

TDK dolgozat

Szerző: Dobay Tamás

Széchenyi István Egyetem
Gépészmérnöki, Informatikai és Villamosmérnöki Kar
Informatika Tanszék

**Objektum detektáló neurális hálózat
predikciós pontosságának vizsgálata
zajterheléssel**

Dobay Tamás
(IV. éves mechatronikai mérnök BSc hallgató)

Konzulens:
Hollósi János
(egyetemi tanársegéd)

Tartalomjegyzék

1. Bevezetés	2
2. Mesterséges neurális hálózatok.....	3
2.1. Mesterséges neuronhálózatok – ANN.....	3
2.2. Konvolúciós neurális hálózatok – CNN	5
2.2.1. Konvolúciós réteg	5
2.2.2. Összevonó réteg	7
2.2.3. Teljesen összekapcsolt réteg	8
2.3. Mesterséges neurális hálózatok tanulása.....	8
2.3.1. Hiba visszaterjesztés	9
2.4. Felhasznált adatkészlet.....	11
2.5. Felhasznált architektúra	12
2.5.1. Modell pontosságának kiértékelése	13
2.5.2. Modell felépítése.....	14
3. Felhasznált keretrendszer	16
3.1. PyTorch	16
4. A zajterhelések	19
4.1. FGSM (Fast Gradient Sign Method).....	19
4.2. Eső szimulálása	20
5. A kiértékelő adatkészletek létrehozása	20
5.1. A nyers adatkészlet	20
5.2. Zajterhelt adatkészletek.....	22
6. YOLOv5 kiértékelése	24
6.1. A modellek teljesítménye a nyers adatkészleten.....	24
6.2. A modellek teljesítménye az eső zajterhelt adatkészleten	28
6.3. A modellek teljesítménye az FGSM zajterhelt adatkészleten	32
7. Összefoglalás, a kapott eredmények kiértékelése	37
8. Irodalomjegyzék.....	39

Jelölések

$w_i^{(L)}$	az L rétegben i-edik neuron kapcsolat súlyértéke
$a_i^{(L)}$	az L rétegben az i-edik neuron kimeneti értéke
b	eltolássúly
ϕ	aktivációs függvény
s	konvolúciós kernel lépésköze
C	veszteség függvény
t	mérési alap értéke
K	konvolúciós kernel mátrixa

1. Bevezetés

A 20. század közepétől kezdve hatalmas fejlődésen mentek keresztül a gépi látás területén alkalmazott mesterséges neurális hálózatok. Az első konvolúciós neurális hálózatoktól, amelyek még csak egy képen szereplő objektumnak az éleit tudták meghatározni, mára már képesek pixelpontosan külön választani a különböző osztályokba tartozó objektumokat, és egyben lokalizálni is tudják azokat. Mivel már ilyen komplexitású problémát is megbírnak oldani, ezért különböző autonóm működésű berendezések és járművek szerves részeit képezik. Egy hasonló működéssel bíró képfeldolgozó szoftver pontosabb és részletesebb információt biztosít számukra, ezért bonyolultabb környezetekben is (mint például a közúti forgalom) pontosabb irányítást tudnak végrehajtani. Gyakran használt megoldás kamera alapú környezetérzékelésre az objektum detektáló neurális hálózatok, amelyek képesek a bemeneti kép alapján befoglaló téglalappal meghatározni bizonyos objektumok helyzetét és osztályozzák is azokat. Viszont ezeknek a hálózatoknak a megbízhatóságát könnyen befolyásolhatják akár az időjárási viszonyok (például a köd), vagy éppen a felismerendő objektum rongáltsága (például egy tábla össze van firkálva).

Dolgozatom célja megvizsgálni a YOLO objektum detektáló modell család legújabb verzióján (YOLOv5 [4][12]) keresztül, hogy egy hálózat tanulható paraméterszámától függően hogyan teljesít zajterhelt bemeneti képeken. Általában a nagyobb paraméterszámmal rendelkező hálózatok szoktak pontosabb predikciót adni a bemeneti adatokra. Vizsgálattal kimutatható, hogy ez zajterhelt bemenetek esetén is teljesül-e, és ha nem akkor milyen mértékben változnak az eredmények.

Ennek a modellnek a gyakorlati felhasználását a PyTorchon [9] keresztül fogom megvalósítani, ami a Python nyelvhez írt mély tanulási keretrendszer. Beépített függvényeinek köszönhetően a kiértékelést is egyszerűbben tudom majd végrehajtani az elkészített kiértékelő adatkészleteken.

A projektben kétféle zajterhelést használok, amit a COCO (Common Objects in Context) adatkészletből levágott kiértékelő adatkészletemen fogok alkalmazni. Az első módszer az esőnek a szimulációja, ami inkább egy gyakorlati zajterhelés, a második pedig az FGSM (Fast Gradient Sign Method) [6] támadás, ami pedig inkább egy elméleti.

Dolgozatom eredményeként, a kiértékelte mérési adatok nagyobb belátást adhatnak arra, hogy zajterhelt adatokon hogyan teljesít a mai egyik legkorszerűbb objektum detektáló hálózat tanulható paraméterszámától függően, a zajmentes adatokhoz képest.

2. Mesterséges neurális hálózatok

A mesterséges neurális hálókat az agyban található neuronok és azok összeköttetése ihlette. Ebben a fejezetben először a mesterséges neuronok és az előre-csatolt neuronhálózatok egy fajtáját, a többrétegű neuronhálózatok általános felépítését fogom bemutatni. Ezek után egy speciális esetüket, a konvolúciós neurális hálózatokat, végezetül pedig a feladatom szempontjából lényeges CNN (Convolutional Neural Network) architektúrákat és adatkészleteket.

2.1. Mesterséges neuronhálózatok – ANN

A mesterséges neuronhálózatok legfőbb elemei a mesterséges neuronok (perceptrons), másnéven a csomópontok (nodes) [1]. Ezek a neuronok az előre-csatolt többrétegű neuronhálózatoknál mind kapcsolódnak egymáshoz, és rétegekbe vannak rendezve. Két neuron közötti kapcsolat egy w súlyértékkel (weight) van ellátva. Ezzel a súlyértékkel van megszorozva az előző rétegből kapcsolt neuron kimenete, így tudja a háló mérlegelni az előjel és amplitúdó függvényében, hogy az a neuron kapcsolat az adott probléma megoldásában mennyire fontos. A csomópontokhoz egy b eltolássúly (bias) is hozzá van rendelve, az (1) -es képletből következően egy küszöbértéket képez a neuronok aktivitásának. Általánosságban rétegenként ugyanazzal az eltolássúllyal számolnak a neuronok. Az összes kapcsolaton keresztül érkező szorzatok összege bemenetét képzik az adott neuronhoz tartozó, úgynevezett ϕ aktivációs függvénynek, ami egy nemlineáris leképezést adja a bemenetnek az aktuális neuron kimenetén, ezzel biztosítva a hálózat nem linearitását. Ilyen függvény például a logisztikai szigmoid (2), vagy a ReLU (3). Egy többrétegű neuronhálózatban egy adott neuron kimeneti értékét az (1) -es képlet adja meg, grafikus szemléltetése a 2. ábrán látható.

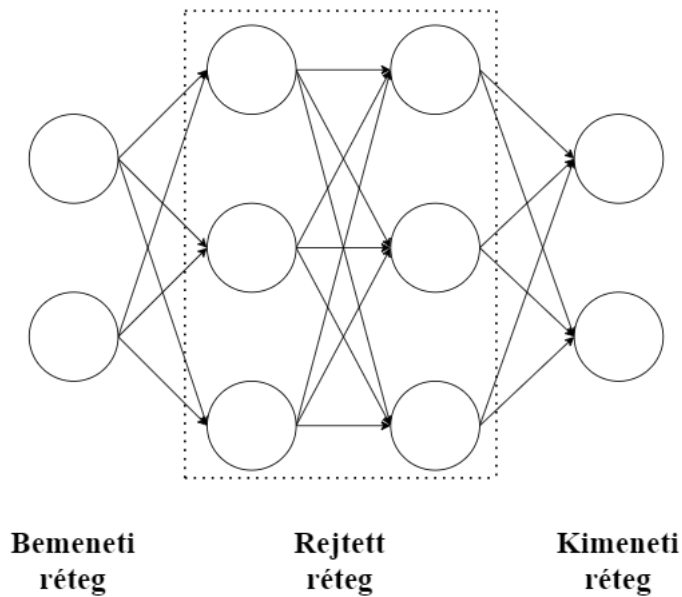
$$a_i^{(L)} = \phi \left(\sum_{i=1}^n w_i^{(L)} \cdot a_i^{(L-1)} + b^{(L)} \right) \quad (1)$$

ahol $a_i^{(L)}$ az adott neuron kimenete.

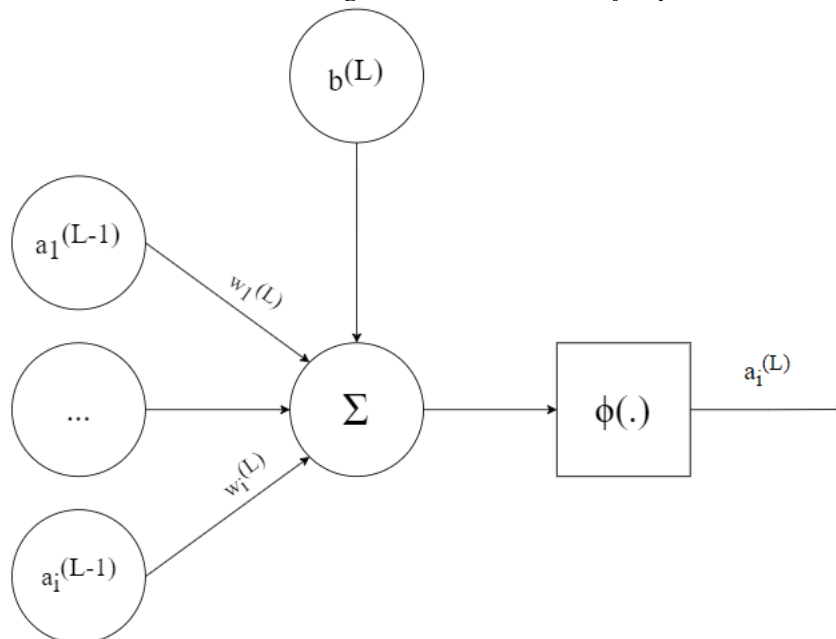
$$\phi(x) = \frac{e^x}{e^x + 1} \quad (2)$$

$$\phi(x) = \max(0, x) \quad (3)$$

A többrétegű neuronhálózatok esetében három réteg különböztethető meg: a bemeneti réteg, a rejtett réteg és a kimeneti réteg, ahogy az az 1. ábrán is látható.



1. ábra : Mesterséges neuronhálózat elvi felépítése



2. ábra : Egy mesterséges neuron művelet gráfrepresentációja

A korábbiakban leírtak alapján elmondható, hogy a rendszer, ami keletkezik, sok nem lineáris függvény szuperpozíciója [2], tehát ezek a hálózatok a bemenet nem lineáris leképezését adják a kimeneten. Ez a tulajdonságuk teszi lehetővé a mesterséges neuronhálók alkalmazását olyan erősen nem lineáris problémák megoldásához, mint egy adott képen való objektumok osztályozása.

2.2. Konvolúciós neurális hálózatok – CNN

A mesterséges neurális hálózatok egy speciális részét képezik a konvolúciós neurális hálózatok, amelyek manapság a gépi látás szerves részét képezik. Felépítésük annyiban különbözik az előbbieken bemutatott mesterséges neuronhálózat modelljétől, hogy nincsen mindegyik neuron összeköttetésben a konvolúciós rétegek között, tehát nem teljesen összekapcsolt hálózatok, és a súlyozások konvolúciós rétegenként megegyeznek. Ebben az alfejezetben fogom bemutatni a felépítésüket rejtett rétegük szerint, amit általánosságban konvolúciós rétegek, teljesen összekapcsolt rétegek és összevonó rétegek alkotnak.

2.2.1. Konvolúciós réteg

A konvolúciós neurális hálózatok legértékesebb rétege a konvolúciós réteg [3], neve a konvolúció műveletből származik. Segítségével a háló a bemeneti képről olyan hasznos tulajdonságokat tud leképezni (egy adott objektum éleitől kezdve teljes alakzatokig) amelyek felhasználásával képes pontos predikciót adni a kimenetén.

Egy ilyen réteg általában jellemezhető egy D csatorna számmal, egy K konvolúciós kernellel, aminek térbeli kiterjedését $n \times m$ alakban szokás megadni, egy s lépésközzel és egy p kitöltési tényezővel. Szerepeikre később külön-külön fogok kitérni. Bemenete általában vagy maga a bemeneti kép, vagy az előző konvolúciós réteg kimenete. Ezek adattípusa egy mátrix $H \times W \times D$ térbeli kiterjedéssel, ahol $H \times W$ a kép pixelszerinti magassága és szélessége, és D a csatornák száma (RGB kép esetén $D = 3$).

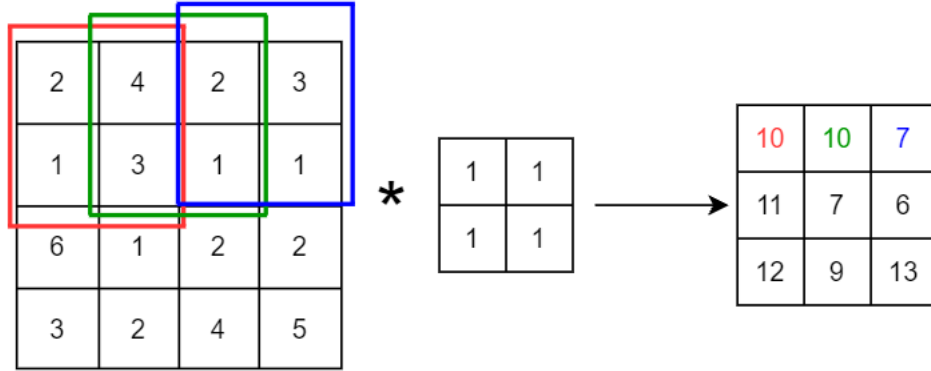
A konvolúció művelet bemutatásához a 3. ábra ad segítséget. Kimenetének a matematikai leírása háromdimenziós bemenet esetén a (4) -es egyenlet, ahol az egyszerűség kedvéért eltolássúllyal nem számolok, kitöltés nincs, $s = 1$.

$$y_{i^{(L+1)},j^{(L+1)},d} = \sum_{i=0}^H \sum_{j=0}^W \sum_{d^{(L)}=0}^{D^{(L)}} k_{i,j,d^{(L)},d} \cdot x_{i^{(L+1)}+i,j^{(L+1)}+j,d^{(L)}} \quad (4)$$

$$\text{ahol } y_{i^{(L+1)},j^{(L+1)},d} = a^{(L+1)}, \quad x_{i^{(L+1)}+i,j^{(L+1)}+j,d^{(L)}} = a^{(L)}, \quad k_{i,j,d^{(L)},d} = K$$

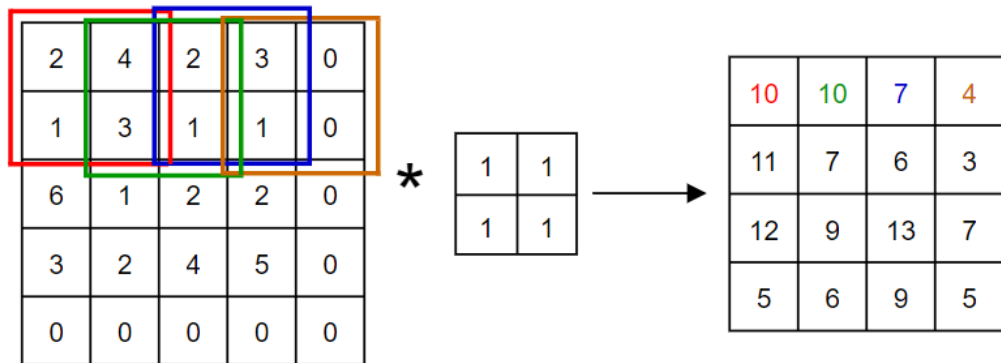
Veszek egy 2×2 -es térbeli kiterjedésű konvolúciós kernelt, aminek mindegyik eleme 1. Veszek egy $4 \times 4 \times 1$ -es bemeneti képet, amelynek elemei egy-egy pixelértéknek feleltethetők meg. Ha a konvolúciós kernelt a bemeneti kép felé helyezem, és minden azonos helyen elhelyezkedő kernel értéket és képmátrix értéket összeszorozok, majd ezeket összegezem, akkor végeredményként egy egész számot kapok. A 3. ábra alapján a kernel első pozíciójában ezt az

alábbi módon kapom: $2 \cdot 1 + 4 \cdot 1 + 1 \cdot 1 + 3 \cdot 1 = 10$. Majd ezt egy pixelnyi lépésközzel ($s = 1$) végrehajtom a vízszintes és függőleges tengelyek mentén, amíg el nem érek a képünk bal felső sarkától a jobb alsó sarkáig. Így kapom meg a művelet végeredményét, ami szintén egy mátrix.



3. ábra : A konvolúció illusztrációja

Észrevehető az ábra alapján, hogy a kimenet mérete a bemenethez képest változott, most már $(\frac{W-n}{s}) + 1$ sorral és $(\frac{H-m}{s}) + 1$ oszloppal rendelkezik, az én esetemben $(\frac{4-2}{1}) + 1$ sorral és $(\frac{4-2}{1}) + 1$ oszloppal, vagyis egy 3×3 -as kimenetet kapok a 4×4 -es bemenetből. Vannak olyan esetek, amikor azonos méretű kimeneti képre van szükség mint a bemeneti kép. Ekkor szokás használni kitöltést (p). A konvolúció előtt $\lfloor \frac{n-1}{2} \rfloor$ sort illesztnek a bemenet felső széléhez, $\lfloor \frac{n}{2} \rfloor$ sort az alsó széléhez, $\lfloor \frac{m-1}{2} \rfloor$ oszlopot a bal oldali széléhez és $\lfloor \frac{m}{2} \rfloor$ oszlopot a jobb oldali széléhez, ahogy az a 4. ábrán is látható. Elemeinek értéke általában 0, de természetesen lehet más értékeket is használni. Ennek hatására a kimeneti dimenzió az alábbi módon változik: $(\frac{H+p-n}{s} + 1) \times (\frac{W+p-m}{s} + 1)$ kiterjedésű lesz, ahol p értékét a fentebb leírtak alapján könnyen meg lehet határozni. Behelyettesítés után megkapom, hogy a kimenetünk a 4. ábrán egy 4×4 mátrix lesz, tehát nem történik méretbeli változás.



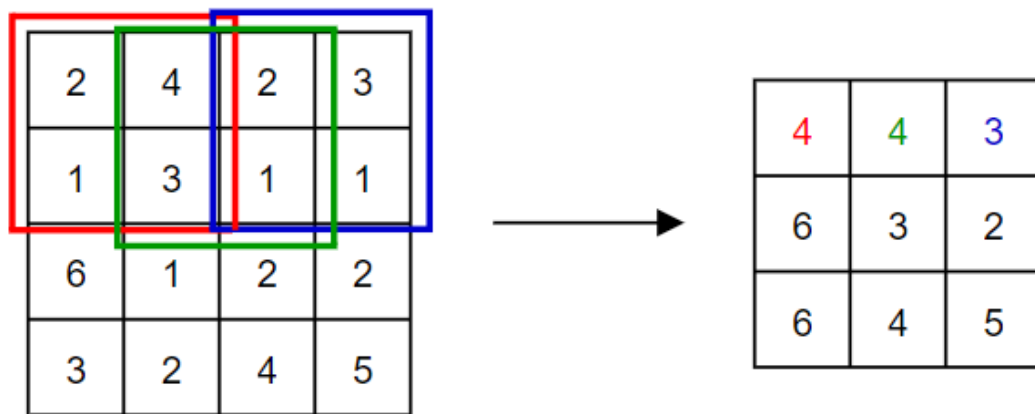
4. ábra : Konvolúció illusztrációja kitöltés alkalmazásával

Másik hasznos tulajdonsága a kitöltésnek, hogy a normál konvolúció során kevésbé érintett pixelek nagyobb szerepet kapnak. Amíg kitöltés nélkül a 3. ábrán a jobb alsó sarokban elhelyezkedő 5-ös értékű pixelt csak egyszer érinti a konvolúciós kernel, addig kitöltéssel már 4-szer. Így a kép szélein elhelyezkedő pixelértékek hasonló szerepet játszanak a tulajdonságtérkép leképezésében, mint a kép közepén elhelyezkedő pixelértékek.

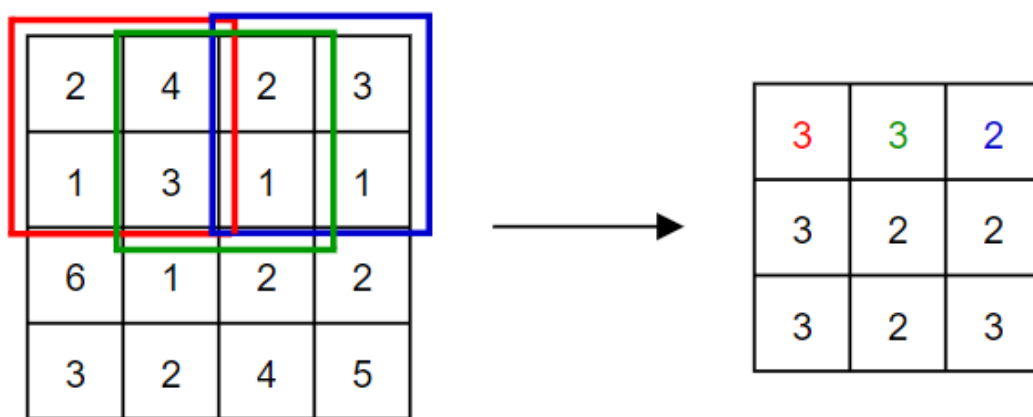
Egy konvolúciós réteg súlyai maguk a konvolúciós kernelek értékei. A rétegeken belül csatornánként azonos konvolúciós kernelt használ a hálózat a műveletekhez. Kimenetének csatornaszámát a rétegen belül felhasznált konvolúciós kernelek száma határozza meg. A rétegek kimeneteit tulajdonságtérképeknek szokás nevezni egy hálózaton belül. Minél több ilyen réteg van felhasználva egy modellben, annál mélyebb információkat lehet szerezni az adott bemeneti képről. A mélyebb rétegek jobban képesek „megtanulni” a tanulási fázis során, hogy csak bizonyos komplex mintákra legyenek aktívak. Egyre mélyebb rétegekben, ezek a minták már szemantikailag értelmes objektumrészek, vagy teljes objektumtípusok. Általánosságban elmondható, hogy minél több konvolúciós réteg van felhasználva egy CNN-ben, annál pontosabb predikciók érhetőek el, mondjuk egy adott objektum osztályozásában, viszont figyelembe kell venni, hogy egyre nagyobb számítási igénye lesz a hálózatnak a sok konvolúciós művelet miatt.

2.2.2. Összevonó réteg

Az összevonó rétegek [3] egy konvolúciós neurális hálózatban általában a konvolúciós rétegek között szoktak elhelyezkedni. Szerepe az, hogy redukálja a tulajdonságtérképek méreteit, ezzel csökken a hálózat paraméter száma, amivel a számítási igények is mérséklődnek. Egy ilyen réteg nulla tanulható paraméterszámmal rendelkezik, tehát nem vesznek részt a tanulási fázisban. Mint a konvolúciós réteg, az összevonó réteg is rendelkezik egy kernellel, amelynek mérete meghatározza, hogy mekkora régióként összegzi a bemenetként kapott tulajdonságtérkép értékeit. Ezen kívül jellemezhető egy lépésközzel, ami meghatározza mekkora a pixel szerinti léptéke a kernelnek, csak úgy, mint a konvolúciós rétegnél. Leggyakrabban maximum összevonó és átlagolt összevonó rétegről lehet beszélni. Az előbbi működése az 5. ábrán, az utóbbi a 6. ábrán látható.



5. ábra : A maximum összevonás illusztrációja



6. ábra : Az átlagolt összevonás illusztrációja

A réteg kimenetének mérete hasonlóképpen számolható ki, mint a konvolúciós réteg kimenete.

2.2.3. Teljesen összekapcsolt réteg

Teljesen összekapcsolt rétegnek [3] hívnak minden olyan réteget egy neurális hálózatban, amiről elmondható, hogy bármely elemének a kiszámításához szükség van az előző réteg összes elemére. Egy konvolúciós neurális hálózatban azért van szükség teljesen összekapcsolt rétegekre (általában a hálózat végén), mert hozzá képest az utolsó réteg a bemeneti kép elosztott reprezentációit tartalmazza, és az összes kinyert tulajdonságot érdemes felhasználni a predikcióhoz, vagyis a kimeneti réteghez.

2.3. Mesterséges neurális hálózatok tanulása

A mesterséges neurális hálózatok tanulása során a cél az, hogy a neuronok kapcsolatainak súlyértékei (és a rétegek súlyeltolásait) olyan módon legyenek módosítva minden végigfuttatott tanító adat/adatköteg után, hogy a hálózat veszteségfüggvényének (C) értéke minimális legyen.

A veszteségfüggvény általában az elvárt kimeneti érték és a kapott kimeneti érték különbségének a leképzése. A súlyozások frissítését az úgynevezett hiba visszaterjesztési algoritmussal szokás elvégezni. Még mielőtt erre részletesebben kitérnék, fontos először magáról a tanító adatkészletről és a tanítás menetéről néhány szót ejteni.

A tanításhoz használt adatkészletet három részre lehet bontani: egy tanító készletre, egy érvényesítő készletre és egy teszt készletre. Ezek felügyelt tanulás esetén mind címkézett adatok, tehát tartalmazzák az elvárt kimeneti értéket, amit később össze lehet hasonlítani a hálózat aktuális kimenetével. A tanítást korszakokra lehet lebontani, egy korszakot akkor tekinthető befejezettnek, ha a rendelkezésre álló összes tanítási minta átment a hálózaton. A korszakok iterációkból állnak, aminek száma mintánként való tanítás esetén megegyezik a tanító készlet adatainak darabszámával. Általában nem mintákkal, hanem kötegekkel szokás tanítani a számítási igények csökkentésének érdekében. Ez azt jelenti, hogy egy minta helyett egyszerre több minta fut keresztül a hálón egy iteráció során, így a tanulható paramétereket (súlyokat) elég kötegenként frissíteni. Az érvényesítő készletet általában minden korszak végén szokás végig futtatni a hálón. Mivel az érvényesítő mintákkal a hálózat az adott korszak során nem találkozott, ezért a tanításból és az érvényesítésből kapott veszteség függvények összehasonlításával meg lehet határozni, hogy a modell mennyire volt képes elsajátítani egy általánosított megoldást az adott problémára. Ha ezek az értékek nagyban eltérnek egymástól (a tanítás során kapott veszteségfüggvény értéke jóval kisebb, mint az érvényesítés során kapotté), az azt jelenti, hogy a modell túlilleszkedik a tanító készletre, és nem volt képes általánosítani a problémát. Ezen a tanításhoz használt hiperparaméterek módosításával (tanulási ráta (η), köteg méret...stb.) lehet változtatni. Egy tanítást addig fut általában, amíg a validációs készletre adott veszteségfüggvény értéke el nem kezd nőni. Fontos megjegyezni, hogy az érvényesítő készlet nem vesz részt a hálózat paramétereinek frissítésében. A teszt készlettel lehet kiértékelni tanítás végén a modell predikciós pontosságát.

2.3.1. Hiba visszaterjesztés

Hiba visszaterjesztésnek [1] hívják azt a módszert, amivel a veszteség függvény értéke rétegenként vissza van terjesztve, és a hálózat tanulható paramétereit (súlyait) olyan irányba és mértékbe frissíti a gradiens ereszkedés módszerével (6), hogy a veszteség függvény értéke minimális legyen. Az alfejezetben ismertetett egyenletek mind többrétegű neuronhálózatok tanítására vonatkoznak.

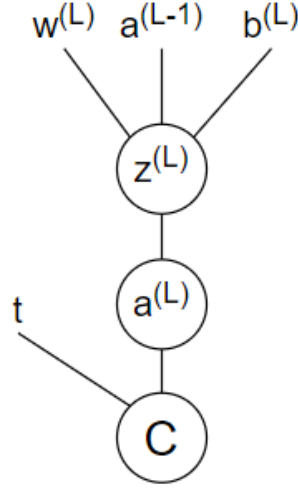
Minden iteráció végén meghatározható egy veszteségfüggvény érték. Ahhoz, hogy ezt minimalizálni lehessen, ahhoz először meg kell határozni $\frac{\partial C}{\partial w_i^{(L)}}$ értékét. Ezt a láncszabály szerint

egyszerűen meg lehet kapni (5).

$$\frac{\partial C}{\partial w_i^{(L)}} = \frac{\partial z_i^{(L)}}{\partial w_i^{(L)}} \cdot \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \cdot \frac{\partial C}{\partial a_i^{(L)}} \quad (5)$$

$$\text{ahol } z_i^{(L)} = w_i^{(L)} \cdot a_i^{(L-1)} + b^{(L)}.$$

Hogy miért alkalmazható a láncszabályt az a 7. ábrán látható.



7. ábra : Láncszabály a parciális deriválthoz

Az (5) -ös egyenlet bal oldala megmutatja, hogy az aktuális súlyérték változtatására mennyire érzékeny a veszteségfüggvény. Ha ez a művelet végre van hajtva az adott réteghez tartozó összes súlyhoz, megkapható a réteghez tartozó ∇C gradiens vektort, aminek (-1) -szeresével frissíteni lehet az adott réteghez tartozó súlyvektort (6) alapján (ha nem a (-1) -szeresével frissítené akkor nem csökkenne, hanem nőne a veszteségfüggvény értéke).

$$w^{(\tau+1)} = w^{(\tau)} - \eta \cdot \nabla C(w^{(\tau)}) \quad (6)$$

ahol $w^{(\tau+1)}$ a frissített súlyvektor az adott (L) rétegben.

Az (6) -ben szereplő η tanulási ráta értéke általában 0,01...0,0001, szerepe, hogy meghatározza a gradiens ereszkedés léptékét.

Az előző rétegre történő hiba visszaterjesztése (7) alapján történik, ahol szintén a 7. ábrán látható láncszabályt lehet alkalmazni.

$$\frac{\partial C}{\partial a_i^{(L-1)}} = \sum_{i=0}^{n^{(L-1)}} \frac{\partial z_i^{(L)}}{\partial a_i^{(L-1)}} \cdot \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \cdot \frac{\partial C}{\partial a_i^{(L)}} \quad (7)$$

A (7) -es egyenlet alapján elmondható, hogy ha ismerem, hogy az utolsó előtti réteg aktivitására mennyire érzékeny a veszteségfüggvény, akkor ezt a lépést meg lehet ismételni az összes korábbi rétegre, és (6) alapján frissíteni lehet a súlyértékeket.

A fejezet zárásául fontosnak tartom megjegyezni, hogy mivel általában a tanítás elején

véletlenszerűen kerülnek felvételre a súlyok kezdeti értékei (ezzel a veszteségfüggvény síkján is a kezdeti pontot), és a veszteségfüggvény sok változónak a függvénye, ezért valószínű, hogy a függvénynek nem a globális minimuma lesz megtalálva tanítás végére, hanem a kezdeti ponthoz legközelebb eső lokális minimum. Más módszerek is léteznek a gradiens ereszkedésen kívül a hálózat paramétereinek frissítésére, ezeket dolgozatomban külön nem fogom ismertetni, mivel egy általános ismertetés a célom.

2.4. Felhasznált adatkészlet

A modellek kiértékeléshez a Microsoft COCO: Common Objects in Context [5] adatkészletet használtam fel. Választásom egyik oka, hogy a YOLOv5 modell ezen az adatkészleten lett tanítva, másik pedig, hogy a kiértékelés szempontjából is a legrelevánsabb eredményt adja összetettsége miatt.

Az adatkészlet 2014-es megjelenése óta mára már 327 000 képet foglal magába és összesen 2,5 millió címkézett példát tartalmaz. Az adatkészlet változását az évek során az 1. táblázat mutatja be.

1. táblázat : COCO adatkészlet felosztása

Módszerek	Év		
	2014	2015	2017
Tanítás	83 000 db	83 000 db	118 000 db
Validáció	41 000 db	41 000 db	5 000 db
Teszt	41 000 db	81 000 db	81 000 db
Címkézetlen	-	-	123 000 db
Összesen	165 000 db	205 000 db	327 000 db

91 osztály van megcímkézve, abból 82 osztályra több, mint 5000 címke jut. Képenként átlagosan 7,7 objektum példány van megjelölve, ami a korábbi adatkészletekéhez képest egy egészen magas szám (átlagosan 3 volt korábban). Ezzel akarták a készítői elősegíteni a tanuló neurális hálózatok tartalmi információszerzését az adott objektumokról, hiszen minél több példát kap egy adott osztályból a hálózat, annál részletesebb tulajdonság térképet tud készíteni róla.

Az osztályok meghatározásánál fontos szempont volt a készítőknél, hogy releváns gyakorlati szituációkban előforduló példákat tartalmazzanak, valamint gyakran előfordulóak legyenek, mivel így könnyebb róluk nagy adatmennyiséget gyűjteni. Ez a 91 osztály csak térben körbe határolható objektumokat tartalmaz, azon belül is az egyszerűbben megnevezhetőket.

A képek többsége nem ikonikus kép, ez annyit jelent, hogy az adott objektumok nem feltétlen a kép közepén szerepelnek, valamint nem olyan háttérrel és módon, ahogyan az

sokszor látható, részletesebben a 8. ábra ad szemléltetést.



8. ábra : Példa ikonikus objektumokra (bal), ikonikus helyekre (közép), nem ikonikus képekre (jobb)
[5, Fig. 2]

Mivel az adatkészlet képei az objektumok hétköznapiabb megjelenéseit reprezentálják, ezért a rajta tanított hálózat is egy jobb általánosítást tud készíteni róluk, ami tanítás során az elsődleges cél.

Az adatkészlet univerzálisnak mondható, mivel több feladat szerint vannak megcímkézve a képei, mint a szegmentálás, objektum detektálás, kulcspont detektálás, „stuff” detektálás (térben nem körbe határolható dolgok, mint az utca), ezért (is) széleskörben alkalmazzák a fejlesztők tanításra és kiértékelésre. Manapság a legkorszerűbb konvolúciós neurális hálózatokra épülő modellek is ezen az adatkészleten vannak tanítva a korábban felsorolt tulajdonságai miatt.

2.5. Felhasznált architektúra

A kiértékeléshez a YOLO modell család ötödik verzióját fogom felhasználni, ezért ebben az alfejezetben röviden befogom mutatni a működését, felépítését, kiértékelésének módját.

A YOLO az egyszakaszú objektum detektáló hálózatok közé tartozik, tehát nincs szüksége egy külön modulra (ez lehet egy egyszerűbb algoritmustól egy külön hálózat), ami javaslatokat tesz, hogy mely területeken lehetnek a képen objektumok, hanem a hálózat önmagában képes meghatározni azokat. Működési elve dióhéjban a következő. Minden bemeneti képet feloszt először $S \times S$ méretű cellákra. Minden cellához meghatározza, hogy mekkora valószínűséggel szerepelhet abban egy meghatározott objektum, és minden lehetséges c osztályú objektumhoz hozzárendel egy t_x, t_y, t_w, t_h értékekkel reprezentált befoglaló téglalapot. A nem-maximum kiejtés segítségével megtartja a legerősebbnek ítélt predikciókat a kimeneten. Nem-maximum kiejtés az a művelet, amikor egy meghatározott konfidencia és IoU (Intersection over Union) küszöbérték mellett megvizsgálja a hálózat a legerősebb predikcióit és a küszöbértékek alapján kiejti a redundánsnak ítélteteket.

2.5.1. Modell pontosságának kiértékelése

Objektum detektáló hálózatoknál a modell kiértékelése többféleképpen történhet, mindegyik módszer a hiba mátrixból van származtatva, amit a 9. ábra szemléltet.

		Prediktált	
		Negatív	Pozitív
Aktuális	Negatív	Igaz Negatív	Hamis Pozitív
	Pozitív	Hamis Negatív	Igaz Pozitív

9. ábra : A hiba mátrix

Először fontos definiálni a táblázatban látható fogalmakat. Igaz negatív az az eset, amikor a modell nem prediktálja az adott osztályt és nem is volt ott a bemeneten. Hamis negatív, amikor a modell nem prediktálja az adott osztályt, pedig rajta volt a bemeneten. Hamis pozitív, amikor a modell prediktálja az adott osztályt, pedig nem volt rajta a képen. Igaz pozitív pedig az az eset, amikor a hálózat prediktálja az adott osztályt és a képen is szerepelt.

Két fontosabb érték van, amivel jellemezni lehet egy objektum detektáló hálózat jóságát, a precizitás (precision) (8) és a visszahívás (recall) (9).

$$\text{Precizitás} = \frac{\text{Igaz Pozitív}}{\text{Igaz Pozitív} + \text{Hamis Pozitív}} \quad (8)$$

$$\text{Visszahívás} = \frac{\text{Igaz Pozitív}}{\text{Igaz Pozitív} + \text{Hamis Negatív}} \quad (9)$$

Precizitással azt mérjük, hogy a modellünk, az összes prediktált osztály közül mennyit talált el helyesen, visszahívással pedig azt, hogy mennyi osztályt talált meg az aktuális osztályok közül.

Minden objektum detektáló hálózat jellemezhető továbbá egy átlag pontossággal (AP) és egy F1 pontszámmal (10).

$$F1 = 2 \cdot \frac{\text{Precizitás} \cdot \text{Visszahívás}}{\text{Precizitás} + \text{Visszahívás}} \quad (10)$$

AP értéknek a precizitás és visszahívás görbe (PR görbe) alatti területet szoktuk nevezni. Manapság gyakran feltüntetik egy hálózatnak az mAP értékét, például a COCO adatkészleten, ami az átlaga az osztályonkénti AP értékeknek. Gyakorlatban az AP-t és mAP-t nem szokták megkülönböztetni.

Az F1 pontszám néha egy jobb mérési érték a modell jóságára, főleg, ha egyensúlyt akarunk keresni a precizitás és visszahívás között, mivel ezeknek az értékeknek a súlyozott átlagát

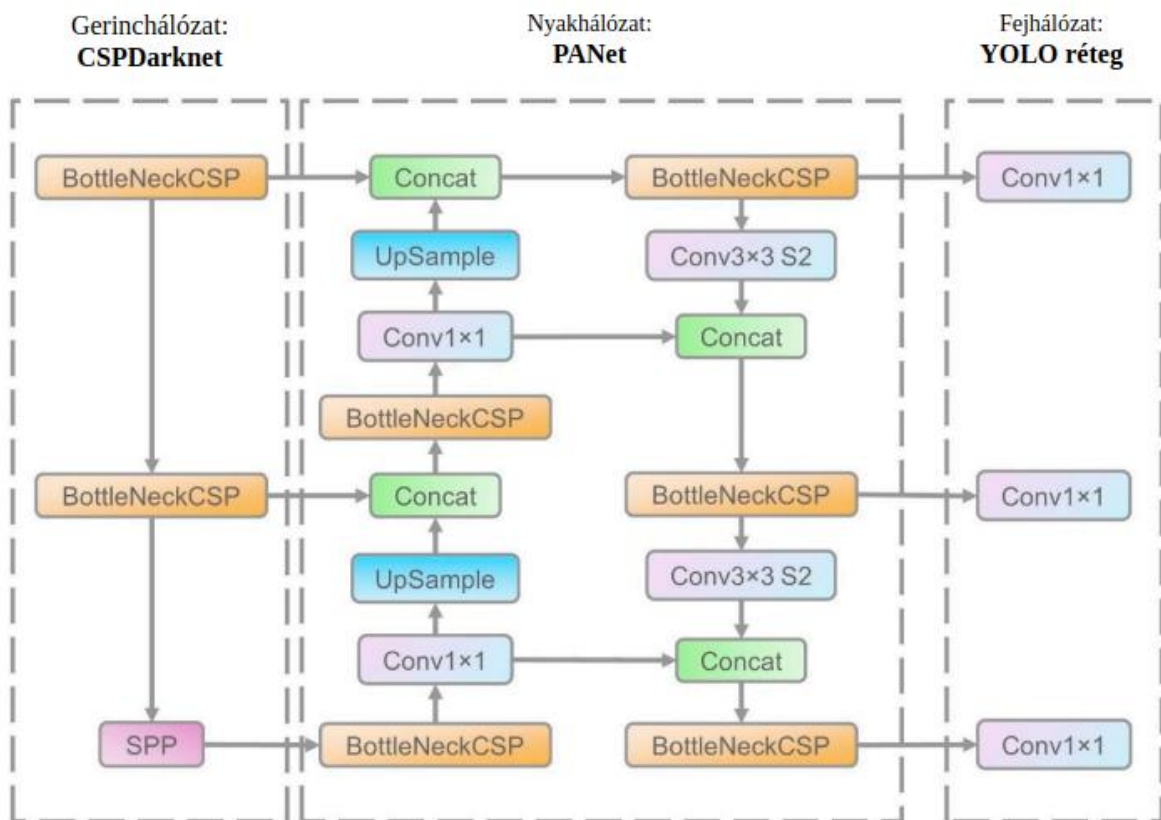
képviseli. Továbbá akkor is jobban tükrözi a valóságot, ha a kiértékelő adatkészleten belül nem egyenletes az osztályok eloszlása. A kiértékelés folyamata során függvényként ábrázolható a hálózat konfidencia értékéhez képest. Segítségével könnyen meghatározható az a konfidencia küszöbérték, amivel később a hálózatot a gyakorlati alkalmazása során futtatni érdemes.

2.5.2. Modell felépítése

A YOLOv5 [4][12] egyike a mai legkorszerűbb objektum detektáló hálózatoknak. A YOLO korábbi verzióira épül (v1-v4), felépítése a 10. ábrán látható.

Gerinchálózata a CSPDarknet, ami annyiban különbözik a korábbi verzióknál használt Darknet-től, hogy bele van építve a CSPNet [7]. Így a gradiens változások integrálva vannak a tulajdonság térképekbe, ezzel csökkentve a modell paraméter számát és a másodpercenként elvégzendő lebegőpontos számításokat (FLOPS). Segítségével csökken a hálózat mérete és nő a gyorsasága, pontossága. A nyakhálónak használt PANet [8] segít az objektumok pontosabb lokalizációjában és alacsonyabb szintű tulajdonságok továbbításában a fokozott alulról-felfele történő információ áramlásának köszönhetően. A fejhálónak használt YOLO réteg három különböző nagyságú tulajdonság térképet szolgáltat a kimenetre (18×18 , 36×36 , 72×72), hogy a különböző nagyságú (kicsi, közepes, nagy) objektumok predikcióját pontosítsa.

A dolgozatomban felhasznált YOLOv5s verzió a COCO adatkészleten 37,2 mAP (0.5:0.95 IoU) értékkel rendelkezik, valamint 7,2 millió tanulható paraméterszámmal. A YOLOv5l verzió a COCO adatkészleten 48,8 mAP (0.5:0.95 IoU) értéket ér el, 46,5 millió tanulható paraméterszámmal.



10. ábra : YOLOv5 architektúra felépítése [5, Fig. 4]

3. Felhasznált keretrendszer

3.1. PyTorch

A PyTorch [9] mély tanulási keretrendszer létrehozásakor a fejlesztők célja az volt, hogy egyszerre legyen a felhasználó számára könnyen értelmezhető és használható, teljesítményében pedig gyors. Korábbi mély tanulási keretrendszerekről, vagy csak az előbbi, vagy csak az utóbbi volt elmondható. Például a Caffe-nál, vagy a Tensorflow-nál statikus adatfolyam gráf reprezentálja a számítást, amely idő előtti belátást biztosít a teljes számítási igényre, amely felhasználható a méretezhetőséget és a teljesítmény javítására, cserébe a könnyű érthetőséget és rugalmasságot feláldozva. Másik példa a Chainer vagy a DyNet, amiknek pozitívuma a dinamikus végrehajtás, és a definie-by-run megközelítés, viszont ez mind a teljesítmény rovására megy, amit például a DyNet kevésbé kifejezőbb, gyorsabb nyelv (C++) használatával próbál kiküszöbölni.

A PyTorch dinamikus tenzor számításokkal dolgozik a gyorsaság megtartása mellett, ezek mellett egy kifejező programnyelvet (Python) felhasználva. Követve a szabad szoftver mozgalmat, a PyTorch is egy nyílt forráskódú keretrendszer, mint más Python könyvtárak (például NumPy vagy pandas), elvetve a zárt, szabadalmaztatott szoftverek elvét (például Matlab). Egy tömb-alapú programozási modell, amely GPU-kal (Graphics Processing Unit) felgyorsítható, köszönhetően korábban más speciális könyvtárak által (mint a cuDNN) kialakított mély tanulási kerneleknek. Emellett automatizált deriválásra is képes a Python ökoszisztémába integrált automatikus differenciálásnak köszönhetően. Ez elengedhetetlen a gépi tanulás világában, ahol sok optimalizáló a gradiens ereszkedés módszerét felhasználja.

A fejlesztők több elvre építették a tervezését. Egyik ilyen volt, hogy maradjon „Pythonic” a rendszer, ami annyit jelent, hogy a lehetséges megoldások között mindig legyen egy olyan megoldás, amely a Python programozási nyelv filozófiáját követi (ilyen például egy összetettebb művelet megoldása egy sorban). Másik, hogy a gépi tanulásban rejlő összetettség belülről legyen megoldva, ezzel biztosítva, hogy a különböző gépi tanulási modellek, hozzájuk tartozó adatbetöltők, optimalizálók és egyéb funkciók egyszerűen lehessenek implementálva, egy program kódként lehessen rátekinteni az egész modellre. Példa erre, hogy a különböző összetettebb rétegek, vagy modellek létrehozása egy programozási osztályként történik. Harmadik, hogy inkább áldozzák be a futási sebesség egy kis részét a jelentősen egyszerűbb használatért cserébe. Ha vannak a felhasználónak eszközei arra, hogy manuálisan befolyásolni tudja a kód lefutását, könnyebben talál megoldást a teljesítmény növelésére a könyvtártól

függetlenül, sokszor többszörösen megtérítve az erre beáldozott gyorsaságot. Végül pedig, a belső funkciók felépítése legyen egyszerűbb, ne komplex függvények legyenek, így új módszerek implementálása nem okoz nagy kihívást, ezzel is növelve a keretrendszer rugalmasságát, amire a mesterséges intelligencia gyorsan fejlődő területén nagy szükség van.

A PyTorch átjárhatóságot biztosít más nyílt forráskódú könyvtárakkal, erre példa a NumPy tömbök és a Torch tenzorok könnyű átváltására alkalmas, keretrendszeren belül beépített függvények (mint a `.numpy()`, vagy a `torch.from_numpy()`).

A vezérlési folyamat (például a ciklusok) és az adatfolyam (tenzorok és a rajtuk végzett műveletek) külön vannak választva. Az előbbi Python és optimalizált C++ kódként van kezelve a CPU-n. Az utóbbit a felhasználó dönti el, hogy CPU-n vagy GPU-n akarja futtatni, és a műveletek végrehajtása asszinkron módon történik. Ezért a Python kód végrehajtása CPU-n történik, míg a műveletek akár GPU-n.

A többszörös feldolgozást a multiprocessing nevezetű Python könyvtár kiegészítésével, `torch.multiprocessing` néven beépítették, amely segítségével automatikusan a megosztott memóriára helyezi a más folyamatokhoz küldött tenzor adatokat, ahelyett, hogy a kommunikációs csatornára kerülnének, ezzel is növelve a rendszer teljesítményét.

Rendelkezik egy beépített referencia számlálóval, ami leköveti a tenzorok használatát. Amint egy tenzorra már nincsen szükség, egyből felszabadítja a számára lefoglalt memória területet. A jó teljesítményhez, például egy konvolúciós neurális hálózat tanításánál ez elengedhetetlen, ahol teljes memóriakihasználtsággal számol a felhasználó.

A 2. táblázat jól mutatja, hogy a PyTorch a többi mély tanulási rendszerhez képest milyen teljesítménnyel rendelkezik.

2. táblázat : 6 modell tanítási sebessége különböző keretrendszereken. Az átviteli teljesítmény képek per másodpercben értelmezendő. [9, Table 1]

Keretrendszer	Átviteli teljesítmény (a nagyobb jobb)					
	AlexNet	VGG-19	ResNet-50	MobileNet	GNMTv2	NCF
Chainer	778 ± 15	N/A	219 ± 1	N/A	N/A	N/A
CNTK	845 ± 8	84 ± 3	210 ± 1	N/A	N/A	N/A
MXNet	1554 ± 22	113 ± 1	218 ± 2	444 ± 2	N/A	N/A
PaddlePaddle	933 ± 123	112 ± 2	192 ± 4	557 ± 24	N/A	N/A
Tensorflow	1422 ± 27	66 ± 2	200 ± 1	216 ± 15	9631	4,8e6
PyTorch	1547 ± 316	119 ± 1	212 ± 2	463 ± 17	15512	5,4e6

A táblázatban jól látható, hogy még ha nem is mindegyik hálózat tanításánál érte el a

legnagyobb gyorsaságot a keretrendszer, de mindegyiknél megközelítette a legmagasabb értéket. Összességében így mind felhasználhatóság és gyorsaság szempontjából a legkorszerűbb keretrendszernek mondható a PyTorch.

4. A zajterhelések

4.1. FGSM (Fast Gradient Sign Method)

Az FGSM [6] zajterhelés a lineáris zajterhelések közé sorolható, mivel az x bemeneti képek pixelértékei egy η nagyságú és $\text{sign}(\eta)$ irányú eltolással változnak meg (11).

$$x_p = x + \eta \quad (11)$$

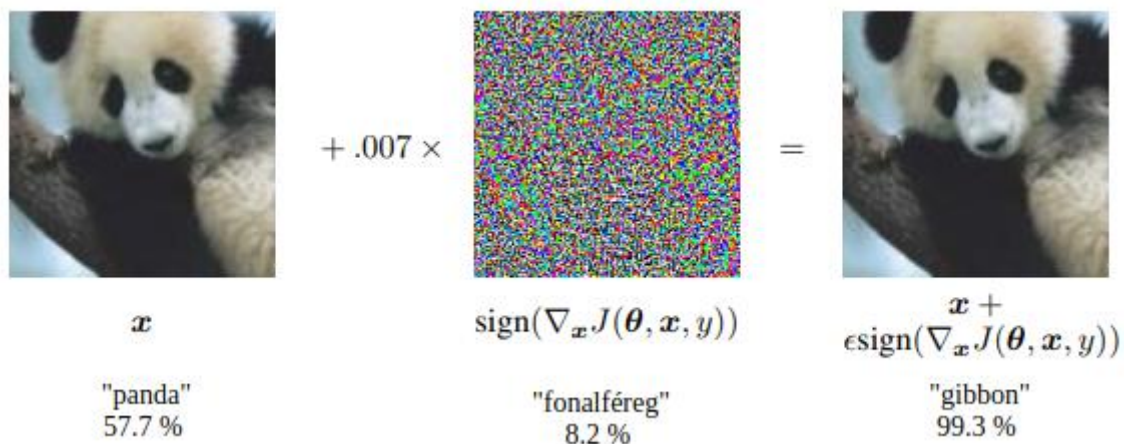
ahol x_p a kapott módosított kép.

Korábban már bebizonyították, hogy ellenséges mintákkal, amelyek csak picivel különböznek az eredeti képtől, könnyen támadhatóak a konvolúciós neurális hálózatok. Többen úgy feltételezték, hogy ez az extrém nem linearitásnak köszönhető, amivel a mély neurális hálózatok rendelkeznek. FGSM és más hasonló módszerek létrehozásával akarták bebizonyítani, hogy nagy dimenziójú terekben elegendő lineárisan változtatni az adatokon, hogy ellenséges mintákat lehessen generálni.

Készítése, (12) alapján, úgy történik, hogy egy bemeneti képet át kell küldeni a hálózaton. Ezután ki kell számítani a rá kapott veszteség függvénynek a gradiensét a bemenethez képest, amit a 2. fejezetben bemutatott hiba visszaterjesztéssel egyszerűen meg lehet határozni. Végül egy ϵ kis számot (0,001 ... 0,5) kell megszorozni a gradiens vektor elemeinek előjelével. Ezt követően (12) -öt behelyettesítve (11) -be megkapjuk az FGSM által zajterhelt képet. Ennek illusztrációja a 10. ábrán látható.

$$\eta = \epsilon \cdot \text{sign}(\nabla_x C(w, x, t)) \quad (12)$$

Általában iterációval szokás változtatni (11) -el az eredeti kép értékét, addig, amíg a CNN nem téveszti el az adott célosztályt az eredeti képre adott predikcióhoz képest.



11. ábra : GoogLeNet-en alkalmazott FGSM példa generálása az ImageNet adatkészlet egyik képéről [6, Fig. 1]

4.2. Eső szimulálása

Eső szimulálásához [11] először meg kell határozni azokat a paramétereket, amik leginkább jellemeznék egy esős képet. Ilyen paraméterek, az esőcsepp térbeli kiterjedése (szélesség, hosszúság), esőcsepp színe, esőcseppek darabszáma és eloszlása, a kép elmosottsága, fényessége.

Az általam felhasznált eső generáló módszerhez először meg kell határozni a kép területét, ezt egy együtthatóval elosztva meghatározható a képre kerülő esőcseppek száma. Minél kisebb ez az együttható, annál erősebb esőzést lehet készíteni a képre. Az esőcseppek egyenként 1 pixel szélesek és 20 pixel magasak, valamint (200, 200, 200) RGB kóddal rendelkeznek, ami a szürke színnek feleltethető meg. Az esőcseppek eloszlása a diszkrét egyenletes eloszlást követik. Miután az esőcseppek rákerülnek az eredeti képre, a kép egy 5×5 -ös normalizált konvolúciós kernellel

$$(K = \frac{1}{25} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}) \text{ van elmosva, mivel esőzésnél a való életben se tiszta a látkép.}$$

Végül a képet RGB-ről HLS-re kell átváltani, és az L komponenst egy 0,7-es értékű együtthatóval megszorozni, hogy sötétebb legyen. Ezáltal a kapott kép jobban közelíti a valóságot, példa rá a 12. ábrán látható.

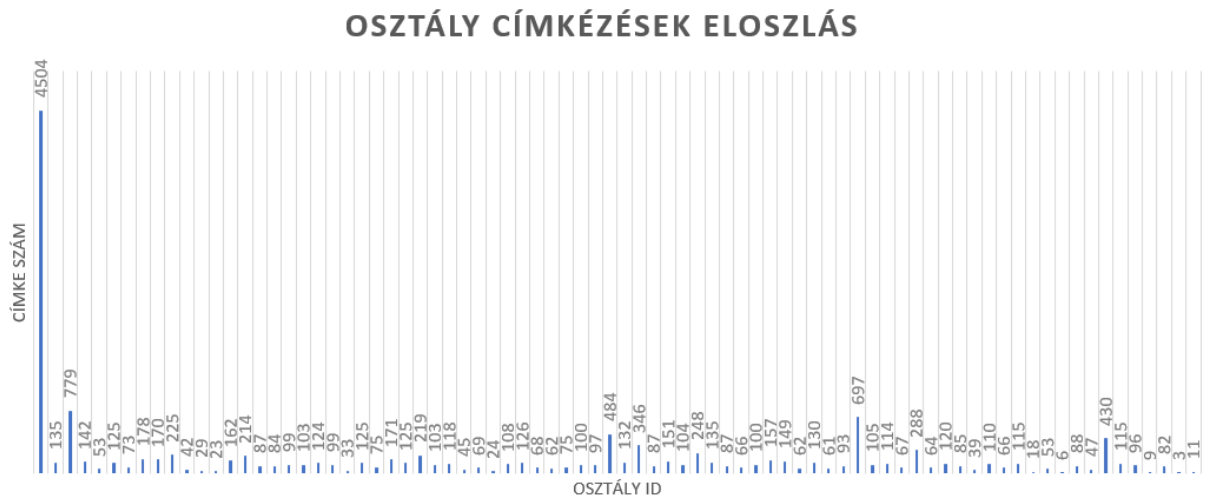


12. ábra : Esős minta generálásához illusztráció

5. A kiértékelő adatkészletek létrehozása

5.1. A nyers adatkészlet

A kezdő kiértékelő adatkészlethez a COCO adatkészlet képeit használtam fel, azon belül is a 2017-es érvényesítő adatkészletnek egy részét. Ez 5000 képet foglal magába, amiből az első 2000 képet használtam fel. A keletkezett adatkészletem összesen 14613 db címkézést foglal magába, osztályok szerinti eloszlása a 12. ábrán látható.



13. ábra : A címkézések eloszlása osztályok szerint az adatkészletben

Hogy a YOLOv5-höz felhasználható legyen ez az adatkészlet, át kellett konvertálnom a címkézések formátumát a COCO-ról YOLO formátumba. A 13., 14. ábrákon látható, hogy az átváltáshoz használt bemenet miként változott meg.

```

annotation{
  "id": int,
  "image_id": int,
  "category_id": int,
  "segmentation": RLE or [polygon],
  "area": float, "bbox": [x,y,width,height],
  "iscrowd": 0 or 1,
}

categories[{
  "id": int,
  "name": str,
  "supercategory": str,
}]

```

↓

```

"category_id": int      "bbox": [x,y,width,height]

```

14. ábra : COCO formátumról YOLO formátumra történő átváltás elve



21 0.50122 0.548094 0.99756 0.881156

15. ábra : Példa egy képre a saját adatkészletből a hozzátartozó címkével

A változtatások után ez az adatkészlet már készen állt a kiértékelésre.

5.2. Zajterhelt adatkészletek

A zajterhelt adatkészleteket a korábban elkészített kiértékelő adatkészletem képein történő módosítással hoztam létre.

Az eső szimulációval módosított adatkészletet a korábbiakban 3.3-as alfejezetben részletezett módszer alapján készítettem el, 15. ábrán látható néhány minta belőle.



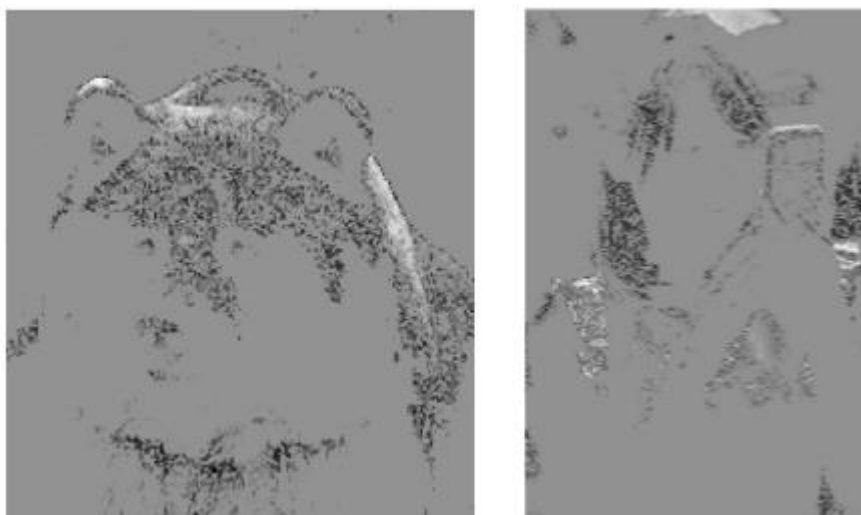
16. ábra : Minták az „esős” adatkészletből

Az FGSM zajterheléssel készített adatkészlethez először egy FGSM támadást kellett csinálnom. A támadást a ResNet101 [10] konvolúciós neurális hálózaton hajtottam végre, veszteség függvénynek a negatív logaritmikus eloszlás függvényt (13) használtam fel, epsilon értékét pedig 0,25-re választottam meg ($\epsilon = 0,25$).

$$C(x) = -\log(x) \quad (13)$$

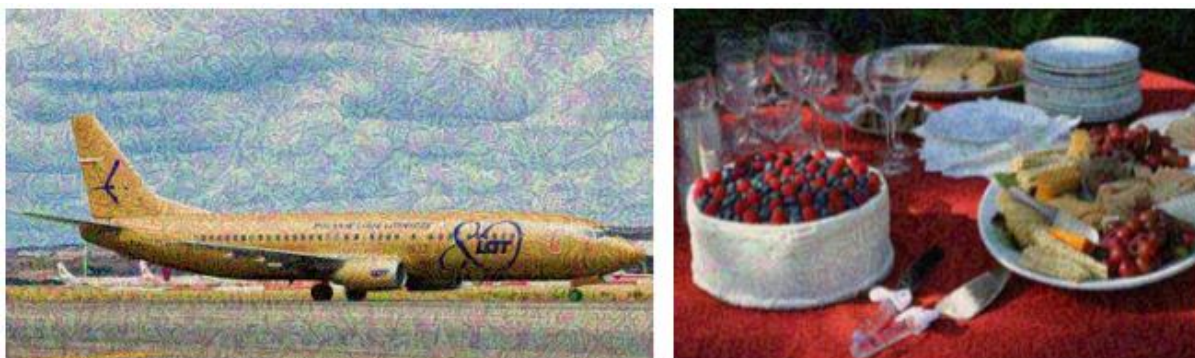
Azért választottam a negatív logaritmikus eloszlást, mivel ez egy gyakran használt veszteség függvény egy n osztályú klasszifikálási problémának a tanításához. Az epsilon értékét próbálgatással vettem fel. A végső választott érték elég magasnak számít, mivel a kimenet és a bemenet közötti különbsége szabad szemmel is látható, tehát a terhelés mértéke erős. Egy mai korszerű objektum detektáló hálózat zajterheléses teszteléséhez erre szükség van.

Minden egyes képet maximum 10 iteráción keresztül vizsgáltam, hogy eltért-e a hálózat predikciója az adott iterációban az elsőétől, és ha igen, akkor az ott megkapott gradiens vektor előjeleit vettem az epsilon-nal történő eltoláshoz. Az eltolások mértékére a 16. ábrán láthatók példák.



17. ábra : A bemeneti és az FGSM-el készített kép pixeleinek abszolút eltérése

Ezekkel belátást is nyerhettem abba, hogy a képeken mely régiókra fókuszál a hálózat egy adott osztály meghatározásánál. Végül a bemeneti képek pixeleinek értékét eltolva FGSM alapján megkaptam a másik zajterhelt adatkészletet, példák belőle a 17. ábrán láthatók.



18. ábra : Minták az FGSM támadásból készített adatkészletből

Ezek után már minden eszköz rendelkezésemre állt, hogy elkezdhessem a modell kiértékelését.

6. YOLOv5 kiértékelése

A modellek kiértékeléséhez az adatkészletekre kapott hiba mátrixot, precízió-visszahívás görbét, és az F1 pontszám-konfidencia görbét vettem alapul. A kiértékeléshez fontosabb felhasznált hardverek: Intel® i9 – 11900 CPU, RTX 3070 8 GB GPU, 32 GB RAM. Ebben a fejezetben fogom bemutatni milyen eredményeket kaptam. A precízebb összehasonlításhoz 2 táblázatot is készítettem mindegyik alfejezethez. Az elsőben látható az összes osztályra kapott átlagos pontossági érték 0.5 IoU értékkel és 0.5:0.95:0.05 IoU értékkel, ahol 0.05 a lépésközt jelenti. A másodikhoz kiválasztottam az 5 legnagyobb átlag pontossággal prediktált osztályt azok közül, amik a 2000 képből legalább 50 címkézéssel rendelkeztek, ezáltal is jobb belátást nyerve a pontosság változásába.

6.1. A modellek teljesítménye a nyers adatkészleten

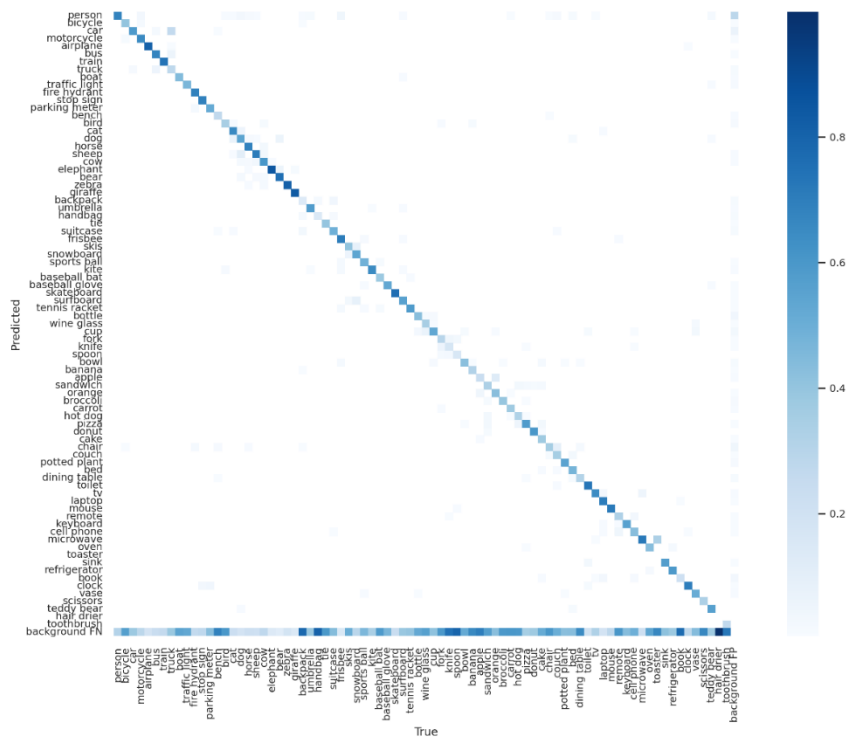
3. táblázat : Modellek sebessége és átlagos pontossága a nyers adatkészleten

	YOLOv5 mérete	
	S	L
Predikció Sebessége	5,8 ms	14,1 ms
mAP @.5	0,609	0,693
mAP @.5:.95	0,443	0,536

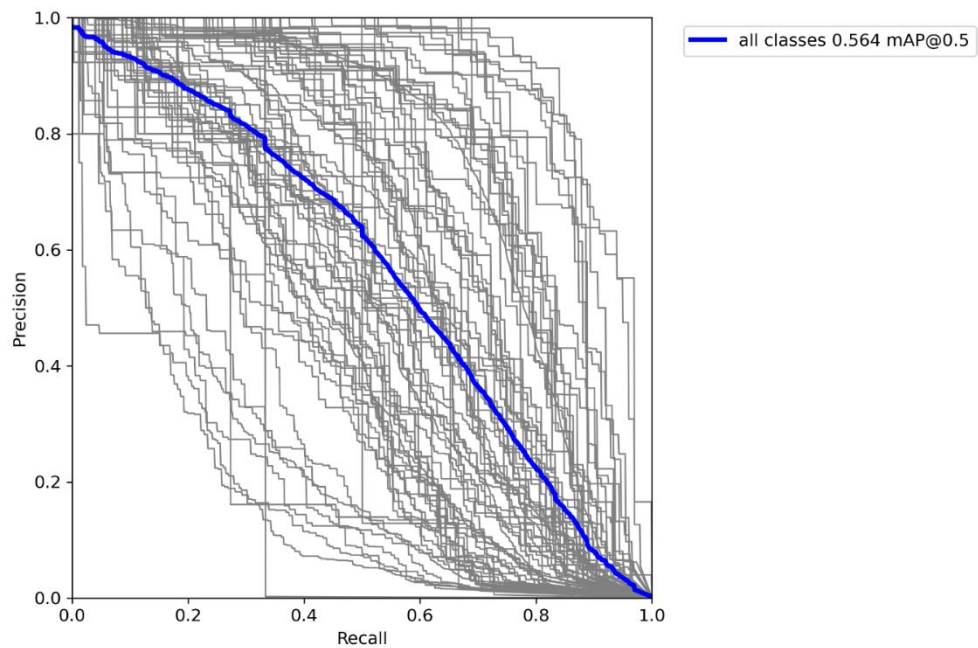
4. táblázat : Modellek átlagos pontossága az 5 legpontosabban prediktált osztályokra a nyers adatkészleten

Top 5 osztály mAP @.5:.95	YOLOv5 mérete	
	S	L
Repülő	0,67	0,81
Elefánt	0,63	0,72
Zsiráf	0,68	0,78
Toalett	0,65	0,7
Laptop	0,62	0,74

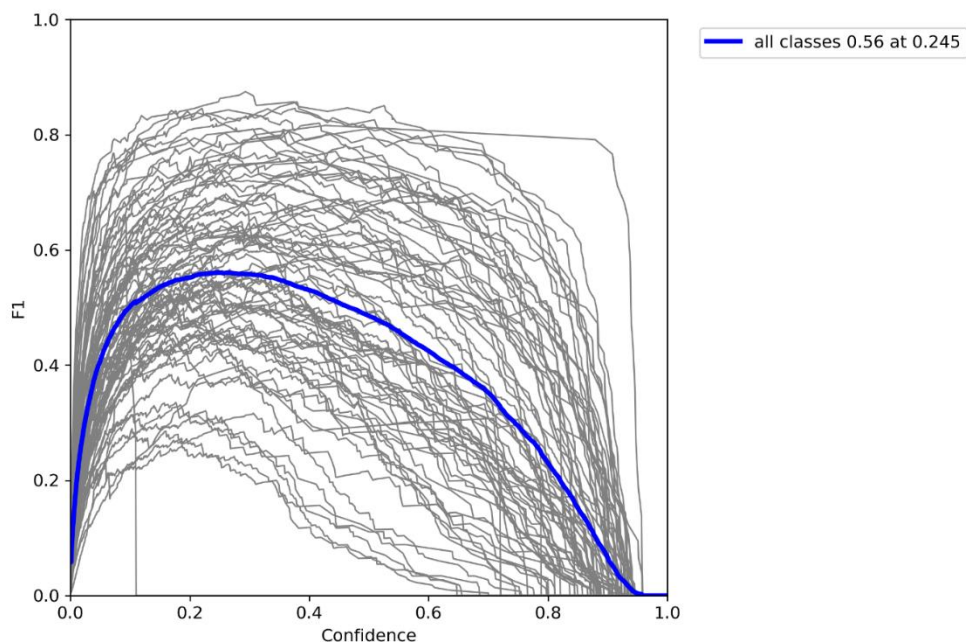
A 3. és 4. táblázatok alapján elmondható, hogy a nagyobb tanulható paraméterszámú verzió átlagos pontossága az összes osztályon körülbelül 17% -kal pontosabb, míg a legpontosabban prediktálható öt osztályra adott átlag pontossága 13% -kal jobb.



19. ábra : YOLOv5s hiba mátrixa a nyers adatkészleten

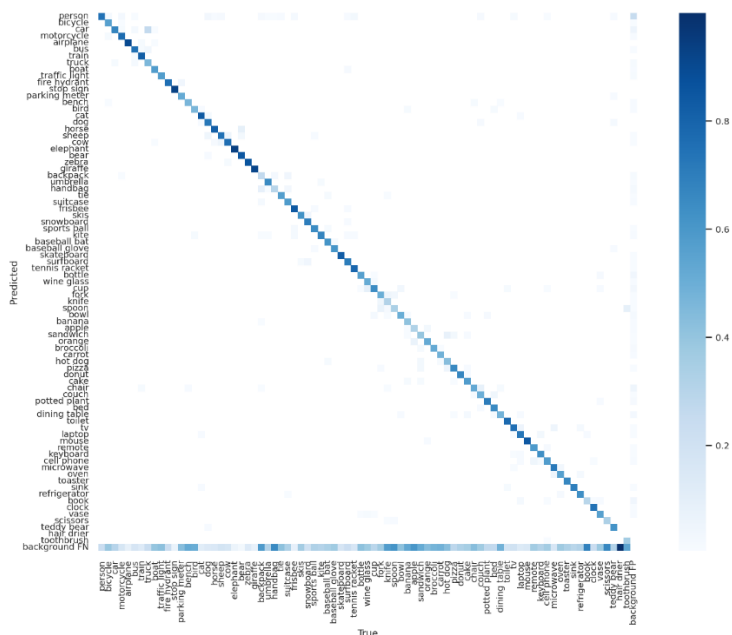


20. ábra : YOLOv5s PR-görbéje a nyers adatkészleten

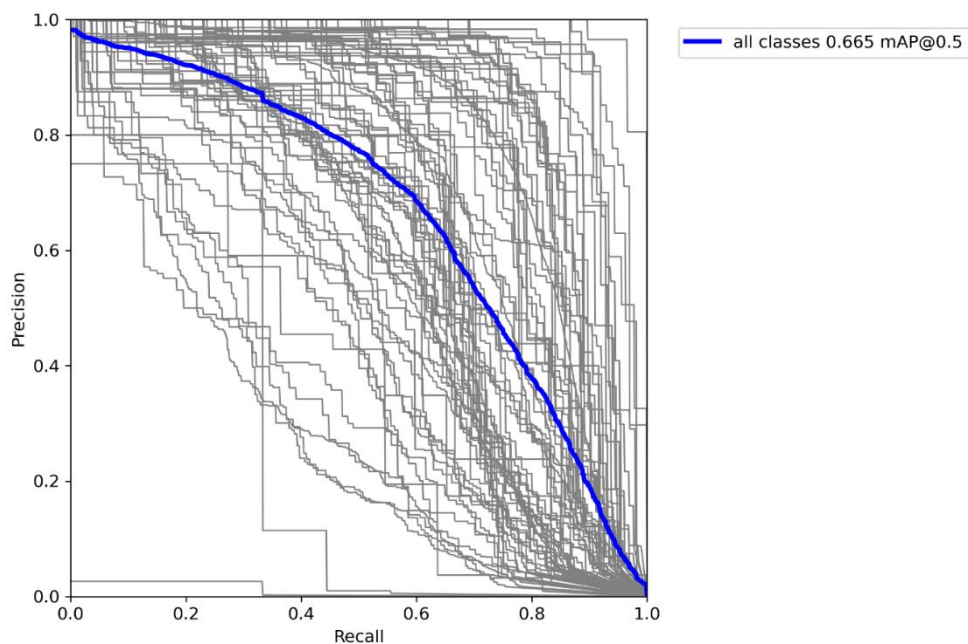


21. ábra : YOLOv5s F1 görbéje a nyers adatkészleten

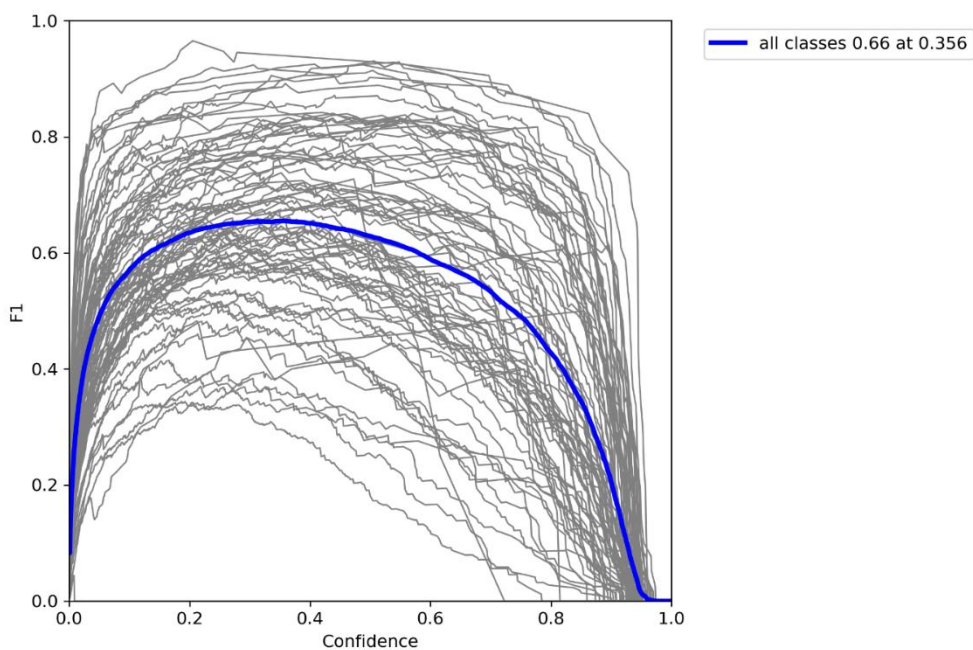
A 18., 19., 20. ábrákra hivatkozva a következők mondhatóak el a YOLOv5s teljesítményéről: A hiba mátrix főátlóján látszik, hogy nagy konfidencia értékek mellett ítélte meg helyesen a hálózat az osztályokat. A PR görbén egyenletesen oszlanak el az értékek, ezért is esik a görbe alatti terület értéke a 0.5 közelébe. Az F1 görbén látszik, hogy a legmagasabb pontszámot 0.245-ös konfidencia értéknél veszi fel a függvény, viszont 0.4-es értékig nem csökken drasztikusan. Gyakorlati alkalmazásnál érdemes lenne emiatt a konfidencia küszöbértéket 0.4-re állítani, hiszen még ott is nagy az egyensúly a pontosság és visszahívás között.



22. ábra : YOLOv5l hiba mátrixa a nyers adatkészleten



23. ábra : YOLOv5l PR görbéje a nyers adatkészleten



24. ábra : YOLOv5l F1 görbéje a nyers adatkészleten

A 21., 22., 23. ábrákon látszik, hogy a YOLOv5l hogyan teljesített a nyers adatkészleten. A hiba mátrixról hasonlóak mondhatók el, mint a kisebb hálózatról. A PR görbén jól látszik, hogy itt az értékek inkább már a jobb felső sarok közelében tartózkodnak, amit a 0.655-ös átlag pontosság érték is megerősít. Az F1 görbén 0.66 a legmagasabb pontszám 0.356-os konfidencia érték mellett, viszont a másik modellhez képest jóval magasabb konfidencia értékeknél kezd drasztikusan csökkenni a függvény értéke. Gyakorlati alkalmazáshoz érdemes lenne 0.5-0.6

között megválasztani a konfidencia küszöbértéket.

6.2. A modellek teljesítménye az eső zajterhelt adatkészleten

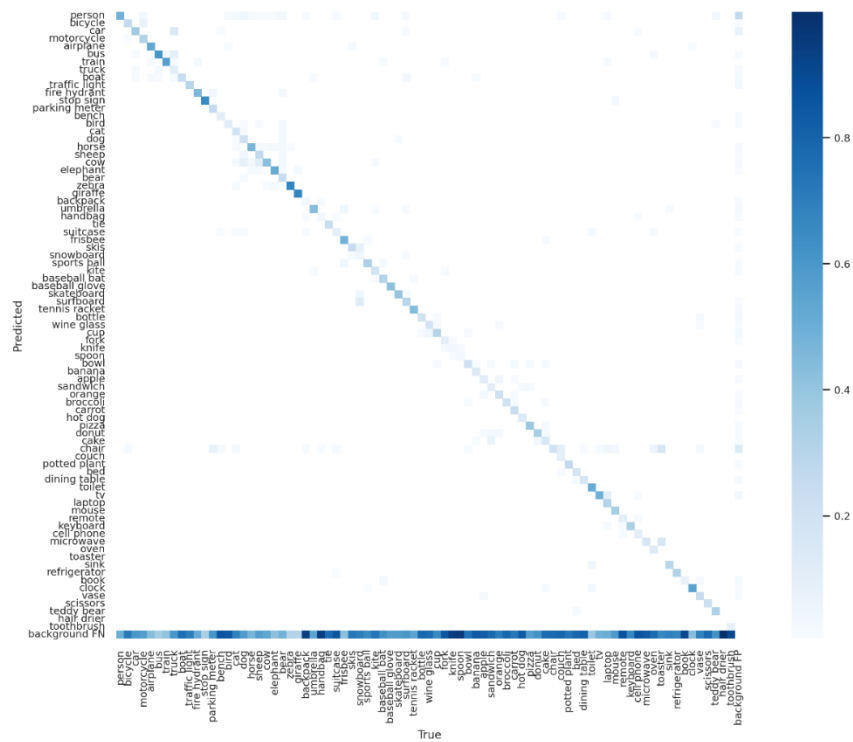
5. táblázat : Modellek sebessége és átlagos pontossága az eső adatkészleten

	YOLOv5 mérete	
	S	L
Predikció Sebessége	5,7 ms	14,1 ms
mAP @.5	0,455	0,552
mAP @.5:.95	0,317	0,408

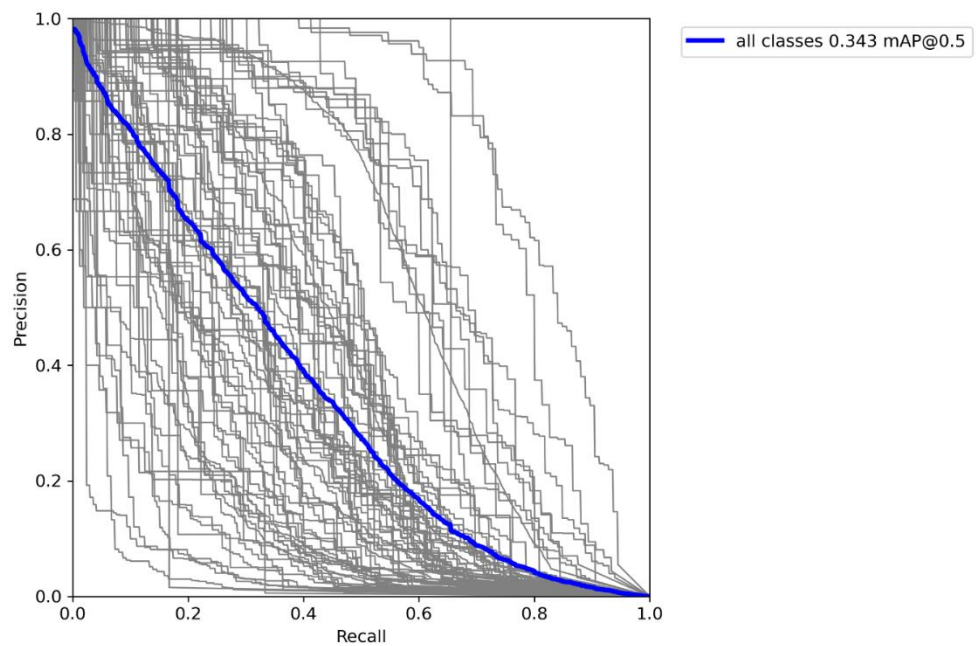
6. táblázat : Modellek átlagos pontossága az 5 legpontosabban prediktált osztályokra az eső adatkészleten

	YOLOv5 mérete	
	S	L
Top 5 osztály mAP @.5:.95		
Repülő	0,41	0,64
Elefánt	0,38	0,55
Zsiráf	0,53	0,64
Toalett	0,46	0,63
Laptop	0,29	0,49

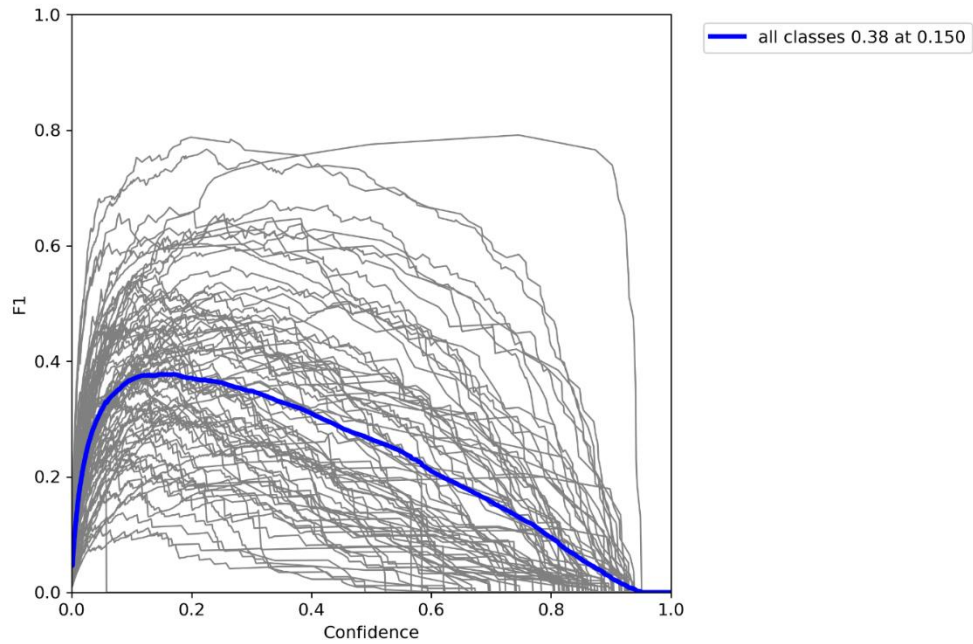
Az 5., 6. táblázatokban szereplő értékekből meghatározható, hogy a kisebb modell a nagyobbhoz képest 22% -kal rosszabb pontossággal rendelkezik az összes osztályra átlagolva, ez az érték a korábbi öt osztályhoz viszonyítva már 30% -ot jelent.



25. ábra : YOLOv5s hiba mátrixa az eső adatkészleten

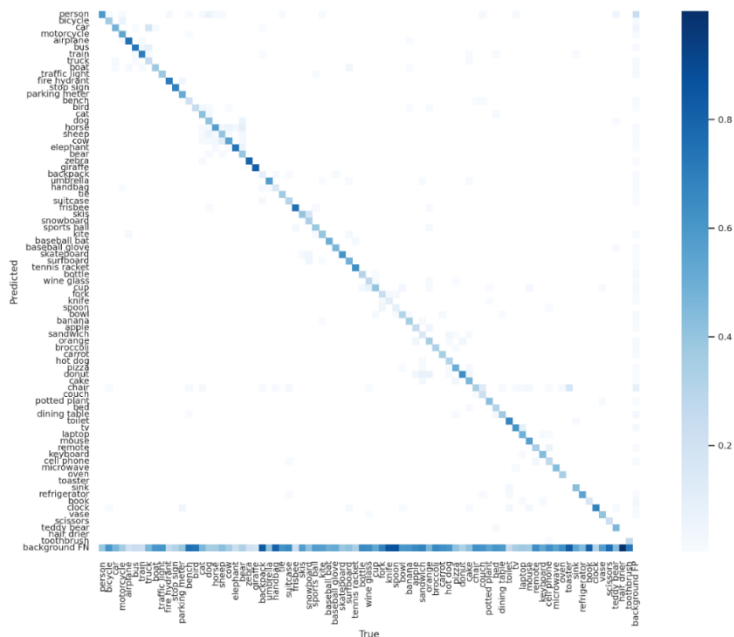


26. ábra : YOLOv5s PR görbéje az eső adatkészleten

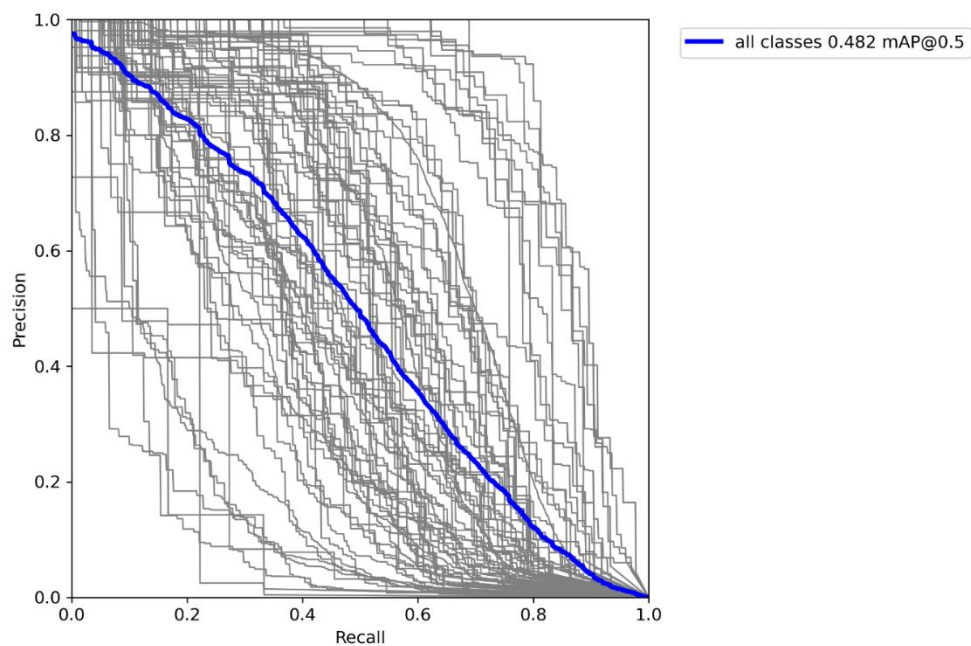


27. ábra : YOLOv5s F1 görbéje az eső adatkészleten

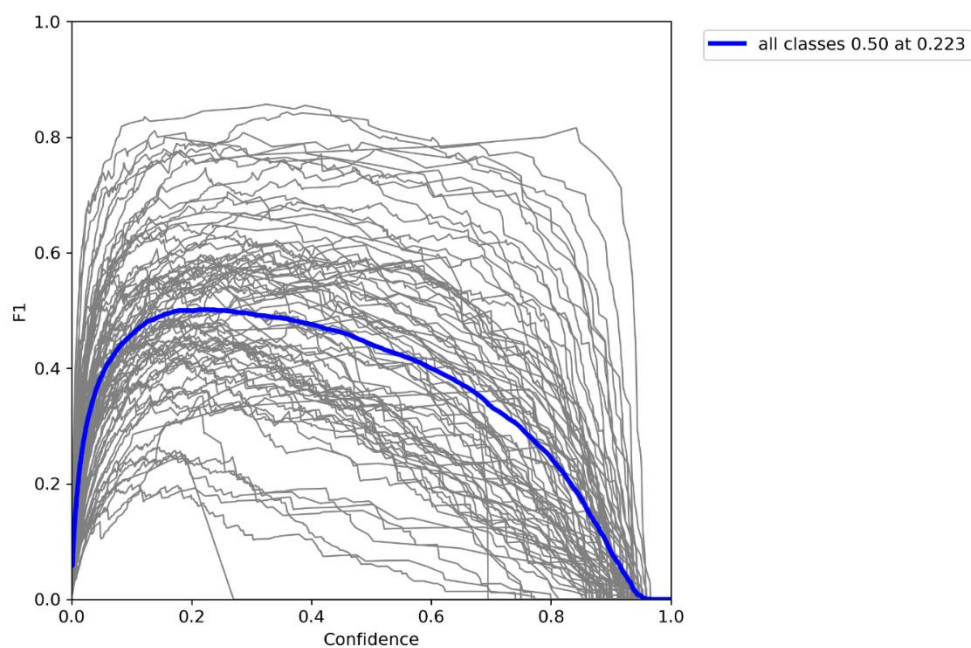
A 24., 25., 26. ábrákon látható, hogy a YOLOv5s modell hogyan teljesített az eső adatkészleten. A hiba mátrix főátlója a korábbihoz képest el van halványodva, ez alapján bizonytalanabb a hálózatunk helyes predikciók megítélésében. A PR görbén az osztályokat reprezentáló függvények a bal alsó sarok felé csoportosulnak inkább, viszont az összes osztályt reprezentáló görbe csökkenése 0.65-ös visszahívási értékig közel lineárisan csökken. Az F1 görbén az optimum helye kisebb konfidencia értéknél helyezkedik el és tartománya is jelentősen leszűkült.



28. ábra : YOLOv5l hiba mátrixa az eső adatkészleten



29. ábra : YOLOv5l PR görbéje az eső adatkészleten



30. ábra : YOLOv5l F1 görbéje az eső adatkészleten

A 27., 28., 29. ábrákon a YOLOv5l eredményei láthatóak az eső adatkészleten. A hiba mátrix főátlójában magasabb konfidencia értékek láthatóak, mint az S modell esetében. A PR görbe átlaga majdnem kiegyensúlyozott értéket vesz fel, és lineáris csökkenése 0.8-as visszahívási értékig tart, ami magasabb, mint a kisebb architektúránál. Az F1 görbe optimális tartománya nem romlott drasztikusan a nyers adatkészlethez képest.

6.3. A modellek teljesítménye az FGSM zajterhelt adatkészleten

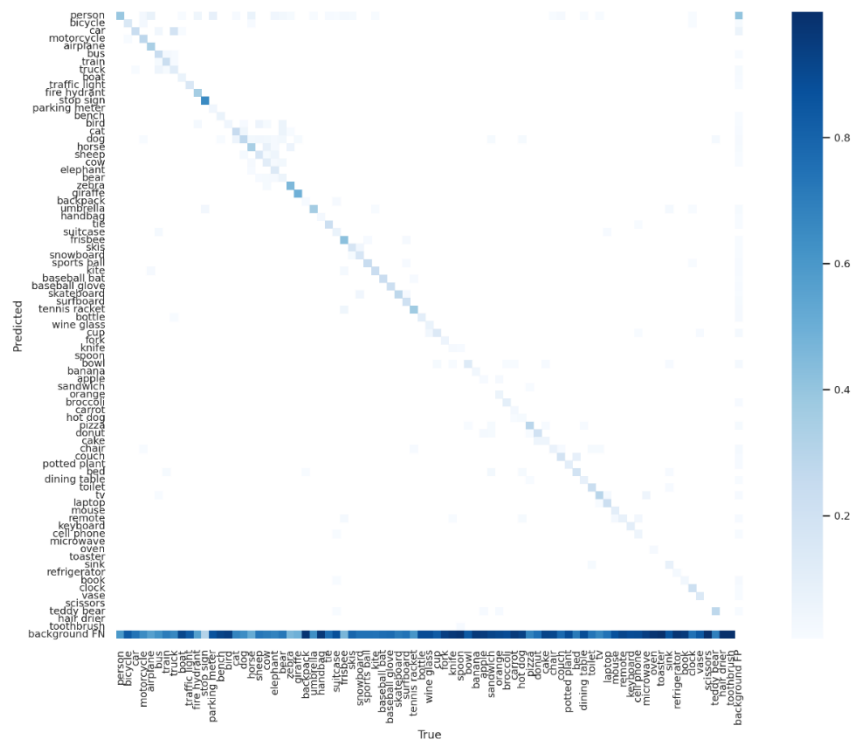
7. táblázat : Modellek sebessége és átlagos pontossága az FGSM adatkészleten

	YOLOv5 mérete	
	S	L
Predikció Sebessége	5,9 ms	14,1 ms
mAP @.5	0,373	0,522
mAP @.5:.95	0,255	0,385

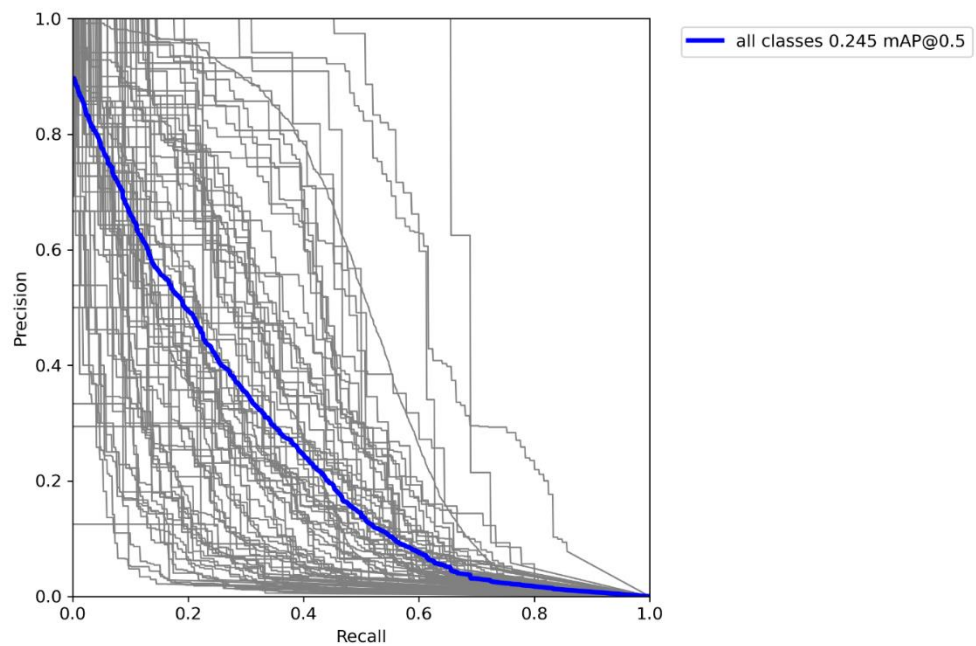
8. táblázat : Modellek átlagos pontossága az 5 legpontosabban prediktált osztályokra az FGSM adatkészleten

Top 5 osztály mAP @.5:.95	YOLOv5 mérete	
	S	L
Repülő	0,3	0,57
Elefánt	0,17	0,41
Zsiráf	0,43	0,54
Toalett	0,24	0,49
Laptop	0,22	0,46

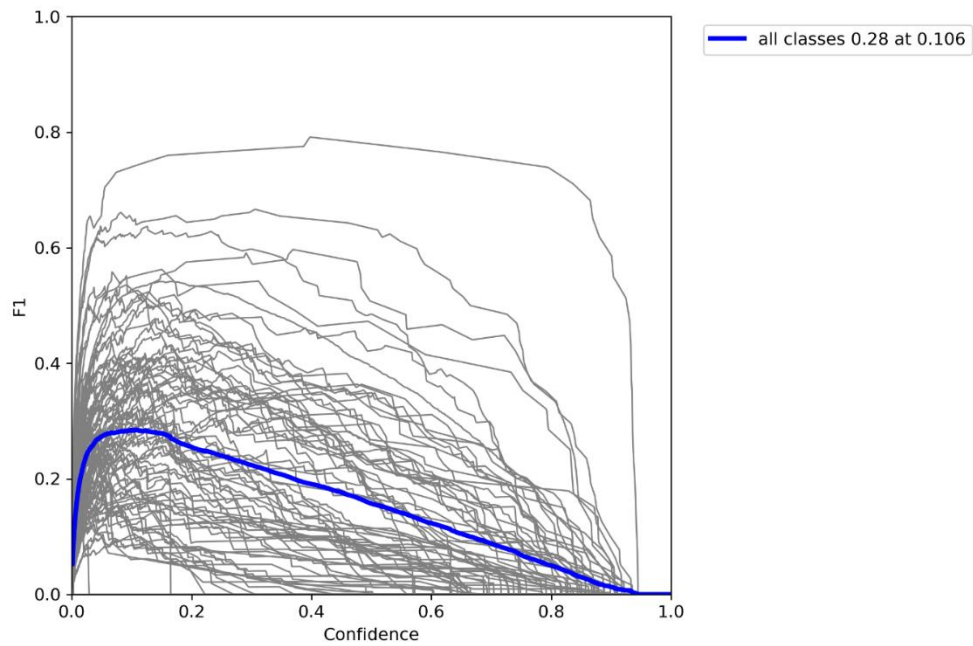
A 7., 8. táblázatok megmutatják, hogy mindkét konfiguráció a legrosszabb átlagos pontossággal rendelkezik a többi adatkészlethez képest. A kisebb architektúra itt már az összes osztályra nézve körülbelül 34% -kal pontatlanabb, az öt osztályos mérésnél pedig már 45% -kal pontatlanabb, mint a nagyobb hálózat.



31. ábra : YOLOv5s hiba mátrixa az FGSM adatkészleten

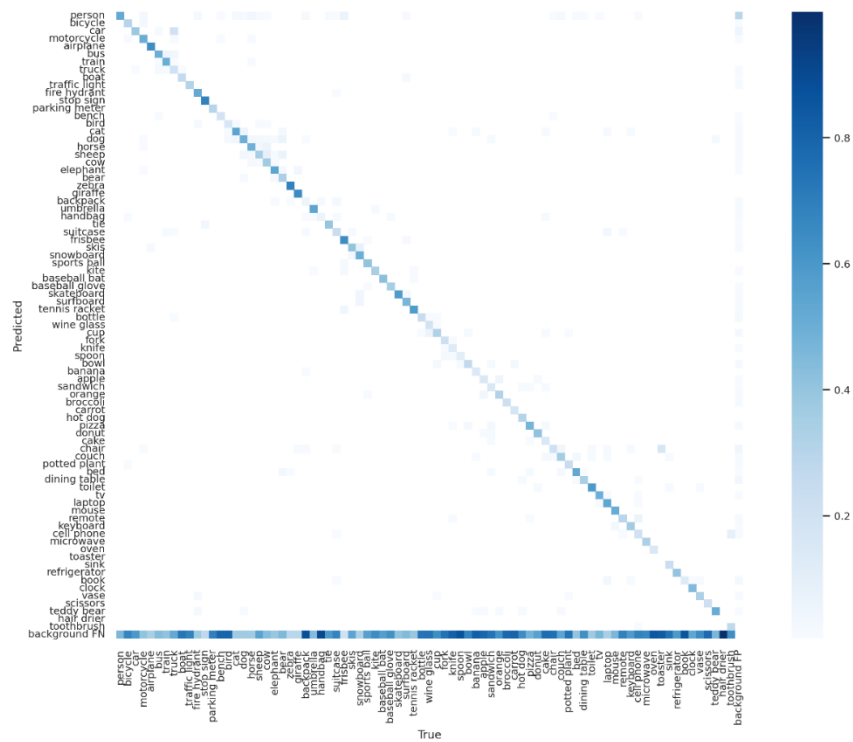


32. ábra : YOLOv5s PR görbéje az FGSM adatkészleten

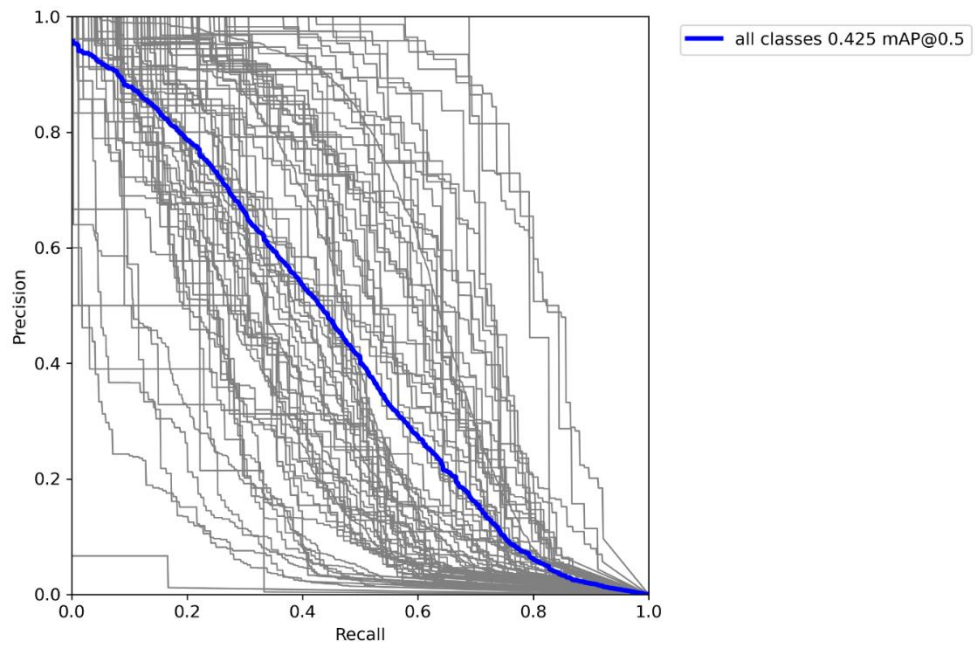


33. ábra : YOLOv5s F1 görbéje az FGSM adatkészleten

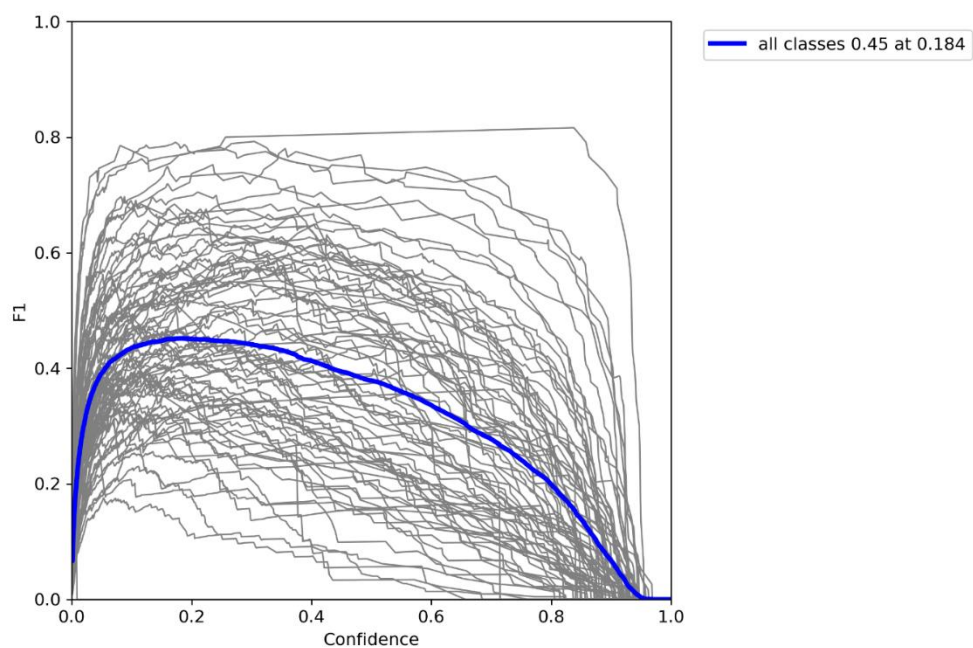
A 30., 31., 32. ábrákon jól látható, hogy a YOLOv5s eléggé bizonytalan az FGSM -el terhelt képeken szereplő objektumok megítélésében. A hiba mátrix főátlója több helyen hasonló konfidencia értékeket vesz fel, mint a hibás találatokat jelzett mezők. A legalsó sáv sötétsége arra utal, hogy sok objektumot háttérnek tekint a hálózat, ebből következően gyenge a visszahívási képessége. A PR görbe meredekebb, mint korábban, és linearitását is szinte teljesen elvesztette, az precízió értéke exponenciális csökken. Az F1 görbe alacsony csúcserőke után meredekebben közelít a nullához, mint korábban bármikor.



34. ábra : YOLOv5l hiba mátrixa az FGSM adatkészleten



35. ábra : YOLOv5l PR görbéje az FGSM adatkészleten



36. ábra : YOLOv5l F1 görbéje az FGSM adatkészleten

A 33., 34., 35. ábrákon látható a YOLOv5l teljesítménye az FGSM adatkészleten. A hiba mátrix főátlója nem halványult el annyira, mint a kisebb modellnél. A PR görbéje közel hasonlóan néz ki, mint az eső adatkészletnél, és nagy részében lineárisnak mondható az átlagolt függvény. Az F1 görbe picit ellaposodott, de jellegében hasonló, mint a másik zajterhelésnél.

7. Összefoglalás, a kapott eredmények kiértékelése

A feladatom az volt, hogy egy objektum detektáló neurális hálózat predikciós pontosságának változását vizsgáljam meg tanulható paraméter számának függvényében zajterhelt adatkészleteken. A célom az volt, hogy kiderítsem, hogy a nagyobb tanulható paraméterszámmal rendelkező modell zajterhelés esetén is jobban teljesít-e pontosságban, mint a kisebb tanulható paraméterszámmal rendelkező. Ezt a vizsgálatot azért tartottam fontosnak, mert jobb belátást adhat arra, hogy például egy önvezető jármű kamera alapú környezetfelismerő rendszere hogyan fog viselkedni rosszabb időjárási körülmények között. A kapott eredményeim az alábbiakban láthatóak.

A különböző adatszetteken történt kiértékelések során a modellek gyorsaságában nem volt változás. Ez várható volt, hiszen a képeknek csak a pixelértékei változtak, a méretük nem, így a feldolgozandó adatmennyiség sem változott. A tanulható paraméterek számainak függvényében a romlási százalékokat a 9. táblázatban láthatóak.

9. táblázat : A nyers adatkészleten történt kiértékeléshez képest megfigyelhető romlások százalékban kifejezve a zajterhelt adatkészleteknél

	A modell mérete			
	S		L	
	Eső	FGSM	Eső	FGSM
Minden osztály (mAP @.5)	-25 %	-39 %	-20 %	-25 %
Minden osztály (mAP @.5:.95)	-27 %	-41 %	-24 %	-29 %
Repülő (mAP @.5:.95)	-38 %	-55 %	-21 %	-30 %
Elefánt (mAP @.5:.95)	-40 %	-73 %	- 24 %	-43 %
Zsiráf (mAP @.5:.95)	-22 %	-37%	- 18 %	-31 %
Toalett (mAP @.5:.95)	-29 %	-63 %	-10 %	-30 %
Laptop (mAP @.5:.95)	-53 %	-65 %	-34 %	-38 %

Jól látszik, hogy a kisebb modell predikciós képessége már az eső hatással terhelt képeken is drasztikusan csökkent néhány osztálynál, amíg a nagyobb modellnél van, ahol alig csökkent az átlag pontosság a nyers adatkészlet eredményeihez képest. Az FGSM adatkészleten pedig sokkal jobban teljesített a nagyobb tanulható paraméterszámmal rendelkező modell. Ha gyakorlatban kéne alkalmazni a YOLOV5s architektúrát FGSM-hez hasonló zajjal rendelkező képeken, szinte használhatatlan lenne már pontatlansága miatt.

Összességében elmondható, hogy ha van egy modellünk, és többféle architektúra nagysággal

rendelkezik, akkor mindenféleképpen érdemes a nagyobb tanulható paraméterszámút választani olyan gyakorlati szintű alkalmazásokhoz, ahol tudjuk, hogy a kamerából érkező bemeneti képek többségén valamilyen szintű zaj is létezik, legyen az időjárási viszonyok miatt, vagy egyéb más körülmények miatt. Akkor is érdekesebb a nagyobb modellt választani, ahol előre nem meghatározott módon kell majd felismerni az objektumokat. Például egy autonóm működtetésű járműnél sok környezeti változó befolyásolhatja a kameraképünk tisztaságát, vagy az éppen a megjelenő objektumok idealizáltságát. Ilyen körülmények között is egy minél robosztusabb környezetérzékelési módszerre van szükség. Igaz, hogy a bonyolultabb felépítésű modellek nem mindig futtathatók alapjáraton valós időben, viszont ma már sok optimalizációs eszköz létezik, amikkel le lehet csökkenteni a feldolgozási időt, feljebb bemutatott eredményeim alapján érdemes ezekre az optimalizációkra időt fordítani.

8. Irodalomjegyzék

- [1] Bishop, C. M.: *Neural Networks for Pattern Recognition*, „Könyv”: Clarendon Press, Oxford, pp. 77-380, 1995
- [2] Gardner, M.W.; Dorling, S. R.: *Artificial neural networks (the multilayer perceptron) – a review of applications in the atmospheric sciences*, Atmos. Environ., 32 (14-15), pp. 2627-2636, 1998
- [3] Jianxin, W.: *Introduction to Convolutional Neural Networks*, „Egyetemi jegyzet”: Nanjing University, National Key Lab for Novel Software Technology, pp. 5-25, 2017
- [4] Xu, R.; Lin, H.; Lu, K.; Cao, L.; Liu, Y.: *A Forest Fire Detection System Based on Ensemble Learning*, „Cikk”: Forests. 12(2):217, 2021
- [5] Lin, T.; Maire, M.; Belongie, S.; Bourdev, L.; Girshick, R.; Hays, J.; Perona, P.; Ramanan, D.; Zitnick, C. L.; Dollár, P.: *Microsoft COCO: Common Objects in Context*, „Konferenciakiadvány”: European Conference on computer vision, Zurich, Switzerland, pp. 740-755, 2014
- [6] Goodfellow, I. J.; Shlens J.; Szegedy, C.: *Explaining and Harnessing Adversarial Examples*, „Konferenciakiadvány”: International Conference on Learning Representations, pp. 1-7, 2015
- [7] Wang, C. Y.; Mark Liao, H. Y.; Wu, Y. H.; Chen, P. Y.; Hsieh, J. W.; Yeh, I. H.: *CSPNet: A new backbone that can enhance learning capability of cnn*, „Konferenciakiadvány”: Conference on Computer Vision and Pattern Recognition (CVPR 2020), Washington, DC, USA pp. 390-391, 2020
- [8] Wang, K.; Liew, J. H.; Zou, Y.; Zhou, D.; Feng, J. Panet: *Few-shot image semantic segmentation with prototype alignment*, „Konferenciakiadvány”: International Conference on Computer Vision (ICCV 2019), Seoul, Korea, pp. 9197-9206, 2019
- [9] Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmasion, A.; Köpf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; Chintala, S.: *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, „Konferenciakiadvány”: 33rd Conference on Neural Information Processing Systems, Vancouver, Canada pp. 2-9, 2019

- [10] He, K.; Zhang, X.; Ren, S.; Sun, J.: *Deep residual learning for image recognition*, „Konferenciakiadvány”: Conference on Computer Vision and Pattern Recognition, pp. 770-778, 2016
- [11] Saxena, U.: *Automold Road Augmentation Library*: github.com/UjjwalSaxena/Automold—Road-Augmentation-Library (Letöltési dátum: 2021.11.01)
- [12] Jocher, G.: *YOLOv5*: github.com/ultralytics/yolov5 (Letöltési dátum: 2021.11.01)