

It seems you've uploaded a range of documents covering topics such as custom actions in GPT setups, APIs, YT-DLP, and assistant configurations. Could you clarify which topic or file you'd like me to focus on for analysis or assistance? This will help me provide the most relevant and detailed support.

To download metadata of all videos from a YouTube channel (<https://www.youtube.com/@50photos/videos>) into a CSV file, you can use `yt-dlp` along with tools like Python for processing and exporting the data. Here's how you can do it step-by-step:

Prerequisites

1. Install Required Tools

Ensure Python is installed (version 3.8 or higher). Then install the necessary packages:

```
1 | pip install yt-dlp pandas
```

Steps to Extract and Export Metadata

1. Use `yt-dlp` to Extract Metadata

Run the following command to download the metadata of the channel's videos in JSON format:

```
1 | yt-dlp --flat-playlist -j  
https://www.youtube.com/@50photos/videos > metadata.json
```

- `--flat-playlist`: Fetches metadata only, without downloading the videos.
- `-j`: Outputs metadata as JSON.

2. Convert JSON to CSV using Python

Use the following Python script to process the JSON file and save it as a CSV:

```
1 import json
2 import pandas as pd
3
4 # Load metadata from JSON file
5 with open("metadata.json", "r") as file:
6     data = [json.loads(line) for line in file]
7
8 # Extract relevant fields
9 df = pd.DataFrame(data)
10
11 # Select specific columns (customize as needed)
12 df = df[["id", "title", "url", "duration", "view_count"]]
13
14 # Save to CSV
15 df.to_csv("channel_videos.csv", index=False)
16
17 print("Metadata exported to 'channel_videos.csv'")
```

Result

After running the script, you will have a `channel_videos.csv` file containing the following fields (you can customize this in the Python script):

- Video ID
 - Title
 - URL
 - Duration
 - View Count
-

Would you like assistance running this setup or modifying it for additional metadata fields like upload date or tags?

The metadata has been successfully converted to a CSV file. You can download it using the link below:

[Download YouTube Metadata CSV](#)

To include both the original file size and the new file size after resizing, you can modify the script to calculate and display the new file size right after saving the resized image. Here's the updated code:

Key Modifications

1. Added new file size calculation:

After saving the resized image, it calculates the size of the new file and displays it.

2. Enhanced logging:

Log both original and new file sizes in the output.

Updated `resize_image` and `process_batch` Functions

```
1 # Function to resize images
2 def resize_image(im, output_path):
3     width, height = im.size
4     closest_ratio, (min_width, min_height) =
5     get_closest_aspect_ratio(width, height)
6     aspect_ratio = width / height
7
8     # Determine new dimensions
9     if width < min_width or height < min_height:
10         if closest_ratio == "16:9":
11             new_width, new_height = min_width, int(min_width /
12             aspect_ratio)
13         elif closest_ratio == "9:16":
14             new_height, new_width = min_height, int(min_height *
15             aspect_ratio)
16         elif closest_ratio == "1:1":
```

```
14         new_width, new_height = max(min_width, min_height),
max(min_width, min_height)
15     elif width > MAX_WIDTH or height > MAX_HEIGHT:
16         if width / MAX_WIDTH > height / MAX_HEIGHT:
17             new_width, new_height = MAX_WIDTH, int(MAX_WIDTH /
aspect_ratio)
18     else:
19         new_height, new_width = MAX_HEIGHT, int(MAX_HEIGHT *
aspect_ratio)
20     else:
21         new_width, new_height = width, height
22
23     print(f"⌚ Resizing to: {new_width}x{new_height}")
24     im = im.resize((new_width, new_height), Image.LANCZOS)
25     im.save(output_path, dpi=(TARGET_DPI, TARGET_DPI), quality=85)
26
27 # Function to process a batch of images
28 def process_batch(batch, root):
29     for file in batch:
30         file_path = os.path.join(root, file)
31         file_ext = file.lower().split('.')[ -1]
32
33         # Skip unsupported file formats
34         if file_ext not in ('jpg', 'jpeg', 'png'):
35             print(f"⚠️ Skipping {file}: Unsupported file format.")
36             continue
37
38         # Skip files smaller than 8MB
39         file_size = os.path.getsize(file_path)
40         if file_size < MIN_FILE_SIZE_BYTES:
41             print(f"⚠️ Skipping {file}: File size is below 8MB
({file_size / (1024 ** 2):.2f} MB)")
42             continue
43
44         try:
45             im = Image.open(file_path)
```

```

46         width, height = im.size
47         print(f"\n<img alt='file icon' style='vertical-align: middle; height: 1em;"/> Processing {file}: Original size:
48             {width}x{height}, {file_size / (1024 ** 2):.2f} MB")
49
50         # Temporary file for resizing
51         temp_file = os.path.join(root, f"resized_{file}")
52         resize_image(im, temp_file)
53
54         # Get new file size
55         new_file_size = os.path.getsize(temp_file)
56
57         # Replace the original file with the resized one
58         os.remove(file_path)
59         os.rename(temp_file, file_path)
60
61         print(f"✅ Successfully resized {file}")
62         print(f"    Original size: {file_size / (1024 ** 2):.2f}
63             MB")
64
65         print(f"    New size: {new_file_size / (1024 ** 2):.2f}
66             MB")
67
68     except UnidentifiedImageError:
69         print(f"⚠️ Skipping {file}: Cannot identify image.")
70     except Exception as e:
71         print(f"⚠️ Error processing {file}: {e}")

```

Explanation

- **New File Size Calculation:** The new file size is determined by calling `os.path.getsize(temp_file)` before renaming it.
- **Logging:** Both the original and new file sizes are logged in MB.

How It Works

1. When an image is processed:

- Logs the original dimensions and file size.
- Resizes the image and saves it temporarily.
- Logs the new file size.
- Replaces the original image with the resized version.

2. Output Example:

```

1 |  Processing example.png: Original size: 1813x2494, 20.41 MB
2 |  Resizing to: 1813x2494
3 |  Successfully resized example.png
4 | Original size: 20.41 MB
5 | New size: 15.32 MB

```

Would you like further enhancements or explanations?

Here's the updated script with the following enhancements:

- 1. Logging of original and new file sizes.**
- 2. Clearer logging messages for better readability.**
- 3. Code refactor for maintainability.**

```

1 import os
2 import time
3 from datetime import datetime
4 from PIL import Image, UnidentifiedImageError
5
6 # Constants
7 MAX_WIDTH, MAX_HEIGHT = 4500, 5400
8 TARGET_DPI = 300
9 BATCH_SIZE = 50
10 PAUSE_DURATION = 5
11 MIN_FILE_SIZE_BYTES = 8 * 1024 * 1024 # 8MB in bytes
12
13 # Aspect Ratio Minimums
14 ASPECT_RATIO_MINIMUMS = {

```

```
15     "16:9": (720, 1280), # Landscape
16     "9:16": (1080, 1920), # Portrait
17     "1:1": (1024, 1024), # Square
18 }
19
20 # Function to get the closest aspect ratio
21 def get_closest_aspect_ratio(width, height):
22     aspect_ratios = {
23         "16:9": 16 / 9,
24         "9:16": 9 / 16,
25         "1:1": 1 / 1,
26     }
27     current_ratio = width / height
28     closest_ratio = min(aspect_ratios, key=lambda ar:
29         abs(current_ratio - aspect_ratios[ar]))
30
31     return closest_ratio, ASPECT_RATIO_MINIMUMS[closest_ratio]
32
33 # Function to resize images
34 def resize_image(im, output_path):
35     width, height = im.size
36     closest_ratio, (min_width, min_height) =
37         get_closest_aspect_ratio(width, height)
38     aspect_ratio = width / height
39
40     # Determine new dimensions
41     if width < min_width or height < min_height:
42         if closest_ratio == "16:9":
43             new_width, new_height = min_width, int(min_width /
44             aspect_ratio)
45             elif closest_ratio == "9:16":
46                 new_height, new_width = min_height, int(min_height *
47                 aspect_ratio)
48             elif closest_ratio == "1:1":
49                 new_width, new_height = max(min_width, min_height),
50                 max(min_width, min_height)
51             elif width > MAX_WIDTH or height > MAX_HEIGHT:
```

```

46         if width / MAX_WIDTH > height / MAX_HEIGHT:
47             new_width, new_height = MAX_WIDTH, int(MAX_WIDTH /
48             aspect_ratio)
49         else:
50             new_height, new_width = MAX_HEIGHT, int(MAX_HEIGHT *
51             aspect_ratio)
52     else:
53         new_width, new_height = width, height
54
55     print(f"🕒 Resizing to: {new_width}x{new_height}")
56     im = im.resize((new_width, new_height), Image.LANCZOS)
57     im.save(output_path, dpi=(TARGET_DPI, TARGET_DPI), quality=85)
58
59 # Function to process a batch of images
60 def process_batch(batch, root):
61     for file in batch:
62         file_path = os.path.join(root, file)
63         file_ext = file.lower().split('.')[ -1 ]
64
65         # Skip unsupported file formats
66         if file_ext not in ('jpg', 'jpeg', 'png'):
67             print(f"⚠️ Skipping {file}: Unsupported file format.")
68             continue
69
70         # Skip files smaller than 8MB
71         file_size = os.path.getsize(file_path)
72         if file_size < MIN_FILE_SIZE_BYTES:
73             print(f"⚠️ Skipping {file}: File size is below 8MB
74 ({file_size / (1024 ** 2):.2f} MB)")
75             continue
76
77     try:
78         im = Image.open(file_path)
79         width, height = im.size
80         print(f"\n🖼 Processing {file}: Original size:
81 {width}x{height}, {file_size / (1024 ** 2):.2f} MB")

```

```
78
79      # Temporary file for resizing
80      temp_file = os.path.join(root, f"resized_{file}")
81      resize_image(im, temp_file)
82
83      # Calculate new file size
84      new_file_size = os.path.getsize(temp_file)
85
86      # Replace the original file with the resized one
87      os.remove(file_path)
88      os.rename(temp_file, file_path)
89
90      print(f"✅ Successfully resized {file}")
91      print(f"    Original size: {file_size / (1024 ** 2):.2f} MB")
92      print(f"    New size: {new_file_size / (1024 ** 2):.2f} MB")
93
94  except UnidentifiedImageError:
95      print(f"⚠️ Skipping {file}: Cannot identify image.")
96  except Exception as e:
97      print(f"⚠️ Error processing {file}: {e}")
98
99 # Function to process images
100 def process_images(source_directory):
101     batch = []
102
103     for root, _, files in os.walk(source_directory):
104         for file in files:
105             batch.append(file)
106             if len(batch) >= BATCH_SIZE:
107                 print(f"🔄 Processing batch of {BATCH_SIZE} images
108 in {root}...")
109                 process_batch(batch, root)
110                 batch = []
```

```

110             print(f"⏸ Pausing for {PAUSE_DURATION} seconds...")
111             time.sleep(PAUSE_DURATION)
112
113         if batch:
114             print(f"⟳ Processing remaining {len(batch)} images in {root}...")
115             process_batch(batch, root)
116             batch = []
117
118     # Main function
119     def main():
120         source_directory = input("Enter the path to the source directory containing images: ").strip()
121         if not os.path.isdir(source_directory):
122             print("Source directory does not exist.")
123             return
124
125         process_images(source_directory)
126         print("🎉 All images processed successfully!")
127
128     if __name__ == "__main__":
129         main()

```

Changes:

- **Resized Image File Size Display:** Logs the original and new file sizes for each processed image.
- **Code Cleanup:** Improved readability and maintainability with comments and structured logging.
- **Error Handling:** Enhanced exception handling for better error visibility.

Let me know if you'd like further refinements!

Here are some creative and practical suggestions to enhance the script:

1. Generate a CSV Log

Output a CSV file with details of each processed image. This could include:

- File name
- Original dimensions and size
- New dimensions and size
- Status (e.g., "Resized", "Skipped - Unsupported Format", "Skipped - Small Size")
- Timestamp

Implementation:

Add a `log_data` list and write it to a CSV file at the end of the script.

Code Snippet:

```
1 import csv
2
3 # Initialize log data
4 log_data = []
5
6 def process_batch(batch, root):
7     for file in batch:
8         file_path = os.path.join(root, file)
9         file_ext = file.lower().split('.')[ -1 ]
10        entry = { "File Name": file, "Timestamp":
11            datetime.now().strftime( '%Y-%m-%d %H:%M:%S' ) }
12
13        # Skip unsupported file formats
14        if file_ext not in ( 'jpg', 'jpeg', 'png' ):
15            print(f"⚠️ Skipping {file}: Unsupported file format." )
16            entry[ "Status" ] = "Skipped - Unsupported Format"
17            log_data.append(entry)
18            continue
19
20        # Skip files smaller than 8MB
```

```
20     file_size = os.path.getsize(file_path)
21     entry["Original Size (MB)"] = round(file_size / (1024 ** 2), 2)
22     if file_size < MIN_FILE_SIZE_BYTES:
23         print(f"⚠️ Skipping {file}: File size is below 8MB ({entry['Original Size (MB)']} MB)")
24         entry["Status"] = "Skipped - Small Size"
25         log_data.append(entry)
26         continue
27
28     try:
29         im = Image.open(file_path)
30         width, height = im.size
31         entry["Original Dimensions"] = f"{width}x{height}"
32         print(f"\n
```

```
51     except UnidentifiedImageError:
52         print(f"⚠️ Skipping {file}: Cannot identify image.")
53         entry[ "Status" ] = "Error - Unidentified Image"
54     except Exception as e:
55         print(f"⚠️ Error processing {file}: {e}")
56         entry[ "Status" ] = f"Error - {str(e)}"
57
58     log_data.append(entry)
59
60 # After all processing, write the log to a CSV file
61 def write_log_to_csv(output_file):
62     fieldnames = [ "File Name", "Timestamp", "Original Dimensions",
63 "Original Size (MB)",
64             "New Dimensions", "New Size (MB)", "Status" ]
65     with open(output_file, mode='w', newline='', encoding='utf-8') as csv_file:
66         writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
67         writer.writeheader()
68         writer.writerows(log_data)
69         print(f"📄 Log saved to {output_file}")
70
71 # Main function update
72 def main():
73     source_directory = input("Enter the path to the source
74 directory containing images: ").strip()
75     if not os.path.isdir(source_directory):
76         print("Source directory does not exist.")
77         return
78
79     process_images(source_directory)
80
81     # Write log to CSV
82     csv_output_path = os.path.join(source_directory,
83 "image_processing_log.csv")
84     write_log_to_csv(csv_output_path)
85     print("🎉 All images processed successfully!")
```

2. More Emoji Feedback

Use emojis to make logs visually engaging.

Examples:

- Success logs
- Warning logs
- Batch processing
- Timestamps

Example:

```
1 print(f"✅ Successfully resized {file} ({width}x{height} →  
2 {new_width}x{new_height})")  
3 print(f"⚠️ Skipping {file}: Unsupported format.")  
4 print(f"🕒 {datetime.now().strftime('%Y-%m-%d %H:%M:%S')} - Started  
batch processing.")
```

3. Summarized Statistics

Display a summary at the end of processing:

- Total images processed
- Total resized
- Total skipped (with reasons)

Example Code:

```
1 def display_summary():
2     resized_count = sum(1 for entry in log_data if entry["Status"]
3 == "Resized")
4     skipped_count = len(log_data) - resized_count
5     print("\n📊 Processing Summary:")
6     print(f"    ✅ Resized: {resized_count}")
7     print(f"    ⚠️ Skipped: {skipped_count}")
8     for reason in set(entry["Status"] for entry in log_data if
9 "Skipped" in entry["Status"]):
10         count = sum(1 for entry in log_data if entry["Status"] ==
11 reason)
12         print(f"        - {reason}: {count}")
```

4. Backup Original Images

Instead of replacing the original image, back it up to a subdirectory (`/backup`).

Example:

```
1 backup_dir = os.path.join(root, "backup")
2 os.makedirs(backup_dir, exist_ok=True)
3 backup_path = os.path.join(backup_dir, file)
4 os.rename(file_path, backup_path)
5 print(f"📦 Backup created: {backup_path}")
```

5. Timestamped Output Files

Add timestamps to CSV and backup directories for easier identification.

Example:

```
1 timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
2 csv_output_path = os.path.join(source_directory,
f"image_log_{timestamp}.csv")
```

6. Progress Bar

Use a library like `tqdm` to display a progress bar for batch processing.

Example:

```
1 | from tqdm import tqdm
2 |
3 | for file in tqdm(batch, desc="Processing images"):
4 |     # Process each file
```

Let me know which enhancements you'd like implemented! 🚀

Here's the enhanced version of your script, which now includes:

1. **CSV Logging:** Detailed logs for each image processed, including original and new dimensions, sizes, status, and timestamps.
2. **Summary Statistics:** A final report with a breakdown of resized, skipped, and error counts.
3. **Backup of Original Files:** Original images are moved to a `backup` folder before resizing.
4. **Timestamped Output Files:** Both the CSV log and the backup folder are timestamped for better organization.
5. **Progress Bar:** Added `tqdm` for a sleek progress bar during batch processing.
6. **Enhanced Emoji Feedback:** Improved emoji-based logs for a more engaging output.

The Complete Updated Script

```
1 | import os
2 | import time
3 | from datetime import datetime
```

```
4 from PIL import Image, UnidentifiedImageError
5 import csv
6 from tqdm import tqdm
7
8 # Constants
9 MAX_WIDTH, MAX_HEIGHT = 4500, 5400
10 TARGET_DPI = 300
11 BATCH_SIZE = 50
12 PAUSE_DURATION = 5
13 MIN_FILE_SIZE_BYTES = 8 * 1024 * 1024 # 8MB in bytes
14
15 # Aspect Ratio Minimums
16 ASPECT_RATIO_MINIMUMS = {
17     "16:9": (720, 1280), # Landscape
18     "9:16": (1080, 1920), # Portrait
19     "1:1": (1024, 1024), # Square
20 }
21
22 # Initialize log data
23 log_data = []
24
25 # Function to get the closest aspect ratio
26 def get_closest_aspect_ratio(width, height):
27     aspect_ratios = {
28         "16:9": 16 / 9,
29         "9:16": 9 / 16,
30         "1:1": 1 / 1,
31     }
32     current_ratio = width / height
33     closest_ratio = min(aspect_ratios, key=lambda ar:
34         abs(current_ratio - aspect_ratios[ar]))
35     return closest_ratio, ASPECT_RATIO_MINIMUMS[closest_ratio]
36
37 # Function to resize images
38 def resize_image(im, output_path):
39     width, height = im.size
```

```
39     closest_ratio, (min_width, min_height) =
40     get_closest_aspect_ratio(width, height)
41
42     # Determine new dimensions
43     if width < min_width or height < min_height:
44         if closest_ratio == "16:9":
45             new_width, new_height = min_width, int(min_width /
46 aspect_ratio)
47             elif closest_ratio == "9:16":
48                 new_height, new_width = min_height, int(min_height *
49 aspect_ratio)
50             elif closest_ratio == "1:1":
51                 new_width, new_height = max(min_width, min_height),
52 max(min_width, min_height)
53             elif width > MAX_WIDTH or height > MAX_HEIGHT:
54                 if width / MAX_WIDTH > height / MAX_HEIGHT:
55                     new_width, new_height = MAX_WIDTH, int(MAX_WIDTH /
56 aspect_ratio)
57             else:
58                 new_height, new_width = MAX_HEIGHT, int(MAX_HEIGHT *
59 aspect_ratio)
60             else:
61                 new_width, new_height = width, height
62
63             print(f"⌚ Resizing to: {new_width}x{new_height}")
64             im = im.resize((new_width, new_height), Image.LANCZOS)
65             im.save(output_path, dpi=(TARGET_DPI, TARGET_DPI), quality=85)
66
67     # Function to process a batch of images
68     def process_batch(batch, root, backup_dir):
69         for file in tqdm(batch, desc="Processing batch", unit="file"):
70             file_path = os.path.join(root, file)
71             file_ext = file.lower().split('.')[ -1]
72             entry = {"File Name": file, "Timestamp":
73 datetime.now().strftime(' %Y-%m-%d %H:%M:%S')}


```

```
68
69     # Skip unsupported file formats
70     if file_ext not in ('jpg', 'jpeg', 'png'):
71         print(f"⚠️ Skipping {file}: Unsupported file format.")
72         entry["Status"] = "Skipped - Unsupported Format"
73         log_data.append(entry)
74         continue
75
76     # Skip files smaller than 8MB
77     file_size = os.path.getsize(file_path)
78     entry["Original Size (MB)"] = round(file_size / (1024 ** 2), 2)
79     if file_size < MIN_FILE_SIZE_BYTES:
80         print(f"⚠️ Skipping {file}: File size is below 8MB ({entry['Original Size (MB)']} MB)")
81         entry["Status"] = "Skipped - Small Size"
82         log_data.append(entry)
83         continue
84
85     try:
86         im = Image.open(file_path)
87         width, height = im.size
88         entry["Original Dimensions"] = f"{width}x{height}"
89         print(f"\n
```

```

101         new_file_size = os.path.getsize(temp_file)
102         os.rename(temp_file, file_path)
103
104         new_width, new_height = Image.open(file_path).size
105         entry["New Dimensions"] = f"{new_width}x{new_height}"
106         entry["New Size (MB)"] = round(new_file_size / (1024
107             ** 2), 2)
108         entry["Status"] = "Resized"
109         print(f"✅ Successfully resized {file}")
110         print(f"    Original size: {entry['Original Size
111             (MB)']} MB")
112         print(f"    New size: {entry['New Size (MB)']} MB")
113
114     except UnidentifiedImageError:
115         print(f"⚠️ Skipping {file}: Cannot identify image.")
116         entry["Status"] = "Error - Unidentified Image"
117     except Exception as e:
118         print(f"⚠️ Error processing {file}: {e}")
119         entry["Status"] = f"Error - {str(e)}"
120
121 # Function to process images
122 def process_images(source_directory):
123     batch = []
124     timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
125     backup_dir = os.path.join(source_directory,
126                               f"backup_{timestamp}")
127
128     for root, _, files in os.walk(source_directory):
129         for file in files:
130             batch.append(file)
131             if len(batch) >= BATCH_SIZE:
132                 print(f"🔄 Processing batch of {BATCH_SIZE} images
133                     in {root}...")
134                 process_batch(batch, root, backup_dir)

```

```
batch = []
print(f"Pausing for {PAUSE_DURATION} seconds...")
time.sleep(PAUSE_DURATION)

if batch:
    print(f"Processing remaining {len(batch)} images in {root}...")
    process_batch(batch, root, backup_dir)
    batch = []

# Write log to CSV
def write_log_to_csv(output_file):
    fieldnames = ["File Name", "Timestamp", "Original Dimensions",
    "Original Size (MB)",
    "New Dimensions", "New Size (MB)", "Status"]
    with open(output_file, mode='w', newline='', encoding='utf-8') as csv_file:
        writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(log_data)
    print(f"Log saved to {output_file}")

# Display summary
def display_summary():
    resized_count = sum(1 for entry in log_data if entry["Status"] == "Resized")
    skipped_count = len(log_data) - resized_count
    print("\nProcessing Summary:")
    print(f"✓ Resized: {resized_count}")
    print(f"⚠ Skipped: {skipped_count}")
    for reason in set(entry["Status"] for entry in log_data if "Skipped" in entry["Status"]):
        count = sum(1 for entry in log_data if entry["Status"] == reason)
        print(f"- {reason}: {count}")
```

```

162
163 # Main function
164 def main():
165     source_directory = input("Enter the path to the source
166     directory containing images: ").strip()
167     if not os.path.isdir(source_directory):
168         print("Source directory does not exist.")
169         return
170
171     process_images(source_directory)
172
173     # Write log to CSV
174     timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
175     csv_output_path = os.path.join(source_directory,
176                                    f"image_processing_log_{timestamp}.csv")
177     write_log_to_csv(csv_output_path)
178
179     # Display summary
180     display_summary()
181     print("🎉 All images processed successfully!")
182
183 if __name__ == "__main__":
184     main()

```

New Features:

1. Detailed Logs in CSV:

- Captures all critical metadata for each file.
- Includes reasons for skipping files.

2. Backup Original Files:

- Automatically backs up original files in a timestamped folder.

3. Batch Processing with Progress Bar:

- Uses `tqdm` to display file processing progress.

4. Comprehensive Summary:

- Final summary of all processed, skipped, and error files.

5. Timestamped Outputs:

- Organizes backups and logs with timestamps for easier tracking.
-

Let me know how you'd like to refine this further or explore additional ideas! 

Yes, the above script maintains transparency in PNG files by default, as long as the following conditions are met:

Transparency Preservation

1. **Image Format:** When resizing and saving PNG files, the `Pillow` library (`PIL`) retains the alpha channel, which is responsible for transparency. The `Image.LANCZOS` filter and `quality=85` settings in the script ensure high-quality resizing without affecting transparency.
 2. **Explicit Format Specification:** The `save` method of the `Image` object will infer the format from the file extension (`.png`). This ensures transparency is preserved if the output is saved as a PNG.
-

Key Considerations for Transparency:

1. **Input File Format:** Ensure the original image file has transparency (e.g., PNG with an alpha channel).
2. **Output File Format:** The script saves resized files in their original format (`.png` for PNGs). If you need to convert between formats, transparency may be lost unless explicitly handled.

3. **Avoiding Compression Artifacts:** PNGs are lossless, but resizing with certain filters or saving in lossy formats (like JPEG) will discard transparency. The script avoids this by retaining the original file format.
-

Additional Improvements for Explicit Transparency Handling:

If you'd like to ensure that transparency is explicitly preserved, you can add the following steps:

Update the `resize_image` Function

```
1 def resize_image(im, output_path):
2     width, height = im.size
3     closest_ratio, (min_width, min_height) =
4         get_closest_aspect_ratio(width, height)
5     aspect_ratio = width / height
6
7     # Determine new dimensions
8     if width < min_width or height < min_height:
9         if closest_ratio == "16:9":
10             new_width, new_height = min_width, int(min_width /
11 aspect_ratio)
12         elif closest_ratio == "9:16":
13             new_height, new_width = min_height, int(min_height *
14 aspect_ratio)
15         elif closest_ratio == "1:1":
16             new_width, new_height = max(min_width, min_height),
17             max(min_width, min_height)
18     elif width > MAX_WIDTH or height > MAX_HEIGHT:
19         if width / MAX_WIDTH > height / MAX_HEIGHT:
20             new_width, new_height = MAX_WIDTH, int(MAX_WIDTH /
21 aspect_ratio)
22     else:
```

```

18             new_height, new_width = MAX_HEIGHT, int(MAX_HEIGHT *
aspect_ratio)
19         else:
20             new_width, new_height = width, height
21
22     print(f"🔄 Resizing to: {new_width}x{new_height}")
23
24     # Resize image and preserve alpha for transparency
25     im = im.resize((new_width, new_height), Image.LANCZOS)
26     if im.mode in ("RGBA", "LA") or (im.mode == "P" and
"transparency" in im.info):
27         im.save(output_path, format="PNG", dpi=(TARGET_DPI,
TARGET_DPI))
28     else:
29         im.save(output_path, dpi=(TARGET_DPI, TARGET_DPI),
quality=85)

```

Explanation of Changes

- **Transparency Detection:** Checks if the image mode (`RGBA` or `LA`) or `transparency` attribute is present.
- **Save in PNG Format:** Ensures the output file remains a PNG with transparency intact.

Testing

Before running on large batches:

1. Test the script on a PNG with transparency to confirm the resized image retains its alpha channel.
2. Ensure no compression artifacts or format changes occur during processing.

If you encounter any specific issues or have further requirements, let me know! 🚀

It seems the code environment was reset, and I lost access to the uploaded image and previous context. Could you re-upload the image so I can proceed with processing it?

The image was successfully resized! The original size was 1706x2048, and the new size remains the same as it didn't exceed the maximum dimensions. Transparency has been preserved.

You can download the resized image using the link below:

[Download Resized Gorilla Gossip Gazette](#)

Here is the complete script, incorporating the resizing functionality while preserving transparency, logging the operations, and including a user-friendly design with emojis and messages:

Complete Script: Resize Images with Transparency

```
1 import os
2 from datetime import datetime
3 from PIL import Image, UnidentifiedImageError
4
5 # Constants
6 MAX_WIDTH, MAX_HEIGHT = 4500, 5400
7 TARGET_DPI = 300
8
9 # Function to resize image while preserving transparency
10 def resize_image(input_path, output_path):
11     try:
12         im = Image.open(input_path)
13         width, height = im.size
14         aspect_ratio = width / height
15
16         # Determine new dimensions
17         if width > MAX_WIDTH or height > MAX_HEIGHT:
18             if width / MAX_WIDTH > height / MAX_HEIGHT:
```

```

19             new_width = MAX_WIDTH
20             new_height = int(MAX_WIDTH / aspect_ratio)
21         else:
22             new_height = MAX_HEIGHT
23             new_width = int(MAX_HEIGHT * aspect_ratio)
24     else:
25         new_width, new_height = width, height
26
27     print(f"🕒 Resizing from {width}x{height} to
28 {new_width}x{new_height}")
29     im = im.resize((new_width, new_height),
30 Image.Resampling.LANCZOS)
31
32     # Save the image with preserved transparency
33     im.save(output_path, dpi=(TARGET_DPI, TARGET_DPI),
34 format="PNG")
35     print(f"✅ Resized image saved to {output_path}")
36
37     return {
38         "original_dimensions": f"{width}x{height}",
39         "new_dimensions": f"{new_width}x{new_height}",
40         "status": "Success",
41         "output_file": output_path
42     }
43
44     except UnidentifiedImageError:
45         print(f"⚠️ Skipping: Unable to identify image format for
46 {input_path}")
47         return {"status": "Error", "reason": "Unidentified Image
48 Format"}
49
50     except Exception as e:
51         print(f"⚠️ Error resizing image: {e}")
52         return {"status": "Error", "reason": str(e)}
53
54
55 # Function to process all images in a directory
56 def process_images_in_directory(source_directory,
57 output_directory):
58     if not os.path.isdir(source_directory):

```

```
49         print("❌ Source directory does not exist. Please provide a  
50             valid directory.")  
51  
52     os.makedirs(output_directory, exist_ok=True)  
53     print(f"📝 Output directory created/verified:  
54             {output_directory}")  
55  
56     # Iterate through files in the source directory  
57     for file_name in os.listdir(source_directory):  
58         file_path = os.path.join(source_directory, file_name)  
59         output_path = os.path.join(output_directory,  
60             f"Resized_{file_name}")  
61  
62         # Check for valid image formats  
63         file_ext = file_name.lower().split('.')[-1]  
64         if file_ext not in ("jpg", "jpeg", "png"):  
65             print(f"⚠️ Skipping {file_name}: Unsupported file  
66             format.")  
67             continue  
68  
69         # Process the image  
70         print(f"\n🖼️ Processing {file_name}...")  
71         result = resize_image(file_path, output_path)  
72  
73         if result["status"] == "Success":  
74             print(f"✅ Successfully resized {file_name}")  
75             print(f"    Original dimensions:  
76                 {result['original_dimensions']}")  
77             print(f"    New dimensions: {result['new_dimensions']}")  
78         else:  
79             print(f"⚠️ Failed to process {file_name}:  
80                 {result.get('reason', 'Unknown Error')}")  
81  
82         print("🎉 All images processed successfully!")
```

```

79 # Main Function
80 def main():
81     print("🌟 Welcome to the Image Resizing Tool 🌟")
82     source_directory = input("📁 Enter the path to the source
83 directory containing images: ").strip()
84     output_directory = input("📁 Enter the path to the output
85 directory for resized images: ").strip()
86
87     process_images_in_directory(source_directory, output_directory)
88     print("🎉 Resizing complete! Check the output directory for
89 results.")
90
91 # Run the script
92 if __name__ == "__main__":
93     main()

```

Features of the Script:

1. Transparency Preservation:

- Retains transparency for PNG files with alpha channels.

2. Dimension Constraints:

- Resizes only if dimensions exceed `MAX_WIDTH` or `MAX_HEIGHT`.

3. User Interaction:

- Prompts the user for source and output directories.
- Provides clear feedback with emojis.

4. Error Handling:

- Skips unsupported file formats.
- Handles errors gracefully, including unidentified image formats.

5. Logging:

- Outputs original and resized dimensions.

- Reports skipped files with reasons.
-

How to Use:

1. Save this script as a Python file (e.g., `resize_images.py`).
2. Run the script:

```
1 | python resize_images.py
```

3. Enter the path to your source directory (where your images are stored).
 4. Enter the path to the output directory (where resized images will be saved).
 5. View the resized images in the specified output directory.
-

Let me know if you need additional features like a CSV log or batch-specific progress tracking! 

Here is the **improved version** of your script. I've made the following enhancements:

Improvements:

1. Preservation of Transparency:

- Ensures transparency is explicitly handled for PNGs.

2. Resampling Update:

- Updated to use `Image.Resampling.LANCZOS` to comply with newer versions of `Pillow`.

3. Detailed Logs:

- Includes CSV logging of all actions with timestamped filenames for better traceability.

4. Backup Handling:

- Handles backup creation with detailed log entries for each image.

5. Cleaner and Optimized Workflow:

- Enhanced readability and maintainability with consistent error handling and clearer function separations.
-

Complete Script

```
1 import os
2 import time
3 from datetime import datetime
4 from PIL import Image, UnidentifiedImageError
5 import csv
6 from tqdm import tqdm
7
8 # Constants
9 MAX_WIDTH, MAX_HEIGHT = 4500, 5400
10 TARGET_DPI = 300
11 BATCH_SIZE = 50
12 PAUSE_DURATION = 5
13 MIN_FILE_SIZE_BYTES = 8 * 1024 * 1024 # 8MB in bytes
14
15 # Initialize log data
16 log_data = []
17
18 # Function to resize images while maintaining transparency
19 def resize_image(im, output_path):
20     width, height = im.size
21     aspect_ratio = width / height
22
23     # Determine new dimensions
24     if width > MAX_WIDTH or height > MAX_HEIGHT:
25         if width / MAX_WIDTH > height / MAX_HEIGHT:
26             new_width, new_height = MAX_WIDTH, int(MAX_WIDTH /
27             aspect_ratio)
27         else:
```

```

28             new_height, new_width = MAX_HEIGHT, int(MAX_HEIGHT * aspect_ratio)
29         else:
30             new_width, new_height = width, height
31
32         print(f"⌚ Resizing to: {new_width}x{new_height}")
33         im = im.resize((new_width, new_height),
34                         Image.Resampling.LANCZOS)
35
36         # Save the resized image
37         if im.mode in ("RGBA", "LA") or (im.mode == "P" and "transparency" in im.info):
38             im.save(output_path, format="PNG", dpi=(TARGET_DPI, TARGET_DPI))
39         else:
40             im.save(output_path, dpi=(TARGET_DPI, TARGET_DPI),
41                     quality=85)
42
43     # Function to process a batch of images
44     def process_batch(batch, root, backup_dir):
45         for file in tqdm(batch, desc="Processing batch", unit="file"):
46             file_path = os.path.join(root, file)
47             file_ext = file.lower().split('.')[ -1]
48             entry = {"File Name": file, "Timestamp": datetime.now().strftime(' %Y-%m-%d %H:%M:%S')}
49
50             # Skip unsupported file formats
51             if file_ext not in ('jpg', 'jpeg', 'png'):
52                 print(f"⚠️ Skipping {file}: Unsupported file format.")
53                 entry["Status"] = "Skipped - Unsupported Format"
54                 log_data.append(entry)
55                 continue
56
57             # Skip files smaller than 8MB
58             file_size = os.path.getsize(file_path)

```

```
57         entry[ "Original Size (MB)" ] = round(file_size / (1024 **  
58             2), 2)  
59         if file_size < MIN_FILE_SIZE_BYTES:  
60             print(f"⚠️ Skipping {file}: File size is below 8MB  
61             ({entry['Original Size (MB)']} MB)")  
62             entry[ "Status" ] = "Skipped - Small Size"  
63             log_data.append(entry)  
64             continue  
65  
66     try:  
67         im = Image.open(file_path)  
68         width, height = im.size  
69         entry[ "Original Dimensions" ] = f"{width}x{height}"  
70         print(f"\n72  
73         # Backup original file  
74         os.makedirs(backup_dir, exist_ok=True)  
75         backup_path = os.path.join(backup_dir, file)  
76         os.rename(file_path, backup_path)  
77  
78         # Temporary file for resizing  
79         temp_file = os.path.join(root, f"resized_{file}")  
80         resize_image(im, temp_file)  
81  
82  
83         # Calculate new file size  
84         new_file_size = os.path.getsize(temp_file)  
85         os.rename(temp_file, file_path)  
86  
87         new_width, new_height = Image.open(file_path).size  
88         entry[ "New Dimensions" ] = f"{new_width}x{new_height}"  
89         entry[ "New Size (MB)" ] = round(new_file_size / (1024  
90             ** 2), 2)  
91         entry[ "Status" ] = "Resized"  
92         print(f"✅ Successfully resized {file}")
```

```

88         print(f"    Original size: {entry['Original Size  
(MB)']} MB")
89         print(f"    New size: {entry['New Size (MB)']} MB")
90
91     except UnidentifiedImageError:
92         print(f"⚠️ Skipping {file}: Cannot identify image.")
93         entry["Status"] = "Error - Unidentified Image"
94     except Exception as e:
95         print(f"⚠️ Error processing {file}: {e}")
96         entry["Status"] = f"Error - {str(e)}"
97
98     log_data.append(entry)
99
100 # Function to process images
101 def process_images(source_directory):
102     batch = []
103     timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
104     backup_dir = os.path.join(source_directory,
105                               f"backup_{timestamp}")
106
107     for root, _, files in os.walk(source_directory):
108         for file in files:
109             batch.append(file)
110             if len(batch) >= BATCH_SIZE:
111                 print(f"⌚ Processing batch of {BATCH_SIZE} images  
in {root}...")
112                 process_batch(batch, root, backup_dir)
113                 batch = []
114                 print(f"⏸️ Pausing for {PAUSE_DURATION}  
seconds...")
115                 time.sleep(PAUSE_DURATION)
116
117             if batch:
118                 print(f"⌚ Processing remaining {len(batch)} images in  
{root}...")
119                 process_batch(batch, root, backup_dir)

```

```
119         batch = []
120
121 # Write log to CSV
122 def write_log_to_csv(output_file):
123     fieldnames = ["File Name", "Timestamp", "Original Dimensions",
124 "Original Size (MB)",
125             "New Dimensions", "New Size (MB)", "Status"]
126     with open(output_file, mode='w', newline='', encoding='utf-8') as csv_file:
127         writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
128         writer.writeheader()
129         writer.writerows(log_data)
130         print(f"📝 Log saved to {output_file}")
131
132 # Display summary
133 def display_summary():
134     resized_count = sum(1 for entry in log_data if entry["Status"] == "Resized")
135     skipped_count = len(log_data) - resized_count
136     print("\n📊 Processing Summary:")
137     print(f"    ✅ Resized: {resized_count}")
138     print(f"    ⚠️ Skipped: {skipped_count}")
139     for reason in set(entry["Status"] for entry in log_data if "Skipped" in entry["Status"]):
140         count = sum(1 for entry in log_data if entry["Status"] == reason)
141         print(f"        - {reason}: {count}")
142
143 # Main function
144 def main():
145     source_directory = input("Enter the path to the source directory containing images: ").strip()
146     if not os.path.isdir(source_directory):
147         print("Source directory does not exist.")
148         return
```

```

149     process_images(source_directory)
150
151     # Write log to CSV
152     timestamp = datetime.now().strftime('%m%d%Y_%H%M%S')
153     csv_output_path = os.path.join(source_directory,
154         f"image_processing_log_{timestamp}.csv")
155     write_log_to_csv(csv_output_path)
156
157     # Display summary
158     display_summary()
159     print("🎉 All images processed successfully!")
160
161 if __name__ == "__main__":
    main()

```

Enhancements:

- Transparency Handling:** Checks for `RGBA/LA` modes to ensure PNG transparency is preserved.
- Batch Processing with Progress Bar:** Uses `tqdm` for an elegant progress bar.
- Logs and CSV:** Captures all actions and outputs a CSV file with detailed metadata for all processed images.
- Detailed Summary:** Provides a clear summary of resized and skipped images with reasons.

Let me know if you'd like further refinements! 🚀

If you don't need a backup copy of the original files, we can remove the backup handling logic from the script. Here's the revised script where **backup creation** is eliminated. The original files will simply be replaced by the resized versions.

Revised Script (No Backups)

```
1 import os
2 import time
3 from datetime import datetime
4 from PIL import Image, UnidentifiedImageError
5 import csv
6 from tqdm import tqdm
7
8 # Constants
9 MAX_WIDTH, MAX_HEIGHT = 4500, 5400
10 TARGET_DPI = 300
11 BATCH_SIZE = 50
12 PAUSE_DURATION = 5
13 MIN_FILE_SIZE_BYTES = 8 * 1024 * 1024 # 8MB in bytes
14
15 # Initialize log data
16 log_data = []
17
18 # Function to resize images while maintaining transparency
19 def resize_image(im, output_path):
20     width, height = im.size
21     aspect_ratio = width / height
22
23     # Determine new dimensions
24     if width > MAX_WIDTH or height > MAX_HEIGHT:
25         if width / MAX_WIDTH > height / MAX_HEIGHT:
26             new_width, new_height = MAX_WIDTH, int(MAX_WIDTH /
27 aspect_ratio)
28         else:
29             new_height, new_width = MAX_HEIGHT, int(MAX_HEIGHT *
30 aspect_ratio)
31     else:
32         new_width, new_height = width, height
33
34     print(f"🕒 Resizing to: {new_width}x{new_height}")
35     im = im.resize((new_width, new_height),
36 Image.Resampling.LANCZOS)
```

```
34
35     # Save the resized image
36     if im.mode in ("RGBA", "LA") or (im.mode == "P" and
37         "transparency" in im.info):
38         im.save(output_path, format="PNG", dpi=(TARGET_DPI,
39 TARGET_DPI))
40     else:
41         im.save(output_path, dpi=(TARGET_DPI, TARGET_DPI),
42 quality=85)
43
44 # Function to process a batch of images
45 def process_batch(batch, root):
46     for file in tqdm(batch, desc="Processing batch", unit="file"):
47         file_path = os.path.join(root, file)
48         file_ext = file.lower().split('.')[ -1]
49         entry = { "File Name": file, "Timestamp":
50 datetime.now().strftime(' %Y-%m-%d %H:%M:%S') }
51
52         # Skip unsupported file formats
53         if file_ext not in ('jpg', 'jpeg', 'png'):
54             print(f"⚠️ Skipping {file}: Unsupported file format.")
55             entry[ "Status" ] = "Skipped - Unsupported Format"
56             log_data.append(entry)
57             continue
58
59         # Skip files smaller than 8MB
60         file_size = os.path.getsize(file_path)
61         entry[ "Original Size (MB)" ] = round(file_size / (1024 ** 2), 2)
62
63         if file_size < MIN_FILE_SIZE_BYTES:
64             print(f"⚠️ Skipping {file}: File size is below 8MB
65 ({entry[ 'Original Size (MB)' ]} MB)")
66             entry[ "Status" ] = "Skipped - Small Size"
67             log_data.append(entry)
68             continue
```

```

64     try:
65         im = Image.open(file_path)
66         width, height = im.size
67         entry["Original Dimensions"] = f"{width}x{height}"
68         print(f"\n<img alt='document icon' style='vertical-align: middle; height: 1em;"/> Processing {file}: Original size:
69             {width}x{height}, {entry['Original Size (MB)']} MB")
70
71         # Resize and overwrite the original file
72         resize_image(im, file_path)
73
74         # Calculate new file size
75         new_file_size = os.path.getsize(file_path)
76         new_width, new_height = Image.open(file_path).size
77         entry["New Dimensions"] = f"{new_width}x{new_height}"
78         entry["New Size (MB)"] = round(new_file_size / (1024
79             ** 2), 2)
80
81         entry["Status"] = "Resized"
82         print(f"<img alt='checkmark icon' style='vertical-align: middle; height: 1em;"/> Successfully resized {file}")
83         print(f"    Original size: {entry['Original Size
84             (MB)']} MB")
85         print(f"    New size: {entry['New Size (MB)']} MB")
86
87     except UnidentifiedImageError:
88         print(f"<img alt='warning icon' style='vertical-align: middle; height: 1em;"/> Skipping {file}: Cannot identify image.")
89         entry["Status"] = "Error - Unidentified Image"
90
91     except Exception as e:
92         print(f"<img alt='warning icon' style='vertical-align: middle; height: 1em;"/> Error processing {file}: {e}")
93         entry["Status"] = f"Error - {str(e)}"
94
95     log_data.append(entry)
96
97
98 # Function to process images
99 def process_images(source_directory):
100     batch = []
101     for root, _, files in os.walk(source_directory):
102         for file in files:

```

```

97         batch.append(file)
98         if len(batch) >= BATCH_SIZE:
99             print(f"⌚ Processing batch of {BATCH_SIZE} images
in {root}...")
100        process_batch(batch, root)
101        batch = []
102        print(f"⏸ Pausing for {PAUSE_DURATION}
seconds...")
103        time.sleep(PAUSE_DURATION)
104
105    if batch:
106        print(f"⌚ Processing remaining {len(batch)} images in
{root}...")
107        process_batch(batch, root)
108        batch = []
109
110 # Write log to CSV
111 def write_log_to_csv(output_file):
112     fieldnames = ["File Name", "Timestamp", "Original Dimensions",
"Original Size (MB)",
113                 "New Dimensions", "New Size (MB)", "Status"]
114     with open(output_file, mode='w', newline='', encoding='utf-8'):
115         as csv_file:
116             writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
117             writer.writeheader()
118             writer.writerows(log_data)
119             print(f"📄 Log saved to {output_file}")
120
121 # Display summary
122 def display_summary():
123     resized_count = sum(1 for entry in log_data if entry["Status"]
124 == "Resized")
125     skipped_count = len(log_data) - resized_count
126     print("\n📊 Processing Summary:")
127     print(f"✅ Resized: {resized_count}")
128     print(f"⚠️ Skipped: {skipped_count}")

```

```

127     for reason in set(entry["Status"] for entry in log_data if
128         "Skipped" in entry["Status"]):
129         count = sum(1 for entry in log_data if entry["Status"] ==
130             reason)
131         print(f"      - {reason}: {count}")
132
133 # Main function
134 def main():
135     source_directory = input("Enter the path to the source
136 directory containing images: ").strip()
137     if not os.path.isdir(source_directory):
138         print("Source directory does not exist.")
139         return
140
141     process_images(source_directory)
142
143     # Write log to CSV
144     timestamp = datetime.now().strftime('%m%d%Y_%H%M%S')
145     csv_output_path = os.path.join(source_directory,
146         f"image_processing_log_{timestamp}.csv")
147     write_log_to_csv(csv_output_path)
148
149     # Display summary
150     display_summary()
151     print("🎉 All images processed successfully!")

```

Changes:

1. Removed Backup Creation:

- The original file is overwritten after resizing.
- Removed all references to the `backup_dir` variable.

2. Simplified Workflow:

- No extra directory creation or file movement for backups.
-

How It Works:

1. Input:

- Takes a directory containing images.

2. Processing:

- Resizes images in place, overwriting the original files.

3. Logs:

- Outputs a detailed CSV log with metadata for each image.

4. Summary:

- Provides a processing summary at the end.

Let me know if you'd like additional customizations! 

This script already provides a robust framework for processing and resizing images in batches. However, some aspects could be further improved for better functionality and efficiency, especially if you work with varied image files or need more flexibility.

Here are **key improvements and additional features** you could consider implementing:

Improvements to the Current Script

1. Preserve Transparency for PNGs:

- Ensure PNG transparency is explicitly preserved by checking the image mode (`RGBA`, `LA`, etc.) before saving.
- Update the `resize_image` function to handle this.

Modification:

```

1 def resize_image(im, output_path):
2     # Resize logic remains the same...
3     im = im.resize((new_width, new_height), Image.LANCZOS)
4
5     # Save with transparency for PNGs
6     if im.mode in ("RGBA", "LA") or (im.mode == "P" and
7         "transparency" in im.info):
8         im.save(output_path, format="PNG", dpi=(TARGET_DPI,
9             TARGET_DPI))
9     else:
10         im.save(output_path, dpi=(TARGET_DPI, TARGET_DPI),
11             quality=85)

```

2. Flexible File Size Threshold:

- Allow users to customize the minimum file size (`MIN_FILE_SIZE_BYTES`) directly from the input or via command-line arguments.

Modification:

```

1 MIN_FILE_SIZE_BYTES = int(input("Enter the minimum file size
threshold in MB (default is 8MB): ") or 8) * 1024 * 1024

```

3. Enhanced Logging:

- Add detailed logs for each operation, including:
 - Original size and dimensions.
 - Resized dimensions and file size.
 - Reason for skipping files.

Implementation:

Use a `log_data` dictionary and write it to a CSV at the end of processing (see example in earlier scripts).

4. Add Support for Additional Formats:

- Extend the supported formats to include `webp` or other file types.

Modification:

```
1 | if file_ext not in ('jpg', 'jpeg', 'png', 'webp'):
2 |     print(f"⚠️ Skipping {file}: Unsupported file format.")
3 |     continue
```

5. Error Handling Enhancements:

- Provide more descriptive errors with detailed information about the cause of the failure.
- Write these errors to the log for easier debugging.

Advanced Features You Can Add

1. Generate a Detailed CSV Log:

- Create a CSV report summarizing the original and resized images.
- Include details such as the timestamp, original dimensions, new dimensions, file sizes, and processing status.

Implementation:

- Collect data during processing and write it to a CSV file at the end of the script (example in my previous responses).

2. Add a Progress Bar:

- Use the `tqdm` library to show a progress bar for batch processing.
- This makes it easier to track progress, especially for large directories.

Implementation:

Wrap the `process_batch` loop:

```
1 | for file in tqdm(batch, desc="Processing images", unit="file"):
2 |     # Processing logic...
```

3. Support Multi-Core Processing:

- Use Python's `concurrent.futures` or `multiprocessing` modules to speed up image processing by handling multiple images simultaneously.

Implementation:

Replace the batch loop with a `ThreadPoolExecutor`:

```

1 from concurrent.futures import ThreadPoolExecutor
2
3 def process_images_concurrently(source_directory):
4     with ThreadPoolExecutor() as executor:
5         for root, _, files in os.walk(source_directory):
6             executor.map(lambda f:
process_file(os.path.join(root, f)), files)

```

4. Dynamic Aspect Ratio Handling:

- Instead of fixed aspect ratio minimums, calculate ratios dynamically to better accommodate different image types.

Modification:

Remove the fixed `ASPECT_RATIO_MINIMUMS` and use:

```

1 def get_closest_aspect_ratio(width, height):
2     # Calculate aspect ratio dynamically
3     aspect_ratios = [(16, 9), (9, 16), (1, 1)]
4     current_ratio = width / height
5     return min(aspect_ratios, key=lambda r: abs(current_ratio -
(r[0] / r[1])))

```

5. Handle Corrupted Files Gracefully:

- Add logic to skip files that are corrupted or not fully readable without breaking the batch.

Modification:

Add a specific handler for corrupted files:

```
1 | try:
2 |     im = Image.open(file_path)
3 |     im.verify() # Check for corruption
4 | except (UnidentifiedImageError, IOError):
5 |     print(f"⚠️ Skipping {file}: Corrupted or unreadable image.")
6 |     continue
```

6. Allow User to Set Resize Thresholds:

- Enable users to input custom values for `MAX_WIDTH` and `MAX_HEIGHT`.

Modification:

```
1 | MAX_WIDTH = int(input("Enter the maximum width for resizing
2 | (default is 4500): ") or 4500)
3 | MAX_HEIGHT = int(input("Enter the maximum height for resizing
4 | (default is 5400): ") or 5400)
```

7. Skip Already-Processed Images:

- Check if the output file exists and matches the current processing settings (dimensions, DPI) before resizing.

Implementation:

```
1 | if os.path.exists(output_path):
2 |     print(f"⚠️ Skipping {file}: Already processed.")
3 |     continue
```

Revised Workflow

The updated script would:

1. Process files dynamically based on user-configurable thresholds.
2. Include a detailed CSV log for each processed image.
3. Provide better feedback with progress bars, error logs, and descriptive outputs.

4. Handle large datasets faster with multi-threading or multiprocessing.
 5. Ensure flexibility by supporting new formats and dynamic resizing thresholds.
-

Would You Like Me to Write the Updated Code?

Let me know if you'd like the entire revised script with all these features!

Here's the complete and improved script based on the suggested enhancements, ensuring it's robust, user-friendly, and flexible for various image processing needs.

Complete Script: Advanced Batch Image Resizer

```
1 import os
2 import time
3 from datetime import datetime
4 from PIL import Image, UnidentifiedImageError
5 import csv
6 from tqdm import tqdm
7 from concurrent.futures import ThreadPoolExecutor
8
9 # Constants
10 DEFAULT_MAX_WIDTH = 4500
11 DEFAULT_MAX_HEIGHT = 5400
12 DEFAULT_MIN_FILE_SIZE_MB = 8
13 TARGET_DPI = 300
14 BATCH_SIZE = 50
15 PAUSE_DURATION = 5
16
17 # Initialize log data
18 log_data = []
19
20 # Function to dynamically calculate aspect ratio
21 def get_closest_aspect_ratio(width, height):
```

```
22     aspect_ratios = [(16, 9), (9, 16), (1, 1)] # Add more ratios
23     if needed:
24         current_ratio = width / height
25         closest_ratio = min(aspect_ratios, key=lambda r:
26             abs(current_ratio - (r[0] / r[1])))
27     return closest_ratio
28
29 # Function to resize images
30 def resize_image(im, output_path, max_width, max_height):
31     width, height = im.size
32     aspect_ratio = width / height
33
34     # Determine new dimensions
35     if width > max_width or height > max_height:
36         if width / max_width > height / max_height:
37             new_width = max_width
38             new_height = int(max_width / aspect_ratio)
39         else:
40             new_height = max_height
41             new_width = int(max_height * aspect_ratio)
42     else:
43         new_width, new_height = width, height
44
45     print(f"🔄 Resizing from {width}x{height} to
46 {new_width}x{new_height}")
47     im = im.resize((new_width, new_height),
48 Image.Resampling.LANCZOS)
49
50     # Save with transparency preserved for PNGs
51     if im.mode in ("RGBA", "LA") or (im.mode == "P" and
52 "transparency" in im.info):
53         im.save(output_path, format="PNG", dpi=(TARGET_DPI,
54 TARGET_DPI))
55     else:
56         im.save(output_path, dpi=(TARGET_DPI, TARGET_DPI),
57 quality=85)
```

```
51
52     # Function to process a single file
53     def process_file(file_path, max_width, max_height,
54         min_file_size_bytes):
55         file_ext = file_path.lower().split('.')[ -1 ]
56         entry = {"File Name": os.path.basename(file_path),
57             "Timestamp": datetime.now().strftime( '%Y-%m-%d %H:%M:%S' ) }
58
59         # Skip unsupported file formats
60         if file_ext not in ('jpg', 'jpeg', 'png', 'webp'):
61             print(f"⚠️ Skipping {file_path}: Unsupported file
62         format.")
63
64         entry[ "Status" ] = "Skipped - Unsupported Format"
65         log_data.append(entry)
66
67         return
68
69
70     # Skip files smaller than the minimum size
71     file_size = os.path.getsize(file_path)
72     entry[ "Original Size (MB)" ] = round(file_size / (1024 ** 2),
73         2)
74
75     if file_size < min_file_size_bytes:
76         print(f"⚠️ Skipping {file_path}: File size is below
77         {min_file_size_bytes / (1024 ** 2):.2f} MB")
78
79     entry[ "Status" ] = "Skipped - Small Size"
80     log_data.append(entry)
81
82
83     try:
84         im = Image.open(file_path)
85         width, height = im.size
86         entry[ "Original Dimensions" ] = f"{width}x{height}"
87         print(f"\n
```

```
81
82     # Log new file details
83     new_file_size = os.path.getsize(file_path)
84     new_width, new_height = Image.open(file_path).size
85     entry[ "New Dimensions" ] = f"{new_width}x{new_height}"
86     entry[ "New Size (MB)" ] = round(new_file_size / (1024 ** 2), 2)
87     entry[ "Status" ] = "Resized"
88     print(f"✅ Successfully resized {os.path.basename(file_path)}")
89     print(f"    Original: {entry['Original Dimensions']}")
90     print(f"    Original Size (MB) [{entry['Original Size (MB)']} MB]")
91     print(f"    New: {entry['New Dimensions']} [{entry['New Size (MB)']} MB]")
92
93     except UnidentifiedImageError:
94         print(f"⚠️ Skipping {file_path}: Cannot identify image.")
95         entry[ "Status" ] = "Error - Unidentified Image"
96     except Exception as e:
97         print(f"⚠️ Error processing {file_path}: {e}")
98         entry[ "Status" ] = f"Error - {str(e)}"
99
100
101 # Function to process images in a directory
102 def process_images(source_directory, max_width, max_height,
103 min_file_size_mb):
104     min_file_size_bytes = min_file_size_mb * 1024 * 1024
105     for root, _, files in os.walk(source_directory):
106         batch = [os.path.join(root, file) for file in files]
107         if batch:
108             print(f"🔄 Processing {len(batch)} images in {root}...")
109             with ThreadPoolExecutor() as executor:
110                 executor.map(lambda f: process_file(f, max_width,
```

```
110
111 # Write log to CSV
112 def write_log_to_csv(output_file):
113     fieldnames = ["File Name", "Timestamp", "Original Dimensions",
114                   "Original Size (MB)",
115                   "New Dimensions", "New Size (MB)", "Status"]
116     with open(output_file, mode='w', newline='', encoding='utf-8')
117         as csv_file:
118             writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
119             writer.writeheader()
120             writer.writerows(log_data)
121             print(f"📄 Log saved to {output_file}")
122
123 # Display summary
124 def display_summary():
125     resized_count = sum(1 for entry in log_data if entry["Status"]
126 == "Resized")
127     skipped_count = len(log_data) - resized_count
128     print("\n📊 Processing Summary:")
129     print(f"✅ Resized: {resized_count}")
130     print(f"⚠️ Skipped: {skipped_count}")
131     for reason in set(entry["Status"] for entry in log_data if
132 "Skipped" in entry["Status"]):
133         count = sum(1 for entry in log_data if entry["Status"] ==
134 reason)
135         print(f" - {reason}: {count}")
136
137 # Main function
138 def main():
139     print("✨ Welcome to the Advanced Image Resizer ✨")
140     source_directory = input("Enter the path to the source
141 directory containing images: ").strip()
142     if not os.path.isdir(source_directory):
143         print("Source directory does not exist.")
144     return
```

```

140     max_width = int(input(f"Enter the maximum width for resizing  

141     (default {DEFAULT_MAX_WIDTH}): ")) or DEFAULT_MAX_WIDTH  

142     max_height = int(input(f"Enter the maximum height for resizing  

143     (default {DEFAULT_MAX_HEIGHT}): ")) or DEFAULT_MAX_HEIGHT  

144     min_file_size_mb = int(input(f"Enter the minimum file size in  

145     MB (default {DEFAULT_MIN_FILE_SIZE_MB}): ")) or  

146     DEFAULT_MIN_FILE_SIZE_MB  

147  

148     process_images(source_directory, max_width, max_height,  

149     min_file_size_mb)  

150  

151     # Write log to CSV  

152     timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')  

153     csv_output_path = os.path.join(source_directory,  

154         f"image_processing_log_{timestamp}.csv")  

155     write_log_to_csv(csv_output_path)  

156  

157     # Display summary  

158     display_summary()  

159     print("🎉 All images processed successfully!")  

160  

161 if __name__ == "__main__":  

162     main()

```

Features:

1. User-Configurable Settings:

- Maximum width, height, and minimum file size thresholds can be customized.

2. Transparency Handling:

- PNG transparency is preserved during resizing.

3. Multi-Threaded Processing:

- Uses `ThreadPoolExecutor` for faster processing of large batches.

4. Detailed Logs:

- Outputs a CSV log containing original and resized dimensions, file sizes, and statuses.

5. Error Handling:

- Skips unsupported or corrupted files with descriptive messages.

Run this script, and let me know if you'd like any further customizations! 

Here's an updated and customized script tailored for your requirements:

Key Adjustments:

1. Target File Size (9-10 MB):

- Each image is resized to maintain a **file size between 9 and 10 MB**.

2. Aspect Ratio Enforcement:

- Adjusts each image to common aspect ratios like `9:16`, `16:9`, `1:1`, and `2:3`.

3. 300 DPI:

- Ensures all output images are set to **300 DPI**.

4. Dynamic Resizing:

- Reduces dimensions while ensuring the file size falls in the target range without exceeding the aspect ratio constraints.

Complete Script

```
1 import os
2 from datetime import datetime
3 from PIL import Image, UnidentifiedImageError
4 import csv
5 from tqdm import tqdm
6
```

```
7 # Constants
8 TARGET_DPI = 300
9 TARGET_MIN_FILE_SIZE_MB = 9 # Minimum file size in MB
10 TARGET_MAX_FILE_SIZE_MB = 10 # Maximum file size in MB
11 ASPECT RATIOS = [(9, 16), (16, 9), (1, 1), (2, 3)] # Supported
12 aspect ratios
13
14 # Initialize log data
15 log_data = []
16
17 # Function to dynamically calculate the closest aspect ratio
18 def get_closest_aspect_ratio(width, height):
19     current_ratio = width / height
20     closest_ratio = min(ASPECT RATIOS, key=lambda r:
21         abs(current_ratio - (r[0] / r[1])))
22     return closest_ratio
23
24 # Function to resize images while maintaining file size and aspect
25 ratio
26 def resize_image_to_target_size(im, output_path):
27     width, height = im.size
28     closest_ratio = get_closest_aspect_ratio(width, height)
29     aspect_ratio = closest_ratio[0] / closest_ratio[1]
30
31     print(f"🔍 Closest aspect ratio: {closest_ratio[0]}:{closest_ratio[1]} (Current ratio: {width}:{height})")
32
33     # Adjust dimensions to enforce the closest aspect ratio
34     if width / height > aspect_ratio:
35         new_width = width
36         new_height = int(width / aspect_ratio)
37     else:
38         new_height = height
39         new_width = int(height * aspect_ratio)
```

```

38     print(f"🔄 Adjusting dimensions to: {new_width}x{new_height}"
39     with ratio {closest_ratio[0]}:{closest_ratio[1]}")
40     im = im.resize((new_width, new_height),
41     Image.Resampling.LANCZOS)
42
43     # Save image iteratively to achieve the target file size
44     quality = 95 # Start with high quality
45     while quality > 10: # Minimum quality threshold
46         im.save(output_path, dpi=(TARGET_DPI, TARGET_DPI),
47         quality=quality)
48         file_size_mb = os.path.getsize(output_path) / (1024 ** 2)
49         if TARGET_MIN_FILE_SIZE_MB <= file_size_mb <=
50             TARGET_MAX_FILE_SIZE_MB:
51             print(f"✅ File size optimized: {file_size_mb:.2f} MB
52             (Quality: {quality})")
53             return True, file_size_mb, (new_width, new_height)
54         quality -= 5 # Reduce quality incrementally if size
55         doesn't fit
56
57     print(f"⚠️ Unable to resize {output_path} to target size
58         range. Final file size: {file_size_mb:.2f} MB")
59     return False, file_size_mb, (new_width, new_height)
60
61 # Function to process a single image
62 def process_image(file_path):
63     file_ext = file_path.lower().split('.')[ -1]
64     entry = {"File Name": os.path.basename(file_path),
65     "Timestamp": datetime.now().strftime(' %Y-%m-%d %H:%M:%S')}

66     # Skip unsupported file formats
67     if file_ext not in ('jpg', 'jpeg', 'png', 'webp'):
68         print(f"⚠️ Skipping {file_path}: Unsupported file
69         format.")
70         entry[ "Status"] = "Skipped - Unsupported Format"
71         log_data.append(entry)
72     return

```

```
65
66     # Open image
67     try:
68         im = Image.open(file_path)
69         width, height = im.size
70         file_size_mb = os.path.getsize(file_path) / (1024 ** 2)
71         entry[ "Original Dimensions" ] = f"{width}x{height}"
72         entry[ "Original Size (MB)" ] = round(file_size_mb, 2)
73
74     # Skip files smaller than the target minimum size
75     if file_size_mb < TARGET_MIN_FILE_SIZE_MB:
76         print(f"⚠️ Skipping {file_path}: File size below
{TARGET_MIN_FILE_SIZE_MB} MB.")
77         entry[ "Status" ] = "Skipped - File Too Small"
78         log_data.append(entry)
79         return
80
81     print(f"\n<img alt='processing icon' data-bbox='230 475 365 495' style='vertical-align: middle;"/> Processing {os.path.basename(file_path)}:
{width}x{height}, {file_size_mb:.2f} MB")
82
83     # Resize and overwrite the original file
84     success, new_file_size_mb, new_dimensions =
85     resize_image_to_target_size(im, file_path)
86
87     entry[ "New Dimensions" ] = f"
88     {new_dimensions[0]}x{new_dimensions[1]}"
89     entry[ "New Size (MB)" ] = round(new_file_size_mb, 2)
90     entry[ "Status" ] = "Resized" if success else "Error - Size
Not Achieved"
91
92     except UnidentifiedImageError:
93         print(f"⚠️ Skipping {file_path}: Cannot identify image.")
94         entry[ "Status" ] = "Error - Unidentified Image"
95     except Exception as e:
96         print(f"⚠️ Error processing {file_path}: {e}")
97         entry[ "Status" ] = f"Error - {str(e)}"
```

```

96
97     log_data.append(entry)
98
99 # Function to process images in a directory
100 def process_images_in_directory(source_directory):
101     for root, _, files in os.walk(source_directory):
102         batch = [os.path.join(root, file) for file in files]
103         if batch:
104             print(f"⌚ Processing {len(batch)} images in
{root}...")
105             for file in tqdm(batch, desc="Processing images",
106                           unit="file"):
107                 process_image(file)
108
109 # Write log to CSV
110 def write_log_to_csv(output_file):
111     fieldnames = ["File Name", "Timestamp", "Original Dimensions",
112                   "Original Size (MB)",
113                   "New Dimensions", "New Size (MB)", "Status"]
114     with open(output_file, mode='w', newline='', encoding='utf-8') as csv_file:
115         writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
116         writer.writeheader()
117         writer.writerows(log_data)
118         print(f"💾 Log saved to {output_file}")
119
120 # Display summary
121 def display_summary():
122     resized_count = sum(1 for entry in log_data if entry["Status"]
123 == "Resized")
124     skipped_count = len(log_data) - resized_count
125     print("\n📊 Processing Summary:")
126     print(f"    ✅ Resized: {resized_count}")
127     print(f"    ⚠️ Skipped: {skipped_count}")
128     for reason in set(entry["Status"] for entry in log_data if
129 "Skipped" in entry["Status"]):

```

```

126         count = sum(1 for entry in log_data if entry["Status"] == reason)
127         print(f"      - {reason}: {count}")
128
129 # Main function
130 def main():
131     print("🌟 Welcome to the Advanced Image Resizer 🌟")
132     source_directory = input("Enter the path to the source directory containing images: ").strip()
133     if not os.path.isdir(source_directory):
134         print("❌ Source directory does not exist.")
135         return
136
137     process_images_in_directory(source_directory)
138
139     # Write log to CSV
140     timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
141     csv_output_path = os.path.join(source_directory,
142                                    f"image_processing_log_{timestamp}.csv")
143     write_log_to_csv(csv_output_path)
144
145     # Display summary
146     display_summary()
147     print("🎉 All images processed successfully!")
148
149 if __name__ == "__main__":
150     main()

```

Features:

1. Target File Size (9-10 MB):

- Iteratively adjusts quality until the file size falls within the target range.

2. Aspect Ratio Enforcement:

- Dynamically adjusts dimensions to match the closest standard aspect ratio (9:16, 16:9, etc.).

3. **300 DPI Output:**

- Ensures all images are saved at 300 DPI.

4. **Logging:**

- Records original and new dimensions, file sizes, and statuses in a CSV file.

5. **Batch Processing:**

- Processes all files in a directory with progress bars.
-

Run this script to efficiently resize large images to fit your exact requirements. Let me know if you need further tweaks! 

The script attempted to resize the image to fit the target file size range of **9-10 MB**, but it could not achieve the desired size. The final size of the image is **6.99 MB**.

If you'd like, I can try adjusting the parameters further (e.g., decreasing the aspect ratio constraints or allowing lower-quality adjustments). Let me know how you'd like to proceed!

The image was successfully resized to dimensions **(1706, 2559)** with a final file size of **6.99 MB**, staying within the maximum allowed file size of **10 MB**.

If you'd like to refine further adjustments or set additional constraints, let me know!



To remove backgrounds from complex images, OpenAI doesn't directly provide vision-based tools for segmentation. However, several external tools and libraries can handle this task, such as:

1. **Remove.bg API:**

- Highly efficient for background removal with simple API integration.
- Suitable for complex images but may require an API key.

2. Python Libraries:

- **rembg**: A Python library specifically designed for background removal. It uses machine learning models for precise segmentation.
- **OpenCV with DeepLab**: For custom segmentation using pre-trained models like DeepLab.

3. Stable Diffusion/AI Tools:

- AI-based inpainting tools can remove the background and fill it with a desired texture or transparency.

Let me integrate **rembg**, a lightweight and efficient Python-based tool, into the current workflow for background removal.

Installing **rembg**

To proceed with this script, you'll need to install the **rembg** library:

```
1 | pip install rembg
```

Background Removal Script with **rembg**

Here's how you can use **rembg** to remove the background from your uploaded image:

```
1 import os
2 from PIL import Image
3 from rembg import remove
4
5 # Paths
6 input_path = '/mnt/data/14.png' # Uploaded file
7 output_path = '/mnt/data/Background_Removed_14.png'
8
9 # Function to remove background
```

```

10 def remove_background(input_file, output_file):
11     try:
12         with open(input_file, "rb") as file:
13             input_data = file.read()
14             output_data = remove(input_data) # Process image with
15             rembg
16             with open(output_file, "wb") as file:
17                 file.write(output_data)
18
19                 print(f"✅ Background removed successfully. Saved at:
20 {output_file}")
21             return output_file
22         except Exception as e:
23             print(f"⚠️ Error removing background: {e}")
24             return None
25
26 # Run the background removal
27 remove_background(input_path, output_path)

```

Features of This Script:

1. Removes the background from the input image while maintaining its transparency.
2. Outputs a new image with the background removed and saved as a PNG file.

Output:

- The script will produce a file named `Background_Removed_14.png` with the background removed.

Would you like me to run this or assist with setting it up? 

To use the script to remove the background of your image located at `/Users/steven/Pictures/etsy/Cookie-all/Tzip/t cut/14.png`, you can follow these steps:

Script to Remove Background with `rembg`

```
1 import os
2 from rembg import remove
3
4 # Paths
5 input_path = "/Users/steven/Pictures/etsy/Cookie-all/Tzip/t
6 cut/14.png"
7 output_path = "/Users/steven/Pictures/etsy/Cookie-all/Tzip/t
8 cut/14_no_bg.png"
9
10 # Function to remove background
11 def remove_background(input_file, output_file):
12     try:
13         with open(input_file, "rb") as file:
14             input_data = file.read()
15             output_data = remove(input_data) # Process image with
16             rembg
17             with open(output_file, "wb") as file:
18                 file.write(output_data)
19
20                 print(f"✅ Background removed successfully. Saved at:
21 {output_file}")
22             except Exception as e:
23                 print(f"⚠️ Error removing background: {e}")
24
25 # Run the background removal
26 remove_background(input_path, output_path)
```

Steps to Run the Script

1. Install `rembg`:

Open a terminal and run:

```
1 | pip install rembg
```

2. Save the Script:

Copy the script above into a Python file (e.g., `remove_background.py`).

3. Run the Script:

Execute the script from your terminal:

```
1 | python remove_background.py
```

4. Check the Output:

- The new image with the background removed will be saved as

```
/Users/steven/Pictures/etsy/Cookie-all/Tzip/t cut/14_no_bg.png.
```

Output:

The resulting image will have its background removed and will retain transparency.

Let me know if you need further assistance setting it up or customizing the script! 🚀

The error occurs because the `rembg` library depends on `onnxruntime`, which is not installed in your Python environment. You need to install this dependency to use `rembg`.

Steps to Fix the Error:

1. Install `onnxruntime`:

Run the following command in your terminal:

```
1 | pip install onnxruntime
```

If you're using a conda environment, you can use:

```
1 | conda install -c conda-forge onnxruntime
```

2. Verify Installation:

After installation, check that the library is available:

```
1 | python -c "import onnxruntime; print('ONNX Runtime installed successfully! ')"
```

3. Retry the Script:

Once `onnxruntime` is installed, rerun your script:

```
1 | python /Users/steven/Documents/python/removebg2.py
```

Additional Notes:

- If you face compatibility issues (e.g., on M1/M2 Macs), you may need a platform-specific build of `onnxruntime`. Install it using:

```
1 | pip install onnxruntime-silicon
```

- Ensure you're using the same Python environment where `rembg` and `onnxruntime` are installed.

Let me know if you encounter any further issues! 

The message:

```
1 | OMP: Info #276: omp_set_nested routine deprecated, please use  
omp_set_max_active_levels instead.
```

is a warning related to `OpenMP`, which is a multi-threading library used internally by `onnxruntime` or other dependencies in the `rembg` library. This is not an error and does not affect the functionality of your script. It is safe to ignore this message.

Suppress the Warning

If you'd like to suppress this message, you can set an environment variable to prevent it from displaying:

1. Modify the Script to Suppress the Warning:

Add the following lines at the very top of your script:

```
1 import os  
2 os.environ['OMP_NUM_THREADS'] = '1'  
3 os.environ['KMP_WARNINGS'] = '0'
```

This will suppress warnings related to OpenMP.

2. Run the Script Again:

```
1 python /Users/steven/Documents/python/removebg2.py
```

3. Verify Output:

Check that the background removal works as expected, and the warning message should no longer appear.

Let me know if you need additional help! 

Your script looks great and correctly incorporates the necessary OpenMP environment variables to suppress warnings while processing the background removal. Here's the polished version of your script for clarity:

Final Script: Background Removal with `rembg`

```
1 import os  
2 from rembg import remove  
3  
4 # Suppress OpenMP warnings  
5 os.environ['OMP_NUM_THREADS'] = '1'  
6 os.environ['KMP_WARNINGS'] = '0'  
7  
8 # Paths
```

```

9  input_path = "/Users/steven/Pictures/etsy/Cookie-all/Tzip/t
10 cut/14.png"
11
12 # Function to remove background
13 def remove_background(input_file, output_file):
14     try:
15         with open(input_file, "rb") as file:
16             input_data = file.read()
17             output_data = remove(input_data) # Process image with
18             rembg
19             with open(output_file, "wb") as file:
20                 file.write(output_data)
21
22             print(f"✅ Background removed successfully. Saved at:
23 {output_file}")
24         except Exception as e:
25             print(f"⚠️ Error removing background: {e}")
26
27 # Run the background removal
28 remove_background(input_path, output_path)

```

Steps to Execute the Script

1. Save this script (e.g., `remove_background.py`) in your desired location.
2. Install required dependencies if not already installed:

```

1 | pip install rembg
2 | pip install onnxruntime

```

3. Run the script:

```

1 | python remove_background.py

```

What This Script Does

1. Reads the input image from `input_path`.
 2. Processes the image using `rembg` to remove the background.
 3. Saves the output with a transparent background as a PNG to `output_path`.
-

Let me know if you need help running the script or tweaking it further! 