

Debreceni Egyetem
Műszaki Kar
Villamosmérnöki és Mechatronikai Tanszék



Tűz és füst felismerés gépi tanulás segítségével

Szerző: **Balogh Máté BSc hallgató**

Témavezető: Sarvajcz Kornél

Tanszékvezető: Prof. Dr. Husi Géza

Debrecen
2024.

Tartalomjegyzék

1. Bevezetés/Inspiráció	4
2. Tűz és füst felismerés technológiája	6
2.1. Szenzorok előtt	6
2.2. Tűz és füst felismerése szenzorokkal	6
2.2.1. Bimetál hőmérő	6
2.2.2. Ionizációs füstérzékelő	7
2.2.3. Fotoelektromos tűzérzékelők	7
2.3. Tűz és füst felismerése képeken	7
3. Mesterséges neurális hálózatok.....	8
3.1. Gradiens ereszkedés	9
3.1.1. Gradiens ereszkedés problémák	9
3.2. Backpropagation (visszaterjesztés).....	10
3.3. Aktivációs függvények feladata és fajtái.....	13
4. Convolutional Neural Networks (CNNs) és YOLO11	16
4.1. A convolutional neural network előnyei	16
4.2. Általános szerkezet	16
4.3. YOLO11 szerkezete	17
4.3.1. Backbone	18
4.3.2. Neck	18
4.3.3. Head	19
4.4. Transfer learning.....	19
4.4.1. Transfer learning használata.....	19
5. Nevesebb nemzetközi folyóiratok feldolgozása	21
6. Tűz és füst felismerés gépi tanulással	22
6.1. Absztrakt.....	22

6.2.	A modell teljesítmény mutatói	22
6.3.	YOLOv8m és YOLO11m betanítása és eredményei	24
6.3.1.	Data augmentáció használata	27
6.3.2.	A képek számának növelése.....	29
6.3.3.	Növelt képszám augmentációval.....	31
6.3.4.	YOLO11m augmentációval	33
7.	Konklúzió.....	37

Tűz és füst felismerés

1. Bevezetés/Inspiráció

Az ember nagyon rég óta kontaktusba van a tűzzel, körülbelül egymillió éve tudatosan is használja azt. A tűz nagyon hasznos számunkra, és hasznos volt mindig is, a Neandervölgyi idejében szerszámkészítésre, később edények és téglák kiégetésére használtuk, napjainkban pedig a gázkonvektoroktól, a kontrollált égetéseken át rengeteg mindenre használjuk.[1] Viszont a hasznossága ellenére, pusztító is tud lenni, Debrecenben 1802. június 11-én a város nagyrésze leégett, valamint az elmúlt években rendszeresen értesülünk erdőtüzekről.[2]

A gépi tanulás elérése és használata egyre egyszerűbbé és könnyebbé válik napjainkban köszönhetően a technológia fejlődésének, ez azt jelenti hogy lehetőségünk van használni rengeteg szituációban. Az egyik ilyen szituáció amiben ezt a technológiát érdemes használni az a tűz illetve füst felismerés. Minél hamarabb érzékeljük egy tüzeset kialakulását annál hamarabb tudunk beavatkozni ez által életet és értéket megmentve. Ez a detektálási módszer bizonyos körülmények között az eddig használtaknál, jobb lehet, ezért érdemes erről az oldalról is megközelíteni a tűz detektálást mivel olyan előnyöket nyújthat ez a módszer, a korábbi módszerekkel szemben amik hasznosak lehetnek.

A manapság elterjedt automatikus tűzoltó berendezések nem képesek a tűz helyének meghatározására, ezért területileg egyenletesen bocsájtják ki az oltóanyagot(pl.: víz, hab ,por). Gépi tanulás, objektum detektálás és egy kamera rendszer segítségével lehetőségünk lenne a tűz helyének a pontos meghatározására. Ha megtudjuk határozni a tűz helyét akkor közvetlenül oda tudjuk kibocsájtani az oltóanyagot, ez a hagyományos automatizált tűzoltási módszerrel szemben jóval hatékonyabb. Nem csak a tüzet tudjuk eloltani gyorsabban, kisebb anyagfelhasználással mellett, de például lehetőség szerint a víz nem tesz kárt az elektromos berendezésekben.

Ezt a tűz felismerési és helymeghatározási technológia sok esetben hasznos lehet, ebből kettőt szeretnék kiemelni, az egyik a sprinkler rendszerek fejlesztése, a másik pedig egy hang alapú figyelmeztető rendszer létrehozása. Ha egy kamera segítségével meg tudjuk határozni a tűz helyét egy szobában vagy épületben akkor létre lehetne hozni egy olyan rendszert amelyben a sprinkler nem kör alakban bocsájt ki vizet hanem csak a tűz irányába például körcíkk vagy félkör alakba. Természetesen az előfordulhat hogy különböző okok miatt, mint például a kamera pozíciója vagy a füst mennyisége, nem érzékeli a tűzfelismerő rendszer a tüzet, ebben

az esetben a sprinkler a megszokott módon működne. Mivel a sprinkler rendszerek egy adott hőmérséklet elérése után kezdenek el oltani egy elég biztos tűzfelismerő modellt használva megkezdődhetne hamarabb a sprinkler által a víz kibocsátása, ezzel is csökkentve a kár és veszély mértékét. A hang alapú figyelmeztető rendszer a tűz és füst felismerő modell illetve hangszórók segítségével működne úgy, hogy ha egy kamera az adott épület egyik részében például termében vagy szobájában tüzet és vagy füstöt érzékel akkor a hangszórók az összes épületben tartózkodó ember tudtára adják hogy melyik helységben alakult ki a veszélyhelyzet. Ha van „A” „B” és „C” szoba az épületben akkor ha az „A” szobában tüzet vagy füstöt érzékel a rendszer akkor, olyan hangalapú jelzést lehetne kibocsátani, mint például a „tűzeset az „A” szobában”. Ez a hangalapú rendszer segítséget nyújtana a veszély elkerülésében illetve a menekülési útvonal meghatározásában.

Úgy gondolom hogy, egy elég biztos és pontos objektum felismerő modellel, a régi módszerekhez képest jobb tűzoltó rendszert lehetne létrehozni. Az előzőekben megfogalmazott hipotézis inspirálta a kutatómunkámat, és azt tűztem ki magamnak célul, hogy fejlesszek egy olyan rendszert, amely gépi tanulás alkalmazásával képes automatikusan detektálni a tüzet és füstöt.

2. Tűz és füst felismerés technológiája

2.1.Szenzorok előtt

Az első tűz elleni védekezést a romaiaknál lehet felfedezni, többször is leégett Róma ezért Néró császár elrendelte hogy tűzálló anyagból építsék a házakat, épületeket. A tűz detektálására nem volt eszközük, csak a látásukkal tudták felismerni. Ez szintén így volt a 17. században Londonban amikor is „a nagy londoni tűzvész” során a város kb. 80%-a leégett. A tűzvész után Londonba, kézi vízpumpákat és tűcsapokat telepítettek.[3] A tüzeset kialakulását sokáig sípokkal és vagy, a templomok harangjaival jelezték. A távíró(telegráf) felfedezése után 1852-ben Moses Farmer és William F. Channing létrehozta egy riasztó rendszert amivel egy kar tekerésével a legközelebbi tűzoltóság információt kapott a tűz helyéről így gyorsabban tudták elkezdni az oltást. Manapság sokféleképpen tudjuk értesíteni a tűzoltóságot, például, falra szerelt tűzriasztóval és a telefonunkkal.[4]



1. ábra: Tűzvédelmi kézi jelzésadó falon kívüli tűzriasztó [5]

2.2.Tűz és füst felismerése szenzorokkal

A 2.2 fejezet, a tűz és a füst felismerésének az érzékelőit mutatja be részletesen.

2.2.1. Bimetál hőmérő

A működése a fémek hőtágulási tényezőjén alapul. Ha két eltérő hőtágulási tényezőjű fémeket egymás mellé teszünk, akkor melegítés hatására a kisebb hőtágulási tényezőjű fém felé fog hajolni a szerkezet. Ezzel a technikával a hőváltozást tudjuk mérni viszont ez nem túl gyors módja a tűz felismerésének.[3]

2.2.2. Ionizációs füstérzékelő

1930-ban egy svájci fizikus fedezte fel, Walter Jaeger egy mérges gáz érzékelő szenzor szeretett volna létrehozni, ez nem sikerült, viszont a cigarettája füstjére bejelzett az által készített készülék. A szenzor érzékelt az ionok áramlásában keletkezett változásokat és mivel a füst megzavarta az áramlást ezért ezzel a módszerrel érzékelni lehetett a füst jelenlétét. Az 1940-es években, Ernst Meil-el együtt szabadalmaztatták a felfedezést, és 1942-ben megindult a gyártása.[3]

Az 1960-as években, az amerícium elem 241-es számú izotópját kezdték el használni füstérzékelőkben. Az amerícium 241 bomlása során alfa részecskéket bocsát ki magából ami pozitívan töltött, ez által töltésáramlást tud létrehozni. A füst részecskék megzavarják az alfa részecskék áramlását, ezt a változást pedig az eszköz érzékelni tudja. Az ilyen elven működő, amerícium 241-et használó füstérzékelőket ma is széles körben használják.[3]

2.2.3. Fotoelektromos tűzérzékelők

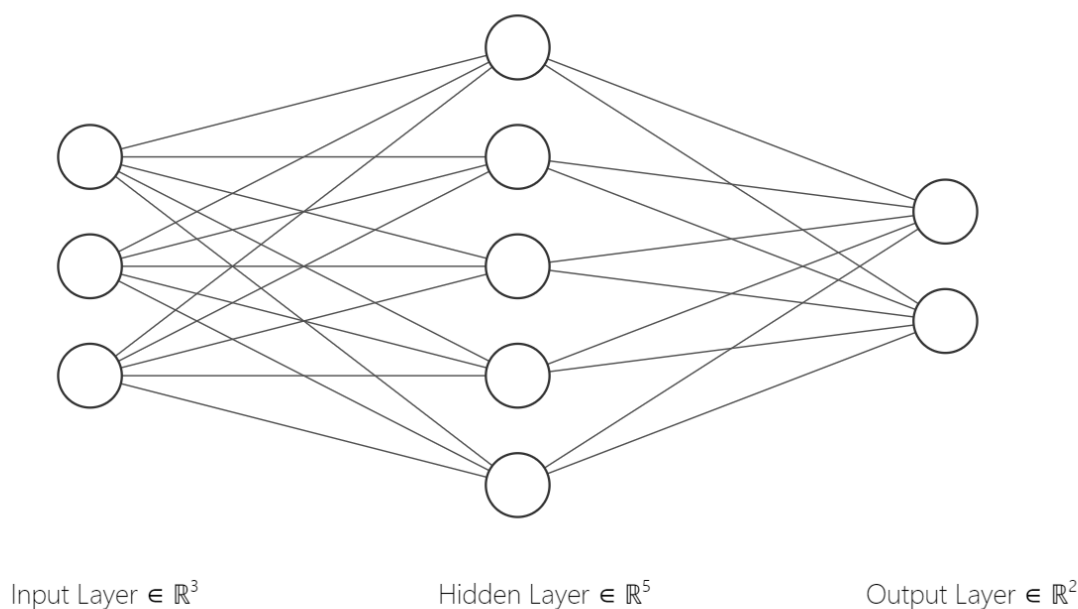
A fotoelektromos füstérzékelők úgy működnek hogy, van egy fényforrás ami a modern füstérzékelőkben általában LED, illetve van egy fény érzékelő szenzor, ezek kb. 90 fokban vannak egymáshoz képest. Amikor füst kerül a készülékbe, a fényforrás által kibocsátott fény megtörik, ez által eléri a szenzor ami pedig ez esetben jelet küld a riasztónak.[3]

2.3. Tűz és füst felismerése képeken

Ahhoz hogy egy kamerával detektálni tudjuk automatikusan a tüzet és füstöt ahhoz szükségünk van egy objektum detektáló modellre. Mivel egy kamerának a felvétele az csak képeknek a sorozata ezért ezt a modellt képek segítségével be lehet tanítani. Ezeken a képeken az objektumok „körvonalának” észlelése történik ami alapján más hasonló körvonalú objektumokat fel fog tudni ismerni. Ezt a folyamatot, technológiát machine learning-nek azon belül is deep learning-nek nevezzük. A deep learning, komplex mesterséges neurális hálók „tanításával” foglalkozik, az 1980-as, 90-es években népszerű volt ez a technológia, viszont a kisteljesítményű számítógépek és a nagymennyiségű adat hiánya megakadályozta a fejlődését. 2006-ban kezdtek el megint foglalkozni ezzel a témával ekkor már az adat és a számítógépes teljesítmény hiánya nem volt probléma. [6]A nagyméretű képek feldolgozásához konvolúciós neurális hálózatokat használnak amiknek a segítségével gyorsabb és hatékonyabb a tanulási folyamat. [1*]

3. Mesterséges neurális hálózatok

A mesterséges neurális hálózatok (Artificial neural networks)(ANNS) az agyi neurális hálózathoz nagymértékben hasonlítanak. Rétegekbe rendezett mesterséges neuronokból állnak, egy réteg összes neuronja össze van kapcsolva a következő réteg összes neuronjával. Ezek a neuronok végzik a számításokat előre meghatározott feltételek alapján, és küldenek jelet a neuronnak a következő rétegbe. A különbség a mesterséges és a mély neurális hálózatok között az hogy míg a mesterséges neurális hálózatok három rétegből állnak, név szerint bemeneti réteg(input layer), rejtett réteg(hidden layer), kimeneti réteg(output layer), addig a mély neurális hálózatoknak több mint egy rejtett rétege van.[7]



2. ábra: Mesterséges neurális hálózatok szerkezeti felépítése

Jelöljük a bementi réteg neuronjait rendre, $a_0^{(0)}, a_1^{(0)}, \dots, a_n^{(0)}$ (ahol az alsó index a rétegen belül a neuron számát a felső index pedig a rétegnek a számát jelöli) ezek egy adott értéket jelölnek, a súlyokat(weights) pedig $w_0^{(0)}, w_1^{(0)}, \dots, w_n^{(0)}$ -ként(ahol az alsó index azt jelöli hogy melyik neuronhoz tartozik a rétegen belül, míg a felsőindex arra vonatkozik hogy a következő rétegből melyik neuronra vonatkoznak ezek a súlyok). Ezeket összeszorozva majd összeadva és végül egy befolyásoló tényező(bias) hozzáadásával minden megvan a rejtett réteg neuronjának az aktivációs függvényéhez. Ennek az aktivációs függvénynek az értéke fog „tovább haladni” a

neurális háló következő rétegéhez. A rejtett réteg első neuronjának az értéke (az ábrán látható leegyszerűsített esetben): [7]

$$a_0^{(1)} = f(w_0^{(0)} * a_0^{(0)} + w_1^{(0)} * a_1^{(0)} + w_2^{(0)} * a_2^{(0)} + b_0) \quad (1)$$

N-számú bemeneti neuron esetében:

$$a_0^{(1)} = f(w_0^{(0)} * a_0^{(0)} + w_1^{(0)} * a_1^{(0)} + \dots + w_n^{(0)} * a_n^{(0)} + b_0) \quad (2)$$

A súlyok(weights) számokat jelölnek amik meghatározzák hogy a neuronok között milyen erős a kapcsolat, vagyis hogy melyik neuron mennyire „aktiválódik” be, ez által nagy szerepük van a végső kimenet meghatározásában.[7]

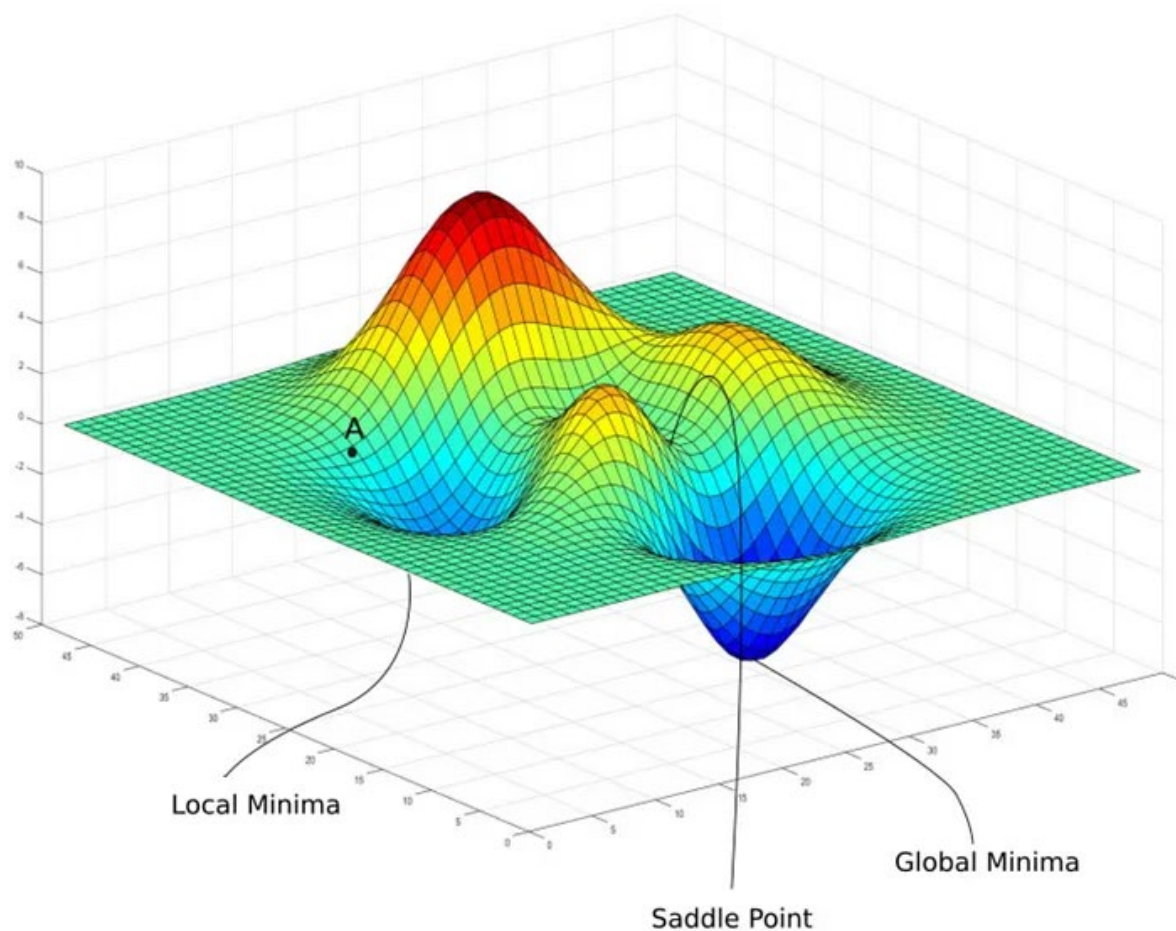
3.1. Gradiens ereszkedés

A gradiens ereszkedés a hiba függvény egyik minimumának megkeresésére utal ezt a minimumot a súlyok és bias változtatásával érjük el. A hiba függvény deriváltjának a tanulási folyamat elején viszonylag nagy a meredeksége, viszont a tanulási folyamat előrehaladtával egyre csökken míg végül nulla nem lesz. Minden esetben amikor változnak a súlyok és bias-ok azt akarjuk elérni hogy a legmeredekebb irányba „menjünk” mivel így érhetjük el a leggyorsabban a minimumot.

Tanulási mérték: A hiba függvény viselkedése alapján változik, ha túl nagy akkor nem biztos hogy elérjük a minimumot mivel „túlmegyünk” rajta, ha pedig túl kicsi akkor sok idő és számítás mire a minimumhoz érünk.

3.1.1. Gradiens ereszkedés problémák

Lokális minimum és nyereg probléma: Konvex példák esetén könnyű megtalálni a globális minimumot viszont, ha nem konvex esetről beszélünk megakadhatunk lokális minimumokon és nyeregpontokon mivel ezekben az esetekben is nulla az érintő meredeksége. Ilyen fajta megakadásokat a megfelelő nagyságú tanulási léptékkal kiküszöbölhetünk.[8]



3. ábra: Gradiens ereszkedés [9]

Eltűnő gradiens: Ha kezdetben a gradiens nagyon kicsi ahogy a visszaterjesztés során haladunk visszafelé a gradiens egyre kisebb lesz ezáltal az előző rétegek egyre lassabban tanulnak, ami azt eredményezi hogy a súlyok nullához fognak közelíteni és végül teljesen megáll a tanulás.

Felrobbanó gradiens: Ha a gradiens túl nagy a súlyok értelmezhetetlenül nagy számok lesznek ez által használhatatlanok.[8]

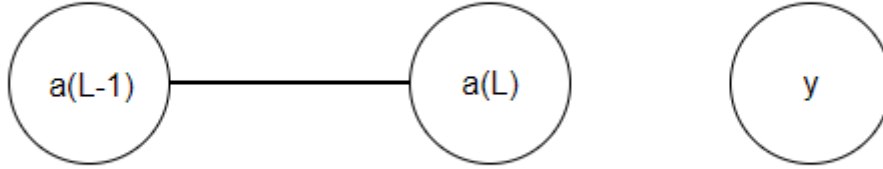
3.2.Backpropagation (visszaterjesztés)

A visszaterjesztés vagy hibavisszaterjesztés a mesterséges neurális hálók tanulásának a kulcsfontosságú eleme. A tanulási folyamat két lépés ismétléséből áll: 1.forward pass (kimenet számítás) 2. backpropagation (hiba visszaterjesztés), ezek addig ismétlődnek amíg a legkisebb hibaértéket kapjuk a súlyok illetve a bias változtatásával.

Egy egyszerűsített példát nézve, a hiba függvényt a hálózat neuronjának értéke és a kívánt érték különbségének négyzeteként tudnánk megadni:

$$C_0 = (a^{(L)} - y)^2 \quad (3)$$

Ahol $a^{(L)}$ az utolsó réteg neuronja, y a kívánt érték, C_0 pedig a hibafüggvény.[10]



4. ábra: Az utolsó, utolsó előtti neuron és a hibaérték

A hibavisszaterjesztés célja az hogy, megkapjuk a hiba(hibák) – súlyok, hiba – biasok, hiba – neuron aktiválási értékek kapcsolatát. Ha a fenti le egyszerűsített példát vesszük akkor felírhatjuk hogy:

$$a^{(L)} = \sigma(w^{(L)} * a^{(L-1)} + b^{(L)}) \quad (4)$$

Ezzel megadtuk az L rétegben lévő neuron értékét ahol $w^{(L)}$ a hozzá tartozó súly $a^{(L-1)}$ az előző réteg neuronjának értéke, $b^{(L)}$ az $a^{(L)}$ -hez tartozó bias, σ pedig a szigmoid aktivációs függvény.

Kapcsolat a hiba és a súly között:

$$\frac{\partial C_0}{\partial w^{(L)}} \quad (5)$$

A megfelelő deriválási szabályok után:

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial(w^{(L)} * a^{(L-1)} + b^{(L)})}{\partial w^{(L)}} * \frac{a^{(L)}}{\partial(w^{(L)} * a^{(L-1)} + b^{(L)})} * \frac{\partial C_0}{\partial a^{(L)}} \quad (6)$$

Egyenlet tagjai kifejezve:

$$\frac{\partial(w^{(L)} * a^{(L-1)} + b^{(L)})}{\partial w^{(L)}} = a^{(L-1)} \quad (7)$$

$$\frac{a^{(L)}}{\partial(w^{(L)} * a^{(L-1)} + b^{(L)})} = \sigma'(w^{(L)} * a^{(L-1)} + b^{(L)}) \quad (8)$$

$$\frac{\partial C_0}{\partial a^{(L)}} = 2 * (a^{(L)} - y) \quad (9)$$

Behelyettesítés után:

$$\frac{\partial C_0}{\partial w^{(L)}} = a^{(L-1)} * \sigma'(w^{(L)} * a^{(L-1)} + b^{(L)}) * 2 * (a^{(L)} - y) \quad (10)$$

Kapcsolat a bias és hiba érték között:

$$\frac{\partial C_0}{\partial b^{(L)}} \quad (11)$$

A megfelelő deriválási szabályok után:

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial(w^{(L)} * a^{(L-1)} + b^{(L)})}{\partial b^{(L)}} * \frac{a^{(L)}}{\partial(w^{(L)} * a^{(L-1)} + b^{(L)})} * \frac{\partial C_0}{\partial a^{(L)}} \quad (12)$$

Alakítás után:

$$\frac{\partial C_0}{\partial b^{(L)}} = 2 * (a^{(L)} - y) * \sigma'(w^{(L)} * a^{(L-1)} + b^{(L)}) * 1 \quad (13)$$

Az összes tanulási példát átlagolva megkapjuk a gradiens vektor egy komponensét:

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}} \quad (14)$$

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial b^{(L)}} \quad (15)$$

Ezekből a komponensekből fog állni a gradiens $(-\nabla C)$ (ami egy vektor-értékű többváltozós függvény) ami megadja a legnagyobb változás(legmeredekebb irányt) irányát, ha ebbe az irányba „lépünk”(változtatjuk a súlyt, bias-t) akkor csökken a hiba értéke, tehát tanul a modell.

[11][10]

3.3. Aktivációs függvények feladata és fajtái

Aktivációs függvényre azért van szükség mert nem-linearitást vezet be a rendszerbe amire szükségünk van valós felhasználás esetén. Az aktivációs függvény értéke adja a neuron aktiválásának a mértékét.

Manapság a legelterjedtebb aktivációs függvény a ReLU(Rectified Linear Unit) aminek előnye a deriválhatóság ezáltal a hibákat tudjuk visszaterjeszteni, illetve az értékének a kiszámítása könnyebb a többihez képest. [12]

Lineáris függvény:

Hibavisszaterjesztés során ugyan azzal az értékkel szoroznánk be az összes súlyt és bias-t. valamint A bemeneti értékeknek a kimeneti értékek konstans szorosai ez által a több rejtett réteg akár egy rejtett rétegnek is tekinthető. Nem lenne értelme lineáris aktivációs függvényt használni mivel nem tanulna a rendszer. Az (16) egyenlet megadja a lineáris aktivációs függvényt.[12]

$$f(x) = x \quad (16)$$

Nem – lineáris aktivációs függvények

Lehetővé teszik a hibavisszaterjesztést, mivel így meg tudjuk határozni hogy melyik neuront(neuronokat) kell változtatni a jobb eredmény érdekében. Több réteget tudunk használni mivel nem lineáris kapcsolat lesz közöttük ez által komplex modelleket hozhatunk létre.[12]

Sigmoid függvény:

Széles körben használt függvény, a kimeneti értéke nulla és egy között van, nagyon nagy bemenetek esetén az értéke egyhez közelít, nagyon kicsik esetén pedig nullához. Két problémába ütközhetünk a használata során, az eltűnő gradiens problémába, illetve egy ahhoz hasonló problémába ami során túl nagy számok esetén adott változásra nagyon kicsit változik a függvény értéke ami a tanulás lelassulásához vezet. Ezek a problémák a komplexebb neurális hálózatoknál fordulnak elő leginkább.[12] A (17) egyenlet megadja a Sigmoid aktivációs függvényt.

$$f(x) = \frac{1}{1 + e^{(-x)}} \quad (17)$$

Tanh függvény

Hasonlít a szigmoid függvényhez annyi a különbség hogy átmegy az origón, az értéke -1 és 1 között van illetve 0-nál nagyobb a meredeksége. Általában ezt preferálják a szigmoid függvénnyel szemben, szinte minden esetben gyorsabb a tanulása rendszernek mivel fel tud venni negatív értékeket.[13] A (3.) egyenlet megadja a tangens hyperbolicus aktivációs függvényt.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (18)$$

Rectified linear unit (ReLU) függvény

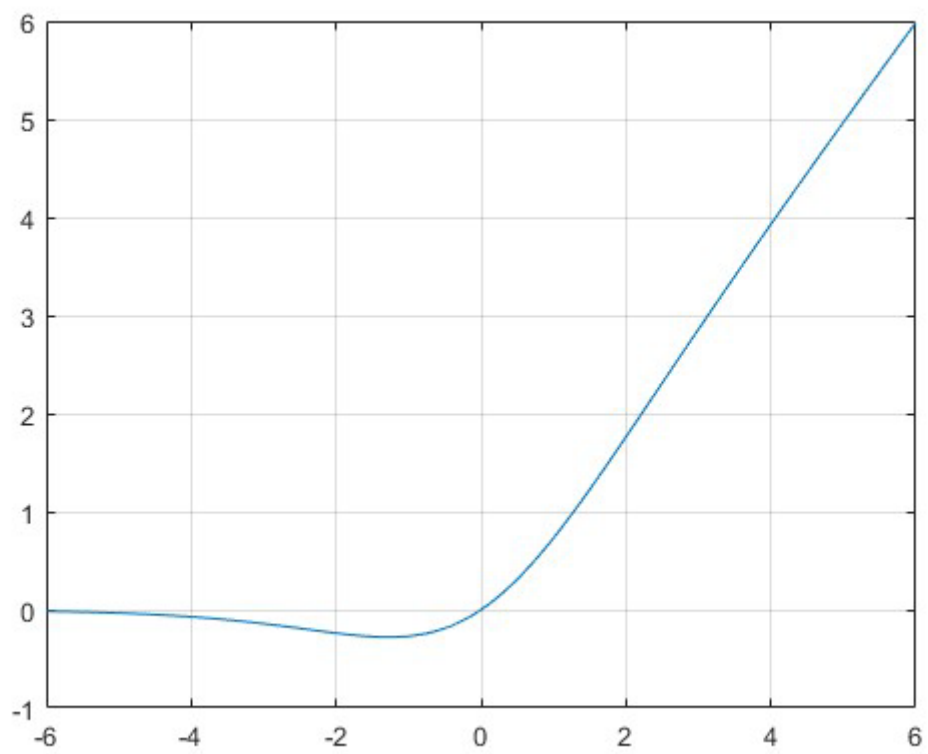
Legnagyobb előnye az egyszerűség ez által a gyorsaság, e-miatt nagyon népszerű. Azért tud gyors lenni mivel általában nincsen beaktiválva az összes neuron egyszerre, csak azok amiknek az adott lefutás során a bemenő jele nagyobb mint nulla. Az origón kívül minden pontban deriválható, illetve az értékkészlete: $[0 ; -\infty]$. [14] Az (4.) egyenlet megadja a ReLU aktivációs függvényt.

$$f(x) = \max(0, x) \quad (19)$$

Sigmoid-weighted Linear Unit(SiLU)(swish) függvény

A YOLOv8 és YOLO11 objektum felismerő mélytanulási modellek aktivációs függvénye a SiLU. Formára a ReLU-ra emlékeztethet, viszont nincsen benne „törés” az origónál, nem monoton növekvő illetve tud negatív értéket visszaadni. Elhárítja a „haldokló ReLU” problémát vagyis azt hogy túl sok neuron válik inaktívvá ez által megáll a tanulás. Matematikailag a SiLU az a szigmoid függvény és változó szorzata.[15] Az (5.) egyenlet megadja a SiLU aktivációs függvényt.

$$\text{SiLU}(x) = x * \frac{1}{1 + e^{(-x)}} \quad (20)$$



5. ábra: SiLU függvény

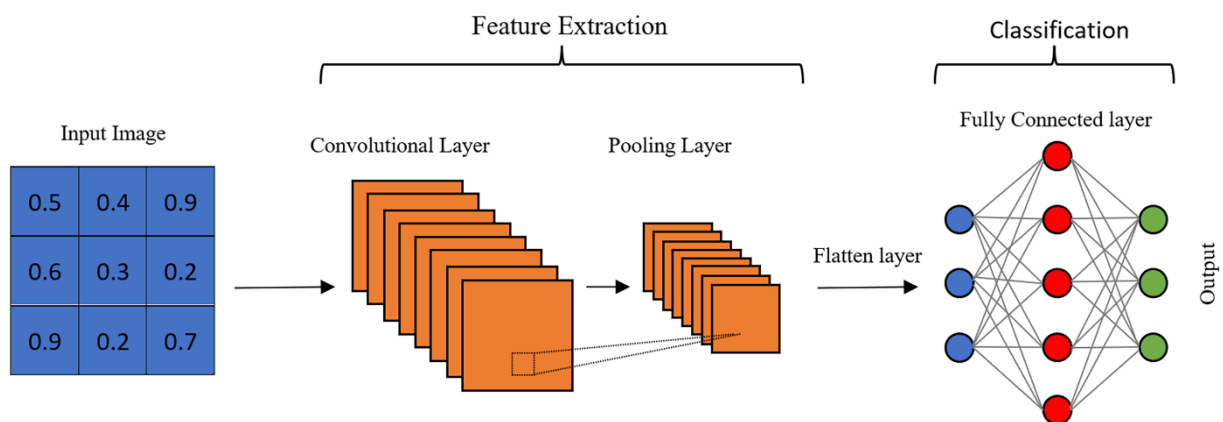
4. Convolutional Neural Networks (CNNs) és YOLO11

4.1. A convolutional neural network előnyei

A mesterséges neurális hálózatokkal (ANNs) szemben a CNN-ket elsősorban a képeken belüli minták felismerésére használják, ezért az ilyen modelleket kifejezetten kép specifikus feladatokra fejlesztik, optimalizálják. Az egyik legnagyobb limitáló faktor a hagyományos ANN-ek használata esetén hogy a számítási komplexitásuk nem elegendő a képek feldolgozásához, ha egy 64x64-es színes képet veszünk alapul már akkor is az első réteg egyik neuronjához 12,288 súly tartozna, valamint a neurális hálózatnak túlságosan nagyoknak kéne lennie. [16]

4.2. Általános szerkezet

Ahhoz hogy CNN-t kapjunk három réteg kell, konvolúciós réteg, pooling réteg és fully-connected réteg, ezekből általában több is van egy modellben. A 6. ábra egy egyszerűsített CNN szerkezetet ábrázol. [16]



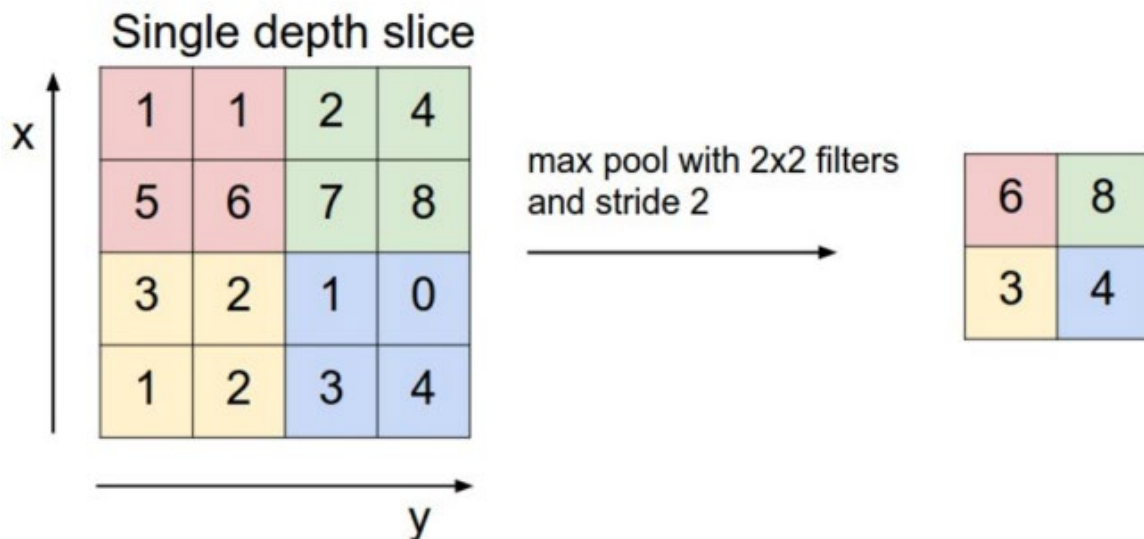
6. ábra: CNN struktúra [17]

A bemeneti vagy input layer a kép pixeleinek az értékeit tartalmazza, az egész modellnek ez az egyetlen bemenete van.

A konvolúciós rétegben egy kernel „végigmegy” a bemeneti kép értékein és kiszámítja a szorzatot a kernel súlya és az adott kép rész között. A kernel értékei a konvolúciós neurális hálózatnak a súlyai ezek változnak a tanulási folyamat során a backpropagation és gradiens ereszkedés szerint. Ennek a rétegnek az outputját feature map-nek nevezzük, ez egy mártix

aminek az egyes elemeit betápláljuk egy aktivációs függvénybe (YOLO11 és YOLOv8 esetén SiLU) ez által nem-linearitást vezetünk be.

A pooling layer méret csökkentést végez, vagyis csökkenti a paraméterek számát ezzel növeli a gyorsaságot és a hatékonyságot. A legnépszerűbb pooling metódusok a max pooling, average pooling, min pooling és global pooling (a YOLO11 Spatial Pyramid Pooling-ot használ). [18]



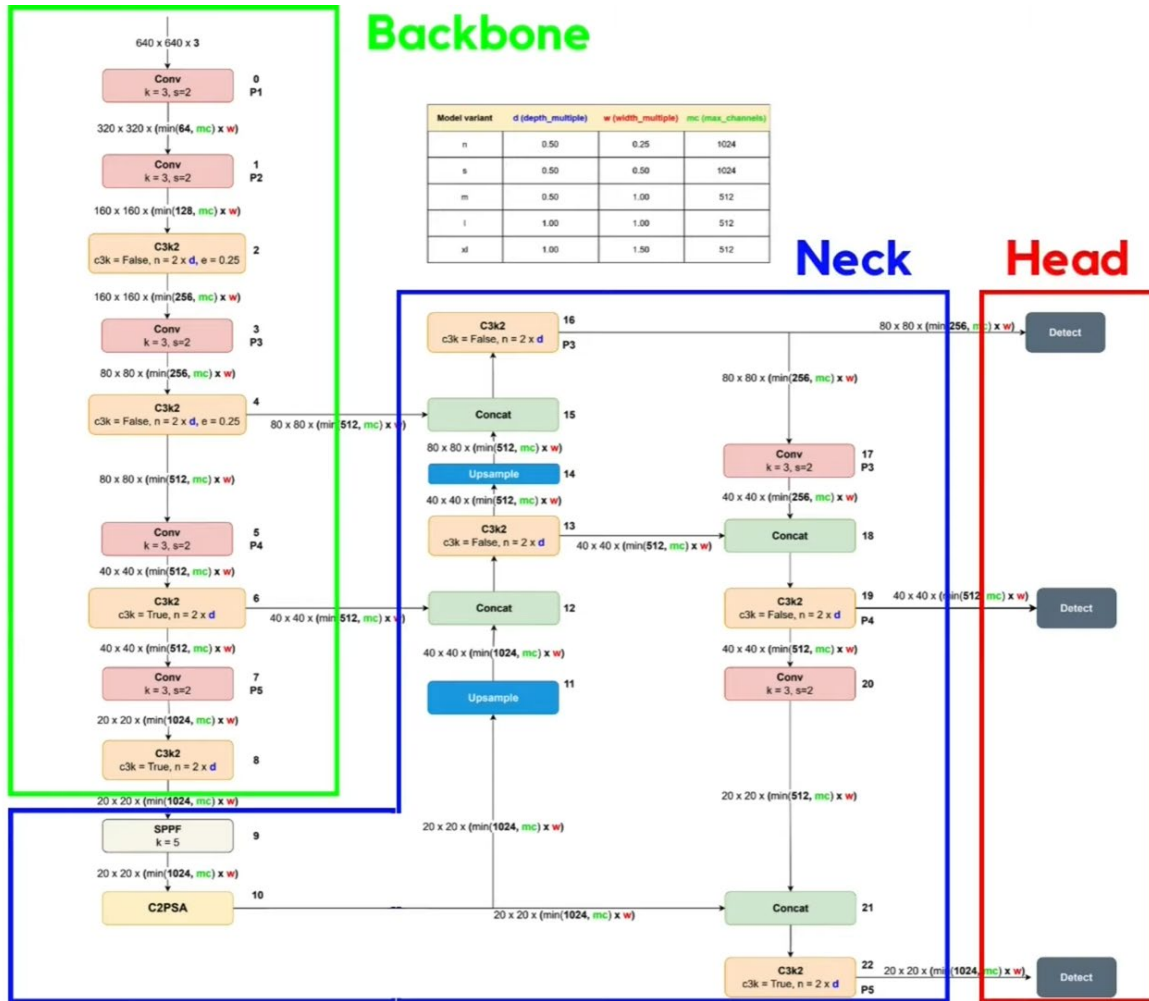
7. árba: Max pooling (az adott 2x2-es rész legnagyobb értékét veszi) [18]

A fully connected layer előtt történik egy flattening művelet ami egy, egydimenziós vektort hoz létre amit aztán a fully connected layer fel tud használni a kategorizáláshoz, becsléshez. Ehhez kategorizáláshoz egy normál mesterséges neurális hálót (ANN) használunk. Az output-ok pedig a neuronok értékei lesznek ami alapján megkapjuk, hogy milyen objektumot detektált a rendszer. [18]

4.3. YOLO11 szerkezete

A YOLO architektúra 3 fő komponensből áll, a backbone-ból (gerinc), a neck-ből (nyak) és a head-ből (fej). A backbone végzi a feature extraction (jelleg kinyerés) legnagyobb részét konvolúciós blokkok segítségével. A neck az egy köztes feldolgozó rész amely speciális blokkok segítségével a különböző mértékű, nagyságú feature map-ek feldolgozza, felerősíti. A head a végső becslést végzi el, ami során az objektum felismeréshez szükséges adatokat kapjuk meg a végső feature map-ek alapján. Egy hagyományos CNN-el szemben a YOLO11 nem tartalmaz fully connected layer-t, ezért nem szükséges adott méretű bemenetet használnunk,

bármilyen bemenet ami méretben (pixel magasság, szélesség) 32-nek a többszöröse megfelel a modellnek. [19]



8. ábra: YOLO11 struktúrája [21]

4.3.1. Backbone

A backbone 9 blokkjából, 5 normál konvolúciós blokk, 4 pedig C3k2 blokk. A normál konvolúcióshoz blokkhoz képest a C3k2 blokk abban különbözik hogy a konvolúciót két lépésben hajtja végre kisebb kernel méret használatával. [20]

4.3.2. Neck

Itt történik a pooling az SPPF (Spatial Pyramid Pooling – Fast) blokk használatával. Spatial piramid pooling segítségével három (vagy több) arány szerint tudjuk elvégezni a pooling folyamatot. A C2PSA felerősíti a feature map-ek régiók szerinti „figyelmét”, így a modell jobban tud koncentrálni a kép fontosabb területeire, ezzel javítva a modell pontosságát a különböző méretű és pozíciójú objektumok esetén. Az upsample blokk növeli a bemenő feature

map felbontását hogy a concatenate blokk segítségével össze lehessen fűzni egy korábbi feature map-el. [20] [22]

4.3.3. Head

Az objektum detektálásért felelős, a neck-ből kapott feature map-ek alapján kapjuk meg a kimenetet. A head, detect blokkjaiban található, C3k2 konvolúciós blokk ami tovább finomítja a feature map-eket, CBS blokk ami batch normalization-t végez valamint betáplálja az értékeket a SiLU aktivációs függvénybe, Conv2D blokk ami csökkenti a feature-ök számát annyira hogy a bounding box kordináták meghatározásához és az osztály meghatározásához fel lehessen használni, valamint a végső detect blokk feldolgozza a bounding box koordinátákat, az objektum értékeket amik az objektum jelenlétét jelzik és az osztály értékeket amik az objektum osztályát határozzák meg. [20]

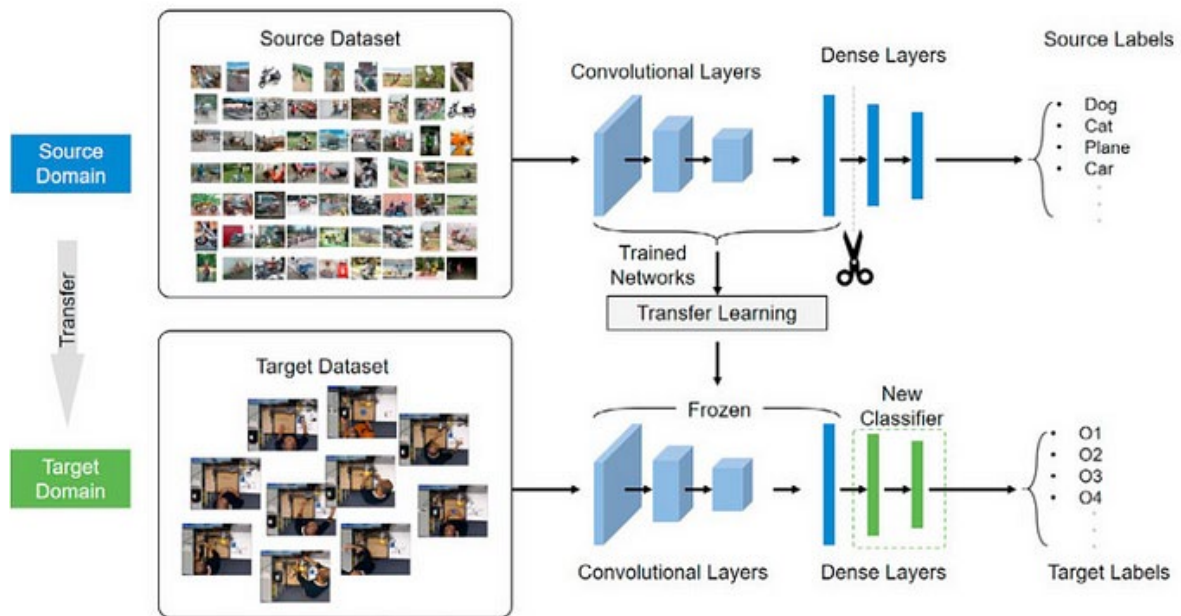
4.4. Transfer learning

Transfer learning-el egy előre betanított modellt fel tudunk használni, a betanítotthoz hasonló feladat végzésére. A pre-trained (előre betanított) modell már elsajátított tudását – objekt detektálás esetén például hogy hogyan kell körvonalakat detektálni – használjuk fel, így ha másmilyen objektum felismerésére akarjuk betanítani nem kell hatalmas adatbázist használnunk, hogy újra megtanuljon körvonalakat felismerni mivel azt már tudja. A transfer learning előnyei, a tanítási vagy számítási költség és az adatmennyiség csökkenése valamint a jobb általánosítás. Egy nagy adatbázison való tanításhoz rengeteg idő és nagy számítási teljesítmény kell, ezekre nincsen szükség transfer learning használata során mivel viszonylag kevés adattal is jó eredményeket lehet elérni. A modell általánosítása az által javul hogy a súlyokat (amik nincsen befagyasztva) szimplán tovább tanítjuk, vagyis végeredményben a modell az eredetihez képest több adat alapján tanul. [23]

4.4.1. Transfer learning használata

Az első lépés egy előre betanított modell kiválasztása ami hasonló feladatra lett kifejlesztve mint amire majd használni szeretnénk.

A második lépésben konfigurálnunk kell az előre betanított modellt. A YOLO11 esetében a 0-tó a 9-es számú blokkig – vagyis a backbone-t plusz az SPPF blokkot - be kell fagyasztani (freeze) a súlyokat hogy azok ne változzanak, ezzel megtartva a modell által már megszerzett „tudást”. Új osztályok létrehozása és vagy régi osztályok törlése esetén módosítani kell a becslést végző blokkokon. [24]



9. ábra: Transfer learning [25]

Harmadik lépésben pedig betanítjuk a modellt a saját adatainkkal. Nem megfelelő eredmények esetén a hiperparaméterek változtatásával javíthatunk a modell pontosságán. Hiperparaméter például a tanításra validációra és tesztelésre szánt képek aránya, a választott aktivációs függvény, a gradiens ereszkedéshez használt algoritmus, a batch méret, a tanítási epoch-ok száma. [24][26]

5. Nevesebb nemzetközi folyóiratok feldolgozása

A Deep Learning Based Forest Fire Detection Approach Using UAV and YOLOv3, cikkben a pilóta nélküli repülő eszközök(UAVs) elterjedésére hivatkozva, egy ilyen eszköz által szolgáltatott videófelvételen valós időben detektálták az erdőtüzeket. A YOLOv3 segítségével létrehozott, kisebb számítási teljesítményt igénylő algoritmust hoztak létre amit aztán az UAV-ben található hardver-re telepítettek. Az algoritmust a készített videófelvételeken tanították be, az esetek 83%-ban helyesen ismeri fel a tűz jelenlétét, illetve kb. 0,3125[s] a felismerés gyorsasága. [27]

A Deep Learning Based Object Identification System for Forest Fire Detection cikk az erdőtüzek során keletkezett füstoszlopok osztályozásával foglalkozik. Három osztályba sorolták a füstoszlopokat a horizonthoz viszonyított helyzetük alapján. Két féle algoritmus is betanítottak a RetinaNet-et és a Faster R-CNN-t. Hasonló eredményeket kaptak mindkettővel, a 90%-ban felismerték a füst jelenlétét a teszt képeken. Ezek után a modelleket tesztelték 24 füst keletkezési folyamaton amiket a HPWREN adatbázison értek el, átlagosan a tűz kialakulása után 6,3 perccel detektálta a füstoszlopot az algoritmus. [28]

Deep Convolutional Neural Networks for Forest Fire Detection cikkben a tűzfelismerés úgy történik hogy a mesterséges neurális hálózat először megállapítja hogy van e tűz a képen, ha van akkor egy „fine grained patch fire classifier” pontosan meghatározza a helyét. Ezzel a módszerrel a training képeken 97%-os a teszt képeken pedig 90%-os pontosságot tudtak elérni. Ez a modell elsődlegesen az erdő tüzek felismerésére lett készítve. [29]

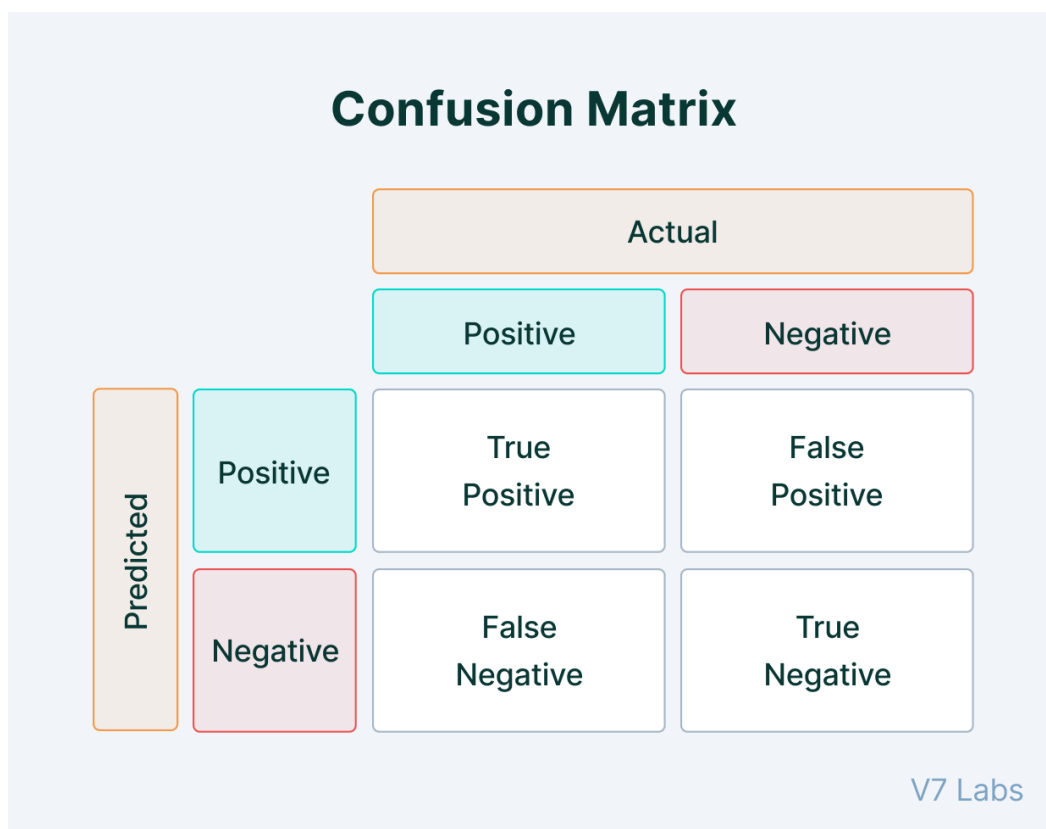
6. Tűz és füst felismerés gépi tanulással

6.1. Absztrakt

A munkám célja az volt hogy egy tűz és füstfelismerő algoritmust tanítsak be ami, videón és képeken fel tudja ismerni az említett objektumokat. A modell pontosságát több kép gyűjtésével (716 helyett 8243), data augmentaton-el továbbá az alap modell egy újabb változatának a betanításával (YOLOv8, YOLO11) próbáltam javítani. A legjobb változat pontossága tűz esetén X füst esetén pedig Y az mAP50 pedig Z. A betanított modell a számítógépen futva egy webkamerán keresztül, illetve közvetlenül teszt képen képes volt felismerni a tüzet illetve a füstöt.

6.2. A modell teljesítmény mutatói

A modell pontosságának jellemzésére szolgál az mAP50, AP(fire), AP(smoke), precision, recall és az mAP50-90. A detektálásokat 4 csoportba sorolhatjuk true positive, False positive, false negative, true negative. Azt hogy egy adott detektálás melyikbe tartozik azt a 6. ábra alapján könnyen el lehet dönteni. [30]



10. ábra: Confusion matrix [30]

True positive(TP): A modell egy bizonyos tűréshatáron belül helyesen detektálja az objektumot.

True negative(TN): A modell nem detektál semmit és valójában sincs semmi

False positive(FP): A modell detektált viszont nem kellett volna neki

False negative(FN): A modell nem detektált viszont kellett volna neki

A precision-t vagyis a pontosságot az alábbi módon számoljuk ki:

$$Precision = \frac{(TP)}{(TP) + (FP)} \quad (21)$$

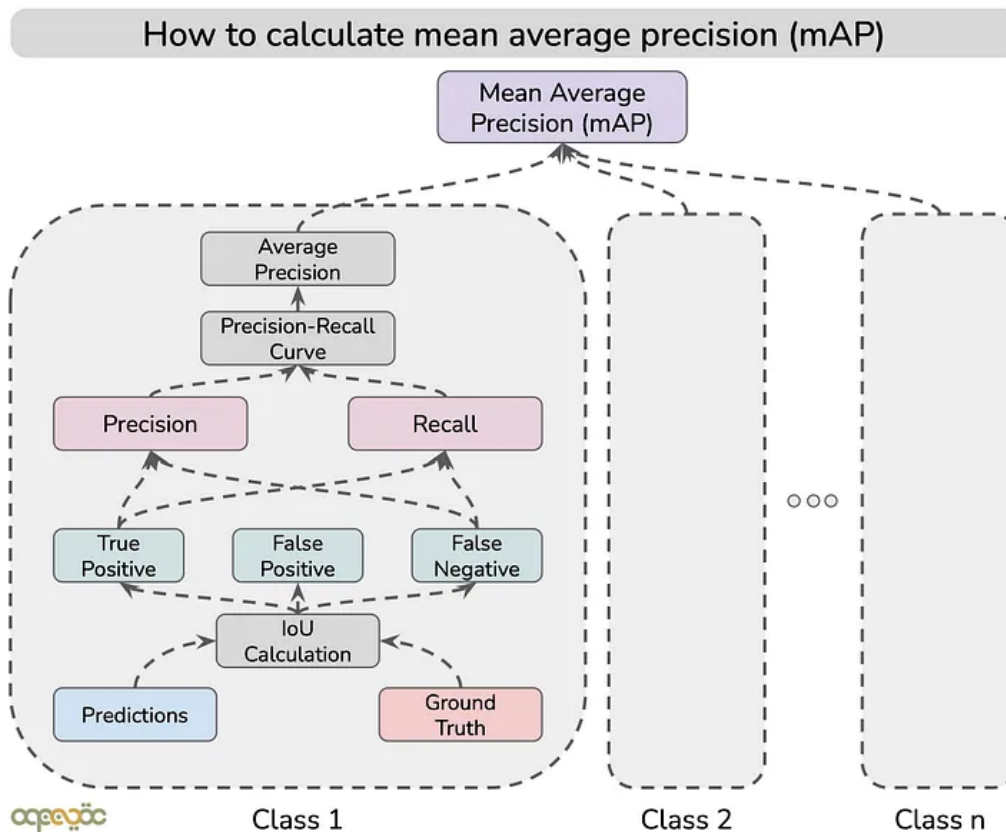
A recall-t vagyis a „visszaemlékezést” pedig így számoljuk ki:

$$Recall = \frac{(TP)}{(TP) + (FN)} \quad (22)$$

Ha függvényen ábrázoljuk a precision, recall értékeket, ahol a precision az y, a recall pedig az x tengelyen van akkor megkapjuk a precision-recall görbét, amivel meghatározható a precision és a recall közötti arány. A precision recall görbe alatti területe megadja az AP-t (average precision) amit ha minden osztályra kiszámolunk (fire, smoke) és aztán átlagoljuk akkor megkapjuk az mAP-t (mean average precision). Az Intersection over Union megadja becsült és a tényleges, objektum körüli keret(bounding box) mekkora mértékben fedi egymást.[30]

$$IoU = \frac{\text{becsült és tényleges keret metszetének területe}}{\text{becsült és tényleges keret úniójának területe}} \quad (23)$$

Az mAP50 megadja a modell mAP értékét abban az esetben amikor az IoU nagyobb vagy egyenlő 0.50-nél.[31]



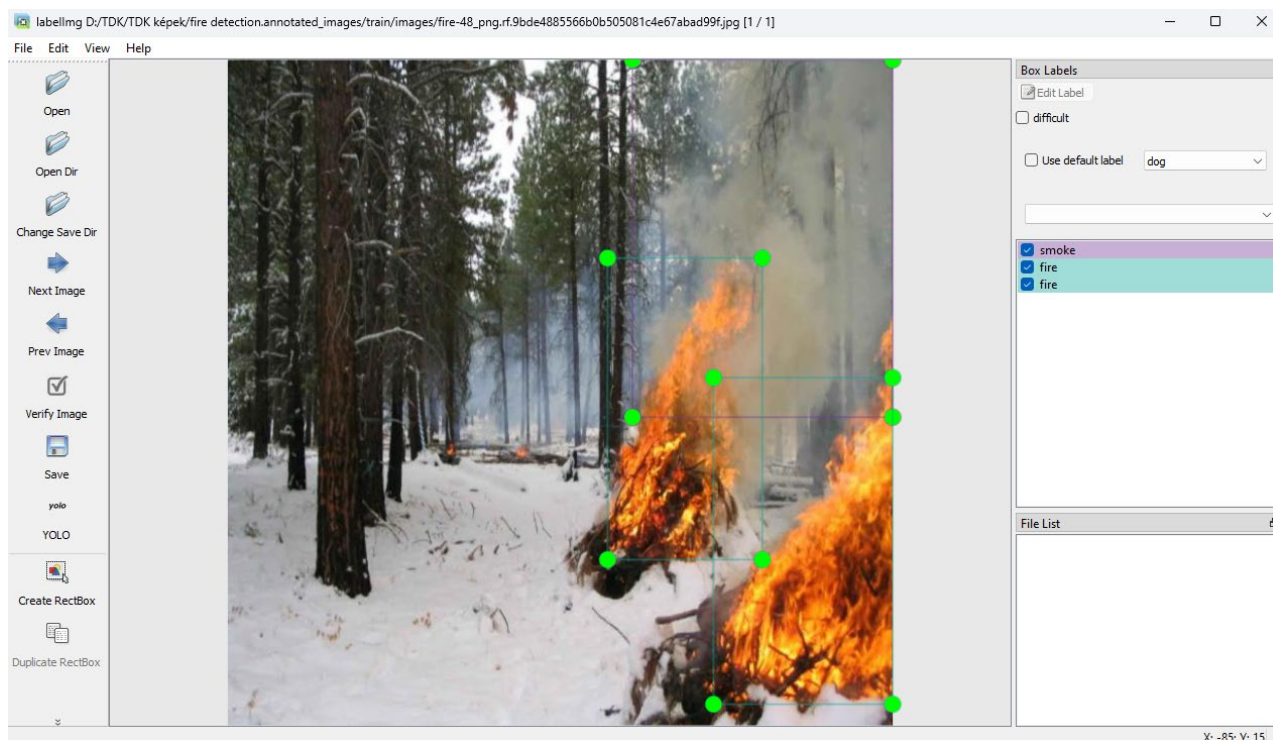
11. ábra: mAP számítása [32]

6.3. YOLOv8m és YOLO11m betanítása és eredményei

A YOLO (You Only Look Once) az Ultralytics-nek az előre betanított objektum felismerő modell sorozata, 2016-ban indult a YOLOv2-vel. A YOLOv8 2023 januárjában, a YOLO11 pedig 2024 szeptemberében jelent meg, mindkettő a Microsoft COCO (Common Objects in Context) adatbázison lett betanítva. Transzfer learning segítségével ezeket az előre betanított modelleket be lehet tanítani más objektumok felismerésére, viszonylag kevés példán (képen) keresztül. A YOLOv8 modellek közül a közepeset (YOLOv8m) használtam mivel pontosságban és gyorsaságban is megfelelőnek találtam. YOLOv8m adatai: képméret 640x640 pixel, RGB színtér (color space), detection time a COCO dataset-en tesztelve: 234,7 ms, mAP50-95 a COCO dataset-en tesztelve: 0.502. [33]

Az első verzióban az összes kép száma 716 volt ezeket a képeket az internetről különböző helyekről letöltöttem és a tűz eseteket label-el („címkével”) elláttam. [34] Ezeken a képeken összesen 3049 „fire” és 984 „smoke” címkével ellátott objektum szerepel, változó körülmények

között. A képek erdő tüzeket, városi és lakás tüzeket, kigyulladt autókat illetve sok másféle tüzesetet ábrázolnak.

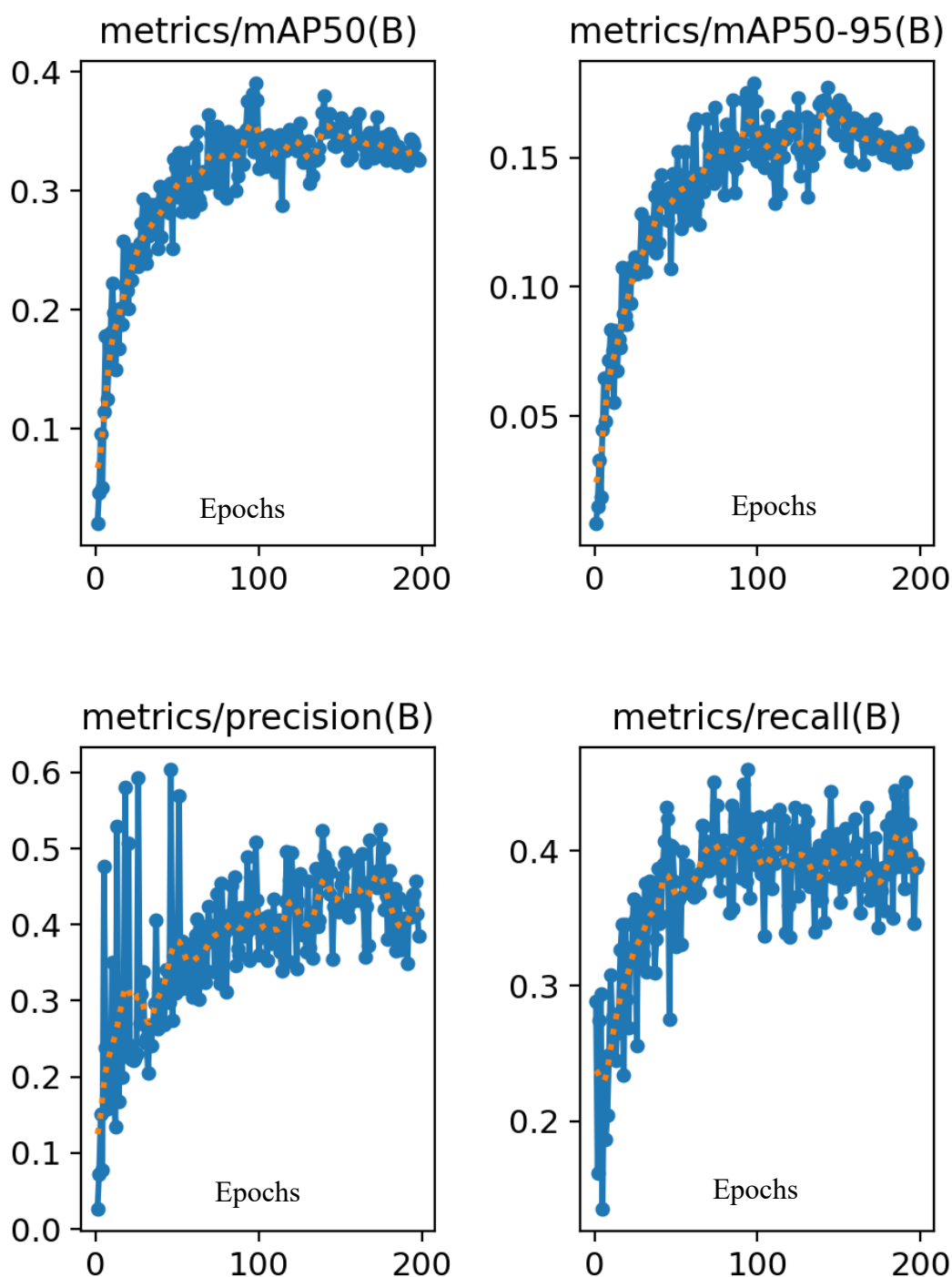


12. ábra: Címkézett kép „fire” és „smoke” objektumokkal

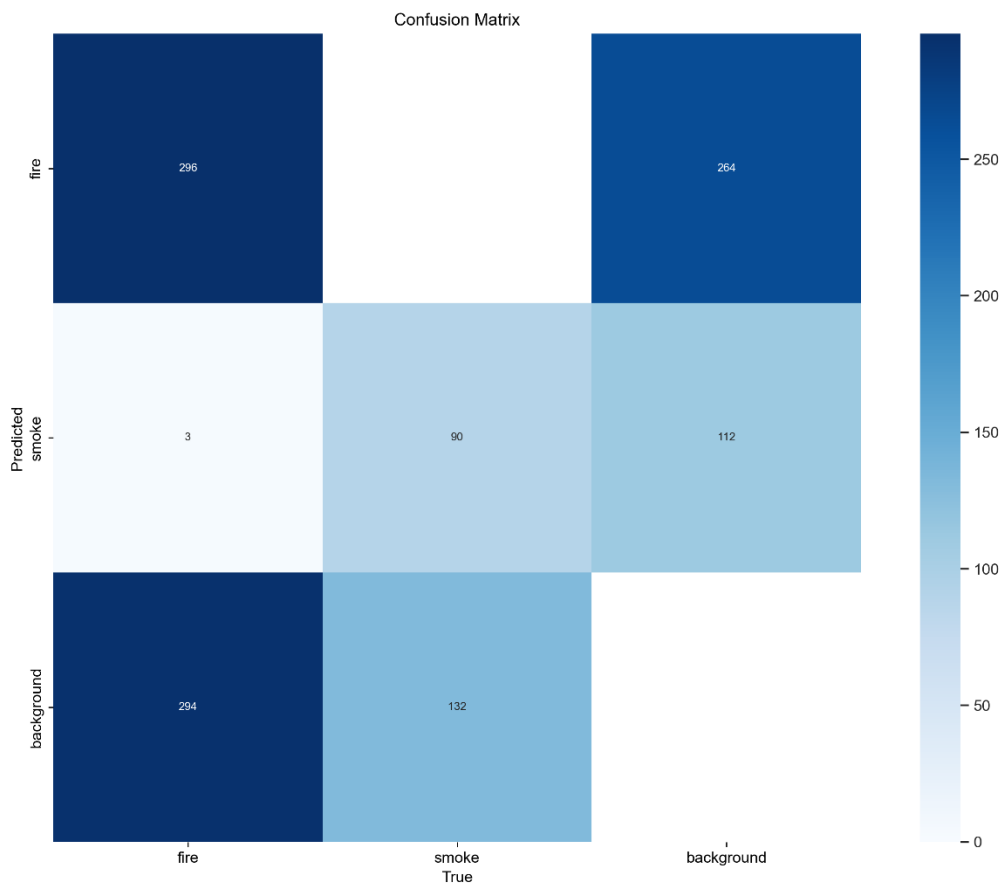
A megfelelő könyvtárak és programok telepítése után a képeket három felé rendeztem: „train”, „valid”(validation) és „test” csoportokba. A rendszer train mappába lévő képek alapján tanulja meg felismerni az objektumokat, az alakjuk (körvonaluk) alapján. A validation mappa szintén képeket tartalmaz amik eltérnek a train mappába lévőktől, a validation tanulás közben történik minden „ciklus”(epoch) után, és ennek segítségével ellenőrzi magát a modell, hogy a folyamat jó irányba halad-e, illetve kis változtatásokat végez (fine-tune) a paramétereiken. A „test” mappa magától értetődően a kész (betanított) modell tesztelésére van, a teszt képek alapján kapjuk meg a modell teljesítmény mutatóit. [35] A képek elosztása a három mappába számszerűen: „train” 500 darab (~69,83%), „valid” 152 darab (~21,23%), „test” 64 darab (~8,9%). A betanításhoz használt számítógép adatai: Windows 10, 8gb RAM, Intel core i5-4440 processzor, GTX 1050Ti videokártya. A betanítás során a batch size 2 volt. A modell teljesítménye a legjobb, 98. epoch-nál:

1. táblázat: yolov8m_custom 716 kép augmentáció nélkül

yolov8m_custom 716 kép augmentáció nélkül					
mAP50:	AP(fire):	AP(smoke):	Precision:	Recall:	mAP(50-90):
0.39	0.415	0.365	0.508	0.3916	0.1787



13. ábra: yolov8m_custom 716 kép augmentáció nélkül mAP50, mAP50-95, precision, recall



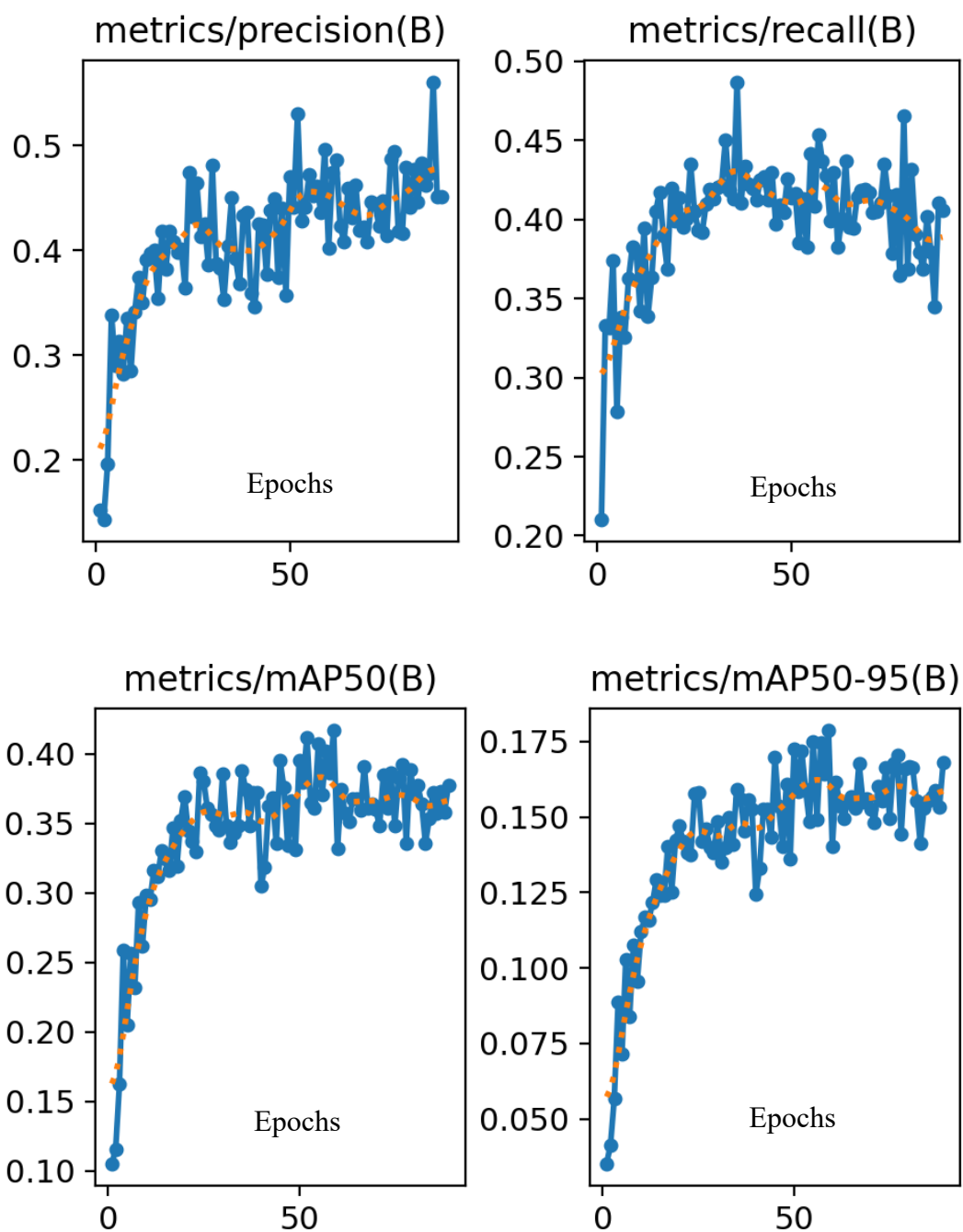
14. ábra: yolov8m_custom 716 kép augmentáció nélkül, confusion matrix

6.3.1. Data augmentáció használata

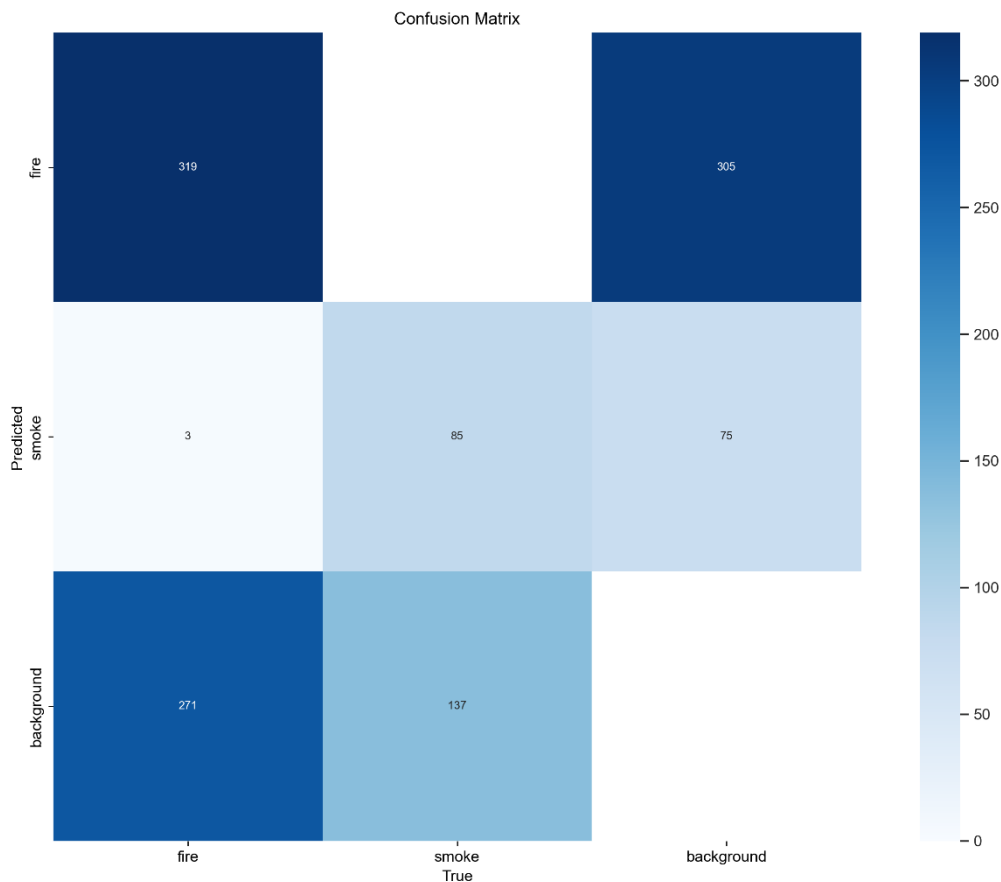
Az eredmények javítása érdekében data augmentációt alkalmaztam, ami segíthet a modellnek a jobb eredmények elérésében, mivel megnövekszik a tanításhoz használt képek száma, valamint egyes augmentációs módszerek segíthetnek az objektum körvonalának a háttértől való elkülönülésében. Három féle augmentációt használtam: horizontális tükrözést, elforgatást -15° és $+15^\circ$ között, és exposure változtatást -20% és $+20\%$ között. Az augmentációt a Roboflow nevű oldal augmentációs eszközével végeztem. [36] A tanításra használt képek száma ez által háromszorosára nőtt tehát 1500-ra, a validation és teszt képek száma az eredeti maradt, 152 és 64 darab. Data augmentációval a modell teljesítménye a legjobb, 59. epoch-nál:

2. táblázat: yolov8m_custom 716 kép három féle augmentációval

yolov8m_custom 716 kép három féle augmentációval					
mAP50:	AP(fire):	AP(smoke):	Precision:	Recall:	mAP(50-90):
0.417	0.439	0.395	0.561	0.4868	0.1911



15. ábra: yolov8m_custom 716 kép három féle augmentációval mAP50, mAP50-95, precision, recall



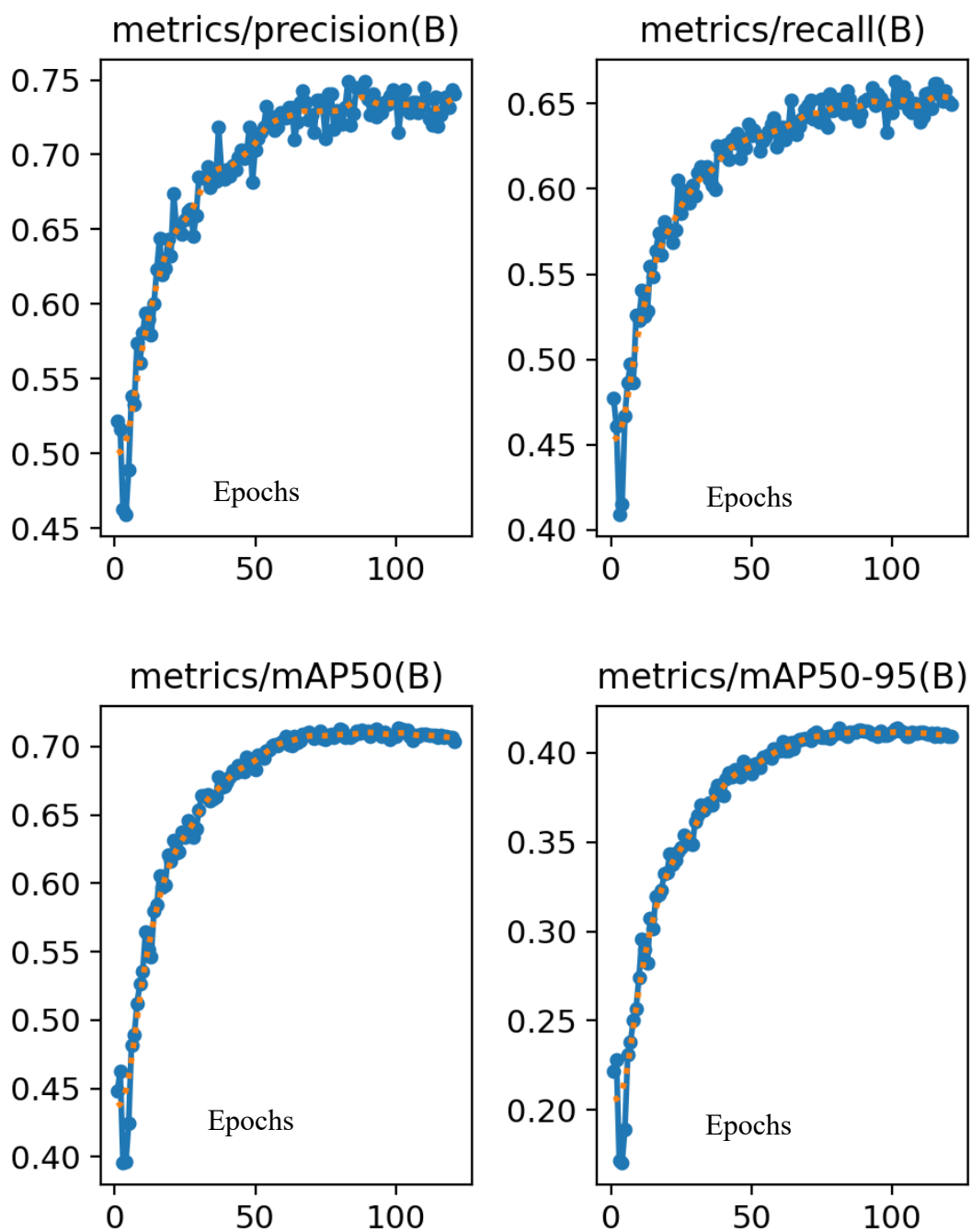
16. ábra: yolov8m_custom 716 kép három féle augmentációval, confusion matrix

6.3.2. A képek számának növelése

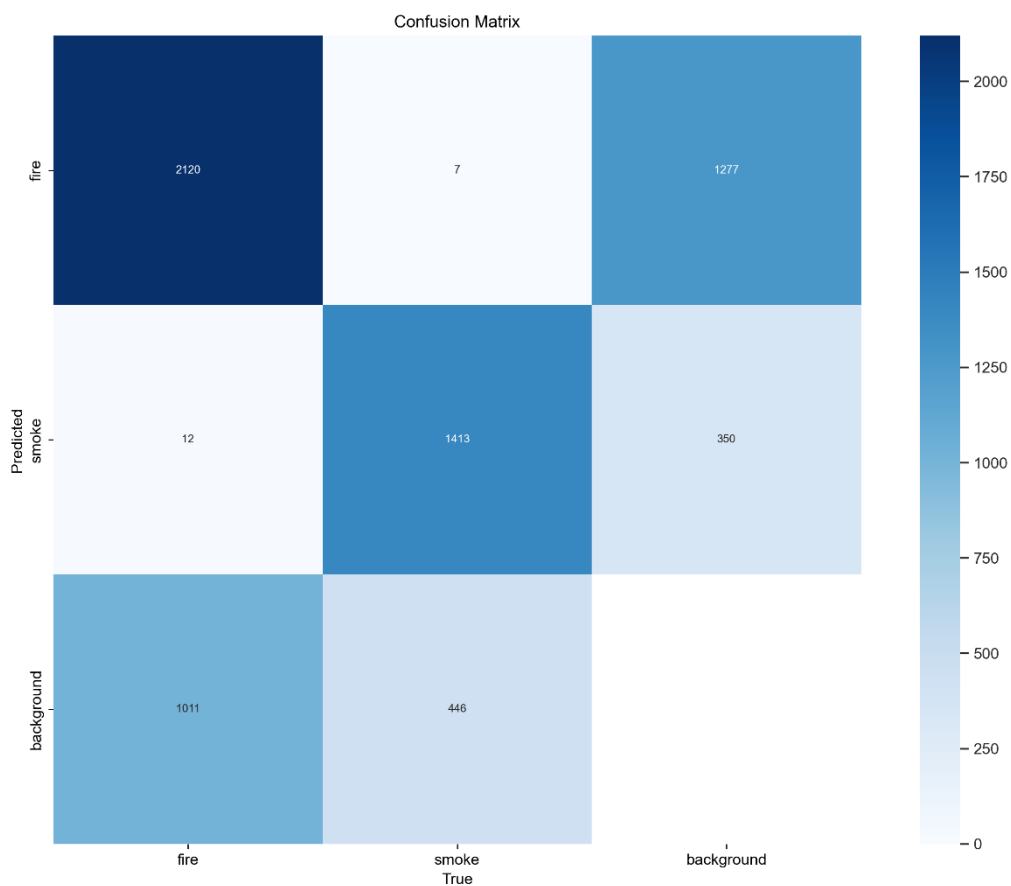
Mivel ezek az eredmények data augmentáció, után is rossznak mondhatóak ezért a jobb teljesítmény elérése érdekében jóval nagyobb mennyiségű képet használtam fel a következő betanításnál. A betanításra szánt képek a régi képekből és egy ingyenesen elérhető dataset képeinek jelentős részéből állt, így a képek száma összesen 8243.[37] A képek új elosztása így nézett ki: „train” 5770 darab(~70%), „valid” 1652 darab(~20%), 821 darab(~10%). „Fire” objektumból 15554 darab „smoke” objektumból pedig 9050 darab van az egész datasetben. A betanítás ugyan azzal a számítógéppel, ugyan akkora batch-el(2) és ugyan úgy a YOLOv8m-es modellel történt. A legjobb teljesítményt a 80. epoch-nál produkálta a modell

3. táblázat: yolov8m_custom 8243 kép augmentáció nélkül

yolov8m_custom 8243 kép augmentáció nélkül					
mAP50:	AP(fire):	AP(smoke):	Precision:	Recall:	mAP(50-90):
0.712	0.638	0.785	0.721	0.652	0.414



17. ábra: yolov8m_custom 8243 kép augmentáció nélkül mAP50, mAP50-95, precision, recall



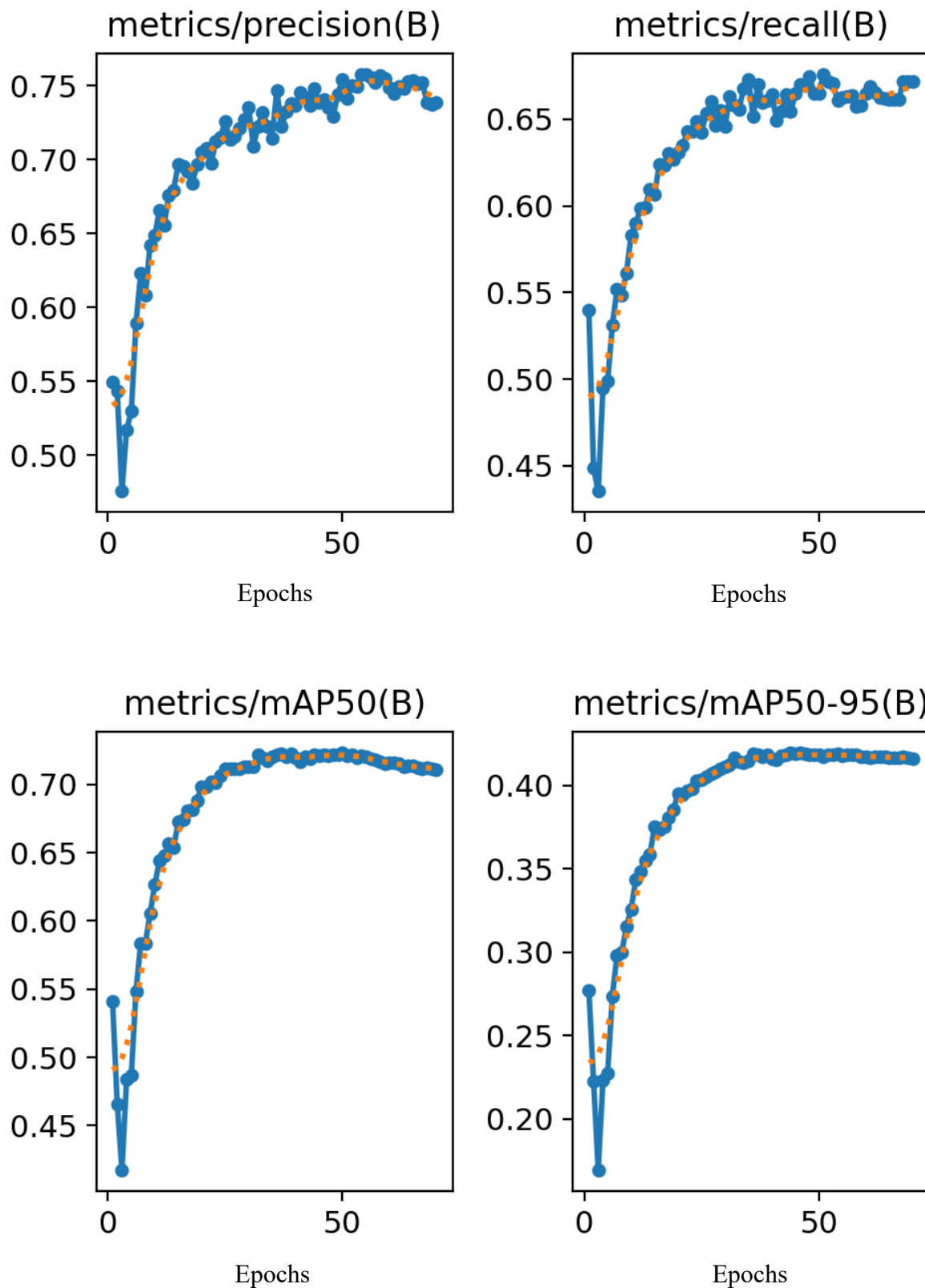
18. ábra: yolov8m_custom 8243 kép augmentáció nélkül confusion matrix

6.3.3. Növelt képszám augmentációval

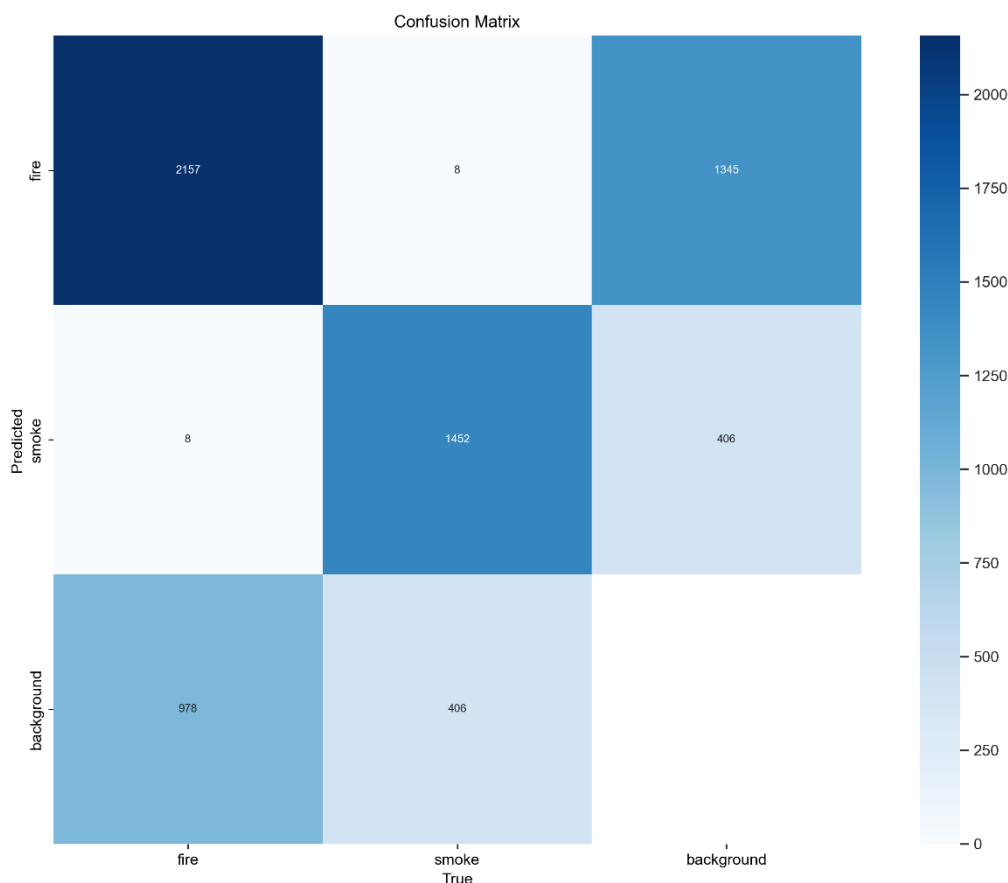
Az kisebb képmennyiségnél említett augmentációs módszereket(horizontális tükrözést, elforgatást -15° és $+15^\circ$ között, és exposure változtatást -20% és $+20\%$ között) használva próbáltam a nagyobb képszámnál tovább javítani a modell teljesítményét. Ebben az esetben is a „train” képek száma növekedett, 5770-ről 17310-re. Így a 43. epoch-nál voltak a legjobbak az eredmények:

4. táblázat: yolov8m_custom 8243 kép három féle augmentációval

yolov8m_custom 8243 kép három féle augmentációval					
mAP50:	AP(fire):	AP(smoke):	Precision:	Recall:	mAP(50-90):
0.72	0.646	0.794	0.72	0.659	0.417



19. ábra: yolov8m_custom 8243 kép három féle augmentációval mAP50, mAP50-95, precision, recall



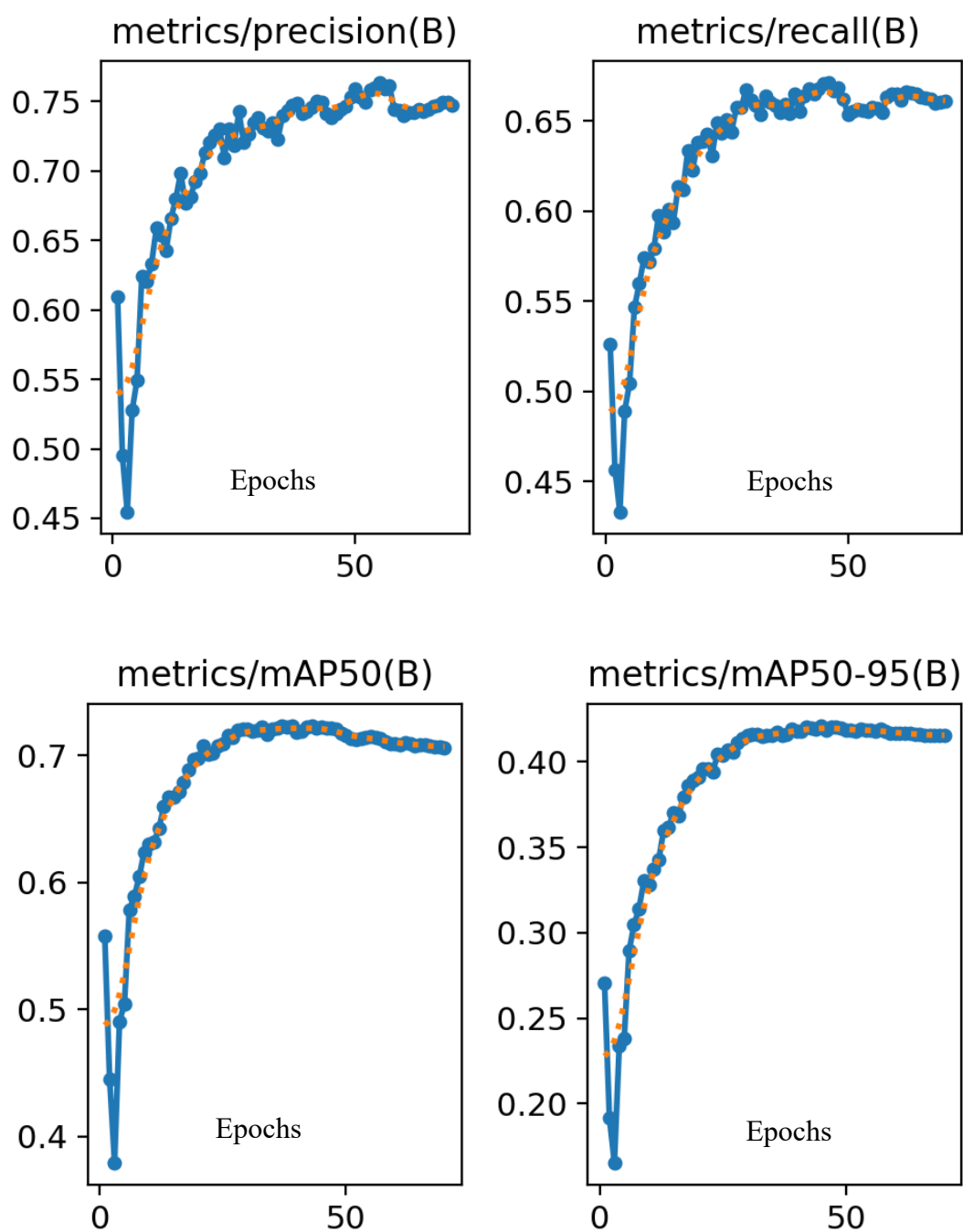
20. ábra: yolov8m_custom 8243 kép három féle augmentációval confusion matrix

6.3.4. YOLO11m augmentációval

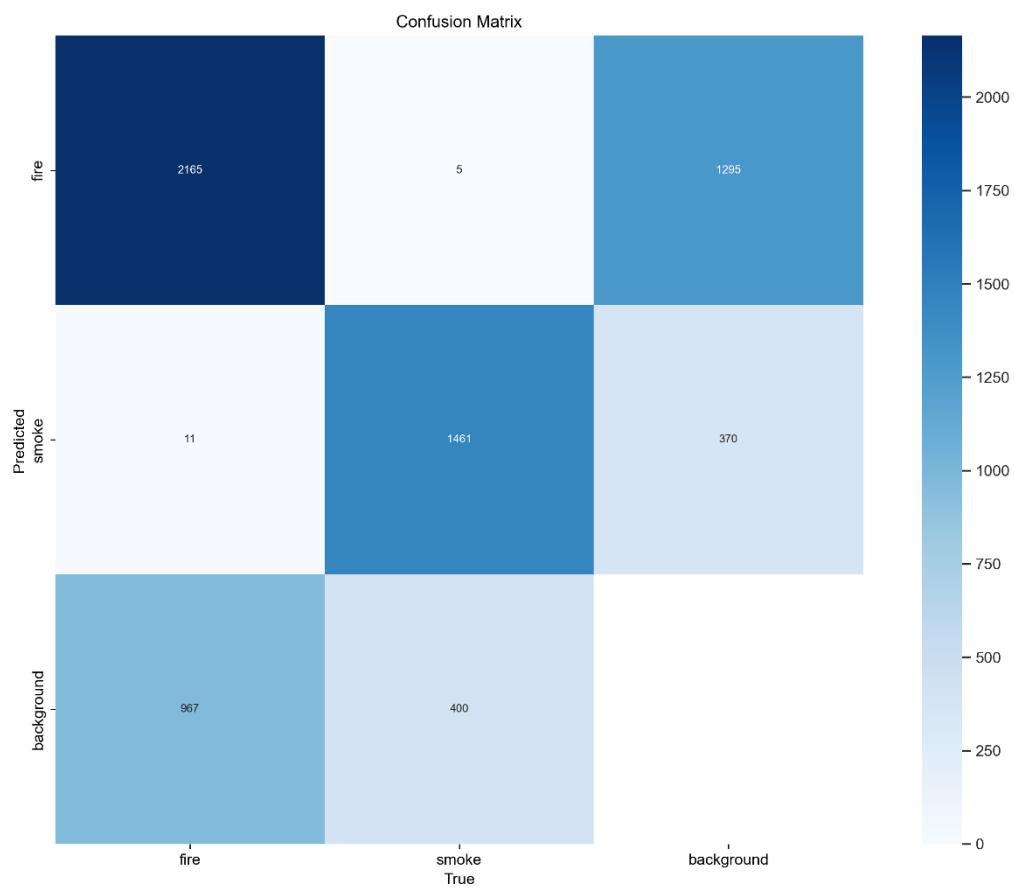
A YOLO11 az eddigi legjobb a YOLO modellek közül, az „m” vagyis közepes változatot választottam most is. A YOLO11m adatai: képméret 640x640 pixel, RGB színtér(color space), detection time a COCO dataset-en tesztelve: 183.2 ms, mAP50-95 a COCO dataset-en tesztelve: 0.515. [38] Ebben az esetben egyből augmentált adatokkal tanítottam be, a használt augmentációk: horizontális tükrözést, elforgatást -15° és $+15^\circ$ között, és exposure változtatást -20% és $+20\%$ között. az előző YOLOv8m-es esethez hasonlóan a „train” képek száma 17310. A betanításra használt számítógép adatai: Windows 11, 32gb RAM, Intel core i5-12400F processzor, GTX 1080Ti videokártya. A betanítás során a batch size 24 volt. Az teljesítmény a 43. epoch-nál volt a legjobb:

5. táblázat: yolo11_custom 8243 kép három féle augmentációval

yolo11_custom 8243 kép három féle augmentációval					
mAP50:	AP(fire):	AP(smoke):	Precision:	Recall:	mAP(50-90):
0.723	0.646	0.799	0.7414	0.665	0.421



21. ábra: yolo11_custom 8243 kép három féle augmentációval mAP50, mAP50-95, precision, recall



22. ábra: yolo11_custom 8243 kép három fële augmentációval confusion matrix

A yolo11_custom modell tesztelése képeken:



23. ábra: A „test” mappa képein detektálás

A tesztképeken átlagosan 106,72ms alatt ismerte fel a „fire” és „smoke” objektumokat. Ahogy a képeken is látszik a detektálások nem tökéletesek, a füst egy részét nem detektálja, vagy éppen túl nagyoknak veszi, néhány kisebb tűzfoltot nem detektál.

Az eredmények összefoglalva:

Az eredményekből azt láthatjuk hogy a legnagyobb javulást a teljesítményben a képek számának növelése hozta, mAP50: 0.39-ről 0.712-re. Data augmentáció segítségével a kisebb

képszámmal rendelkező modell mAP50-e 0.027-et javult míg a nagyobb képszámmal rendelkező modell esetében csak 0.008-at. 0.003-al javult a teljesítmény az újabb modell használatával. A legfontosabb eredmények táblázatba foglalva:

7. táblázat: A legjobb eredmények

	mAP50	Precision:	Recall:
yolov8m_custom 716 kép augmentáció nélkül	0.39	0.508	0.3916
yolov8m_custom 716 kép három féle augmentációval	0.417	0.561	0.4868
yolov8m_custom 8243 kép augmentáció nélkül	0.712	0.721	0.652
yolov8m_custom 8243 kép három féle augmentációval	0.72	0.72	0.659
yolo11_custom 8243 kép három féle augmentációval	0.723	0.7414	0.665

7. Konklúzió

Sikeresen betanítottam egy objektum felismerő modellt, tűz és füst felismerésére, a legjobb eredményeket YOLO11 produkálta, 8243 kép használata mellett, amiből a „train” és „valid” képeket háromféle módon augmentáltam (horizontális tükrözést, elforgatást -15° és $+15^\circ$ között, és exposure változtatást -20% és $+20\%$ között). A kiindulási mAP50-et 33,3%-al, a pontosságot 23,34%-al, a recall-t pedig 27,34%-al sikerült javítani a használt technikák (képek számának növelése, augmentáció, másik modell használata) segítségével. A betanításra használt képek változatossága (esti, városi, erdei, közeli, távoli képek) miatt a modell jól általánosít ezért a teszt képen jó eredményeket tud elérni, valamint a valóságban széles körben használható.

- [1] Sara Novak, „The Dawn of Fire: When Did Early Humans First Discover Fire?” [Online] Elérhető: <https://www.discovermagazine.com/planet-earth/when-did-early-hominins-first-discover-fire> [Hozzáférés dátuma: 2024.09.28]
- [2] Vass Attila, „A nagy debreceni tűzvész 220 éve égette fel a cívisvárost” [Online] Elérhető: <https://www.haon.hu/helyi-kozelet/2022/06/a-nagy-debreceeni-tuzvesz-220-eve-egette-fel-a-civisvarost> [Hozzáférés dátuma: 2024.09.28]
- [4] Protect & Detect, „The History of Fire Alarms and Smoke Detectors” [Online] Elérhető: <https://vagiesfiai.hu/hu/szolgaltatasok/tuzjelzo-rendszer/> [Hozzáférés dátuma: 2024.09.29]
- [5] Vági és fiai kft., Tűzvédelmi kézi jelzésadó falon kívüli tűzriasztó [Online] Elérhető: <https://protectanddetect.co.uk/blog/history-fire-alarms-smoke-detectors/> [Hozzáférés dátuma: 2024.09.29]
- [3] Miroslav Bistrović, Danko Kezić, Domagoj Komorčec, „Historical Development of Fire Detection System Technology on Ships” [Online] Elérhető: <https://www.proquest.com/docview/1490550612?sourcetype=Scholarly%20Journals> [Hozzáférés dátuma: 2024.09.29]
- [6] Zhong-Qiu Zhao, Member, IEEE, Peng Zheng, Shou-tao Xu, and Xindong Wu, „Object Detection with Deep Learning: A Review” [Online] Elérhető: <https://ieeexplore.ieee.org/abstract/document/8627998> [Hozzáférés dátuma: 2024.10.19]
- [7], IBM, „What is a neural network?” [Online] Elérhető: <https://www.ibm.com/topics/neural-networks> [Hozzáférés dátuma: 2024.10.19]
- [8] IBM, „What is gradient descent?” [Online] Elérhető: <https://www.ibm.com/topics/gradient-descent> [Hozzáférés dátuma: 2024.09.17]
- [9] Gradiens ereszkedés ábra [Online] Elérhető: <https://medium.com/analytics-vidhya/journey-of-gradient-descent-from-local-to-global-c851eba3d367> [Hozzáférés dátuma: 2024.09.17]
- [10] Grant Sanderson, „What is backpropagation really doing?” [Online] Elérhető: <https://www.3blue1brown.com/lessons/backpropagation> [Hozzáférés dátuma: 2024.10.19]
- [11] Michael Nilsen, „How the backpropagation algorithm works” [Online] Elérhető: <http://neuralnetworksanddeeplearning.com/chap2.html> [Hozzáférés dátuma: 2024.09.17]

- [12] Pragati Baheti, „Activation Functions in Neural Networks [12 Types & Use Cases]” [Online] Elérhető: <https://www.v7labs.com/blog/neural-networks-activation-functions#h1> [Hozzáférés dátuma: 2024.09.25]
- [13] Neri Van Otten, „Activation Function: Top 9 Most Popular Explained & When To Use Them” [Online] Elérhető: <https://spotintelligence.com/2023/06/16/activation-function/> [Hozzáférés dátuma: 2024.09.25]
- [14] Siddharth Sharma, Simone Sharma, Anidhya Athaiya, „ACTIVATION FUNCTIONS IN NEURAL NETWORKS” [Online] Elérhető: <https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf> [Hozzáférés dátuma: 2024.09.25]
- [15] Abhishek Kumar Pandey, „SiLU (Sigmoid Linear Unit) activation function” [Online] Elérhető: <https://medium.com/@akp83540/silu-sigmoid-linear-unit-activation-function-d9b6845f0c81> [Hozzáférés dátuma: 2024.09.25]
- [16] O'Shea, K. "An introduction to convolutional neural networks." *arXiv preprint arXiv:1511.08458* (2015).
- [17] Alsaleh A, Perkgoz C. 2023. „A space and time efficient convolutional neural network for age group estimation from facial images.” *PeerJ Computer Science* 9:e1395 <https://doi.org/10.7717/peerj-cs.1395>
- [18] geeksforgeeks, „Introduction to Convolution Neural Network” [Online] Elérhető: <https://www.geeksforgeeks.org/introduction-convolution-neural-network/> [Hozzáférés dátuma: 2024.10.25]
- [19] Khanam, Rahima, and Muhammad Hussain. "YOLOv11: An overview of the key architectural enhancements." *arXiv preprint arXiv:2410.17725* (2024).
- [21] Dr. Priyanto Hidayatullah, „YOLO11 Architecture - Detailed Explanation” [Online] Elérhető: <https://www.youtube.com/watch?v=L9Va7Y9UT8E> [Hozzáférés dátuma: 2024.10.25]
- [22] He, Kaiming, et al. "Spatial pyramid pooling in deep convolutional networks for visual recognition." *IEEE transactions on pattern analysis and machine intelligence* 37.9 (2015): 1904-1916.
- [23] IBM, „What is transfer learning?” [Online] Elérhető: <https://www.ibm.com/topics/transfer-learning> [Hozzáférés dátuma: 2024.10.26]

- [24] aws, „What is Transfer Learning?” [Online] Elérhető: <https://aws.amazon.com/what-is/transfer-learning/> [Hozzáférés dátuma: 2024.10.26]
- [25] Saurabh Kumar, „Unlocking the Power of Transfer Learning and Data Augmentation in Deep Learning” [Online] Elérhető: <https://www.tredence.com/blog/transfer-learning-and-data-augmentation-in-deep-learning> [Hozzáférés dátuma: 2024.10.26]
- [26] Kizito Nyuytiyimbui, „Parameters and Hyperparameters in Machine Learning and Deep Learning” [Online] Elérhető: <https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac> [Hozzáférés dátuma: 2024.10.26]
- [27] Z. Jiao *et al.*, "A Deep Learning Based Forest Fire Detection Approach Using UAV and YOLOv3," *2019 1st International Conference on Industrial Artificial Intelligence (IAI)*, Shenyang, China, 2019, pp. 1-5, doi: 10.1109/ICIAI.2019.8850815.
- [28] Guede-Fernández, F.; Martins, L.; de Almeida, R.V.; Gamboa, H.; Vieira, P. A Deep Learning Based Object Identification System for Forest Fire Detection. *Fire* **2021**, *4*, 75. <https://doi.org/10.3390/fire4040075>
- [29] Qingjie Zhang, Jiaolong Xu, Liang Xu, Haifeng Guo, „Deep Convolutional Neural Networks for Forest Fire Detection”, *Proceedings of the 2016 International Forum on Management, Education and Information Technology Application*, 2016 Január
- [30] Deval Shah, „Mean Average Precision (mAP) Explained: Everything You Need to Know” [Online] Elérhető: <https://www.v7labs.com/blog/mean-average-precision> [Hozzáférés dátuma: 2024.10.16]
- [31] Ultralytics „Performance Metrics Deep Dive” [Online] Elérhető: <https://docs.ultralytics.com/guides/yolo-performance-metrics/> [Hozzáférés dátuma: 2024.10.16]
- [32] Aqeel Anwar, „What is Average Precision in Object Detection & Localization Algorithms and how to calculate it?” [Online] Elérhető: <https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b> [Hozzáférés dátuma: 2024.10.16]
- [33] Glenn Jocher, „yolo-V8” [Online] Elérhető: <https://github.com/autogyro/yolo-V8> [Hozzáférés dátuma: 2024.10.17]

- [34], Lauren Sell, LabelImg [Online] Elérhető: <https://www.atlantispress.com/proceedings/ifmeita-16/25850411> [Hozzáférés dátuma: 2024.10.15]
- [35] Kili, „Training, Validation and Test Sets: How To Split Machine Learning Data” [Online] Elérhető: <https://kili-technology.com/training-data/training-validation-and-test-sets-how-to-split-machine-learning-data> [Hozzáférés dátuma: 2024.10.16]
- [36] Roboflow, [Online] Elérhető: <https://roboflow.com/> [Hozzáférés dátuma: 2024.10.17]
- [37], siyuanwu, „DFS-FIRE-SMOKE-Dataset” [Online] Elérhető: <https://github.com/siyuanwu/DFS-FIRE-SMOKE-Dataset?tab=readme-ov-file> [Hozzáférés dátuma: 2024.10.17]
- [38] Glenn Jocher, „Ultralytics YOLO11” [Online] Elérhető: <https://github.com/ultralytics/ultralytics> [Hozzáférés dátuma: 2024.10.17]