



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
ELEKTRONIKAI ÉS KOMMUNIKÁCIÓS RENDSZEREK INTÉZET
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI TANSZÉK



SZAKDOLGOZAT



OE-KVK
2023.

Takács Kornél Zoltán
T008895/F112904/K



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Kandó Kálmán Villamosmérnöki Kar
Műszertechnikai és Automatizálási Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: Takács Kornél Zoltán

Szakedolgozat száma: SZD2309040923455065

Törzskönyvi száma: T008895/FI12904/K

Neptun kódja:

Szak: villamosmérnöki

Specializáció: Műszer-automatika specializáció

A dolgozat címe: Gépi látás és objektumfelismerés ESP segítségével

A dolgozat címe angolul: Machine vision and object recognition with ESP

A feladat részletezése: A gépi látás és objektum felismerés a számítógépek képessé tételére arra, hogy feldolgozzák és értelmezzék a képi információkat, ahogy azt az emberi látás teszi.

Mikrokontroller és kamera segítségével gépi objektum felismerés, adatfeldolgozás és kiértékelés megvalósítása.

Megvalósítás ESP mikrokontrollerrel és ezzel kompatibilis kamera alkalmazásával történik.

Intézményi konzulens neve: Borsos Döníz

A kiadott téma elévülési határideje: 2025. 12. 31.

Beadási határidő: 2023. 12. 15.

A szakedolgozat: Nem titkos.

Kiadva: Budapest, 2023. 10. 24.



Intézetigazgató

A dolgozatot beadásra alkalmasnak találom:

belső konzulens

külső konzulens



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
ELEKTRONIKAI ÉS KOMMUNIKÁCIÓS RENDSZEREK INTÉZETI
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI TANSZÉK



HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja, a titkosításra vonatkozó esetleges megkötések mellett.

Budapest, 2023. 12. 11.



[Handwritten signature]

.....
hallgató aláírása

TARTALOMJEGYZÉK

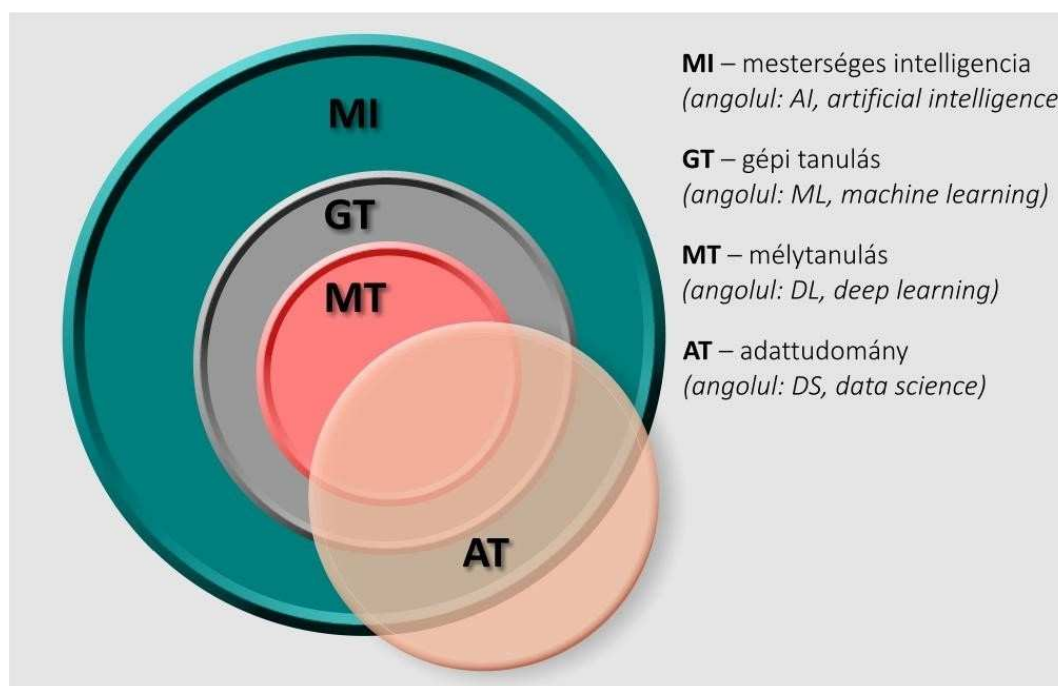
Tartalomjegyzék.....	1
1 Bevezetés.....	3
1.1 A dolgozat témája.....	3
1.2 Motiváció	4
2 A Neurális háló.....	6
2.1 Rövid történelem	6
2.2 Mi a neurális háló?	9
2.3 Neuronok.....	10
2.4 Aktivációs függvények.....	13
2.4.1 Lépcsős függvény.....	13
2.4.2 Signum függvény	14
2.4.3 A lineáris függvény	15
2.4.4 A szigmoid függvény	15
2.4.5 A ReLU függvény	16
2.4.6 Egyéb függvények.....	17
2.5 Veszteség, hiba.....	18
2.5.1 Átlagos abszolút hiba (MAE).....	18
2.5.2 Átlagos négyzetes hiba (MSE).....	19
2.5.3 Kategorizált keresztentrópia veszteség	20
2.6 Visszaterjesztés (backpropagation).....	21
2.7 Neurális háló tanítása	22
2.8 Adatkészletek	23
2.9 Túltanulás (overfitting)	24
2.10 Dropout	25
2.11 Optimalizáció.....	27
2.11.1 Sztochasztikus gradiens csökkentés (SGD) [26]	28

2.11.2	Sztochasztikus gradiens csökkentés momentummal [27]	28
2.11.3	AdaGrad [29]	29
2.11.4	Adam [30]	30
2.12	Regularizáció [35]	30
2.13	Neurális hálót támogató hardver	31
3	Gépi látás kategóriái.....	33
3.1	Mi a gépi látás?	33
3.2	Mélységbecslés.....	33
3.3	Képbesorolás	34
3.4	Objektumfelismerés	36
3.5	Képszegmentálás	37
3.6	Képből-képet	39
4	Algoritmusok.....	41
4.1	YOLO Algoritmus [41]	41
4.2	Colorful Image Colorization [16].....	44
5	Projektmunka	46
5.1	Projekt célja és motivációja.....	46
5.2	A neurális háló felépítése	47
5.3	Adatszerkezet	49
5.4	Hiperparaméterek	50
5.5	Tesztelés	51
5.6	Megvalósítás.....	53
5.7	Fejlesztési lehetőségek	56
6	Összefoglalás	57
7	Summary	58
8	Definíciók és rövidítések	59
9	Irodalomjegyzék	60

1 BEVEZETÉS

1.1 A dolgozat témája

Dolgozatom témája az objektumfelismerés és napjainkban a modern objektumfelismerési algoritmusok neurális hálózatokon alapulnak. A neurális hálók a gépi tanulás csoportjába tartoznak, és ez az egész témakör a mesterséges intelligencia alá sorolható.



1. ábra Mesterséges intelligencia területei és elhelyezkedésük¹

„A gépi látás olyan eljárások összessége, melynek a végén egy a vizuális információkra támaszkodó rendszer a történésekre képes reagálni. Az ezen eljárásokban alkalmazott megoldások manapság már sok tekintetben a biológiai rendszerekhez hasonló megközelítést alkalmaznak a feladatosztályok megoldására.”²

A gépi látás napjaink kurrens témája, gondoljunk csak a széles közönséget elérő telefonokban lévő képfeldolgozásra. A telefonok képesek kategorizálni fényképeinket a tartalmuk alapján. Ezen felül a vizuális élmény javítás érdekében a képek automatikus

¹ <https://tudas.hu/minden-amiorol-azt-hitted-tudod-a-mesterseges-intelligenciarol/> (2023.11.22.)

² „Tér-időbeli vizuális jelenségek vizsgálata biológiai alapú algoritmusokkal”, Fülöp Tamás, DOI:10.15774/PPKE.ITK.2020.002 (2023.11.22.)

feljavítását is már automatikusan végzik. Egy másik populáris téma az önvezető autók, itt szinte minden döntés képfeldolgozás alapján történik. Ezen felül gondoljunk olyan megoldásokra is, amik szűkebb csoportokat céloznak meg, mint amilyenek a vakokat segítő eszközök fejlesztései. Sokféle módon lehet segíteni a látássérült embereket ilyen algoritmusokkal, mint például a pénz felismerés [1] vagy akár a gyalogátkelőhelyek felismerése egy bionikus szemüveg segítségével. [2]

A képfeldolgozási algoritmusok a 2010-es évek óta robbanás szerűen terjednek, amióta olcsón és beágyazott rendszerekben is elérhetőek a tömbprocesszorok (GPU-k). Valamint a rendelkezésre álló hardver teljesítményét kihasználó fejlesztői könyvtárak (pl. Cuda[21], openCV[22]) melyek, lehetővé teszik, a neurális hálózhoz tartozó matematikai számítások gyors megoldását.

Ezeknek az algoritmusoknak képessége megtanulni a képekben rejlő összetett jellemzőket, és ezen a tudáson keresztül azonosítani az objektumokat még változatos környezetben és különböző nehezítő körülmények között is. A való világban az objektumok megjelenése számtalan változón múlik, mint például a fényviszonyok, perspektíva, elhelyezkedés vagy akár a tárgyak átfedése, ezért rendkívül komplex modellek fejlesztését forszírozza. Az objektumok ritkán jelennek meg ugyanabban a formában vagy ugyanabból a szögből, ami azt jelenti, hogy az algoritmusoknak képesnek kell lenniük a felismerésre különböző szögekből és körülmények között. Ez a változatosság nehezíti az algoritmusok tanítását és tesztelését is.

A nehezítő körülmények azt jelentik, hogy a képfelismerő algoritmusoknak nagyon rugalmasnak és adaptívnak kell lenniük. A fejlesztőknek folyamatosan tesztelniük kell őket a lehető legváltozatosabb adatkészleteken, hogy javítsák a pontosságot és a megbízhatóságot.

1.2 Motiváció

Számomra a számítógépek, és működésük mindig is lenyűgöztek. Eleinte nem értettem, és a működésük legfelső rétegeit kezdtem el megérteni, tanulmányozni, de mindig is érdekelt, hogy egy bitcsoportból, hogyan is lesz operációs rendszer. Hasonlóan a mesterséges intelligenciával is, lenyűgözött, hogy olyan dolgokat képesek megmondani, ami akár 10 évvel ezelőtt elképzelhetetlen lett volna.

Kíváncsi voltam, hogy hogyan működik, hogyan kell elképzelni a gyűjtőnévvel sokat emlegetett mesterséges intelligenciát. A témával kapcsolatban minél több mindent megértettem annál inkább jöttem rá, hogy mennyi mindent nem értek még belőle. A legelső felismerésem az volt, hogy milyen egyszerű az egész, végül is csak összeadás és szorzás, de mégis, ha nagyon sok egyszerű dolgot egymás mellé teszünk, mennyire bonyolult is lesz az.

2 A NEURÁLIS HÁLÓ

A választott témám, mint gépi látás teljes mértékben a neurális hálózatokra épül. Ez a témakör több szempontból is megközelíthető, mint biológiailag vagy akár csak egyszerűbb számítógépes algoritmusokkal. Mivel az alapköve a dolgozatom témájának a neurális háló, ezért ebben a fejezetben az elemi részeit, és a felépítéséhez szükséges elemeit, algoritmusait mutatom be, ezen kívül egy a neurális háló szempontjából opcionálisnak tekinthető témakört a regularizációt.

2.1 Rövid történelem

A számítógépek megjelenése óta a tudósok igyekeznek olyan algoritmusokat alkotni, amelyek képessé teszik a számítógépeket arra, hogy bizonyos bemenetekből mintát vegyenek és ez alapján előre meghatározott algoritmusok alapján formáljanak egy kimenetet, amely olyan feladatokat lát el, mint például az osztályozás vagy a regresszió.

A neurális hálózatok az 1940-es években születtek, az első ilyen publikáció 1949-ben történt Donald Hebb által, de nagyjából 20 éven át rejtély volt, hogy hogyan lehet ezeket feltanítani.[3] Az 1940-es évek végén Donald Hebb pszichológus felállított egy hipotézist, amely az idegi plaszticitás mechanizmusán alapul, ezt ma Hebb tanulásként ismerjük. A Hebb tanulás „tipikus” példája a felügyelet nélküli tanulásnak, későbbi változatai pedig a „hosszú távú potencírozás” (long term potentiation) korai modelljei voltak. Ezeket az ötleteket kezte el alkalmazni Alan Turing. [4]

„Turing egy kutatási jelentést írt *Intelligens Gépek (Intelligent Machinery)* címen (Turing, 1948), amely a következő mondattal kezdődik: „Javaslom megvizsgálni azt a kérdést, hogy vajon lehetséges-e az, hogy a gépek intelligens viselkedést mutassanak”, majd leírja a rekurrens neurális hálózatokat, amelyeket „B típusú nem szervezett gépek” néven vezet be, és megadja tanításuknak egy lehetséges megközelítését.”³

A visszaterjesztés (backpropagation) fogalma az 1960-as években jelent meg, de a neurális hálózatok még akkor sem kaptak különösebb figyelmet. Ennek egyik fontos oka, hogy akkoriban nem volt elég számítási kapacitás. Az ilyen jellegű problémákra eleinte megoldást adott az 1990-es években megjelent ACE4k[18], ACE16j[19] vagy Xenon[20]

³http://project.mit.bme.hu/mi_almanach/books/aima/ch20s08 (2023.11.05.)

chipek, viszont ezeknek teljesítménye még mindig csak kevés feladat megoldására voltak alkalmasak. Áttörést a 2010-es évek környékén értek el mikor megjelentek az Nvidia GPU chipjei, amelyekkel már megoldható volt, hogy a számításokat párhuzamosítsák, ezzel teljesítménynövekedést elérve.[23]

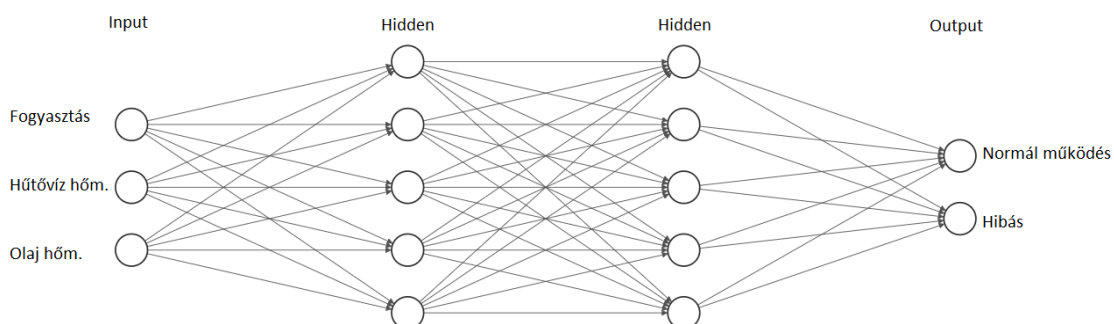
2010-től kezdődően az ImageNet szervezői évente megrendezték az ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [33] versenyt. Az ImageNet nagyméretű vizuális felismerési versenyt azért hozták létre, hogy elősegítsék a gépi látás, különösen a képfelismerés és a képi objektum-felismerési technológiák fejlődését. A célja egy olyan adatbázis létrehozása volt, amely segíthet a gépi látás fejlődésében. Az ImageNet verseny egy képfelismerési kihívás volt, ahol a résztvevő algoritmusoknak nagy számú képet kellett osztályozniuk és címkézniük. Az AlexNet[34] csapata azzal a céllal indult a versenyen, hogy bemutassa a mély tanulási modellek és különösen a konvolúciós neurális hálózatok potenciálját a képfelismerés terén. Az AlexNet sikere után a mély tanulás és a konvolúciós neurális hálózatok iránti érdeklődés jelentősen megnőtt, ezzel előre lendítve a fejlődést.[24] Az AlexNet áttörése gyorsította a mély tanulás kutatását és alkalmazását, valamint hozzájárult a modern gépi látás rendszerek fejlődéséhez.

Azóta a neurális hálózatok rohamosan fejlődésen estek át, a néha varázslatosnak tűnő megoldási képességük miatt, amelyek korábban megoldhatatlannak ítélt problémák voltak, például kép leírás generálás, nyelvi fordítás, hang- és videósintézis. [5]

Jelenleg a neurális hálózatok jelentik az elsődleges megoldást a legtöbb olyan kihívást jelentő technológiai problémára, mint az önvezető autók, a kockázatszámítás, a csalások felderítése és a rák korai felismerése, hogy csak néhányat említsek.

Általánosságban három féle gépi tanulás (Machine Learning) létezik, ezek a felügyelt, a felügyelet nélküli és a megerősített tanulási módszerek. A felügyelt gépi tanulást akkor használják, ha a meglévő adathalmaz előre felcímkeztető (kiértékelhető), és meghatározható értékekkel rendelkezik, amelyeket lehet a feltanításhoz használni. Például tegyük fel, hogy egy autó adatait naplózzuk bizonyos időközönként. Az autó üzemanyag fogyasztását, hűtővíz hőmérsékletét, olajhőmérsékletét figyeljük, és mentjük le percenként. Megtesszük ezt akkor is, amikor az autó megfelelően működik és akkor is, amikor hibásan. Az adatokat kiértékeljük, abban az esetben amikor nem volt probléma az autóval azokat az értékeket besoroljuk a „normál működés” kategóriába, illetve azokat az értékeket amikor valami hibás volt az autón, azon értékeket a „hibás” kategóriába. Ezen felül, hogy képesek

legyünk megjósolni a meghibásodást előre, azokat az értékeket is „hibás” -nak kell besorolni, amelyeket a meghibásodás előtt rögzítettünk.



2. ábra Autódiagnosztika példa, neurális háló⁴

A „hibás” és „normál működés” értékeket osztályoknak vagy címkéknek nevezzük. Ezeket az értékeket nevezhetjük célpontnak is, vagy alapigazságnak is, miközben a gépi tanulási algoritmust alkalmazzuk. Ezen értékeket használjuk fel ahhoz, hogy az algoritmust feltanítsuk, mert ezekről az értékekről tudjuk, hogy ezek biztosan helyesek, így alkalmasak egy cél érték meghatározásához. Ebben a példában a cél az, hogy az algoritmus úgy legyen feltanítva, hogy az érzékelő adatait beolvasva pontos előrejelzést tudjon adni, amikor a hiba már a küszöbön áll. De ez csak egy példa a felügyelt gépi tanulásnak, és azon belül is az osztályozás formájának. Ezen felül nem csak az osztályozásra, hanem regresszióra is lehet alkalmazni egy neurális hálót, például a pénzügyi világban részvény értékeinek jövőbeli becslésére, csak, hogy még egy példát említsek.

Van felügyelet nélküli gépi tanulás is, ahol az algoritmus úgy ismeri fel a címkéket, vagy osztályokat, hogy azokat nem látta idő előtt. Ezen kívül létezik még visszacsatolt vagy megerősített tanulás (reinforcement learning), illetve félig felügyelt (semi-supervised machine learning) és felügyelt is.

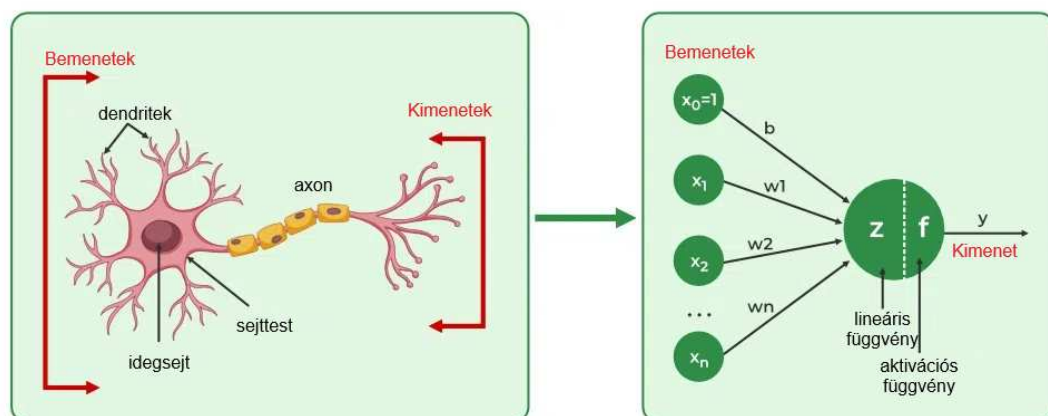
⁴ <https://alexlenail.me/NN-SVG/index.html> alkalmazással generálva (2023.11.05.)

2.2 Mi a neurális háló?

A „mesterséges” neurális hálózatok olyan számítási modellek, amelyek forradalmasították a gépi tanulás területét.

„A neurális hálózat nem más, mint egymással összekötött neuronok összessége. A neuron (idegsejt) egy információ feldolgozó egység, a neurális hálózat alapegysége.”⁵

Ezek a neuronok közel sem tökéletes megközelítése az emberi agyban lévő neuronok működésének, de azokhoz hasonlóan itt is, mint az emberi agyban vannak aktiválások és sok kapcsolódási pont. Ezek a hasonlóságok egy emberi idegsejthez képest, még akkor is, ha a mögöttes folyamatok egészen mások.



3. ábra Biológiai idegsejt(neuron) összehasonlítása mesterséges neuronnal.⁶

Egyetlen neuron önmagában viszonylag haszontalan, de több száz vagy több ezer (vagy sokkal több) más neuronnal kombinálva olyan kapcsolatokat és eredményeket hoz létre, amelyek gyakran felülmúlnak minden más gépi algoritmust.

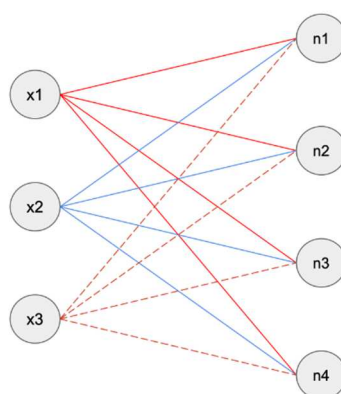
⁵ Fazekas István, „Neurális hálózatok” Debrecen, 2013 (2023.11.05.)

⁶ Forrás: <https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/> (2023.11.05.)

A neurális hálózatokat „fekete dobozoknak” tekinthetjük, mivel gyakran fogalmunk sincs, hogy miért is jutnak el arra a következtetésekre, mint amilyenekre, de megértjük, hogy mindezt hogyan teszik meg.

Egy neurális háló alapvetően három részre csoportosítható, egy bemeneti rétegre, amelyek a vizsgált értékeket tartalmazzák, rejtett rétegekre, amelyek a függvényt, algoritmust valósítják meg és egy kimeneti rétegre, ami az algoritmus kimenetét mutatja.

A sűrű rétegek (dense layers), a leggyakoribb réteg fajta, ezek a rétegek egymással összefüggő neuronokból állnak. Egy sűrű rétegben egy adott réteg minden neuronja a következő réteg minden neuronjához kapcsolódik, ami azt jelenti, hogy a kimeneti értéke a következő neuronok bemenetévé válik. Egy ilyen rétegkapcsolódást szemléltet a 4. ábra.

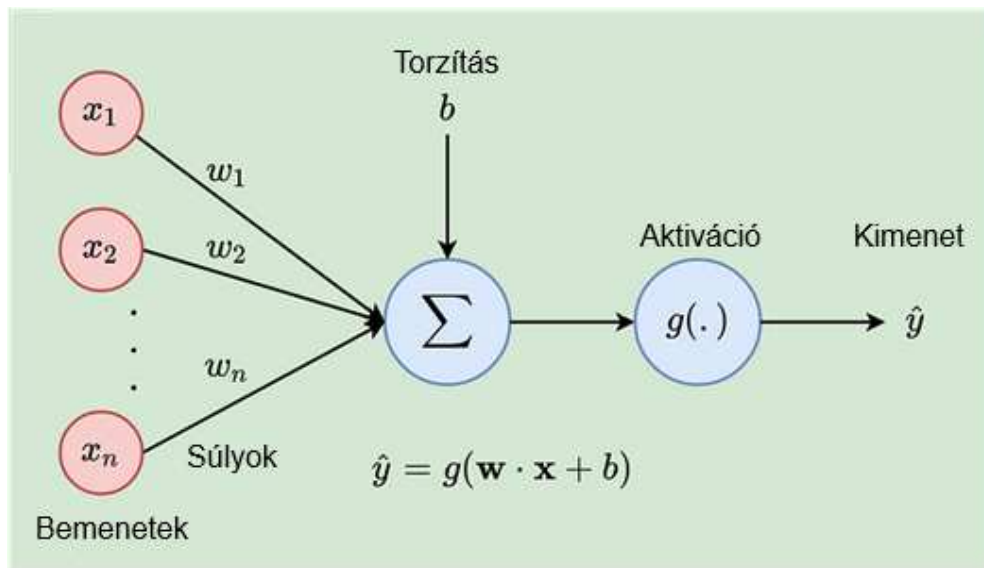


4. ábra Teljesen összekapcsolt neurális hálózat (dense layer)⁷

2.3 Neuronok

A neuronok közötti minden kapcsolathoz tartozik egy súly (weight) érték, amely egy tanítható tényezője annak, hogy ebből a bemenetből mennyit kell felhasználni, és ezt a súly értéket szorozzuk meg a neuron bemeneti értékével. Miután a neuron összes bemeneti értékét összeszorozzuk a hozzá kapcsolódó súllyal, összegezzük és hozzáadunk egy torzítást (bias), ez egy másik tanítható paramétert. A torzítás célja, hogy pozitívan vagy negatívan ellensúlyozza, eltolja a kimenetet, ami tovább segítheti az adatok megtanulását. Ezt a számítást szemlélteti a 5. ábra.

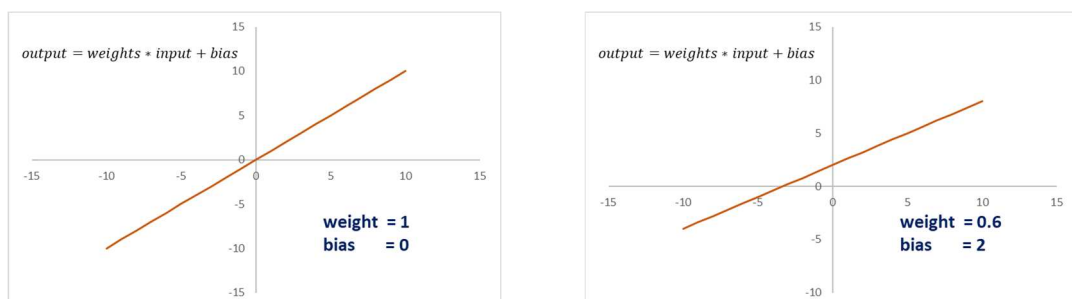
⁷<https://wandb.ai/ayush-thakur/keras-dense/reports/Keras-Dense-Layer-How-to-Use-It-Correctly--Vmlldzo0MjAzNDY1> (2023.11.05.)



5. ábra Egy neuron számítási példa⁸

A súlyok (weights) és torzítások (biases) fogalmát „potmétereknek” tekinthetjük, amelyeket úgy hangolhatunk, hogy a modell, neuron az adatokhoz illeszkedjen. Egy neurális hálózatban gyakran több ezer vagy akár millió ilyen paraméter is lehet. A torzítások és a súlyok egyaránt hangolható paraméterek, és mindkettő hatással van a neuronok kimenetére, de mind ezt különböző módon teszik. Mivel a súlyokat szorozzuk, csak a nagyságot változtatják meg, vagy akár teljesen átfordítják az előjelet pozitívról negatívra, vagy fordítva. A neuron számításának módja így a *Kimenet* = *súly* · *bemenet* + *torzítás*, ez meg egyezik az $y = mx + b$ lineáris egyenlettel.

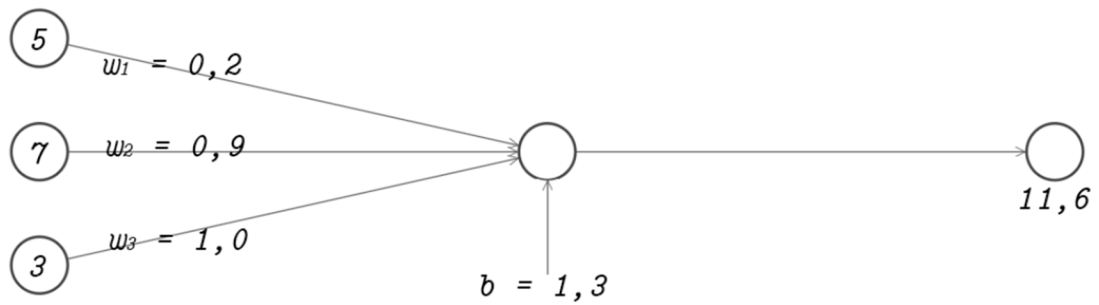
$$y = w * x + b \quad (1)$$



6. ábra Egy neuron kimeneti grafikonja

⁸<https://towardsdatascience.com/the-basics-of-neural-networks-neural-network-series-part-1-4419e343b2b> (2023.11.05.)

Például:



$$y = (5 * 0,2 + 7 * 0,9 + 3 * 1) + 1,3 = 11,6 \quad (2)$$

7. ábra Egy neuron 3 bemenettel

Amint látható, a súlyok és torzítások segítenek befolyásolni a neuronok kimenetét, de ezt kissé eltérő módon teszik. Csak súlyokkal és torzításokkal nem lehet „bárhogy” formálni egy neuron kimenetét, ezért van szükségünk az aktivációs függvényekre. Az aktivációs függvényekkel tudjuk egy agyi neuronnak azt a tulajdonságát utánózni, hogy a neuron kimenete éppen aktív vagy sem, illetve azt is, hogy milyen mértékű legyen a kimenete. A kimenet mértéke, milyensége függ a választott aktivációs függvénytől. Miután elvégeztük a súlyokkal való szorzást és hozzá adtuk a torzítást, a kapott értéket még az aktivációs függvénnyel manipuláljuk. Az így kapott érték lesz a neuron tényleges kimenete. Ezzel a neuron kimenetének számítása az alábbi lesz.

$$y = g(w * x + b) \quad (3)$$

Mivel egy neuronnak több bemenete is lesz, ezért ezt a számítást az összes bemeneti értékkel el kell végezni.

$$y = g(w_1 * x_1 + w_2 * x_2 + \dots + w_n * x_n + b) \quad (4)$$

2.4 Aktivációs függvények

Az aktivációs függvény egy olyan függvény, ami a neuron kimenetét számítja ki, befolyásolja a bemenetek súllyal való szorzás és torzítás eredményét a számítás után. Különböző esetekben különböző aktivációs függvényeket használunk, és ezek működésének megértése szükséges, hogy megfelelően kiválaszthassuk, melyik a legjobb az adott feladathoz. Az aktivációs függvényeket azért használjuk, mert ha maga az aktivációs függvény önmagában nemlineáris, attól még két vagy több rejtett rétegű neurális hálózatok számára lehetővé teszi a nemlineáris függvények leképezését.

Általában egy neurális hálóban kétféle aktivációs függvényt használunk. Az egyik az, ami a rejtett rétegek neuronjainak kimenete, a másik, pedig a kimeneti réteg aktivációs függvénye. Leggyakrabban a rejtett rétegek aktivációs függvényei megegyeznek egymással, de ez nem szükségszerű.

A következő fejezetek néhány aktivációs függvényt ismertetnek. A bemutatott függvényeket az egyszerűségük és alapvető könnyű használatuk miatt választottam.

2.4.1 Lépcsős függvény

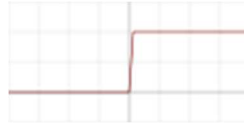
Az úgy nevezett Heaviside-függvény, vagy műszaki területen egységugrás függvény, mint aktivációs függvény.

Ennek az aktivációs függvénynek a működése, hogy a bemeneti információk alapján, az adott neuronnak 1 legyen a kimenete vagy 0, tehát aktív vagy inaktív. Egyetlen neuronban, ha a $súlyok * bemenet + torzítás$ 0-nál nagyobb értéket ad vissza, a neuron aktiválódik, és 1 lesz a kimenete ellenkező esetben 0-t ad ki.

Ezt az aktivációs függvényt a régebben rejtett rétegekben használták, de manapság már ritkán esik erre a választás.

A függvény tartománya $\{0,1\}$

$$y = \begin{cases} 0 & \text{ha } x < 0 \\ 1 & \text{ha } x \geq 0 \end{cases} \quad (5)$$



8. ábra Lépcsős függvény⁹

A lépcsős függvénnyel az a probléma, hogy nem túl informatív, mivel nagyon kevés információ gyűjthető be ennek a függvénynek a kimenetéből. Aktív (1), vagy inaktív (0). Nem lehet megmondani, mennyire volt „közel” az érték az aktiváláshoz vagy inaktiváláshoz. Talán nagyon közel volt, talán nagyon messze.

2.4.2 Signum függvény

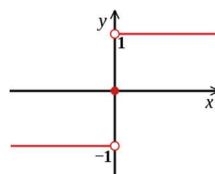
A signum függvény szinte teljesen megegyezik az előzőleg bemutatott lépcsős függvénnyel, a különbség a kimeneti tartománya. A kimenete 1, ha a bemenetén kapott érték nagyobb mint 0. Abban az esetben, ha a bemeneti értéke kisebb mint 0, akkor a függvény kimenete -1, illetve 0 a kimenete, ha a bemenet megegyezik 0-val.

Az ilyen típusú lépéses aktiválási függvény hasznos lehet bináris osztályozási feladatoknál. Más szóval, ha egy bemeneti mintát két csoport valamelyikébe szeretnénk besorolni, használhatunk egy bináris osztályozót egy signum aktiválási funkcióval.

Hasonlóan a lépcsős függvénnyel, ez a megoldás sem ad információt a bemenetéről, így, ha a kimenetét kiszámítjuk, abból már nem lehet visszafejteni a bemenetet.

A függvény tartománya $\{-1,0,1\}$

$$y = \begin{cases} -1 & \text{ha } x < 0 \\ 0 & \text{ha } x = 0 \\ 1 & \text{ha } x > 0 \end{cases} \quad (6)$$



9. ábra Signum függvény¹⁰

⁹ https://en.wikipedia.org/wiki/Activation_function (2023.11.07.)

¹⁰ <https://hu.wikipedia.org/wiki/Signumf%C3%BCggv%C3%A9ny> (2023.12.12.)

2.4.3 A lineáris függvény

A lineáris függvény egyszerűen az egyenes egyenlete. Grafikon ábrázolásakor egyenes vonalként jelenik meg, ahol $y=x$ és a kimeneti érték megegyezik a bemeneti értékkel.

Ez az aktivációs függvény leginkább az utolsó, kimeneti réteg során alkalmazható, például regressziós modell esetén. Olyan modell megvalósítása során, amely skaláris értéket ad ki.

A függvény tartománya: $(-\infty, \infty)$

$$y = x \quad (7)$$



10. ábra Lineáris függvény¹¹

2.4.4 A szigmoid függvény

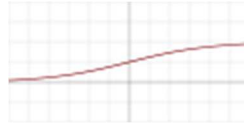
Eredetileg a neurális hálózatokhoz használt részletesebb információval bíró aktivációs függvény a szigmoid.

A szigmoid esetén olyan értéket kapunk kimeneteként, amely visszafordítható az eredeti értékre. A visszaadott érték tartalmazza az összes információt a bemenetről, ellentétben az olyan függvényekkel, mint a lépcsős függvény, ahol, ha a bemenet értéke 3, ugyanazt az értéket adná ki, mint ha a bemenet 300 000 lenne. A szigmoid függvény kimenete 0-tól 1-ig terjedő tartományban a neurális hálózatokkal is jobban működik. Ezen felül nemlinearitást ad a hálózathoz. A régebben rejtett rétegekben használt szigmoid függvényt végül a ReLU (Rectified Linear Unit) váltotta fel.

A függvény tartománya: $(0,1)$

$$y = \frac{1}{1 + e^{-x}} \quad (8)$$

¹¹ https://en.wikipedia.org/wiki/Activation_function (2023.11.07.)



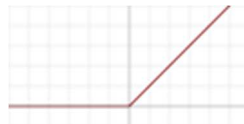
11. ábra Sigmoid függvény¹²

2.4.5 A ReLU függvény

Az „egyenirányított lineáris aktiválási” függvény Rectified Linear Unit (ReLU) egyszerűbb, mint a szigmoid. A függvény $y=x$, a negatív oldalról 0-ra vágva. Ha x kisebb vagy egyenlő, mint 0, akkor y értéke 0, ellenkező esetben y egyenlő x -szel. Ez az egyszerű, de hatékony aktiválási függvény a legszélesebb körben használt aktiválási függvény, különféle okokból. Elsősorban a sebesség és a hatékonyság miatt. Bár a szigmoid aktiválási függvény nem a legbonyolultabb, kiszámítása még mindig sokkal nagyobb kihívást jelent, mint a ReLU függvényé. A ReLU aktiválási függvény rendkívül közel áll a lineáris aktiválási függvényhez, miközben nemlineáris marad, a 0 utáni hajlítása miatt. Ez az egyszerű tulajdonsága azonban nagyon hatékonná teszi.

A függvény tartománya: $[0, \infty)$

$$y = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \quad (9)$$



12. ábra ReLU aktivációs függvény¹³

¹² https://en.wikipedia.org/wiki/Activation_function (2023.11.07.)

¹³ https://en.wikipedia.org/wiki/Activation_function (2023.11.07.)

2.4.6 Egyéb függvények

A bemutatott függvényeken kívül még számos olyan függvény létezik, amely alkalmas egy neuron aktivációs funkcióját ellátni.

Ezek a függvények például: [9]

Hyperbolic tangent:

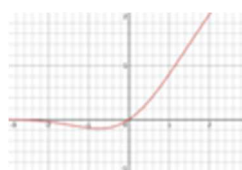
$$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



13. ábra Hyperbolic tangent

Gaussian:

$$y = e^{-x^2}$$



14. ábra Gaussian

Softplus:

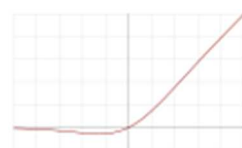
$$y = \ln(1 + e^x)$$



15. ábra Softplus

Sigmoid linear unit (SiLU):

$$y = \frac{x}{1 + e^{-x}}$$



16. ábra Sigmoid linear unit

2.5 Veszteség, hiba

A neurális háló hibaszámítása alapján történik a hálózat súlyainak, torzításainak frissítése. A hibaszámítás első lépése az, hogy összehasonlítjuk a neurális hálózat kimenetét a kívánt kimenettel, igazságokkal. Ez a különbség adja meg a hiba mértékét, amelyet valamilyen hibafüggvény segítségével számítunk ki. A hibafüggvény kiválasztása függ a konkrét feladattól és a neurális hálózat típusától.

Regressziós hiba kiszámításának két fő módszere az átlagos négyzetes hiba (Mean Squared Error - MSE) és az átlagos abszolút hiba (Mean Average Error - MAE). Egy másik gyakran használt hibafüggvény a keresztentrópia hiba (cross-entropy loss), amely különösen osztályozási feladatok hiba megállapítására alkalmas. Ezen kívül még számos módszer létezik, amivel meg lehet állapítani egy neurális háló hibáját.

A hibaszámítás során fontos figyelembe venni az aktivációs függvények hatását, amelyek befolyásolják a neurális hálózat kimeneteit. A hiba kiszámítása során a láncszabályt alkalmazzuk, ami lehetővé teszi a hiba visszatérjesztését (backpropagation) a hálózat rétegein keresztül. A hibagradiens segítségével optimalizáljuk a hálózat súlyait, hogy csökkentsük a teljes hibát.

A hibaszámítás nem csak a tanítási fázisban fontos, hanem a hálózat validálásakor és tesztelésekor is, mivel ezek alapján értékelhetjük a hálózat teljesítményét. Az iteratív tanulási folyamat során a cél az, hogy a hibaszámítással és ez alapján súly és torzítás módosítással folyamatosan csökkentsük a hálózat általános hibáját, ezzel javítva annak pontosságát és megbízhatóságát.

Az alább bemutatott hibaszámítási megközelítések a legnépszerűbbek közé tartoznak, és számítási módjuk egészen egyszerű. Hiba vagy veszteség számítás során az eredmény kiszámításának metodikája könnyedén nagyon bonyolulttá válhat.

2.5.1 Átlagos abszolút hiba (MAE)

Az átlagos abszolút érték egy statisztikai módszer a hibák mérésére. Ez az érték az egyes neuronok kimenete és az előre definiált igazságok közötti abszolút eltérések átlagát mutatja meg. A MAE segít megítélni, hogy egy modell mennyire pontos a valós értékek előrejelzésében. Az alacsonyabb MAE érték azt jelenti, hogy a modell pontosabb, míg a magasabb érték nagyobb eltérést jelez a valós és a becsült értékek között. A MAE előnye,

hogy könnyen értelmezhető és számolható, viszont nem veszi figyelembe az eltérések irányát és nagyságát. [38]

Átlagos abszolút hiba:
$$L_i = \frac{1}{n} \sum_{i=1}^n |y_i - \check{y}_i| \quad (10)$$

Például: Egy három kimenettel rendelkező háló vesztesége

igazságok = (1, 5, 9)

jósolt érték = (1, 5, 10)

$$\frac{(1 - 1) + (5 - 5) + (9 - 10)}{3} = -0,33\bar{3} \quad (11)$$

nagyobb tévedés esetén:

igazságok = (1, 5, 9)

jósolt érték = (7, 1, 15)

$$\frac{(1 - 7) + (5 - 1) + (9 - 15)}{3} = -2,66\bar{6} \quad (12)$$

2.5.2 Átlagos négyzetes hiba (MSE)

Egyik leggyakoribb hibafüggvény az átlagos négyzetes hiba, amely az előre jelzett és a valós értékek közötti eltérések négyzetének átlagát veszi.

Az átlagos négyzetes hiba négyzetre emeli az egyes kimenetek becsült és valós értékei közötti különbséget, és átlagolja ezeket az értékeket. Az átlagos négyzetes hiba lényege, hogy kisebb eltérésre kisebb választ ad, viszont nagyobb eltérésre négyzetesen nagyobb reakciót ad így az optimalizáló nagyobb módosítást hajt végre, ezzel gyorsabban elérhető a kívánt eredmény és így gyorsabban „feltanul” a neurális háló. [39]

Átlagos négyzetes hiba:
$$L_i = \frac{1}{n} \sum_{i=1}^n (y_i - \check{y}_i)^2 \quad (13)$$

Például:

igazságok = (1, 5, 9)

jósolt érték = (1, 5, 10)

$$\frac{(1-1)^2 + (5-5)^2 + (9-10)^2}{3} = -0,33\dot{3} \quad (14)$$

nagyobb tévedés esetén:

igazságok = (1, 5, 9)

jósolt érték = (7, 1, 15)

$$\frac{(1-7)^2 + (5-1)^2 + (9-15)^2}{3} = 29,33 \quad (15)$$

A példákból látható, az MSE és a MAE közötti különbség. Jelen esetben kisebb hibára a hiba értéke megegyezik, viszont nagyobb eltérés esetén a hiba értéke az MSE használata esetén lényegesen nagyobb.

2.5.3 Kategorizált keresztentrópia veszteség

A kategorizált keresztentrópia (Categorical Cross-Entropy Loss) veszteség, kategorizálási veszteség megállapítására való. Ezt a veszteség számítási módszert kifejezetten az alapigazságok és a jósolt értékek összehasonlítására használjuk. Ez az egyik leggyakrabban használt veszteségfüggvény a kimeneti rétegen softmax aktiválással.[39]

Kategorizált keresztentrópia veszteség: $L_i = - \sum_i (y * \ln(y_i))$ (16)

Például:

Egy hálózatnak a szerepe, hogy besorolja a képet. A hálózat állatokra lett feltanítva és jelen esetben kap egy képet egy kutyáról. A neurális hálónak 3 kimenete van, amelyek a kutya, macska és ló kategóriákat reprezentálják.

A kimenet 0 és 1 közötti értéket vehet fel, míg az igazság, vagy célzott kimenetből csak az egyik érték lehet 1 míg a többi 0, mert a képen vagy kutya vagy macska vagy ló van. Az ehhez hasonló tömböket, vagy vektorokat one-hot-nak nevezzük, ami azt jelenti, hogy csak az egyik érték „forró” (bekapcsolva), a többi pedig „hideg” (kikapcsolva).

$$\text{célzott kimenet} = (1, 0, 0)$$

$$\text{jósolt érték} = (0.1, 0.9, 0.7)$$

$$-1 * (\ln(1 * 0.1) + \ln(0 * 0.9) + \ln(0 * 0.7)) = 2.302 \quad (17)$$

A veszteség ebben az esetben 2,302. Tegyük fel, hogy a hálózat feltanításra kerül és az adott képet nagy százalékban felismeri.

$$\text{célzott kimenet} = (1, 0, 0)$$

$$\text{jósolt érték} = (0.9, 0.2, 0.1)$$

$$-1 * (\ln(1 * 0.9) + \ln(0 * 0.2) + \ln(0 * 0.1)) = 0.105 \quad (18)$$

A veszteség értéke lényegesen kisebb lesz.

2.6 Visszaterjesztés (backpropagation)

A visszaterjesztés (backpropagation) az az algoritmus, ami hálózat kimeneti hibáját használja fel annak érdekében, hogy finomítsa a hálózat súlyait és elősegítse a tanulási folyamatot. A visszaterjesztés működésének alapja az a felismerés, hogy a hiba hogyan változik a hálózat egyes súlyainak módosításával. A hiba kiszámításához a hálózat kimenetét összehasonlítjuk a várt kimenettel, és az eltérések alapján számítjuk ki a hibát.

Ezt követően a hiba gradiensét használjuk fel a súlyok frissítéséhez, amelyet a hálózat minden egyes rétegében végrehajtunk. A gradiens az a vektor, amely megmutatja, hogy mely irányban kell változtatni a súlyokat a hiba csökkentése érdekében. A visszaterjesztés során a hiba gradiensét a hálózat kimenetétől visszafelé, a bemeneti réteg felé terjesztjük vissza. Ez lehetővé teszi, hogy minden rétegben pontosan meghatározhassuk, hogyan kell módosítani a súlyokat.

A visszaterjesztés egy iteratív folyamat, amely során a hálózaton többször is áthaladva finomhangoljuk a súlyokat. A visszaterjesztés hatékonysága abban rejlik, hogy képes megtalálni az optimális súlybeállításokat a hálózat számára, amelyekkel a legpontosabb kimenetet érhetjük el.

A visszaterjesztés a függvények deriváltjain alapul. Egy neuron kimenetének kiszámítási folyamata, hogy a súlyokat szorozzuk a bemenetekkel, majd ezekhez

hozzáadjuk a torzítást. Az így kapott kimenetet egy aktivációs függvénnyel számítjuk tovább, és így megkapjuk a neuron kimenetét. A visszaterjesztés során vesszük az aktivációs függvény deriváltját, majd a torzítás és a szorzás deriváltját. A derivált értékeket összeszorozzuk és a láncszabály alapján ezekkel az értékekkel meghatározható a gradiense a neuron kimenetének. Így, hogy ismerjük ezt az értéket, tudjuk, hogy melyik irányba, és mekkora mértékben kell módosítani a súlyokat és torzításokat a pontosabb eredmény eléréséhez.

2.7 Neurális háló tanítása

A neurális hálózat feltanítása során először adatokkal kell ellátni a hálózatot, amelyek lehetnek képek, szövegek, vagy más típusú információk. Ezek az adatok szolgálják a hálózat bemenetét, és teszik ki a tanítási adathalmazt. Ezeket az adatokat címkézni kell, hogy a hálózat megérthesse, milyen eredményt várunk el tőle.

Megkezdve a hálózat tanítását a súlyoknak és a torzításoknak célszerű véletlenszerű értékeket beállítani. Ezek az értékek általában kis értékek. A tanítási folyamat során a hálózat súlyait és paramétereit fokozatosan állítják be, hogy a kívánt eredményeket produkálja. Ez a beállítás a hiba visszaterjesztésének módszerén keresztül történik, ahol a hálózat kimenetének és a kívánt kimenet közötti különbséget minimalizáljuk. Ezt a feladatot az optimalizáló segítségével hajtjuk végre.

A neurális hálózatok tanítása iteratív folyamat, ami azt jelenti, hogy a hálózatot többször kell "edzeni" ugyanazon adatkészleten. Az úgynevezett tanulási ráta szabályozza, hogy milyen gyorsan módosulnak a hálózat súlyai a tanulási folyamat során. Ezt a tanulási rátát szintén az optimalizáló algoritmus kezeli.

A túltanulás (overfitting) elkerülése érdekében gyakran használnak különféle technikákat, mint például a korai leállítást vagy a dropoutot, vagyis „neuron kidobást”.

A tanítás folyamatai közé tartozik a validáció is, ahol a modell teljesítménye kerül tesztelésre még nem látott adatokon. A neurális hálózatok mély tanulás (deep learning) esetén több rejtett réteggel rendelkezhetnek, amelyek összetettebb jellemzőket képesek megtanulni. A sikeresen betanított neurális hálózat képes önállóan döntéseket hozni, mintapéldák alapján, még olyan helyzetekben is, amelyekkel a tanítás során nem találkozott.

2.8 Adatkészletek

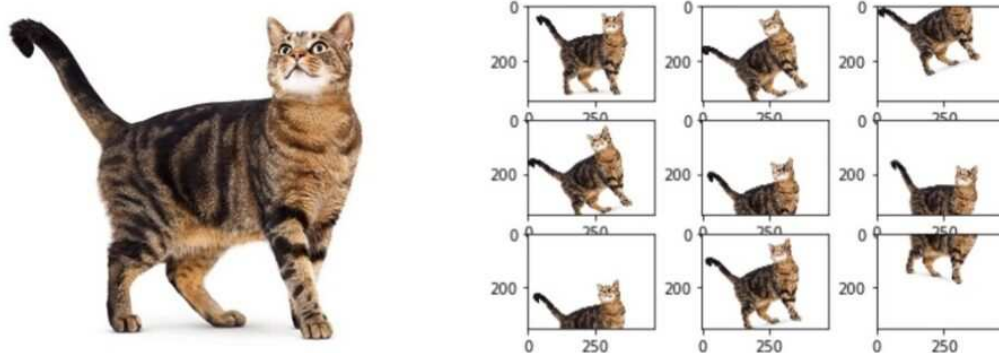
Egy neurális hálózat feltanításához többféle adatkészlet, és megközelítés is alkalmazható. Ezek az adatkészletek mind más-más szerepet látnak el.

A "training adat" az adatkészlet azon része, amely közvetlenül a neurális hálózat tanítási folyamatában van felhasználva, ez a tanítási adathalmaz. Ezeken az adatokon keresztül a modell megtanulja azokat a mintázatokat és összefüggéseket, amelyek szükségesek a cél eléréséhez. Ezek azok az adatok, amelyen a tanítási folyamat ciklikusan többször is átfut.

A validációs adat egy különálló adathalmaz, amit a modell teljesítményének ellenőrzésére használatosak a tanítási fázis során. A validációs adat segítségével észlelhető a túltanulás, amikor a modell túlságosan az adott tanító adatokra „tanul rá”. A validációs adatok nem részei a tanító adathalmaznak, így a modell sosem "látja" őket a tanulás során. Ezen felül a validációs adatokat alkalmazhatjuk még arra a célra, hogy a tanulási rátát állítsuk vagy finom hangoljuk.

A data augmentation vagyis adatbővítés olyan technika, amely mesterségesen növeli a tanító adatkészlet méretét és sokféleségét. Ezáltal a modell jobban generalizálhat és ellenállóbbá válhat a nem látott adatokkal szemben. Az adatbővítés gyakori módszerei közé tartozik az adatok forgatása, méretezése, tükrözése vagy zaj hozzáadása. Az sokszorozott adatok segítségével a neurális hálózat megtanulhatja felismerni az objektumokat különböző orientációkban és körülmények között. A képfeldolgozásban az adatbővítés különösen fontos, mivel segít a modellnek jobban megbirkózni a valós világ változatosságával. Azonban a túlzott adatbővítés torzíthatja az adatokat és rossz teljesítményhez vezethet.

Az alábbi adatbővítés példában, ugyan azt a képet láthatjuk a macskáról, csak az egyik esetben elforgatva, a másik esetben bizonyos része le van vágva a képnek. És így növelve az adat mennyiségét.



17. ábra Adat augmentáció példa¹⁴

A megfelelő adatok nélkül a modell nem lesz képes tanulásra és a valós világ problémáinak megoldására. Az adatkészleteknek reprezentatívnak kell lenniük a probléma szempontjából, hogy a modell jól generalizáljon. A neurális hálózatok különböző típusai különböző adatformátumokat és struktúrákat igényelnek.

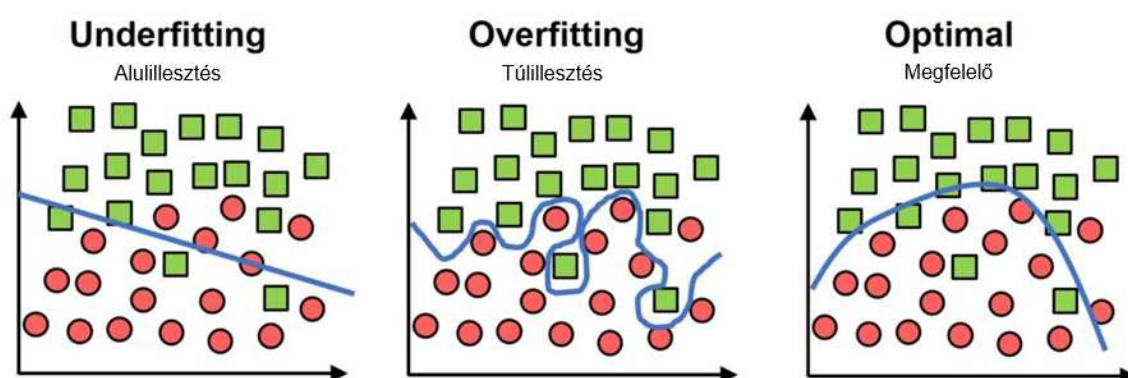
Egy neurális háló feltanításának sikerességét ugyan a tanítási adat határozza meg leginkább, de az adatok előkészítése és tisztítás folyamata is fontos szerepet játszik.

2.9 Túltanulás (overfitting)

A túltanulás (overfitting)[37] egy olyan jelenség a gépi tanulásban, amikor egy modell túlságosan illeszkedik a tanítóadatokra és nem képes generalizálni új, még nem látott adatokat. Ez azt jelenti, hogy a modell olyan részleteket is megtanul, amelyek csak a tanítóadathalmazra jellemzőek, és nem általánosíthatók más adatkészletekre. Túltanulás általában akkor fordul elő, amikor a modell túl komplex, vagy amikor nem elegendő a tanítóadat mennyisége. A túltanulás megakadályozza, hogy a modell hatékonyan alkalmazkodjon új, ismeretlen adatokhoz, mivel a modell a tanítóadatokon lévő véletlen zajokat is "megtanulja". A túltanulást validációval és regularizációval lehet csökkenteni, melyek segítenek a modell általánosító képességének javításában. A túltanulás felismerése lényeges a modell teljesítményének értékelésében.

¹⁴ <https://datahacker.rs/018-pytorch-popular-techniques-to-prevent-the-overfitting-in-a-neural-networks/>
(2023.11.23.)

Ha a modell túl bonyolult, néha elkezdheti megtanulni az adatkészletben lévő irreleváns információkat, zajokat. Ez azt jelenti, hogy a modell megjegyzi azt a zajt, amely csak a tanítási adatkészlethez kapcsolódik. Ebben az esetben a modell nagyon pontatlan, mivel a memorizált minta nem tükrözi az adatokban jelenlévő fontos információkat. Egy ilyen modellre azt mondjuk, hogy túl van illesztve, és nem képes jól általánosítani új adatokat. Rendkívül jól megtanulta a tréningkészlet jellemzőit, de ha olyan adatot adunk meg a modellnek, amely kismértékben eltér a tanítás során használt pontos adatoktól, akkor nem tudja általánosítani és pontosan megjósolni a kimenetet. [10]



18. ábra Tanítás hibái¹⁵

A 18. ábra szemlélteti a tanunási hibákat.

A bal szélső képen az alulillesztés látható. Alulillesztésről beszélünk mikor a modell nem képes különbséget tenni az adatok között megfelelő módon.

A középső képen az túlillesztés látható. Túlillesztés esetén nem megbízható a modell és túlságosan is azokra az adatokra tanul rá, amelyeken tanítva lett.

A jobb szélső képen az ideális feltanítás látható, itt a modell megtanulta a részleteket, és képes lehet generalizálni azokat az adatokat, amiket még nem látott.

2.10 Dropout

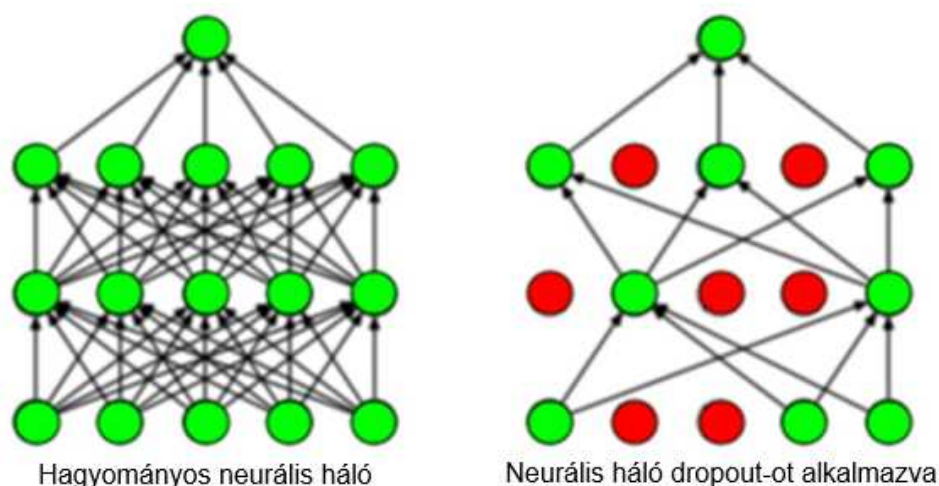
A dropout [36], vagy „neuron kidobás” egy népszerű regularizációs technika a gépi tanulásban. Ennek a módszernek az alapja, hogy a tanítási fázis során véletlenszerűen "kikapcsolunk" egyes neuronokat, ezzel csökkentve a hálózat bonyolultságát. A dropout

¹⁵ <https://datahacker.rs/018-pytorch-popular-techniques-to-prevent-the-overfitting-in-a-neural-networks/>
(2023.11.23.)

segít megelőzni a túltanulást, mivel megakadályozza, hogy a hálózat túlzottan támaszkodjon egyes bemenetekre vagy neuronokra. Ezáltal a hálózat megbízhatóbbá válik, mivel megtanulja kezelni a hiányzó információkat és nem függ egyes neuronok specifikus aktivitásától.

A dropout mértékét egy hiperparaméterrel szabályozhatjuk, amely meghatározza, hogy milyen arányban kapcsoljuk ki a neuronokat. A dropout alkalmazása legfőképp a rejtett rétegekre terjed ki, és elkerülendő a bemenet, illetve a kimeneti rétegek alkalmazása során. A tanítás befejezése után, a modell alkalmazásakor, tesztelésekor és validáció során már nem használjuk a dropout, az összes neuront aktívan tartjuk. A dropout hatékonyan csökkenti a neurális hálózatok túltanulását, miközben viszonylag egyszerű implementálni. Ez a módszer ma már széles körben elterjedt és fontos eszköze a mélytanulási modellek optimalizálásának.

Az alábbi képen egy példát láthatunk a dropout alkalmazására. A bal oldali képen az a hálózat látható, amelyben nincs használva a dropout technika, ennek a hálózatnak sűrű rétegei vannak, vagyis mindegyik neuron kapcsolódik az előtte és mögötte lévő neuronnal. A jobb oldali képen egy olyan hálózat látható, amiben alkalmazva van a dropout, itt a piros neuronok teljes mértékben ki vannak kapcsolva, és a kimenetük is nulla, így egyáltalán nem tudják módosítani a neurális hálónak a kimeneteit.



19. ábra Dropout alkalmazása¹⁶

¹⁶ <https://datahacker.rs/018-pytorch-popular-techniques-to-prevent-the-overfitting-in-a-neural-networks/>
(2023.11.23.)

2.11 Optimalizáció

Az optimalizáció a gépi tanulásban egy olyan folyamat, amelynek célja a legjobb súly és torzítás értékek megtalálása a modell teljesítményének érdekében. Az optimalizációs algoritmusok a veszteség, hiba minimalizálására törekszenek, ami azt jelenti, hogy a legpontosabb előrejelzéseket eredményező paramétereket keresik.

Az optimalizációs algoritmusoknak hiperparaméterei vannak, amikkel lehet finomhagolni a feltanulásnak számos részét. A hiperparaméterek olyan konfigurációs változók, amelyeket a tanulási folyamat előtt állítunk be, és ezeket nem tanulja meg közvetlenül a modell. A hiperparamétereket csak a feltanítás során alkalmazunk, a modell használata közben már nincs szükség ezekre.

A hiperparaméterek közé tartozik például a tanulási ráta, amely meghatározza, milyen gyorsan igazítja a modell a súlyokat a tanulás során. Egy másik fontos hiperparaméter a batch méret, ami befolyásolja, hogy a modell hány adatpontot használ fel egyetlen tanítási lépés, ciklus során. A regularizációs paraméterek, mint például az L1 vagy L2 regularizáció, segítenek elkerülni a túltanulást a modell súlyainak korlátozásával. A regularizációknak néhány példáját egy későbbi fejezetben ismertetem.

Az optimalizációs algoritmusok széles skálája áll rendelkezésre, mindegyik különböző stratégiákkal és tulajdonságokkal. Az egyik legelterjedtebb a sztochasztikus gradiens csökkentés (SGD), amit a következőkben be is mutatok.

A momentum alapú algoritmusok, mint például a Nesterov Accelerated Gradient (NAG) [25], a korábbi gradiens lépések "sebességét" is figyelembe veszik, ami gyorsabb konvergenciát eredményezhet. Az adaptív tanulási rátájú algoritmusok, mint az AdaGrad és Adam, automatikusan módosítják a tanulási rátát a tanulás során, ami segít a hatékonyabb optimalizációban. Ezek az algoritmusok különösen hasznosak nagy méretű adatkészletek és összetett modellek esetén.

A megfelelő optimalizációs stratégia kiválasztása a modell jellegétől, a rendelkezésre álló adatoktól, és a kívánt eredményektől függ.

A következőkben bemutatok néhány optimalizációs algoritmus. Azért választottam ezeket az algoritmusokat, mert alapvető információkat hordoznak magukban, például az SGD az egyik legrégebbi és leginkább alapvető optimalizáló módszer, amely széles körben alkalmazható és egyszerűen implementálható. Egy másik szempont, hogy ezekkel az

algoritmusokkal sok kutatás és irodalom foglalkozik, így széleskörű figyelmet kap. Végül, ezek az algoritmusok alapvető építőkövei a modern tanulási architektúráknak, így a kutatások gyakran ezekre a módszerekre épülnek vagy ezeket használják kiindulópontként.

2.11.1 Sztochasztikus gradiens csökkentés (SGD) [26]

Az algoritmus célja, hogy minimalizálja a veszteségfüggvény értékét, ami egy mérőszám arra, hogy mennyire jól teljesít a modell a tanító adatokon.

A sztochasztikus gradiens csökkentés esetén tanulási sebességet választunk (learning rate), például 1,0-t. Ezután kivonjuk a *learning rate* · *parameter gradiens* értéket a tényleges paraméterértékekből. Ha a tanulási arányunk 1, akkor nem befolyásolja az értékek módosítását.

Ennek az optimalizálónak a használatával könnyen ragadhat a tanítási fázis egy helyi minimumban, és így megáll a tanulási folyamat.

$$\begin{aligned} \text{súlyok} &= \text{learning rate} * \text{súly csökkentés} \\ \text{torzítások} &= \text{learning rate} * \text{torzítás csökkentés} \end{aligned} \tag{19}$$

2.11.2 Sztochasztikus gradiens csökkentés momentummal [27]

A momentum vagy lendület bizonyos számú iteráció során létrehozza a gradiensnek gördülő átlagát, és ezt az átlagot használja az egyedi gradienssel minden lépésnél. Ez segíthet azokban az esetekben, amikor valamilyen helyi minimumban megáll a tanulás és nem halad tovább. A momentummal a modell nagyobb valószínűséggel megy át a helyi minimumokon, tovább csökkentve ezzel a veszteséget, hibát.

A lendületet úgy használjuk ki, hogy 0 és 1 közé állítunk be egy paramétert, amely az előző paraméterfrissítés megtartandó hányadát jelenti, és ebből kivonjuk a tényleges gradienst, megszorozva a tanulási sebességgel. A frissítés az előző lépésekből származó gradiens egy részét tartalmazza. Ezek a részek együtt alkotják a paraméterek tényleges változását, és minél nagyobb szerepet kap a lendület a frissítésben, annál lassabban tudja megváltoztatni az irányt. Ha túl magasra állítjuk a lendület értékét, előfordulhat, hogy a modell egyáltalán nem tanul, mivel a frissítések irányát nem tudja követni a globális gradiens süllyedés.

$$\begin{aligned} \text{súlyok} = & \text{momentum} * \text{súly momentum} - (\text{learning rate} \\ & * \text{súly csökkentés}) \end{aligned} \quad (20)$$

$$\begin{aligned} \text{torzítások} = & \text{momentum} * \text{torz.momentum} - (\text{learning rate} \\ & * \text{torz.csökkentés}) \end{aligned}$$

Ezek után a kiszámított súly értéket eltároljuk a súly momentum-ban. A súly momentumnak az első iterációban 0 értéke van.

$$\text{súly momentum} = \text{súlyok} \quad (21)$$

$$\text{torzítás momentum} = \text{torzítások}$$

Az iterációk során a learning rate értékét csökkentjük. Ezt az értéket a decay hiperparaméterrel tesszük meg.

$$\text{learning rate} = \text{learning rate} * \left(\frac{1}{1 + \text{decay} * \text{iterációk száma}} \right) \quad (22)$$

2.11.3 AdaGrad [29]

Az AdaGrad, az adaptív gradiens rövidítése, paraméterenkénti tanulási arányt vezet be, nem pedig globálisan megosztott értéket. Az ötlet az, hogy normalizáljuk a funkciók frissítéseit. A tanítási folyamat során egyes súlyok jelentősen megemelkedhetnek, míg mások nem sokat változnak. Általában jobb, ha a súlyok nem emelkednek túlzottan ki a többi súlyhoz képest. Az AdaGrad módot biztosít a paraméterfrissítések normalizálására a korábbi frissítések előzményeinek megőrzésével. Minél nagyobb a frissítések összege mindkét irányban (pozitív vagy negatív), annál kisebb frissítéseket hajtanak végre a tanítás során. Ez lehetővé teszi, hogy a kevésbé gyakran frissített paraméterek lépést tarthassanak a változásokkal, ezzel elérve, hogy minden neuron részt vegyen a tanulási folyamatban.

$$\begin{aligned} \text{cache} &= \text{cache} + \text{parm gradiens}^2 \\ \text{parm frissítés} &= \text{learning rate} * \frac{\text{parm gradiens}}{\sqrt{\text{cache} + \varepsilon}} \end{aligned} \quad (23)$$

A cache a négyzetes gradiens előzményeit tartalmazza, és a parm frissítés a tanulási sebesség és a gradiens (eddig alap SGD) szorzatának függvénye, majd elosztva a cache

négyzetgyökével plusz némi epsilon értékkel. A folyamatosan növekvő cache érték a tanulás leállítását is okozhatja, mivel a frissítések idővel kisebbek lesznek, a frissítések monoton jellege miatt. Ezért ezt az optimalizálót nem használják széles körben, kivéve néhány speciális alkalmazást. Az epsilon egy hiperparaméter, amely megakadályozza a 0-val való osztást. Az epsilon értéke általában kicsi, például $1 * 10^{-7}$. A négyzetes gradiens értéket összegezzük és csak később számítjuk ki a négyzetgyökét.

2.11.4 Adam [30]

Az Adam (Adaptive Moment) optimalizáló egy népszerű gradiens-alapú optimalizálási algoritmus a gépi tanulásban, amely adaptívan szabályozza a tanulási rátát. Az Adam három fő összetevőt használ: az első momentum, a gradiens átlaga és a második momentum, a gradiens négyzeteinek átlaga, amelyek segítenek a súlyok finomításában. Az első momentum segít a konvergencia gyorsításában, míg a második momentum stabilizálja az optimalizációs folyamatot a különböző gradiens értékek változékonyságának kezelésével.

2.12 Regularizáció [35]

A regularizációs módszerek azok, amelyek csökkentik az általánosítási hibát. Az L1 és L2 regularizációt egy szám, úgynevezett büntetés kiszámítására használjuk, amely hozzáadódik a veszteség értékéhez, hogy „megbüntesse” a modellt a nagy súlyok és torzítások miatt. A nagy súlyok azt jelezhetik, hogy egy neuron egy adatelemet próbál meg memorizálni. Egy hálózat sokkal jobban működik, hogy ha az összes neuron hozzájárul a kimenet létrehozásához és nem pedig csak néhány, amit „kiválasztott” a hálózat.

Az L1 regularizáció a súlyok és torzítások abszolút értékének összege. Ez egy lineáris büntetés, mivel a függvény által visszaadott szabályozási veszteség egyenesen arányos a paraméterértékekkel. Az L2 regularizáció a súlyok és torzítások négyzetének összege. Ez a megközelítés a nagyobb értékeket jobban bünteti, mint a kisebbeket a négyzetre emelés miatt. Más szavakkal, az L2 regularizációt gyakrabban használják, mivel nem befolyásolja lényegesen a kis paraméterértékeket, és nem teszi lehetővé, hogy a modell túlságosan nagyra nőjön azáltal, hogy erősen bünteti a viszonylag nagy értékeket.

Az L1 regularizáció lineáris jellege miatt jobban bünteti a kis súlyokat, mint az L2 regularizáció, ami azt eredményezi, hogy a modell invariáns lesz a kis bemenetekre, és csak a nagyobbakra változik. Ez az oka annak, hogy az L1 regularizációt ritkán használják önmagában, és általában kombinálják az L2 regularizációval. Az ilyen típusú regularizációs függvények a súlyok összegét és a paraméterek összegét 0 felé tolják, ami robbanásszerű gradiensek esetén is segíthet.

Az L1 és L2 regularizáció is hiperparaméterként jelennek meg. Ezeket az értékeket is lehet finhangolni. A büntetés mértékét a lambda szabja meg. Minél nagyobb a lambda értéke annál nagyobb a büntetés is.

L1 súlyregularizáció:

$$L_{1w} = \lambda \sum_m |w_m| \quad (24)$$

L1 torzításregularizáció

$$L_{1b} = \lambda \sum_n |b_n| \quad (25)$$

L2 súlyregularizáció:

$$L_{2w} = \lambda \sum_n w_n^2 \quad (26)$$

L1 torzításregularizáció

$$L_{2b} = \lambda \sum_n b_n^2 \quad (27)$$

A végleges veszteség, hiba értéke:

$$Veszteség = Hiba + L_{1w} + L_{1b} + L_{2w} + L_{2b} \quad (28)$$

2.13 Neurális hálót támogató hardver

A hardveres gyorsításnak köszönhetően érhetőek el azok az eredmények neurális hálózatokkal, amelyeket ma ismerünk.

A neurális hálózatok összetett számításokat igényelnek, amelyek számítás és időigényesek. Az olyan hardvergyorsítók, mint a GPU-k (Graphics Processing Units) és a TPU-k (Tensor Processing Units) úgy tervezték, hogy sokkal gyorsabban kezeljék ezeket a számításokat, mint egy általános célú CPU (Central Processing Unit). Párhuzamos feldolgozást tudnak végezni, ami jelentősen felgyorsítja a neurális hálózatok feltanítási és következtetési fázisait.

Az egyik legegyszerűbb különbség a CPU és a GPU között a magok száma. Ha összehasonlítjuk a kettőt, jelentős különbségek jönnek ki. CPU-ból és GPU-ból is ma kapható felső kategóriás termékeket kerestem. Processzorból az Intel® Core™ i9 processor 14900K¹⁷ eszköz, illetve az NVIDIA GeForce RTX 4090¹⁸ grafikai gyorsítónak a magjainak számát hasonlítottam össze. Míg az Intel processzor összesen 24 magot tartalmaz, addig az NVIDIA 16 384 darab magot foglal magában. Ez az érték abból a szempontból fontos, hogy egyszerűbb számításokat, mint ami egy neurális hálózathoz szükséges, összeadás, szorzás, egy GPU sokkal gyorsabban tud elvégezni párhuzamosan, mint egy CPU, és ez a magok számának köszönhető.

Hardverelemből már létezik olyan, ami kimondottan mesterséges intelligencia támogatására lett kitalálva, ilyenek a TPU-k. [31]

A gyors számítási sebesség elérését ezen felül még nehezíti, hogy rendkívül sok adattal kell dolgozni a nagy pontosság eléréséhez. A hardveres gyorsítók nagy mennyiségű adatot hatékonyabban tudnak feldolgozni.

Energiahatékonyság szempontjából a CPU-khoz képest a GPU-k és a TPU-k energiahatékonyabbak a neurális hálózatokban szükséges párhuzamos számításokhoz. Ez a hatékonyság a nagyméretű telepítéseknel és adatközpontokban, ahol az energiafogyasztás jelentős költséget jelenthet, rendkívül fontos.

A kimondottan erre a célra fejlesztett hardveres gyorsítók jobb méretezhetőséget tesznek lehetővé a neurális hálózati alkalmazásokban. A modell összetettségének vagy az adatkészlet méretének növekedésével ezek az eszközök képesek biztosítani a szükséges számítási teljesítményt.

¹⁷ <https://www.intel.com/content/www/us/en/products/sku/236773/intel-core-i9-processor-14900k-36m-cache-up-to-6-00-ghz/specifications.html> (2023.11.17.)

¹⁸ <https://www.nvidia.com/en-eu/geforce/graphics-cards/40-series/rtx-4090/> (2023.11.17.)

3 GÉPI LÁTÁS KATEGÓRIÁI

3.1 Mi a gépi látás?

A gépi látás olyan technológiai terület, amely lehetővé teszi a számítógépek és más digitális eszközök számára, hogy érzékeljék és értelmezzék a környezetüket vizuális adatok alapján, amelynek célja a döntéshozatali folyamatok automatizálása. A rendszerek képesek felismerni mintákat, azonosítani tárgyakat, követni mozgásokat és értelmezni térbeli információkat. A gépi látás fejlesztése magában foglalja az algoritmusok, a hardveres eszközök és a mesterséges intelligencia módszereinek ötvözését. A technológia fejlődése folyamatosan bővülő lehetőségeket nyit meg. Gépi látás segítségével az eszközök képesek önállóan döntéseket hozni, előre látni és reagálni a környezeti változásokra.

Ebben a fejezetben a gépi látás és ahhoz kapcsolódó alkalmazási megoldásoknak egy részét ismertetem. Milyen módon lehet alkalmazni a gépi látást, hogyan lehet csoportosítani és mire lehet alkalmazni. A bemutatott kategóriák a leginkább populáris témaköröknek egy része.

3.2 Mélységbecslés

A mélységbecslés vagy depth estimation egy olyan feladat, ami minden pixel távolságát hivatott meghatározni a képet készítő kamerához képest. A kimenete a mélységbecslésnek egy úgynevezett depth map, ami gyakorlatilag egy mátrix, amiben minden elem megfeleltethető egy a bemeneti kép pixelének becsült távolsági értékével.

Ez a technológia alkalmas 3 dimenziós távolságokat meghatározni egy egyszerű kép alapján. Ennek a feladatnak a végrehajtásához csak egy képre van szükség, ami azt jelenti, hogy gyakorlatilag bármilyen eszközt használhatunk, aminek az az előnye, hogy nincs szükség drága távolság meghatározásra fejlesztett eszközt használni, mint például LIDAR. A használatának előnye lehet, hogy szinte bármilyen képből meg lehet határozni egy mélység térképet, ellenben hátránya, hogy kevésbé pontos a lézeres megoldáson alapuló technológiákhoz képest.

A neurális háló kimenete, a depth map 3 értéket tartalmaz pixelenként, melyek az „x”, „y” és „z” koordináták. Az „x” és „y”, amely a képnek a sor és oszlop koordinátái. A „z” érték a becsült távolság. Ezzel a kimenettel lehetséges egy képet generálni, amely vizuálisan is megjeleníti a depth map-ot mint amit a 20. ábra is szemléltet.



20. ábra – Mélységbecslés bemeneti és kimeneti kép¹⁹

Az ilyen fajta képfeldolgozás kimenetét ezek után fel lehet használni egy 3D háló készítésére, és így egy képből generálni egy 3 dimenziós objektumot.

Egy tovább gondolt megvalósítás az úgy nevezett fotogrammetria. Ennek lényege, hogy egy adott objektumról több képet készítünk több szögből, úgy, hogy a képek között legyen átlapolás. Egy ilyen képhalmaz feldolgozása után egy teljes 3D modellt kaphatunk, ami a fényképezett objektum minden oldalának tartalmazza a részleteit.

A fotogrammetria a 3 dimenziós információk fotókból történő kinyerésének művészete és tudománya. A folyamat az objektumok, szerkezetek vagy területek átlapoló fotóinak készítését, és azok 2D-s vagy 3D-s digitális modellekké való konvertálását foglalja magában. [6]

A mélységbecslésnek gyakorlati haszna lehet a földmérőknek, építészeknek és mérnököknek, akik topográfiai térképek, hálózatok, pontfelhők vagy a valós világban alapuló rajzokat készítenek.

3.3 Képbesorolás

Az image classification, vagyis képbesorolás, az a folyamat, amikor egy neurális háló képes automatikusan kategorizálni a képeket előre meghatározott osztályokba. Az elmúlt évtizedben a konvolúciós neurális hálók forradalmasították ezt a területet, lehetővé téve számunkra, hogy rendkívül nagy pontossággal soroljunk be képeket, legyen szó természetes világban előforduló objektumokról, vagy akár emberi tevékenységekről.

¹⁹ https://huggingface.co/docs/transformers/main/en/tasks/monocular_depth_estimation#depth-estimation-pipeline (2023.11.03.)

Ez a megoldás az előre definiált csoportokba képes osztályozni a képeket, tehát ebben az esetben csak a tanítási adatszerkezet határozza meg, hogy mit vagyunk képesek osztályozni. Az ilyen fajta neurális háló nem csak képek besorolására alkalmas, hanem bármilyen számszerűsített bemenet, bemeneti halmaz osztályozására.

A hálózat bemenete egy vektor, amely az osztályozni kívánt kép pixeleinek értékét tartalmazza, illetve a kimenete szintén egy vektor, amelynek mérete az előre meghatározott és feltanított csoportok darabszámával egyezik meg.

A hálózat kimenete függ a kimeneti réteg aktivációs függvényétől, de általában egy szám, ami $[0,1]$ tartomány közé esik. Az értéke reprezentálja, hogy a neurális háló szerint mennyire tartozik a bemeneti kép az adott kategóriába. Ha az érték nulla, akkor egyáltalán nem tartozik az adott csoportba, ha egy, akkor teljesen biztos, hogy az a kategória.

A képbesorolásnak gyakorlati haszna lehet például fényképalbumban lévő képek kategorizálása, például: kisállatos képek, csoportkép, épületek stb., egy másik opció lehet, gyártás során alkatrész homogenitás vagy sérülés megállapítására, abban az esetben, ha csak egy darab alkatrész szerepel a képen.

Példa:



```
[{'score': 0.318 [...]: 'beignets'},  
{ 'score': 0.015 [...]: 'bruschetta'},  
{ 'score': 0.015 [...]: 'chicken_wings'},  
{ 'score': 0.0130 [...]: 'pork_chop'},  
{ 'score': 0.0127 [...]: 'prime_rib'}]
```

21. ábra – Képbesorolás és a kimenete²⁰

²⁰ https://huggingface.co/docs/transformers/main/en/tasks/image_classification
(2023.11.03.)

3.4 Objektumfelismerés

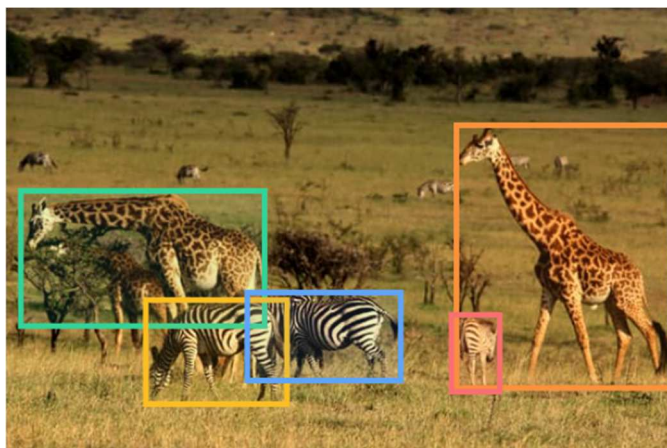
Az objektumfelismerés, vagy más néven object detection a képen lévő bizonyos elemek kiemelésére képes. Feladata azonosítani és lokalizálni a képen lévő objektumokat.

Az objektumfelismerés két fő feladatot lát el, az objektumok osztályozását, hogy mi is van a képen, illetve az objektumok detektálását, hogy hol vannak az objektumok a képen. A kimenetét tekintve határoló dobozokat kapunk, ezeknek a koordinátáit (x, y), szélességét és magasságát, illetve azt, hogy az adott határoló doboz melyik osztályba tartozik és milyen valószínűségi pontértéket kapott. Egy másik lehetőség, hogy a határoló dobozoknak a bal felső, illetve jobb alsó koordináta párját kapjuk vissza, ami determinálja számunkra a szélességet és magasságot is.

Egy kép több objektumot is tartalmazhat, mindegyiknek saját határolódobozja és címkéje van, például lehet egy autó vagy egy kutya. Ezen felül egy csoportba tartozó objektumok jelen lehetnek több helyen is a képen, például a képen több autó is szerepelhet.

Ezt a feladatot gyakran használják az autonóm vezetés során olyan dolgok észlelésére, mint a gyalogosok, az útjelző táblák és a közlekedési lámpák. Egyéb alkalmazások közé tartozik például az objektumok számlálása a képekben, vagy a képkeresés.

Ebben a példában a határoló dobozok értékei két koordináta párral vannak megadva.



```
[ {
  "score": 0.9988,
  "label": "zebra",
  "box": {
    "xmin": 360,
    "ymin": 250,
    "xmax": 399,
    "ymax": 312
  }
}, {
  "score": 0.9592,
  "label": "giraffe",
  "box": {
    [...]
```

22. ábra Objektumfelismerés²¹

²¹ <https://huggingface.co/facebook/detr-resnet-50> (2023.11.04.)

3.5 Képszegmentálás

A képszegmentálás vagyis image segmentation megoldással a cél az, hogy egy képet több értelmezhető részre bontsunk.

A képszegmentáció során egy algoritmus azonosítja a kép különböző jellemzőit, mint például színt, textúrát, intenzitást és ezek alapján csoportosítja a pixeleket. A szegmentáció módszerei széles skálán mozognak az egyszerű színalapú technikáktól a bonyolultabb, komplex formák felismeréséig.

A szegmentációnak több típusa is van, az egyik kategóriája az úgy nevezett szemantikus szegmentáció, amit a 23. ábra bal szélső képe szemléltet, ebben az esetben minden pixel egy adott osztályhoz tartozik legyen az akár háttér, akár személy. Ezenkívül egy adott osztályhoz tartozó összes pixel ugyanazzal a színnel van ábrázolva fekete háttérrel és rózsaszín személyekkel.

A 23. ábra középső képén látható kimenet a példányszegmentációs algoritmus eredménye. Ezen a képen is hozzá lett rendelve minden pixelhez egy osztály, azonban az azonos osztályhoz tartozó személyek eltérő színűek, tehát elkülönülnek egymástól.[7]

A 23. ábra jobb szélső képe úgy nevezett panoptikus szegmentációval készült. Ez az előző kettőhöz képest egy újabb technika, és gyakran hivatkoznak rá úgy, hogy a szemantikus és a példányszegmentációs megoldások kombinációja. Minden objektum azonosságát megjósolja elkülönítve a képen szereplő összes objektum példányát.[8]

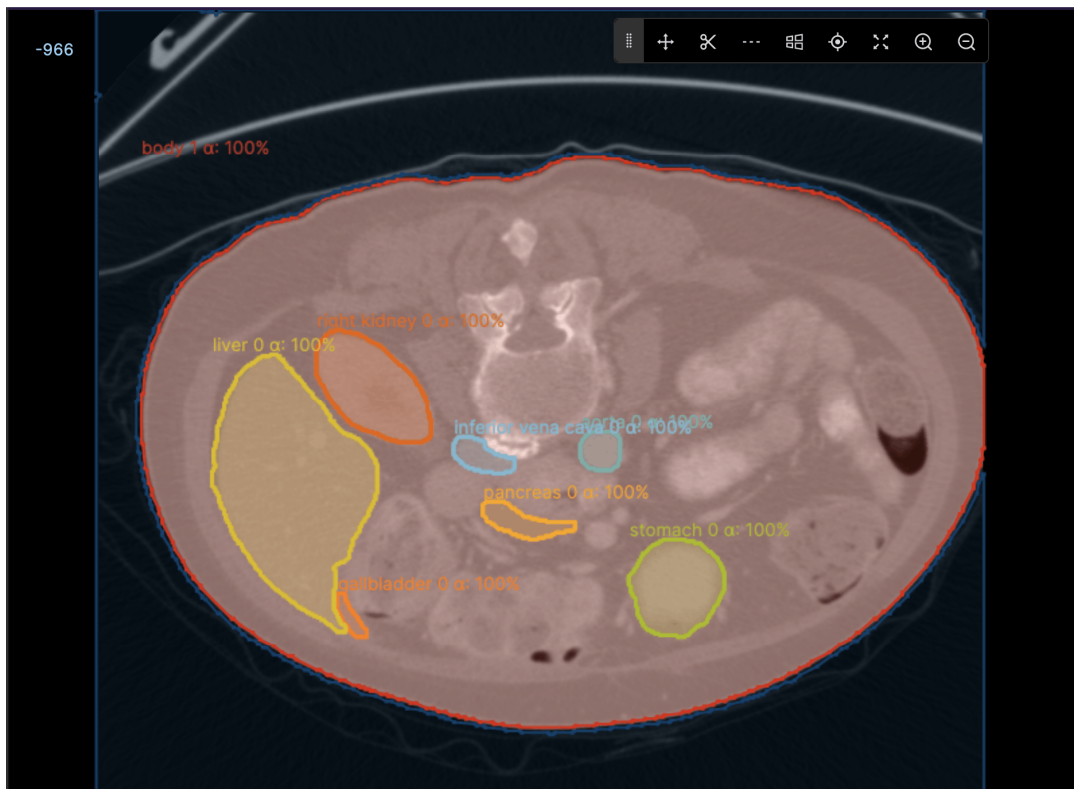


23. ábra - Szegmentáció típusai²²

A képszegmentációs eljárás egyik alkalmazási területe az orvoslás. A nem-invazív diagnosztikai képalkotás kimenetének feldolgozását könnyíti meg. Ez gyorsabb, mint ha egy ember csinálná mindezt, ezen felül, ha megfelelő a modell, akkor akár nagyobb pontossággal is dolgozhat, mint egy személy. Ebben az esetben a részletekre való odafigyelés kritikus fontosságú.

Az orvosi területen alkalmazott képszegmentációra az 24. ábra mutat egy példát. Ez a kép a panoptikus szegmentációs eljárással került feldolgozásra.

²²<https://mindy-support.com/news-post/what-is-image-segmentation-the-basics-and-key-techniques/>
(2023.11.03.)



ENCORD

24. ábra Szegmentáció az orvoslásban²³

A képszegmentációs neurális hálók széles körben van létjogosultságuk. Alkalmazható orvoslásban, önvezető autók kapcsán, fényképszerkesztés során például Photoshop-ban és ezeken felül még megannyi területen.

3.6 Képből-képet

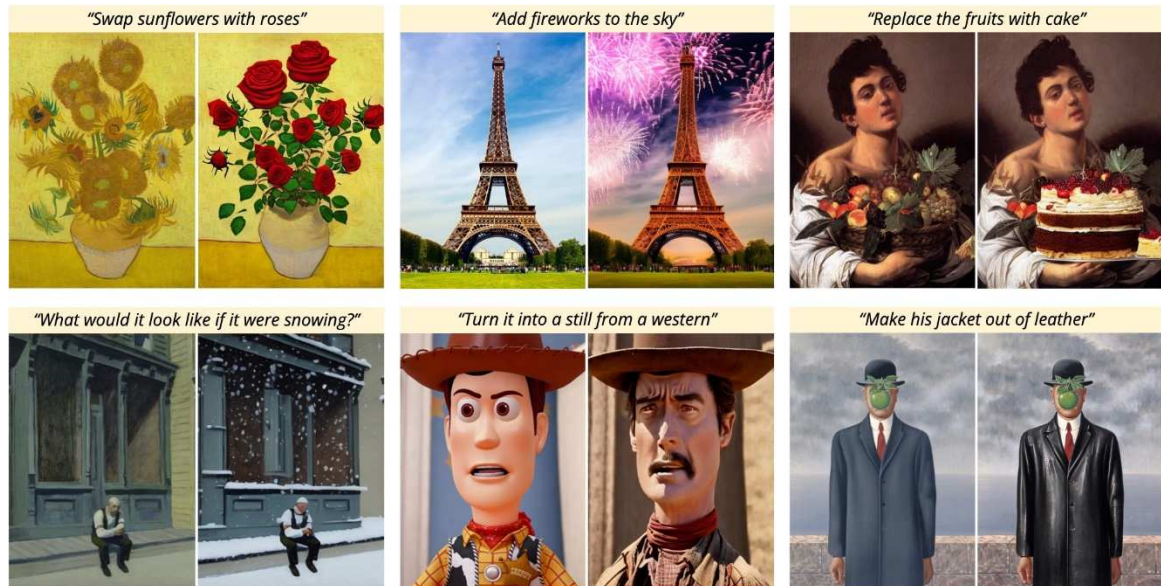
A képből-képet, vagy Image-to-Image gépi látási technológia az előzőekhez képest jelentősen különbözik, hogy ennek a neurális hálónak a bemenete és a kimenete is egy-egy kép.

Ez a neurális háló képes egy adott bemeneti képet egy másik, kívánt formátumú kimeneti képpé átalakítani, úgy, hogy az eredeti képet módosítja, vagy akár egy teljesen új képet

²³ <https://encord.com/blog/medical-image-segmentation/> (2023.11.04.)

generál. Ezek a hálózatok képesek lenyűgöző eredményeket elérni, mint például a valósághű képek generálása, a művészeti stílusok átvitele, vagy akár a képek részletgazdagabbá tétele.

A 25. ábra egy olyan példát szemléltet, amelyben van egy bemeneti kép, illetve egy szöveges útmutatás, ami szerint átformálja a képet.



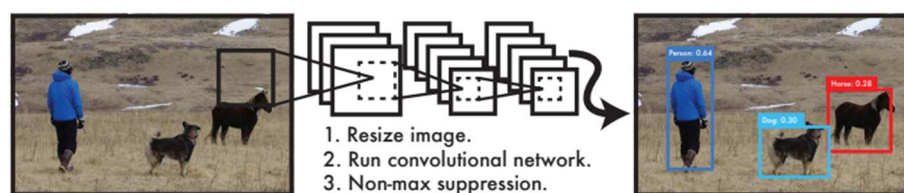
25. ábra Képből-képet minták²⁴

²⁴ <https://huggingface.co/timbrooks/instruct-pix2pix> (2023.11.04.)

4 ALGORITMUSOK

4.1 YOLO Algoritmus [41]

A YOLO algoritmus egy mozaikszó, ami a You Only Look Once szavaknak a kezdőbetűiből áll össze. Ennek az algoritmusnak több fajtája, verziója is létezik. Ahogy haladt előre az idő mindig finomítottak rajta. Ezt az algoritmust objektumfelismerési feladatokra használják, oly módon, hogy a felismert objektumok köré keretet tesz.

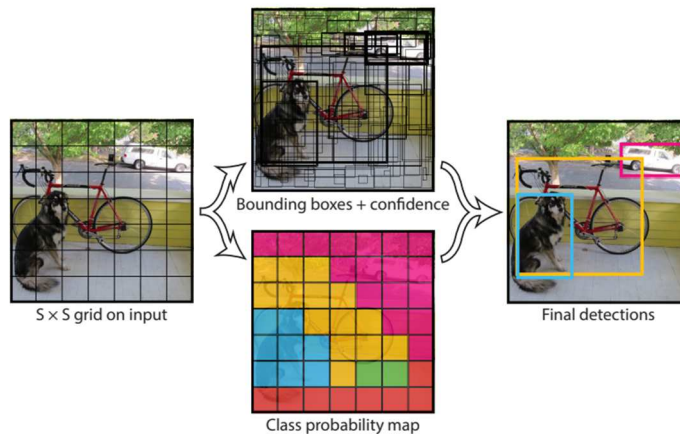


26. ábra YOLO algoritmus működése²⁵

A YOLO algoritmus egyszerre tud észlelni több objektumot is egy képen, mindezt úgy, hogy a bemeneti képet egy iterációban vizsgálja végig, és ebből is ered a neve.

A YOLO algoritmus a bemeneti képet egy $S \times S$ méretű rácstra osztja fel. Ha egy az algoritmus által ismert objektum közepe egy rácscellába esik, akkor ez a rácscella lesz felelős az objektum észleléséért. Minden rácscella előre jelzi az objektum keretét és ezekhez a négyzetekhez tartozó valószínűségi pontszámokat. Ezek a valószínűségi pontszámok azt tükrözik, hogy a modell mennyire biztos abban, hogy a keret tartalmaz egy objektumot, és mennyire tartja pontosnak a megjósolt objektum osztályát, amit a kerettel határol körbe. A YOLO algoritmus kimenete tartalmazza az objektumok keretét, osztályát és valószínűségét. A keret meghatározza az objektum helyzetét és méretét is a képen. Az osztályazonosító mutatja meg az objektum típusát, mint például autó, bicikli vagy kutya. A valószínűség megmutatja, mennyire biztos az algoritmus az objektum jelenlétében és osztályában. [11]

²⁵ <https://arxiv.org/pdf/1506.02640.pdf> (2023.11.14.)



A YOLO több száz objektumosztályt is képes lehet felismerni. Az algoritmus konvolúciós neurális hálózatot (CNN) használ a jellemzők felismerésére. A CNN rétegei különböző szintű jellemzőket ismernek fel, például széleket, alakokat és textúrákat. A YOLO a tanulási folyamat során megtanulja, milyen tulajdonságok jellemzik az egyes osztályokat. Az algoritmus tanításához nagy mennyiségű felcímkézett adatra van szükség. A tanulás során az algoritmus folyamatosan finomítja az előrejelzéseit. Az eredmények pontosságát mérhetjük a valódi pozitívok és falszpozitív értékek arányával.

A YOLO gyorsaságát azzal éri el, hogy az összes objektum határolás vagy is jóslást egyetlen hálózati áthaladással végzi. Ez lehetővé teszi a valós idejű objektumfelismerést.

²⁷ <https://arxiv.org/pdf/1506.02640.pdf> (2023.11.14.)

még videón is. A YOLO-t gyakran használják biztonsági rendszerekben, önvezető autókban, robotikában és egyéb automatizált megfigyelőrendszerekben. [12] Az algoritmus hatékony a különböző méretű és alakú objektumok felismerésében is.

A YOLO-nak számos verziója létezik mára. Míg a YOLO modellek teljesítménye az idő múlásával javult, érdemes megjegyezni, hogy gyakran a sebesség és pontosság kiegyensúlyozását helyezik előtérbe, nem pedig kizárólag a pontosságot. Ez a kompromisszum elengedhetetlen a YOLO keretrendszerhez, amely lehetővé teszi a valós idejű objektumészlelést a különböző alkalmazásokban.[13]

Version	Date	Anchor	Framework	Backbone	AP (%)
YOLO	2015	No	Darknet	Darknet24	63.4
YOLOv2	2016	Yes	Darknet	Darknet24	63.4
YOLOv3	2018	Yes	Darknet	Darknet53	36.2
YOLOv4	2020	Yes	Darknet	CSPDarknet53	43.5
YOLOv5	2020	Yes	Pytorch	YOLOv5CSPDarknet	55.8
PP-YOLO	2020	Yes	PaddlePaddle	ResNet50-vd	45.9
Scaled-YOLOv4	2021	Yes	Pytorch	CSPDarknet	56.0
PP-YOLOv2	2021	Yes	PaddlePaddle	ResNet101-vd	50.3
YOLOR	2021	Yes	Pytorch	CSPDarknet	55.4
YOLOX	2021	No	Pytorch	YOLOXCSPDarknet	51.2
PP-YOLOE	2022	No	PaddlePaddle	CSPRepResNet	54.7
YOLOv6	2022	No	Pytorch	EfficientRep	52.5
YOLOv7	2022	No	Pytorch	YOLOv7Backbone	56.8
DAMO-YOLO	2022	No	Pytorch	MAE-NAS	50.0
YOLOv8	2023	No	Pytorch	YOLOv8CSPDarknet	53.9
YOLO-NAS	2023	No	Pytorch	NAS	52.2

29. ábra A YOLO verziói²⁸

A YOLO gyengesége azonban az, hogy kevésbé pontos kis objektumoknál vagy nagyon zsúfolt jeleneteknél. Különösen kihívást jelenthet számára a kis méretű vagy részben eltakart objektumok felismerése, mivel csak egyszer tekint át a képen, ami korlátozza a részletes elemzés lehetőségét. Ezen felül hajlamos lehet arra, hogy jobban teljesít egyes típusú objektumok felismerésében. Ez részben annak köszönhető, hogy a tanítási adatkészletek gyakran nem tartalmazznak egyenletes eloszlású vagy változatos példákat, ami torzítja az algoritmus teljesítményét különböző objektumosztályok esetén. [14]

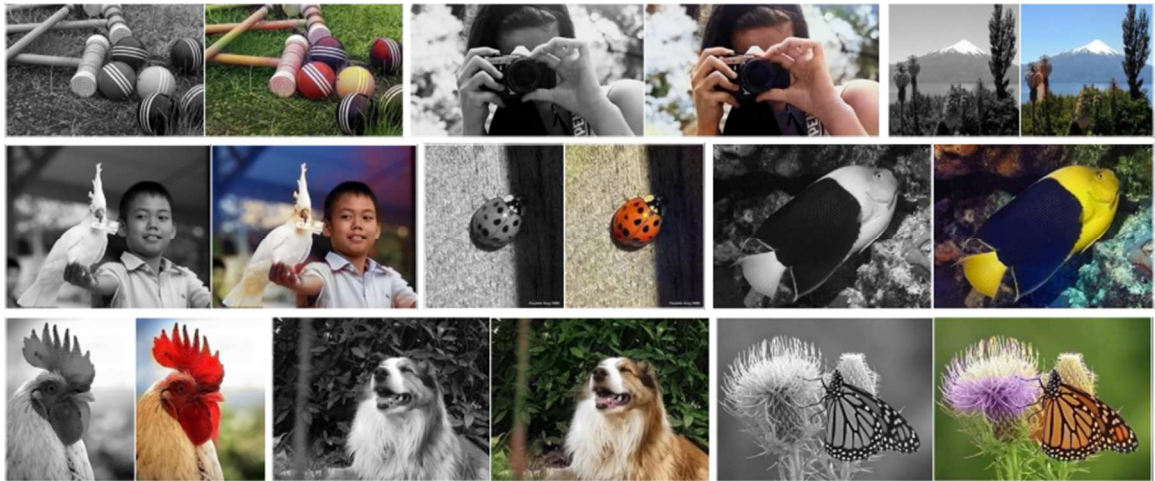
Ez az algoritmus nem használ hagyományos módszereket, mint a csúszó-ablak vagy a régió javaslat alapú megközelítés. Ez a megközelítés radikálisan eltér a korábbi objektumfelismerési módszerektől. A YOLO folyamatosan fejlődik, és az új verziók még

²⁸ <https://arxiv.org/pdf/2304.00501.pdf> Table 4 (2023.11.10.)

jobb teljesítményt nyújtanak. A YOLO megmutatta, hogy a komplex problémákat is meg lehet oldani egyszerű, mégis hatékony módszerekkel.

4.2 Colorful Image Colorization [16]

A Colorful Image Colorization egy gépi tanuláson alapuló módszer, amelyet arra terveztek, hogy színeket adjon a fekete-fehér képekhez.



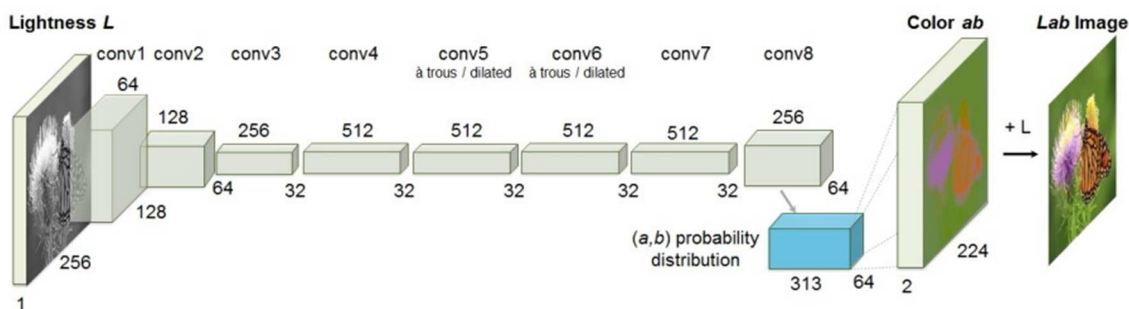
30. ábra Példa a colorful image colorization-re²⁹

A hálózathoz egy fekete-fehér kép a bemenet. Ezek a képek általában csak az intenzitás információkat tartalmazzák. Az algoritmus több konvolúciós réteget is tartalmaz, amelyek a kép különböző jellemzőit és mintázatait azonosítják. Ezek a rétegek képesek a kép különböző részeinek felismerésére. A neurális háló feltanítása során megtanulja, hogy milyen színeket rendeljen a kép különböző részeihez. Mind ezt elsajátítva színes képekből, amelyeket a feltanítás során használtak.

A Colorful image colorization volt az egyik első kísérlet a szürkeárnyaltos képek színezésére. A hálózat bemenetként egy szürkeárnyaltos képet vesz, és előrejelzi az „ab” tartománypárt, amelyek a valószínűségi eloszlást mutatják, melyek ezután a „Lab” színtér „a” és „b” csatornáivá alakítanak. A hálózat lineárisan halmozza fel a konvolúciós rétegeket, nyolc blokkot alkotva. Mindegyik blokk két vagy három konvolúciós rétegből áll, amelyet egy ReLU réteg és egy batch normalizációs[15] réteg követ. A kép méretének

²⁹ <https://arxiv.org/pdf/1603.08511.pdf> Fig1 (2023.11.11.)

csökkentésére a készletezés helyett a léptetést használják. A hálózat bemenete 256×256 , míg a kimenet 224×224 , azonban később átméretezi az eredeti képméretre a kimenetét. Az algoritmus eltávolítja az alacsony „ab” értékek hatását azáltal, hogy a veszteséget az tanítás során előforduló pixelritkaság alapján súlyozza át. A szerzők ezt a technikát class rebalancing-nak nevezték el. [16]



31. ábra A colorful image colorization hálózatmodellje³⁰

A Colorful Image Colorization nagy pontossággal képes rekonstruálni a potenciális színeket, még akkor is, ha az eredeti kép információi korlátozottak. Az eredmények gyakran meglepően valósághűek és részletesek, ami új lehetőségeket nyit meg a régi fényképek és dokumentumok újraélesztésében.

Az eljárás hasznos lehet a múzeumok és archívumok számára, ahol a történelmi fényképek digitalizálása és színesítése folyamatban van. Ez segíthet az embereknek jobban kapcsolódni a történelemhez, mivel a színes képek gyakran élettelibbek és vonzóbbak. Ezen kívül lehetőséget biztosíthat a régi filmek színesítésére is. Ez új életet lehelhet a klasszikus fekete-fehér filmekbe, megkönnyítve az újabb generációk számára a kapcsolódást.

A technológia azonban nem csak a múlt rekonstruálására használható. Alkalmazható a modern művészetben is, ahol az alkotók kísérletezhetnek a szín és forma új kombinációival. Ez új dimenziókat nyithat meg a vizuális művészetekben, ösztönözve az innovációt és a kreativitást. [17]

³⁰ <https://arxiv.org/pdf/1603.08511.pdf> Fig2 (2023.11.14.)

5 PROJEKTMUNKA

5.1 Projekt célja és motivációja

A projektemet és szakdolgozatomat szerettem volna a mesterséges intelligencia alapjára építeni. Érdekelnek az újdonságok és a technológia, így erre a témára esett a választás. Ez a témakör nagyon gyorsan fejlődik, így nehéz vele lépést tartani. A mesterséges intelligenciának első sorban nem az alkalmazása volt, ami leginkább foglalkoztatott, hanem hogy hogyan működik. Ezzel a témával régebb óta foglalkozom, és lenyűgöző azt látni, hogy milyen eredményeket lehet elérni, milyen alkalmazásai lehetnek és hogy a mindennapi életbe hogyan épül be. Számomra ez a téma rendkívül bonyolult, mert nagyon szerte ágazó, legyen szó olyan hálózatról, amelynek különböző a be és kimenete, mint például szöveg-kép konverter, képből szöveg kinyerés, vagy akár audio feldolgozás, nyelvfeldolgozás, vagy gépi látás.

A motivációm, hogy tanuljak erről a témáról, megértsem az alapjait, hogy hogyan is működik, ezen felül meglegyen az a képességem és látásmódom, amely ahhoz szükséges, hogy egy saját alkalmazást tudjak létrehozni. A környezetemben előfordulnak olyan kérdések, amelyekre egy neurális hálózat egy elképzelésem szerint elég jó megoldás lehetne.

A projektem elsődleges célja, hogy a neurális hálóknak a legalapvetőbb működését bemutassam és megtanuljam, és hogy ezeket, hogyan lehet használni a való életben. A közéletben az emberek általában úgy tekintenek a mesterséges intelligenciára, mint egy mágikus program, amely bármit meg tud oldani, holott ez közel sem igaz. Így az elsődleges célom az volt, hogy számomra ne érthetetlen feketedoboz legyen.

A dolgozatomban leírtak a neurális hálók alapjairól több fejlesztői könyvtárban implementáva vannak, mint például TensorFlow³¹, PyTorch³² vagy Keras³³, így ezek a könyvtárak lényegesen meg tudják könnyíteni egy fejlesztés folyamatát, időt spórolnak, ezen felül optimalizált kóddal rendelkeznek a maximális teljesítmény és sebesség elérése érdekében.

A dolgozatom implementációjához az ESP32 fejlesztői eszközt választottam, ezen belül is az ESP32-CAM nevű boardot.

³¹ <https://www.tensorflow.org/about> (2023.12.03.)

³² <https://pytorch.org/features/> (2023.12.03.)

³³ <https://keras.io/> (2023.12.03.)



32. ábra ESP32-Cam board³⁴

A választásom azért esett erre az eszközre, mert széles körben hozzáférhető, rendelkezik kamerával, ezen felül alkalmas vezeték nélküli TCP/IP kommunikációra.

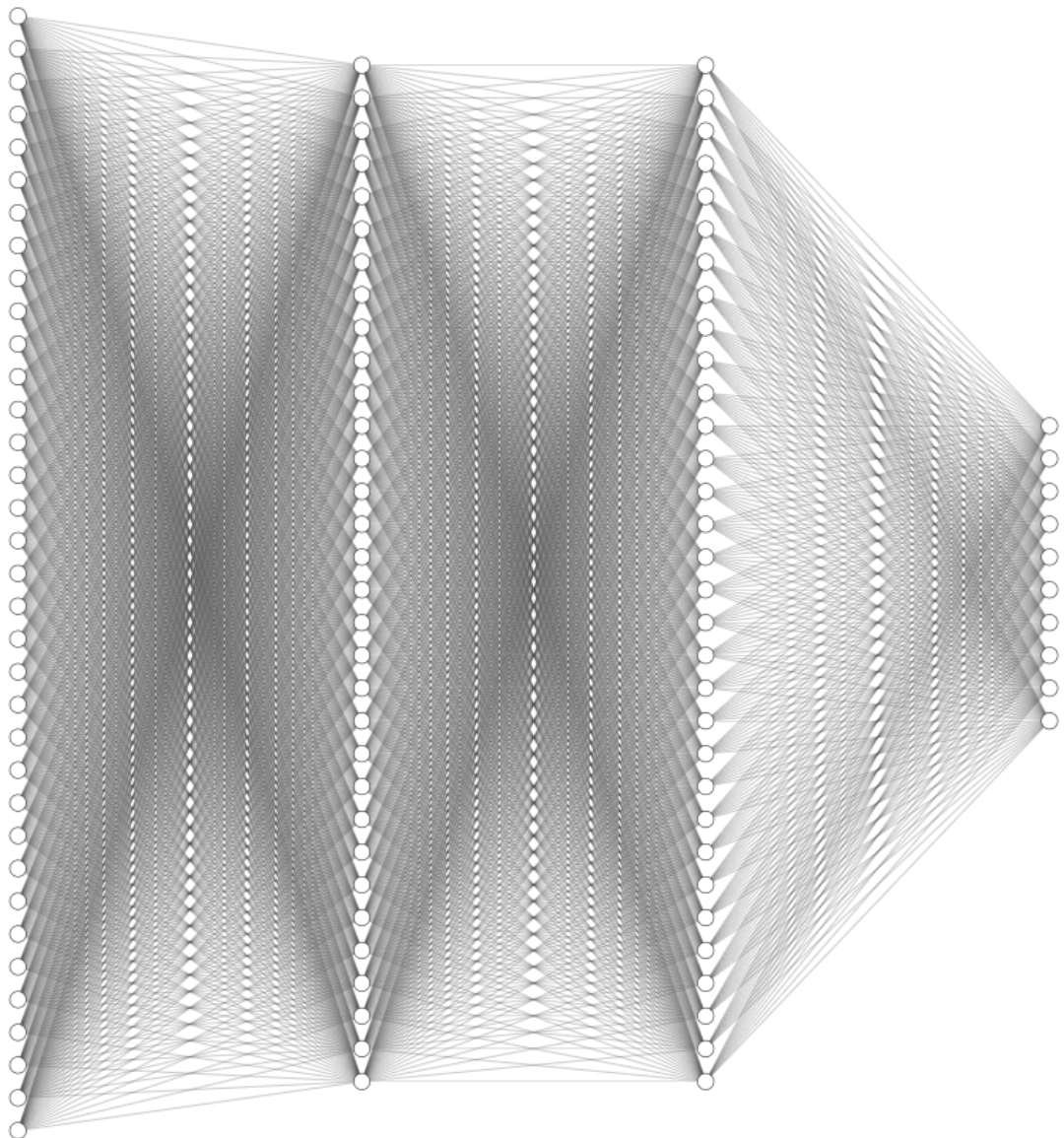
Ez a fejlesztői eszköz egy ESP32-S típusú mikroprocesszorra élül, és a működéséhez szükséges hardvert a fejlesztői eszköz nyáklapja tartalmazza a kamera csatlakozóval együtt.

5.2 A neurális háló felépítése

A megvalósított neurális háló képosztályozási feladatot képes ellátni. A háló felépítését tekintve 2 rejtett rétegből áll, ebből a második réteg kimenete már a neurális háló kimenetét valósítja meg, illetve 1 bemeneti réteggel rendelkezik. A bemeneti réteg 784 neuronból áll, amely a feltanított bemeneti képnek a pixelei, ez a kép 28x28 képpontból áll. A bemeneti réteg ezután egy sűrű réteggel van összekötve és ez a rejtett réteg 128 neuront tartalmaz. Az első és második rejtett réteg között is sűrű kapcsolat van, és a második réteg is 128 neuront tartalmaz viszont ennek a rétegnek már csak 10 kimenete van, ezzel elérve a 10 kimeneti neuront. A tanítási adathalmaz határozza meg a 10 kimenetet.

Az első és második réteg aktivációs függvénye ReLU aktivációs függvénnyel működik, míg a kimeneti rétegnek az aktivációs függvénye a Softmax függvény, amely alkalmas az osztályozási feladatok ellátására.

³⁴ <https://www.doitsmart.hu/wp-content/uploads/2020/10/ESP32CAM1.jpg> (2023.11.18.)



33. ábra Az alkalmazott neurális háló felépítése

Ezt a hálózatot próbálja szemléltetni a 33. ábra, csak a bemeneti réteg 16 szorosa a képen lévő neuronok számának míg a rejtett rétegek 4 szerese a rajzolt képnek.

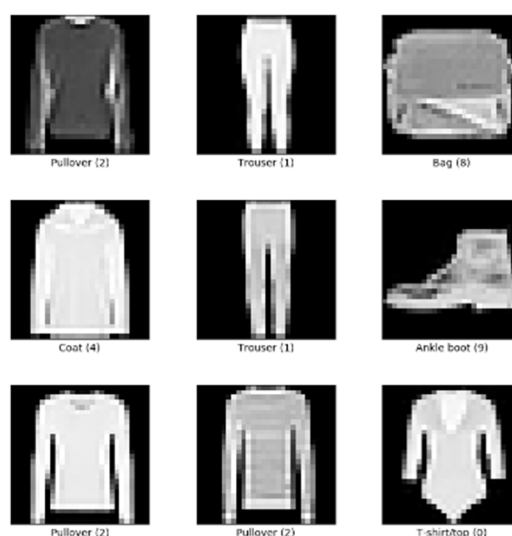
A mátrix és vektorszámítások valamennyi részét a NumPy³⁵ fejlesztői könyvtár végzi. Ennek a könyvtár használatának az előnye, hogy lényegesen képes egyszerűsíteni a kódot, például egy mátrix szorzást ennek a könyvtárnak a használatával egy sorban meg lehet valósítani, míg, ha hagyományos módszerrel közelítenénk meg, több egymásba ágyazott ciklusra is szükség lenne és lényegesen több sort foglalna el. Ezen felül a NumPy könyvtár előnye, hogy a számítások optimalizálva vannak és képes a számításokat a hagyományos

³⁵ <https://numpy.org/> (2023.11.19)

Pythonhoz képest töredék idő alatt elvégezni, mert a NumPy könyvtár számításai C nyelven vannak implementálva, és így nem a Python értelmező (interpreter) rendszerének kell feldolgoznia a számításokat. [42]

5.3 Adatszerkezet

A modell feltanításához, és teszteléséhez a Fashion Mnist³⁶ adatkészletet használtam. Ez az adatkészlet ruhadarabokról tartalmaz fekete-fehér képeket 28x28 pixeles felbontásban. Az adatkészlet tartalmaz feltanításra és tesztelésre is szeparált képeket, a feltanítás adatkészlete 60 000 képet tartalmaz míg a tesztelési adat 10 000 képből áll. Ezeket a képeket 0 és 9 között indexelték, így elérve, a 10 képcsoportot, ami azt jelenti, hogy a tanítási adatkészlet minden kategóriából 6000 képet, míg a tesztelési adatkészlet minden kategóriából 1000 képet tartalmaz.



34. ábra A fashion mnist adatkészletből példák³⁷

Az adatszerkezet 2 fő mappában lett felépítve, ebből az egyik a „train” mappa, amit a tanításra használtam fel, a másik a „test” mappa, amit a tesztelés során alkalmaztam. Mind a két fő mappa tartalmaz 10-10 almappát, ezek 0-tól 9-ig számozva, és ezek alatt vannak kategóriánként bontva a képek, így megcímkézve azokat.

³⁶ <https://github.com/zalandoresearch/fashion-mnist> (2023.11.19)

³⁷ https://www.tensorflow.org/datasets/catalog/fashion_mnist (2023.11.19)

Fashion mnist adat címkéi: [32]

- 0 T-shirt/top
- 1 Trouser
- 2 Pullover
- 3 Dress
- 4 Coat
- 5 Sandal
- 6 Shirt
- 7 Sneaker
- 8 Bag
- 9 Ankle boot

5.4 Hiperparaméterek

A modell a már említett [784, 128, 128, 10] méretű rétegekből épül fel. A veszteség számításához a kategorizált kereszt entrópia veszteségi függvényt használok. Optimalizáláshoz az Adam optimalizációs algoritmust használtam, amelynek a tanulási rátája 0.001, epsilon $1e^{-7}$ és a decay= $1e^{-3}$ értékekkel állítottam be. Az alábbi kód felel az optimalizáló és hiperparamétereinek beállításáért.

```
optimizer=Optimizer_Adam(learning_rate=0.001, epsilon=1e-7, decay=1e-3)
```

A hálózat kimenetének pontosság megállapításához kategorizált veszteség számítást alkalmazok, ez a számítási mód a „one-hot” kimenet értékének a pontosságát számítja, ami azt jelenti, hogy csak egy olyan kimenete lehet a neurális hálózatnak, ami igaz a bemenetekre, mivel egy ruhadarab nem tartozhat több csoportba is. A hiba számításának implementációja:

```
y = np.argmax(y, axis=1)

return predictions == y
```

A modell tanítását 10 iterációban végeztem el. A neurális háló súlyai és torzításai alapértékeit a NumPy random number függvényével hoztam létre.

A feltanítás menete és kimeneti értékei:

```
epoch: 1
training, 468, accuracy: 0.755, loss: 0.658
validation, accuracy: 0.822, loss: 0.490

epoch: 2
training, 468, accuracy: 0.843, loss: 0.430
validation, accuracy: 0.843, loss: 0.430

[...]

epoch: 9
training, 468, accuracy: 0.892, loss: 0.298
validation, accuracy: 0.873, loss: 0.357

epoch: 10
training, 468, accuracy: 0.894, loss: 0.291
validation, accuracy: 0.874, loss: 0.354
```

A feltanítás végére a modell pontossága 0,894 lett és a veszteség 0,291. Ezeknél az értékeknél talán el lehetne érni kicsit jobbat is, de ez a modell már egész nagy pontossággal képes kategorizálni, és az értékek ránézésre azt sugallják, hogy nem történt túltanulás (overfitting).

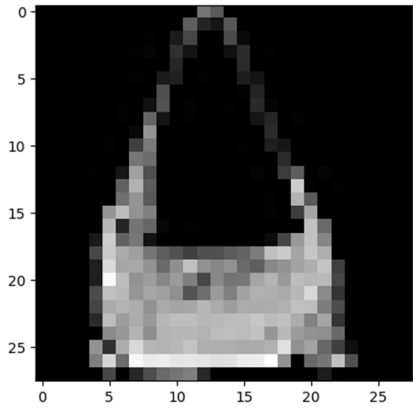
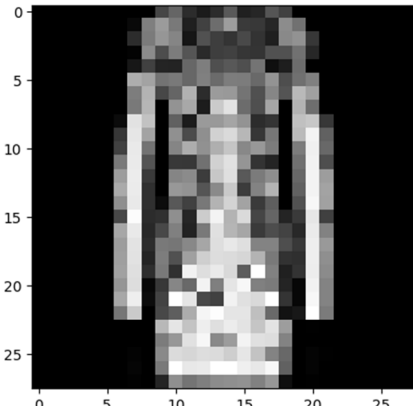
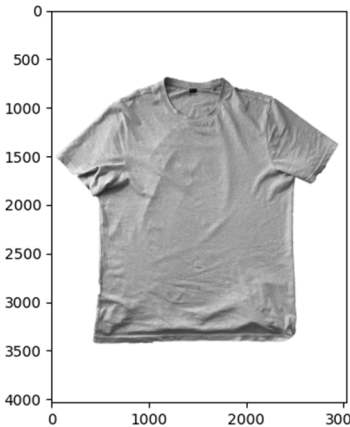
A validációs képet lefuttatva a neurális hálózaton a pontosság 0,874 a veszteség pedig 0,354. Ezeket a képet a feltanítás során nem látta a neurális háló, ezeket tartalmazza a fashion mnist adathalmaz a „test” könyvtárban. Ezekből az értékekből következtethető, hogy a feltanított modell jól megállja a helyét.

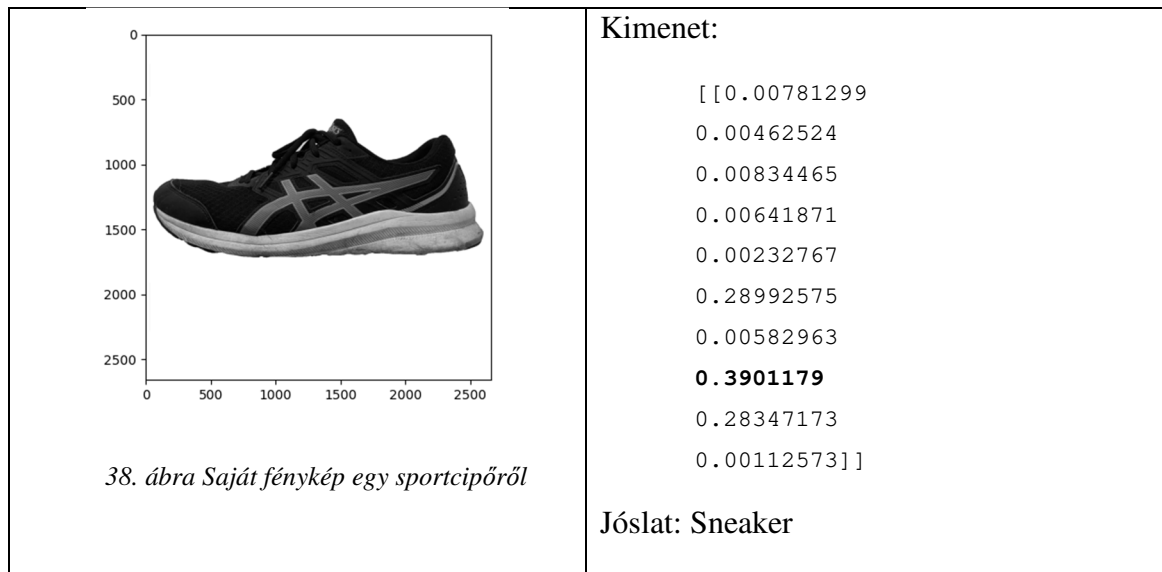
A feltanítás utána lementettem a modellt, ez magában foglalja a rétegeket, a rétegek méreteit és az összes neuronhoz tartozó súly és torzítási értékeket. Ez után a modellt később bármikor be lehet tölteni és használni aszerint, ahogyan fel lett tanítva.

5.5 Tesztelés

A modell teszteléséhez néhány képet kiválasztottam az adatszerkezet „test” mappájából, illetve készítettem saját képeket is. A saját képet fel kellett dolgozni, hogy megegyezzen azzal a felépítéssel, mint amilyen képekkel fel lett tanítva a neurális háló, mint a méret és az invertált színek.

A kimenet a 10 osztály valószínűségi értékét mutatja, a jóslat pedig a legmagasabb érték osztályának megnevezését.

 35. ábra Fashionon mnis test/8/0045.png	Kimenet: <pre>[[1.8556994e-04 1.0551260e-07 5.4129509e-06 5.0859013e-05 2.4382207e-05 6.3962402e-04 3.1598931e-05 9.4581374e-09 9.9906248e-01 8.4869069e-13]]</pre> Jóslat: Bag
 36. ábra Fashionon mnis test/3/0073.png	Kimenet: <pre>[[3.6209187e-04 4.6286779e-05 7.9244506e-03 9.7174287e-01 1.4402252e-02 2.3868954e-06 4.7325683e-03 1.2468708e-09 7.8716234e-04 7.8740960e-09]]</pre> Jóslat: Dress
 37. ábra Saját fénykép egy pólóról	Kimenet: <pre>[[1.46810085e-01 7.33598252e-04 1.19723864e-01 1.42342057e-02 2.18570307e-02 8.54802784e-04 3.61188561e-01 8.35374294e-07 3.34596902e-01 5.19114636e-08]]</pre> Jóslat: Shirt



5.6 Megvalósítás

A projekt megvalósítását úgy oldottam meg, hogy az ESP-32 eszközön futtattam egy szolgáltatást, ami a kamerának a képét osztotta meg 240x240 pixeles méretben, illetve az ESP csatlakozott WiFi-n a helyi hálózatra, így lehetővé téve azt, hogy a kameraképet el lehessen érni számítógépről is.

A kapott kép feldolgozását egy asztali számítógép végzi. A számítógép csatlakozik az ESP-n futó kameraszerverhez, és ez alapján dolgozza fel a képet és adja át a feltanított neurális hálózathoz.

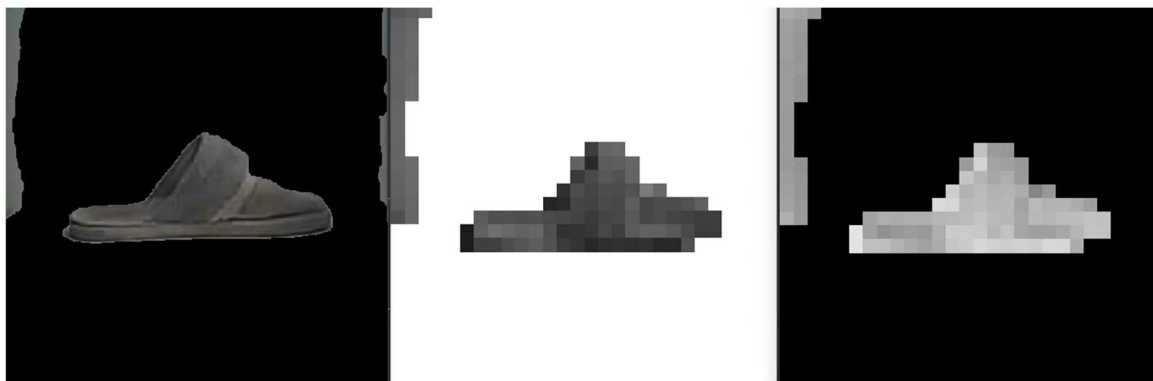
A kapott képnek először maszkolja a háttérét, erre azért van szükség, mert a tanítási adathalmazban a képeknek monokróm fekete háttérük van és így lehet a legjobb eredményt elérni vele, ha nincs a képnek háttér.



39. ábra Stream kép maszkolása

A 39. ábra bal szélső képe a kapott kameraképet mutatja, majd a második képen látható a maszk, amit alkalmazunk és a harmadik képen a kimeneti kép, amin már nincsen háttér.

Következő lépésben átméretezi 240x240 pixelről 28x28 pixel méretűre, átalakítja szürkeárnyaltosra, majd invertálja a kép színeit. Ennek feladata, hogy megegyezzen a kép a tanítási adathalmaz paramétereivel, mivel a tanítás során csak 28x28 pixel méretű és invertált színű képeket használtam.



40. ábra Stream kép feldolgozása

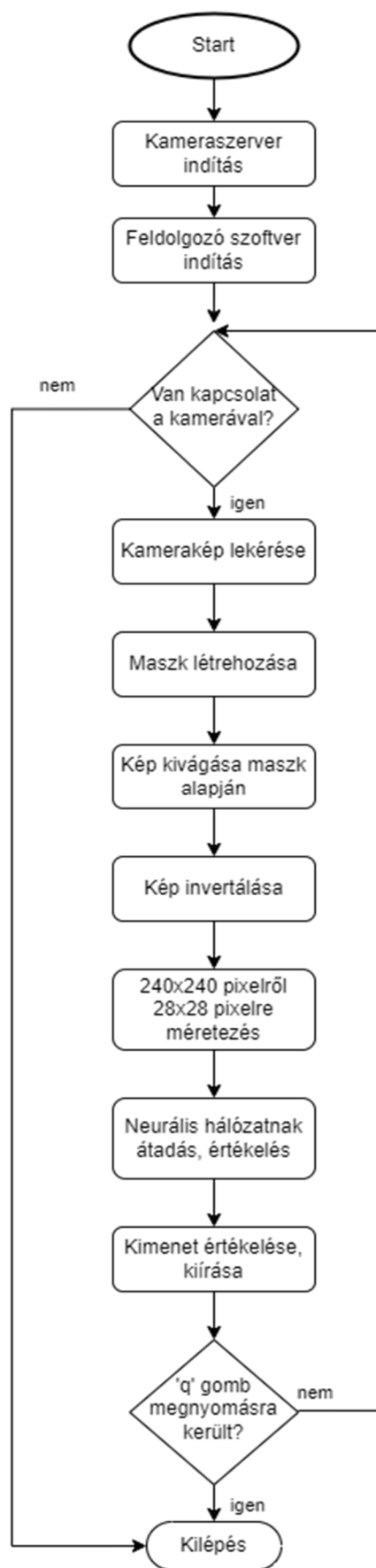
A 40. ábra első képe a már maszkolt kép, a második képen az átméretezett és szürkeárnyaltosra alakított kép látható és az utolsó képen az invertált kép, ami a neurális háló bemenete lesz.

A kapott bemeneti képet a neurális hálón végig futtatva kapok egy jóslatot, hogy mi lehet a képen.

```
prediction: Sandal 0.97654223
prediction: Sandal 0.9508196
prediction: Sandal 0.9636909
prediction: Sandal 0.9667176
prediction: Sandal 0.9638773
prediction: Sandal 0.96731377
```

41. ábra Neurális háló kimenete

Ezt a folyamatot ciklikusan végzi a számítógép képkockáról-képkockára addig, amíg meg nem szakítom a futtatást a 'q' gomb megnyomásával.



42. ábra Projektmunka folyamatábrája

5.7 Fejlesztési lehetőségek

Egy ilyen egyszerűbb modell, neurális háló rengeteg fejlesztési lehetőséget hordoz magával.

Az első és talán legszembetűnőbb fejlesztési lehetőség, hogy 28x28 pixel felbontású képek helyett, valamivel nagyobb felbontású képeket használjon, így sokkal pontosabb becsléseket és felismeréseket lehetne elérni vele.

A képméret növelésén felül, ha a színes képet használná a neurális háló azzal is el lehetne érni előrelépést a felismerés pontossága tekintetében.

A tanításhoz használt adathalmaz bővítése, növelés is elősegítené a pontosságát a modellnek, ebbe beleértve, hogy saját képekkel feltanítani a modellt.

Egy fejlesztés lehetne még, hogy dedikált hardver végezze a számításokat, így készíthető lenne egy eszköz, ami magában foglalja a teljes kategorizálási algoritmust és adatmodellt. Erre alkalmas lehet egy jelenleg használt egyszerűbb neurális hálózat feldolgozására ez az ESP is.

Továbbá érdemes lenne ötvözni ezt a felismerési módot, egy olyan módszerrel, ami a képet előre feldolgozza és csak az éleket emeli ki, majd ezek után az eredeti színes képet feldolgozni és ezzel a technikával is jobb hatásfokot lehetne elérni.

6 ÖSSZEFOGLALÁS

A dolgozatomban igyekeztem bemutatni a gépi látáshoz vezető utat. Ez magában foglalja a neurális hálózatok felépítését az alapköveitől. Rendkívül sok elemből épül fel egy mesterséges neurális hálózat, így nehéz bemutatni. Azokat az elemeket, amiket bemutattam, sokkal bővebben is ki lehetne fejteni, akár úgy is, hogy egy önálló tudományos publikáció legyen belőle, ezért nehéz összefoglalni ezeket. Ezen kívül bizonyos témaköröknek még megannyi megoldása van, például aktivációs függvényekből is már sok száz létezik, vagy akár veszteség számításnak is rengeteg módja van. Ahogy fejlődik egyre több és korszerűbb megoldások jönnek létre, így az, hogy „ezek az alapkövei” egyáltalán nem jelenti azt, hogy ez nem fog a jövőben megváltozni. A neurális hálózat alapjainak bemutatása után a gépi látás témakört, és ennek alkalmazási lehetőségeit, illetve konkrét példákkal szemléltettem. Ezek a konkrét példák nagyon jól be tudják mutatni, hogy egy neurális hálózatban milyen lehetőségek rejlenek. A neurális hálókat, típusait és alkalmazásait ennél sokkal több egységre is lehet bontani, illetve más megközelítés szerint is csoportosíthatóak. A hardveres gyorsítás is egy széles témakör, hogy milyen elektronika megoldásokat találnak arra, hogy még hatékonyabban történjenek a számítások, illetve a gyártók száma és ezen belül a termékeik száma is folyamatosan nő, amit szándékoltan mesterséges intelligencia alkalmazására szánunk.

Szinte minden az alkalmazott algoritmuson dől el, hogy mennyire állja meg a helyét az adott neurális háló. Az algoritmus foglalja magában az összes hiperparamétert, a hálózat teljes felépítését, és az összes építőelemét egy neurális hálózatnak. Ezen felül a feltanítás módját, az ott alkalmazott technikákat és még megannyi variációt és lehetőséget hordoz magában. Ezek az algoritmusok már rendkívül bonyolultak és nagy méretű hálózatokat is képesek megvalósítani. Az sem ritka, hogy egy neurális hálózatnak akár milliós nagyságrendű súly és torzítás érték darabszáma legyen.

Összességében ezt a témakört feldolgozni kihívást jelent számomra a megannyi megoldás és variáció miatt, de ettől a kihívástól lesz érdekes. Számomra ez a témakör, a neurális hálótól kezdve, a meglévő alkalmazásokig nagyon izgalmas és bőven látok benne fantáziát. Szeretném bővíteni a tudásomat ebben a témakörben és még jobban elmélyülni benne, illetve saját neurális hálókat feltanítani és fejleszteni, illetve ezt a projektmunkámat továbbfejleszteni.

7 SUMMARY

In my thesis I present how a neural network is built up for computer vision including the foundations of a neural network. It is hard to present every part of a neural network because it could be built up from many elements. The elements which I have presented could be explained in much more detail, even such a way that it becomes an independent scientific paper and that is why it is difficult to summarize them. In addition there are still many solutions for activation functions and loss calculations. As it develops, more and more modern solutions are created, so the fact that "these are the foundations" does not mean at all that this will not change in the future.

After presenting the basics of a neural network, I have demonstrated how the computer vision works and its application possibilities with specific examples. These examples could well demonstrate the possibilities of a neural network and its application methods. Neural networks, their types and applications could be divided into many more parts, and they could also be grouped according to other approaches.

The hardware acceleration for neural networks is also a broad topic, what kind of electronic solutions are found to make calculations even more faster and efficient. This topic is also constantly developing, because the number of manufacturers and, within that, the number of their products, which are intended for the use of artificial intelligence, is constantly growing.

Everything depends on the algorithm used, how well a given neural network holds up. The algorithm includes all hyperparameters, the entire structure of the network, and all the building blocks of it. In addition, it includes the method of training the network, and many more variations and possibilities what could lead a more functioning and efficient neural network. These algorithms are extremely complex and can implement large neural networks. It is also not uncommon for a neural network to have weights and biases values in the order of millions.

Overall this whole topic was a challenge for me to work, because of the many solutions and variations, but this challenge makes it interesting. For me, this topic, from neural networks to existing applications, is very exciting and I see plenty of imagination in it. I would like to expand my knowledge in this topic and learn the gist of it to build my own neural networks, as well as further develop my project work.

8 DEFINÍCIÓK ÉS RÖVIDÍTÉSEK

Az ebben a dokumentumban használt rövidítések, mozaik szavak definíciója

NÉV	DEFINÍCIÓ
ESP32	Espressif32, Espressif Systems által kifejlesztett fejlesztői eszköz. 32 bites mikrokontroller vezeték nélküli hálózattal és Bluetooth-al.
AI	Artificial Intelligence
MI	Mesterséges intelligencia
LIDAR	Light Detection and Ranging, lézer alapú távolság érzékelés
CNN	Convolutional Neural Network, Konvolúciós neurális háló
RELU	Rectified Linear Units
MSE	Mean Squared Error
GPU	Graphics Processing Units
TPU	Tensor Processing Units
CPU	Central Processing Units

9 IRODALOMJEGYZÉK

- [1] Mihály Radványi, Zóra Solymár, Attila Stubendek, Kristóf Karacs, „Mobile Banknote Recognition”, doi/10.1145/2093698.2093883, (2023. 11. 16.)
- [2] Mihály Radványi, Balázs Varga, Kristóf Karacs, „Advanced Crosswalk Detection for the Bionic Eyeglass”, DOI: 10.1109/CNNA.2010.5430281, (2023. 11. 15.)
- [3] D. O. Hebb. The Organization of Behavior: A neuropsychological theory. New York: Wiley, 1949, (2023. 11. 05.)
- [4] „Neural network”, https://en.wikipedia.org/wiki/Neural_network (2023. 11. 05.)
- [5] „Backpropagation”, <https://en.wikipedia.org/wiki/Backpropagation> (2023. 11. 05.)
- [6] <https://www.autodesk.com/solutions/photogrammetry-software> „Autodesk® ReCap™ What is photogrammetry?” (2023. 11. 05.)
- [7] <https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python> „Different Types of Image Segmentation” (2023. 11. 07.)
- [8] <https://mindy-support.com/news-post/what-is-image-segmentation-the-basics-and-key-techniques/> „What is Image Segmentation” (2023. 11. 07.)
- [9] „Activation function”
https://en.wikipedia.org/wiki/Activation_function#Table_of_activation_functions (2023. 11. 07.)
- [10] Strahinja Zivkovic, „PyTorch – Popular techniques to prevent the Overfitting in a Neural Networks”, <https://datahacker.rs/018-pytorch-popular-techniques-to-prevent-the-overfitting-in-a-neural-networks/> (2023. 11. 08.)
- [11] Rohit Kundu, „YOLO: Algorithm for Object Detection Explained”
<https://www.v7labs.com/blog/yolo-object-detection> (2023. 11. 08.)
- [12] Karpuram Dhanalakshmi Srivani, „YOLO Algorithm for Custom Object Detection”, <https://www.analyticsvidhya.com/blog/2022/06/yolo-algorithm-for-custom-object-detection/> (2023. 11. 10.)
- [13] Juan R. Terven, Diana M. Cordova-Esparaza, „A COMPREHENSIVE REVIEW OF YOLO: FROM YOLOV1 AND BEYOND”,
<https://arxiv.org/pdf/2304.00501.pdf> (2023. 11. 10.)

- [14] „Review of YOLO: drawback and improvement from v1 to v3, ”https://sheng-fang.github.io/2020-04-25-review_yolo/ (2023. 11. 10.)
- [15] Sergey Ioffe, Christian Szegedy, „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”
<https://arxiv.org/pdf/1502.03167.pdf> (2023. 11. 11.)
- [16] Richard Zhang, Phillip Isola, Alexei A. Efros, „Colorful Image Colorization”,
<https://arxiv.org/pdf/1603.08511.pdf> (2023. 11. 11.)
- [17] Jantic, „DeOldify”,
<https://github.com/jantic/DeOldify/blob/master/README.md>, (2023. 11. 14.)
- [18] G. Linan, S. Espejo, R. Dominguez-Castro és A. Rodriguez-Vazquez, „ACE4k: ananalog I/O 64x64 visual microprocessor chip with 7-bit analog accuracy,” (2023. 11. 14.)
- [19] A. Rodriguez-Vazquez, G. Linan-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galan, F. Jimenez-Garrido, R. Dominguez-Castro és S. E. Meana, „ACE16k: the third generation of mixed-signal SIMD-CNN ACE chips toward VSoCs,” (2023. 11. 14.)
- [20] P. Földesy, Á. Zarándy, C. Rekeczky és T. Roska, „Digital implementation of cellular sensor-computers,” (2023. 11. 14.)
- [21] NVIDIA Corporation ©, „About CUDA”, <https://developer.nvidia.com/about-cuda> (2023. 11. 14.)
- [22] OpenCV team, „About CUDA”, <https://opencv.org/about/> (2023. 11. 14.)
- [23] Daniel Strigl, Klaus Kofler and Stefan Podlipnig, „Performance and Scalability of GPU-based Convolutional Neural Networks”, doi:10.1109/pdp.2010.43 (2023. 11. 14.)
- [24] Danny Hernandez, Tom B. Brown, „Measuring the Algorithmic Efficiency of Neural Networks”, arXiv:2005.04305 [cs.LG], (2023. 11. 14.)
- [25] Aleksandar Botev, Guy Lever, David Barber, „Nesterov's Accelerated Gradient and Momentum as approximations to Regularised Update Descent”, arXiv:1607.01981, (2023. 11. 15.)
- [26] Sebastian Ruder, „An overview of gradient descent optimization algorithms”, arXiv:1609.04747, (2023. 11. 15.)
- [27] Yanli Liu, Yuan Gao, Wotao Yin, „An Improved Analysis of Stochastic Gradient Descent with Momentum” arXiv:2007.07989, (2023. 11. 15.)

- [28] Aleksandar Botev, Guy Lever and David Barber, „Nesterov’s Accelerated Gradient and Momentum as approximations to Regularised Update Descent”, arXiv:1607.01981, (2023. 11. 16.)
- [29] Rachel Ward, Xiaoxia Wu, Leon Bottou, „AdaGrad stepsizes: Sharp convergence over nonconvex landscapes”, arXiv:1806.01811, (2023. 11. 16.)
- [30] Diederik P. Kingma, Jimmy Ba, „Adam: A Method for Stochastic Optimization”, arXiv:1412.6980, (2023. 11. 16.)
- [31] Yu Emma Wang, Gu-Yeon Wei, David Brooks, „Benchmarking TPU, GPU, and CPU Platforms for Deep Learning”, arXiv:1907.10701 (2023. 11. 16.)
- [32] Zalando SE, <https://github.com/zalandoresearch/fashion-mnist> (2023. 11. 19.)
- [33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei, „ImageNet Large Scale Visual Recognition Challenge” arXiv:1409.0575 (2023.11.21.)
- [34] Alex Krizhevsky , Ilya Sutskever, Geoffrey E. Hinton, „ImageNet Classification with Deep Convolutional Neural Networks”, doi:10.1145/3065386 (2023.11.21.)
- [35] Xu, Z., Zhang, H., Wang, Y. *et al.* $L_{1/2}$ regularization. *Sci. China Inf. Sci.* **53**, 1159–1169 (2010). <https://doi.org/10.1007/s11432-010-0090-0> (2023.11.23.)
- [36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting” (2023.11.23.)
- [37] Shaeke Salman and Xiuwen Liu, „Overfitting Mechanism and Avoidance in Deep Neural Networks”, arXiv:1901.06566 (2023.11.23.)
- [38] Jun Qi, Jun Du, Sabato Marco Siniscalchi, Xiaoli Ma, Chin-Hui Lee, „On Mean Absolute Error for Deep Neural Network Based Vector-to-Vector Regression”, arXiv:2008.07281 (2023.11.30.)
- [39] Timothy O. Hodson, Thomas M. Over, Sydney S. Foks, „Mean Squared Error, Deconstructed”, <https://doi.org/10.1029/2021MS002681> (2023.11.30.)
- [40] Anqi Mao, Mehryar Mohri, Yutao Zhong, „Cross-Entropy Loss Functions: Theoretical Analysis and Applications”, arXiv:2304.07288 (2023.11.30.)
- [41] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, „You Only Look Once: Unified, Real-Time Object Detection”, arXiv:1506.02640 (2023.11.14)

- [42] Shiva Verma, „How Fast Numpy Really is and Why?”,
<https://towardsdatascience.com/how-fast-numpy-really-is-e9111df44347>
(2023.12.14)