BV1 2025
Bericht zu Übungsblatt 1:
Primzahlen und Grundlagen der Bildverarbeitung mit OpenCV

Nik Tykhomyrov Jonas Pardeyke

4. April 2025

Inhaltsverzeichnis

1 Einleitung

1.1 Thema und Kontext

Diese Arbeit dokumentiert die Bearbeitung ausgewählter Aufgaben des Übungsblattes 1 im Modul BV1 (Bildverarbeitung 1). Der Fokus liegt dabei auf zwei Kernbereichen: der algorithmischen Generierung von Primzahlen und den grundlegenden Operationen der digitalen Bildverarbeitung mit Python und der Bibliothek OpenCV. Die Aufgaben dienen als Einstieg in die praktische Arbeit mit Entwicklungsumgebungen und relevanten Bibliotheken für die Bildverarbeitung.

1.2 Ziele

Das primäre Ziel dieser Arbeit ist die Dokumentation der Lösungen für die Aufgaben 3 bis 7 des Übungsblattes. Dies umfasst:

- Die Implementierung eines Primzahlengenerators unter Verwendung einer selbstdefinierten Funktion zur Primzahlprüfung.
- Das Einlesen, Anzeigen, Speichern und erneute Laden eines digitalen Bildes unter Verwendung von OpenCV.
- Die Untersuchung der Auswirkungen verlustbehafteter Kompression (JPEG) durch den Vergleich des Originalbildes mit seiner komprimierten Version mittels einer geeigneten Punktoperation.

Das übergeordnete Ziel ist es, grundlegende Programmierkenntnisse in Python zu festigen und erste Erfahrungen im Umgang mit digitalen Bildern und der OpenCV-Bibliothek zu sammeln.

1.3 Relevanz

Die in dieser Arbeit behandelten Themen sind von grundlegender Bedeutung. Primzahlen spielen eine wichtige Rolle in der Kryptographie und der theoretischen Informatik. Die digitale Bildverarbeitung ist ein omnipräsentes Feld mit Anwendungen von der Medizintechnik über die industrielle Qualitätskontrolle bis hin zur Unterhaltungselektronik und autonomen Systemen. Kompressionsverfahren wie JPEG und grundlegenden Analyseoperationen ist essenziell für weiterführende Arbeiten in diesem Bereich.

1.4 Aktueller Stand und Grundlagen

1.4.1 Bildverarbeitung mit Python und OpenCV

Python hat sich zusammen mit Bibliotheken wie OpenCV (Open Source Computer Vision Library) und NumPy (Numerical Python) als Standardwerkzeug in der Bildverarbeitung etabliert. OpenCV bietet eine breite Palette von Funktionen für Bild- und Videoanalyse, während NumPy die effiziente Handhabung von Bilddaten als mehrdimensionale Arrays ermöglicht.

1.4.2 Digitale Bilder und Kompression

Ein digitales Bild wird typischerweise als Raster von Pixeln repräsentiert, wobei jeder Pixel einen Farb- oder Helligkeitswert speichert (z.B. über RGB-Kanäle). Bilddateiformate unterscheiden sich in ihrer Speicherung und Kompression. JPEG (Joint Photographic Experts Group) ist ein weit verbreitetes *verlustbehaftetes* Kompressionsverfahren, optimiert für fotografische Bilder. Es reduziert die Dateigröße signifikant, indem es Informationen entfernt, die für das menschliche Auge oft schwer wahrnehmbar sind. Dies führt jedoch zu irreversiblen Qualitätsverlusten. Im Gegensatz dazu stehen *verlustfreie* Formate wie PNG, die eine exakte Rekonstruktion des Originalbildes ermöglichen, aber meist weniger Speicherplatz sparen.

1.5 Aufbau der Arbeit

Diese Arbeit gliedert sich in einen Hauptteil und einen Schluss. Der Hauptteil erläutert die verwendeten Werkzeuge und Konzepte, beschreibt die Implementierung des Primzahlengenerators inklusive mathematischer Optimierung und stellt die Schritte der Bildverarbeitung dar, JPEG-Kompression und Analyse von Unterschieden mittels Punktoperation. Abschließend werden die Ergebnisse präsentiert und diskutiert. Der Schluss fasst die Ergebnisse zusammen, bewertet die Zielerreichung und schlägt mögliche weiterführende Schritte vor.

2 Hauptteil

2.1 Verwendete Werkzeuge und Grundlagen

Zur Lösung der Aufgaben wurde die Programmiersprache **Python** (Version 3.12.9) verwendet. Die wesentlichen externen Bibliotheken sind:

- **OpenCV (`cv2`)**: Eine umfassende Bibliothek für Computer Vision und Bildverarbeitung. Sie wurde hier für das Einlesen (`cv2.imread`), Anzeigen (`cv2.imshow` bzw. alternative Darstellung mit Matplotlib), Speichern (`cv2.imwrite`), die Berechnung der absoluten Differenz (`cv2.absdiff`) und die Skalierung von Bilddaten (`cv2.convertScaleAbs`) genutzt.
- **NumPy (`numpy`)**: Eine fundamentale Bibliothek für numerische Berechnungen in Python. OpenCV repräsentiert Bilder als NumPy-Arrays, was effiziente pixelweise Operationen und statistische Auswertungen (z.B. `numpy.mean`, `numpy.max`) ermöglicht.
- **Matplotlib (`matplotlib.pyplot`)**: Wurde optional zur Darstellung der Bilder innerhalb der Jupyter-Notebook-Umgebung verwendet, da `cv2.imshow` dort nicht immer zuverlässig funktioniert.

Wie in der Einleitung erwähnt, ist eine **Punktoperation** eine Bildtransformation, bei der jeder Ausgabepixelwert $g(x, y)$ nur vom Eingabepixelwert an derselben Position $f(x, y)$ abhängt: $g(x, y) = T(f(x, y))$. Die hier verwendete Differenzbildung zwischen zwei Bildern I_1 und I_2, $D(x, y) = |I_1(x, y) - I_2(x, y)|$, ist ebenfalls eine Punktoperation, da jeder Differenzpixel nur von den Werten der Eingangsbilder an der korrespondierenden (x, y)-Position abhängt.

2.2 Aufgabe 3: Primzahlengenerator

Ziel dieser Aufgabe war es, die ersten n Primzahlen zu generieren, wobei die Prüfung in einer eigenen Funktion erfolgen sollte.

2.2.1 Implementierung der Primzahlprüfung

Eine Zahl $p > 1$ ist eine Primzahl, wenn sie nur durch 1 und sich selbst ohne Rest teilbar ist. Die Funktion `is_prime` prüft dies mittels Probedivision.

```
def is_prime(number):
    if number <= 1:
        return False
    # Pruefe Teiler nur bis zur Quadratwurzel der Zahl
    # int(number**0.5) + 1 stellt sicher, dass die Obergrenze korrekt ist
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0: # Wenn teilbar, dann keine Primzahl
            return False
    return True # Keine Teiler gefunden (ausser 1 und sich selbst)
```

Listing 1: Python-Funktion zur Prüfung auf Primalität

Die entscheidende Optimierung liegt in der oberen Grenze der Schleife: `int(number**0.5) + 1`. Mathematisch gilt: Wenn eine Zahl `number` zusammengesetzt ist (d.h. keine Primzahl), dann hat sie mindestens einen Teiler d mit $d \leq \sqrt{\text{number}}$. Gäbe es nämlich keinen solchen Teiler, müssten alle Teiler größer als $\sqrt{\text{number}}$ sein. Wenn man zwei solche Teiler multipliziert, wäre das Produkt größer als number, was ein Widerspruch ist. Daher genügt es, Teiler bis zur Quadratwurzel von `number` zu prüfen ($\sqrt{\text{number}}$ entspricht `number**0.5`). Dies reduziert den Rechenaufwand erheblich gegenüber einer Prüfung bis `number - 1`.

2.2.2 Implementierung des Generators

Die Funktion `primeNumberGenerator` nutzt `is_prime`, um sequenziell Zahlen zu testen und die ersten n gefundenen Primzahlen in einer Liste zu sammeln.

```
def primeNumberGenerator(n):
    primes = []
    number = 2 # Erste zu pruefende Zahl
    while len(primes) < n:
```

```
        if is_prime(number):
            primes.append(number)
        number += 1
    return primes
```

Listing 2: Python-Funktion zur Generierung der ersten n Primzahlen

2.2.3 Ergebnis

Für $n = 10$ liefert der Generator die korrekte Liste der ersten 10 Primzahlen: `[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]`.

2.3 Aufgaben 4-7: Bildverarbeitung und Kompressionsanalyse

Diese Aufgabenkette befasst sich mit dem Laden eines Bildes, dessen Speicherung im verlustbehafteten JPEG-Format und der anschließenden Analyse der durch die Kompression entstandenen Unterschiede zum Original.

2.3.1 Vorgehen

Das Vorgehen folgte direkt den Aufgabenstellungen:

1. **Laden und Anzeigen (Aufgabe 4):** Das Bild `Set01.jpg` wurde mittels `cv2.imread` geladen und zur Kontrolle angezeigt. Das geladene Bild wird als I bezeichnet.
2. **Komprimiertes Speichern (Aufgabe 5):** Das Bild I wurde mit `cv2.imwrite` als `Set01_compressed.jpg` gespeichert. Dabei wurde das JPEG-Format mit einem Qualitätsparameter von 50 (auf einer Skala von 0-100) gewählt. Eine niedrigere Qualität bedeutet höhere Kompression und potenziell sichtbarere Artefakte.
3. **Erneutes Laden (Aufgabe 6):** Das komprimierte Bild `Set01_compressed.jpg` wurde mit `cv2.imread` geladen und als I_L bezeichnet.
4. **Differenzanalyse (Aufgabe 7):** Die Unterschiede zwischen dem Original I und dem geladenen komprimierten Bild I_L wurden mittels einer Punktoperation berechnet und visualisiert.

2.3.2 Implementierungsdetails

Laden und Speichern: Das Laden erfolgt über `img = cv2.imread('filename.jpg')`. Das Speichern als JPEG mit Qualitätsstufe 50 wird realisiert durch:

```
quality = 50
cv2.imwrite('Set01_compressed.jpg', imgGlobal,
            [cv2.IMWRITE_JPEG_QUALITY, quality])
```

Listing 3: Speichern eines Bildes als JPEG mit Qualitätsstufe 50

Hier ist `imgGlobal` die Variable, die das (Original-)Bild als NumPy-Array enthält.

Differenzberechnung und Visualisierung: Um die Unterschiede zwischen I (`imgGlobal`) und I_L (`imgCompressed`) sichtbar zu machen, wurde die absolute Differenz berechnet und anschließend verstärkt.

```
# Berechne den absoluten Unterschied pro Pixel
# D(x, y) = |I(x, y) - I_L(x, y)|
diff = cv2.absdiff(imgGlobal, imgCompressed)

# Verstaerke die Unterschiede fuer bessere Sichtbarkeit
# D_enhanced(x, y) = saturate(alpha * D(x, y) + beta)
alpha = 10.0 # Verstaerkungsfaktor
beta = 0     # Helligkeits-Offset (hier nicht verwendet)
diff_enhanced = cv2.convertScaleAbs(diff, alpha=alpha, beta=beta)
```

Listing 4: Berechnung und Verstärkung des Differenzbildes

`cv2.absdiff` berechnet die absolute Differenz für jeden Pixel und Farbkanal (B, G, R). Das Ergebnis `diff` ist ein Bild, bei dem Pixelwerte die Stärke der Abweichung repräsentieren. `cv2.convertScaleAbs` verstärkt diese geringen Unterschiede, insbesondere bei hoher JPEG-Qualität, und stellt sicher, dass das Ergebnis im gültigen Anzeigebereich für Bilder (typischerweise 0-255 für 8-Bit-Bilder) bleibt. Das resultierende Bild `diff_enhanced` hebt Bereiche mit signifikanten Kompressionsänderungen hervor.

2.3.3 Ergebnisse und Diskussion

Visueller Vergleich: Das Originalbild (Bild 1) zeigt eine Anordnung von Spielkarten mit verschiedenen geometrischen Symbolen auf einem hölzernen Untergrund. Zu sehen sind weiße Karten mit verschiedenfarbigen Symbolen wie blauen Wellenformen, roten und grünen Rauten sowie unterschiedlichen Kreisformen.

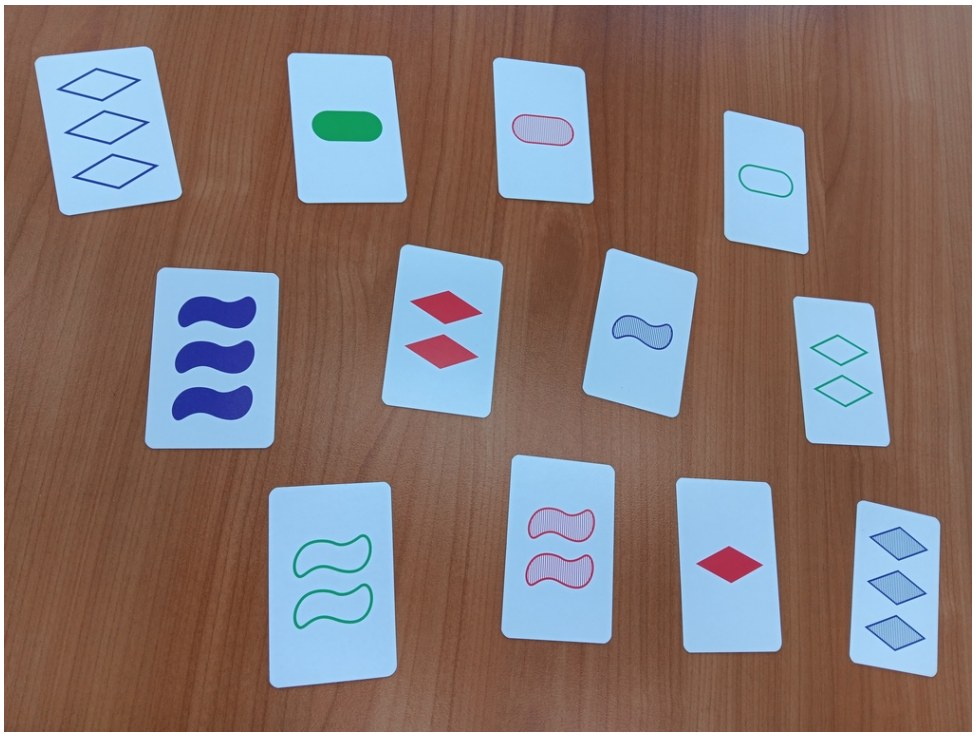

Abbildung 1: Originalbild (Set01.jpg)

Das komprimierte Bild (Bild 2) erscheint auf den ersten Blick sehr ähnlich zum Original. Bei direkter visueller Betrachtung sind kaum Unterschiede erkennbar. Die Farbwiedergabe, die Konturen der Karten und der Symbole erscheinen nahezu identisch.

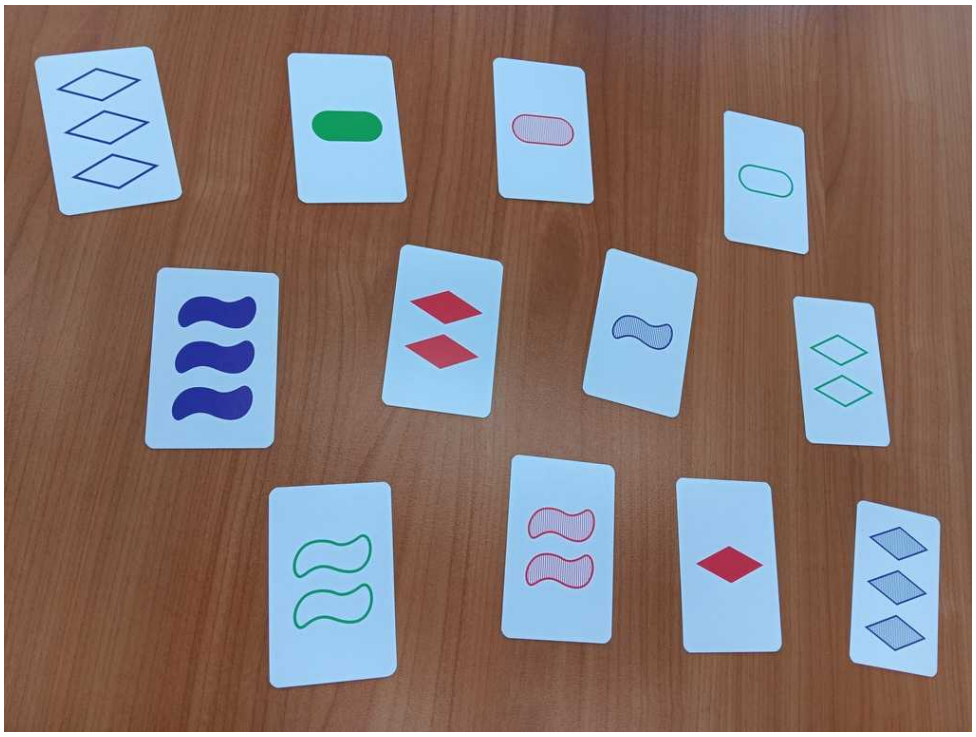

Abbildung 2: Komprimiertes Bild (Set01_compressed.jpg)

Differenzbild: Das verstärkte Differenzbild (Bild 3) zeigt die Auswirkungen der Kompression deutlich. Besonders auffällig sind die Konturen der Karten und der darauf abgebildeten Symbole, die sich im Differenzbild als helle Linien abzeichnen. Dies deutet darauf hin, dass die JPEG-Kompression vor allem an Kanten und Übergängen zwischen verschiedenen Farben Informationen verliert. Die farbigen Flächen innerhalb der Symbole und die homogenen Kartenflächen zeigen hingegen weniger Abweichungen.

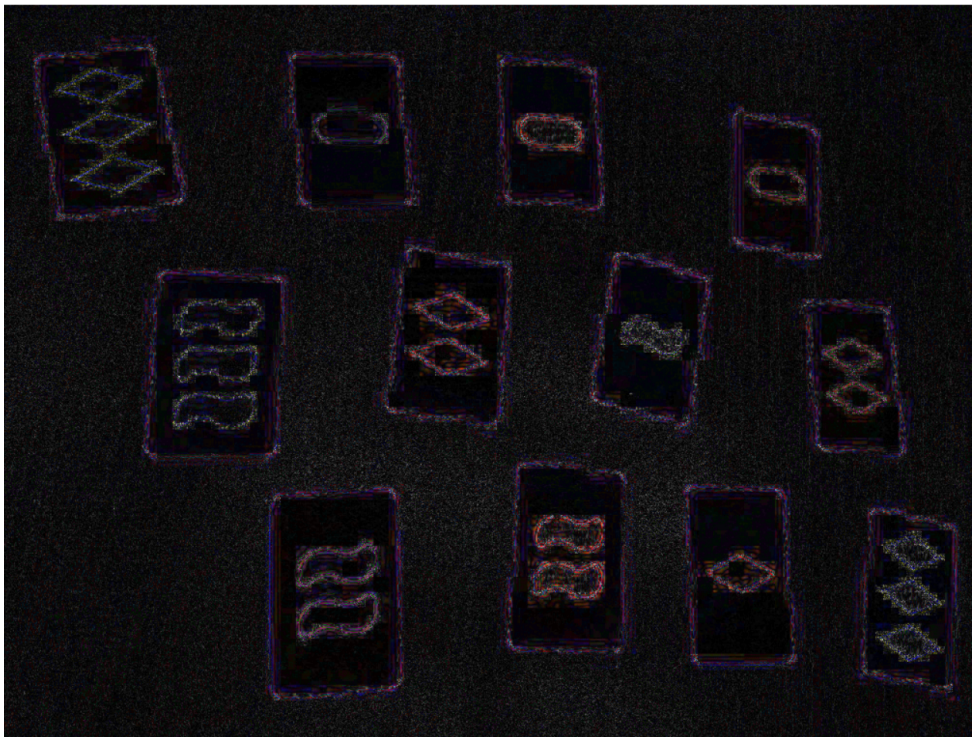

Abbildung 3: Verstärktes Differenzbild zwischen Original und komprimierter Version

3 Zusammenfassung und Ausblick

3.1 Zusammenfassung

In dieser Arbeit wurden grundlegende Aufgaben aus den Bereichen Algorithmik und digitaler Bildverarbeitung bearbeitet. Es wurde ein funktionsfähiger Primzahlengenerator implementiert, der auf einer optimierten Primzahlprüfungsfunktion basiert. Des Weiteren wurden die Schritte zum Laden, Speichern (unter Verwendung von verlustbehafteter JPEG-Kompression mit Qualitätsstufe 50) und erneuten Laden eines digitalen Bildes mittels Python und OpenCV erfolgreich durchgeführt. Schließlich wurden die durch die JPEG-Kompression entstandenen Unterschiede zum Originalbild mittels der Punktoperation `cv2.absdiff` quantifiziert und durch Skalierung (`cv2.convertScaleAbs`) visualisiert.

3.2 Zielerreichung

Die zu Beginn gesetzten Ziele wurden erreicht. Die geforderten Funktionalitäten (Primzahlgenerator, Bild I/O, Kompression, Differenzbildung) wurden erfolgreich implementiert und die Ergebnisse dokumentiert. Es konnte ein grundlegendes Verständnis für die Arbeitsweise des Primzahltests, die Handhabung von Bildern in Python/OpenCV und die sichtbaren sowie messbaren Auswirkungen der JPEG-Kompression erlangt werden.

3.3 Gelerntes

Im Vergleich zum Stand vor der Bearbeitung wurden folgende Erkenntnisse gewonnen:

- Praktische Anwendung und Optimierung von Algorithmen (Primzahltest mit $\sqrt{n}$-Grenze).
- Verwendung der OpenCV-Bibliothek für grundlegende Bildoperationen (I/O, arithmetische Operationen).
- Verständnis des Konzepts verlustbehafteter Kompression (JPEG) und dessen Konsequenzen (visuelle Artefakte, messbare Abweichungen vom Original).
- Anwendung von Punktoperationen zur Analyse und Visualisierung von Bildunterschieden.
- Bedeutung von Bibliotheken wie NumPy für die effiziente Verarbeitung von Bilddaten als Arrays.

nzmetriken sowie die Anwendung der erlernten Techniken auf andere Bilder und Problemstellungen im Bereich der Bildverarbeitung.