Bildfilter

March 9, 2024

1 Problem zu lösen

Das Ziel ist die Entwicklung eines Python-Programms, das verschiedene Bildfilter auf Bilder anwenden kann: Grayscale (Graustufen), Sepia, Reflect (Spiegelung) und Blur (Weichzeichnung). Diese Filter sollen das Originalbild auf unterschiedliche Weise visuell verändern, um verschiedene Effekte zu erzielen.

2 Demo ausführung

Als erstes konvertiere ich eine JPG in ein BMP mit 24 Bit für True Color. Dazu muss die "rows direction" auf "top-bottom" eingestellt werden Dazu nutze ich: https://online-converting.com/image/convert2bmp/

Als nächstes führe ich den notdenwigen Befehl aus, um den Filter anzuwenden und eine neue Datei zu erzeugen

```
./filter -g ./images/hfu_tuttlingen.bmp output1-grayscale.bmp
```

Wenn ich jetzt die Datei output1-grayscale.bmp öffne, kommt folgendes Bild raus:

Genauso, kann ich auch die anderen Filter anwenden, um die folgenden Bilder zu erhalten wie `sepia`:

`reflect`:

oder `blur`:

3 Hintergrundwissen

Eine Datei ist eine Sequenz von Bits. Wir benutzen für diese Aufgabe den 24-Bit BMP Format, um Bilder darzustellen. Die 24-Bit kommen daher, dass man um jede Farbe darzustellen genau 8 Bit braucht, für Rot, Grün und Blau. Mit der Kombination dieser drei Farben, können wir jede andere Farbe darstellen. Also enhällt unsere BMP-Datei den Wert eines Pixels in 24 Bit. In den nächsten 24 Bit wird der nächste Pixel dargestellt. Wir müssen jedoch beachten, dass eine BMP-Datei auch Meta-Daten enthällt. Diese enthalten Informationen, wie die Höhe und Breite des Bildes. Diese Meta-Daten sind am Anfang einer BMP-Datei, im sogenannten Header gespeichert. Es gibt zwei Header am Anfang jeder BMP-Datei.

Der erste Header `BITMAPFILEHEADER` ist 14 Byte lang (also 14 * 8 Bit). Der zweite Header `BITMAPINFOHEADER` ist genau 40 Byte lang. Sofort dannach kommt die BITMAP, also das Array von Bytes, dass die Pixelwerte darstellt. Man muss wissen, dass BMP-Datein, die Farben rückwärts abspeichern. Also nicht als RGB (für rot, grün und blau), sonder als BGR (für blau, grün, rot). Manche BMP-Datein, enthalten die ganze Bitmap rückwärts, aber in unserem Fall ist sie von oben nach unten abgespeichert.

Auf diesem Foto wird eine Bitmap dargestellt, die einen Smiley abbildet. Wir sehen, dass eine Bitmap als ein zwei-dimensionales-Array repräsentiert werden kann. Somit wissen wir dann auch, wie wir damit arbeiten können.

Bitte beachte, dass wir von rechts oben, nach links unten arbeiten.

3.1 Hexadezimale Werte und Farben

In der digitalen Bildbearbeitung wird oft das Hexadezimalsystem verwendet, um Farbwerte anzugeben. Ein Hexadezimalwert setzt sich aus Ziffern zusammen, die von 0 bis 9 und von A bis F reichen, wobei A für 10 und F für 15 steht. Dieses System ermöglicht eine kompakte Darstellung von Farbwerten. In einem 24-Bit-Farbschema, wie es häufig für digitale Bilder verwendet wird, besteht jede Farbe aus drei Komponenten: Rot (R), Grün (G) und Blau (B). Jede Komponente kann Werte von 0 bis 255 annehmen, was in Hexadezimal von 00 bis FF entspricht. Ein Hexadezimalfarbwert wie #FF0000 repräsentiert zum Beispiel ein reines Rot, da der Rotanteil maximal ist (FF), während Grün und Blau auf 0 gesetzt sind (00 und 00). Die Verwendung von Hexadezimalwerten ermöglicht eine präzise und effiziente Steuerung der Farben in digitalen Bildern.

3.2 Verständnis von RGB-Farbwerten

RGB steht für Rot, Grün und Blau - die drei primären Lichtfarben, aus denen durch ihre unterschiedlichen Kombinationen nahezu alle anderen Farben im digitalen Farbraum erzeugt werden können. Jede Farbe im RGB-System wird durch einen spezifischen Wert für Rot, Grün und Blau definiert, wobei jeder Wert typischerweise im Bereich von 0 bis 255 liegt. Diese Zahlen repräsentieren die Intensität oder Helligkeit der jeweiligen Farbe. Zum Beispiel entspricht (255, 0, 0) einem reinen Rot, da die Intensität von Rot maximal ist, während Grün und Blau keine Beiträge leisten. Ein Wert von (0, 0, 0) steht für Schwarz, da alle Lichtquellen ausgeschaltet sind, und (255, 255, 255) repräsentiert Weiß, bei dem Rot, Grün und Blau mit voller Intensität leuchten.

4 Gegebene Code

Diesen Code sollen Sie nutzen und nur die vier Funktion implementieren. Nehmen Sie keine anderen Änderungen vor.

[ ]:
```python
from PIL import Image # nicht ändern

def load_image(infile): # nicht ändern
    # Lädt ein Bild und gibt es als ein PIL Image-Objekt zurück.
    return Image.open(infile)

def save_image(image, outfile): # nicht ändern
    # Speichert ein PIL Image-Objekt in einer Datei.
    image.save(outfile)

def grayscale(image): # TODO

    return image

def sepia(image): # TODO

    return image

def reflect(image): # TODO

    return image

def blur(image): # TODO

    return image

def main(): # nicht Ändern
    # Hauptfunktion zum Anwenden von Filtern auf Bilder.
    import sys
    if len(sys.argv) != 4:
        print("Usage: python filter.py [filter] infile outfile")
        sys.exit(1)

    _, filter_arg, infile, outfile = sys.argv

    # Bild laden
    image = load_image(infile)
    image = image.convert("RGB")  # Stellt sicher, dass das Bild im RGB-Modus␣
 ↪ist

    # Filter auswählen und anwenden
    if filter_arg == 'g':
```

```
        filtered_image = grayscale(image)
    elif filter_arg == 's':
        filtered_image = sepia(image)
    elif filter_arg == 'r':
        filtered_image = reflect(image)
    elif filter_arg == 'b':
        filtered_image = blur(image)
    else:
        print("Invalid filter.")
        sys.exit(2)

    # Gefiltertes Bild speichern
    save_image(filtered_image, outfile)

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

5 Herangehensweise

Für diese Aufgabe muss man die Bibliothek Pillow herunterlanden.

5.1 Installation von Pillow

1. Öffnen Sie die Kommandozeile oder das Terminal:
   - Unter Windows können Sie die Suche verwenden, um nach "cmd" zu suchen und die Eingabeaufforderung zu öffnen.
   - Unter macOS können Sie das Terminal über Finder > Anwendungen > Dienstprogramme > Terminal öffnen.
   - Unter Linux öffnen Sie das Terminal über das Anwendungsmenü oder mit einer Tastenkombination, die je nach Distribution variiert.
2. Installieren Sie Pillow mit pip: Geben Sie den folgenden Befehl ein, um Pillow zu installieren: `pip install Pillow` oder `pip3 install Pillow`
3. Überprüfung der Installation: Nach der Installation können Sie überprüfen, ob Pillow erfolgreich installiert wurde, indem Sie in der Python-Shell den folgenden Befehl ausführen: `from PIL import Image` Wenn dieser Befehl keine Fehler ausgibt, wurde Pillow erfolgreich installiert und ist bereit für die Verwendung in Ihren Projekten.

5.2 Grayscale-Filter

Einführung In der digitalen Bildbearbeitung ist die Umwandlung von Farbbildern in Graustufen eine grundlegende Technik, die vielfältige Anwendungen findet. Ein Graufilter reduziert die Farbinformationen eines Bildes, sodass jedes Pixel nur noch Intensitäten von Schwarz bis Weiß aufweist. Diese Art der Bildbearbeitung ist nicht nur aus ästhetischen Gründen beliebt, sondern auch für praktische Anwendungen unerlässlich. Ein nützliches Beispiel ist die Verbesserung der Texterkennung (OCR - Optical Character Recognition) in gescannten Dokumenten. Farbinformationen können die OCR-Genauigkeit beeinträchtigen, während der Kontrast in einem Graustufenbild oft die Erkennungsrate verbessert, indem er die Klarheit der Textkanten erhöht.

Technische Umsetzung Wenn die Farbwerte eines Pixels 0 sind, ist der Pixel schwarz. Wenn sie 255 oder FF sind, dann ist der Pixel weiß. Wenn in einem Pixel alle Farbwerte den gleichen Wert beinhalten, dann bekommen wir einen Grauton. Diese ist von der größe des Wertes abhängig. Kleine Werte führen zu einem dunkleren Grauton und höhere Werte zu einem helleren Grauton. Beispiele: RGB: (127, 127, 127), (13, 13, 13) oder in HEX: 0xAAAAAA, 0x999999. Die `0x` ist nur eine Formalität. Damit geben wir an, dass es sich um einen Hexwert handelt.

Da wir einen Grauwert erhalten, wenn alle Farbwerte gleich sind, müssen wir also nur die Pixelwerte anpassen. Wie können wir das erreichen? Eine sinnvolle Herangehensweise ist es den Durchschnittwert der einzelnen Farbwerte innerhalb eines Pixels zu berechnen. Dann müssen wir nur noch die Werte aktualisieren.

5.3 Sepia-Filter: Ein Hauch von Nostalgie

Einführung Der Sepia-Filter ist ein beliebtes Werkzeug in der digitalen Bildbearbeitung, um Fotos einen klassischen, altmodischen Look zu verleihen. Der Effekt imitiert das Aussehen früher Fotografien, die oft in Sepia getönt waren, um ihre Langlebigkeit zu erhöhen. Das Ziel des Sepia-Filters ist es, moderne Bilder so aussehen zu lassen, als wären sie mit den fotografischen Techniken des späten 19. oder frühen 20. Jahrhunderts aufgenommen worden, indem er ihnen einen warmen, braunen Farbton verleiht.

Technische Umsetzung Um den Sepia-Effekt zu erzielen, werden die RGB-Werte (Rot, Grün, Blau) jedes Pixels im Bild nach spezifischen Formeln angepasst, die darauf abzielen, die originalen Farben in die charakteristischen Sepia-Töne umzuwandeln. Die Umrechnung erfolgt nach den folgenden Gleichungen:

```
SepiaRot = 0.393 * originalRot + 0.769 * originalGrün + 0.189 * originalBlau
SepiaGrün = 0.349 * originalRot + 0.686 * originalGrün + 0.168 * originalBlau
SepiaBlau = 0.272 * originalRot + 0.534 * originalGrün + 0.131 * originalBlau
```

Diese Berechnungen sorgen dafür, dass die resultierenden Farbwerte einen Braunton annehmen, der typisch für Sepia-Fotografien ist. Es ist wichtig, die Ergebnisse der Berechnung zu überprüfen und sicherzustellen, dass keine der neuen Farbwerte 255 überschreitet. Sollte dies der Fall sein, wird der Wert auf 255 begrenzt, um Überlauf zu vermeiden und die Farben korrekt darzustellen. Anschließend muss man nur noch die Farbwerte aktualisieren.

5.4 Reflect-Filter

Einführung Der Reflect-Filter, auch als Spiegelungsfilter bekannt, ist eine faszinierende Methode, um digitale Bilder auf kreative Weise zu manipulieren. Durch die Anwendung dieses Filters wird ein Bild entlang seiner vertikalen Achse gespiegelt, was zu einer interessanten und manchmal surrealen visuellen Erfahrung führt. Diese Technik kann genutzt werden, um einzigartige Kompositionen zu erstellen, die Aufmerksamkeit erregen oder spezifische visuelle Effekte erzielen, die in der ursprünglichen Aufnahme nicht vorhanden sind. Der Reflect-Filter kann in verschiedenen kreativen Projekten eingesetzt werden. Zum Beispiel kann er dazu verwendet werden, eine symmetrische Komposition zu erstellen, die das Gefühl von Balance und Harmonie verstärkt. In der Landschafts- und Architekturfotografie kann der Filter interessante Effekte erzeugen, indem er die Illusion einer Wasserspiegelung erzeugt oder die Symmetrie eines Gebäudes hervorhebt. In der Porträtfotografie kann die Anwendung des Reflect-Filters ungewöhnliche und auffällige Ergebnisse liefern, die die doppelte Präsenz einer Person oder eines Objekts im Bild betonen.

Technische Umsetzung Die Implementierung des Reflect-Filters in der Bildbearbeitung erfolgt durch das horizontale Umkehren des Bildes. Konkret bedeutet dies, dass für jedes Pixel in der linken Hälfte des Bildes ein entsprechendes Pixel in der rechten Hälfte gefunden und die Farbwerte dieser beiden Pixel miteinander getauscht werden. Diese Operation wird für alle Pixel bis zur Mitte des Bildes durchgeführt, wobei die Mitte als die vertikale Achse fungiert, an der das Bild gespiegelt wird.

5.5 Blur-Filter

Einführung Der Blur-Filter, oder Unschärfefilter, ist ein essenzielles Werkzeug in der digitalen Bildbearbeitung und Fotografie. Er wird verwendet, um ein Bild weicher zu machen, indem die Klarheit der Kanten und Details verringert wird. Diese Technik kann hilfreich sein, um visuelles Rauschen oder Unvollkommenheiten in Fotos zu minimieren, den Fokus auf bestimmte Bildbereiche zu lenken oder einfach um ästhetische Effekte zu erzielen, die ein Gefühl von Bewegung oder Traumhaftigkeit vermitteln.

Technische Umsetzung Die Implementierung des Blur-Filters basiert auf der Durchschnittsbildung der Farbwerte eines Pixels und seiner Umgebung. Durch das Ersetzen jedes Pixelwertes im Bild mit dem Durchschnitt seiner umliegenden Pixel, werden scharfe Kanten und feine Details gemildert, was zu einer allgemeinen Unschärfe führt. Dieser Prozess wird für jedes Pixel im Bild wiederholt, was sicherstellt, dass das gesamte Bild gleichmäßig beeinflusst wird. Beim Anwenden eines Unschärfefilters auf ein Bild ist der Umgang mit den Kanten und Ecken des Bildes eine besondere Herausforderung. An den Rändern des Bildes haben Pixel weniger Nachbarn, was bei der Durchschnittsberechnung berücksichtigt werden muss, um unerwünschte Effekte oder sichtbare Ränder im Bild zu vermeiden.

1. Padding (Auffüllen): Eine Methode besteht darin, das Bild mit einer zusätzlichen Schicht von Pixeln um die Ränder herum "aufzufüllen". Dies kann durch das Kopieren der Randpixel, das Hinzufügen einer festen Farbe oder das Spiegeln der nahegelegenen Pixel erfolgen. Das Padding ermöglicht es, für Randpixel eine vollständige Gruppe von Nachbarn zu haben, was eine gleichmäßigere Unschärfe über das gesamte Bild hinweg ermöglicht.

2. Ignorieren der Randpixel: Eine einfachere, aber weniger bevorzugte Methode ist, die Randpixel bei der Unschärfeberechnung weniger zu berücksichtigen oder ganz auszulassen. Diese Technik kann jedoch zu ungleichmäßigen Effekten an den Bildrändern führen.

3. Anpassung der Berechnung für Randpixel: Eine weitere Möglichkeit besteht darin, die Formel für die Unschärfe so anzupassen, dass für Pixel am Rand oder in der Ecke des Bildes nur die tatsächlich vorhandenen Nachbarn berücksichtigt werden. Dies bedeutet, dass der Durchschnitt basierend auf einer kleineren Anzahl von Pixeln berechnet wird. Diese Methode verhindert harte Kanten oder offensichtliche Übergänge an den Rändern des Bildes.

Probieren Sie die 3. Methode anzuwenden, für diese gibt es auch Unterstützung in diesem Artikel. Es kann sehr anspruchsvoll werden, aber lasst euch nicht entmutigen. Ich habe auch große Hilfe bei diesem Filter gebraucht.

6 Hinweise

Zuerst müssen wir die größe des Bildes bestimmen, damit wir wissen wie groß unser Array sein soll. Dies tuen wir mit `width, height = image.size`

Um auf jedes einzelne Pixel zugreifen zu können, brauchen wir folgende Schleifen:

[ ]:
```
for y in range(height):
    for x in range(width):
```

Jetzt können Sie auf die einzelnen Pixel zugreifen. Jetzt brauchen Sie noch auf die einzelnen Farbwerte zuzugreifen: `r, g, b = image.getpixel((x, y))`.

Um die Farbwerte dann zu aktualisieren: `image.putpixel((x, y), (farbwert, farbwert, farbwert))`. Farbwert ersetzen Sie in diesem Fall mit dem Durchschnitt.

6.1 Grayscale

[ ]:
```
def grayscale(image):
    width, height = image.size
    # durch alle Pixel iterieren
    for y in range(height):
        for x in range(width):
            # die einzelnen Farbwerte erhalten
            # den Durchschnitt von rot, grün und blau berechnen und in einer␣
 ↪Variable abspeichern
            # dann die Pixelwerte für rot, grün und blau aktualisieren, also␣
 ↪durch den Durchschnitt ersetzen
```

6.1.1 weiterer Hinweis (aufklappen)

[ ]:
```
def grayscale(image):
    width, height = image.size
    # durch jedes Pixel iterieren
    for y in range(height):
        for x in range(width):
            r, g, b = image.getpixel((x, y)) # die einzelnen Farbwerte erhalten
            # den Durchschnitt berechnen
            image.putpixel((x, y), (avg, avg, avg)) # Farbwerte aktualisieren
    return image
```

6.2 Sepia

[ ]:
```
def sepia(image):
    width, height = image.size
    # Durch alle Pixel iterieren
    for y in range(height):
        for x in range(width):
            # Die einzelnen Farbwerte erhalten
            # Sepia-Farbwerte berechnen
            # Die Sepia-Farbwerte auf maximal 255 begrenzen
            # Dann die Pixelwerte durch die Sepia-Farbwerte ersetzen
    return image
```

6.2.1 weiterer Hinweis (aufklappen)

```
[ ]: def sepia(image):
         width, height = image.size
         # Durch alle Pixel iterieren
         for y in range(height):
             for x in range(width):
                 r, g, b = image.getpixel((x, y)) # Die einzelnen Farbwerte erhalten
                 # Sepia-Farbwerte berechnen
                 # Die Sepia-Farbwerte auf maximal 255 begrenzen
                 image.putpixel((x, y), (sepiaRed, sepiaGreen, sepiaBlue)) # Dann␣
      ↪die Pixelwerte durch die Sepia-Farbwerte ersetzen
         return image
```

6.3 Reflect

```
[ ]: def reflect(image):
         width, height = image.size
         for y in range(height):
             for x in range(width // 2): # Durch alle Pixel iterieren, aber nur bis␣
      ↪zur Hälfte der Breite, da wir Pixel von beiden Seiten tauschen
                 # Die Pixel auf der linken Seite erhalten
                 # Die entsprechenden Pixel auf der rechten Seite erhalten
                 # Pixel auf der linken Seite mit denen auf der rechten Seite␣
      ↪tauschen
         return image
```

6.3.1 weiterer Hinweis (aufklappen)

```
[ ]: def reflect(image):
         width, height = image.size
         for y in range(height):
             for x in range(width // 2): # Durch alle Pixel iterieren, aber nur bis␣
      ↪zur Hälfte der Breite, da wir Pixel von beiden Seiten tauschen
                 left_pixel = image.getpixel((x, y)) # Die Pixel auf der linken␣
      ↪Seite erhalten
                 # Die entsprechenden Pixel auf der rechten Seite erhalten. Was muss␣
      ↪anders sein gegenüber der linken Pixel?

                 # rechte Pixel aktualisieren. Was muss anders sein, gegenüber der␣
      ↪linken Pixel?
                 image.putpixel((width - 1 - x, y), left_pixel) # linke Pixel␣
      ↪aktualisieren
         return image
```

6.4 Blur

Wie kann man eine Kopie des Bildes erstellen? Mit `copy = image.copy()` Wir müssen es jetzt hinbekommen, alle umgebenden Pixel, mit ihren Farbwerten, zu sammeln. Dies können wir tun, mit zwei `for`-Schleifen.

[ ]:
```
for dy in [-1, 0, 1]:
    for dx in [-1, 0, 1]:
```

Diese verschachtelten Schleifen durchlaufen ein 3x3-Grid um ein zentrales Pixel herum. Jede Schleife (`for dy in [-1, 0, 1]` und `for dx in [-1, 0, 1]`) iteriert über drei Werte: -1, 0 und 1. Diese Werte repräsentieren die relative Position der umgebenden Pixel zum aktuellen Pixel in der Mitte des Grids. - `dy` steht für die vertikale Verschiebung relativ zum aktuellen Pixel: -1 entspricht einem Pixel oberhalb, 0 bleibt auf der gleichen horizontalen Linie, und 1 entspricht einem Pixel unterhalb. - `dx` steht für die horizontale Verschiebung relativ zum aktuellen Pixel: -1 entspricht einem Pixel links, 0 bleibt in derselben vertikalen Spalte, und 1 entspricht einem Pixel rechts.

Um nun die genauen Koordinaten jedes umgebenden Pixels zu bestimmen, verwenden wir:

`nx, ny = x + dx, y + dy` # Berechnung der neuen Koordinaten für jedes umgebende Pixel

Diese Zeile berechnet die neuen Koordinaten `nx` und `ny`, indem sie die Verschiebungen `dx` und `dy` zu den aktuellen Koordinaten `x` und `y` des zentralen Pixels addiert. Dadurch werden die Positionen aller umgebenden Pixel relativ zum zentralen Pixel festgelegt, was es uns ermöglicht, deren Farbwerte für die weitere Verarbeitung zu sammeln.

Nachdem die neuen Koordinaten berechnet wurden, steht uns nun die Herausforderung gegenüber, sicherzustellen, dass diese innerhalb der Bildgrenzen liegen. Dies ist von entscheidender Bedeutung, da ein Zugriff auf Pixel außerhalb des Bildbereichs zu Fehlern führen würde. Besonders an den Rändern und Ecken des Bildes, wo einige umgebende Pixel theoretisch außerhalb des Bildbereichs liegen könnten, muss dies berücksichtigt werden. Die Lösung für dieses Problem wird durch folgende Bedingung realisiert:

[ ]:
```
# Überprüfung, ob die neuen Koordinaten innerhalb der Bildgrenzen liegen
if 0 <= nx < width and 0 <= ny < height:
    pixels.append(copy.getpixel((nx, ny)))
```

Diese Überprüfung gewährleistet, dass nur die Farbwerte von Pixeln gesammelt werden, die innerhalb des gültigen Bereichs des Bildes liegen. Durch diese Methode können wir eine korrekte Berechnung des Durchschnitts für den Unschärfeeffekt durchführen, ohne dass es zu Verzerrungen an den Bildrändern kommt.

[ ]:
```
def blur(image):
    width, height = image.size
    # Erstellen einer Kopie des Bildes, um die Originalpixelwerte zu behalten
    # Durch alle Pixel iterieren
    for y in range(height):
        for x in range(width):
            # Sammeln der Farbwerte der umgebenden Pixel einschließlich des␣
 ↪aktuellen Pixels
```

```
                    # Überprüfen, ob der neue Index innerhalb der Bildgrenzen␣
↪liegt
            # Berechnen des Durchschnitts der Farbwerte
            # Aktualisieren des aktuellen Pixels mit dem Durchschnittswert
    return image
```

6.4.1 weiterer Hinweis (aufklappen)

[ ]:
```
def blur(image):
    width, height = image.size
    copy = image.copy()  # Erstellen einer Kopie des Bildes, um die␣
↪Originalpixelwerte zu behalten
    # Durch alle Pixel iterieren
    for y in range(height):
        for x in range(width):
            # Sammeln der Farbwerte der umgebenden Pixel einschließlich des␣
↪aktuellen Pixels
            pixels = []
            for dy in [-1, 0, 1]:
                for dx in [-1, 0, 1]:
                    nx, ny = x + dx, y + dy
                    # Überprüfen, ob der neue Index innerhalb der Bildgrenzen␣
↪liegt
                    if 0 <= nx < width and 0 <= ny < height:
                        # die Anweisung hinzufügen

            # Berechnen des Durchschnitts der Farbwerte
            avg_r = sum(p[0] for p in pixels) // len(pixels)
            # jetzt das gleiche für grün
            # und dann noch für blau


            # Aktualisieren des aktuellen Pixels mit dem Durchschnittswert,␣
↪sowie im Grayscale-Filter
    return image
```

7 Lösungen

7.1 Grayscale komplette Lösung (aufklappen)

[ ]:
```
def grayscale(image):
    width, height = image.size
    # durch jedes Pixel iterieren
    for y in range(height):
        for x in range(width):
            r, g, b = image.getpixel((x, y)) # die einzelnen Farbwerte erhalten
```

```
            avg = (r + g + b) // 3 # den Durchschnitt berechnen
            image.putpixel((x, y), (avg, avg, avg)) # Farbwerte aktualisieren
    return image
```

7.2 Sepia komplette Lösung (aufklappen)

[ ]:
```
def sepia(image):
    width, height = image.size
    for y in range(height):
        for x in range(width):
            r, g, b = image.getpixel((x, y))
            sepiaRed = int(0.393 * r + 0.769 * g + 0.189 * b)
            sepiaGreen = int(0.349 * r + 0.686 * g + 0.168 * b)
            sepiaBlue = int(0.272 * r + 0.534 * g + 0.131 * b)
            image.putpixel((x, y), (
                min(sepiaRed, 255),
                min(sepiaGreen, 255),
                min(sepiaBlue, 255)
            ))
    return image
```

7.3 Reflect komplette Lösung (aufklappen)

[ ]:
```
def reflect(image):
    width, height = image.size
    for y in range(height):
        for x in range(width // 2): # Durch alle Pixel iterieren, aber nur bis␣
↪zur Hälfte der Breite, da wir Pixel von beiden Seiten tauschen
            left_pixel = image.getpixel((x, y)) # Die Pixel auf der linken␣
↪Seite erhalten
            right_pixel = image.getpixel((width - 1 - x, y)) # Die␣
↪entsprechenden Pixel auf der rechten Seite erhalten
             # Pixel auf der linken Seite mit denen auf der rechten Seite␣
↪tauschen
            image.putpixel((x, y), right_pixel)
            image.putpixel((width - 1 - x, y), left_pixel)
    return image
```

7.4 Blur komplette Lösung (aufklappen)

[ ]:
```
def blur(image):
    width, height = image.size
    copy = image.copy()  # Erstellen einer Kopie des Bildes, um die␣
↪Originalpixelwerte zu behalten
    # Durch alle Pixel iterieren
    for y in range(height):
```

```
        for x in range(width):
            # Sammeln der Farbwerte der umgebenden Pixel einschließlich des␣
↪aktuellen Pixels
            pixels = []
            for dy in [-1, 0, 1]:
                for dx in [-1, 0, 1]:
                    nx, ny = x + dx, y + dy
                    # Überprüfen, ob der neue Index innerhalb der Bildgrenzen␣
↪liegt
                    if 0 <= nx < width and 0 <= ny < height:
                        pixels.append(copy.getpixel((nx, ny)))
            # Berechnen des Durchschnitts der Farbwerte
            avg_r = sum(p[0] for p in pixels) // len(pixels)
            avg_g = sum(p[1] for p in pixels) // len(pixels)
            avg_b = sum(p[2] for p in pixels) // len(pixels)

            image.putpixel((x, y), (avg_r, avg_g, avg_b)) # Aktualisieren des␣
↪aktuellen Pixels mit dem Durchschnittswert
    return image
```

8 Programm ausführen

Nachdem alle wenigstens eine Funktion implementiert ist, können Sie das Programm ausführen.

Gehen Sie dafür in eine Konsole oder Terminal und stellen Sie sicher, dass Sie sich im selben Ordner befinden, wie Ihr Pythonskript.

Dann können Sie folgde Befehle ausführen, um die Filter anzuwenden und neue Bilder erstellen:

```
python3 ./scriptName.py filter eingabeBild ausgabeBild
```

Hier ein spezifisches Beispiel:

```
python3 .\bildfilter.py g 'C:\Users\mikyt\Downloads\hfu_tuttlingen.bmp'
output_grayscale.bmp
```

Bitte beachtet, dass die Eingabe vom Betriebssystem und der Art, wie Python installiert ist abhängig ist. Bei Probleme, kopiert die Fehlermeldung in ein LLM euerer Wahl.

9 Gesamter Code

Nur aufmachen im Notfall. Enhällt gesamte Lösung.

[ ]:
```
from PIL import Image # nicht ändern

def load_image(infile): # nicht ändern
    # Lädt ein Bild und gibt es als ein PIL Image-Objekt zurück.
    return Image.open(infile)
```

```python
def save_image(image, outfile): # nicht ändern
    # Speichert ein PIL Image-Objekt in einer Datei.
    image.save(outfile)

def grayscale(image):
    width, height = image.size
    # durch jedes Pixel iterieren
    for y in range(height):
        for x in range(width):
            r, g, b = image.getpixel((x, y)) # die einzelnen Farbwerte erhalten
            avg = (r + g + b) // 3 # den Durchschnitt berechnen
            image.putpixel((x, y), (avg, avg, avg)) # Farbwerte aktualisieren
    return image

def sepia(image):
    width, height = image.size
    for y in range(height):
        for x in range(width):
            r, g, b = image.getpixel((x, y))
            sepiaRed = int(0.393 * r + 0.769 * g + 0.189 * b)
            sepiaGreen = int(0.349 * r + 0.686 * g + 0.168 * b)
            sepiaBlue = int(0.272 * r + 0.534 * g + 0.131 * b)
            image.putpixel((x, y), (
                min(sepiaRed, 255),
                min(sepiaGreen, 255),
                min(sepiaBlue, 255)
            ))
    return image

def reflect(image):
    width, height = image.size
    for y in range(height):
        for x in range(width // 2): # Durch alle Pixel iterieren, aber nur bis␣
 ↪zur Hälfte der Breite, da wir Pixel von beiden Seiten tauschen
            left_pixel = image.getpixel((x, y)) # Die Pixel auf der linken␣
 ↪Seite erhalten
            right_pixel = image.getpixel((width - 1 - x, y)) # Die␣
 ↪entsprechenden Pixel auf der rechten Seite erhalten
             # Pixel auf der linken Seite mit denen auf der rechten Seite␣
 ↪tauschen
            image.putpixel((x, y), right_pixel)
            image.putpixel((width - 1 - x, y), left_pixel)
    return image

def blur(image):
    width, height = image.size
```

```
    copy = image.copy()  # Erstellen einer Kopie des Bildes, um die
↪Originalpixelwerte zu behalten
    # Durch alle Pixel iterieren
    for y in range(height):
        for x in range(width):
            # Sammeln der Farbwerte der umgebenden Pixel einschließlich des
↪aktuellen Pixels
            pixels = []
            for dy in [-1, 0, 1]:
                for dx in [-1, 0, 1]:
                    nx, ny = x + dx, y + dy
                    # Überprüfen, ob der neue Index innerhalb der Bildgrenzen
↪liegt
                    if 0 <= nx < width and 0 <= ny < height:
                        pixels.append(copy.getpixel((nx, ny)))
            # Berechnen des Durchschnitts der Farbwerte
            avg_r = sum(p[0] for p in pixels) // len(pixels)
            avg_g = sum(p[1] for p in pixels) // len(pixels)
            avg_b = sum(p[2] for p in pixels) // len(pixels)

            image.putpixel((x, y), (avg_r, avg_g, avg_b)) # Aktualisieren des
↪aktuellen Pixels mit dem Durchschnittswert
    return image


def main(): # nicht Ändern
    # Hauptfunktion zum Anwenden von Filtern auf Bilder.
    import sys
    if len(sys.argv) != 4:
        print("Usage: python filter.py [filter] infile outfile")
        sys.exit(1)

    _, filter_arg, infile, outfile = sys.argv

    # Bild laden
    image = load_image(infile)
    image = image.convert("RGB")  # Stellt sicher, dass das Bild im RGB-Modus
↪ist

    # Filter auswählen und anwenden
    if filter_arg == 'g':
        filtered_image = grayscale(image)
    elif filter_arg == 's':
        filtered_image = sepia(image)
    elif filter_arg == 'r':
        filtered_image = reflect(image)
    elif filter_arg == 'b':
```

```
        filtered_image = blur(image)
    else:
        print("Invalid filter.")
        sys.exit(2)

    # Gefiltertes Bild speichern
    save_image(filtered_image, outfile)

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