Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

TDK DOLGOZAT

Bereczki Márk

OBJEKTUM LOKALIZÁLÁS KÉPEKEN MÉLY NEURÁLIS HÁLÓVAL

KONZULENS

Dr. Szűcs Gábor

BUDAPEST, 2018

Tartalomjegyzék

Összefoglaló

A 21. századra az egyik legértékesebb erőforrásunk az adat lett. Az élet folyamatos gyorsulásának, az egyre több eladott technikai eszköznek, az ezekre épülő szolgáltatások folyamatos bővülésének és az új vállalati és felhasználói szokások kialakulásának következtében egyre több adat keletkezik a világban. Ezek kezelése sok technológiai, etikai, jogi és gazdasági kérdést vet fel, amelyek közül számos még megválaszolásra vár, és a következő évek várhatóan újabb kihívásokat állítanak elénk. Mérnök informatikus hallgatóként dolgozatomban a téma egyik technológiai oldalát, az adatelemzést, azon belül pedig a képi adatok elemzését vizsgáltam.

A feladatom megvalósítása során egy olyan neurális háló alapon működő rendszert terveztem és implementáltam, amely nagyszámú, szemantikus információkkal felcímkézett kép - mint tanuló állomány - segítségével ismeretlen fényképeken felismeri, és lokalizálja a különböző objektumokat.

A rendszer megvalósításához a YOLO (You Only Look Once) objektum lokalizációs rendszerből indultam ki. Ez a C-ben és CUDA-ban íródott Darknet nevű neurális háló keretrendszer része, és jelenleg az egyik legpontosabb és leggyorsabb megoldást adja az objektumok képeken és videókon történő lokalizációjára. Gyorsaságát annak köszönheti, hogy a klasszikus megoldásokkal (pl. R-CNN) szemben úgynevezett single shot detection-t használ. Ez azt jelenti, hogy nem kell egy külön neurális hálót használnunk arra, hogy megtaláljuk a potenciális objektumok befoglaló téglalapjait, majd egy másikat arra, hogy minden egyes téglalaphoz tartozó képrészletet végigfuttatva a hálón megmondjuk, hogy az milyen objektumot tartalmaz. Ezzel szemben elég egy neurális hálót használni, amin csak egyszer kell végigfuttatni a képeket.

A dolgozatomban bemutatom a képeken történő objektum felismerés és lokalizálás szakirodalmát, majd az általam elkészített, YOLO alapú rendszer implementációját, betanítását és finomhangolását. Emellett ismertetetem mérési eredményeimet: a létrehozott rendszeren megmértem az objektum felismerés, illetve lokalizáció pontosságát különböző képhalmazokon.

Abstract

In the 21st century one of our most valuable resource is data. The continuous acceleration of life, increases in technical devices selling, services laying on these gadgets and new corporate and user behaviors result in extended data generation. Handling these data raises multiple questions in the field of technology, ethics, law and economics. Some of these are still unanswered and in the next years presumably many more challenges are about to come. As a computer science student, my aim in this work was to evaluate one field of present technical challenges, namely image analysis.

In the framework of the project, I designed and implemented a system based on a neural network which is able to recognize and localize different objects on new images using a dataset containing photos and labels with semantic informations.

The system I implemented is based on an object localization system called YOLO (You Only Look Once). It is part of the Darknet neural network framework which is written in C and CUDA. Currently YOLO gives one of the most accurate and fastest solution for object localization on images and videos. Its high speed is due to the fact that contrary to classical solutions like R-CNN, it uses single shot detection. It means that we don’t have to use a separate neural network to find the bounding boxes of potential objects and then another one to guess what kind of object each of these bounding boxes contain. On the contrary, it’s enough to use only one neural network and run the images only once through it.

After an overview of current literature on object recognition and localization, the paper presents the implementation, training and fine-tuning of a self-made system based on YOLO. Besides that, I present my measurements of the accuracy of object recognition and localization on different image datasets.

1 Bevezetés

Kutatásom során objektumok képeken történő lokalizációjával foglalkoztam. Munkámat irodalomkutatással kezdtem, megvizsgáltam, hogy milyen megoldások születtek a problémára. Az algoritmusok közül végül a YOLO-ra [1] [2] [3] esett a választásom, ennek mindhárom verziójának működését áttekintettem. Erről a 2. fejezetben írok bővebben.

Feladatomként autók képeken történő lokalizációját választottam. A megoldás alapjául a YOLO verziói közül a legfrissebbet, a dokumentum írásának időpontjában mindössze fél éves YOLO v3-at [3] választottam. Kétféle modellt készítettem el, az egyik az autókat általánosan képes lokalizálni, a másik pedig a lokalizáció mellett az autók márkáját is megbecsüli. A 3. fejezetet az általam elkészített rendszer részletes bemutatásának szentelem. Kitérek az implementációt megelőző szükséges és/vagy praktikus lépésekre, bemutatom az adathalmaz előkészítésének módját, leírom, hogy hogyan jutunk el a rendszer tanításáig, a tanítás bemutatását követően pedig bemutatom a validáció módját, majd összefoglalom mérési eredményeimet.

Lezárásként pedig kitérek a lehetséges továbbhaladási irányokra, és bemutatom a rendszerben rejlő felhasználási lehetőségeket.

2 „State of the art” az objektum lokalizációban

Az objektum lokalizáció egy olyan probléma, amely már régóta foglalkoztatja az kutatókat, mérnököket. Több megoldást is született a megoldására, amelyek közül először két klasszikus megközelítést mutatok be röviden, majd pedig a YOLO rendszer három verzióját [1] [2] [3] ismertetem részletesen.

2.1 Klasszikus megoldások

Két széles körben használt megoldás az objektum lokalizáció problémájára az úgynevezett „sliding window” és a „region proposal” technika.

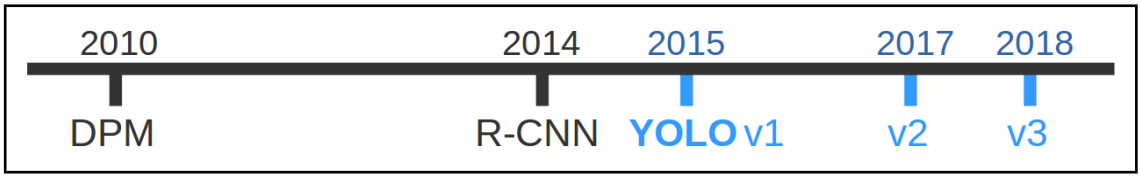
A „sliding window” módszer használata során egy csúszóablakot futtatunk végig a képen, és az így kapott képrészleteken hajtunk végre egy klasszifikációs algoritmust. Ezáltal megtudjuk, hogy az adott ablakon belül található-e valamilyen objektum, és ha igen, milyen típusú. Ezt a módszert használja a DPM (Deformable Part Models) [4] rendszer.

A „region proposal” technika alkalmazása során először meghatározott számú olyan területet keresünk a képen, amely esetlegesen valamilyen objektumot tartalmazhat. Ezt követően csak ezeken a régiókon futtatunk klasszifikációt, így a „sliding window” módszert használó megoldásokhoz képest itt kevesebb osztályozó műveletre van szükség. Ezt a technikát alkalmazza az R-CNN [5] rendszer.

Ezek a megoldások jól működnek, azonban lassúak, mivel mindkét esetben több lépésben hajtjuk végre az objektumok lokalizálását a képeken. A több lépés nagyobb komplexitást is jelent: az R-CNN egyes részeit külön kell tanítani, ami nehezíti a rendszer optimumának megtalálását. Bár mindkét megoldásnak születtek későbbi új, gyorsabb változatai [6] [7] [8], az igazi szemléletváltást a YOLO [1] hozta el.

2.2 YOLO

A YOLO objektum lokalizációs rendszer első verziója 2015-ben készült el, amit két újabb változat követett 2017-ban és 2018-ban. A következőkben a különböző verziók működését mutatom be, kitérve a köztük lévő különbségekre.



1. ábra: Objektum lokalizációs rendszerek idővonala

2.2.1 YOLO v1

A YOLO [1] létrehozása mögött egy fontos igény állt: bizonyos feladatok gyors lokalizációt követelnek meg. Vegyük például a videók valós idejű feldolgozását. Ez másodpercenként minimum 30 kép elemzését jelenti, amit megfelelő videokártya használatával a YOLO képes teljesíteni.

Az algoritmus az objektum lokalizáció folyamatát nem bontja szét részekre, hanem azt egy regressziós problémaként kezeli, és egy olyan konvolúciós neurális hálót nyújt megoldásként, amelynek kimenetén egyszerre kapjuk meg az objektumokat határoló téglalapok helyét, és azt, hogy az azokban található objektumok milyen valószínűséggel tartoznak egy adott osztályhoz. Így, ha egy képet végigfuttatunk ezen a neurális hálón, megkapjuk a keresett objektumok helyét. A rendszernek elég egyszer “ránéznie” a képre, és el tudja végezni a lokalizációs feladatot. A technológia innen is kapta a nevét: *You Only Look Once*. Az a tény, hogy a YOLO egy lépésben dolgozza fel a teljes képet, lehetővé teszi, hogy a rendszer a kontextusra is tá tudjon tanulni, mivel az egyes döntéseket a kép egésze határozza meg.

A következőkben a rendszer részleteit mutatom be. Először a bemeneti képet megadott méretűre skálázzuk, majd egy négyzetrács mentén $S \times S$ egyenlő nagyságú négyzetre osztjuk. Mindegyik ilyen négyzet azon objektumok lokalizációjáért felelős, melyek középpontját az adott négyzet tartalmazza. Az egyes négyzeteket celláknak hívom a következőkben. Minden cellához B darab olyan határoló téglalapot keresünk, amely potenciálisan egy objektumot foglalhat magába. Az egyszerűség kedvéért ezeket a határoló téglalapokat dobozoknak hívom a következőkben. Minden egyes dobozhoz 5

értéket becslünk meg. Ebből 4 a doboz pozícióját és méretét határozza meg: x és y a doboz középpontjának koordinátái a cellához viszonyítva, w és h pedig a doboz szélessége és magassága. Az ötödik egy 0 és 1 közé eső pontszám (*Conf*), amely két tényezőtől függ: hogy mennyire biztos a háló abban, hogy a doboz tartalmaz valamilyen objektumot, illetve hogy mennyire gondolja pontosnak a doboz dimenzióit. A Conf értéket a következő képlet adja meg a tanítás során:

$$\boxed{Conf = P(objektum) \times IOU}$$

1. képlet: Conf pontszám

A $P(objektum)$ érték annak a valószínűsége, hogy a doboz tartalmaz egy objektumot, az IOU érték pedig a doboz és a tanító adathalmazban található hozzátartozó doboz IOU értéke, ami a dobozok metszetének és uniójának a hányadosa:

$$\boxed{IOU = \frac{doboz_1 \cap doboz_2}{doboz_1 \cup doboz_2}}$$

2. képlet: Intersection over Union

A tanító halmazból az a doboz fog a becsült dobozhoz tartozni, amellyel az a legnagyobb IOU értéket adja.

Ezen kívül mindegyik cella további C darab feltételes valószínűség értéket becsül, ahol C az osztályok száma. Tehát minden cellán belül minden osztályhoz tartozik egy feltételes valószínűség, mégpedig annak a valószínűsége, hogy a cella tartalmaz egy adott típusú objektumot, feltételezve, hogy a cella tartalmaz bármilyen objektumot. Ennek jelölése: $P(osztály_i | objektum)$.

Összefoglalva, a becsült értékeink és azoknak számossága:

$$\boxed{\begin{array}{c} x, y, w, h, Conf : S \times S \times B \\ P(osztály_i | objektum) : S \times S \times C \end{array}}$$

A becsült értékeket egy $S \times S \times (B \times 5 + C)$ méretű tenzor tartalmazza, amely a neurális háló kimenetét adja. Az objektum lokalizációt így egy regressziós feladatra vezetjük vissza, amelynél $S \times S \times (B \times 5 + C)$ darab számot kell megbecsülnünk

Kiértékelésnél mindegyik dobozra és azon belül minden osztályra meghatározható egy végső pontszám, amely két tényezőtől függ: egyrészt annak a valószínűségétől, hogy a doboz egy, az adott osztályhoz tartozó objektumot tartalmaz, másrészt pedig a becsült doboz dimenzióinak pontosságától. Ezt a végső pontszámot egy adott dobozra i osztály esetén a következő képlet adja meg:

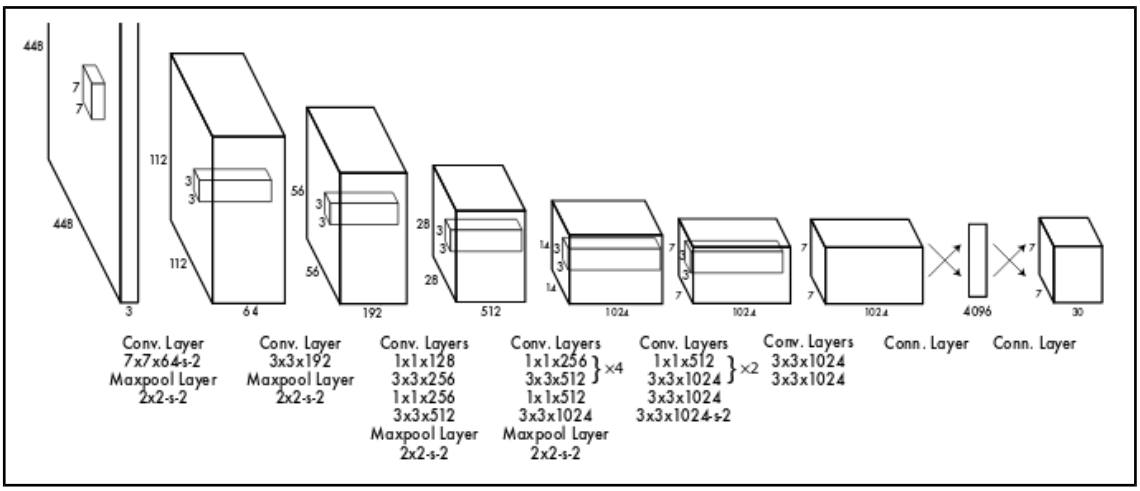
$$pontszám_i = P(osztály_i) \times IOU = P(osztály_i | objektum) \times P(objektum) \times IOU =$$
$$= P(osztály_i | objektum) \times Conf$$

3. képlet Doboz- és osztályspecifikus végső pontszám meghatározása

A becsült értékeink a következők: $x, y, w, h, Conf, P(osztály_i | objektum)$, tehát a becsült értékekből végül megkaphatjuk az egyes dobozok dimenzióin kívül a hozzájuk tartozó $pontszám_i$ értékeket is.

A Conf és a $P(osztály_i | objektum)$ számok a $[0,1]$ intervallumba esnek. Hogy az összes becsülendő értékünk egy tartományhoz tartozzon, az x, y, w, h mezőket normalizálnunk kell. Az x és y értékeket az adott cella elhelyezkedésétől való eltolással adjuk meg, a w és a h értékek pedig azt fejezik ki, hogy a dobozok szélessége és magassága hány százaléka a kép szélességének és magasságának. Ezáltal az összes becsülendő értékünk a $[0,1]$ intervallumba esik.

A YOLO egy konvolúciós neurális hálót használ, melynek felépítése Pascal VOC adathalmaz használata esetén $(C = 20)$ a következő:



2. ábra: A YOLO v1 konvolúciós hálója [1]

A háló 24 konvolúciós és 2 „fully connected” réteget tartalmaz. A „fully connected” réteg egy olyan réteg, amelynek minden neuronja összeköttetésben áll az azt megelőző réteg összes neuronjával. Érdekesség, hogy a háló 1×1 méretű konvolúciós filtereket is magába foglal. Ezeknek célja a „feature map”-ek számának csökkentése. Láthatjuk, hogy a kimeneti tenzor mérete $S\times S\times(B\times5+C)=7\times7\times(2\times5+20)=7\times7\times30$. A YOLO hálójának készült egy kisebb verziója is, ez a Fast YOLO, amely mindössze 9 konvolúciós réteget tartalmaz.

A tanítást egy előtanítási fázis előzi meg. A rendszer első 20 konvolúciós rétegét az ImageNet 1000-class versenyhez tartozó adathalmazon [9] tanítják be, amely egy klasszifikációs adathalmaz. Ezáltal inicializálásra kerül az első 20 konvolúciós réteg, ami a teljesítmény javítását szolgálja. Ezt követően az előtanított rétegekhez hozzácsatolják a maradék 4 konvolúciós és 2 „fully connected” réteget, és a háló bemeneti felbontásának megnövelése után tovább folytatják a tanítást, immáron egy lokalizációra felcímkézett adathalmazon.

Az utolsó réteg lineáris aktivációs függvényt használ, a többi pedig egy „leaky” ReLU függvényt. A „leaky” ReLU függvény abban különbözik a klasszikus ReLU függvénytől, hogy negatív x értékek esetén nem 0-t vesz fel, hanem az x-nek és egy 0 és 1 közötti számnak (YOLO esetében a 0,1-nek) a szorzatát. A tanítás során egy „sum-squared error” értéket optimalizálunk, amelyet a következő módon számolunk ki:

$$
\begin{aligned}
loss = & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathrm{I}_{obj_{ij}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] + \\
& + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathrm{I}_{obj_{ij}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right] + \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathrm{I}_{obj_{ij}} \left(Conf_i - \hat{Conf}_i \right)^2 + \\
& + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathrm{I}_{noobj_{ij}} \left(Conf_i - \hat{Conf}_i \right)^2 + \\
& + \sum_{i=0}^{S^2} \mathrm{I}_{obj_{ij}} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}
$$

4. képlet: Loss függvény

A λ_{coord} és λ_{noobj} értékek feladata az egyes komponensek súlyozása. $\lambda_{coord} = 5$, míg $\lambda_{noobj} = 0.5$, tehát jobban büntetjük a dobozok dimenzióinak becslésekor keletkezett hibákat, és kisebb mértékben vesszük figyelembe azon dobozok Conf értékének becslése során felmerült eltéréseket, amelyek nem tartalmaznak objektumot. $I_{obj_{i,j}}$ értéke 1, ha az i. cella j. doboza tartalmaz objektumot, különben pedig 0. $I_{noobj_{i,j}}$ értéke pedig akkor 1, ha az i. cella j. doboza nem tartalmaz objektumot, különben pedig 0. A képletben található $p_i(c)$ érték a c osztályhoz tartozó feltételes valószínűség értéket jelöli. A „^” jelöléssel ellátott értékek a becsült értékeket fejezik ki. A Látható, hogy a dobozok szélességének és magasságának becsléséhez tartozó hiba számításakor az értékek négyzetgyökét vesszük. Ezzel csökkentjük azt a hatást, hogy a nagyobb dobozoknál keletkező kisebb hibák ugyanolyan súllyal számítanak bele a veszteségfüggvény értékébe, mint a kisebb dobozoknál keletkező ugyanakkora hibák.

A túltanulás elkerülésére „dropout”-ot használunk. Ennek során a tanítás közben a neuronok egy meghatározott nagyságú, véletlenszerű módon kiválasztott halmazát nem vesszük figyelembe. Ennek a halmaznak a tartalmát tanítás során többször megváltoztatjuk. Emellett a tanuló adathalmaz bővítésével is segítjük a rendszerünket, hogy nagyobb legyen az általánosító képessége. Az adathalmaz kiterjesztését a képek

méretének változtatásával, eltolásával, illetve a képekhez tartozó fényerősség és a szaturáció értékek módosításával érjük el.

Előfordul, hogy egy objektumot több lehetséges doboz lokalizál. A duplikációkat a „Non-maximal suppression” algoritmussal szűri ki a rendszer, amely egy utófeldolgozó lépés. Ennek során vesszük egy adott osztályhoz tartozó, meghatározott határértéknél nagyobb valószínűségi értékkel rendelkező dobozokat (H halmaz). Ezek közül kiválasztjuk azt, amelyhez a legnagyobb valószínűség érték tartozik, majd töröljük az összes olyan dobozt, amely a kiválasztott dobozzal egy meghatározott határértéknél nagyobb átfedésben van. Ezt a lépést addig folytatjuk, amíg nem maradnak dobozok a H halmazban. Ezt a folyamatot megismételjük az összes osztályra.

2.2.2 YOLO v2

A YOLO v2 [2] számos újítást hozott az elődjéhez képest, amelyek növelték annak pontosságát, még mindig megtartva a valós idejű feldolgozást.

Az egyik ilyen módosítás a „batch normalization” alkalmazása. Mindegyik konvolúciós rétegen használva, egyrészt nő a modell pontossága, másrészt pedig nincs szükség „dropout”-ra a túltanulás elkerüléséhez.

Ahogy a YOLO v1-et, úgy a YOLO v2-t is először az ImageNet 1000-class versenyhez tartozó adathalmazon [9] tanítják be, hogy ezzel inicializálják a konvolúciós filterek egy részét. Azonban történt egy fontos változtatás a YOLO v1-hez képest: míg az első verzióban az osztályozó adathalmazon történő betanítást követően rögtön megnövelték a bemeneti képek felbontását, és úgy tanították tovább a rendszert egy lokalizációs adathalmazon, addig a második verzióban a lokalizációs adathalmazra való átállás előtt néhány „epoch”-on keresztül az osztályozó adathalmazon tanítják tovább a rendszert, megnövelt bemeneti felbontással. Ezáltal csökkentik a háló terhelését a lokalizációs tanítás során, hiszen nem kell a lokalizációra való átállással párhuzamosan a nagyobb bemeneti felbontásra rátanulnia, hiszen az megtörténik még a lokalizációs lépés előtt.

Nagy újítást hozott az úgynevezett „anchor box” rendszer. Ennek lényege az, hogy a modellünk már a tanítás előtt tartalmaz referencia dobozokat (ezeket nevezzük „anchor box”-oknak), és az objektumok határoló dobozainak helyét a lokalizációkor ezekhez viszonyítjuk. Ez kisebb visszaesést jelent a pontosságban, azonban a képen

jóval több objektumot tudunk felismerni, mint a YOLO v1 esetében. Fontos kérdés, hogy hogyan definiáljuk az „anchor box”-okat. Az írók a „K-means clustering” algoritmus használatát javasolják, melynek futása során a tanítóhalmazban található dobozok jelentik a bemeneti adatokat, és a végső K db „centroid-box” érték adja meg az „anchor box”-okat. A következő távolságfüggvényt használjuk az algoritmus futása közben:

$$d(doboz, centroid\,box) = 1 - IOU(doboz, centroid\,box)$$

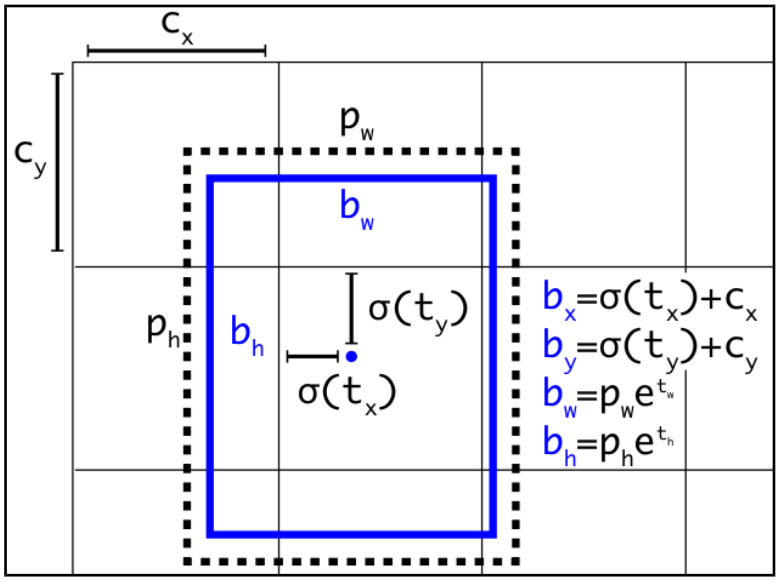
5. képlet: A "K-means clustering" által használt távolságfüggvény

A YOLO v1-hez hasonlóan a cellákon belül becslünk dobozokat. Mindegyik cellához 5 doboz, és mindegyik dobozhoz 5 becsült érték tartozik: t_x, t_y, t_w, t_h, t_o. A cellák pozícióját a kép bal felső sarkához viszonyítva (c_x, c_y)-nal jelöljük, az „anchor box”-ok szélességét és magasságát pedig p_w-vel és p_h-val. Ekkor a becsült dobozok dimenzióit (b_x, b_y, b_w, b_h) és Conf értékét a következő képletek adják meg, ahol σ a logisztikus függvényt jelzi:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \\ Conf &= P(objektum) \times IOU = \sigma(t_o) \end{aligned}$$

6. képlet: A dobozok dimenzióinak és Conf értékének kiszámítása YOLO v2-ben

A cellák, az „anchor box”-ok és a dobozok viszonyát a következő ábra mutatja be, ahol a pontozott téglalap az „anchor box”-ot, a kék téglalap pedig a becsült dobozt mutatja:



3. ábra: A dobozok dimenzióinak számítása YOLO v2-ben [2]

Ahhoz, hogy a rendszer a kisebb objektumokat nagyobb pontossággal ismerje fel, a hálóba egy új architekturális elem került bele, egy úgynevezett „passthrough” réteg, amely a végső „feature map”-et kibővíti egy, a háló korábbi szakaszában szereplő, nagyobb felbontású „feature map”-pel. A lokalizáció ezen a kiegészített tenzoron fog futni, ezáltal pontosabb lesz.

Azért, hogy a modell pontosan fusson különböző képméreteken, a YOLO v2-ben a tanítás során nem állandó a bemeneti kép felbontása, hanem 10 „batch”-enként véletlenszerűen megváltoztatják azt. Mivel a végső „feature map”-ünk mérete $\frac{1}{32}$ -e a bemeneti kép méretének, ezért a tanítás során a bemeneti kép szélességének és magasságának 32 többszörösének kell lennie. A készítők a véletlenszerű kiválasztás során használt képméretet 320×320 -ban minimalizálták, és 608×608 -ban maximalizálták. Abból, hogy ugyanaz a háló többféle képméretre rátanul, az következik, hogy a betanított modell tesztelésekor beállíthatunk kisebb és nagyobb bemeneti felbontást is. Bár kisebb felbontásnál a rendszer kevésbé pontos, mint nagy felbontásnál, nagy előny, hogy gyorsabban fut. Az alkalmazási területtől függően kell döntést hoznunk a bemeneti felbontást illetően, attól függően, hogy mennyire kritikus a futási idő és a lokalizációs pontosság. A YOLO v2-höz készített háló kisebb, mint az elődje, a konvolúciós rétegek száma mindössze 19. A háló architektúrája a következő képen látható:

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

4. ábra: A YOLO v2-höz tartozó háló felépítése [2]

Akárcsak a YOLO v1 esetében, az adathalmazt a YOLO v2 is kibővítve használja. Ezt a képek kivágásával, forgatásával, illetve a képek szaturációjának, fényerősségének és színvilágának változtatásával éri el.

Mivel az objektum lokalizációs adathalmazok elkészítése jóval nagyobb erőforrást igényel, mint az osztályozó halmazoké, általában elmondható, hogy a klasszifikációs adathalmazok több képet és több osztályt tartalmaznak. Ez egy lényeges korlát a lokalizációs algoritmusok fejlesztésekor. Ezt próbálja meg kiküszöbölni a cikkben bemutatott másik rendszer, a YOLO9000 [2], mégpedig úgy, hogy egyszerre tanítja a hálót lokalizációra és klasszifikációra felcímkézett adatokkal.

A készítők az ImageNet klasszifikációs adathalmazából [9] indultak ki. Ehhez a halmazhoz a címkéket a WordNet fogalomhalmaz [10] adja, amely fogalmakat és azoknak a kapcsolatát tartalmazza. A WordNet halmazban található szavak közötti szemantikai kapcsolatot egy irányított gráf írja le. A YOLO9000 készítői – a probléma leegyszerűsítésének érdekében – a gráfból egy fát készítettek, amely hierarchikusan tárolja a fogalmakat. Ehhez kiválasztották a WordNet azon fogalmait, amik valamilyen objektumot írnak le, és megkeresték a gráfban a gyökérelemhez vezető legrövidebb utat. Ezteket az utakat csatolták hozzá a fához. Így született meg a WordTree [2]. A

WordTree-hez a lokalizációs adathalmazban szereplő fogalmakat is hozzákapcsolták, így oldották meg a klasszifikációs és a lokalizációs halmaz összeillesztését.

A dobozban található objektum meghatározásához meg kell becsülnünk a fában található összes fogalomhoz tartozó feltételes valószínűséget, ahol a feltétel az, hogy a dobozban szerepel a fogalom szülőfogalmához tartozó objektum. Ezt a feltételes valószínűség értéket i fogalomra a következőképp jelöljük:

$$P(fogalom_i|szülő(fogalom_i))$$

6. képlet: Egy adott fogalomhoz tartozó feltételes valószínűség a WordTree-ben

A kiértékelés során annyi feltételes valószínűség értéket becsül meg a modellünk, ahány csúcs található a fánkban (a gyökér kivételével). Egy adott fogalomhoz tartozó abszolút valószínűséget úgy kapjuk meg, hogy végigszorozzuk a hozzátartozó feltételes valószínűséget a hierarchiában fölötte lévő fogalmak feltételes valószínűségével (kivéve a gyökérfogalmat, mivel ahhoz nem tartozik feltételes valószínűség). Klasszifikáció esetén feltételezzük, hogy a kép biztosan tartalmazza a gyökérobjektumot, más szóval biztosan tartalmaz egy objektumot, tehát $P(gyökér)=1$, lokalizáció esetén pedig értelemszerűen $P(gyökér)=P(objektum)$. A tanítás során egy dobozhoz a saját címkéjén kívül hozzárendeljük a címkéhez tartozó szülőfogalmakat is, így egy doboz több címkével is rendelkezik. Ezt kihasználhatjuk akkor, ha egy fogalom kis valószínűséggel szerepel a dobozban, de az egyik szülőfogalma nagyobb valószínűséggel van jelen. Az objektum meghatározásakor azt a legnagyobb valószínűséggel rendelkező és legspecifikusabb fogalmat választjuk, amely egy bizonyos határértéknél nagyobb valószínűséggel rendelkezik.

A YOLO9000 esetén a teljes ImageNet halmaz 9000 osztályát kapcsolták össze a COCO [11] objektum lokalizációs adathalmazban található osztályokkal. A tanítás során az „anchor box”-ok számától eltekintve a YOLO v2 architektúrát használták a készítők. Amikor a lokalizációs adathalmazból érkezik kép, akkor a YOLO teljes veszteségfüggvényét használjuk, amikor pedig a klasszifikációs halmazból, akkor csak azokat a részeket, amik relevánsak a klasszifikáció szempontjából. A két halmazon történő közös tanítás következtében a modell egyszerre használja ki a lokalizációs halmazban található pozícióinformációkat, illetve azt a tényt, hogy a klasszifikációs adathalmaz számos osztállyal rendelkezik, és az osztályai között fennálló kapcsolatokat is ismerjük.

2.2.3 YOLO v3

A YOLO legfrissebb, 2018 tavaszán megjelent verziója a YOLO v3 [3]. Nem tartalmaz olyan sok újítást, mint amennyit a második verzió hozott, azonban néhány módosításnak köszönhetően a rendszer még pontosabb lett.

A készítők egy új háló architektúrát mutattak be, amely jóval mélyebb az elődjénél, 53 konvolúciós réteget tartalmaz. A háló egy „residual network” [12], amely egy olyan technológia, amelyet mély neurális hálók tanítási folyamatának stabilizálására használnak. A rendszer a mélységének köszönhetően pontosabb, de értelemszerűen lassabban működik. Ezt valamelyest ellensúlyozza az a tény, hogy a háló jobban kihasználja a GPU-t, mint az elődei. A háló felépítése a következő ábrán látható:

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

5. ábra: A YOLO v3-hoz tartozó háló felépítése [3]

Tanítás során az osztályok becslésének optimalizációjához az előző verzióktól eltérően „binary cross-entropy loss”-t használunk. A dobozokat három különböző méretű „feature map”-en becsüljük meg, és mindegyik dobozra megbecsüljük a második verzióban is bemutatott 5 koordinátát, illetve az egyes osztályokhoz tartozó valószínűség értékeket. A tanítás során - a második verzióhoz hasonlóan - a bemeneti képméretet véletlenszerűen változtatjuk.

3 Autók lokalizációja és osztályzása

3.1 A feladat

A feladatomnak autók képeken történő lokalizációját választottam. Ehhez két betanított modellt készítettem:

- Egy 1 osztállyal rendelkező modellt, mely általánosan az autókat lokalizálja.
- Egy több osztállyal rendelkező modellt, amely a lokalizáció mellett az autók márkáját is felismeri.

A feladat leírása során kitérek mindkét modellre. Az implementációt megelőző lépésekről a függelékben írok.

3.2 Az adathalmaz beszerzése

Mint minden gépi tanulásos feladatnál, itt is nagyon fontos lépés a megfelelő adathalmaz beszerzése. Egy olyan halmazra volt szükségem, amely autókról készült képeket tartalmaz, és fel van annotálva az autókat határoló téglalapok dimenzióival, illetve az autók márkájával. A Stanford Egyetem Cars Dataset-ére [14] esett a választásom.

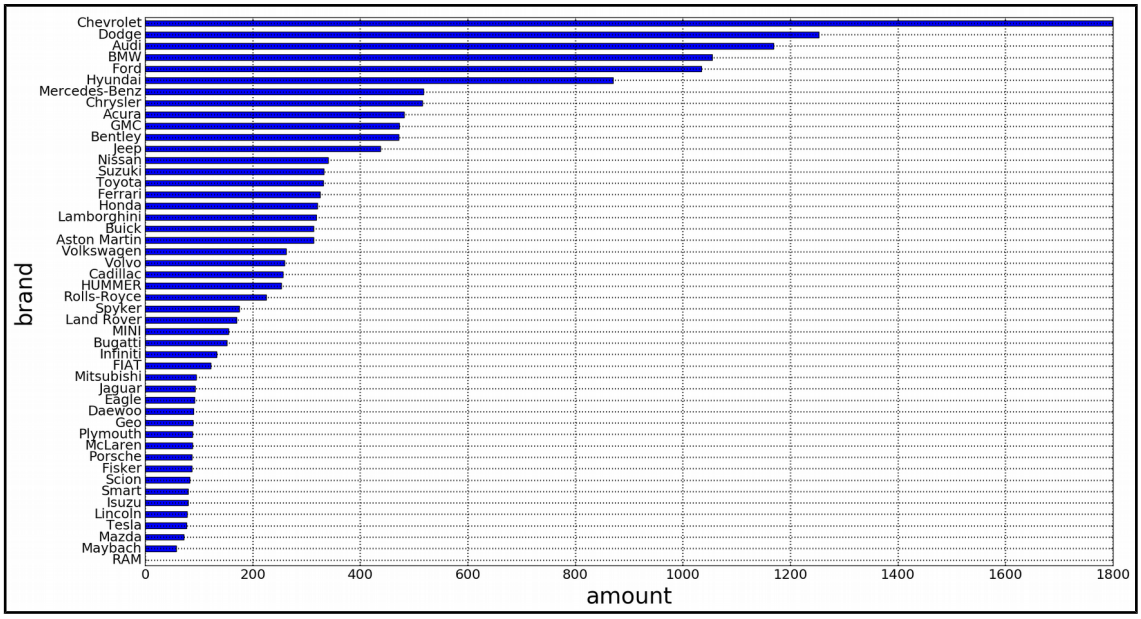
Ez 16185 képet tartalmaz autókról, amikhez egy annotáció fájlban kapjuk meg az autókat határoló téglalapokat, illetve az autók márkáját, azon belül modelljét, illetve az adott modellhez tartozó évszámot.

A következő ábrán az adathalmazunk egyik elemét, a 008880-as ID-jú képet láthatjuk. A képen az adathalmazban hozzátartozó határoló téglalapot is elhelyeztem:

6. ábra: Az adathalmaz egyik eleme

Az annotációkat egy .mat kiterjesztésű MATLAB fájl tartalmazza, amely a ScyPy Python könyvtárral könnyedén beolvasható.

Megvizsgáltam hogy az egyes márkákhoz hány kép tartozik. A képek márkák szerinti eloszlását a következő ábra mutatja:



7. ábra: Az autómárkák eloszlása az adathalmazban

Láthatjuk, hogy az eloszlás közel sem egyenletes, erre figyelnünk kell tanítás során, melynek részleteire a dolgozat későbbi részében térek ki.

3.3 A workflow

A következőkben bemutatom a folyamatot, amely során eljutottam a modellek betanításáig és teszteléséig.
Ennek lépései a következők:

1. **Az adathalmaz feldolgozása**
2. **A feldolgozott címkék ellenőrzése**
3. **YOLO-specifikus *.names* és *.data* fájlok elkészítése**
4. **„Anchor box”-ok meghatározása**
5. **A modell konfigurálása**
6. **A modell tanítása**
7. **A modell validálása**
8. **A modell tesztelése**

A következő alfejezetekben ezeket a pontokat fejtem ki részletesebben.

3.3.1 Az adathalmaz feldolgozása

Az adathalmaz beszerzése után az első lépés annak feldolgozása. Ennél a pontnál három feladatunk van: ki kell választanunk, hogy mely adatokat használjuk fel az adathalmazból, ezután fel kell osztanunk a kiválasztott adatokat valamilyen elv szerint, illetve megfelelő formátumúra kell alakítani az adathalmaz címkéit.

A kétféle betanított modellhez az adathalmaz nem ugyanazon részét használtam fel. Az 1 osztályt tartalmazó modellhez a teljes adathalmazt felhasználtam, a teljes adathalmazból készítettem a tanítóhalmazt amelynek minden képére betanítottam a rendszert, hogy az minél jobban tudjon általánosítani. A több osztályt tartalmazó modell esetében figyelembe kellett vennem az adathalmaz tulajdonságait. Az adathalmaz a neurális hálók tanítására használt halmazok körében kis méretűnek számít. Az egy márkához tartozó képek számát 1000-ben minimalizáltam, csak azokat a márkákat hagytam benne a halmazban, amelyekből rendelkezésre állt legalább ennyi adat. Ez a feltétel 5 márkára teljesült, melyek a következők: *Audi*, *BMW, Chevrolet, Dodge, Ford*. Így a második modellemet 5 osztály lokalizációjára tanítottam be.

Az adathalmazt három diszjunkt részre osztottam fel: „train”, „dev” és „test” halmazra. A „train” halmazt a modell tanításához használtam fel. Ez a legnagyobb méretű a három részhalmaz közül, hiszen a modellnek sokféle képet kell látnia a tanulás közben, hogy megfelelően tudjon általánosítani a kiértékelésnél. A „dev” halmazt a

modell validálásakor használjuk fel. Ennek segítségével döntjük el, hogy a tanítás során elmentett állapotok közül melyik legyen a rendszer végső állapota. Ennek célja az, hogy megakadályozzuk a rendszer túltanulását, azaz azt, hogy a rendszer túlságosan rátanuljon a „train” halmazra, ezzel elveszítve általánosító képességét. Fontos, hogy ez a halmaz ne tartalmazzon elemeket a „train” halmazból, mivel az a célunk, hogy a validálás teljesen független legyen a tanítástól. A harmadik halmaz a „test” halmaz, amelyen a validáció során kiválasztott modellt értékeljük ki, és mérjük meg vele a rendszer pontosságát. A „train” halmaz a teljes adathalmaz 80 százalékát, a „dev” és a „test” halmaz pedig a 10-10 százalékát tartalmazza. Ahogy korábban említettem, az adathalmazban a különböző márkájú autók nem ugyanolyan arányban szerepelnek, hanem nagy különbség figyelhető meg a számukban. Ezt megfelelő módon kell kezelnünk az adathalmaz felosztásánál, törekedni kell arra, hogy mindhárom részben közel azonos legyen a különböző márkák eloszlása. Mivel gyakran egy márkán belül is sokféle kinézetű autó található, ezért arra is odafigyeltem, hogy a három részhalmazon belül ne csak a márkák, hanem a modellek és a hozzájuk tartozó évjáratok eloszlása is ugyanolyan legyen. A YOLO tanítás során egy olyan szövegfájlt vár paraméterként, amely felsorolva tartalmazza a tanítófájlok elérési útját. A teszteléshez egy ugyanilyen fájlt vár, értelemszerűen a tesztfájlok elérési útjával. Az adathalmaz felosztásánál ezeket a fájlokat is legeneráltam.

A harmadik feladatunk az adathalmaz címkéinek megfelelő formátumúra alakítása. A Stanford Egyetem Cars Dataset-ben egy osztály nem az autómárkához, hanem egy autómárkán belüli adott típus adott évjáratához tartozik. A több osztályt tartalmazó modellünk esetén az adatfeldolgozás során ebből ki kell nyernünk a márkát, hiszen arra tanítjuk be a modellünket. A tanítás során az osztályokat nem a nevükkel, hanem egy egész számmal kell megadni YOLO-nak. A számozást 0-tól kezdődik, és C osztály esetén $(C-1)$ -ig tart. Az 1 osztályt tartalmazó modellünk esetén mindegyik téglalaphoz a 0 osztálykód tartozik, a több osztályt tartalmazó modell esetén pedig az osztálytól függően a 0, 1, 2, 3, 4 kódok egyike, hiszen 5 osztályt különböztetünk meg. A Stanford Egyetem Cars Dataset-e [14] a következő formátumban tárolja az autókat határoló téglalapok dimenzióit: $(x_{1,}\, y_{1,} x_{2,}\, y_2)$, ahol $(x_{1,}\, y_1)$ a téglalap bal felső, $(x_{2,}\, y_2)$ pedig a téglalap jobb alsó pontjának koordinátáit jelöli. A koordináták abszolútak, és a kép bal felső sarkát vesszük a $(0{,}0)$ pontnak. Ezzel szemben a YOLO tanítás során a következő formátumban várja a

határoló téglalapok dimenzióit: (c_x, c_y, w, h), ahol (c_x, c_y) a téglalap középpontjának koordinátáit jelöli, w és h pedig a téglalap szélességét és magasságát. Mind a középpont koordinátái, mind pedig a szélesség és magasság értékek relatívak, a teljes kép szélességéhez és magasságához viszonyítjuk őket, tehát a $[0,1]$ intervalluma esnek. A YOLO-nak a címkéket mindegyik képre külön szövegfájlban kell megadni, melynek egy sora az egy téglalaphoz tartozó értékeket tartalmazza: az osztály sorszámát, majd pedig a téglalap dimenzióit, szóközzel elválasztva. Ezt a transzformációt ebben a lépésben végezzük el.

3.3.2 A feldolgozott címkék ellenőrzése

A címkék feldolgozása után érdemes leellenőrizni, hogy tényleg megfelelnek-e a YOLO által elvárt formátumnak. A Yolo_mark [15] nevű program alapvetően arra való, hogy a saját adathalmazunkat felcímkézzük vele, és a címkéket a YOLO által elvárt formátumban mentsük el. Ha betöltünk a programba egy képet és a hozzátartozó címkefájlt, egy grafikus felületen látjuk a képet a rárajzolt téglalapokkal, és így le tudjuk ellenőrizni, hogy megfelelnek-e a téglalapok az elvárásainknak. Ezt érdemes szúrópróbaszerűen leellenőrizni néhány képre, hogy lássuk, jól dolgoztunk-e az előző lépésben.

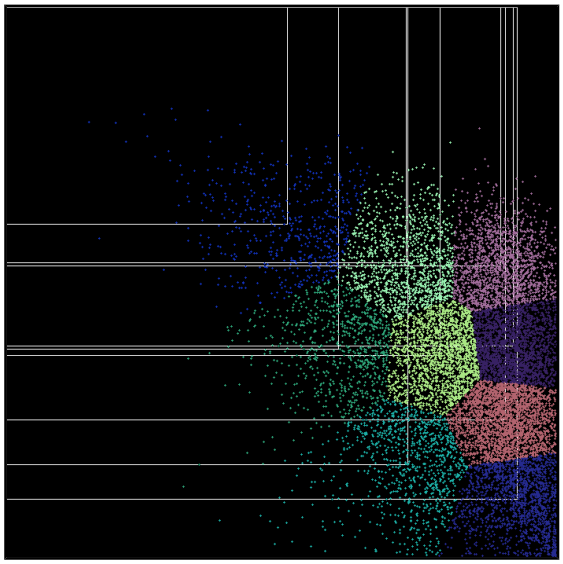
3.3.3 YOLO-specifikus *.names* és *.data* fájlok elkészítése

Ahogy korábban említettem, a YOLO tanítás közben egész számként kezeli az osztályokat. A számkód és osztálynév közötti leképzést egy *.names nevű fájl segítségével végzi el a rendszer. Ez egymás alatt felsorolva tartalmazza az osztályok neveit. Az n számkód a *.names fájl $(n+1)$. sorában található osztálynévre képződik le, például a 0-s számkód az 1. sorra, az 1-es számkód a 2. sorra, és így tovább.

A tanítás során használt fájlok elérési útvonalát a *.data nevű fájl tartalmazza. Ebben megadjuk az osztályok számát, a tanítás, illetve a tesztelés során használt fájlok elérési útvonalát tartalmazó fájlok helyét, a *.names fájl elérési útvonalát, illetve azt a mappát, ahova a rendszer a modell aktuális állapotát tartalmazó súlyfájlokat menti a tanítás során.

3.3.4 „Anchor box”-ok meghatározása

A YOLO által használt „anchor box”-okat „K-means clustering” algoritmussal határozzuk meg, amelyet a tanító adathalmazban található határoló téglalapokon futtatunk. Erre a YOLO egy kiterjesztett verzióját használtam [16], amely tartalmazza ezt a funkciót. Megadva neki a tanítóadathalmazt, a klaszterek számát és a képméretet, elvégzi a klaszterezést, és megadja a „centroid”-okat, amik az „anchor box”-ok lesznek. A „K-means-clustering” futtatása során 9 klasztert készítettem. Ennyi klaszterrel dolgoztak a YOLO v3 készítői is. Az 1 osztályt tartalmazó adathalmaz esetén a tanító adathalmazomból készült klasztereket, illetve a hozzájuk tartozó „anchor box”-okat a 8. ábra mutatja. Ezeket az „anchor box”-okat használtam fel a tanítás során.



8. ábra: Klaszterek és "anchor box"-ok

3.3.5 A modell konfigurálása

A tanítást a *.cfg fájllal konfiguráljuk. Ez tartalmazza a háló architektúráját, és a hozzátartozó beállításokat. A darknet keretrendszer yolov3.cfg konfigurációs fájljából indultam ki, ezt módosítottam a saját feladatomnak megfelelően.

Mini-batch tanítást használunk, amelynél a batch méretét a konfigurációs fájl tartalmazza. Mindkét modellemhez 64-es batch méretet választottam. A teljes batch-et úgynevezett „subdivision”-ökkel továbboszthatjuk, hogy tanítás során ne teljen meg a VRAM-unk. A subdivison méretét is 64-re állítottam, ami azt jelenti, hogy az egyes mini-batch-eket 64 további részre osztom, tehát egyszerre csak 1 képet olvasok be. Tanítási hiperparaméterekként a YOLO v3 által használt paramétereket állítottam be.

A hálóban három „yolo” típusú réteg található. Ezek adják a becsléseket a három különböző méretű „feature map”-en. Az ezek előtti konvolúciós rétegek filterszámát [17] alapján a következő képlet adja meg, ahol C az osztályok számát jelöli:

$$\boxed{n=3\times(5+C)}$$

7. képlet: A yolo réteg előtti réteg filtereinek száma

A 3 „yolo” típusú rétegben meg kell adni az osztályok számát, illetve az előző lépésben generált „anchor box”-ok koordinátáit. Használhatjuk az alapértelmezetten megadott „anchor box”-okat is, azonban pontosabb eredményt érhetünk el, ha azokat a saját adathalmazunkhoz igazítjuk. A saját implementációmban az „anchor box”-okat - ahogy a workflow előző lépésében bemutattam - az általam használt adathalmaz sajátosságai alapján generáltam.

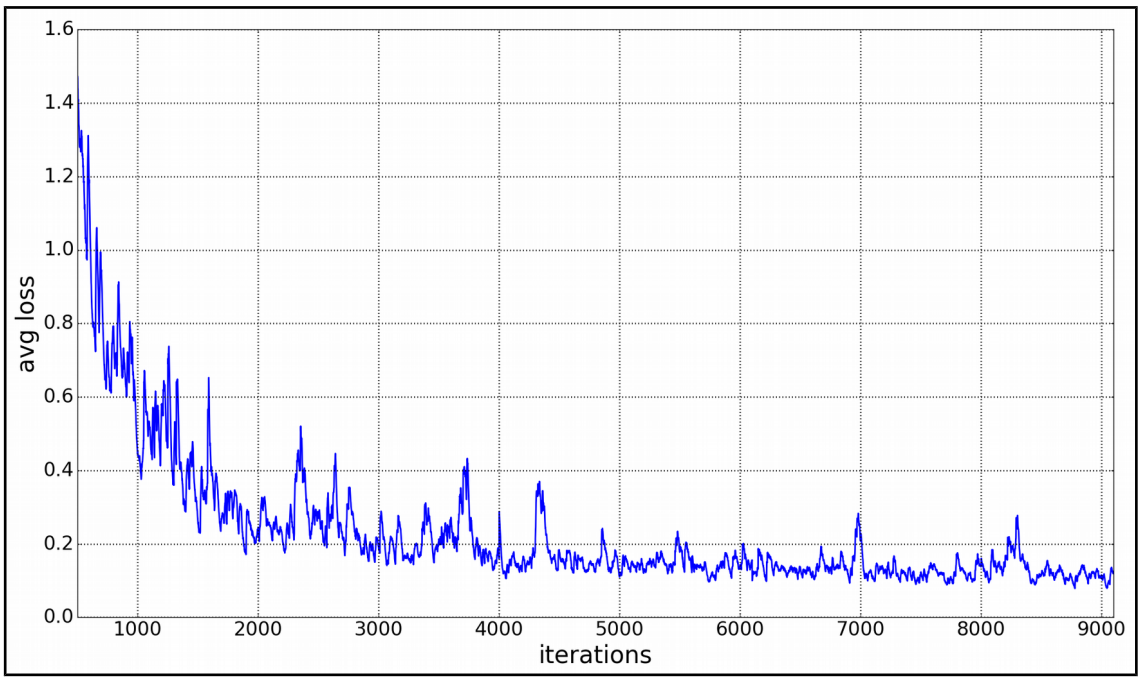
3.3.6 A modell tanítása

A modell tanítása során a darknet-nek az előző lépésekben bemutatott *.*data* és *.*cfg* fájlokat adjuk meg, illetve ha egy előtanított háló tanítását folytatjuk, akkor egy, az előtanított háló súlyait tartalmazó *.*weight* fájlt. Ahogy a 2. fejezetben írtam, a YOLO készítői előtanították a hálót az ImageNet 1000-class versenyhez tartozó adathalmazon [9]. Az így előtanított háló súlyait tartalmazó fájlt közzétették a darknet [13] honlapján. Tanításaim során ebből az előtanított hálóból indultam ki.

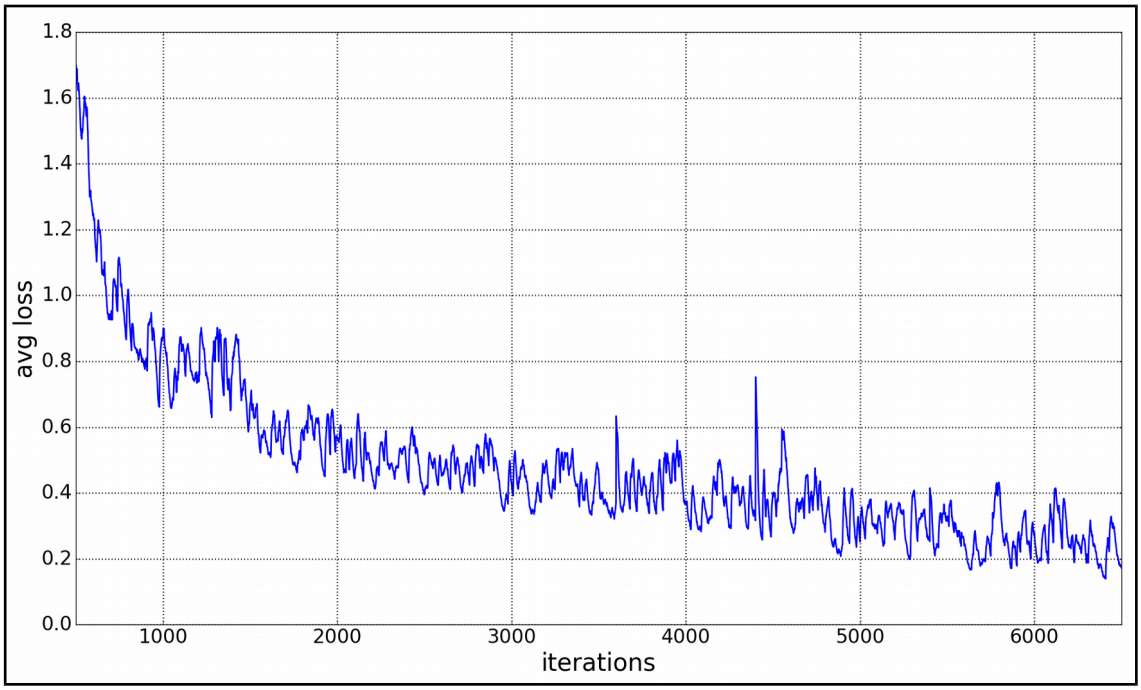
A súlyfájlokat 100 iterációként elmentettem, hogy validálni tudjam a rendszert, illetve, hogy ha valamiért le kellene állítani a tanítást, később folytatni tudjam azt. A tanítás kimenetét egy log fájlba vezettem ki, amelyet feldolgozva követni lehet a veszteségfüggvény értékének változását.

Az 1 osztállyal rendelkező modellt 9100 iteráción át tanítottam, amely kb. 27 órába telt. A több osztállyal rendelkező modellt pedig 6500 iteráción át, amely kb. 20 órát vett igénybe.

Az átlagos veszteségértékek (avg loss) így alakultak az iterációszám függvényében (9. ábra: 1 osztály, 10. ábra: több osztály):



9. ábra: Az 1 osztályú modellhez tartozó átlagos veszteségértékek



10. ábra: A több osztályú modellhez tartozó átlagos veszteségértékek

Az átlagos veszteségértékek az elvártnak megfelelően csökkennek, tehát rá tudott tanulni a modell az autók képeken történő lokalizációjára és osztályzására.

3.3.7 A modell validálása

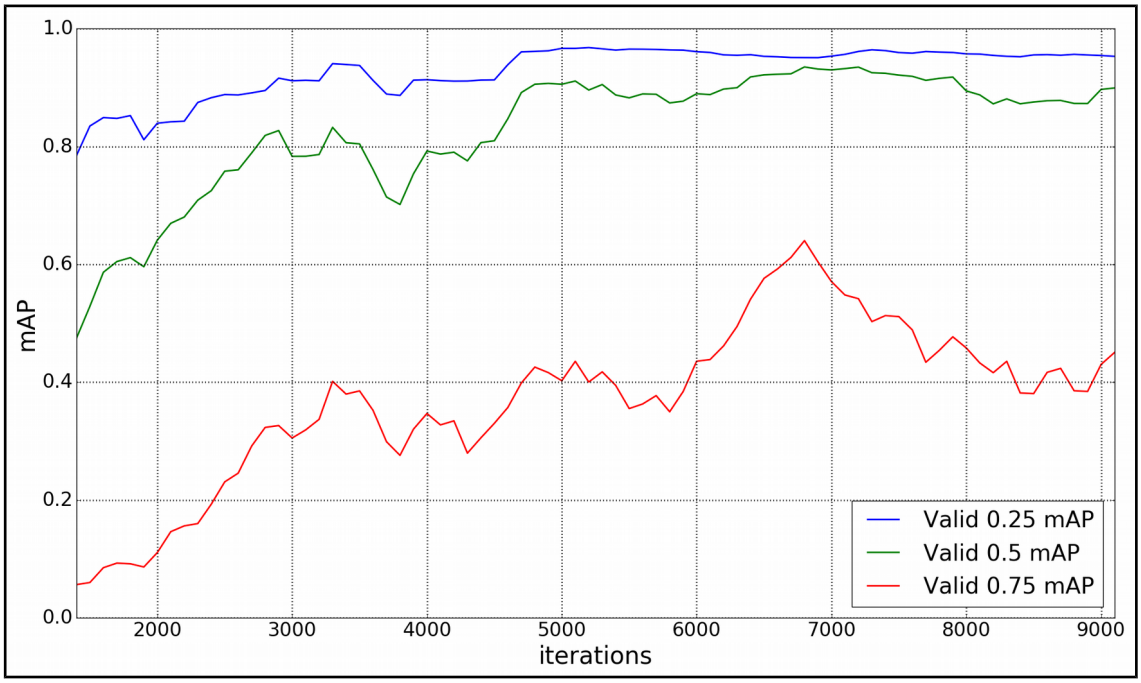
A veszteségfüggvényünk értéke folyamatosan csökken. Bár ezzel párhuzamosan egy ideig a rendszerünk pontosságát mérő függvény értéke folyamatosan nő, egy

bizonyos pont után csökkenni kezd. Ez az a pont, ami után a rendszer túlságosan rátanul a tanító adathalmazra, amelynek következtében romlik az általánosító képessége. Ezt a pontot kell megtalálnunk, hogy visszaállíthassuk a rendszert arra az állapotra, amely az egyik legpontosabb eredményt adja. Ezt a pontot keressük a validáció során.

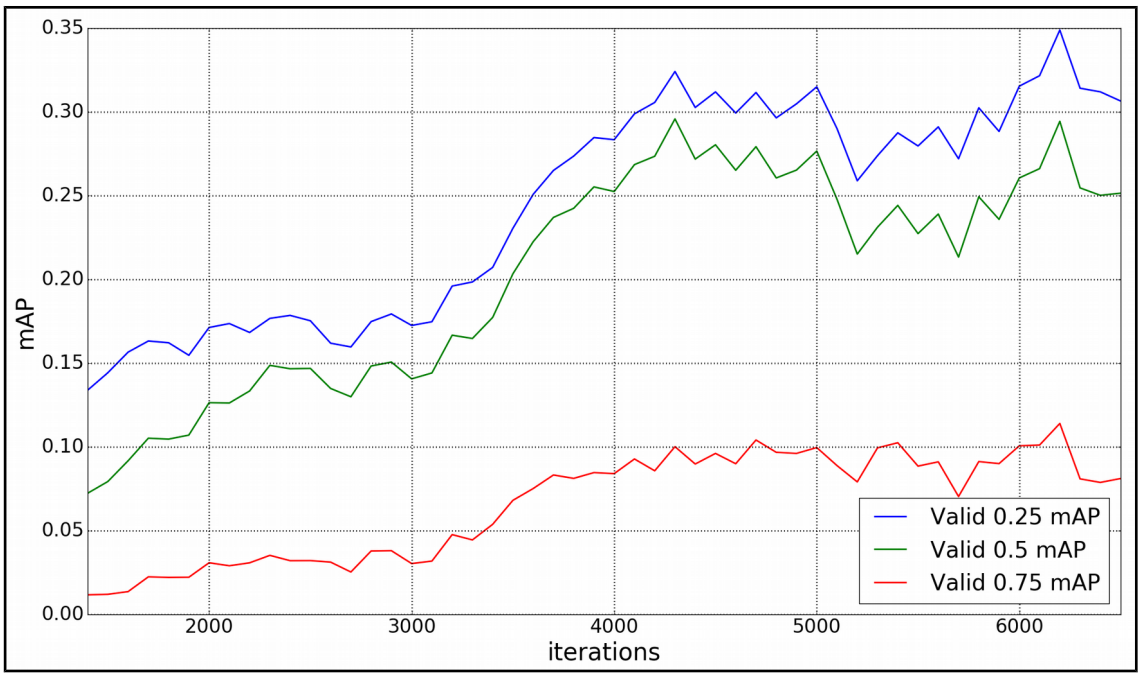
Ehhez először ki kell választanunk, hogy hogyan mérjük a rendszerünk pontosságát. Az objektum lokalizáció világában az egyik standard mérőszám a „mean Average Precision”, továbbiakban mAP. Az „Average Precision”-t egy adott IOU határértékre, egy adott osztályra számítjuk ki, mégpedig úgy, hogy vesszük az adott osztály adott IOU értékéhez tartozó „Precision-Recall” görbét, majd megbecsüljük a görbe alá eső területet. Ez a görbe úgy jön létre, hogy vesszük az validációs halmazon vett becsléseinket, ezeket a becsléshez tartozó valószínűség érték alapján csökkenő sorrendbe tesszük, majd végighaladva a becsléseken kiszámítjuk az validációs halmaz adott pontigjáig mért „precision” és „recall” értékeket. Ezeket az értékeket ábrázoljuk az összes becslésre. A „precision” értéket a $\frac{TP}{TP+FP}$ képlet, a „recall” értéket pedig a $\frac{TP}{TP+FN}$ képlet adja meg, ahol TP („true positive”) egy osztályon belül azoknak a becsült dobozoknak a számát adja meg, melyek a tényleges dobozokkal egy megadott IOU értéknél nem kisebb mértékben vannak átfedésben. A FP („false positive”) egy osztályon belül azoknak a becsült dobozoknak a számát adja, melyek a tényleges dobozokkal egy megadott IOU értéknél kisebb mértékben vannak átfedésben. A FN („false negative”) pedig azoknak a tényleges dobozoknak a számát adja, amelyet nem becsült meg egy megadott IOU értéknél nemkisebb átfedéssel az algoritmusunk. Az AP számításakor - tehát a „precision” és a „recall” számításakor is - a TP, FP és FN kiszámításakor használt IOU határérték megegyezik. Miután mindegyik osztályra kiszámoltuk az „Average Precision” értékeket egy adott IOU határértéket használva, ezeket az értékeket átlagoljuk, hogy megkapjuk az adott IOU határértékhez tartozó „mean Average Precision”-t. A mAP értékek kiszámításához a [18] *pascalvoc.py* programját használtam.

Egy betanított modellre a validálás két lépésben zajlott. Először megmértem a mAP-et 100 iterációnként 0,25-ös, 0,5-ös és 0,75-ös IOU határértékekre, és mindegyik IOU határértékre 10-es csoportonként átlagoltam ezeket. A 10-es csoportok nem diszjunktak, két egymást követő 10-es csoport 9 közös elemet tartalmaz. Az így kapott átlagos mAP értékeket ábrázoltam. Az ábrán egy adott iteráció sorszámához tartozó

érték az adott iteráció, és az azt megelőző 9 mért iteráció mAP értékeinek az átlaga. A két modellre a következő mérések születtek (11. ábra: 1 osztály, 12. ábra: több osztály):



11. ábra: 1 osztályú modellhez tartozó átlagos mAP értékek



12. ábra: Több osztályú modellhez tartozó átlagos mAP értékek

Az 1 osztályú modell legmagasabb átlagos mAP értékét 0,75-ös IOU határértéknél a 6800. iterációban mérjük, a több osztályú modellét pedig a 6200.-ban.

A validálás második lépésében mindkét modell esetén azt a 10 elmentett modellállapotot vettem, amelyekhez tartozó mAP értékekből a maximális átlagos mAP

értéket számoltuk ki 0,75-ös IOU határérték esetén. Az 1 osztályú modellnél ezek az 5900 és 6800, a több osztályú modellnél pedig az 5300 és 6200 közötti iterációkhoz tartozó modellállapotok voltak. Modellenként a 10-10 modellállapot közül végül azt választottam ki, amelynél a 0,75-ös IOU határértékkel számított mAP érték a legmagasabb. Az 1 osztályú modellnél ez a 6700. iterációhoz, a több osztályú modellnél pedig az 5300. iterációhoz tartozó modellállapot volt. Ezek adják a modell végső állapotát, ezekre teszteltem le a rendszeremet. A teszteredményeimet a következő részben mutatom be.

3.3.8 A modell tesztelése

A workflow utolsó eleme az elkészült és validált modell tesztelése. A tesztelés eredményei a következő táblázatban találhatóak:

	1 osztály	**Több osztály**
0.25 mAP	95.5%	53.3%
0.50 mAP	95.2%	52.1%
0.75 mAP	76.3%	26.2%

A több osztályú modellnél az egyes osztályokhoz tartozó „Average Precision” értékek 0,25-ös, 0,5-ös és 0,75-ös IOU határérték esetén:

	Audi	**BMW**	**Chevrolet**	**Dodge**	**Ford**
0.25 mAP	59.5%	65.3%	64.7%	44%	33%
0.50 mAP	58.9%	62.1%	63%	44%	32.6%
0.75 mAP	33%	26.9%	24%	27.6%	19.4%

4 Konklúzió

A dolgozatomban bemutattam az objektum lokalizáció irodalmát, kiválasztottam a YOLO algoritmust, bemutattam egy arra épülő, általam készített rendszer implementációját, végül pedig leteszteltem a betanított modelleket.

A rendszer számos irányban továbbfejleszthető. A teljesség igénye nélkül felsorolok néhány potenciális továbbhaladási irányt a jövőre nézve:

- Az 1 osztályú és több osztályú modell összekapcsolása közös döntéshozatallal
- A rendszer továbbtanítása, és az így létrejött modell elemzése
- Betanítás az eredeti „anchor box”-okkal, az így kapott modellek összehasonlítása a jelenlegi modellekkel
- Tanítás nagyobb adathalmazzal
- Tanítás új objektumokra
- Az adathalmaz kiterjesztése újfajta módszerekkel
- Egyéb validációs módszerek kipróbálása
- Kiindulás a betanított YOLO v3 modellből, és annak a finomhangolása
- Egyéb hálóarchitektúrák kipróbálása

Bár a dolgozatomban bemutatott rendszer még gyerekcipőben jár, egy továbbfejlesztett verziója számos területen használható lenne, melyek közül néhány:

- Önvezető autók területe
- Forgalomszámlálás
- Autó sebességének mérése kamerakép alapján
- Autómárkák lefedettségének vizsgálata
- Adott területre jellemző közlekedési szokások elemzése
- Filmek, videók automatikus elemzése, annotálása

Irodalomjegyzék

[1] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi: *You Only Look Once: Unified, Real-Time Object Detection*, arXiv preprint arXiv: 1506.02640, 2015

[2] Joseph Redmon, Ali Farhadi: *YOLO9000: Better, Faster, Stronger,* In Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, pages 6517–6525. IEEE, 2017

[3] Joseph Redmon, Ali Farhadi: *YOLOv3:* A*n Incremental Improvement*, arXiv, 2018

[4] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, Deva Ramanan: *Object Detection with Discriminatively TrainedPart Based Models*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(9): 1627–1645, 2010

[5] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik: *Rich feature hierarchies for accurate object detection and semantic segmentation*, In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pages 580–587. IEEE, 2014

[6] Ross Girshick: *Fast R-CNN*, CoRR, abs/1504.08083, 2015

[7] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun: *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, arXiv preprint arXiv: 1506.01497, 2015

[8] M. A. Sadeghi and D. Forsyth: *30hz Object Detection with DPM V5*, In Computer Vision–ECCV 2014, pages 65–79. Springer, 2014

[9] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh,S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein,A. C. Berg, and L. Fei-Fei: *ImageNet Large Scale Visual Recognition Challenge*, International Journal of ComputerVision (IJCV), 2015

[10] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J.Miller: *Introduction to wordnet: An on-line lexical database.International journal of lexicography*, 3(4):235–244, 1990

[11] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ra-manan, P. Dollár, and C. L. Zitnick. *Microsoft coco: Com-mon objects in context*. In European Conference on Com-puter Vision, pages 740–755. Springer, 2014

[12] K. He, X. Zhang, S. Ren, and J. Sun: *Deep residual learning for image recognition*. In Proceedings of the IEEE con-ference on computer vision and pattern recognition, pages 770–778, 2016

[13] J. Redmon: *Darknet: Open source neural networks in c*. http://pjreddie.com/darknet/, 2013–2018

[14] Jonathan Krause, Michael Stark, Jia Deng, Li Fei-Fei: *3D Object Representations for Fine-Grained Categorization*, 4th IEEE Workshop on 3D Representation and Recognition, at ICCV 2013 (3dRR-13). Sydney, Australia. Dec. 8, 2013

[15] *Yolo_mark*, URL: https://github.com/AlexeyAB/Yolo_mark, Utolsó letöltés ideje: 2018-10-26

[16] *Yolo-v3 and Yolo-v2 for Windows and Linux*, URL: https://github.com/AlexeyAB/darknet, Utolsó letöltés ideje: 2018-10-26

[17] *How to train (to detect your custom objects)*, URL: https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects, Utolsó letöltés ideje: 2018-10-26

[18] *Metrics for object detection*, URL: https://github.com/rafaelpadilla/Object-Detection-Metrics, Utolsó letöltés ideje: 2018-10-26

Függelék

Az rendszer implementációját megelőző lépések

Mielőtt elkezdenénk foglalkozni a rendszerünk felépítésével, néhány lépést szükséges megtennünk.

Először is be kell szereznünk a YOLO kódját. Ezt a darknet [13] nevű nyílt forráskódú neurális háló keretrendszer tartalmazza.

Ha van rá lehetőség, használjunk videokártyát a tanításhoz és a futtatáshoz. Nvidia GPU-t érdemes választanunk, mivel a gépi tanulással foglalkozó közösségnél ez számít standardnak, és a videokártyán futó rendszerek nagy része úgy készül, hogy komolyabb átalakítások nélkül csak Nvidia videókártyán tudnak futni. Én egy Nvidia GeForce GTX 1060-at használtam 3GB-os VRAM-mal. Két programot kellett installálni, egyrészt a CUDA-t, amely egy GPU API Nvidia videokártyákhoz, illetve a cuDNN-t, amely egy mély neurális háló könyvtár. A YOLO mindkettő programot használja, a GPU használata tanításnál kb. 120-szoros, tesztelésnél pedig kb. 400-szoros sebességnövekedést jelentett a CPU-n való futtatáshoz képest.

Érdemes az OpenCV-t is felinstallálni, amely egy „computer vision” könyvtár. Segítségével a háló tesztelésénél nemcsak fájlba menthetjük a képet a rajta bejelölt dobozokkal és objektumtípusokkal, hanem rögtön megjeleníthetjük azt. Ez könnyebséget jelent, ha egymás után sok képen próbáljuk ki a modellünket. Emellett az OpenCV használatával videókat is könnyedén be tudunk olvasni a rendszerbe, és a képkockákat a videó lejátszása közben tudjuk futtatni a hálón, így a videó nézése közben láthatjuk a bejelölt objektumokat.

A YOLO-t ezután úgy build-eljük hogy a hozzátartozó Makefile-ban, a CUDA-hoz, cuDNN-hez, illetve OpenCV-hez tartozó flag-ek értékét 1-re állítjuk.