PYTHON Bilder-Stapelverarbeitung

> Kurze Vorwarnung dass dies hier überwiegend KI-generiertes Zeug ist. Hat funktioniert, muss aber nicht erneut funktionieren. Es ist wichtig dass ihr in Grundzügen nachvollziehen könnt, was die Scripts tun, auch wenn sie generiert sind.
> Pastet diese Scripts z.B. in eine KI rein und lasst sie euch erklären. Easy as that... Nur kurz schildern wer ihr seid und was ihr beruflich macht und warum ihr euch jetzt unbedingt das Script anschauen müsst.
>
> Generatoren: ChatGPT, Claude.ai, manchmal Llama, je nachdem was so passte.

Umgang mit auch größeren Anzahlen an Bildern. Teils getestet an einem ganzen Terabyte Bildern.

Sortierung nach Bildgröße

Fest definierte Größe

ChatGPT 3.5 als Hilfestellung

Skript sammelt Bilder die kleiner als festgelegte Auflösung sind in einem anderen Ordner und belässt die, die größer sind im Ordner.

Das Beispiel hier trennt Bilder >4K von denen die es nicht sind.

```
param (
[string]$SourceFolder = "C:\Path\to\SourceFolder",
    [string]$DestinationFolder = "C:\Path\to\DestinationFolder",
    [int]$MinWidth = 3840,
    [int]$MinHeight = 2160
)

# Create the destination folder if it doesn't exist
if (-not (Test-Path $DestinationFolder)) {
```

```
    New-Item -ItemType Directory -Path $DestinationFolder | Out-Null
}

# Get all image files in the source folder and its subfolders
$images = Get-ChildItem -Path $SourceFolder -Filter "*.jpg" -File -Recurse

$totalImages = $images.Count
$processedImages = 0

# Process each image file
foreach ($image in $images) {
    Write-Host "Processing $($image.FullName)"

    # Use .NET classes to read image dimensions
    $imageStream = New-Object System.IO.FileStream($image.FullName,
[System.IO.FileMode]::Open)
    $imageBitmap = New-Object System.Drawing.Bitmap($imageStream)
    $width = $imageBitmap.Width
    $height = $imageBitmap.Height
    $imageStream.Close()

    # Check if the image is smaller than the specified dimensions
    if ($width -lt $MinWidth -or $height -lt $MinHeight) {
        $destinationPath = Join-Path -Path $DestinationFolder -ChildPath $image.Name
        Write-Host "Moving $($image.FullName) to $destinationPath"
        Move-Item -Path $image.FullName -Destination $destinationPath
    }

    $processedImages++
    $progress = [math]::Round(($processedImages / $totalImages) * 100, 2)
    Write-Progress -Activity "Moving images" -Status "Progress: $progress%" -
PercentComplete $progress
}

Write-Progress -Activity "Moving images" -Status "Progress: 100%" -PercentComplete 100
Write-Host "Image move complete!"
```

Auswahldialoge für Auflösung und Ordner

Claude 3.5 Sonnet

1. Mindestauflösung Höhe und Breite in Pixeln auswählen
2. Ordnerdialoge für Quell- und Zielordner wählen
3. Alle Bilder, die die Mindestauflösung haben oder überschreiten, werden in den gewählten Zielordner verschoben

Fehler sind in ./error_log.txt

```
Add-Type -AssemblyName System.Drawing
Add-Type -AssemblyName System.Windows.Forms

# Get the directory of the script
$scriptPath = Split-Path -Parent $MyInvocation.MyCommand.Path
$logFile = Join-Path $scriptPath "error_log.txt"

# Function to log errors
function Log-Error {
    param (
        [string]$message
    )
    $timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    "$timestamp - $message" | Out-File -FilePath $logFile -Append
}

# Function to show folder selection dialog
function Select-Folder {
    param (
        [string]$Description
    )
    $folderBrowser = New-Object System.Windows.Forms.FolderBrowserDialog
    $folderBrowser.Description = $Description
    $folderBrowser.RootFolder = [System.Environment+SpecialFolder]::MyComputer
    if ($folderBrowser.ShowDialog() -eq "OK") {
        return $folderBrowser.SelectedPath
    }
    return $null
}
```

```
# Prompt user for resolution
do {
    $targetWidth = Read-Host "Enter the target width in pixels (e.g., 3840 for 4K)"
} while (-not ($targetWidth -match '^\d+$'))

do {
    $targetHeight = Read-Host "Enter the target height in pixels (e.g., 2160 for 4K)"
} while (-not ($targetHeight -match '^\d+$'))

$targetWidth = [int]$targetWidth
$targetHeight = [int]$targetHeight

Write-Host "Target resolution: $targetWidth x $targetHeight"

# Prompt user to select source and destination folders
$sourceFolder = Select-Folder "Select the source folder containing images"
if (-not $sourceFolder) {
    Write-Host "Source folder selection cancelled. Exiting script."
    exit
}

$destinationFolder = Select-Folder "Select the destination folder for matching images"
if (-not $destinationFolder) {
    Write-Host "Destination folder selection cancelled. Exiting script."
    exit
}

# Create destination folder if it doesn't exist
if (!(Test-Path -Path $destinationFolder)) {
    New-Item -ItemType Directory -Path $destinationFolder | Out-Null
    Write-Host "Created destination folder: $destinationFolder"
}

# Get all image files in the source folder
$imageFiles = Get-ChildItem -Path $sourceFolder -Include *.jpg, *.jpeg, *.png, *.bmp -File
-Recurse

if ($imageFiles.Count -eq 0) {
```

```
    $message = "No image files found in $sourceFolder"
    Write-Host $message
    Log-Error $message
    exit
}

Write-Host "Found $($imageFiles.Count) image files to process"

# Initialize counter and total
$i = 0
$total = $imageFiles.Count
$movedCount = 0

# Initialize progress bar
Write-Progress -Activity "Processing Images" -Status "0% Complete" -PercentComplete 0

foreach ($file in $imageFiles) {
    try {
        $image = $null
        try {
            $image = [System.Drawing.Image]::FromFile($file.FullName)
            $width = $image.Width
            $height = $image.Height

            # Check if image meets or exceeds target resolution
            if ($width -ge $targetWidth -and $height -ge $targetHeight) {
                # Close the image before moving
                $image.Dispose()
                $image = $null

                # Move the file to the destination folder
                Move-Item -Path $file.FullName -Destination $destinationFolder -Force
                Write-Host "Moved: $($file.Name) (${width}x${height})"
                $movedCount++
            }
        }
        finally {
            # Ensure image is disposed even if an error occurs
            if ($image -ne $null) {
```

```
                $image.Dispose()
            }
        }
    }
    catch {
        $errorMessage = "Error processing $($file.Name): $_"
        Write-Host $errorMessage
        Log-Error $errorMessage
    }

    # Update progress
    $i++
    $percentComplete = ($i / $total) * 100
    Write-Progress -Activity "Processing Images" -Status "$i of $total processed" -
PercentComplete $percentComplete
}

Write-Progress -Activity "Processing Images" -Completed
Write-Host "Processing complete. Moved $movedCount out of $total images."
```

ungetestete BETA

Mit Claude Sonnet erweitert. Hinzu kommen Abfragen nach Ordnern und Auflösung, Erkennung der Bildschirmauflösung etc.

> ungetesteter direkter AI output

```
Add-Type -AssemblyName System.Drawing
Add-Type -AssemblyName System.Windows.Forms

# Get the directory of the script
$scriptPath = Split-Path -Parent $MyInvocation.MyCommand.Path
$logFile = Join-Path $scriptPath "error_log.txt"

# Function to log errors
function Log-Error {
    param (
        [string]$message
```

```
    )
    $timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    "$timestamp - $message" | Out-File -FilePath $logFile -Append
}

# Function to show folder selection dialog
function Select-Folder {
    param (
        [string]$Description
    )
    $folderBrowser = New-Object System.Windows.Forms.FolderBrowserDialog
    $folderBrowser.Description = $Description
    $folderBrowser.RootFolder = [System.Environment+SpecialFolder]::MyComputer
    if ($folderBrowser.ShowDialog() -eq "OK") {
        return $folderBrowser.SelectedPath
    }
    return $null
}

# Get current screen resolution
$currentResolution = (Get-WmiObject -Class Win32_VideoController).VideoModeDescription
$resolutionMatch = $currentResolution -match '(\d+)\s*x\s*(\d+)'
if ($resolutionMatch) {
    $currentWidth = [int]$Matches[1]
    $currentHeight = [int]$Matches[2]
    Write-Host "Current screen resolution: $currentWidth x $currentHeight"

    $useCurrentResolution = Read-Host "Do you want to use the current screen
resolution? (Y/N)"
    if ($useCurrentResolution -eq 'Y' -or $useCurrentResolution -eq 'y') {
        $targetWidth = $currentWidth
        $targetHeight = $currentHeight
    }
}

# If not using current resolution, prompt user for resolution
if (-not $targetWidth -or -not $targetHeight) {
    do {
```

```
        $targetWidth = Read-Host "Enter the target width in pixels (e.g., 3840 for 4K)"
    } while (-not ($targetWidth -match '^\d+$'))

    do {
        $targetHeight = Read-Host "Enter the target height in pixels (e.g., 2160 for
4K)"
    } while (-not ($targetHeight -match '^\d+$'))

    $targetWidth = [int]$targetWidth
    $targetHeight = [int]$targetHeight
}

Write-Host "Target resolution: $targetWidth x $targetHeight"

# Prompt user to select source and destination folders
$sourceFolder = Select-Folder "Select the source folder containing images"
if (-not $sourceFolder) {
    Write-Host "Source folder selection cancelled. Exiting script."
    exit
}

$destinationFolder = Select-Folder "Select the destination folder for matching images"
if (-not $destinationFolder) {
    Write-Host "Destination folder selection cancelled. Exiting script."
    exit
}

# Create destination folder if it doesn't exist
if (!(Test-Path -Path $destinationFolder)) {
    New-Item -ItemType Directory -Path $destinationFolder | Out-Null
    Write-Host "Created destination folder: $destinationFolder"
}

# Get all image files in the source folder
$imageFiles = Get-ChildItem -Path $sourceFolder -Include *.jpg, *.jpeg, *.png, *.bmp -
File -Recurse

if ($imageFiles.Count -eq 0) {
```

```
    $message = "No image files found in $sourceFolder"
    Write-Host $message
    Log-Error $message
    exit
}

Write-Host "Found $($imageFiles.Count) image files to process"

# Initialize counter and total
$i = 0
$total = $imageFiles.Count
$movedCount = 0

# Initialize progress bar
Write-Progress -Activity "Processing Images" -Status "0% Complete" -PercentComplete 0

foreach ($file in $imageFiles) {
    try {
        $image = $null
        try {
            $image = [System.Drawing.Image]::FromFile($file.FullName)
            $width = $image.Width
            $height = $image.Height

            # Check if image meets or exceeds target resolution
            if ($width -ge $targetWidth -and $height -ge $targetHeight) {
                # Close the image before moving
                $image.Dispose()
                $image = $null

                # Move the file to the destination folder
                Move-Item -Path $file.FullName -Destination $destinationFolder -Force
                Write-Host "Moved: $($file.Name) (${width}x${height})"
                $movedCount++
            }
        }
        finally {
            # Ensure image is disposed even if an error occurs
```

```
            if ($image -ne $null) {
                $image.Dispose()
            }
        }
    }
    catch {
        $errorMessage = "Error processing $($file.Name): $_"
        Write-Host $errorMessage
        Log-Error $errorMessage
    }

    # Update progress
    $i++
    $percentComplete = ($i / $total) * 100
    Write-Progress -Activity "Processing Images" -Status "$i of $total processed" -
PercentComplete $percentComplete
}

Write-Progress -Activity "Processing Images" -Completed
Write-Host "Processing complete. Moved $movedCount out of $total images."
```

Bilder <4K verschieben

> PYTHON, kein Powershell

pip install Pillow tqdm

```
import os
from PIL import Image
import shutil
from pathlib import Path
from concurrent.futures import ThreadPoolExecutor
from tqdm import tqdm
# Import and disable the DecompressionBombWarning
from PIL import Image, ImageFile
```

```
Image.MAX_IMAGE_PIXELS = None  # Disable the warning
ImageFile.LOAD_TRUNCATED_IMAGES = True  # Handle truncated images

def get_image_resolution(image_path):
    """Get the resolution of an image file."""
    try:
        with Image.open(image_path) as img:
            return img.size
    except Exception as e:
        print(f"Error reading {image_path}: {str(e)}")
        return None

def is_4k_or_higher(width, height):
    """Check if the resolution is 4K (3840x2160) or higher."""
    return width >= 3840 and height >= 2160

def process_image(image_path, destination_folder):
    """Process a single image and move it if not 4K."""
    try:
        resolution = get_image_resolution(image_path)
        if resolution is None:
            return

        width, height = resolution
        if not is_4k_or_higher(width, height):
            # Create destination folder if it doesn't exist
            Path(destination_folder).mkdir(parents=True, exist_ok=True)

            # Get destination path
            dest_path = os.path.join(destination_folder, os.path.basename(image_path))

            # Move file, overwriting if it exists
            shutil.move(image_path, dest_path)

    except Exception as e:
        print(f"Error processing {image_path}: {str(e)}")

def main():
    # Configuration
```

```
    source_folder = input("Enter the source folder path: ")
    destination_folder = input("Enter the destination folder for non-4K images: ")

    # Validate source folder
    if not os.path.exists(source_folder):
        print("Source folder does not exist!")
        return

    # Get list of image files
    image_files = [
        os.path.join(source_folder, f)
        for f in os.listdir(source_folder)
        if f.lower().endswith(('.png', '.jpg', '.jpeg'))
    ]

    if not image_files:
        print("No image files found in the source folder!")
        return

    print(f"Found {len(image_files)} images. Processing...")

    # Process images using thread pool for better performance
    with ThreadPoolExecutor(max_workers=8) as executor:
        list(tqdm(
            executor.map(
                lambda x: process_image(x, destination_folder),
                image_files
            ),
            total=len(image_files),
            desc="Processing images"
        ))

    print("Done! All non-4K images have been moved.")

if __name__ == "__main__":
    main()
```

Bilder >= 4K verschieben

PYTHON
pip install Pillow tqdm

```
import os
from PIL import Image
import shutil
from pathlib import Path
from concurrent.futures import ThreadPoolExecutor
from tqdm import tqdm
import warnings
import logging
import sys
from datetime import datetime
# Import and disable the DecompressionBombWarning
from PIL import Image, ImageFile
Image.MAX_IMAGE_PIXELS = None  # Disable the warning
ImageFile.LOAD_TRUNCATED_IMAGES = True  # Handle truncated images

# Get the script's filename without extension to create log filename
script_path = sys.argv[0]
script_name = os.path.splitext(os.path.basename(script_path))[0]
log_filename = f"{script_name}_errors.log"

# Setup logging
logging.basicConfig(
    filename=log_filename,
    level=logging.WARNING,
    format='%(asctime)s - %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S'
)

# Custom warning handler to redirect warnings to logging
def handle_warning(message, category, filename, lineno, file=None, line=None):
    logging.warning(f"Warning: {message}")

# Replace the default warning handler
warnings.showwarning = handle_warning

def get_image_resolution(image_path):
```

```
    """Get the resolution of an image file."""
    try:
        with Image.open(image_path) as img:
            return img.size
    except Exception as e:
        logging.error(f"Error reading image {image_path}: {str(e)}")
        return None

def is_4k_or_higher(width, height):
    """Check if the resolution is 4K (3840x2160) or higher."""
    return width >= 3840 and height >= 2160

def process_image(image_path, destination_folder):
    """Process a single image and move it if it IS 4K or higher."""
    try:
        resolution = get_image_resolution(image_path)
        if resolution is None:
            return

        width, height = resolution
        if is_4k_or_higher(width, height):
            # Create destination folder if it doesn't exist
            Path(destination_folder).mkdir(parents=True, exist_ok=True)

            # Get destination path
            dest_path = os.path.join(destination_folder, os.path.basename(image_path))

            # Move file, overwriting if it exists
            shutil.move(image_path, dest_path)

    except Exception as e:
        logging.error(f"Error processing {image_path}: {str(e)}")

def main():
    # Log script start
    logging.info(f"Script started at {datetime.now()}")

    # Configuration
    source_folder = input("Enter the source folder path: ")
```

```
    destination_folder = input("Enter the destination folder for 4K and higher resolution images: ")

    # Log folders
    logging.info(f"Source folder: {source_folder}")
    logging.info(f"Destination folder: {destination_folder}")

    # Validate source folder
    if not os.path.exists(source_folder):
        logging.error("Source folder does not exist!")
        print("Source folder does not exist!")
        return

    # Get list of image files
    image_files = [
        os.path.join(source_folder, f)
        for f in os.listdir(source_folder)
        if f.lower().endswith(('.png', '.jpg', '.jpeg'))
    ]

    if not image_files:
        logging.warning("No image files found in the source folder!")
        print("No image files found in the source folder!")
        return

    print(f"Found {len(image_files)} images. Processing...")
    logging.info(f"Found {len(image_files)} images to process")

    # Process images using thread pool for better performance
    with ThreadPoolExecutor(max_workers=8) as executor:
        list(tqdm(
            executor.map(
                lambda x: process_image(x, destination_folder),
                image_files
            ),
            total=len(image_files),
            desc="Processing images"
        ))
```

```
    logging.info(f"Script completed at {datetime.now()}")
    print("Done! All 4K and higher resolution images have been moved.")
    print(f"Check {log_filename} for any errors that occurred during processing.")

if __name__ == "__main__":
    main()
```

nach Seitenverhältnis sortieren

PYTHON

```
import os
import tkinter as tk
from tkinter import filedialog
from PIL import Image, ImageFile
from tqdm import tqdm
import re
import shutil

# Disable decompression bomb warning and increase file reading limit
ImageFile.LOAD_TRUNCATED_IMAGES = True
Image.MAX_IMAGE_PIXELS = None

# Define common aspect ratios with their tolerance
COMMON_ASPECT_RATIOS = {
    # Widescreen and Ultra-wide
    'widescreen_16_9': (16/9, 0.02),     # Standard widescreen
    'ultrawide_21_9': (21/9, 0.02),      # Ultra-wide
    'superwide_32_9': (32/9, 0.02),      # Super ultra-wide

    # Portrait Ratios
    'portrait_9_16': (9/16, 0.02),       # Vertical smartphone
    'portrait_10_16': (10/16, 0.02),     # Alternative portrait

    # Square and near-square
    'square_1_1': (1, 0.02),             # Perfect square
```

```
    'classic_4_3': (4/3, 0.02),          # Classic 4:3
    'classic_3_2': (3/2, 0.02),          # Classic film/print
    'large_format_5_4': (5/4, 0.02),     # Large format photography

    # Other common ratios
    'widescreen_16_10': (16/10, 0.02),   # Slightly wider widescreen
}

def sanitize_path(path):
    """
    Sanitize path to be safe for Windows file system
    """
    # Replace invalid path characters
    sanitized = re.sub(r'[<>:"/\\|?*]', '_', path)

    # Ensure path doesn't end with a dot or space
    sanitized = sanitized.rstrip('. ')

    return sanitized

def get_aspect_ratio_name(aspect_ratio):
    """
    Find the closest matching aspect ratio name
    """
    for name, (target_ratio, tolerance) in COMMON_ASPECT_RATIOS.items():
        if abs(aspect_ratio - target_ratio) < tolerance:
            return name
    return 'other_aspect_ratio'

def process_images():
    # Create root window (will be hidden)
    root = tk.Tk()
    root.withdraw()

    # Ask for input folder
    input_folder = filedialog.askdirectory(title="Select Input Folder with Images")
    if not input_folder:
        print("No input folder selected. Exiting.")
```

```
        return

    # Get list of all files first
    all_files = []
    for root, dirs, files in os.walk(input_folder):
        for file in files:
            # Check for image extensions
            if file.lower().endswith(('.png', '.jpg', '.jpeg', '.gif', '.bmp', '.tiff',
'.webp')):
                all_files.append(os.path.join(root, file))

    # Tracking variables
    sorted_count = 0
    error_count = 0
    ratio_counts = {}

    # Use tqdm for progress bar
    for input_path in tqdm(all_files, desc="Sorting Images", unit="image"):
        try:
            # Open image and immediately close it after extracting info
            img = Image.open(input_path)
            width, height = img.size

            # Explicitly close the image file
            img.close()

            # Calculate aspect ratio
            aspect_ratio = width / height
            ratio_name = get_aspect_ratio_name(aspect_ratio)

            # Create subfolder if it doesn't exist
            output_subfolder = os.path.join(input_folder, sanitize_path(ratio_name))
            os.makedirs(output_subfolder, exist_ok=True)

            # Sanitize filename and create output path
            filename = os.path.basename(input_path)
            sanitized_filename = sanitize_path(filename)
            output_path = os.path.join(output_subfolder, sanitized_filename)
```

```
            # Move the file
            shutil.move(input_path, output_path)

            # Update tracking
            sorted_count += 1
            ratio_counts[ratio_name] = ratio_counts.get(ratio_name, 0) + 1

        except Exception as e:
            tqdm.write(f"Error processing {input_path}: {e}")
            error_count += 1

    # Final report
    print(f"\nTotal images sorted: {sorted_count}")
    print("Sorting breakdown:")
    for ratio, count in sorted(ratio_counts.items(), key=lambda x: x[1], reverse=True):
        print(f"  {ratio}: {count} images")
    print(f"Errors encountered: {error_count}")

if __name__ == "__main__":
    process_images()
```

Zuschneidung

Auf Seitenverhältnis zuschneiden

Bild auswählen, Fokuspunkt setzen, Ziel-Seitenverhältnis eingeben, speichern...

Nicht mehr komplett KI, ich hab da schon Anpassungen drin...

pip install pillow numpy scipy

```
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk
import numpy as np
```

```
from scipy.ndimage import gaussian_filter
import logging

# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def generate_seamless_fill(image, target_width, target_height, focus_x=0.5, focus_y=0.5):
    """
    Fits an image into a new aspect ratio using content-aware fill with focus point.

    Parameters:
        focus_x, focus_y: Float between 0 and 1 indicating the focus point position
    """
    logger.info("Starting seamless fill generation...")
    logger.info(f"Focus point: ({focus_x}, {focus_y})")

    # Convert to numpy array
    img_array = np.array(image)
    logger.info(f"Converting image to array of shape {img_array.shape}")

    # Create the new canvas
    if len(img_array.shape) == 3:
        result = np.zeros((target_height, target_width, img_array.shape[2]),
dtype=np.uint8)
        logger.info("Created RGB canvas")
    else:
        result = np.zeros((target_height, target_width), dtype=np.uint8)
        logger.info("Created grayscale canvas")

    # Calculate crop dimensions if needed
    src_height, src_width = img_array.shape[:2]
    height_diff = src_height - target_height
    width_diff = src_width - target_width

    logger.info(f"Height difference: {height_diff}, Width difference: {width_diff}")

    # Calculate source and destination regions using focus point
```

```
    if height_diff > 0:  # Need to crop height
        # Calculate crop position based on focus point
        focus_pixel_y = int(src_height * focus_y)
        half_target = target_height // 2
        src_y_start = max(0, min(src_height - target_height, focus_pixel_y - half_target))
        src_y_end = src_y_start + target_height
        dst_y_start = 0
        dst_y_end = target_height
        logger.info(f"Cropping height: {src_y_start}:{src_y_end} ->
{dst_y_start}:{dst_y_end}")
    else:  # Need to pad height
        src_y_start = 0
        src_y_end = src_height
        dst_y_start = abs(height_diff) // 2
        dst_y_end = dst_y_start + src_height
        logger.info(f"Padding height: {src_y_start}:{src_y_end} ->
{dst_y_start}:{dst_y_end}")

    if width_diff > 0:  # Need to crop width
        # Calculate crop position based on focus point
        focus_pixel_x = int(src_width * focus_x)
        half_target = target_width // 2
        src_x_start = max(0, min(src_width - target_width, focus_pixel_x - half_target))
        src_x_end = src_x_start + target_width
        dst_x_start = 0
        dst_x_end = target_width
        logger.info(f"Cropping width: {src_x_start}:{src_x_end} ->
{dst_x_start}:{dst_x_end}")
    else:  # Need to pad width
        src_x_start = 0
        src_x_end = src_width
        dst_x_start = abs(width_diff) // 2
        dst_x_end = dst_x_start + src_width
        logger.info(f"Padding width: {src_x_start}:{src_x_end} ->
{dst_x_start}:{dst_x_end}")

    # Copy the appropriate region of the source image to the destination
    result[dst_y_start:dst_y_end, dst_x_start:dst_x_end] =
```

```
img_array[src_y_start:src_y_end, src_x_start:src_x_end]

    # If we need to fill any borders
    if height_diff < 0 or width_diff < 0:
        logger.info("Generating content-aware fill for borders...")
        mask = np.zeros_like(result, dtype=bool)
        mask[result.sum(axis=2 if len(result.shape) == 3 else -1) == 0] = True

        if len(result.shape) == 3:
            for channel in range(result.shape[2]):
                logger.info(f"Processing channel {channel}")
                temp = result[:, :, channel].copy()
                edge_pixels = []
                for i in range(result.shape[0]):
                    for j in range(result.shape[1]):
                        if mask[i, j] and not np.all(mask[max(0, i-1):i+2, max(0, j-
1):j+2]):
                            edge_pixels.append((i, j))

                logger.info(f"Found {len(edge_pixels)} edge pixels to process")
                for i, j in edge_pixels:
                    neighborhood = temp[max(0, i-2):i+3, max(0, j-2):j+3]
                    valid_pixels = neighborhood[~mask[max(0, i-2):i+3, max(0, j-2):j+3]]
                    if len(valid_pixels) > 0:
                        temp[i, j] = np.mean(valid_pixels)

                logger.info("Applying gaussian blur for smooth transitions")
                temp_filled = gaussian_filter(temp, sigma=2)
                result[:, :, channel][mask[:, :]] = temp_filled[mask[:, :]]

    logger.info("Seamless fill generation complete")
    return Image.fromarray(result)

class AspectRatioChanger:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Image Aspect Ratio Changer")
        self.root.geometry("1920x1080")  # This would set the initial window size
```

```
        self.focus_point = (0.5, 0.5)  # Default center focus
        self.setup_ui()

    def setup_ui(self):
        # Main container
        main_frame = tk.Frame(self.root)
        main_frame.pack(padx=10, pady=10, expand=True, fill=tk.BOTH)

        # Left panel for controls
        controls_frame = tk.Frame(main_frame)
        controls_frame.pack(side=tk.LEFT, padx=10, fill=tk.Y)

        # File selection
        tk.Button(controls_frame, text="Select Image",
command=self.select_image).pack(pady=10)

        # Aspect ratio frame
        ratio_frame = tk.Frame(controls_frame)
        ratio_frame.pack(pady=10)

        tk.Label(ratio_frame, text="Width ratio:").pack(side=tk.LEFT)
        self.width_entry = tk.Entry(ratio_frame, width=10)
        self.width_entry.pack(side=tk.LEFT, padx=5)

        tk.Label(ratio_frame, text="Height ratio:").pack(side=tk.LEFT)
        self.height_entry = tk.Entry(ratio_frame, width=10)
        self.height_entry.pack(side=tk.LEFT, padx=5)

        # Common aspect ratios
        self.add_preset_buttons(controls_frame)

        # Process button
        tk.Button(controls_frame, text="Process Image",
command=self.process_image).pack(pady=10)

        # Focus point instructions
        tk.Label(controls_frame,
                 text="Click on the image to set center point for cropping!",
```

```
                wraplength=200,
                font=('TkDefaultFont', 10, 'bold'),  # Makes the text bold
                fg='red'  # Makes the text red
        ).pack(pady=10)

        # Add Reset Focus Point button
        tk.Button(controls_frame,
                  text="Reset Focus Point",
                  command=self.reset_focus_point
        ).pack(pady=5)

        # Right panel for image preview
        self.preview_frame = tk.Frame(main_frame, width=800, height=600)
        self.preview_frame.pack(side=tk.LEFT, padx=10, expand=True, fill=tk.BOTH)

        self.preview_canvas = tk.Canvas(self.preview_frame, bg='gray')
        self.preview_canvas.pack(expand=True, fill=tk.BOTH)
        self.preview_canvas.bind("<Button-1>", self.set_focus_point)

    def reset_focus_point(self):
        self.focus_point = (0.5, 0.5)  # Reset to center

        if hasattr(self, 'preview_image'):  # Only redraw if we have an image
            # Get canvas dimensions
            canvas_width = self.preview_canvas.winfo_width()
            canvas_height = self.preview_canvas.winfo_height()

            # Calculate center position
            center_x = canvas_width // 2
            center_y = canvas_height // 2

            # Redraw focus point at center
            self.draw_focus_point(center_x, center_y)

    def add_preset_buttons(self, parent):
        presets_frame = tk.Frame(parent)
        presets_frame.pack(pady=10)
```

```
        presets_row1 = {
            "16:9 (PC)": (16, 9),
            "16:10": (16, 10),
            "4:3 (Tablet)": (4, 3),
        }

        presets_row2 = {
            "1:1 (Social Media)": (1, 1),
            "9:16 (Smartphone)": (9, 16),
            "1.44:1 (iPad 2022)": (144, 100)
        }

        tk.Label(presets_frame, text="Common Ratios:").pack()

        # First row of buttons
        buttons_frame1 = tk.Frame(presets_frame)
        buttons_frame1.pack(pady=(5, 2))

        for name, (w, h) in presets_row1.items():
            tk.Button(buttons_frame1, text=name,
                      command=lambda w=w, h=h: self.set_aspect_ratio(w,
h)).pack(side=tk.LEFT, padx=2)

        # Second row of buttons
        buttons_frame2 = tk.Frame(presets_frame)
        buttons_frame2.pack(pady=(2, 5))

        for name, (w, h) in presets_row2.items():
            tk.Button(buttons_frame2, text=name,
                      command=lambda w=w, h=h: self.set_aspect_ratio(w,
h)).pack(side=tk.LEFT, padx=2)



    def set_aspect_ratio(self, width, height):
        self.width_entry.delete(0, tk.END)
        self.height_entry.delete(0, tk.END)
        self.width_entry.insert(0, str(width))
        self.height_entry.insert(0, str(height))
```

```
def select_image(self):
    self.image_path = filedialog.askopenfilename(
        filetypes=[("Image files", "*.png *.jpg *.jpeg *.gif *.bmp")]
    )
    if self.image_path:
        logger.info(f"Selected image: {self.image_path}")
        self.load_preview_image()

def load_preview_image(self):
    # Load and resize image for preview
    image = Image.open(self.image_path)

    # Calculate resize dimensions to fit canvas
    canvas_width = self.preview_frame.winfo_width()
    canvas_height = self.preview_frame.winfo_height()

    # Resize image maintaining aspect ratio
    image.thumbnail((canvas_width, canvas_height), Image.Resampling.LANCZOS)

    # Store original image dimensions for focus point calculation
    self.original_size = Image.open(self.image_path).size
    self.preview_size = image.size

    # Create PhotoImage and store reference
    self.preview_image = ImageTk.PhotoImage(image)

    # Clear canvas and draw new image
    self.preview_canvas.delete("all")
    self.preview_canvas.create_image(
        canvas_width//2, canvas_height//2,
        image=self.preview_image,
        anchor=tk.CENTER,
        tags="preview"
    )

    # Draw initial focus point
    self.draw_focus_point(canvas_width//2, canvas_height//2)
```

```
def set_focus_point(self, event):
    if not hasattr(self, 'preview_image'):
        return

    # Get canvas dimensions
    canvas_width = self.preview_canvas.winfo_width()
    canvas_height = self.preview_canvas.winfo_height()

    # Calculate image position and size on canvas
    img_width, img_height = self.preview_size
    x_offset = (canvas_width - img_width) // 2
    y_offset = (canvas_height - img_height) // 2

    # Convert click position to image coordinates
    x = event.x - x_offset
    y = event.y - y_offset

    # Check if click is within image bounds
    if 0 <= x < img_width and 0 <= y < img_height:
        # Calculate relative position (0-1)
        self.focus_point = (x / img_width, y / img_height)
        logger.info(f"Focus point set to: {self.focus_point}")

        # Update focus point marker
        self.draw_focus_point(event.x, event.y)

def draw_focus_point(self, x, y):
    # Clear previous focus point
    self.preview_canvas.delete("focus_point")

    # Draw new focus point
    radius = 5
    self.preview_canvas.create_oval(
        x - radius, y - radius,
        x + radius, y + radius,
        fill='red',
        tags="focus_point"
```

```
        )

        # Draw crosshair
        size = 10
        self.preview_canvas.create_line(
            x - size, y, x + size, y,
            fill='red',
            tags="focus_point"
        )
        self.preview_canvas.create_line(
            x, y - size, x, y + size,
            fill='red',
            tags="focus_point"
        )

    def process_image(self):
        if not hasattr(self, 'image_path'):
            messagebox.showerror("Error", "Please select an image first")
            return

        try:
            target_width = int(self.width_entry.get())
            target_height = int(self.height_entry.get())

            if target_width <= 0 or target_height <= 0:
                raise ValueError("Dimensions must be positive")

        except ValueError as e:
            messagebox.showerror("Error", "Please enter valid positive numbers for width
and height")
            return

        # Load and process image
        try:
            image = Image.open(self.image_path)
            # Calculate new dimensions while maintaining aspect ratio
            orig_width, orig_height = image.size
            scale = min(orig_width / target_width, orig_height / target_height)
```

```
            new_width = int(target_width * scale)
            new_height = int(target_height * scale)

            # Process the image with focus point
            processed_image = generate_seamless_fill(
                image, new_width, new_height,
                focus_x=self.focus_point[0],
                focus_y=self.focus_point[1]
            )

            # Save the processed image
            save_path = filedialog.asksaveasfilename(
                defaultextension=".png",
                filetypes=[("PNG files", "*.png"), ("All files", "*.*")]
            )
            if save_path:
                processed_image.save(save_path)
                messagebox.showinfo("Success", "Image processed and saved successfully!")

        except Exception as e:
            messagebox.showerror("Error", f"An error occurred: {str(e)}")
            logger.error(f"Error processing image: {e}", exc_info=True)

    def run(self):
        self.root.mainloop()

if __name__ == "__main__":
    app = AspectRatioChanger()
    app.run()
```

Upscaling

Lanczos auf 4K (min eine Seitenlänge)

Script skaliert unter Beibehaltung des Seitenverhältnisses die Bilder soweit hoch dass mindestens eine Seite die 4K-Auflösung matchen kann. Originale werden in Unterordner verschoben, die hochskalierten landen in dem Ordner den ihr angebt.

Lanczos ist nicht der beste Algorithmus dafür, aber brauchbar. Es kommt teilweise grobkörniges dabei raus, aber es ist ein passender Schnittpunkt aus Geschwindigkeit und Qualität.

Requirements:

Pillow -> die Bildlibrary, ohne die geht nix mit Bildern
tqdm -> für eine Fortschrittsanzeige

```
from PIL import Image
import os
from pathlib import Path
import sys
import shutil
from tqdm import tqdm

def upscale_to_4k(image_path, output_path):
    """
    Upscale an image to 4K resolution (3840×2160) while maintaining aspect ratio.
    At least one dimension will match 4K resolution.
    """
    try:
        # Open the image
        with Image.open(image_path) as img:
            # Get original dimensions
            width, height = img.size

            # Calculate aspect ratio
            aspect_ratio = width / height

            # Calculate new dimensions
            if aspect_ratio > 16/9:  # Wider than 4K aspect ratio
                new_height = 2160
                new_width = int(new_height * aspect_ratio)
            else:  # Taller than 4K aspect ratio
                new_width = 3840
```

```
                new_height = int(new_width / aspect_ratio)

            # Upscale image using Lanczos resampling
            upscaled_img = img.resize((new_width, new_height), Image.Resampling.LANCZOS)

            # Create output directory if it doesn't exist
            os.makedirs(os.path.dirname(output_path), exist_ok=True)

            # Save the upscaled image with original format
            upscaled_img.save(output_path, quality=95)
            return True
    except Exception as e:
        print(f"Error processing {image_path}: {str(e)}")
        return False

def move_to_originals(file_path):
    """
    Move the original file to an 'originals' subfolder in the source directory.
    """
    source_dir = file_path.parent
    originals_dir = source_dir / "originals"
    os.makedirs(originals_dir, exist_ok=True)

    # Create destination path
    dest_path = originals_dir / file_path.name

    # Handle file name conflicts
    counter = 1
    while dest_path.exists():
        stem = file_path.stem
        suffix = file_path.suffix
        dest_path = originals_dir / f"{stem}_{counter}{suffix}"
        counter += 1

    # Move the file
    shutil.move(str(file_path), str(dest_path))

def main():
```

```
# Get input and output directories from user
input_dir = input("Enter the path to the folder containing images: ").strip()
output_dir = input("Enter the path where upscaled images should be saved: ").strip()

# Validate input directory
if not os.path.exists(input_dir):
    print("Error: Input directory does not exist!")
    sys.exit(1)

# Create output directory if it doesn't exist
os.makedirs(output_dir, exist_ok=True)

# Supported image formats
supported_formats = {'.jpg', '.jpeg', '.png', '.bmp', '.tiff'}

# Get list of all image files first
image_files = [f for f in Path(input_dir).rglob('*')
               if f.suffix.lower() in supported_formats]

if not image_files:
    print("No supported image files found in the input directory!")
    sys.exit(1)

processed = 0
failed = 0

print("\nStarting image upscaling process...")

# Process files with progress bar
with tqdm(total=len(image_files), desc="Upscaling images",
          unit="image", ncols=80) as pbar:
    for file_path in image_files:
        # Create corresponding output path
        relative_path = file_path.relative_to(input_dir)
        output_path = Path(output_dir) / relative_path

        if upscale_to_4k(str(file_path), str(output_path)):
            # Move original file to originals folder
```

```
                move_to_originals(file_path)
                processed += 1
            else:
                failed += 1

            pbar.update(1)

    # Print summary
    print(f"\nUpscaling complete!")
    print(f"Successfully processed: {processed} images")
    print(f"Failed: {failed} images")
    print(f"Original images have been moved to the 'originals' subfolder")

if __name__ == "__main__":
    main()
```

KI-Tools

Foto-Triage

Ihr müsst für KI teils größere Datensätze bauen. Das geht mit einem einfachen Triage-Tool besser und deutlich schneller.
Drei Ordner auswählen, auswählen was mit good oder bad passieren soll und ab die Post.
Kommt mit größeren Bildern klar, ist aber etwas hakelig wenns ans Fenster verschieben geht, weil Python das wohl neu zeichnet beim Verschieben...

Tastatur: 1 für gute Bilder, 0 für schlechte Bilder und # fürs Überspringen.

Manchmal gibts Fehlermeldungen, je nach Bild, dafür gibts dann den Skip-Button.
Manchmal wirds langsam, je nachdem wie schnell die Festplatten sind und wie viel davon geladen werden muss.
Ich hantiere mit Bildern von fast 16k x 25k Pixeln. Es kommt also schon sehr auf eure Hardware an...

Die Decompression Bomb Protection ist eine Schutzmaßnahme die bei meinen großen Bildern eigentlich ständig anschlug. Ich hab das Gefühl dass es schon bei bisschen über 4K anschlägt.

```
import os
import shutil
import tkinter as tk
from tkinter import filedialog, ttk, messagebox
from PIL import Image, ImageTk
Image.MAX_IMAGE_PIXELS = None  # Remove decompression bomb protection

class ImageReviewTool:
    def __init__(self, master):
        self.master = master
        master.title("Image Triage Tool")
        master.geometry("800x600")  # Set initial window size
        master.minsize(400, 300)  # Set minimum window size

        # Create the main frame
        self.frame = tk.Frame(master)
        self.frame.pack(fill=tk.BOTH, expand=True)

        # Create the image display area
        self.image_label = tk.Label(self.frame)
        self.image_label.pack(fill=tk.BOTH, expand=True)

        # Create frame for file handling options
        options_frame = tk.Frame(self.frame)
        options_frame.pack(side=tk.BOTTOM, fill=tk.X)

        # Create dropdown for "Good" images handling
        good_options_frame = tk.Frame(options_frame)
        good_options_frame.pack(side=tk.LEFT, padx=10, pady=5)

        tk.Label(good_options_frame, text="Good images:").pack(side=tk.LEFT)
        self.good_handling = ttk.Combobox(good_options_frame,
                                          values=["Copy", "Move"],
                                          state="readonly",
                                          width=10)
        self.good_handling.set("Copy")
        self.good_handling.pack(side=tk.LEFT, padx=5)

        # Create dropdown for "Bad" images handling
```

```
        bad_options_frame = tk.Frame(options_frame)
        bad_options_frame.pack(side=tk.LEFT, padx=10, pady=5)

        tk.Label(bad_options_frame, text="Bad images:").pack(side=tk.LEFT)
        self.bad_handling = ttk.Combobox(bad_options_frame,
                                         values=["Copy", "Move"],
                                         state="readonly",
                                         width=10)
        self.bad_handling.set("Move")
        self.bad_handling.pack(side=tk.LEFT, padx=5)

        # Create the "Good", "Bad", and "Skip" buttons
        button_frame = tk.Frame(self.frame)
        button_frame.pack(side=tk.BOTTOM, fill=tk.X)

        self.good_button = tk.Button(button_frame, text="Good (1)",
command=self.handle_good)
        self.good_button.pack(side=tk.LEFT, padx=10, pady=10)

        self.bad_button = tk.Button(button_frame, text="Bad (0)", command=self.handle_bad)
        self.bad_button.pack(side=tk.LEFT, padx=10, pady=10)

        self.skip_button = tk.Button(button_frame, text="Skip (#)",
command=self.handle_skip)
        self.skip_button.pack(side=tk.LEFT, padx=10, pady=10)

        # Create the directory selection buttons
        dir_button_frame = tk.Frame(self.frame)
        dir_button_frame.pack(side=tk.BOTTOM, fill=tk.X)

        self.select_directory_button = tk.Button(dir_button_frame, text="Select
Directory", command=self.select_directory)
        self.select_directory_button.pack(side=tk.LEFT, padx=10, pady=10)

        self.select_good_directory_button = tk.Button(dir_button_frame, text="Select Good
Directory", command=self.select_good_directory)
        self.select_good_directory_button.pack(side=tk.LEFT, padx=10, pady=10)

        self.select_bad_directory_button = tk.Button(dir_button_frame, text="Select Bad
```

```
Directory", command=self.select_bad_directory)
        self.select_bad_directory_button.pack(side=tk.LEFT, padx=10, pady=10)

        # Add current file label
        self.file_label = tk.Label(self.frame, text="")
        self.file_label.pack(side=tk.BOTTOM, pady=5)

        self.current_directory = None
        self.good_folder = None
        self.bad_folder = None
        self.current_image_index = 0
        self.image_files = []
        self.reviewed_images = []
        self.log_file = "image_review_log.txt"
        self.current_image = None

        # Bind the window resize event to the resize_image method
        self.master.bind("<Configure>", self.resize_image)

        # Bind keyboard shortcuts
        self.master.bind("1", self.handle_good)
        self.master.bind("0", self.handle_bad)
        self.master.bind("#", self.handle_skip)

    def select_directory(self):
        self.current_directory = filedialog.askdirectory()
        if self.current_directory:
            self.image_files = [f for f in os.listdir(self.current_directory) if
f.lower().endswith((".jpg", ".jpeg", ".png", ".gif"))]
            self.current_image_index = 0
            self.reviewed_images = self.load_reviewed_images()
            self.display_image()

    def select_good_directory(self):
        self.good_folder = filedialog.askdirectory()

    def select_bad_directory(self):
        self.bad_folder = filedialog.askdirectory()
```

```
    def display_image(self):
        if self.current_directory and self.image_files:
            if self.current_image_index >= len(self.image_files):
                self.image_label.configure(image='')
                self.file_label.config(text="")
                messagebox.showinfo("Complete", "All images have been reviewed!")
                return

            try:
                image_path = os.path.join(self.current_directory,
self.image_files[self.current_image_index])
                self.current_image = Image.open(image_path)

                # Update file info label
                img_size = os.path.getsize(image_path) / (1024 * 1024)  # Convert to MB
                self.file_label.config(
                    text=f"File: {self.image_files[self.current_image_index]} | "
                         f"Size: {img_size:.1f}MB | "
                         f"Dimensions:
{self.current_image.width}x{self.current_image.height}"
                )

                self.resize_image(None)
            except Exception as e:
                messagebox.showerror("Error", f"Failed to load image: {str(e)}")
                self.current_image_index += 1
                self.display_image()

    def resize_image(self, event):
        if self.current_image:
            try:
                # Get the current window size
                window_width = self.master.winfo_width()
                window_height = self.master.winfo_height() - 150  # Adjust for buttons and
options

                # Scale the image to fit the window while maintaining aspect ratio
                image_ratio = self.current_image.width / self.current_image.height
                if window_width / window_height > image_ratio:
```

```
                    new_height = window_height
                    new_width = int(new_height * image_ratio)
                else:
                    new_width = window_width
                    new_height = int(new_width / image_ratio)

                # Use thumbnail for memory-efficient resizing of large images
                img_copy = self.current_image.copy()
                img_copy.thumbnail((new_width, new_height), Image.Resampling.LANCZOS)
                self.photo = ImageTk.PhotoImage(img_copy)
                self.image_label.configure(image=self.photo)
            except Exception as e:
                messagebox.showerror("Error", f"Failed to resize image: {str(e)}")

    def handle_good(self, event=None):
        if self.current_directory and self.good_folder and self.image_files:
            src_path = os.path.join(self.current_directory,
self.image_files[self.current_image_index])
            dst_path = os.path.join(self.good_folder,
self.image_files[self.current_image_index])

            try:
                if self.good_handling.get() == "Copy":
                    shutil.copy2(src_path, dst_path)
                else:  # Move
                    shutil.move(src_path, dst_path)

                self.reviewed_images.append(self.image_files[self.current_image_index])
                self.save_reviewed_images()
                self.current_image_index += 1
                self.display_image()
            except Exception as e:
                messagebox.showerror("Error", f"Failed to process image: {str(e)}")

    def handle_bad(self, event=None):
        if self.current_directory and self.bad_folder and self.image_files:
            src_path = os.path.join(self.current_directory,
self.image_files[self.current_image_index])
            dst_path = os.path.join(self.bad_folder,
```

```
self.image_files[self.current_image_index])

            try:
                if self.bad_handling.get() == "Copy":
                    shutil.copy2(src_path, dst_path)
                else:  # Move
                    shutil.move(src_path, dst_path)

                self.reviewed_images.append(self.image_files[self.current_image_index])
                self.save_reviewed_images()
                self.current_image_index += 1
                self.display_image()
            except Exception as e:
                messagebox.showerror("Error", f"Failed to process image: {str(e)}")

    def handle_skip(self, event=None):
        if self.current_directory and self.image_files:
            self.current_image_index += 1
            self.display_image()

    def load_reviewed_images(self):
        reviewed_images = []
        if os.path.exists(self.log_file):
            with open(self.log_file, "r") as f:
                reviewed_images = [line.strip() for line in f.readlines()]
        return reviewed_images

    def save_reviewed_images(self):
        with open(self.log_file, "w") as f:
            for image in self.reviewed_images:
                f.write(image + "\n")

if __name__ == "__main__":
    root = tk.Tk()
    app = ImageReviewTool(root)
    root.mainloop()
```

"bad" and "good" Sortierung nach Beispielen

Funktionsweise

Das Script benötigt drei Ordner:

- schlechte Bilder
- gute Bilder
- Ordner mit Fotos die sortiert werden müssen

je nachdem wie viele Beispiele in beiden Ordnern (schlecht und gut) sind desto besser wird sortiert...
Im Ordner mit den zu sortierenden Bildern werden zwei Ordner erstellt in denen die jeweiligen Fotos verschoben werden.

Anlernphase

Es ist völlig egal nach welchen Kriterien ihr sortieren wollt, worum ihr nicht rum kommt ist 100-200 Fotos händisch zu sortieren die schlecht und die gut sind.

Danach lasst ihr das Script einmal über euren Bildersatz laufen.
Aus dem "bad" Ordner sortiert ihr dann die guten Bilder, die da noch fälschlicherweise sind in den Trainingssatz für die guten Bilder rein.
Das was übrig bleibt sind dann die echt schlechten Bilder, die ihr am besten dann auch in den schlechten Ordner verschiebt.

Dann nehmt ihr den Ordner in dem die laut Script guten Fotos reinsortiert worden sind und lasst das Script mit dem neuen Datensatz noch mal drüber laufen.

Das ganze wiederholt ihr dann bis das Ergebnis für euch passt. Die Ordner mit den Datensätzen behaltet ihr dann...
Je nachdem welchen Datensatz ihr vergrößert verbessert ihr die Erkennung der schlechten oder der guten Bilder.

Das ist basic Bilder-KI Training...

Script

> DIe Ordner und Variablen im Script heißen noch "digital art" und "photos" da meine Aufgabe es war rein gezeichnete Bilder von Fotografien zu trennen.

> Nun auch mit Menü zur Auswahl und Vorab-Training eines Modells und allem Firlefanz...

Script ist PYTHON

pip install pillow torch torchvision

```
import torch
import torchvision.transforms as transforms
import torchvision.models as models
from torchvision.models import ResNet50_Weights  # Add this import
from PIL import Image
import os
import shutil
from torch import nn
import numpy as np
from pathlib import Path
import tkinter as tk
from tkinter import filedialog
from torch.utils.data import Dataset, DataLoader
from torchvision.datasets import ImageFolder
import warnings
from PIL import Image, ImageFile

# Increase PIL image size limit and allow truncated images
Image.MAX_IMAGE_PIXELS = None  # Remove size limit
ImageFile.LOAD_TRUNCATED_IMAGES = True
warnings.filterwarnings('ignore', category=Image.DecompressionBombWarning)

class CustomImageDataset(Dataset):
    def __init__(self, photos_path, digital_path, transform=None):
        self.transform = transform
        self.images = []
        self.labels = []

        # Load photos (label 0)
        for img_path in Path(photos_path).glob('*'):
            if img_path.suffix.lower() in {'.jpg', '.jpeg', '.png', '.bmp'}:
                self.images.append(str(img_path))
                self.labels.append(0)

        # Load digital art (label 1)
```

```
        for img_path in Path(digital_path).glob('*'):
            if img_path.suffix.lower() in {'.jpg', '.jpeg', '.png', '.bmp'}:
                self.images.append(str(img_path))
                self.labels.append(1)

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        image_path = self.images[idx]
        try:
            with Image.open(image_path) as img:
                img = img.convert('RGB')

                if max(img.size) > 4096:
                    ratio = 4096 / max(img.size)
                    new_size = tuple(int(dim * ratio) for dim in img.size)
                    img = img.resize(new_size, Image.Resampling.LANCZOS)

                if self.transform:
                    image = self.transform(img)
                return image, self.labels[idx]
        except Exception as e:
            print(f"Error loading image {image_path}: {e}")
            return torch.zeros((3, 224, 224)), self.labels[idx]

class ImageSorter:
    def __init__(self):
        # Use the new weights parameter instead of pretrained
        weights = ResNet50_Weights.DEFAULT
        self.model = models.resnet50(weights=weights)
        num_features = self.model.fc.in_features
        self.model.fc = nn.Linear(num_features, 2)

        # Update transform to use the weights preprocessing
        self.transform = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.ToTensor(),
```

```
            weights.transforms()  # Use the preprocessing from the weights
        ])

        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model = self.model.to(self.device)
        self.criterion = nn.CrossEntropyLoss()
        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=0.001)

    def save_model(self, path):
        """Save the trained model to a file"""
        save_dict = {
            'model_state_dict': self.model.state_dict(),
            'optimizer_state_dict': self.optimizer.state_dict(),
        }
        torch.save(save_dict, path)
        print(f"Model saved to {path}")

    def load_model(self, path):
        """Load a trained model from a file"""
        if not os.path.exists(path):
            raise FileNotFoundError(f"No model file found at {path}")

        checkpoint = torch.load(path, map_location=self.device)
        self.model.load_state_dict(checkpoint['model_state_dict'])
        self.optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
        print(f"Model loaded from {path}")

    def safe_load_image(self, image_path):
        try:
            with Image.open(image_path) as img:
                img = img.convert('RGB')
                if max(img.size) > 4096:
                    ratio = 4096 / max(img.size)
                    new_size = tuple(int(dim * ratio) for dim in img.size)
                    img = img.resize(new_size, Image.Resampling.LANCZOS)
                return img
        except Exception as e:
            print(f"Error loading image {image_path}: {e}")
```

```
            return None

    def train_model(self, photos_path, digital_path, epochs=10):
        dataset = CustomImageDataset(
            photos_path=photos_path,
            digital_path=digital_path,
            transform=self.transform
        )

        dataloader = DataLoader(dataset, batch_size=16, shuffle=True)

        print("\nStarting training...")
        print(f"Training with {len(dataset)} images")
        print(f"Using device: {self.device}")

        self.model.train()
        for epoch in range(epochs):
            running_loss = 0.0
            correct = 0
            total = 0

            for i, (images, labels) in enumerate(dataloader):
                images = images.to(self.device)
                labels = labels.to(self.device)

                self.optimizer.zero_grad()
                outputs = self.model(images)
                loss = self.criterion(outputs, labels)
                loss.backward()
                self.optimizer.step()

                running_loss += loss.item()
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

            epoch_loss = running_loss / len(dataloader)
            accuracy = 100 * correct / total
```

```
            print(f'Epoch {epoch + 1}/{epochs} - Loss: {epoch_loss:.4f} - Accuracy:
{accuracy:.2f}%')

    def predict_image(self, image_path):
        try:
            img = self.safe_load_image(image_path)
            if img is None:
                return None, None

            image_tensor = self.transform(img).unsqueeze(0).to(self.device)

            self.model.eval()
            with torch.no_grad():
                outputs = self.model(image_tensor)
                probabilities = torch.softmax(outputs, dim=1)
                confidence, prediction = torch.max(probabilities, 1)

            return prediction.item(), confidence.item()

        except Exception as e:
            print(f"Error processing {image_path}: {str(e)}")
            return None, None

    def sort_images(self, input_directory):
        input_path = Path(input_directory).resolve()
        digital_art_dir = input_path / 'digital_art'
        photos_dir = input_path / 'photos'

        digital_art_dir.mkdir(exist_ok=True, parents=True)
        photos_dir.mkdir(exist_ok=True, parents=True)

        print(f"\nCreated directories:")
        print(f"Digital Art: {digital_art_dir}")
        print(f"Photos: {photos_dir}\n")

        supported_formats = {'.jpg', '.jpeg', '.png', '.bmp'}
        files_processed = 0
        failed_files = 0
```

```
        total_files = len([f for f in input_path.iterdir()
                           if f.is_file() and f.suffix.lower() in supported_formats])

        for file_path in input_path.iterdir():
            if file_path.is_file() and file_path.suffix.lower() in supported_formats:
                if 'digital_art' in str(file_path) or 'photos' in str(file_path):
                    continue

                try:
                    files_processed += 1
                    print(f"Processing image {files_processed}/{total_files}:
{file_path.name}")

                    prediction, confidence = self.predict_image(str(file_path))
                    if prediction is None:
                        failed_files += 1
                        continue

                    if prediction == 1:
                        destination = digital_art_dir / file_path.name
                        print(f"→ Moving to digital_art folder (confidence:
{confidence:.2%})")
                    else:
                        destination = photos_dir / file_path.name
                        print(f"→ Moving to photos folder (confidence: {confidence:.2%})")

                    shutil.move(str(file_path), str(destination))

                except Exception as e:
                    failed_files += 1
                    print(f"Error processing {file_path.name}: {str(e)}")

        print(f"\nProcessing complete!")
        print(f"Total images processed: {files_processed}")
        print(f"Failed to process: {failed_files}")

def get_folder_path(title):
    root = tk.Tk()
```

```
    root.withdraw()
    folder_path = filedialog.askdirectory(title=title)
    return folder_path if folder_path else None

def get_file_path(title, file_types):
    root = tk.Tk()
    root.withdraw()
    file_path = filedialog.askopenfilename(title=title, filetypes=file_types)
    return file_path if file_path else None

def main():
    print("Digital Art Sorter")
    print("==================")

    sorter = ImageSorter()

    while True:
        print("\nOptions:")
        print("1. Train and save new model")
        print("2. Load existing model")
        print("3. Sort images using current model")
        print("4. Exit")

        choice = input("\nEnter your choice (1-4): ")

        if choice == '1':
            print("\nSelect folder containing PHOTO examples")
            photos_training = get_folder_path("Select folder with photo examples")
            if not photos_training:
                continue

            print("\nSelect folder containing DIGITAL ART examples")
            digital_training = get_folder_path("Select folder with digital art examples")
            if not digital_training:
                continue

            sorter.train_model(photos_training, digital_training)
```

```
            print("\nSelect where to save the trained model")
            save_path = filedialog.asksaveasfilename(
                defaultextension=".pth",
                filetypes=[("PyTorch Model", "*.pth")]
            )
            if save_path:
                sorter.save_model(save_path)

        elif choice == '2':
            print("\nSelect model file to load")
            model_path = get_file_path("Select model file", [("PyTorch Model", "*.pth")])
            if model_path:
                try:
                    sorter.load_model(model_path)
                except Exception as e:
                    print(f"Error loading model: {str(e)}")

        elif choice == '3':
            print("\nSelect folder containing images to sort")
            source_folder = get_folder_path("Select folder with images to sort")
            if source_folder:
                sorter.sort_images(source_folder)

        elif choice == '4':
            print("\nExiting...")
            break

        else:
            print("\nInvalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

Dedup

macht man am besten mit Czkawka oder DupeGuru

Basic Dedup

ChatGPT 3.5

(falls Copy statt Move im Script vorab gemacht wurde)

> Das Ding tut noch nicht das was ich will, womöglich komme ich selbst durcheinander mit Source und Destination. Ich setz also unten noch mal anders an.

```
param (
  # Mag etwas durcheinander und verkehrt sein. KI halt... Jedenfalls ist Destination der Ordner der über bleiben soll. und source der Ordner in dem die Duplikate liegen
    # Ordner in dem gelöscht wird.
    [string]$SourceFolder = "XXXXXXXXXXXXXXXXXXXXXXXX",
    # Ordner gegen den Verglichen wird
    [string]$DestinationFolder = "XXXXXXXXXXXXXXXXXXX",

)

# Get all image files in the destination folder
$destinationImages = Get-ChildItem -Path $DestinationFolder -Filter "*.jpg" -File

# Initialize counters for deleted images in each folder
$deletedInSourceFolderCount = 0
$deletedInDestinationFolderCount = 0

# Process each image file in the destination folder
foreach ($destinationImage in $destinationImages) {
    Write-Host "Processing $($destinationImage.FullName)"

    # Construct the source file path based on the destination file name
    $sourceFilePath = Join-Path -Path $SourceFolder -ChildPath $destinationImage.Name

    # Check if the corresponding file exists in the source folder
    if (Test-Path $sourceFilePath) {
        Write-Host "Deleting $($sourceFilePath)"
```

```
        # Remove the item (move to Recycle Bin)
        Remove-Item -Path $sourceFilePath -Force -Confirm:$false

        # Increment the counter for deleted images in the source folder
        $deletedInSourceFolderCount++
    }

    # Increment the counter for deleted images in the destination folder
    $deletedInDestinationFolderCount++
}

# Output the deletion statistics
Write-Host "Duplicate removal complete!"
Write-Host "Deleted $deletedInSourceFolderCount images in the source folder:
$SourceFolder"
```

Prüfscripts

Prüfen ob keine Bilder >4K existieren

ChatGPT 3.5

```
$SourceFolder = "C:\Path\to\SourceFolder"
$MinWidth = 3840
$MinHeight = 2160

# Get all image files in the source folder and its subfolders
$images = Get-ChildItem -Path $SourceFolder -Filter "*.jpg" -File -Recurse

$totalImages = $images.Count
$processedImages = 0

# Process each image file
foreach ($image in $images) {
    # Use .NET classes to read image dimensions
    $imageStream = New-Object System.IO.FileStream($image.FullName,
[System.IO.FileMode]::Open)
```

```
    $imageBitmap = New-Object System.Drawing.Bitmap($imageStream)
    $width = $imageBitmap.Width
    $height = $imageBitmap.Height
    $imageStream.Close()

    # Check if the image is larger than the specified dimensions
    if ($width -ge $MinWidth -and $height -ge $MinHeight) {
        Write-Host "Found large image: $($image.FullName)"
    }

    $processedImages++
    $progress = [math]::Round(($processedImages / $totalImages) * 100, 2)
    Write-Progress -Activity "Checking image sizes" -Status "Progress: $progress%" -
PercentComplete $progress
}

Write-Progress -Activity "Checking image sizes" -Status "Progress: 100%" -PercentComplete
100
Write-Host "Image size check complete!"
```

Alle Bilder > 4K löschen

```
$SourceFolder = "C:\Path\to\SourceFolder"
$MinWidth = 3840
$MinHeight = 2160

# Get all image files in the source folder and its subfolders
$images = Get-ChildItem -Path $SourceFolder -Filter "*.jpg" -File -Recurse

$totalImages = $images.Count
$processedImages = 0

# Process each image file
foreach ($image in $images) {
    # Use .NET classes to read image dimensions
    $imageStream = New-Object System.IO.FileStream($image.FullName,
[System.IO.FileMode]::Open)
```

```
    $imageBitmap = New-Object System.Drawing.Bitmap($imageStream)
    $width = $imageBitmap.Width
    $height = $imageBitmap.Height
    $imageStream.Close()

    # Check if the image is larger than the specified dimensions
    if ($width -ge $MinWidth -and $height -ge $MinHeight) {
        Write-Host "Deleting large image: $($image.FullName)"
        Remove-Item -Path $image.FullName -Force
    }

    $processedImages++
    $progress = [math]::Round(($processedImages / $totalImages) * 100, 2)
    Write-Progress -Activity "Checking and deleting large images" -Status "Progress:
$progress%" -PercentComplete $progress
}

Write-Progress -Activity "Checking and deleting large images" -Status "Progress: 100%" -
PercentComplete 100
Write-Host "Image size check and deletion complete!"
```

Version #50
Erstellt: 2023-07-14 17:40:51 UTC von Konstantin
Zuletzt aktualisiert: 2025-06-05 14:44:46 UTC von Konstantin