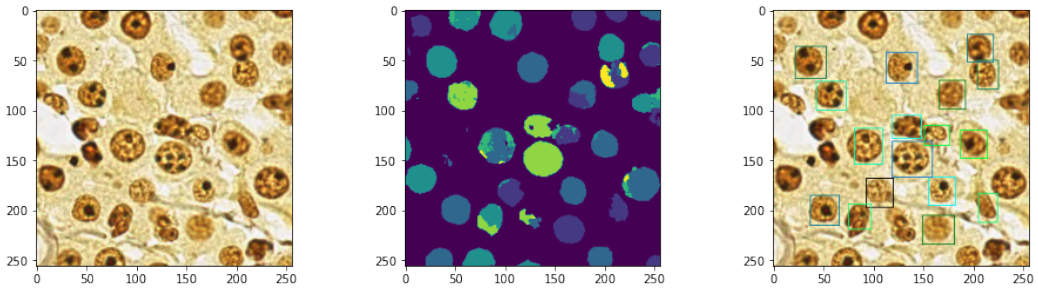
Abgabebericht Computer Vision Challenge

Philip, Weinmann, phw8736@thi.de, Gruppe: Philip Weinmann

8. Juni 2022

1 Einleitung

Ziel der Challenge ist es, anhand eines Datensatzes von drei gelabelten Bildern verschiedene Zellklassen zu erkennen und im Bild zu lokalisieren. Eines der Hauptprobleme hierbei ist der extrem kleine Datensatz. Um diesem Problem entgegen zu wirken, haben wir die Bilder des Datensatzes in *crops* gleicher Größe geschnitten und somit den Datensatz um ein Vielfaches vergrößert. Der Ansatz den wir verfolgt haben ist folgender: mit einem *U-Net* Segmentierungsmodell haben wir die verschiedenen Klassen in den Bildern segmentiert. Diese Segmentierungsmasken haben wir dann verwendet, um *bounding boxes* zu berechnen. (siehe Abbildung 1) Unser Modell hat auf dem Validierungs-Bild eine *mean average precision* von *52 %* erreicht.



(a) Input Image (b) Segmentation Mask (c) Predicted BBoxes

Abbildung 1

2 Aufbau U-Net

Die UNet Architektur wurde 2015 veröffentlicht und ermöglicht eine semantische Segmentierung von Bildern. Das Netzwerk verfolgt einen *fully convolutional* Ansatz, das bedeutet, dass keine *fully connected layers* zum Einsatz kommen.
Die Architektur (siehe Abbildung 3) teilt sich auf in einen *Encoder* und einen *Decoder* Block. Im *Encoder* werden die räumlichen Dimensionen des Input Bildes durch *convolutions* und *max poolings* immer weiter reduziert, wobei die Tiefe, also die Anzahl an Feature Maps, in jedem Schritt verdoppelt wird. Die Ausgabe des Encoder Blocks ist eine *low resolution feature map* in unserer Architektur mit 1024 verschiedenen *feature maps*.
Der *Decoder* verarbeitet die *latent feature map* die aus dem *Encoder* weitergegeben wird. In jedem Schritt wird die *feature map* weiter geupsampelt, bis man im letzten Schritt wieder die ursprünglichen räumlichen Dimensionen des Input Bildes mit einer Tiefe von der Anzahl an vorhandenen Klassen hat. Zusätzlich zur Encoder-Decoder-Struktur sind auf jeder Schicht *skip-connections* vom *Encoder* zum *Decoder* eingebaut, die schnelleres Training und eine Verringerung von *vanishing gradients* schaffen.

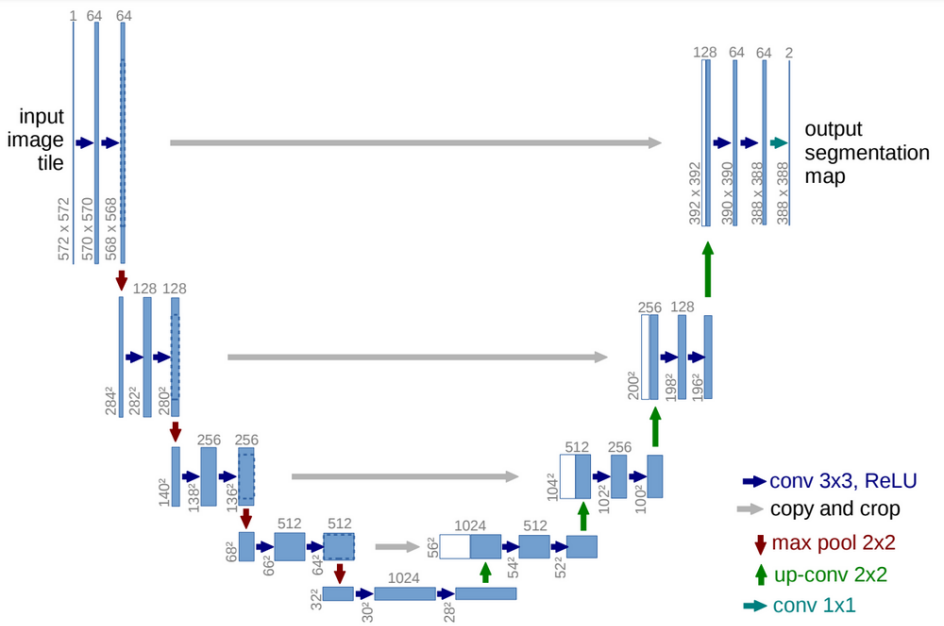


Abbildung 2: Architektur U-Net

3 Vorgehensweise bei der Implementierung

Bei der Implementierung unseres Object Detectors können grundsätzlich drei Hauptphasen unterteilt werden. Die nachfolgend genauer beschrieben werden.

3.1 Dataloader und Preprocessing

Der Dataloader wurde aus dem Baseline Modell üernommen und für die Anforderungen unseres Segmentierungsmodells angepasst. Zusätzlich zu den Bounding Boxes und den Images haben wir mithilfe der Methode *cv2.fillPoly()*, die die x- und y-Koordinaten der Bounding Boxes entgegeben nimmt, für alle Bilder eine Segmentation Mask erstellt. Diese dient zum Trainieren des U-Net im nächsten Schritt.

3.2 Segmentation Model

Bei der Architektur des U-Net haben wir uns stark an dem ursprünglichen Paper aus 2015 orientiert.Die Räumlichen Dimensionen dabei mit einem *Stride* von 2 und *Padding* von 1 immer weiter verringert, während die Anzahl der *feature channels* in jedem Schritt bis auf 1024 verdoppelt wird. Nach dem Upsampling im Decoder erhalten wir einen Output des Modells mit folgenden Dimensionen: W x H x 7. Der *class-channel* von 7 kommt daher, dass wir bei der Segmentierung zusätzlich zu den 6 Klassen noch die Hintergrundklasse 0 haben.
Insbesondere an der Anzahl an finalen *feature channels* haben wir verschiedene Parameter getestet. Das Modell liefert für den Parameterwert 1024 die besten Ergebnisse.

3.2.1 Training des Modells

Wir haben hier mit verschiedenen *loss functions* experimentiert. Default für eine Segmentierung ist eine *cross entropy loss function*. Da die Klassen allerdings nicht balanced sind, haben wir uns für eine *focal loss function* entschieden, die die unterschiedlichen Klassenverteilungen ausgleicht.
Wir haben außerdem versucht, das Modell durch den Einsatz von einer adaptiven Lernrate zu verbessern. Dies hat allerdings keine Verbesserung gebracht und wurde deshalb wieder entfernt.

3.3 Berechnung der Bounding Boxes

Die Bounding Boxes der wurden mit folgender Funktion (siehe Abbildung 3) erzeugt.

```python
def cv_contours(img):
    cnts = cv2.findContours(img.type(torch.uint8).numpy(), cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
    cnts = imutils.grab_contours(cnts)
    if len(cnts) == 0:
        return []
    cnts = contours.sort_contours(cnts)[0]
    # # loop over the contours
    bboxes = []
    for (i, c) in enumerate(cnts):
        (x, y, w, h) = cv2.boundingRect(c)
        if w * h >= 500:
            bboxes.append([x, y, x+w, y+h])
    bboxes = remove_border_bboxes(bboxes)
    bboxes = torch.Tensor(bboxes)
    # z = draw_bounding_boxes((images[0]*255).type(torch.uint8), bboxes)
    return bboxes
```

Abbildung 3: Funktion zur Bounding Box Berechnung auf einer Segmentation Mask

Der Datenfluss war dabei folgender:

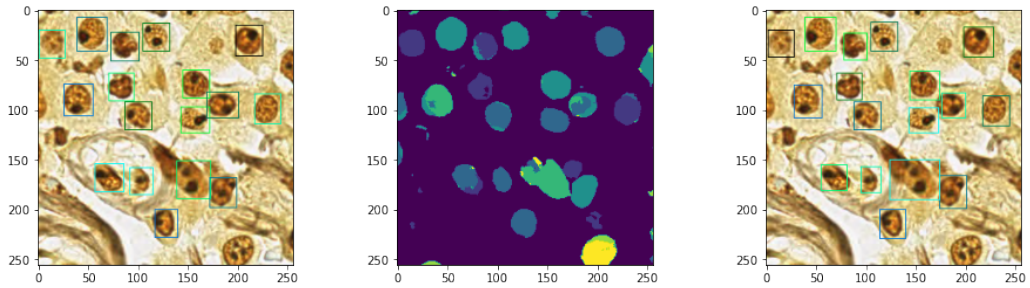
1. Output aus Segmentation Model: Image W x H x 7
2. Iteration durch die Channels 1-7 des Images (Channel 0 = Hintergrund; wird nicht berücksichtigt.):
   (a) berechne alle Bounding Boxes des Channels mit der Funktion *cv2.findContours()*
   (b) verwerfe alle Bounding Boxes mit Flächeninhalt kleiner als 500
   (c) verwerfe alle Bounding Boxes, die Kontakt mit den Randpixeln des Bildes haben
   (d) gebe die restlichen Bounding Boxes zurück
   (e) gebe für jede Bounding Box die Klasse (also den aktuellen Channel) zurück
3. vergebe jeder Bounding Box den Score 1

In den ersten Versuchen, haben wir die Bounding Boxes nicht pro Channel, sondern auf der vollständigen Segmentierungsmaske, die unser Modell zurückgibt, berechnet. Da hier häufig Zellen verschiedener Klassen überlappten, wurde versucht mit Hilfe von *Erosion* und *Dilation* Filtern für eine Trennung dieser Zellen zu sorgen. Dies hat sehr gut funktioniert, war für unseren finalen Ansatz dann allerdings nicht nötig.

4 Fazit und Ausblick

Mit dem U-Net Modell haben wir einen Segmentierungsansatz gewählt, der im Vergleich zu aktuelleren Modellen weniger komplex ist. Wir konnten allerdings zeigen, dass auch mit geringerer Komplexität definitiv vernünftige Ergebnisse erzielt werden können. Auf den Validation Daten konnten wir einen mAP von ca. 51% erreichen, was unter Betracht der wenigen Trainingsslides und der Trainingszeit des Modells sicherlich kompetitiv. Insbesondere beim Vergleich der vorhergesagten Bounding Boxes mit den *ground truth* Bounding Boxes wird erkenntlich, dass das Modell die meisten vorhandenen Objekte erkennt. (siehe Abbildung 4)

Insbesonder unter Anbetracht der dargestellten Bounding Boxes wird die Schwäche unseres Ansatzes deutlich: Die gut erkannten Objekte werden falsch klassifiziert. Der nächste Ansatzpunkt um das Modell zu verbessern ist, herauszufinden woran die Fehlklassifikationen liegen und einen Lösungsansatz zur Verbesserung zu erarbeiten. Beispielsweise könnte ein *Image Classification Model* trainiert werden. Diesem Modell gibt man die Bildausschnitte innerhalb der *ground truth* Bounding Boxes in einer normalisierten Form. Wenn das Modell eine gute Güte aufweist, könnte man damit die vorhergesagten Bounding Boxes damit klassifizieren.



(a) Ground Truth BBoxes (b) Segmentation Mask (c) Predicted BBoxes

Abbildung 4