FOTO-Konvertierung: Detailed Instructions

Overview

This script is designed to process images by upscaling them using Topaz Photo AI, detecting optimal crop coordinates using a YOLOv5 model, and then cropping the images to specified aspect ratios. The script listens for messages from a RabbitMQ queue and processes the images accordingly.

Requirements

Python Packages

Ensure the following Python packages are installed:

- `os`
- `subprocess`
- `shutil`
- `pika`
- `json`
- `requests`
- `torch`
- `PIL` (Pillow)
- `time`

Install the required packages using:

```
pip install pika requests torch pillow
```

External Tools

- **FFmpeg**: Download and install FFmpeg. Ensure `ffmpeg.exe` is accessible in your system's PATH or provide the full path in the script.
- **Topaz Photo AI**: Download and install Topaz Photo AI. Provide the path to `tpai.exe` in the script.
- **YOLOv5**: Download the YOLOv5 model and ensure the repository is available locally. Provide the paths to the model and repository in the script.

Configuration

Paths Configuration

Update the paths in the script to match your local setup:

```
local_temp_folder = "C:\\Users\\khali\\Desktop\\Vocies\\Project\\tempfoto"
output_folder = "C:\\Users\\khali\\Desktop\\Vocies\\Project\\output_foto"
ffmpeg_path = "ffmpeg.exe"
photo_ai_path = "C:\\Program Files\\Topaz Labs LLC\\Topaz Photo AI\\tpai.exe"
yolov5_model_path = "C:\\Users\\khali\\Desktop\\Vocies\\Project\\yolov5\\yolov5s.pt"
yolov5_repo_path = "C:\\Users\\khali\\Desktop\\Vocies\\Project\\yolov5"
```

Timeout Configuration

Configure the timeouts for subprocess calls:

```
subprocess_timeout = 600  # Timeout for subprocess calls in seconds
ffmpeg_timeout = 30       # Timeout for FFmpeg calls in seconds
```

Functions

`clear_temp_folder(folder)`

Clears all files and directories in the specified folder.

Parameters:

- `folder` (str): Path to the folder to be cleared.

Usage:

```
clear_temp_folder("C:\\path\\to\\folder")
```

`upscale_image_with_photo_ai(input_file, output_folder)`

Upscales an image using Topaz Photo AI.

Parameters:

- `input_file` (str): Path to the input image file.

- `output_folder` (str): Path to the folder where the upscaled image will be saved.

Returns:

- Path to the upscaled image or `None` if an error occurs.

Usage:

```
upscaled_image = upscale_image_with_photo_ai("input.jpg", "output_folder")
```

Details:

1. Constructs the command to call `tpai.exe` with necessary parameters.
2. Executes the command using `subprocess.run` .
3. Returns the path to the upscaled image or `None` if there is an error.

`get_optimal_crop_coordinates(input_file)`

Detects optimal crop coordinates using the YOLOv5 model.

Parameters:

- `input_file` (str): Path to the input image file.

Returns:

- Tuple containing the center coordinates `(center_x, center_y)` or `None` if an error occurs.

Usage:

```
center_x, center_y = get_optimal_crop_coordinates("input.jpg")
```

Details:

1. Loads the YOLOv5 model.
2. Opens and verifies the input image.
3. Processes the image with the YOLOv5 model to detect objects.
4. Calculates and returns the center coordinates of the detected object.

`crop_image(input_file, output_file, center_x, center_y, aspect_ratio)`

Crops the image to the specified aspect ratio.

Parameters:

- `input_file` (str): Path to the input image file.
- `output_file` (str): Path to the output cropped image file.
- `center_x` (int): X coordinate of the crop center.
- `center_y` (int): Y coordinate of the crop center.
- `aspect_ratio` (str): Desired aspect ratio (e.g., "16:9", "1:1").

Returns:

- Path to the cropped image or `None` if an error occurs.

Usage:

```
cropped_image = crop_image("input.jpg", "output.jpg", 500, 500, "16:9")
```

Details:

1. Opens the input image.
2. Calculates the new width and height based on the desired aspect ratio.
3. Determines the crop area based on the center coordinates.
4. Constructs the FFmpeg command to crop the image.
5. Executes the command using `subprocess.run`.
6. Returns the path to the cropped image or `None` if there is an error.

`on_message(channel, method_frame, header_frame, body)`

Callback function to process a message from the RabbitMQ queue.

Parameters:

- `channel` - RabbitMQ channel.
- `method_frame` - Delivery method.
- `header_frame` - Message headers.
- `body` (str) - Message body containing asset details.

Usage: This function is used internally by the RabbitMQ consumer and is not called directly.

Details:

1. Parses the message body to extract asset details.

2. Upscales the image using `upscale_image_with_photo_ai`.
3. Gets the optimal crop coordinates using `get_optimal_crop_coordinates`.
4. Crops the image to various formats using `crop_image`.
5. Sends success or error messages back to RabbitMQ.

Main Execution Flow

`main()`

Main function that sets up the RabbitMQ connection and starts listening for messages.

Usage:

```
main()
```

Details:

1. Ensures the local temporary folder exists.
2. Connects to RabbitMQ.
3. Configures the RabbitMQ channel and queue.
4. Processes existing images in the local temporary folder.
5. Starts consuming messages from the `convert-image-format` queue.

Execution Loop

Runs the main function in a loop to handle crashes or connection errors.

Usage: The script includes a loop to handle exceptions and restart the main function after a short delay:

```python
if __name__ == "__main__":
    print("FOTO-Konvertierung: Starting processing script...")
    while True:
        try:
            main()
        except Exception as e:
            print(f"Main function crashed: {e}. Restarting in 5 seconds...")
            time.sleep(5)
        except ConnectionError as e:
            print(f"Failed to establish initial connection: {e}. Retrying in 5 seconds...")
            time.sleep(5)
```

Running the Script

To start the script, execute the following command:

```
python script_name.py
```

Ensure all dependencies are installed and external tools are configured correctly. The script will listen for messages and process images as described.

Example Workflow

1. **Setup**: Ensure all paths and configurations are correct.
2. **Start Script**: Run the script using `python script_name.py` .
3. **Message Reception**: The script listens for messages from RabbitMQ.
4. **Image Processing**: Upon receiving a message, the script:
    - Upscales the image.
    - Detects optimal crop coordinates.
    - Crops the image to specified aspect ratios.
5. **Message Acknowledgment**: Sends success or error messages back to RabbitMQ.
6. **Loop**: Continues to listen for new messages and process images.

This detailed guide provides step-by-step instructions for configuring, running, and understanding the script. Ensure all dependencies and paths are correctly set up for successful execution.