

Közlekedési objektumok detektálása YOLO architektúrájú hálókkal

Nyilas Péter*

2024. május 21.

*Konzulensek: Dr. Hullám Gábor, Révy Gábor

Tartalomjegyzék

I. Közlekedési objektumok detektálása Yolov5-ös hálóval	4
1. Bevezetés	4
1.1. Felvetett problémakör	4
2. Konvolúciós Neurális hálók	5
2.1. 2D diszkrét Konvolúció	5
2.2. Pooling	6
2.3. Teljesen összekapcsolt réteg	6
2.4. Batch normalizálás	6
3. Yolov5 és gyakorlati alkalmazása	7
3.1. YOLOv5 és használt verziók	7
3.2. Adathalmaz, Cityscapes	8
3.3. Formátumkonverzió	9
3.4. Tanítás és validáció, adatok kiértékelése	10
3.5. Megoldás	11
II. Yolov8-as szegmentációs háló és annak magyarázhatósága EigenCAM típusú modellfüggő magyarázó rendszerrel	12
1. Bevezetés	13
2. Mély tanulás és interpretálhatóság	14
2.1. Magyarázhatóság	14
3. YOLOv8: Szemantikus szegmentációs háló és működése	15
3.1. Yolov8 architektúra	15
3.2. Szemantikus szegmentáció	17
3.3. Adathalmazok és kísérletek	18
3.4. MLOps	18
4. EigenCAM: Modellfüggő magyarázó	20
4.1. Aktivációk	20
5. Az EigenCAM alkalmazása a YOLOv8-ra	21
5.1. Implementáció lépései	22

6. Magyarázhatalmazás	23
6.1. Magyarázat	24
7. Az EigenCAM alkalmazásának gyakorlati haszná	25
7.1. Eredmények összegzése	25
8. Jövőbeli irányok és kutatási lehetőségek	26
9. Szó- és rövidítés jegyzék	28
9.1. Rövidítésjegyzék	29

I. rész

Közlekedési objektumok detektálása Yolov5-ös hálóval

Konzulensek: Dr. Hullám Gábor és Révy Gábor

1. Bevezetés

Az AI alapú képfelismerést a konvolúciós neurális hálók forradalmasították és széles körben alkalmazzák őket a mai napig különböző objektumfelismerési és osztályozási problémák megoldására.

A konvolúciós neurális hálók speciális rétegei között találhatók az ún. konvolúciós rétegek (Conv), amelyek az input képet szűrőkkel (kernelekkel) dolgozzák fel. Ezek a szűrők olyan jellegzetességeket keresnek a képen, például élek vagy színek, és a kép területi szintű információit vonják ki. Azután a háló további rétegei, például a pooling rétegek (Pool) és teljesen összekapcsolt rétegek, tovább finomítják és kinyerik az absztrakt jellemzőket és továbbítják a következő rétegnek.

A konvolúciós neurális hálók alkalmazása kiterjed a képfelismerésen túl, beleértve az arckifejezések és érzelmek felismerését, Viselkedés analízisen keresztül az önvezető autók látási rendszeréig. Ezek a hálózatok effektíven képesek megtanulni és reprezentálni a vizuális információt.

Én is egy ilyen problémának a megoldásához használtam ezeket a hálókat, pontosabban közlekedési objektumok felismerésére és osztályozására, Dashcam (fedélzeti kamerás) felvételekből. Mely fontos az önvezető autók vagy a vezetést támogató rendszerek fejlesztésében, a méréstechnika fejlesztésére méréselemzési feladatok gyorsítására.

1.1. Felvetett problémakör

A dashcames felvételeket általában ADAS (fejlett vezetéstámogató rendszer) fejlesztésénél gyakran vesznek fel radar és lidar, vagy akár videószenzor validálására egy méréselemzés közben, ezt manuálisan, irodai dolgozók elemzik és dokumentálják. Ezt gyorsítandó a megoldásom aminél ezeket az objektumokat előre szegmentáljuk, így gyorsítva és könnyítve ezt a feladatot, azzal hogy előre felcímkezzük őket.

2. Konvolúciós Neurális hálók

Az alábbiakban egy konvolúciós háló fontos és rá különösen jellemző építőelemeit mutatom be.

2.1. 2D diszkrét Konvolúció

A 2D konvolúció a neurális hálókban egy olyan művelet, amelyet az input tenzoron (ebben az esetben egy képen) hajtanak végre egy kernel (vagy szűrő) segítségével. A kernel egy kis méretű súlyozott mátrix, amelyet végigmozgatnak az input tenzoron, és a súlyozott összegeket számolják ki a megfelelő részknél (lásd 1. ábra). Ezáltal a konvolúció révén újabb tenzor (feature map vagy jellemzőterkép) keletkezik, amely tartalmazza az input tenzor különböző jellemzőit. Számolása az alábbi módon történik:

- **I:** Az input tenzor (kép).
- **K:** Kernel(szűrő) mátrix.
- **O:** output tenzor (feature map).
- **(i,j):** tenzorindexek, input és output tenzorok cellapozíciói.

$$O[i, j] = \sum_m \sum_n I[i + m, i + n] \cdot K[m, n] \quad (1)$$

A O tenzor első két dimenziója a pixel (vagy annál már nagyobb egység a képen), a

3 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

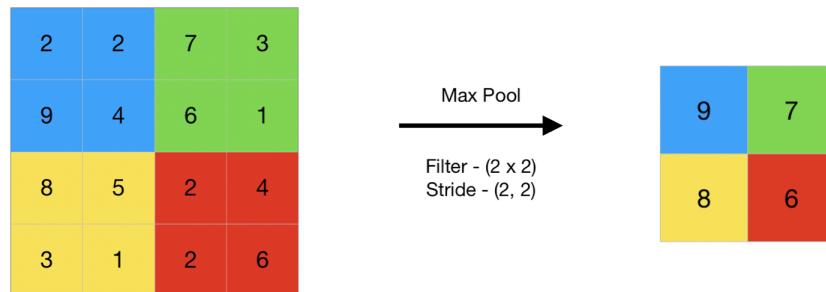
1. ábra. Mátrix és kernel

harmadik pedig a pixel színcsatornái (általában 3). Egy konvolúciós réteg olyan egységeket tartalmaz, amelyek mezői az előző réteg egy-egy területét(2x2,3x3,5x5) fedik le. Az ilyen egység súlyvektorát (az adaptív paraméterek halmazát) gyakran nevezik szűrőnek.

2.2. Pooling

A Pooling (a 2. képen látható) réteget a bemenet térbeli dimenzióinak (szélességének és magasságának) csökkentésére használják, miközben megőrzik a mélységet (azaz a csatornák számát).

Leggyakoribb verziója a Max pooling, ami előre meghatározott területen számol egy a területen belüli maximum értéket az elemeken és a következő, kisebb dimenziószámú mátrix értéke lesz.



2. ábra. MaxPooling

2.3. Teljesen összekapcsolt réteg

Teljesen kapcsolt réteg (Fully Connected, Dense, Linear Combination): előállítja a bemenetek és egy tárolt súlymátrix lineáris kombinációját: $H = XW + b$, ahol X a bemeneti mátrix, W a súlymátrix, b egy opcionális eltolósúly-vektor.

2.4. Batch normalizálás

A batch normalizációs rétegek segítik a túltanulás megelőzését és normalizált batchek mellett megfigyelhető a háló tanulási sebességének javulása is, ugyanis a Neurális hálók az információt "kötegekben" vagyis batch-ekben dolgozzák fel, így ez a réteg egy köteg átlagát és szórását egységesen normalizálja az alábbi módon:

$$\text{Átlag: } \bar{x} = 1/m \sum_{i=1}^m x_i$$

$$\text{Szórás: } \rho = 1/m \sum_{i=1}^m (x_i - \bar{x})^2$$

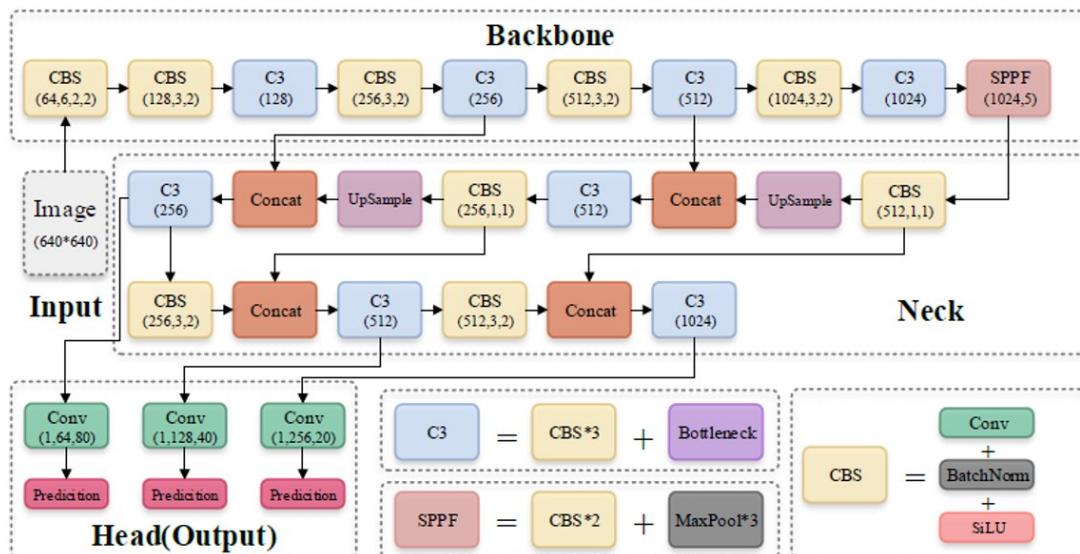
$$\text{Normalizált aktivációk: } yi = (x_i - \bar{x}) \sqrt{(\rho + \epsilon)}$$

Átskálázott és eltolt aktivációk: $z_i = \gamma y_i + \beta$, ahol γ és β tanult paraméterek

3. Yolov5 és gyakorlati alkalmazása

3.1. YOLOv5 és használt verziók

A Yolo (You only look once) algoritmus egy Joseph Redmon, Santosh Divvala, Ross Girshick és Ali Farhadi által kitalált algoritmuson alapuló neurális háló amiből a félév során a Yolov5-öt (arhitektúra a 3. ábrán) használtam amit az Ultralytics készített. A Yolo egy egészen új megközelítés a computer vision világában. Működés közben a képet rácsrendszerre osztja, és minden egyes rács a saját területén lévő objektumokat érzékeli. Innen is ered a név, egy lépésben végzi az objektumdetekciót és a lokalizációt is, tehát "Egyszer néz rá". A háló tanító bemenetére elvárja a képet és egy txt szövegfájlt amiben a képen található objektumok találhatóak id, középpont, szélesség, és hosszúság, szerint.



3. ábra. Yolov5 architecture from Ultralytics github

A továbbiakban a Yolov5-öt használtam az Ultralytics github repository-jából töltöttem le: <https://github.com/ultralytics/yolov5> A Yolov5 [7] hálónak öt ismert verziója van.

Ezek a különböző verziók méretükben és így effektivitásukban és sebességükben térnek el egymástól:

1. Yolov5n (nano) 1,9 millió paraméter,
2. Yolov5s (small) 7,2 millió paraméter.
3. Yolov5m (medium) 21,2 millió paraméter.
4. Yolov5l (large) 46,5 millió paraméter.
5. Yolov5x (extra large) 86,7 millió paraméter.

Ezek közül hardveres megkötések miatt a Yolov5s-et választottam, ami bár alulteljesít a nagyobb hálóknak, de a céljainknak megfelel, hiszen ez a háló bár előretanított a COCO adathalmazon (ez egy sokosztályos adathalmaz, nem feltétlenül közlekedési objektumokra, 80 objektuma között állatok és hétköznapi tárgyak) rendelkezik egy valamilyen szintű kompetenciával a közlekedéi objektumok detektálására.

3.2. Adathalmaz, Cityscapes

Ezt a hálót tanítottam felül hogy az általam kitűzött cél elérésére, a közlekedési objektumok detektálására. A tanításához használtam fel a "Cityscapes dataset"[1]-et. Hogy a hálónk effektívebb legyen csak 8 osztályt hagytam meg abból inkább 6 osztály észlelésere tanítottam felül, 8 osztály az eredeti adathalmazból (*gyalogos, autó, biciklis, motoros, busz, vonat, kamion, ismeretlen*) de a Cityscapes-ben csak ebből 6 található (*gyalogos, autó biciklis, motoros, busz, kamion*), erre felültanítottam, a másik adathalmaz a **CityScapes** annak is a "*gtfine*" nevű finoman annotált adatbázisával, aminek az osztályai, és a COCO adathalmaz osztályainak a metszete hat osztály (*gyalogos, autó, biciklis, motoros, kamion, busz*). A finom annotáció ebben a helyzetben azt jelenti hogy az objektumokat egymástól elszegmentálták és a 'corse' annotációhoz képest lényegesen több pontot használtak poligonok leírásához, amik a szegmentációs maszkokat írják le. Az adatbázis annotált és nyers képeket tartalmaz közúti/városi helyzetekről, több évszakban (tavasz, nyár, ősz), változó fényviszonyok között és különböző időjárási viszonyok között lett felvéve egy fedélzeti kamera (dashcam) szemszögéből. A szegmentációs maszkokat minden képhez egy ".json" kiterjesztésű fájl tárolja, ezekben az objektumok poligonokként. Ezek a poligonok véges számú pontjaival vannak leírva (és a pontok x és y pozíciójukkal megadva, ezek esetén a mértékegységet a kép adott tengelyen való kiterjedésében adták meg) Ezek az objektumok el vannak látva egy id-vel ami az objektum osztályát hordozza információként.

3.3. Formátumkonverzió

Az előbbi bekezdésekben megérthettük hogy az adathalmaz és a háló bemenete nem ugyanazon a formátumon van ($\text{poligon} \neq \text{bounding box}$), ezért írni kellett egy átalakítót ami ezt az átalakítást, gyorsan és megbízhatóan elvégzi, erre egy egyszerű algoritmust írtam.

```
for v in "poligon pontjai":
```

```
    maxx=max(v.x,maxx)  
    //négyzet legjobboldalibb pontja  
  
    minx=min(v.x,minx)  
    //négyzet legbaloldalibb pontja  
  
    maxy=max(v.y,maxy)      //négyzet legfelső pontja  
  
    miny=min(v.y,miny)      //négyzet legsósó pontja
```

Ezekből a pontokból csinálunk egy téglalapot az alábbi módon határozzuk meg:

```
for v in "poligon pontjai":
```

```
    width=maxx-minx  
    //szélesség két szélso x koordináta különbsége  
  
    height=maxy-miny      á  
    //magasság a két szélso y koordináta különbsége  
  
    centerx=(maxx+minx)/2 //a középpont a két szélso  
  
    centery=(maxy+miny)/2 //koordinátapáros átlaga
```

3.4. Tanítás és validáció, adatok kiértékelése

A Yolov5s modellt felültanítottam a már hálónak megfelelő formátumra, írtam egy gyors adathalmaz leíró '.yaml' kiterjesztésű fájlt az adathalmaz helyéről és megkezdtettem a tanítást.

```
$ python train.py --img 1024 --epochs 10 --data Traffic.yaml --weights yolov5s.pt --batch-size 4
```

Tehát a tanítást a yolov5s.pt-n végeztem 1024 pixeleses felbontáson, 10 epochon keresztül végeztem, és egyszerre a háló 4 képet kapott, ezt írja le a fenti parancs. A tanítás 13 órát vett igénybe és sikkerrel zárult, a yolov5 háló elkészítette a validáció eredményeit bemutató infografikákat:



4. ábra. Confusion matrix

A confusion matrix kiértékelésekor azt tatláltam hogy az autókra is gyalogosokra kimondottan érzékeny a háló, ahhoz hogy ezt megértsük látnunk kell a tanító adathalmazt és azon belül azt hogy milyen a tanító adathalmaz osztályainak eloszlása.

Ahogy láthatjuk a gyalogosok és az autók túlnyomó többségben vannak így túlreprezentáltak és eltorzítják az érzékenységét a hálónak az irányukba.

3.5. Megoldás

A problémát az érzékenységi egyensúly megtalálása jelenti. Mivel a gyalogosok és autók túlnyomó többségben vannak, z YOLO háló érzékenyebbe válik rájuk. A megoldás lehet a tanító adathalmaz újraegyenossága.

Az egyensúlyhiány kiküszöböléséhez több módszer is alkalmazható:

- Adatmódosítás:** Több képet kell beszerezni olyan osztályokról, amelyek alulreprezentáltak (például buszok), hogy a háló ne favorizálja a felülreprezentált osztályokat.
- Szemantikus szegmentáció:** Finomhangolhatjuk a YOLO hálót szemantikus szegmentációra, ami egy sokkal pontosabb, adatprezentációs forma.
- Nagyobb hálózat:** Nagyobb YOLO verziókra tanítás (pl. YOLOv5m, YOLOv5l, YOLOv5x), melyek több paraméterrel rendelkeznek és lehet, hogy jobb eredményeket érünk el használatukkal.
- Új erőforrás beszerzése** Új(hardveres) erőforrás és jobb szoftveres kiszolgálás beszerzése és felépítése

II. rész

Yolov8-as szegmentációs háló és annak magyarázhatósága EigenCAM típusú modellfüggő magyarázó rendszerrel

1. Bevezetés

Közlekedési objektumok detektálásával foglalkoztam a tavalyi témalaboratóriumban (I), gépi látás (Computer Vision (CV)) témakörében, mindezt Yolo architektúrájú neurális hálóval. Akkor a YOLOv5-tel foglalkoztam hagyományos keretes adatreprézentáció mellett és a dolgozatom végén haladási tervet fogalmaztam meg azért, hogy jobban elmerüljek ebben a területben (lásd a 11. oldal listájában). Ezek akkor az alábbiak voltak:

- a. újabb hálóarchitektúra vagy nagyobb háló alkalmazása
- b. áttérni a szemantikus szegmentációs objektumreprézentációra a bounding box alapúról
- c. új (hardveres) erőforrás és jobb szoftveres kiszolgálás beszerzése és felépítése

A két félév között viszont megismerkedtem a modellmagyarázási módszerekkel így hozzáadódott az alábbi pont:

- d. a háló döntéseit milyen magyarázó módszer mellett tudnám lehető legérthetőbben megjeleníteni és ezáltal megérthetővé tenni.

Előző féléves szakmai gyakorlatomban, képfeldolgozó neurális hálókkal dolgoztam. Itt feladatom volt megismerkedni az adatfeldolgozással, Machine Learning Operation (MLOps)-al is ami, a mesterséges intelligenciával segített szoftverek kiszolgálását és fejlesztését könnyíti. Ekkor szembesültem azzal, hogy a neurális hálók azonban gyakran fekete dobozként (blackbox) működnek, ami komplikálttá teszi azok megértését. Azért, hogy biztonságosabbnak tudhassuk ezeket és megértésükkel későbbi hibáikat kijavíthassuk, fontos interpretálhatóság. Az interpretálhatóság nem más mint a modell döntéseinek megértése és magyarázata, ami kulcsfontosságú ezen modellek elfogadhatóságában és alkalmazhatóságában.

A YOLOv8 (You Only Look Once version 8) egy népszerű és hatékony konvolúciós neurális háló. Ennek a szegmentációs változatát használom (Yolov8m_seg), ez például objektumfelismerésre és objektum-klasszifikációra is tökéletesen használható. A modell architektúrája ismeretében betekinthetünk majd valamilyen külső eszközzel a héjjai közé, és megróbálhatjuk megérteni a héjjak (layerek) közötti aktivációs függvények kimenetei alapján, a háló viselkedését.

Egy ilyen eszköz az EigenCAM (Eigen Class Activation Mapping) ami egy modell-függő és gradiensmentes magyarázó rendszer, amely képes vizualizálni, hogy a mély tanuló hálózatok mely részei járultak hozzá a döntéshozatalhoz. Ehhez pedig az aktivációs függvények értékeit kiemeli a háló héjjaiból és a bemeneti képre vetíti ezeket, majd kombinálja ezt a mátrixot és az eredeti képet eggyé, megmutatva azt, hogy a kép mely részei milyen fontosak a döntések meghozásában.

2. Mély tanulás és interpretálhatóság

A mesterséges intelligenciát tartalmazó szoftvereket azért használjuk gyakran, hogy rugalmasabb megoldást adjon akár nehezen algoritmizálható problémáinkra is. Azonban ezek működése nehezen nevezhető zártnak és kauzálisnak. Ezért elengedhetetlen az interpretálhatóság, vagyis annak képessége, hogy a modellek döntéseit érthető és ésszerű módon magyarázza meg. Különböző magyarázó módszerek léteznek a mély tanulási modellek interpretálhatóságának növelésére.

2.1. Magyarázhatóság

A gépi tanulásban használt magyarázó módszerek két fő kategóriára oszthatók: modellfüggő és modellfüggetlen. A modellfüggő magyarázó módszerek közvetlenül figyelembe veszik a modellek belső szerkezetét és működését a magyarázatok létrehozásában. Ezek a módszerek arra törekednek, hogy feltárrák a modell döntéshozatali folyamatainak mechanizmusait és az egyes predikciók alapját. Például a gradiens visszaszámítatás, osztály-aktivációs térképek(4.fejezetben bővebben) és a LIME modellfüggő magyarázó módszerek.

Azonban a modellfüggetlen magyarázó módszerek általánosabb megközelítést alkalmaznak az interpretálhatóságra. Ezek a módszerek nem igénylik a modell belső szerkezetének ismeretét a magyarázat létrehozásához, és általában a bemeneti adatok és a modell predikciói közötti kapcsolatokat vizsgálják. Például a perturbációs alapú módszerek, mint például a SHAP és a LIME együttes használata.

1. KÉRDÉS :Miért is ilyen fontos a modell interpretálhatósága és a magyarázhatósága?

1. **Válasz:**: Különösen fontos a jogi szabályozásban, hogy bármilyen szoftver-termék használatakor a döntések átláthatóak és érthetők legyenek. Például az Európai Unió (EU) által 2016-ban életbe léptetett Általános Adatvédelmi Rendelet (Általános Adatvédelmi Rendelet (GDPR)) előírja, hogy az automatizált döntéshozatalnak átláthatónak kell lennie, és az érintetteknek joguk van tudni, hogy egy algoritmus milyen döntéseket hoz róluk.

3. YOLOv8: Szemantikus szegmentációs háló és működése



5. ábra.
Ultralytics
logo

A YOLOv8 ([8]. hivatkozás alatti repository) egy hatékony, népszerű osztályozó és detektáló neurális hálózatsalád legújabb példánya, amelyet számos számítógépes látás (CV) feladatban használnak. A "You Only Look Once" (YOLO) megközelítést alkalmazza, amely gyors és pontos objektumdetektálást tesz lehetővé egyetlen neurális háló segítségével.

A YOLOv8 működése során a bemeneti képet egyszer veszi figyelembe, és az objektumok pozícióját és osztályát egyetlen predikcióval határozza meg. Ez a modell különösen alkalmas valós idejű alkalmazásokhoz, mint például az önvezető járművek vagy a valós idejű videoelemzés.

Ennek a hálónak én a szegmentációs változatát használtam, amely manapság egy elég új és népszerű irány a közlekedési objektumok detektálásában. Eddig ugyanis inkább 2–3 dimenziós ún. "bounding boxokat" azaz kereteket használtak a detektálni kívánt objektumok reprezentációjára, azonban a szegmentációs hálók képesek ezeknek pontosabb azonosítására és lokalizációra. Az információ-gazdagabb adatreprézentáció miatt, az objektumok pontos körfonalainak meghatározása lehetővé válik. Ezáltal a szegmentációs hálók pontosabb és részletesebb információkat nyújtanak az objektumról, mint a keretes objektum-interpretáció.

3.1. Yolov8 architektúra

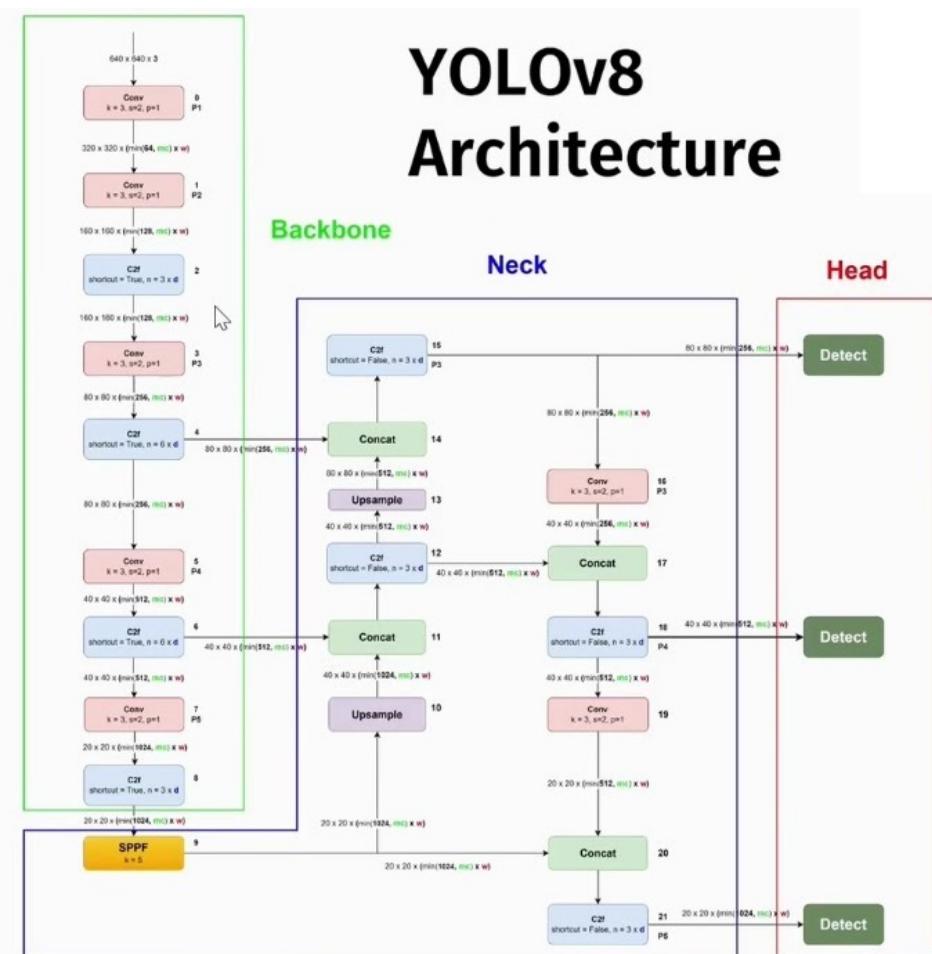
A YOLOv8 egy képfeldolgozó (konvolúciós) mély neurális hálózat. Ami három különböző réteget használ a hálóarchitektúrában a képek feldolgozásához:

- konvolúciós rétegeket
- teljesen összekapcsolt rétegeket
- "MaxPooling" rétegeket,

A háló bemenetién a képet konvolúciós és "MaxPooling" rétegeken keresztül feldolgozza, majd a kimeneti rétegekben meghatározza az objektumok pozícióját és osztályát, mindezt egy iteráció alatt, innen is jön a "You Only Look Once" elnevezés.

A YOLOv8 architektúra 3.1 általában három részre osztható:

1. Az előfeldolgozó rétegek, amelyek a bemeneti képet előkészítik a további feldolgozásra.
2. A konvolúciós rétegek, amelyek a képet feldolgozzák és az objektumokat azonosítják.
3. A kimeneti rétegek, amelyek meghatározzák az objektumok pozícióját és osztályát.



A YOLOv8 architektúra eltérő méretű és kapacitású változatokban érhető el, amelyek különböző feladatokhoz és alkalmazásokhoz használhatóak: (Attribútumok COCO adathalmaz alapján) Ezenfelül beszélhetünk szegmentációs hálókról (bővebben: a 3.2 alfejezetben) és hagyományosan a bounding boxokat detektáló hálókról, amik az objektumok köré illesztenek egy befogó minimális területű téglalapot/téglatestet.(Yolov8_seg vs. Yolov8)

YOLOv8 verzió	Méret	Max. Pontosság (mAP)	Sebesség (ms)
YOLOv8s	Kis	37.3	0.99
YOLOv8m	Közepes	44.9	1.2
YOLOv8l	Nagy	50.2	2.39
YOLOv8X	Hatalmas	53.9	3.53

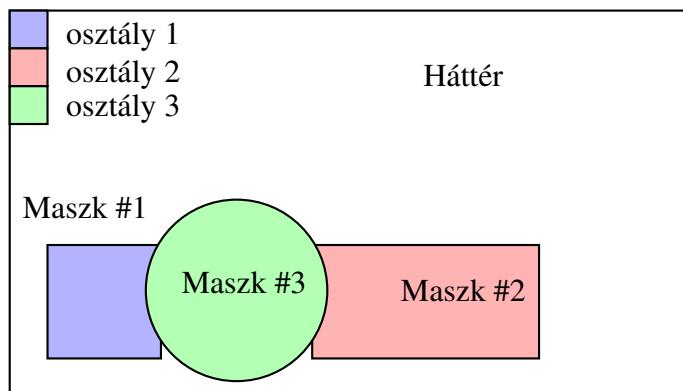
Témalaboratórium alatt bounding box adatrepräsentációjú neurális hálót használtam, most viszont, áttértem a szemantikus szegmentációra. Ennek okait a következő alfejezetben felsorolom.

3.2. Szemantikus szegmentáció

A szemantikus szegmentáció egy olyan adatrepräsentációs forma, amelyben a cél az objektumokat tartalmazó kép egyes részeinek (például pixeljeinek) címkézése az objektumokhoz tartozó osztályok szerint. Van emellett egy másik szegmentációs adatrepräsentációs forma, az ún. egyed szegmentáció Egyedszegmentáció, amelyben minden objektumot külön, azaz egyedenként kell detektálni és akár számon tartani (indexelni), míg a szemantikus szegmentációban csak az objektumok osztályait kell meghatározni az egyed megkülönböztetése nélkül. Más szavakkal, minden képpontot hozzá kell rendelni egy osztályhoz vagy kategóriához, például autó, biciklis, gyalogos stb. Szakmai gyakorlatom során eddig csak szemantikus szegmentációval foglalkoztam. Amely módszer lehetővé teszi a rendszereknek, hogy pontosan azonosítsák és lokalizálják az objektumokat egy adott képen.

Tehát a szemantikus szegmentációt leírhatjuk úgy, ha egy képet jelölünk I -vel, akkor a szemantikus szegmentáció pedig egy olyan függvény, $F : I \rightarrow L$, ahol L a lehetséges osztályok halmaza, és F minden képpontot hozzárendel egy osztályhoz, ez az osztály-hozzárendelés lesz a maszk. A szemantikus szegmentáció kiemelt fontosságú az önvezető autók, a videoelemzés, a térképek építése és sok más alkalmazásban, ahol pontos és részletes objektum-felismerésre van szükség. Tehát az adatrepräsentációs módszerváltást azért tartottam fontosnak:

1. mert pontossága jóval felülmúlja az előző módszerét.
2. mert az adathalmaz alapvetően szemantikus szegmentálási módszerrel van annotálva, kevés adat-előkészítési munkálat szükséges.
3. mert könnyebben érthető a feldolgozott kép, nincs olyan hogy az objektumok túlzott sűrűsége miatt túl sok doboz keletkezik ami megnehezíti vagy akár ellehetetlenítheti a kiolvasást.



6. ábra. szegmentáció példa

3.3. Adathalmazok és kísérletek

Adathalmazként a témalaboratórium alatt végzett munkámhoz hasonlóan a CityScapes (<https://www.cityscapes-dataset.com>) szemantikusan szegmentált adathalmazt, annak is a pontosan annotált (Fine Annotated), a 7.képen bemutatott, adathalmazát használtam, ami osztályszegmentációs maszkokat biztosít az elérhető képekhez. Ezek a képek kicsit több mint 5000 városi közlekedési szituációt ábrázolnak németországi városokból.

Ez az adathalmaz elég nagy varianciával rendelkezik számunkra ahhoz, hogy az adathalmazból egy robusztus szegmentációs hálót tudjunk tanítani közlekedési objektumok detektálására és osztályozására.

A tanításhoz és validációhoz a maszkokat poligonok formájában kaptam meg szöveges (.yaml) formában, ezeket a poligonokat minimális átalakítás után be is tudtam adni a háló bemenetére a megfelelő képekhez rendelve (szegmentációs maszkokról: 3.2. oldalon értekezlek.) A kísérletek az 1. táblázatban találhatóak.

A magyarázat a kísérletekhez a 24. oldalon található.

Ezeket az adatokat használom a későbbiekben arra, hogy megpróbáljam megérteni a hálónkat az aktivációs függvényeinek kimenetei alapján. Ezeket a teszteket amiket az EigenCAM-mal végeztem, a németországi Bonn városában vették fel és mivel a teszt-halmaz része, így a háló tanításban és a validációban eddig nem szerepelt.

3.4. MLOps

A háló tanításához és validálásához használtam egy online kiértékelő, és eredményösszesítő felületet, ez pedig a Comet.ml online platform ("alternatívája a Weights & Biases"-nek), amely egy Github fiókhoz rendelve, segít a Machine Learning (ML) projektek adminisztrációjában. A Comet.ml (<https://www.comet.ml>) egy kollaboratív MLOps platform, amely lehetővé teszi, hogy könnyen és hatékonyan figyelhessük



7. ábra. CityScapes Examples

a futó tanításainkat, validációinkat és inferenceinket. Számos funkciót kínál:

- Rögzíti és követi a kísérletek metrikáit, hiperparamétereit, modellváltozatokat és naplófájait.
- Különböző diagramok és grafikonok segítségével jeleníti meg és elemzi az adatokat és az eredményeket.
- Lehetőséget ad a különböző kísérletek összehasonlítására és megosztására, akár csapatmunkára is használható.
- Automatikusan rögzíti és követi a modell változásait és fejlődését a tanítás folyamán az idő műlásával.
- Integrálható más keretrendszerrel és eszközökkel, és rendelkezik egy API-val is, amely lehetővé teszi a platform saját igényeiknek megfelelő testreszabását.

Összességében a Comet.ml egy teljes körű platform, amely segíti a ML folyamatok hatékony kezelését és nyomon követését, valamint lehetővé teszi a felhasználók számára, hogy gyorsabban és hatékonyabban dolgozzanak valamint jobban manageljék kísérleteiket.

4. EigenCAM: Modellfüggő magyarázó

Az EigenCAM (Eigen Class Activation Mapping) egy modellfüggő magyarázó módszer (forrás [2]:), amelyet a mély tanulási modellek interpretálhatóságának növelésére fejlesztettek ki. Célja, hogy vizualizálja és magyarázza meg a modellek döntéseit a bemeneti adatok alapján, osztály-diszkrimináció nélkül, tehát ebben az esetben nem lesznek külön osztályokra szedve az aktivációk, hanem az összes osztály átlagos aktivációját kapjuk meg.

Az EigenCAM működése során az algoritmus végiglépked a háló kiválasztott héjain és kiértékeli az aktivációs függvények mátrixát. Ezt a mátrixot kiterjeszti (ha kell) annak a képnek méretére amire éppen az inferenceet futtatjuk. (forrás [4]:) Az ez által generált "hőterkép" (Heatmap) már jó esetben lehetővé teszi azt, hogy közelebb kerüljünk annak megértéséhez, hogy a kép mely tartományai játszanak nagyobb szerepet a detekcióban és klasszifikációban.

Azonban fontos tudni, hogy az EigenCAM csak egy interpretálhatósági eszköz, és nem biztosít teljes képet a modell működéséről. Ugyanis ezek az eszközök csak a hálónak egy szeletébe engednek betekintést nyerni, amik önmagukban nehezen értelmezhetőek. Sokszor ez nem is enged logikus következtetést levonni.

Az aktivációs csoportok képenként újrarendeződnek így metszeteket is nehéz automatizáltan készíteni. Ez pedig elengedhetetlen annak érdekében hogy biztosra mondjuk egy következtetés magyarázását.

4.1. Aktivációk

Az aktivációs függvények a modell héjainka kimeneti függvényei ahol "x" az input, "W" a súlymátrix és "b" a "bias" vagyis eltolási vektor. Az utolsó héjban egy lineáris aktivációs(Rectified Linear Unit (ReLU)) (3). függvényt használunk, míg az összes többi héjban egy szivárgó rektifikált lineáris aktivációs függvényt(Leaky Rectified Linear Unit (LReLU)) (2) használunk:

$$\text{LReLU: } \varphi(x) = \begin{cases} x, & \text{ha } x > 0 \\ 0.1x, & \text{különben} \end{cases} \quad (2)$$

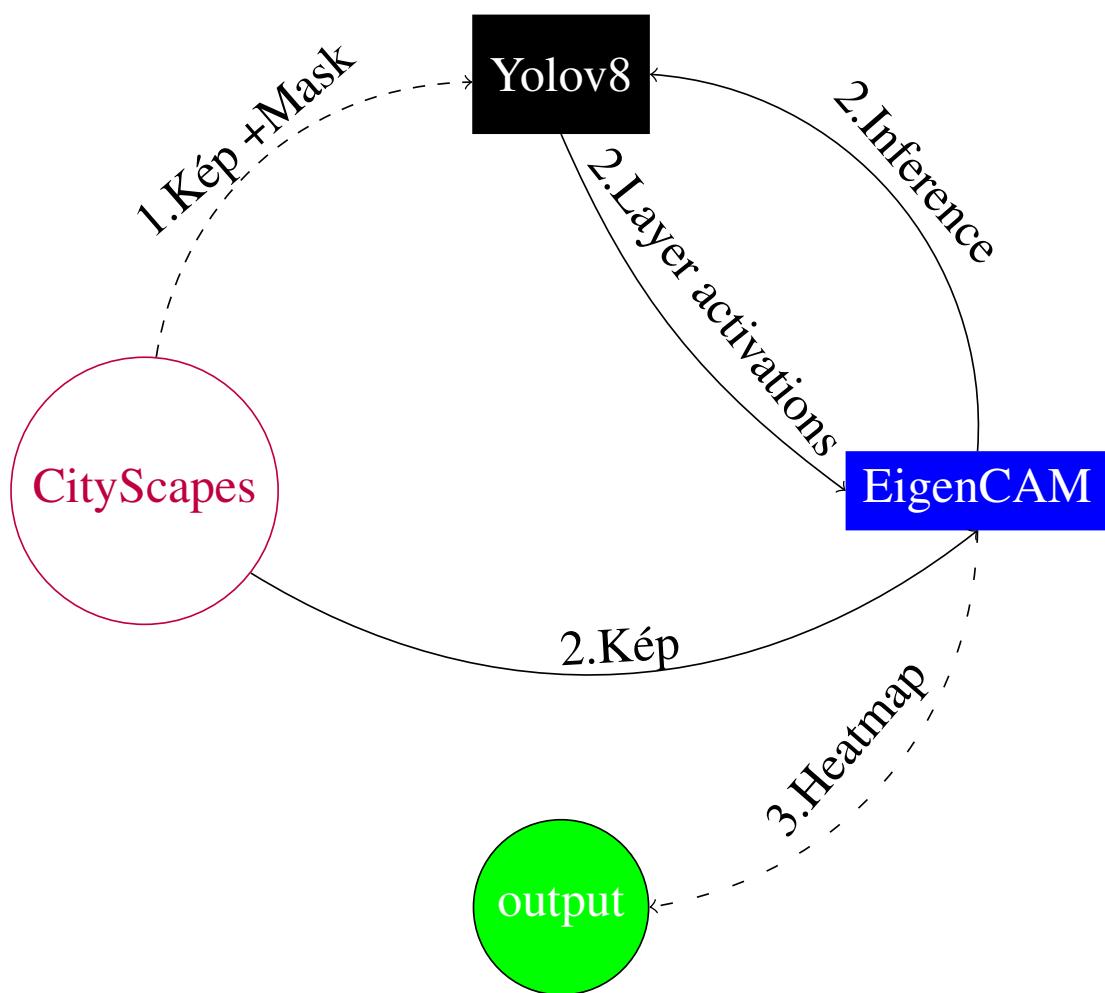
$$\text{ReLU: } \varphi(x) = \begin{cases} x, & \text{ha } x > 0 \\ 0, & \text{különben} \end{cases} \quad (3)$$

$$\text{Sigmoid: } \varPhi(x) = \frac{1}{1 + e^{-x}} \quad (4)$$

5. Az EigenCAM alkalmazása a YOLOv8-ra

Az EigenCAM alkalmazása a YOLOv8 szemantikus szegmentációs hálóra lehetővé teszi számunkra, hogy megértsük miképpen azonosít és lokalizál objektumokat képeken és videókon a modell. ([6]. hivatkozás repository-ja alapján).

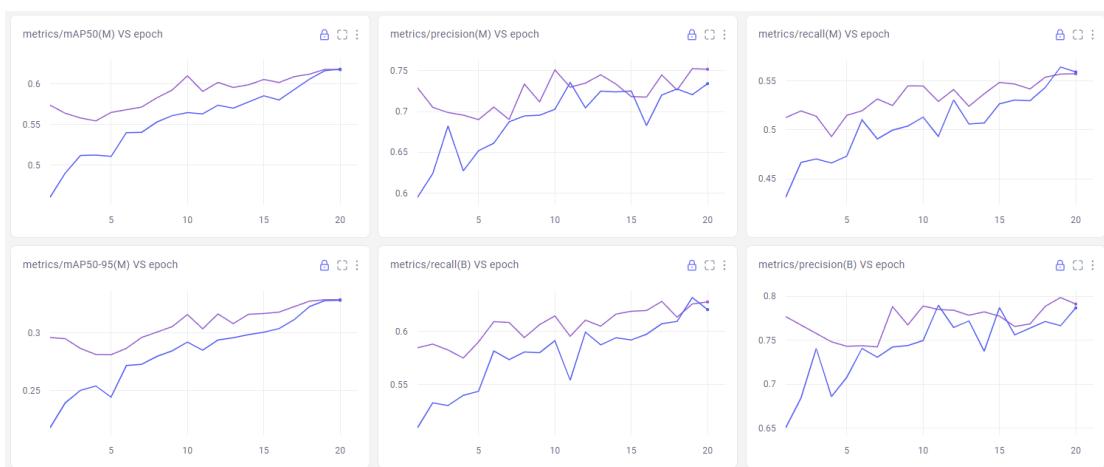
Az EigenCAM konkrét működési folyamata az 5 ábrán látható. A CityScapes adathalmazon tanítottuk a Yolov8m-seg hálót (ami egy közepes méretű Yolov8 szegmentáló háló), majd az EigenCAM segítségével vizualizáltuk az aktivációkat, amelyeket a háló a képek feldolgozása közben produkál (ezt az EigenCAM adja a hálónak egy olyan folyamatban keresztül amit inferencenek nevezünk). Ezeket az EigenCAM kivezeti és rávetíti a bemenő képre (ezt a folyamatot lásd az 5 képen észak 5 pontban), a különböző héjak aktivációit külön-külön. Amikor végez a kép előállításával azt HEATMAP formájában az output mappájába helyezi. Ezeket a heatmapokat majd 23. oldalon láthatjuk.



- Normál Szaggatott: Tanítási időben.
- Teli nyíl: inference (EigenCAM használata) közben.
- Laza szaggatott nyíl: inference idő után.

5.1. Implementáció lépései

1. Github és Comet repository létrehozása és felkonfigurálása
2. Tanító, validációs és detektáló állományok megírása
3. Adatfeldolgozás, és adatbázis-management
4. Tanítás és validáció
5. Modellanalízis
6. Modellmagyarázó módszer bevetése
7. Modellmagyarázó módszer kimenetének értékelés és magyarázata



8. ábra. A tanítás folyamata

6. Magyarázhatosági eredmények és értékelés

A két hálót, legyen az egyik ”old” a másik ”new”, ”old” egy yolov8m-seg háló 20 epoch keresztől tanult a ugyanazon teljes adathalmazon, a másik pedig 40 epoch keresztül tanítottam.

Az EigenCAM által nyújtott kimenet amit ábrázoltam az 1. táblázatban. A két háló detekciója alig különbözik. Kizárolag abban különböznek, hogy milyen bizonyosságban képesek megmondani átlagosan az objektumok osztályát. Ezért elegendőnek találtam a következőkben csak az ”new” háló eredményeit tárgyalni.

Layer	”old”	”new”	detection
-4.			
-3.			
-2.			

1. táblázat. Héj aktivációk összehasonlítása a két háló között.

6.1. Magyarázat

Nem megszokott módon, a heatmapeken az alacsony aktivációs értékeket vöröses árnyalatokkal míg a magas aktivációs értékeket kékes árnyalatokkal jelölik.

Layer	Magyarázat
-4	Ez a réteg egyetlen konvolúciós réteget tartalmaz. Ami nagyrészt eldetektícióval foglalkozik, így az aktiváció nem konzisztenčen jelzi a detektálni kívánt osztályokat. Inkább elekre aktiválódik amelyek az osztálydetekcióban hasznosak lehetnek ezeket featureökké fogja össze és ezeket a featureöket továbbítja a következő hálónak. Az ennél előrébb lévő rétegek, túl kis méretű featureökkel foglalkoznak és ezek közül sokkal kevesebb lesz használva, viszont a sok aktiváció miatt emberi szemmel keveset érhetünk belőle. Túl keveset ahhoz, hogy a magyarázhatóság vizsgálatába belevegyem, hiszem emberi szem által nehezen olvasható és érthető.
-3	Ez egy konkatenációs réteg itt a rétegek előzőleg különböző méretű bemeneteket dolgoznak fel. Ez a réteg valószínűleg az összes feldolgozott adatot egyetlen tensorba egyesíti, hogy aztán további feldolgozásra kerülhessen. Persze az, hogy hol történik feature detekció (hol nagyobb az aktiváció) az jól látszik a képeken. Ezek a fontos összefűzött feature-k melyek a következő rétegek is fontosak lehetnek.
-2	Ez a réteg konvolúciós rétegeket (cv1 és cv2) tartalmaz, amelyeket egy ModuleList (m) követ, amely két Bottleneck blokkot tartalmaz. Ezek a blokkok az adatok dimenzióinak csökkentésére és a reprezentációk összeállítására szolgálnak, lehetőséget adva az egyszerűbb és hatékonyabb feldolgozásra az utolsó réteg számára. Az utolsó héj kimenetei maguk az osztályaktivációk, ezek egyszer sem produkáltak olyan heatmapet amit érdemes lett volna vizsgálni, a kijövő kép "horizontális vonalkód" szerű, így azt nem vettet figyelembe a magyarázhatósági vizsgálatba.

2. táblázat. Magyarázat

Ezen felül az EigenCAM által nyújtott magyarázatokat összehasonlítottam a két hálóra vonatkozóan. Az "old" háló kimenetén jól láthatjuk a gyengébb aktivációkat, míg "new" háló centralizált aktivációkat mutat ott, ahol ténylegesen fel kell ismernie az objektumokat. Ezen felül még egyértelműen látható, hogy a tanulással arányosan csökken az aktivációk értéke olyan helyeken, ahol csak valamiféle tetszőleges él található mint például a -4. Layer estében. Tehát egyértelműen kijelenthetjük hogy a háló tanulását kitudom mutatni kizárálag csak az aktivációk alapján, az EigenCAM segítségével.

7. Az EigenCAM alkalmazásának gyakorlati haszna

Az EigenCAM és hasonló magyarázó rendszerek alkalmazása esélyt adhatnak nekünk arra, hogy esetekre , tehát egyéni bemenetekre (a képfelismerés esetében ezek a képek), vonatkozó döntéseihez a kép minden részei milyen fontossággal asszisztáltak. Lehet értelmezni a háló architektúrájának tudatában, hogy éppen melyik réteg milyen információt dolgoz fel és ezekből milyen hiedelmeket (értelmezett információkat) alkot. Érdekes az is, hogy mint ahogy 24. oldalon láthattuk a háló tanulását is meg tudjuk figyelni az aktivációk alapján és ebből következtetéseket tudunk levonni arra vonatkozóan, hogy mik alapján észlelik az objektumokat, kiküszöbölv olyan hibákat, amelyeket a korai mesterséges intelligencia alapú CV szoftverek tartalmazhattak, például a háttér túlzott befolyása az objektum osztályának meghatározásában.

7.1. Eredmények összegzése

Féléves munkámban tehát a EigenCAM modellmagyarázó módszert és a YOLOv8 szemantikus szegmentációs háló interpretálhatóságát vizsgáltam valamint feladatom volt az adat előkészítése és elemzése is.

Az EigenCAM segítségével vizualizáltam az aktivációkat, és kísérletet tettem a hálók működésének megértésére. Ezáltal figyelhettem meg a háló tanulását: mit jelenthet adott esetben a túltanulás fogalma és azt, miben is különbözik egy háló aminek jobb az "Accuracy" és a "Recall" mutatója is, egy másik ugyanazon az adaton és osztályokra tanított, ugyanolyan architektúrájú hálótól.

Ezen felül foglalkoztam az új architektúrával, Yolov5-ről Yolov8-ra tértem át, és először használtam felhő alapú MLOps megoldásokat munkám segítésére, mint például a Comet (<https://www.comet.ml>) amelyben az ott végzett munkámat és szerzett tapasztalataimat a 3.4. fejezetben részleteztem.

8. Jövőbeli irányok és kutatási lehetőségek

A YOLOv8 neurális háló most is vezető szerepet élvez az autóipari szolgáltatásokban képfelismerés téma körében. Azon belül a szemantikus szegmentációban, ami az elmúlt években előnyt élvez a hagyományos két- és háromdimenziós keretekkel szemben, a kereteknél jóval nagyobb pontossága és a maszkok által hordozott temérdek információ gazdagsága miatt.

A következőkben tehát a jelenlegi munkámban még a kezdetlegesebb, kiforratalabb módszert, a modellmagyarázást (vagyis Explainable AI-t Explainable AI (XAI)) szeretné továbbfejleszteni. Mivel a jelenlegi megoldásom modellfüggő és héjaktiváció-alapú, nem vesz figyelembe gradienseket, valamint érzékeny a modell belső felépítésére is. Célom a jövőben a modellek tanulásának folyamatát figyelni más módszerekkel is, amellyek például a héjjak gradienseit figyelembe veszik, azért hogy a magyarázataik pontosabbak, rendszerszerűbbek és megbízhatóak legyenek. A következő konkrét tervet határoztam meg további munkámhoz:

1. Lime vagy Shap módszerek behozása és összehasonlítása az EigenCAM-mal. [5, 3]
2. EigenGradCAM bevezetése a modell tanulásának folyamatának figyelembevétele.

Tárgymutató

CityScapes, 8, 18, 21

Comet, 18

EigenCAM, 18, 20, 21, 23

szegmentáció

egyedszegmentáció, 17

szemantikus szegmentáció, 17, 26

Yolo

Yolov5, 7

Yolov8, 13

9. Szó- és rövidítés jegyzék

Szójegyzék

Eigen Class Activation Mapping Egy modellfüggő magyarázó rendszer, amely képes vizualizálni, hogy a mély tanuló hálózatok mely részei járultak hozzá a döntéshozatalhoz. 13

epoch Az Epochok száma adja meg hányszor megy végig a tanulás során a hűlő a teljes adathalmazon. 23

feature A feature egy adott kép vagy adat egyik jellemzője, amelyet a modell felhasznál a döntéshozatalhoz. 24

Inference Az inferencia a gépi tanulásban a modell által megtanult minták alkalmazását jelenti új adatokon. 19–21

You Only Look Once You Only Look Once, egy gyors és hatékony objektumdetektáló neurális hálózat. 15

You Only Look Once version 8 A YOLOv8 egy hatékony és népszerű szemantikus szegmentációs hálózatcsalád legújjabb példánya. 13

9.1. Rövidítésjegyzék

EU Európai Únió	14
GDPR Általános Adatvédelmi Rendelet	14
CV Computer Vision	13
XAI Explainable AI	26
MLOps Machine Learning Operation	13
ML Machine Learning	18
mAP Mean Average Precision	
ReLU Rectified Linear Unit	20
LReLU Leaky Rectified Linear Unit	20
ms milliszekundum	

Hivatkozások

- [1] Cityscapes. Cityscapes dataset. <https://www.cityscapes-dataset.com>, 2024.
- [2] jacobgil and Pytorch Contributors. GRADCAM implementation. <https://github.com/jacobgil/pytorch-grad-cam>, 2024.
- [3] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.
- [4] M. Bilal Muhammad and Mohammed Yeasin. Eigen-cam: Class activation map using principal components. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, July 2020.
- [5] Marco Ribeiro, Sameer Singh, and Carlos Guestrin. „Why Should I Trust You?“: Explaining the predictions of any classifier. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 97–101, San Diego, California, June 2016. Association for Computational Linguistics.
- [6] rigvedrs. Yolov8-CAM implementation. <https://github.com/rigvedrs/YOLO-V8-CAM>, 2024.
- [7] Ultralytics. Yolov5 github repository. <https://github.com/ultralytics/yolov5>, 2024.
- [8] Ultralytics. You Only See Once Project at Ultralytics. <https://github.com/ultralytics/yolov5>, 2024.

Formai elem	Megvalósítás
irodalomjegyzék	itt
tartalomjegyzék	itt
rövidítésjegyzék	itt
indexjegyzék	itt
táblázat	itt és itt
hivatkozás táblázatra	itt
vektor-grafikus kép	itt
raszter-grafikus kép	itt
hivatkozás képre	itt
tikz ábra	itt
hivatkozás ábrára	itt
képlet	itt
hivatkozás képletre	itt
képletcsoport	itt
hivatkozás képletcsoport egy képletére	itt
fejezet	itt
hivatkozás fejezetre	itt
lista	itt
hivatkozás lista elemre	itt
hivatkozás oldalszámra	itt
hivatkozás irodalomra	itt
saját makró használata	itt