Max Jappert, Maximilian Barth

1 Seperierbare Filter

1.1 2D Faltung

Wir können den 2D-Kernel Algorithmus folgend grob in Code schreiben:

Bild der Grösse $M \times N$ und Filtermaske der Grösse $k \times k$

```
for (int i = 0; i < M; i++) {                                          //O(n)
   for (int j = 0; j < N; j++) {                                       //O(n)
      // durch jedes Pixel des Bildes iterieren
      for (int p = 0; p < k; p++) {                                    //O(n)
         for (int q = 0; q < k; q++) {                                 //O(n)
            /* auf jedes Pixel muss der Filter angewandt werden, also pro Pixel k * k
                Operationen (Filtergroesse)*/
            applyFilter(i, j, p, q);
         }
      }
   }
}
```

Dabei muss durch jedes Pixel im Originalbild gegangen werden und jeweils $k \cdot k$ Operationen pro Pixel durchgeführt werden (Inhalt des Filter-Kernels). In $\mathcal{O}$-Notation sieht das folgend aus:
$\mathcal{O}(n \cdot n \cdot n \cdot n) = \mathcal{O}(n^4)$
Es handelt sich also um eine quadrierte Laufzeit.
TODO empirische daten

1.2 1D Faltung

Der 1D Algorithmus unterscheidet sich nicht gross von der 2D implementation, er ist jedoch weitaus schneller. Dies kann man im folgenden Pseudo-Code Block und in der darauffolgenden $\mathcal{O}$-Notation sehen:

```
// zuerst die erste Faltung ausfuehren
for (int i = 0; i < M; i++) {                                  //O(n)
   for (int j = 0; j < N; j++) {                               //O(n)
   // durch jedes Pixel des Bildes iterieren
      for (int p = 0; p < k; p++) {                            //O(n)
         // 1D Filter anwenden -> p Operationen
         applyFilter(i, j, p);
      }
   }
}

// dann die zweite Faltung auf das entstandene Bild
for (int i = 0; i < M; i++) {                                  //O(n)
   for (int j = 0; j < N; j++) {                               //O(n)
      // durch jedes Pixel des Bildes iterieren
      for (int p = 0; p < k; p++) {                            //O(n)
         // 1D Filter anwenden -> p Operationen
```

```
        applyFilter(i, j, p);
      }
    }
}
```

Der grosse Unterschied ist, dass im inneren der beiden Loops, die über die einzelnen Pixel des Bildes iterieren, nicht p^2 Operationen ausgeführt werden müssen, sonder nur p (und das zweimal; einmal bei der ersten Faltung und danach in der zweiten). Dies führt jedoch zu einer massiven Verbesserung in der Laufzeit des Algorithmus:

$\mathcal{O}(n \cdot n \cdot n + n \cdot n \cdot n) = \mathcal{O}(2 \cdot n^3) \geq \mathcal{O}(n^3)$

Der Algorithmus ist folglich im Eindimensionalem um eine Grössenordnung schneller als im Zweidimensionalem.

TODO: empirisch