matrix
    M = cv2.getAffineTransform(pts1, pts2)

    # Apply the affine transformation (skewing)
    skewed_img = cv2.warpAffine(img, M, (cols, rows))

    return skewed_img
```

Scaling Transformation

This function scales an image by a specified factor to simulate size variations. It accepts an image as a NumPy array and a positive scale factor, calculates the new dimensions, and resizes the image using OpenCV's `resize` function.

```
def apply_scaling(img, scale=1.2):
    # Ensure the input is a valid image (NumPy array)
    if not isinstance(img, np.ndarray):
        raise TypeError("Input image must be a NumPy array.")

    if img.ndim != 3:
        raise ValueError("Input image must be a 3-channel (RGB) or 4-
channel (RGBA) image.")

    if scale <= 0:
        raise ValueError("Scale factor must be a positive value.")

    # Get the original dimensions of the image
    height, width = img.shape[:2]

    # Calculate new dimensions based on the scale factor
    new_width = int(width * scale)
    new_height = int(height * scale)

    # Ensure that the new dimensions are positive
    if new_width <= 0 or new_height <= 0:
        raise ValueError("Scaling factor results in invalid image
dimensions.")

    # Resize the image
```

```
    resized_img = cv2.resize(img, (new_width, new_height))

    return resized_img
```

Rotation Transformation

This function rotates an image by a random angle within a specified range. It calculates the rotation matrix using OpenCV's `getRotationMatrix2D` and applies the transformation to the image using `warpAffine`.

```
def apply_rotation(img, max_angle=15):
    """Rotate the image by a random angle within the given range."""
    rows, cols, ch = img.shape
    angle = random.uniform(-max_angle, max_angle)
    M = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
    rotated_img = cv2.warpAffine(img, M, (cols, rows))
    return rotated_img
```

Flip Transformation

This function flips an image either horizontally or vertically. It accepts the image as a NumPy array and a flip type (`horizontal` or `vertical`). The OpenCV `flip` function performs the operation, and the flipped image is returned, enabling various image augmentation scenarios.

```
def apply_flip(img, flip_type="horizontal"):
    """Flip the image either horizontally or vertically."""
    if flip_type == "horizontal":
        return cv2.flip(img, 1)  # Horizontal flip
    elif flip_type == "vertical":
        return cv2.flip(img, 0)  # Vertical flip
    else:
        raise ValueError("flip_type must be either 'horizontal' or
'vertical'.")
```

Color Jitter

This function randomly adjusts the brightness, contrast, saturation, and hue of an image to simulate different lighting conditions. It operates in the HSV color space, making pixel-wise adjustments before converting the image back to BGR format.

```
def apply_color_jitter(img, brightness=0, contrast=0, saturation=0,
hue=0):
    """Randomly adjust brightness, contrast, saturation, and hue of
the image."""
    hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # Apply brightness
```

```
    hsv_img[:, :, 2] = np.clip(hsv_img[:, :, 2] + brightness, 0, 255)

    # Apply contrast
    hsv_img[:, :, 1] = np.clip(hsv_img[:, :, 1] + contrast, 0, 255)

    # Apply saturation
    hsv_img[:, :, 1] = np.clip(hsv_img[:, :, 1] + saturation, 0, 255)

    # Apply hue
    hsv_img[:, :, 0] = np.clip(hsv_img[:, :, 0] + hue, 0, 179)

    jittered_img = cv2.cvtColor(hsv_img, cv2.COLOR_HSV2BGR)
    return jittered_img
```

Perspective Transformation

This function applies a perspective transformation to create a warped view of an image. The transformation strength determines the degree of distortion, with corner points shifted proportionally. OpenCV's `getPerspectiveTransform` computes the transformation matrix, and `warpPerspective` applies it to the image.

```
def apply_perspective_transform(img, strength=1.0):
    """Apply a perspective transformation to the image with a given
strength."""
    rows, cols, _ = img.shape

    # The strength will control how much the corners of the image are
shifted
    shift_x = cols * (0.1 * strength)  # Horizontal shift based on
strength
    shift_y = rows * (0.1 * strength)  # Vertical shift based on
strength

    # Define the original and transformed points
    pts1 = np.float32([[0, 0], [cols, 0], [0, rows], [cols, rows]])
    pts2 = np.float32([
        [shift_x, shift_y],
        [cols - shift_x, shift_y],
        [shift_x, rows - shift_y],
        [cols - shift_x, rows - shift_y]
    ])

    # Compute the perspective transform matrix
    M = cv2.getPerspectiveTransform(pts1, pts2)

    # Apply the perspective warp
    warped_img = cv2.warpPerspective(img, M, (cols, rows))
    return warped_img
```

Data Augmentation and Processing

This section performs data augmentation on images from the input directories and saves the augmented results in the corresponding output directories. The process includes:

1. **Directory Setup:**
   - The base directory is defined, and input/output directories are validated and created if necessary.
   - All image files from the input directory are collected for processing.
2. **Image Augmentation:**
   - Various transformations are applied to each image, including adjustments for brightness, contrast, noise, blur, skew, scaling, rotation, flipping, color jitter, and perspective transformations.
   - Each transformation produces a uniquely augmented image with a descriptive filename.
3. **Progress Monitoring:**
   - The `tqdm` library provides a progress bar to track the processing of images within each directory.
4. **Output:**
   - The augmented images are saved in their respective output directories with appropriate names reflecting the applied transformation.

This process enhances the dataset by increasing variability, which is beneficial for tasks requiring robust training and testing datasets.

```
##########################
#    Data Augmentation   #
##########################
# Define the base directory for '8CD' as being one level up from the
current working directory
base_dir = Path(os.getcwd()).parent / "8CD"

# Loop through all images in the input directory
for idx, (in_dir, out_dir) in enumerate(zip(input_dir, output_dir)):
    # Construct the full paths for input and output directories
    in_dir = base_dir / in_dir
    out_dir = base_dir / out_dir

    # Ensure the input directory exists
    if not in_dir.exists():
        print(f"Error: Input directory '{in_dir}' does not exist.
Skipping...")
        continue

    # Ensure the output directory exists or create it
    out_dir.mkdir(parents=True, exist_ok=True)

    # Get the list of image files
```

```
    image_files = [f for f in os.listdir(in_dir) if
f.endswith((".jpg", ".png"))]
    total_files = len(image_files)

    if total_files == 0:
        print(f"No images found in directory '{in_dir}'. Skipping...")
        continue

    print(f"Processing Templates in {in_dir}:")

    # Use tqdm to create a progress bar
    for filename in tqdm(image_files, desc="Processing Templates",
unit="file"):
        img_path = in_dir / filename

        # Read the image and check if it's loaded correctly
        img = cv2.imread(str(img_path))  # Ensure path is string
        if img is None:
            print(f"Warning: Failed to load image {filename}.
Skipping...")
            continue

        # Apply various augmentations with different values for each
transformation
        augmented_images = [
            ("brightness_contrast_+30_+30",
apply_brightness_contrast(img, brightness=30, contrast=30)),
            ("brightness_contrast_-30_-30",
apply_brightness_contrast(img, brightness=-30, contrast=-30)),
            ("brightness_contrast_+50_+50",
apply_brightness_contrast(img, brightness=50, contrast=50)),
            ("brightness_contrast_-50_-50",
apply_brightness_contrast(img, brightness=-50, contrast=-50)),
            ("noise_25", add_noise(img)),
            ("noise_50", add_noise(img, mean=0, stddev=50)),
            ("blur_ksize_3", apply_blur(img, ksize=3)),
            ("blur_ksize_5", apply_blur(img, ksize=5)),
            ("blur_ksize_9", apply_blur(img, ksize=9)),
            ("skew", apply_skew(img)),
            ("skew_stronger", apply_skew(img, skew_factor=1.5)),
            ("scaling_1.1", apply_scaling(img, scale=1.1)),
            ("scaling_0.9", apply_scaling(img, scale=0.9)),
            ("scaling_1.2", apply_scaling(img, scale=1.2)),
            ("scaling_0.8", apply_scaling(img, scale=0.8)),
            ("rotation_15", apply_rotation(img, max_angle=15)),
            ("rotation_30", apply_rotation(img, max_angle=30)),
            ("rotation_45", apply_rotation(img, max_angle=45)),
            ("flip_horizontal", apply_flip(img,
flip_type="horizontal")),
            ("flip_vertical", apply_flip(img, flip_type="vertical")),
```

```
            ("color_jitter_10_10_10_5", apply_color_jitter(img,
brightness=10, contrast=10, saturation=10, hue=5)),
            ("color_jitter_30_30_30_15", apply_color_jitter(img,
brightness=30, contrast=30, saturation=30, hue=15)),
            ("perspective_transform",
apply_perspective_transform(img)),
            ("perspective_transform_strong",
apply_perspective_transform(img, strength=1.5)),
        ]

        # Save augmented images with descriptive names
        for aug_name, aug_img in augmented_images:
            if aug_img is not None:  # Ensure the transformed image is
valid
                aug_img_path = out_dir /
f"{Path(filename).stem}_{aug_name}.png"
                cv2.imwrite(str(aug_img_path), aug_img)

# Print the success message in green
print(f"{GREEN}Data augmentation completed successfully.{RESET}")
```

```
Processing Templates in /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced
ML/Constellation-Finder/8CD/../8CD/CoronaBorealis:

Processing Templates: 100%|██████████| 10/10 [00:03<00:00,
2.65file/s]

Processing Templates in /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced ML/Constellation-Finder/8CD/../8CD/Lacerta:

Processing Templates: 100%|██████████| 10/10 [00:06<00:00,
1.56file/s]

Processing Templates in /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced ML/Constellation-Finder/8CD/../8CD/LeoMinor:

Processing Templates: 100%|██████████| 8/8 [00:03<00:00,  2.12file/s]

Processing Templates in /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced ML/Constellation-Finder/8CD/../8CD/Lepus:

Processing Templates: 100%|██████████| 10/10 [00:07<00:00,
1.41file/s]

Processing Templates in /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced ML/Constellation-Finder/8CD/../8CD/Norma:

Processing Templates: 100%|██████████| 10/10 [00:05<00:00,
1.84file/s]
```

```
Processing Templates in /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced ML/Constellation-Finder/8CD/../8CD/Pisces:

Processing Templates: 100%|██████████| 10/10 [00:06<00:00,
1.53file/s]

Processing Templates in /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced ML/Constellation-Finder/8CD/../8CD/Sextans:

Processing Templates: 100%|██████████| 10/10 [00:03<00:00,
3.20file/s]

Processing Templates in /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced ML/Constellation-Finder/8CD/../8CD/Virgo:

Processing Templates: 100%|██████████| 10/10 [00:05<00:00,
1.90file/s]

Data augmentation completed successfully.
```