TF & KERAS. РОБОТА із ЗОБРАЖЕННЯМИ

Файл: TF_KERAS_Image_05_001

Нейронна мережа YOLO. Інференс YOLOv3. Приклад детектування та локалізації об'єктів

Базується на матеріалі наступних публікацій

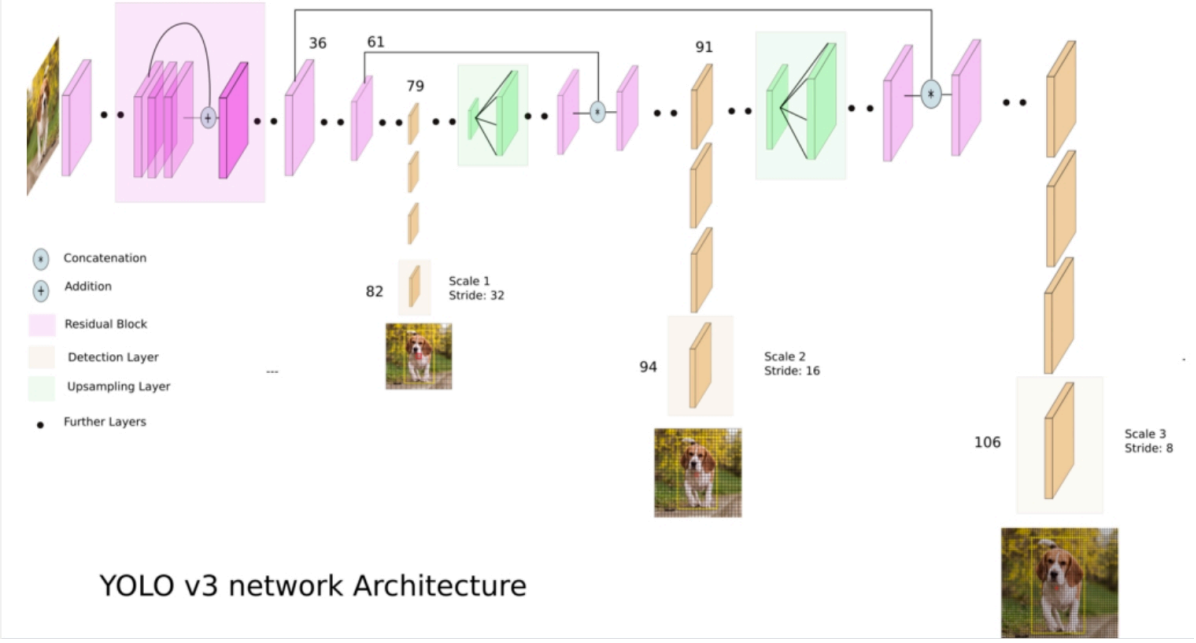
- YOLOv3: An Incremental Improvement https://pjreddie.com/media/files/papers/YOLOv3.pdf
- OpenCV YOLO - object detection https://opencv-tutorial.readthedocs.io/en/latest/yolo/yolo.html
- YOLOv3 – Deep Learning Based Object Detection https://learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python-c/

Загальні відомості YOLO / YOLOv3

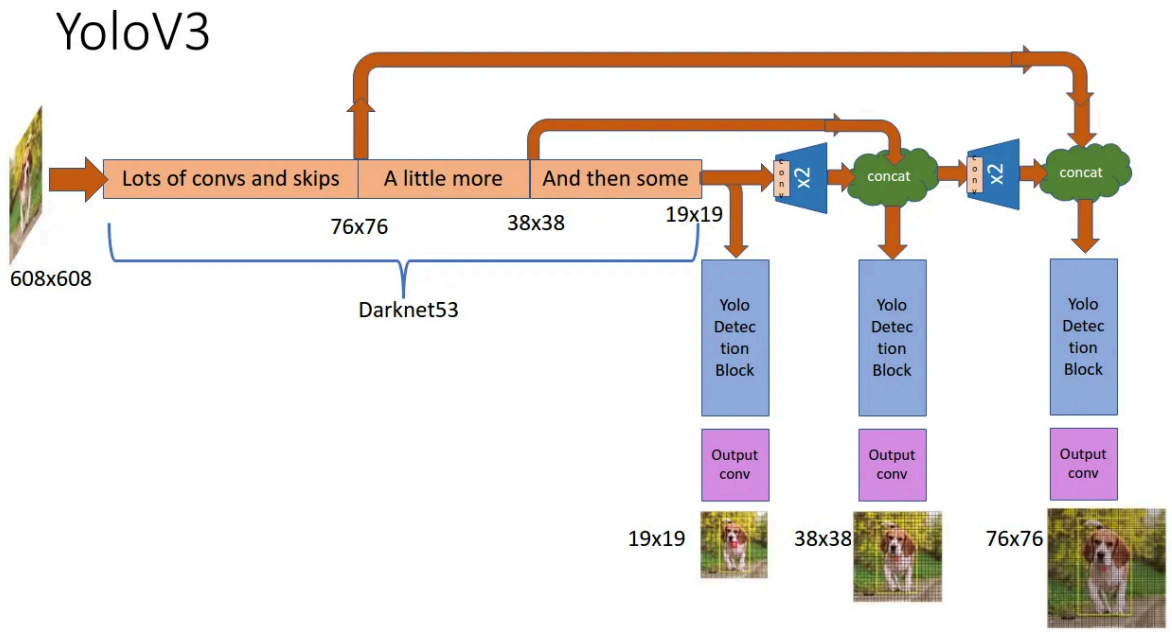
YOLO або You Only Look Once - це популярна на даний момент архітектура **CNN**, яка використовується для розпізнавання багатьох об'єктів на зображенні. Головна особливість цієї архітектури в порівнянні з іншими полягає в тому, що більшість систем застосовують **CNN** кілька разів до різних регіонів зображення, **YOLO CNN** застосовується один раз до всього зображення відразу. Мережа ділить зображення на своєрідну сітку і передбачає обмежувальні рамки (bounding boxes) і ймовірність того, що там є об'єкт, що шукається, для кожної ділянки.

Плюси даного підходу полягає в тому, що мережа дивиться на все зображення одразу і враховує контекст при детектуванні та розпізнаванні об'єкта. Так само **YOLO** в 1000 разів швидше за **R-CNN** і близько 100x швидше ніж **Fast R-CNN**.

YOLOv3_ – це вдосконалена версія архітектури YOLO.__ Вона складається зі 106-ти згорткових шарів і краще детектує невеликі об'єкти в порівнянні з її попередницею



Основна особливість **YOLOv3** полягає в тому, що на виході є ТРИ шари, кожен з яких розрахований на виявлення об'єктів різного розміру. Нижче наведено схематичне зображення архітектури **YOLOv 3** .



Нейромережа YOLOv3 має 3 вихідних шари для обробки зображення із різною деталізацією:

- 507 (13 x 13 x 3) для великих об'єктів
- 2028 (26 x 26 x 3) для середніх об'єктів
- 8112 (52 x 52 x 3) для малих об'єктів

Кожен вихід є вектор довжиною 85

- 4 координати рамки, що обмежує (centerx, centery, width, height)
- 1 довіра рамки (box confidence)
- 80 значення довіри рамки для кожного класу

Основна особливість архітектури YOLOv3

Похибка (loss) YOLO для кожної прогнозної рамки (box prediction) складається з таких складових:

1. **Втрата координат, Coordinate loss** — через прогнохної рамки, що не повністю покриває об'єкт,
2. **Втрата об'єктності, Objectness loss** — через неправильне передбачення IoU рамки-об'єкта,
3. **Втрата класифікації, Classification loss** — через відхилення від передбачення «1» для правильних класів і «0» для всіх інших класів для об'єкта в цьому полі.
4. **Особлива втрата**, буде пояснено нижче.

Функція втрат

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(x_i - x_i'\right)^2 + \left(y_i - y_i'\right)^2 \right]$$
$$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{w_i'}\right)^2 + \left(\sqrt{h_i} - \sqrt{h_i'}\right)^2 \right]$$
$$+\sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left(C_i - C_i'\right)^2$$
$$+\sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left(C_i - C_i'\right)^2$$
$$+\sum_{i=0}^{S^2} 1_{i}^{noobj} \sum_{cc classes} \left(p_i(c) - p_i'(c)\right)^2$$

Лямбда - це коефіцієнти втрат. Перші 3 рядки – це втрати, спричинені «найкращими рамками» (рамками , які найкраще захоплюють GT-об'єкти в кожній просторовій комірці), а 4-й – через коробки, які не захопили об'єкти. У YOLO V2 і V3 прямі передбачення ширини та висоти та квадратний корінь були замінені прогнозом залишкового масштабу, щоб зробити аргумент втрати пропорційним відносної, а не абсолютної помилки масштабу.

Якість передбачення прямокутника вимірюється його IoU (Intersection over Union) з об'єктом, який він намагається передбачити (точніше — з його основним прямокутником правди). Значення IoU варіюються від 0 (прямокутник повністю пропускає об'єкт) до 1,0 (ідеальна посадка).

Для кожної просторової комірки, для кожного передбачення блоку в центрі цієї комірки функція втрат знаходить блок із найкращим IoU з об'єктом, центрованим у цій комірці. Цей механізм розрізнення найкращих прямокутників від усіх інших прямокутників лежить в основі функции втрат **YOLO**.

Найкращі прямокутники та лише вони зазнають втрати координат (через недостатню відповідність об'єкту) та втрати класифікації (через помилки класифікації). Це підштовхує параметри мережі, пов'язані з цими прямокутниками, щоб покращити масштаб і розташування прямокутника, а також класифікацію. Ці прямокутники також призводять до втрати довіри, що буде пояснено нижче. Усі інші прямокутники лише втрачають довіру.

Втрата об'єктивності — знання своєї цінності

З кожним передбаченням прямокутника пов'язане передбачення під назвою «об'єктність». Цей прогноз насправді є прогнозом IoU, тобто наскільки добре мережа вважає, що прямокутник закриває об'єкт. Термін втрати об'єктності вчить мережу передбачати правильний IoU, тоді як втрата координат вчить мережу передбачати кращий прямокутник (що зрештою підштовхує IoU до 1,0). Усі передбачення прямокутників сприяють втраті об'єктності, але лише найкраще підібрані прямокутники в кожній просторовій комірці також сприяють втраті координат і класифікації.

Під час інференсу зазвичай маємо кілька рамок (прямокутників) із різним покриттям для кожного об'єкта. Бажано, щоб алгоритм постобробки вибрав рамку, яка охоплює об'єкт найбільш точним чином. Ми також хочемо вибрати поле, яке дає правильний прогноз класу для об'єкта. Як алгоритм може знати, який прямокутник вибрати? **YOLO** використовує **NMS**

NMS — Non-Maximum Suppression - Немаксимальне придушення - подавлення

NMS — це жадібний алгоритм, який проходить цикл по всіх класах і для кожного класу перевіряє наявність перекривань (IoU) між усіма обмежувальними рамками. Якщо IoU між двома прямокутниками одного класу перевищує певний поріг (зазвичай 0,7), алгоритм робить висновок, що вони стосуються одного об'єкта, і відкидає прямокутники із нижчим показником достовірності - **lower confidence score** (який є добутком показника об'єктності та ймовірність умовного класу).

По-перше, об'єктність говорить нам, наскільки хорошим є покриття — тому прямокутники з дуже малою об'єктністю (<0,005) відкидаються і навіть не потрапляють до блоку NMS. Це допомагає видалити близько 90% прямокутників, які є лише артефактами архітектури, а не реальними виявленнями .

По-друге, NMS виконується для кожного класу окремо, тому оцінка класу масштабується за об'єктністю прямокутника для значущого порівняння. Якщо маємо два прямокутника з високим перекриттям, перший з об'єктністю 0,9 та імовірністю особи 0,8 (зважена оцінка 0,72), а другу з об'єктністю 0,5 та ймовірністю особи 0,3 (зважена оцінка 0,15), перша коробка збережеться, а друга впаде в NMS, оскільки об'єктність першого прямокутника зробила його більш надійним.

Цикл виконується над всіма класами, оскільки необхідно зберегти можливість виявляти класи, які не мають найвищих балів, на випадок, якщо копія класу з найвищим балом буде відкинута через її велике перекриття з іншим прямокутником.

Приклад інференсу YOLOv3.

Використовується навчана YOLOv3 модель з бібліотеки OpenCV

Імпорт бібліотек

```
import cv2 as cv
import numpy as np
```

Шлях до зображення, що обробляється

```
path = '..../'
img = cv.imread(path + 'images/food.jpg')
```

Відображення необробленого зображення

```
while True:
    cv.imshow('window',  img)
    if cv.waitKey(1) == ord('q'):
        break
cv.destroyAllWindows()
```

Завантаження імен класів та визначення випадкових кольорів

```
classes = open(path + 'coco.names').read().strip().split('\n')
np.random.seed(42)
colors = np.random.randint(0, 255, size=(len(classes), 3), dtype='uint8')
```

Завантаження конфігурації моделі та її ваг та визначення обєкта 'net' (модель YOLO3)

```
net = cv.dnn.readNetFromDarknet(path + 'yolov3.cfg',
                                path + 'yolov3.weights')
net.setPreferableBackend(cv.dnn.DNN_BACKEND_OPENCV)
net.setPreferableTarget(cv.dnn.DNN_TARGET_CPU)
```

Функція отримення імен вихідних YOLO шарів

```
# def getOutputsNames(net):
    # Get the names of all the layers in the network
    layersNames = net.getLayerNames()
    # Get the names of the output layers,
    # i.e. the layers with unconnected outputs
    return [layersNames[i - 1] for i in net.getUnconnectedOutLayers()]#
```

Створення блобу із похідного зображення

```
blob = cv.dnn.blobFromImage(img, 1/255.0, (416, 416), swapRB=True, crop=False)
r = blob[0, 0, :, :]
```

Відображення блобу

```
text = f'Blob shape={blob.shape}'

while True:
    cv.imshow('blob', r)
    cv.displayOverlay('blob', text)
    if cv.waitKey(1) == ord('q'):
        break
cv.destroyAllWindows()
```

Блоб на вхід YOLO моделі

```
net.setInput(blob)
```

Виходи YOLO моделі

```
outputs = net.forward(getOutputsNames(net))
```

Перевірка виходів YOLO шарів

```
print('Net Outputs shape', np.shape(outputs))
print('Outputs YOLO layer 1', np.shape(outputs[0]))
print('Outputs YOLO layer 2', np.shape(outputs[1]))
print('Outputs YOLO layer 3', np.shape(outputs[2]))
```

Визначення списків

```
boxes = []
confidences = []
classIDs = []
h, w = img.shape[:2]
```

Параметри відбору рамок (боксів)

```
confThreshold = 0.5  # Поріг довіри
nmsThreshold = 0.4   # Поріг придушення
```

Формування обмежувальних рамок (боксів)

```
# Цикл з виходів незв'язаних шарів
for output in outputs:
    for detection in output:
        # Цикл з по елементам виходів шару
        scores = detection[5:]
        classID = np.argmax(scores)
        confidence = scores[classID]
        if confidence > confThreshold:
            box = detection[:4] * np.array([w, h, w, h])
            (centerX, centerY, width, height) = box.astype("int")
            x = imagescale*int(centerX - (width / 2))
            y = imagescale*int(centerY - (height / 2))
            box = [x, y, imagescale*int(width), imagescale*int(height)]
            boxes.append(box)
            confidences.append(float(confidence))
            classIDs.append(classID)
```

Відбір обмежувальних рамок

Non Maximum Suppression

```
indices = cv.dnn.NMSBoxes(boxes, confidences, confThreshold, nmsThreshold)
print('Boxes num', len(indices),  'Boxes indices', indices)
```

Формування зображень рамок та їх накладання на зображення

```
if len(indices) > 0:
    for i in indices.flatten():
        (x, y) = (boxes[i][0], boxes[i][1])
        (w, h) = (boxes[i][2], boxes[i][3])
        color = [int(c) for c in colors[classIDs[i]]]
        cv.rectangle(imgresized, (x, y), (x + w, y + h), color, 2)
        print('Object Class "', classes[classIDs[i]], '"  '
              'Confidence = ', confidences[i])
        text = "{}: {:.4f}".format(classes[classIDs[i]], confidences[i])
        cv.putText(imgresized, text, (x, y - 5),
                   cv.FONT_HERSHEY_SIMPLEX, 0.5, color, 1)
```

Відображення зображення із обмежувальними рамками

```
while True:
    cv.imshow('window', img)
    if cv.waitKey(1) == ord('q'):
        break
```

Приклади результатів

Зображення 'food.jpg'

Зображення 'vine.jpg'


bottle: 0.9541
wine glass: 0.9876
wine glass: 0.9555