ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Alba Regia Műszaki Kar
Természettudományi és Szoftvertechnológiai Intézet

SZAKDOLGOZAT

Magyarország halfajainak osztályozása mély tanulással

OE-AMK

2024

Hallgató neve: Spinda Máté János

Hallgató törzskönyvi száma: T001051/FI12904/A-M

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Alba Regia Műszaki Kar
Természettudományi és Szoftvertechnológiai Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: Spinda Máté János
Szakdolgozat száma: SZD2301230936122091[redacted]
Törzskönyvi száma: T001051/FI12904/A-M **Neptun kódja:** [redacted]

Szak: mérnökinformatikus
Specializáció: Felhő szolgáltatási technológiák és IT biztonság specializáció

A dolgozat címe: Magyarország halfajainak osztályozása mély tanulással
A dolgozat címe angolul: Classification of Fish Species in Hungary by Deep Learning
A feladat részletezése: A hallgató feladata mély tanuláson alapuló osztályozó modell készítése Magyarország gyakori halfajainak osztályozásához. A hallgató ismertesse a mély tanulás elméleti alapjait, valamint mutassa be a használt eszközöket. Végezze el a feladathoz szükséges adatok begyűjtését és azok előkészítésének folyamatát, a megvalósított mély neurális háló(k) struktúráját, a tanítási folyamatot, a modellel kapott eredményeket, valamint értékelje a megvalósított modellek teljesítményét. Készítsen vizualizációt az eredmények bemutatásához.

Intézményi konzulens neve: Piglerné Dr. Lakner Rozália

A kiadott téma elévülési határideje:2026. 02. 10.
Beadási határidő: 2023. 12. 15.

A szakdolgozat: Nem titkos.

Kiadva: Székesfehérvár, 2023. 10. 17. P.H.

ÓBUDAI EGYETEM Alba Regia Műszaki Kar Természettudományi és Szoftvertechnológiai Intézet Székesfehérvár

.....
Intézetigazgató

A dolgozatot beadásra alkalmasnak találom: 2023. 12. 15.
2024. 05. 15.

.....
belső konzulens

.....
külső konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Alba Regia Műszaki Kar
Természettudományi és Szoftvertechnológiai Intézet

HALLGATÓI NYILATKOZAT

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

Kelt: Lovasberény *(hely)*, 2024.05.09 *(dátum)*

.....
hallgató aláírása

Tartalomjegyzék

1. BEVEZETÉS

Szakdolgozatomban Magyarországon fellelhető halfajok egy részének azonosítására készítek egy mély neurális hálózatot, amely különböző halfajokról készített képek osztályozására képes.

Mivel feladatom a mesterséges intelligencia egy napjainkban virágzó területét, a gépi tanulást, azon belül felügyelt tanulást és osztályozást, s annak egyik módszerét, a neurális hálózatok ismeretét igénylő probléma, dolgozatom irodalmi összefoglalójában bemutatom ezen területek alapvető fogalmait, azon belül nagy hangsúlyt fektetve az osztályozási feladatok megvalósítására.

Részletesen ismertetem a neurális hálókkal és azon belül a mély neurális hálókkal történő osztályozást, a modellek tanításának folyamatát, ezen belül kiemelt hangsúlyt fektetve a feladatmegoldásom során is használt konvolúciós neurális hálózatok és annak rétegeinek működésére. Bemutatom a modellek teljesítményének mérési lehetőségeit, valamint a modellek értékeléséhez használt különböző metrikákat.

Az osztályozó rendszer megvalósítása során bemutatom a képek beszerzésének és előkészítésének lépéseit, valamint a feladat megoldása során elkészített több ismert és általam készített neurális hálózat architektúrát, és értékelem azok teljesítményét az általam több helyről gyűjtött adathalmazon, majd összevetem a különböző modellekkel kapott eredményeket.

Dolgozatom lezárásaként beszélek a modell lehetséges továbbfejlesztéséről, és egy esetleges telefonos alkalmazás elkészítésének lehetőségéről.

2. MESTERSÉGES INTELLIGENCIA, GÉPI TANULÁS

Napjainkban a mesterséges intelligenciának [1] meghatározó szerepe van, életünk minden területén találkozhatunk vele. Néhány ilyen terület például az orvostudomány, ahol az adatok és képek elemzésével betegségek diagnosztizálásában segít. Találkozhatunk még vele vállalatirányítási rendszereknél, különböző trendek előrejelzéséhez, és akár az autóiparban is, valamint amivel már mindenki találkozhatott, a közösségi média platformok ajánló algoritmusai is mesterséges intelligencia módszereket használnak.

A gépi tanulás a mesterséges intelligencia egyik részterülete, amely során közvetlen utasítások nélkül, matematikai adatmodellek segítségével tanítjuk be a számítógépeket. [2]

A gépi tanulás több részfeladatra osztható: ezek a felügyelt, nem felügyelt és megerősítéses tanulás. Felügyelt tanulás esetén rendelkezésünkre áll a kiinduló adathalmaz, amely tartalmazza az adatminták tulajdonságait és az elvárt célértékeket is. A feladat az, hogy ezek alapján létrehozzunk egy modellt, ami előre nem ismert mintákon is képes helyes eredményre jutni (helyes célértéket előállítani). Ilyen például egy céges problémakezelő rendszer, ami a megfelelő osztályra irányítja a beérkező problémákat, ami az eddigi, manuális átirányítások mintáiból megtanult modell alapján történik. [3]

Nem felügyelt tanulás esetén szintén rendelkezésre áll egy kiindulási adathalmaz, azonban az elvárt értékeket nem ismerjük. Itt a tanítás célja különböző összefüggések és minták azonosítása lesz. Ilyen például a képek és videók analízisében az objektumfelismerés, vagy az ajánlórendszerek, amelyek hasonló szokásokkal rendelkező felhasználók csoportosítását használják tartalomajánlásokhoz.

Az egyik talán legfelkapottabb terület napjainkban a generatív AI, aminek a feladata új tartalmak készítése felhasználói utasításra. Ilyen például az OpenAI generatív nyelvi modellje, a ChatGPT is. [4]

2.1 Felügyelt tanulás

A felügyelt tanulás lényege, hogy rendelkezésünkre áll a bemeneti adathalmaz tulajdonságai, és a hozzá elvárt kimenet (célattribútum) is. A tanulás során egy modellt készítünk,

amely a tulajdonságok ismeretében megjósolja a célattribútum értékeket. A modell jóságát az elvárt és a predikált kimenetek összehasonlításával mérhetjük. Ehhez a bemeneti adathalmazt két részre osztjuk, amelyeket egyrészt a modell tanításához (tanító halmaz), másrészt az ellenőrzéshez (teszt halmaz) használunk. A tanulás célja az, hogy a modell képes legyen felismerni eddig számára ismeretlen példákat is.

Az elvárt kimenet alapján a felügyelt tanulásnak két típusát különböztetjük meg. Amennyiben a célattribútum folytonos vagy számszerű, akkor regresszióról beszélünk, amennyiben diszkrét osztályokba sorolható, azaz a feladat az, hogy a modell döntse el, hogy a példa melyik osztályba tartozik, osztályozásról (klasszifikációról) beszélünk.

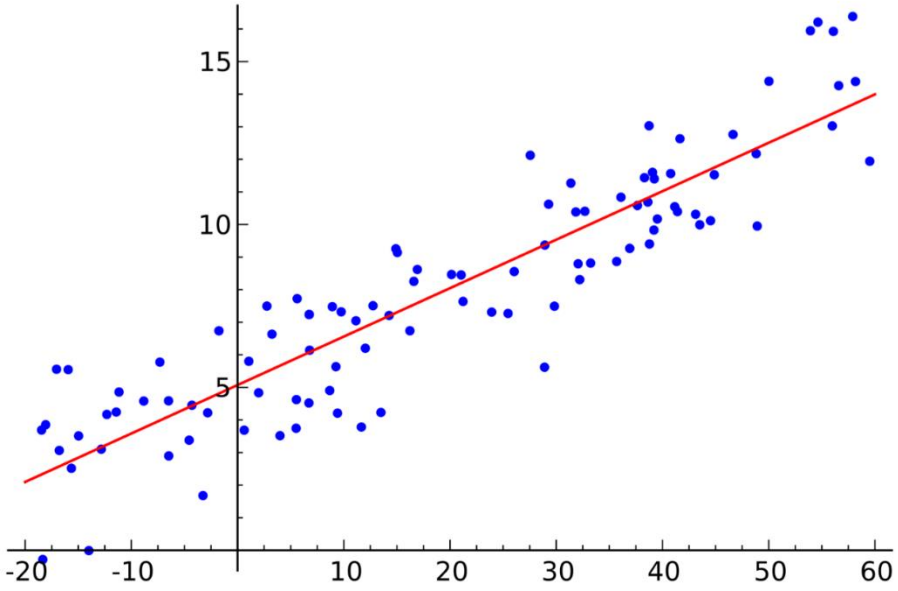
Felügyelt tanulás esetében a feladatunk az, hogy egy olyan modellt határozzunk meg, ami képes a tanító adathalmaz alapján egy számára ismeretlen minta célattribútumának értékét a lehető legnagyobb pontossággal meghatározni. Ebben az esetben is a már ismert tanító és teszt halmazokra fogjuk az rendelkezésünkre álló mintákat bontani. A felügyelt tanulás megvalósítását két fő lépésre bonthatjuk.

1. Tanítás: A mintahalmazunk pontosan fel van címkézve, tehát minden mintáról pontosan tudjuk, hogy milyen a célattribútum értéke. Ez alapján a tanító algoritmus lefuttatásával a megtanítjuk a modell paramétereket.
2. Tesztelés: Tesztelés során a már tanult modellnek mutatunk mintákat, és azt vizsgáljuk, hogy milyen pontossággal tudja a hozzá tartozó célattribútumok értékét meghatározni.

Fontos megjegyezni azt, hogy a tanítás és a tesztelés során nem használhatjuk egyszerre ugyan azokat a mintákat, tehát egy tanulási kör alatt, nem tartalmazhatja mindkét halmaz ugyan azt a bemenő adatot. Egy gyakran használt módszer felügyelt tanulás esetén a k-szoros keresztvalidáció. Ilyenkor az eredeti adathalmazt k (lehetőség szerint egyenlő méretű) részre osztjuk, majd ugyanennyiszer lefuttatjuk a tanítást és tesztelést. Minden tanítási körben az egyik halmazunk lesz a kiértékelő adathalmaz, a maradék $k-1$ pedig a tanító. Így ki tudjuk használni a teljes rendelkezésünkre álló adathalmazt, úgy, hogy egy tanítási lépésben nem jelent meg mindkét helyen ugyanaz a minta.

2.1.1 Regresszió

A felügyelt gépi tanulás regressziós módszerei hasonlítanak a statisztikában használt regresszió analízishez, a fő különbség viszont az, hogy most nem csak jellemezni szeretnénk az adathalmazt, hanem a lehető legpontosabban megbecsülni ismeretlen minták értékeit az eredeti adathalmaz alapján készült regressziós modell segítségével. Ennél a módszernél is tanulásra van szükség a modell elkészítéséhez, hiszen a becslés a tanító adathalmazban felismert minták alapján történik.



2.1. ábra Lineáris regresszió [3]

A 2.1. ábra a legegyszerűbb regressziós modellre, a lineáris regresszióra mutat példát, ahol a cél egy az adatokra legjobban illeszkedő lineáris függvény megtanulása. Az x tengelyen az adathalmaz egy folytonos tulajdonsága, az y tengelyen pedig a célattribútum értéke látható. Az ábrán pirossal látható $f(x)$ egyenes jelöli a tanult modellt, amelynek segítségével tetszőleges adathoz meghatározhatóvá válik a becsült célattribútum érték.

A bemutatott módszer általánosításaként több független input változó értéke alapján is készíthető becslés a célattribútum értékére, ebben az esetben többdimenziós regresszióról beszélünk. További általánosítás lehet a lineáris modell helyett más a független változók

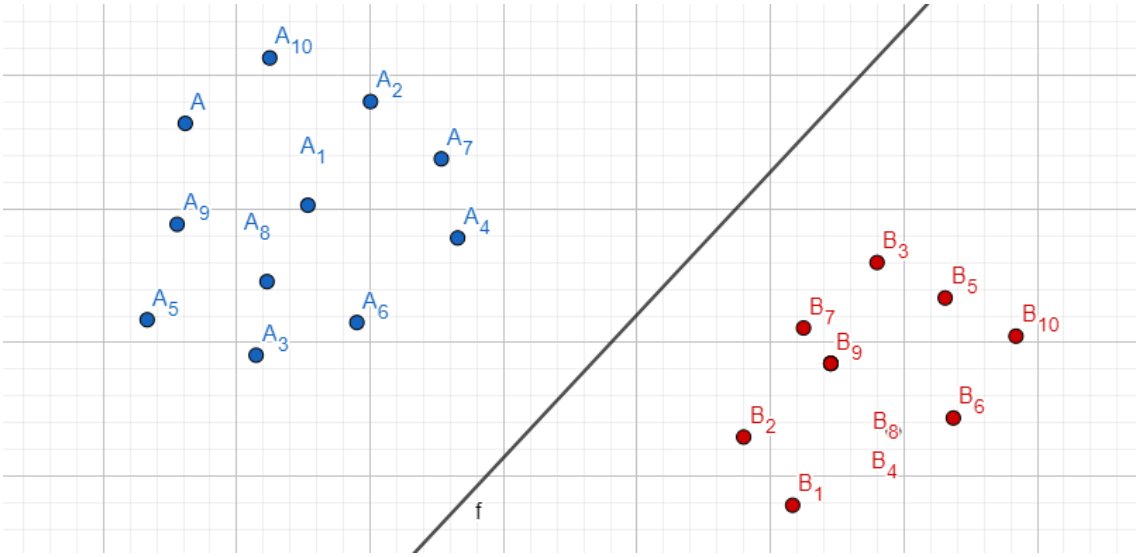
nemlineáris kombinációjával készített függvényekkel (például hatvány, exponenciális, logaritmikus függvények) történő becslés használata.

2.1.2 Osztályozás

A felügyelt tanulás osztályozási módszerei számos különböző technikájú modell használatán alapulnak. [2] A modellek közös vonása, hogy a kategorikus osztálycímkék előrejelzéséhez a célattribútum értékét más attribútumok függvényeként állítják elő. Ilyen módszerek többek között a döntési fák, a Bayes osztályozási módszerek, a legközelebbi szomszéd osztályozók, a tartóvektor gépek, a neurális hálók vagy az együttes módszerek. A továbbiakban ezek közül mutatok be röviden néhányat.

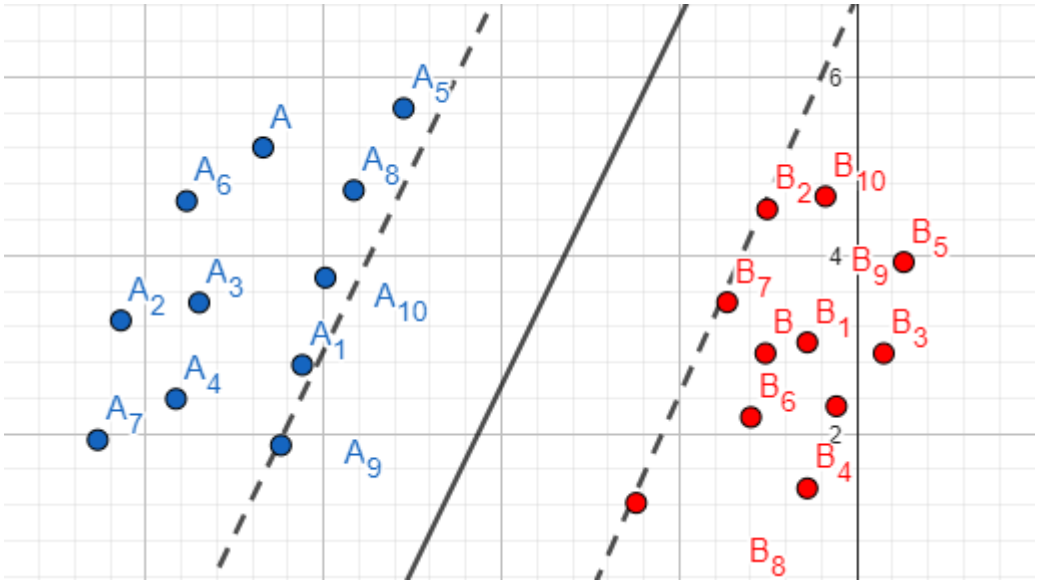
Tartóvektor gépek

A tartóvektor gépek (Support Vector Machines, SVM) egy széles körben elterjedt módszer az osztályozási feladatok megoldására, amely az osztályok közötti döntési határt tanulópéldákra illeszkedő vektorokkal (tartóvektorokkal) modellezi. A módszer célja olyan döntési határ (hipersík) keresése, amely minél jobban elválasztja az adatokat. Ez legegyszerűbb esetben két lineárisan jól szeparálható halmaz elválasztása egy egyenessel (többdimenziós esetben egy hipersíkkal), amelyet a 2.2. ábra szemléltet. [2] [5]



2.2. ábra Lineáris szeparáció

Két halmaz szeparáláshoz végtelen sok f hipersík lehetséges. Ennek a módszernek a feladata meghatározni a lehető legideálisabb hipersíkot, ehhez margókat lehet használni, amelyek a döntési határral párhuzamos, egymástól legmesszebb elhelyezkedő hipersíkokat jelentik, amelyek „legjobban” szétválasztják a két osztályba tartozó adatokat. A nagy margóval rendelkező hipersíkok azért jobbak, mert ebben az esetben a modell kevésbé hajlamos túlilleszkedésre.



2.3. ábra SVM példa

A 2.3. ábrálható esetben, használhatunk szigorú margókat, ami azt jelenti, hogy a margó és az f hipersík között nem lehet a hipersík ellenkező oldalán levő osztály eleme.

Abban az esetben, ha a lineáris szétválasztás nem alkalmazható, egyrészt különböző kernel függvények használatával, másrészt az adatok magasabb dimenziójú térbe történő transzformálása utáni lineáris szeparációval oldható meg a feladat.

Véletlen erdők

A véletlen erdő modellek az együttes módszerek közé tartoznak és a döntési fákon alapszanak. Egy modell több egyszerű döntési fából áll, így a bemenet a belső fák bemenetének az összessége lesz. Minden egyes fa önálló modellként alkot egy osztályozási eredményt, a modellek összesítése pedig egy többségi szavazattal határozza meg a

célattribútum értékét. A megszavazott eredmény mellett, rendelkezésünkre állnak a kimeneten a bemeneti változók fontossági értékei is. Ezek azt jelentik, hogy egyes bemenetek milyen mértékben befolyásolták a kimenet képzését. Tanítás során a bemeneti adatsorokból helyenként elhagy egyet-egyet az algoritmus, ezekkel tudja tesztelni a saját pontosságát. [2]

2.1.3 Multimédia tartalmak osztályozása

A multimédia tartalmak nagy mértékben eltérnek az előzőekben bemutatott osztályozási technológiák bemenetéül szolgáló alfanumerikus adatoktól. Ezeket két fő csoportra bonthatjuk:

1. Statikus: Statikus tartalmak alatt értjük, a nem időfüggő adatokat, ide tartoznak a képek és a szövegek is.
2. Dinamikus: Dinamikusnak tekintjük, ha az adat időfüggő, azaz az értelmezéséhez elengedhetetlen tulajdonsága az, hogy az adat melyik részéről melyik időpontban veszünk mintát. Ide tartoznak a teljesség igénye nélkül a videók és a különböző hangfelvételek.

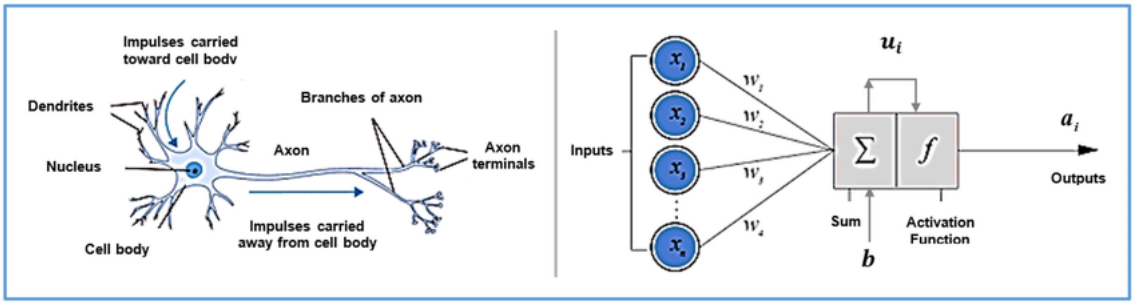
Ezek az adatok sokkal komplexebbek, mint amit az előbbiekben tárgyalt rendszerekkel hatékonyan osztályozni tudunk. Ezek osztályozására az egyik leggyakrabban használt és elfogadott módszer különböző típusú neurális hálózatok használata, melyet részletesebben a következő fejezetben ismertetek.

2.2 Neurális hálók

Neurális hálózatnak nevezzük azt, a párhuzamos, elosztott működésre képes információfeldolgozó eszközt, ami azonos vagy hasonló típusú, lokális feldolgozást végző műveleti elemek, rendezett topológiában összekapcsolt rendszeréből áll. Rendelkezik tanulási algoritmussal, és a megtanult információ felhasználását lehetővé tevő előhívási algoritmussal. [6] [7]

2.2.1 A perceptron modell

A legegyszerűbb neurális háló az egyetlen neuronból álló perceptron modell, amelyet 1957-ben Frank Rosenblatt dolgozott ki. A modell felépítése hasonlít a biológiai neuronokéhoz, amelyet a 2.4. ábra mutat be. A biológiai neuron inputjait, azaz a dendriteket tekinthetjük a mesterséges neuron több bemenetének, az axon pedig megfeleltethető egy kimenetnek. A neuron működése során amennyiben a bemenetek súlyozott összege elér egy meghatározott értékel, azaz küszöböt, a neuron aktiválódik, és a kimenetén megjelenik egy érték. A legegyszerűbb esetben a modell bináris, hiszen vagy aktivált állapotban van a neuron, és „1” van a kimeneten, vagy nincs, tehát a kimenet „0”.



2.4. ábra Biological neuron versus McCulloch and Pitts' artificial neuron model [8]

Fentiek alapján legyen a bemenet egy ismert m-dimenziós vektor

$$x(n) = (x_1(n), \dots, x_m(n))^T$$

ahol, jelölje az n-edik időpontbéli m számú bemenetet, feltéve, hogy minden $n = 1,2, \dots, N$ időpontban érkezik bemenő jel. $w_1, \dots, w_m$ jelölje a súlyokat, amelyek a modell tervezési paraméterei, a tanítás során ezek kiszámítása a célunk. $w_1(n), \dots, w_m(n)$ jelölje az n-edik időpontban a súly aktuális értékét. Ezeket tekinthetjük a valós súlyok n-edik közelítésének. [6] Ebből adódóan

$w = (w_1, \dots, w_m)$ a valós súlyvektor,

$w(n) = (w_1(n), \dots, w_m(n))^T$ pedig a közelítő.

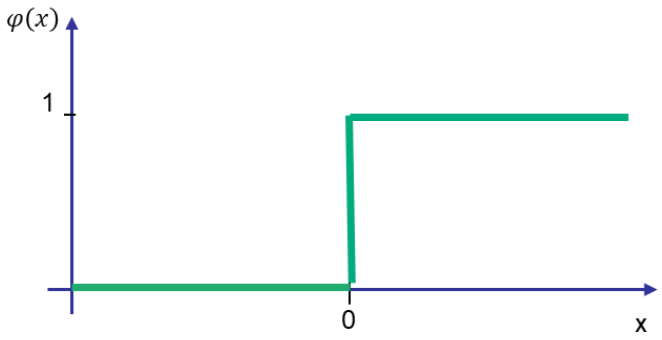
Az összegző csomópontnak van még egy b bemenete is, ez a bias vagyis torzítás. Ez szintén egy modell paraméter, tehát a tanítás során a feladatunk ennek meghatározása is. Ezt a súlyokhoz hasonlóan csak közelíteni tudjuk, ennek jelölése $b(n)$.

Az összegző csomópont nevéből adódóan a fentiek súlyozott összegét képezi az alábbi módon: legyen a csomópont kimenete adott n időpillanatban $v(n)$, ekkor:

$$v(n) = b(n) + \sum_{i=1}^{m} w_i(n)x_i(n)$$

A kimenet képzéséért az aktivációs függvény, másnéven a transzfer függvény felelős. Ez az összegzett eredményt alakítja át, egy a feladat megoldásához specifikus intervallumba. A transzfer függvényt jelöljük φ-vel. Ezáltal a neuron kimenete az adott időpillanatban $y(n) = \varphi(v(n))$. Általában monoton növekvő, jobbról folytonos függvényeket használunk. Ezek határértéke $-\infty$ -ben 0, $+\infty$ -ben pedig 1. Néhány aktivációs függvény:

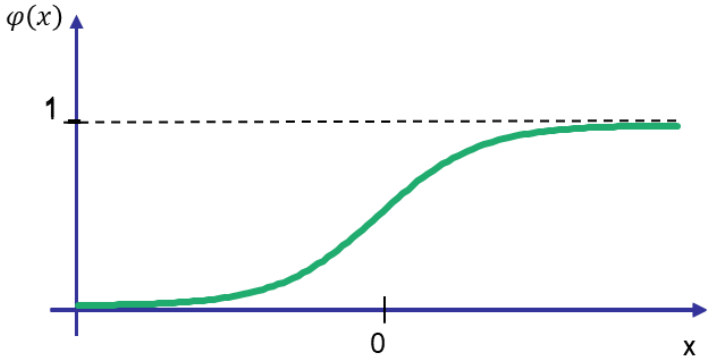
- Küszöb függvény: Ezt említettem a Rosenblatt perceptron bemutatásánál, amennyiben egy adott küszöb érték alatt van x, a kimenet 0, egyébként 1.
  $\varphi(x) = \begin{cases} 1, & ha\ x \geq 0, \\ 0, & ha\ x < 0. \end{cases}$



2.5. ábra Küszöb függvény

- Logisztikus függvény:
  $\varphi(x) = \frac{1}{1 + \exp(-\alpha x)}, \qquad x \in \mathbb{R}$

A küszöb függvény hátránya, hogy nem deriválható, ezért helyette sokszor valamilyen másik függvényhez is fordulhatunk. Ilyen a logisztikus függvény (vagy szigmoid függvény) is, ami közelítőleg hasonló alakú, s amelynek paramétere az $\alpha > 0$ konstans.

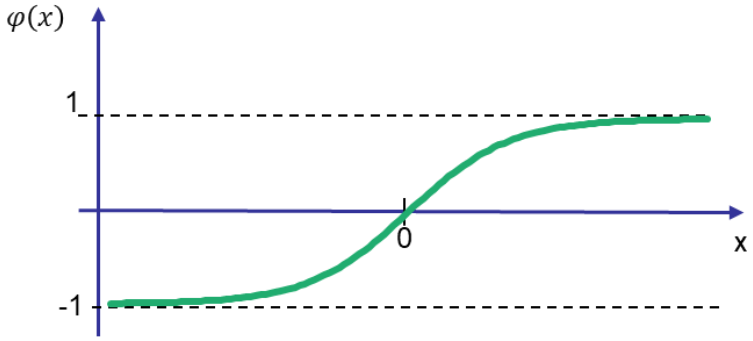


2.6. ábra Logisztikus függvény

- Tangens hiperbolikus függvény:

$$\varphi(x) = \frac{2}{1 + \exp(-2x)} - 1 = \tanh(x)$$

Tekinthetjük a logisztikus függvény módosításának, ahol a határérték $-\infty$ -ben -1 -re változik.

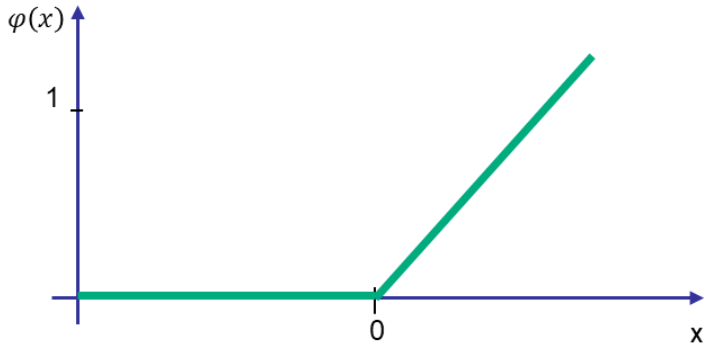


2.7. ábra Tangens hiperbolikus függvény

- ReLU (rectified linear unit) függvény:

$$\varphi(x) = \begin{cases} x, & ha\ x \geq 0, \\ 0, & ha\ x < 0. \end{cases}$$

A ReLU aktivációs függvénnyel egy olyan folyamat modellezhető, ahol csak a pozitív bemenetek képesek átjutni, a negatív bemenetek pedig 0-ként jelennek meg.

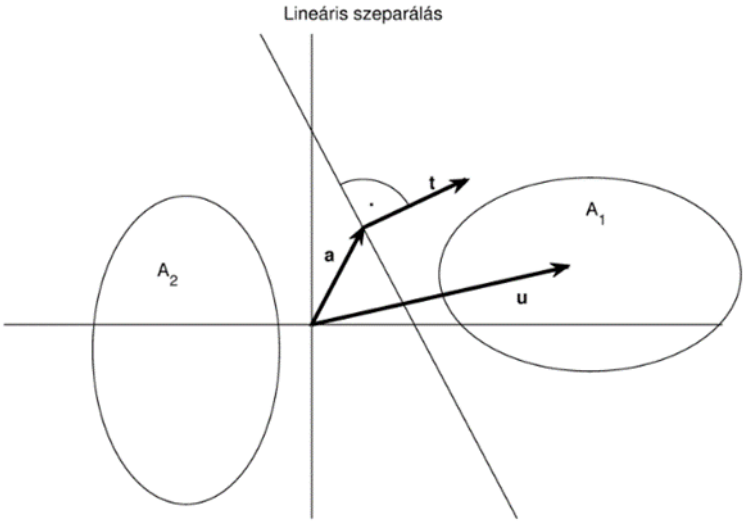


2.8. ábra ReLU aktivációs függvény

2.2.2 A perceptron tanítása

Rosenblatt a modell paramétereinek meghatározásához megadott egy tanító módszert is. Ezt hívjuk „perceptron tanulási szabálynak”, amelynek lényege a következő. Vegyünk egy két csoportból álló A_1 és A_2 adathalmazt, a két csoport legyen lineárisan elválasztható (lásd 2.9. ábra). A perceptron kimenetén a csoporttól függően -1 és 1 értékeket várunk. Ehhez szükségünk van egy iteratív algoritmusra, ami akár többször is végigmegy a tanító adathalmazon és az elvárt értékeknek megfelelően próbálja meg beállítani a modell paramétereit. Amennyiben az iteráció során helyesen osztályoz egy adatpontot, nem

történik semmi, ellenkező esetben módosítja a bemeneti súlyokat. Mivel a két csoport lineárisan szeparálható, ezért véges lépésben jó megoldást tudunk találni. [6]



2.9. ábra Lineáris szeparáció [6]

Rosenblatt algoritmusa a következő. A kezdeti súlyokat véletlenszerűen inicializáljuk. Ezután egy iteráció következik, amiben az összes bemeneti mintát a modell bemenetére engedjük, és a súlyokat módosítjuk az alábbi szabály szerint:

$$w_i = w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

ahol t a várt, o a kapott kimenet, η a tanulási ráta, erről a dolgozatom későbbi részében lesz részletesebben szó. Ezt az iterációt addig folytatjuk ameddig szükségesnek látjuk azt (például a súlytényezők értéke nem változik, vagy egy adott hibahatáron belül vagyunk). Minden körben változtatnunk kell a súlyokat, ennek a változásnak az előjelét a $(t - o)$ dönti el. Amennyiben a két érték megegyezik nem történik módosítás. Növeljük a súly értéket, ha t nagyobb mint o, fordított esetben pedig csökkentjük. [6]

2.2.3 Modellek értékelése és annak metrikái

A modell teljesítményét minden esetben mérjük, hiszen ez az egyik legfontosabb visszajelzés a modell jóságáról, amit kaphatunk. Ennek a leggyakrabban használt módja az

ismeretlen minták értékelésével kapott találati arány, vagy más néven pontosság (accuracy). Ennek meghatározása n minta esetén az

$$acc = \frac{találatok}{n}$$

összefüggéssel történik, ahol a „találatok” a helyesen osztályozott minták számát jelenti.

A modellek kiértékelésének szemléletes ábrázolására használható a konfúziós mátrix (vagy keveredési mátrix), amely elkészítésének az elve a következő. Tegyük fel, hogy a modellünk bináris osztályozásra van tanítva, azaz két osztály közül kell a megfelelő osztályba sorolni (predikálni) a példákat. Legyen ez a két osztály pozitív és negatív. A konfúziós mátrixban négy értéket tudunk rögzíteni.

- Igaz pozitív: Helyesen osztályozott pozitív minták száma (IP).
- Hamis negatív: Negatívnak osztályozott pozitív minták száma (HN).
- Hamis pozitív: Negatív minták száma, amelyeket pozitívnak osztályozott a modell (HP).
- Igaz negatív: Helyesen osztályozott negatív minták száma (IN).

Ez alapján a konfúziós mátrix a következőképpen jeleníthető meg:

		Valós	
Predikált		+	-
	+	IP	HP
	-	HN	IN

2.10. ábra Konfúziós mátrix

A modell tesztelése után, a konfúziós mátrixban összefoglalható, hogy melyik lehetőség hányszor fordult elő. A gépi tanuláshoz használt könyvtárak általában tartalmaznak metódust ennek a mátrixnak az automatikus elkészítéséhez. Az így elkészült mátrix adatait

felhasználhatjuk különböző mérőszámok generálásához. Ilyen a korábban megismert pontosság is, amely a mátrix jelöléseivel a következőképpen adható meg:

$$acc = \frac{IP + IN}{IP + HP + HN + IN}$$

A pontosság értéke önmagában nem mindig megfelelő mértéket szolgáltat a modell teljesítményére. Tegyük fel, hogy rendelkezésünkre áll 100 minta a modell tesztelésére, amelyből 95 pozitív, 5 pedig negatív. Amennyiben a modellünk minden mintát a pozitív osztályba sorol, a pontossága 95%. Ez első ránézésre akár elfogadható is lenne, azonban a modell nem ismer fel egytelen negatív mintát sem. Emiatt a pontosság mellett más teljesítmény mutatók meghatározására is szükség van.

Egy O osztály pontossága megmutatja azt, hogy a modellnek hány százalékban volt igaza amikor az O osztályba sorolta a mintákat. Ez alapján a pontosság az igaz osztályra vonatkozóan a következőképpen számítható ki a konfúziós mátrix példájával:

$$pontosság(O) = \frac{IP}{IP + HP}$$

Az igaz osztályba sorolt mintákon belül a valóban igaz minták arányát más néven megbízhatóságnak is nevezik.

Kíváncsiak lehetünk még arra is, hogy egy adott osztályból hányat sikerült megtalálnia a modellnek, ekkor annak a fedését kell kiszámolnunk. Ezt tudjuk használni akár különböző anomália detektáló rendszerek teljesítményének a mérésére. Szintén O osztály esetén az osztályhoz tartozó fedés a következőképpen határozható meg:

$$fedés(O) = \frac{IP}{IP+HN}$$

A helyesen osztályozott igaz példák arányát az igaz példákban, azaz a helyes igaz felismerési arányt érzékenységnek vagy felidézésnek is nevezik.

Ennek a két metrikának csak együtt van értelme hiszen, ha csak a fedést maximalizáljuk a modell tanítása során, akkor minden mintát az igaz osztályba sorolással a fedés 100% lesz, mivel valóban megtalált minden mintát az adott osztályból.

Ennek kiküszöbölésére használhatjuk az F_1 − mértéket. Ez az előző két metrika harmonikus közepe lesz. Így egyetlen számmá tudjuk aggregálni a különböző mérőszámokat.

A modellek teljesítményének mérésére további teljesítmény mutatók is léteznek, amelyekre jellemző, hogy a modell egy vagy néhány jellemző tulajdonságára fókuszálnak. Az értékelés gyakran több mutató együttes figyelembevételével történik. Általában elmondható, hogy több modellt, többféle paraméterrel is célszerű tanítani a legalkalmasabb kiválasztásának érdekében, viszont az összes metrikát összehasonlítani, és az alapján döntést hozni sokszor nehéz feladat.

2.2.4 Optimalizáló algoritmusok

A való életben ritkán találkozunk teljesen jól szeparálható adathalmazokkal, viszont tökéletesen csak ebben az esetben tudunk osztályozni. Szerencsére az esetek nagy részében megelégedhetünk az „elég jól” működő modellek meghatározásával. Ebben az esetben az elválasztó sík „rossz oldalára” is kerülhet adatpont, ami hibásan osztályozott értéket jelöl. A célunk ennek a hibánk a minimalizálása. A hiba számszerű „méréséhez” hibafüggvényeket, más néven költségfüggvényeket használunk, ennek változói a neuron súlyparaméterei.

Átlagos abszolút hiba (Mean Absolute Error, MAE)

Ez a legegyszerűbb hibafüggvény, amit használhatunk, mert ez az átlagos hiba. A perceptron tanításnál használt módon t legyen a kimeneten várt érték, o a kapott érték. Ebben az esetben az abszolút eltérés $|t - o|$ lesz. Ezt a hibát kell az összes tanítópéldára vennünk, tehát az átlagos abszolút hiba a következő lesz A példahalmaz esetén:

$$MAE(w) = \frac{1}{N} \sum_{a \in A} |t_a - o_a|$$

Átlagos négyzetes hiba (Mean Squared Error, MSE)

Az átlagos négyzetes hiba nem más, mint az eltérések négyzetének átlaga. Ez népszerűbb, mint az átlagos abszolút hiba, mert „bünteti” a nagyobb hibákat. Meghatározása a következő összefüggéssel történik:

$$MSE(w) = \frac{1}{N}\sum_{a\in A}(t_a - o_a)^2$$

Átlagos négyzetes hiba négyzetgyöke (Root Mean Squared Error, RMSE)

Az átlagos négyzetes hiba négyzetgyöke amellett, hogy „bünteti” a nagyobb hibákat, a hiba egységében értelmezhető.

$$RMSE(w) = \sqrt{\frac{1}{N}\sum_{a\in A}(t_a - o_a)^2}$$

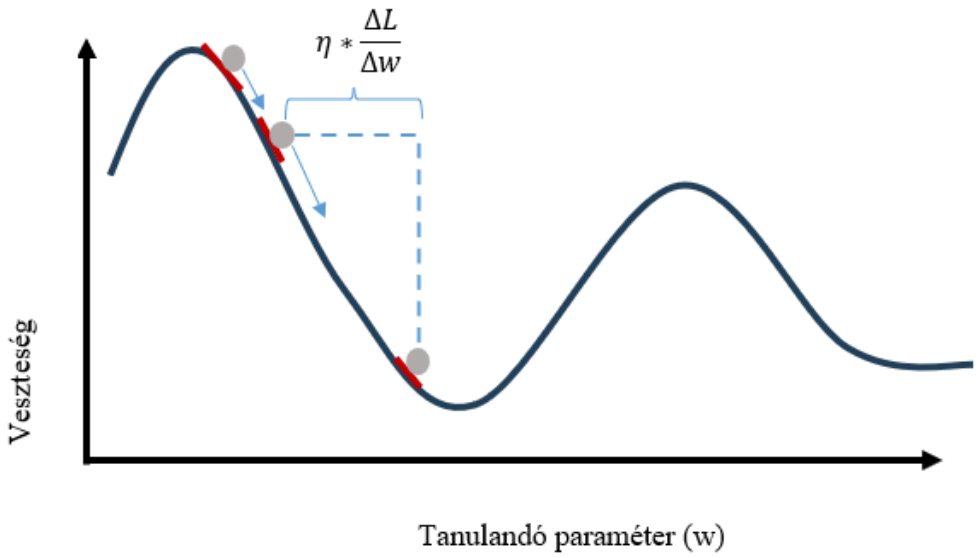
Továbbra is a modell legjobb súlyértékeinek megtalálása a célunk, ezeket a kiválasztott hibafüggvény optimalizálásával (minimalizálásával) tudjuk megkeresni, ezáltal egy sokváltozós optimalizálást végzünk. A hibafüggvény globális optimumát keressük. Ezt egyszerű esetben megadhatjuk egy zárt képlettel, viszont bonyolultabb esetekben iteratív algoritmusokat kell használnunk. Ezek általában nem tudják garantálni azt, hogy globális optimumot találunk, csak azt, hogy lokálisat.

Gradiens alapú optimalizáció

A Gradient Descent egy optimalizáló algoritmus függvények lokális minimumának kereséséhez, amelyet gyakorta használnak neurális hálózatoknál a paraméterek beállítására. A módszer célja a veszteség minimalizálása, amely eléréséhez iteratív módon módosítja a modellben a súlyokat. A módszer lényege, hogy a gradiens megadja azt az irányt, amerre a veszteségfüggvény a legmeredekebb, és a súlyokat az ellenkező irányba módosítja. Ennek a módosításnak a mértékét a η tanulási ráta határozza meg, ezt még a tanítási folyamat megkezdése előtt nekünk kell megadni. A tanulandó paraméter, jelen esetben az adott súly egyszeri kiszámítására ezzel a módszerrel az alábbi formulát alkalmazhatjuk:

$$w = w - \eta * \frac{\Delta L}{\Delta w}$$

ahol, w jelöli a súlyt, L pedig a hibafüggvényt.



2.11. ábra Gradient Descent algoritmus szemléltetése

A Gradient Descent algoritmus működését a 2.11. ábra szemlélteti. Az ábrán is látható, hogy az egyik legfontosabb paraméter a tanulási ráta (η). A minimumhely megkeresésekor kis η mellett a lépésközök kicsik lesznek, ami azt is eredményezheti, hogy nem találjuk meg időben a keresett értéket, viszont egy nagyobb tanulási ráta megadásával az könnyedén elérhető. A túl nagy η érték hátránya pedig az, hogy könnyedén átléphetünk az optimális értékek felett, vagy akár az algoritmus a természete miatt oszcillációba is fulladhat. Az oszcillálás elkerülése érdekében módosíthatjuk a gradiens módszert a következőképpen:

$$\Delta w_{ji}(n) = \eta \sum_{t=1}^{n} \alpha^{n-t} \delta_j(t) y_i(t) \ ,$$

ahol α – momentum konstans és $\Delta w_{ji}(0) = 0$. Ez lesz az általánosított delta szabály, amit következő fejezetekben ismertetett (többrétegű) neurális hálókban használnak. A képletben a j a modellbeli réteg számát, i pedig a rétegen belüli neuron sorszámát jelöli. Önmagában a Gradient Descent algoritmus elég pazarló, mivel minden egyes bemenet, minden egyes tanulandó paraméterre kiszámítjuk ezt. Gyakorlatban egyes hálózatok akár

több millió paramétert is tartalmazhatnak. A leghatékonyabb módja a számítási- és memóriaigény csökkentésére az a módszer, amikor nem minden egyes bemenő adatra számítjuk ezt ki, hanem egy azokból vett kisebb csoportra (mini-batch). Ezzel a módosítással megkapjuk az egyik, képfeldolgozáskor leggyakrabban használt optimalizáló algoritmust, az SGD-t (Stochastic Gradient Descent). [6]

2.3 Mély tanulás

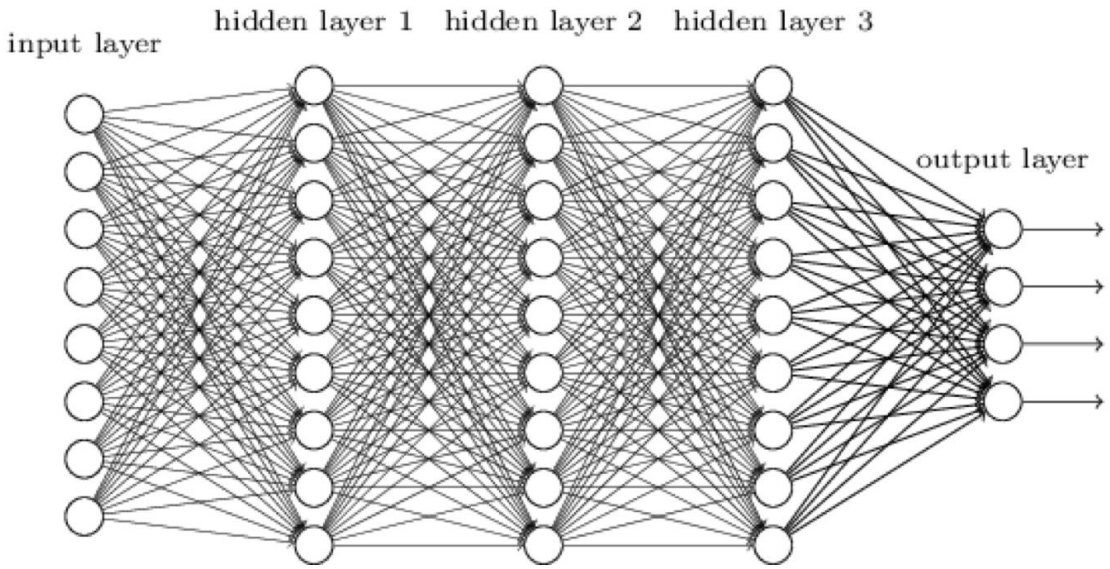
A mély tanulás egy alcsoportja a gépi tanulási technikáknak. A hagyományos gépi tanulás jól definiált algoritmusai helyett itt, mesterséges neurális hálókat használunk. Nincsenek előre megszabott szabályok, a teljes egészében a modellre bízzuk a tanulási folyamatot, ezáltal képes lesz különböző mintákat felismerni a bemeneti adathalmazból, amelyek akár nemstrukturált adatok is lehetnek, beleértve a képeket, videókat, dokumentumokat és szövegeket. Napjainkban rengeteg iparág használja ezt a technikát, az autóipartól egészen az orvosi képalkotásig.

2.3.1 Többrétegű perceptron (MLP)

Gyakorlati alkalmazásokban általában többrétegű neurális hálózatokat használnak. A korábban bemutatott egyszerű perceptron modell egy neuronból állt, amely több bemenettel és egy kimenettel rendelkezett, az MLP modellek több neuronokból épülnek fel és itt már rétegekbe is szervezzük azokat. Az így kapott hálóban három különböző típusú réteget különböztethetünk meg, ezek a bemeneti (input), a kimeneti (output) és a köztes (hidden) rétegek. [3]

A modell rétegei közül a bemeneti réteg lesz a legelső, ez fogja várni a modellbe beérkező adatokat. Az utolsó réteg pedig a kimeneti réteg, ami valamilyen aktivációs függvény segítségével egy később értelmezhető kimenetet fog eredményezni. A kimeneti réteg aktiválása, több-osztályos klasszifikációnál általában a Softmax, amely a Sigmoid aktivációs függvényhez hasonló, azonban az egyes osztályokhoz kapott kimeneti értékeket (exponenciális függvény értékek) az egyes osztályokhoz tartozás valószínűségévé transzformálja úgy, hogy ezen értékek összege egy legyen. [3]

A bemeneti és kimeneti rétegek között találhatóak a rejtett rétegek. Ebből akármennyi definiálható egy modellen belül, viszont kimeneti és bemeneti réteg szigorúan csak egy-egy lehet. Mély neurális hálózatról akkor beszélhetünk, ha a rejtett rétegek száma legalább kettő. [3]



2.12. ábra Három rejtett rétegű hálózat [https://www.inf.u-szeged.hu/~rfarkas/ML20/images/deep_network.png]

Ezeknél a hálózatoknál az egyes rétegeken belüli neuronok nem állnak egymással összeköttetésben, viszont a szomszédos rétegek minden neuronjával igen. Minden egyes neuron saját aktivációs függvénnyel és saját súlyokkal rendelkezik, amelyek a hozzá vezető összeköttetésekhez tartoznak. Több rétegű neurális hálókban kizárólag differenciálható átviteli függvényeket használhatunk, mivel a lokális gradiens kiszámításához szükséges a φ aktiválási függvény deriváltja. [6]

2.3.2 MLP tanítása

Az MLP esetén a jel a bemenettől áramlik a kimenet felé, tehát ezek előre csatolt hálózatok. A tanítás menete hasonlít az egyszerű perceptronéhoz. A kezdeti súlyokat megállapítjuk, majd a bemeneti jelet a bemenetre vezetjük, és végig áramoltatjuk a teljes hálózaton. A hálózat kimenetén előáll egy kimeneti jel. Ezt összevetjük a várt kimenettel, és a kapott hibát visszaterjesztjük a hálózaton és az alapján megváltoztatjuk a súlyokat. A cél továbbra is a hiba minimalizálása. Ezek a hálózatok képesek különböző szabályokat,

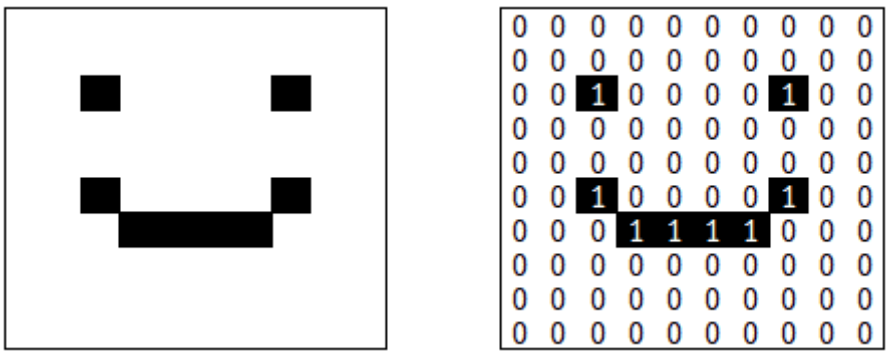
mintákat felismerni az bemeneti adathalmazban, és azok alapján meghatározni a várt kimenetet. Ez lehet a bemenő adatok osztályozása, vagy valamilyen tulajdonság meghatározása.

Egy neurális hálót könnyű túltanítani. Ez azt jelenti, hogy a modell megtanulja a tanító példákat és a hozzá tartozó várt értékeket, de nem tud jól általánosítani, és. Rendkívül pontos a tanulási folyamat végén, de eddig még számára ismeretlen mintákat képtelen a megfelelő osztályokba sorolni. Ez ellen tudunk valamilyen mértékben védekezni, az elérni kívánt hiba érték növelésével, így nem fog addig tanulni a modell, mint kisebb célhiba esetén, vagy csökkenthetjük a modell komplexitását is. Dolgozatom későbbi részében lesz részletesebben szó a túltanulás csökkentésére szolgáló technikák egy részéről.

2.3.3 Konvolúciós neurális hálózatok

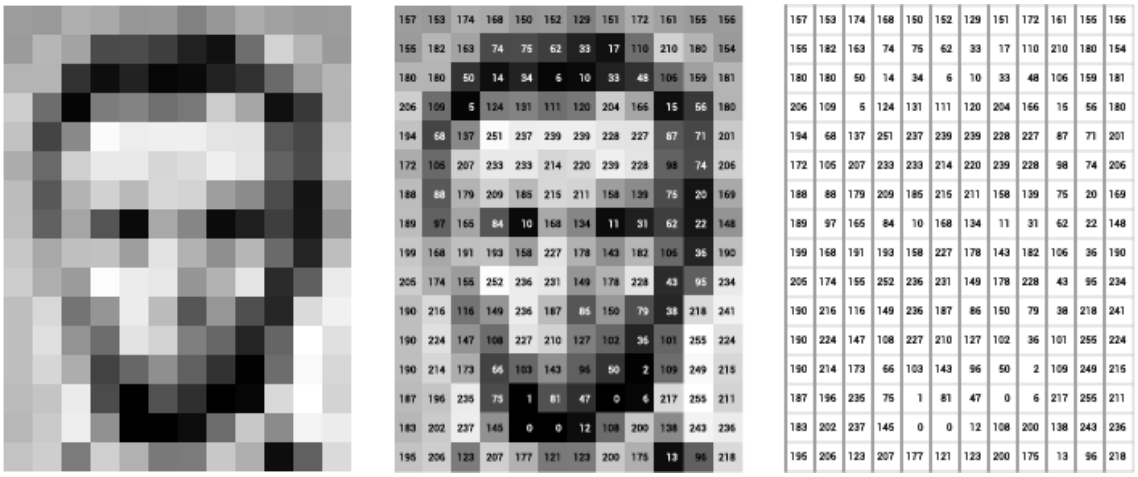
A konvolúciós neurális hálók (CNN – Convolutional Neural Network) a többrétegű perceptronokhoz hasonlíthatók, azzal a különbséggel, hogy ezek általában valamilyen – korábban már bemutatott – multimédia tartalmak értelmezésére specializált hálózatok. [9]

Amennyiben kép feldolgozása a feladat, azokat numerikus módon mátrixokkal tudjuk reprezentálni, ahol a mátrix mérete a kép méretének (magasság és szélesség pixelben megadva) felel meg. Amennyiben a mátrix elemei csak és kizárólag 0 vagy 1 értékek lehetnek, abban az esetben bináris, azaz fekete-fehér képről beszélünk.



2.13. ábra Képek bináris ábrázolása

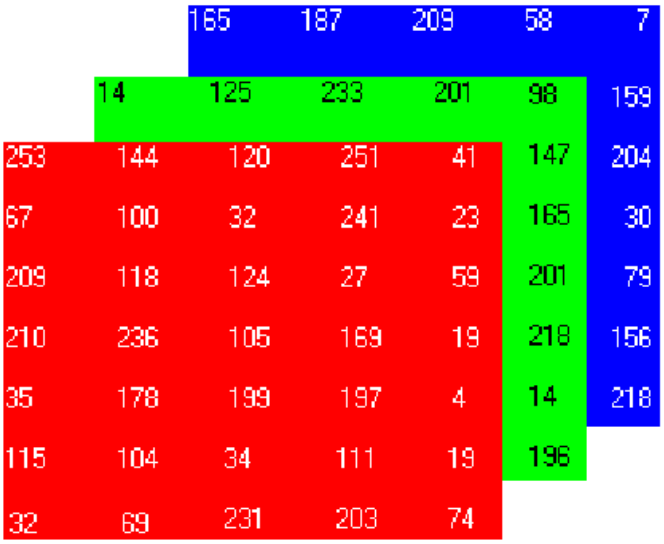
A 2.13. ábra egy képet és a hozzá tartozó bináris ábrázolást tartalmazza, ahol a fehéret a 0, a feketét pedig az 1 értékek jelentik. Amennyiben ennél pontosabban szeretnénk a képeket megjeleníteni, szürkeárnyalatos módon tehetjük meg azt. Ilyenkor szintén egy $n * m$ méretű mátrixot alkalmazunk, ahol n jelenti a kép szélességét, m pedig a magasságot pixelekben megadva. Egy pixel, a kép lekisebb értelmezhető egységét jelenti.



2.14. ábra Szürkeárnyalatos képek ábrázolás [10]

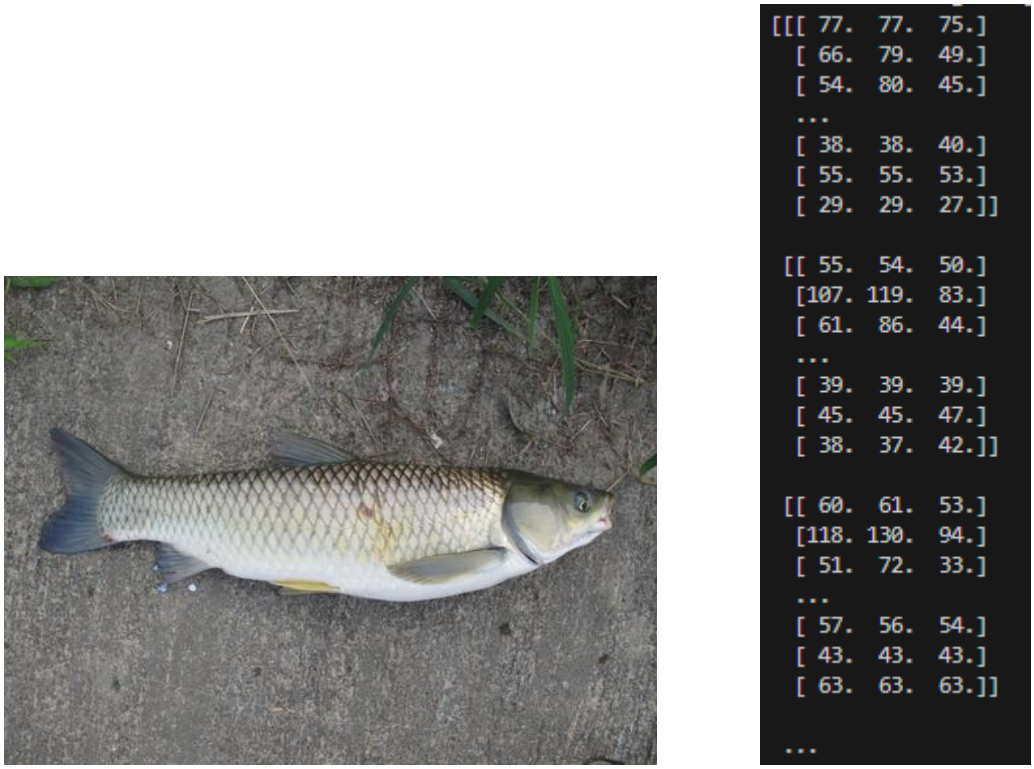
A mátrix elemei 0 és 255 közötti értékeket vehetnek fel, a határokat is beleértve. A 2.14. ábra egy szürkeárnyalatos képet és annak mátrix reprezentációját mutatja be, ahol a tartomány alsó határa a fehér a felső pedig a fekete színt reprezentálja. A feldolgozás egyszerűsítése érdekében, ezeket normálhatjuk, így 0 és 1 közötti értékeket kapunk.

Dolgozatomban színes képekkel foglalkozom. Egy színes képnél minden pixel rendelkezik egy piros (R), kék (B) és egy zöld (G) színcsatornának megfelelő értékkel, így a bemenő adatok $n * m$ méretű RGB képek, amire háromdimenziós mátrixként tekintünk, ahol $n -$ a szélesség, m – a magasság, és a harmadik dimenziót a kép színei adják, így kapunk egy $n * m * 3$ mátrixot, amelyre a 2.15. ábra mutat példát.



2.15. ábra Egy RGB kép mátrixos ábrázolása [11]

Szakdolgozatomban magyarországi halfajták osztályozása volt a feladatom, az ehhez használt adathalmaz egyik mintája (egy amur) látható a 2.16. ábra, és mellette a numerikus megjelenítésének egy részével.

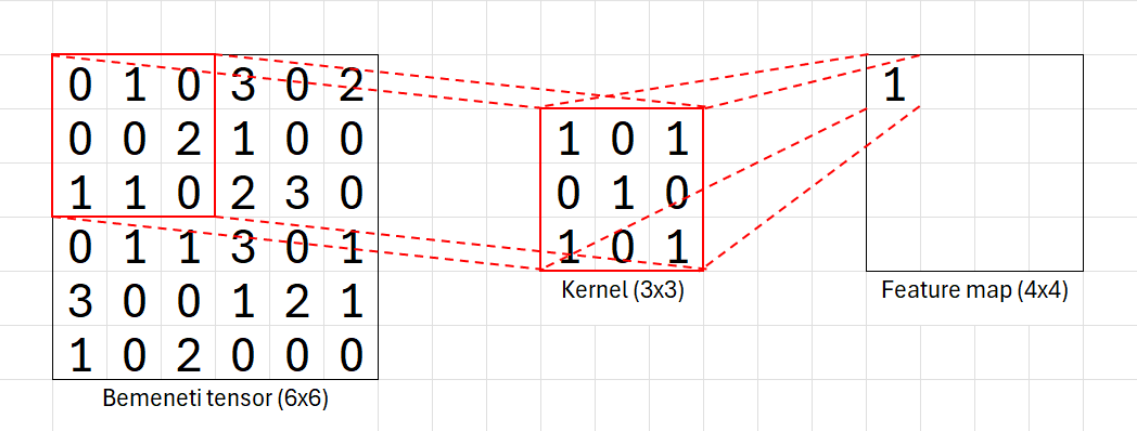


2.16. ábra Színes kép numerikus ábrázolása, a hal egy amur

Ezekből a bemeneti mátrixokból (úgynevezett tensorokból), a modellnek elő kell állítania egy predikciót a hal fajtájára. Az előző ábrán látható adatokon is látható, hogy szimplán a nyers RGB értékekből nem meghatározható az adott halfaj. Különböző műveletek és transzformációk elvégzésével, viszont a modell képes lesz meghatározni ezt. [9]

A konvolúciós neurális hálózatok alapeleme a nevéből is adódóan a konvolúciós réteg. Ez a réteg lineáris és nemlineáris műveletekből áll, amit egy általában egy konvolúció és egy nemlineáris aktivációs függvény alkot. Ebben a rétegben valamilyen jellemzőt próbálunk kinyerni a bemeneti tensorból. A képekből előállított mátrixokra a továbbiakban tensorként fogok hivatkozni. A konvolúció folyamatát a 2.17. ábra szemlélteti, amely során egy úgynevezett kernel mátrix (vagy súlymátrix) minden eleme és a bemeneti tensor szorzatát képezzük, majd összeadjuk ezeket, ez fogja adni a tulajdonság térkép (feature map) egy elemét. A modell konfigurálása során kell meghatároznunk a kernelek méretét

és számát. A kernelek száma bármekkora szám lehet, viszont a méretét célszerű $n * n$ formában megadni. [9]



2.17. ábra Konvolúció szemléltetése

Az ábrán jól látható, hogy a kimenet nem egy az egyhez arányú reprezentációja a bemenetnek, mivel a kernel középső eleme soha nem kerül a bemeneti tensor szélső elemeire. Ez nem minden esetben jó, mivel a rétegek közötti reprezentáció csökkentésére vannak más eszközök is, tehát szükség van valamilyen technikára, amivel kiküszöbölhető ez a probléma. A megoldás az úgynevezett padding, amit a modell konfigurálásakor kell megadnunk minden egyes rétegnél, hogy szeretnénk-e használni, és ha igen, akkor milyen megvalósításban. Alapvetően kétféle padding beállítási módot különböztetünk meg:

1. Same-Padding: A nevéből adódóan azonos méretű tensort ad vissza, amelyben nullákat használunk arra, hogy a szélső elemek is egy helyre essenek a kernel középső elemével. A módszer lényege az, hogy nullákkal kiegészítjük a bemenő tensort, így minden elem lehet a kernel közepén. Ez a kimeneti értékeket nem befolyásolja, mivel az elemenkénti szorzat képzésekor szintén nulla értékként fognak megjelenni, így a Same-Padding alkalmazásával a feature map mérete megegyezik a bemeneti tensor méretével.
2. Valid-Padding: Technikailag ez egyenlő azzal, mikor nem alkalmazunk paddinget, ez esetben a bemenő tensor nem kerül kiegészítésre a konvolúció során.

A művelet természetéből adódóan minél kisebb kerneleket alkalmazunk, annál apróbb részletek felismerésére lesz alkalmas a modell, viszont a tanulandó paraméterek száma is jelentős mértékben növekszik. A tanulandó paramétereket konvolúciós rétegekre meghatározhatjuk az alábbi módon:

$$P = (n * m * l + 1) * k$$

ahol, n és m a kernel méretét jelöli, l az előző réteg kimeneti feature mapjai számát jelöli, k pedig a jelenlegi rétegben található kernelek számát. Látható, hogy egy konvolúciós réteg tanulandó paramétereit, nem csak a saját, hanem az előző réteg beállításai is befolyásolják. A kernel méretének túlzott mértékű növelése, vagy csökkentése könnyedén túltanuláshoz vezethet, mivel a modell nem képes az osztályozási feladat számára releváns mintákat megtanulni.

Egy konvolúciós rétegnek több beállítása is van:

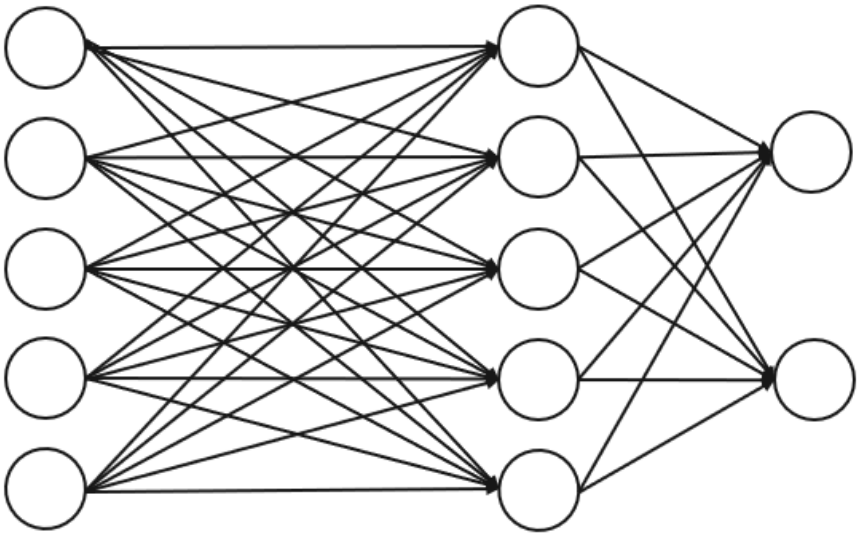
1. Kernelek száma: Meghatározza, hogy hány kernelt léptetünk végig a bemenő adaton, minden kernel egy tensort eredményez a következő rétegben. Minél több kernelt tartalmaz a réteg, annál nagyobb absztrakcióra képes a CNN viszont, ha túl sokat definiálunk, az túltanuláshoz is vezethet.
2. Kernel méret: Nevéből adódóan meghatározza, hogy az egyes kernelek mekkora méretűek.
3. Padding: A padding módját állítja be.
4. Stride: Meghatározza, hogy a kernel konvolúció során hány pixelt lép a bemeneti mátrixon, minél nagyobb ez az érték annál kisebb a kimeneti reprezentáció mértéke.
5. Aktiváció: Milyen aktivációs függvényt használunk az adott rétegben.

Csak konvolúciós rétegeket a tanulandó paraméterek mennyisége miatt pazarló lenne használni, ezért szükséges a reprezentáció csökkentése, erre egy megoldás lehet a stride értékének növelése, viszont a legjobb módszer valamilyen pooling réteg alkalmazása. Alap esetben kétfajta poolingot különböztetünk meg: átlag (Average-Pooling) és maximum (Max-Pooling). Mindkettő feladata a kapott tulajdonságtérképek méretének

csökkentése. Mindkettő tartalmaz egy filtert, amit végigmozgatunk a bemenő adatokon. A léptetés mértékét itt is a stride beállításával határozhatjuk meg. A filter mérete tetszőlegesen megválasztható viszont, ha túl nagy a mérete, akkor a kimeneti mátrix túl kicsi lesz a bemenethez képest. A maximum és az átlag pooling közötti különbség az, hogy a Max-Pooling réteg az aktuális filter állás szerinti maximum értéket teszi a kimenetre, az Average-Pooling pedig az átlagot. Ezekben a rétegekben is állítható a padding módja, viszont célszerű valid-paddinget használni, mivel a nullákkal való kiegészítés befolyásolhatja a réteg kimenetét. Ez réteg nem tartalmaz tanulandó paramétereket, mivel csak a reprezentáció mértékét csökkenti. Létezik egy harmadik típusú pooling is, a global pooling, amely a tulajdonságtérkép minden csatornájának értékét egy értékre csökkenti. Ennek is az előbbiekben taglalt két típusa van, viszont ez a réteg extrém módon csökkenti a reprezentáció mértékét.

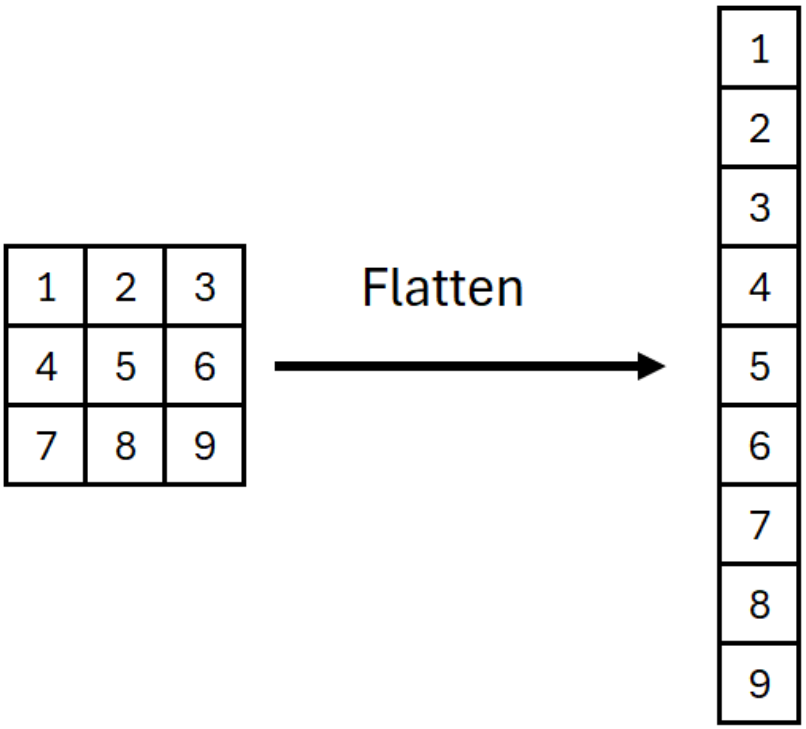
Egy konvolúciós neurális hálóban egymás után több konvolúciós és pooling réteg is használható.

Az eddig bemutatott rétegek bemenetei és a kimenetei is mátrixok voltak, viszont a megvalósítandó osztályozási feladathoz illeszkedően ezeket a mátrixokat át kell alakítanunk egy olyan formára, amit a modell predikciójaként értelmezhetünk. A kimenet képzéséhez teljesen összekötött (Fully Connected, FC) rétegeket használunk. Ezek fő jellemzője az, hogy a bemenet minden pontja összeköttetésben áll a kimenet minden pontjával, innen adódik a neve is.



2.18. ábra Három teljesen összekötött réteg, két kimeneti osztállyal

A teljesen összekötött rétegek fogják adni a klasszifikációhoz szükséges kimenetet, ahol a kimenetek száma megegyezik az osztályok számával, - amennyiben ez az utolsó FC réteg, ez szükséges is - ezáltal előáll az egyes neuronok kimenetén egy szám, softmax aktiváció esetén ez, 0 – 1 között lesz. Mivel az egyes neuronok az utolsó rétegben már egy-egy osztályt jelentenek, ezért az előbbi kimenet az adott osztály valószínűségét jelöli, így kapunk egy n elemű vektort, amelynek minden eleme egy valószínűség, és a vektor elemeinek összege 1, ebből egy maximum kiválasztással kinyerhető a modell által predikált osztály. A teljesen összekötött rétegek bemenetként egy vektort várnak, viszont az előzőekben tárgyalt konvolúciós és pooling rétegek kimenete egy vagy több mátrix. A kettő közé kell tennünk egy flatten réteget, ami a mátrixokból vektort képez. Ezt mutatja a 2.19. ábra.



2.19. ábra Flatten réteg szemléltetése

Korábban már említettem, hogy könnyű a modellt túltanítani. A fogalom angol neve (overfitting) jobban reprezentálja a problémát. A modell a tanulási folyamat során képes a klasszifikációs problémához nem releváns információkat is megtanulni, és összekötni azokat az eredeti osztály címkével. Ennek lehet több oka is:

1. Túl kicsi a tanító halmaz. Ez azt eredményezi, hogy a modellnek szimplán nem áll rendelkezésére elég információ ahhoz, hogy sikeresen megtanulja a szükséges mintákat
2. A tanító halmaz elemei zajosak, azaz sok nem releváns információt tartalmaznak.
3. Túl sokat tanítjuk a modellt az adatokon, ennek a mértékegysége nem feltétlenül az idő, hanem lehet epoch is. Az epoch azt jelenti, hogy az összes adatot egyszer végig áramoltattuk a hálózaton, a maximális számát a tanítás előtt kell meghatároznunk.

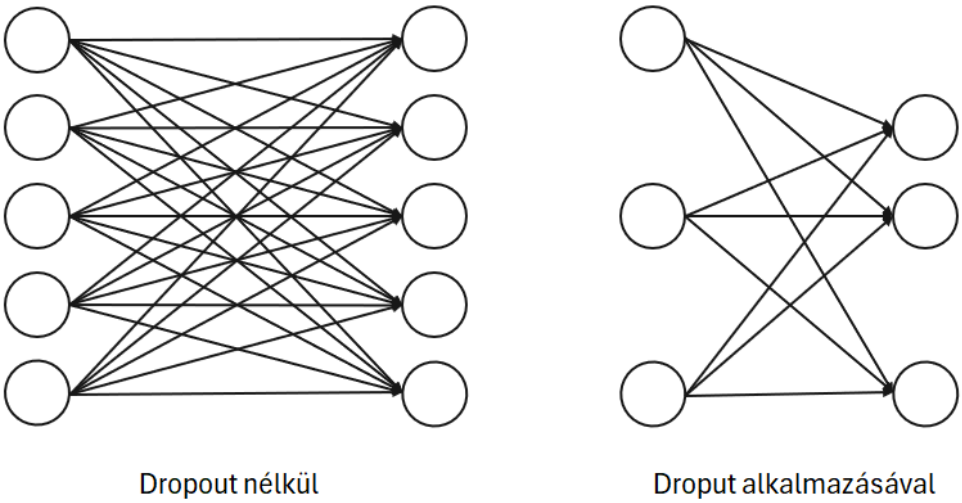
4. A modell túl komplex.

A modellek tanítása során én is tapasztaltam túl tanulást, a fenti okok és azok kombinációja miatt. A legszembetűnőbb az volt, amikor egy-egy halat félreosztályozott a modell pusztán azért, mert azt valaki a kezében tartotta. A tanító példák között több nagytestű halfaj is szerepelt, amiket kézben tartva fényképeztek le, ami azt eredményezte, hogy sok ily módon fényképezett halat egy osztályba sorolt a modell, hiába a szemmel látható jelentős különbség a két halfaj között.

2.20. ábra Hasonló módon fényképezett halak

A 2.20. ábra bal oldalán egy amur látható, még a jobb oldalon egy ponty. A két hal alakja és színe között is nagy eltérés van, viszont előfordulhat, hogy a fent leírt okokból a modell egy osztályba sorolja a kettőt. Jelentkezhet ennek a fordítottja is, például: amikor egy halfajt kézben fényképeztek az esetek nagy részében, akkor előfordulhat, hogy egy matracra helyezett egyedet nem a megfelelő osztályba sorol a modell.

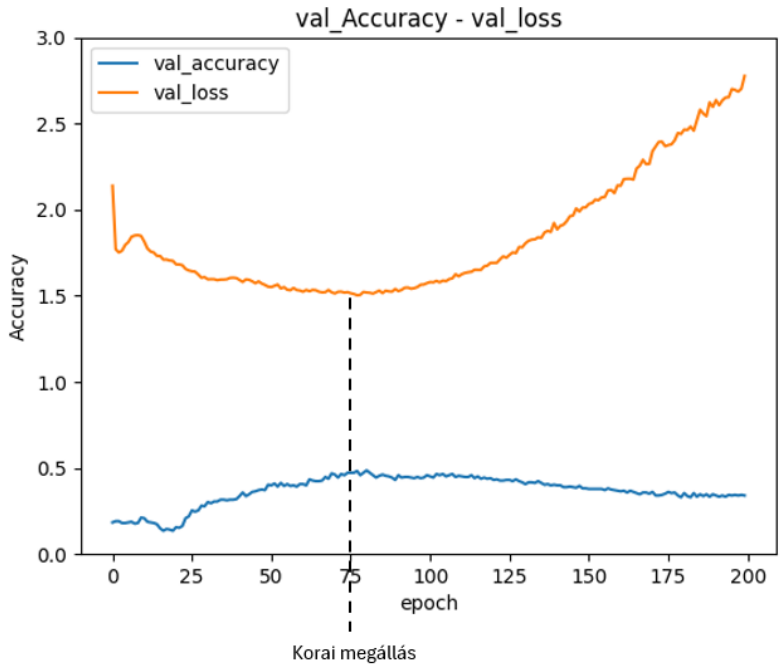
Túltanulás ellen védekezhetünk több módon is. Ebből a legegyszerűbb az, hogy több mintát gyűjtünk a tanító halmaz osztályaihoz, vagy a zajos fényképeket eltávolítjuk. Amennyiben sok a zajos kép, az is egy megoldás lehet, ha magát az osztályozást zavaró tényezőket távolítjuk el a képekről, ilyenek a képekbe belelógó nem odaillő tárgyak, vagy akár maga a kép háttere is. Léteznek azonban ennél sokkal szofisztikáltabb módszerek is a túltanulás csökkentésére. Az egyik ilyen módszer a dropout használata a rétegek között (2.21. ábra).



2.21. ábra Dropout szemléltetése

Ennek lényege az, hogy egy adott réteg neuronjait „kiejtjük”, ez által a „kiesett” neuronhoz tartozó összeköttetések ideiglenesen megszűnnek és ideiglenesen egy új struktúra jön létre. Ez a megoldás főleg nagyobb konvolúciós, valamint a teljesen összekötött rétegek rétegek között eredményes. Modell készítésekor kell meghatároznunk, hogy hova helyezzük el a dropoutokat, valamint meg kell adni egy nulla és egy közötti értéket, ami annak az aránya, hogy az adott rétegben levő neuronok hány százalékát dobja el a modell. Az, hogy az eldobandó neuronok közül pontosan melyik kerül törlésre, véletlenszerűen kerül meghatározásra a tanulás során.

Korábban már említettem, hogy a modell túl sokáig történő tanítása is eredményezhet overfittinget, ennek kiküszöbölésére használható az early stopping technika, ami korai megállást jelent. Az early stopping egy „visszahívó”, azaz callback függvény, ami minden egyes epoch után kiértékeli a tanulás alatti statisztikáit a modellnek.



2.22. ábra Early Stopping

A 2.22. ábra is jól látható, hogy körülbelül a 75. epoch után már felesleges tovább folytatni a tanítást, mivel a veszteség innentől növekszik, és a pontosság csökken. A modell konfigurálásakor megadhatjuk, hogy mely paramétert szeretnénk monitorozni a tanítás során, ezek lehetnek: tanítási pontosság, tanítási veszteség, valamint validációs pontosság és validációs veszteség. Az early stopping algoritmusa figyeli a beállított paraméter értékét, és attól függően, hogy veszteséget vagy pontosságot adtunk meg, minimalizálja vagy maximalizálja azt, tehát ha az utolsó néhány epoch-hoz képest, nem változik vagy romlik a figyelt érték, akkor a tanulási folyamat megáll. Az early stopping másik paramétere a patience, ami megadja azt, hogy hány epoch-nyi időt hagyunk a modellnek a javulásra, miután a figyelt érték romlásnak indult. Egy tanítási folyamat alatt, csak egy paraméter szerint állíthatjuk meg korábban a tanulási folyamatot. Amennyiben többet szeretnénk, azt early stopping-gal nem tudjuk megoldani, viszont beállítható az is, hogy a fent említett paraméterek szerint bármelyik monitorozásával az epoch után el tudjuk menteni a modell súlyait, amit, ha betöltünk a tesztelés előtt, reprodukálni tudjuk az early stopping működését.

3. AZ OSZTÁLYOZÓ RENDSZER MEGVALÓSÍTÁSA

Dolgoztatom során a feladatom az volt, hogy megismerjek különböző gépi tanulási technikákat, és ezek segítségével készítsek egy konvolúciós neurális hálózatot, ami képes Magyarországon fellelhető halfajokat osztályozni. Ehhez a közkedvelt Python programozási nyelvet választottam. A neurális hálózatokat Keras [12] nevű könyvtár segítségével készítettem el, ami a Google TensorFlow [13] könyvtárán alapszik. A kódot eleinte a Google Colab [14] nevű szolgáltatását használva futtattam, de ez nem bizonyult számomra kényelmesnek, így a Colab ingyenes verziója helyett áttértem a Keras GPU optimalizált változatára, ez által jelentős teljesítmény csökkenés nélkül tudtam a modellt a saját számítógépemen tanítani. A modell vizuális megjelenítéséhez a visualkeras könyvtárat használtam, ami segítségével a későbbiekben látható ábrák készültek, teljesen automatikusan a tanítás előtt.

3.1 Képek beszerzése és előkészítése

A halakról készült képeket egy részben saját magam készítettem horgászataim során, míg egy részüket internet segítségével gyűjtöttem be. Internetes forrásaim az iNaturalist és Kaggle nevű weboldalak voltak. Az iNaturalist [15] egy online adatbázis, amelyet amatőrök és szakemberek közösen használnak és töltik fel oda megfigyeléseiket. Az oldalról csv formátumban letöltöttem egy listát, minden az általam kiválasztott halfajokról, ezek a következőek: amur, busa, ponty, harcsa és sügér. A lista a felhasználók által feltöltött megfigyeléseket tartalmazza, és egy linket, ahol elérhető a hozzá tartozó kép. A csv fájlból kimásoltam az összes linket, majd egy szövegfájlba helyeztem őket. A 3.1. ábra látható PowerShell kóddal ezeket automatikusan letöltöttem, és a saját mappájába helyeztem.

```
$i = 0
foreach($line in Get-Content .\pontyexport.txt) {
    Write-Output $line
    curl $line -o ./inatexport/Ponty/$i.jpg
    $i = $i+1
}
```

3.1. ábra Egy halfaj (Ponty) képeinek letöltésére használt algoritmus

Nem találtam minden fajhoz elegendő képet ezen az oldalon, ezért még többet a Kaggle oldalon elérhető datasetből töltöttem le. Minden fényképet kézzel át kellett válogatnom, mivel az iNaturalist adatbázisból letöltöttek között szerepeltek élettelen egyedek is, annak ellenére, hogy kifejezetten csak élő megfigyelésekre szűrtem. A képek túl sok manuális előkészítést nem igényeltek ezen kívül, csak a felesleges zavaró tárgyakat és embereket vágtam le a képekről, ha tartalmazott egyáltalán, hiszen majdnem mindent Keras segítségével oldottam meg.

Az így kapott datasetben található képeket még augmentálni kellett, mivel viszonylag kis méretűek lettek az egyes halmazok.

Faj	**Training (db)**	**Teszt (db)**	**Összesen (db)**
Amur	143	24	167
Busa	340	48	388
Harcsa	320	62	382
Ponty	361	40	401
Sügér	293	60	353
Összesen	1457	234	1691

1. táblázat Nyers képek száma osztályonként és összesítve

A képek augmentálására a Keras ImageDataGenerator beépített paramétereit használtam, így nem volt szükség extra rétegeket adni a modellhez. A paraméterek a következők voltak: elforgatás 30 fokonként, valamint egy zoom tartomány, aminek az értéke 0.2. A zoom tartomány 0 és a megadott érték, jelen esetben 0.2 közötti véletlen nagyítást alkalmaz a képeken, minden kép esetén új értéket generálva. Ennek az eredménye a 3.2 ábrán látható.

3.2. ábra Képek augmentáció után

Tanuló és validációs halmazokra való bontását az eredeti Training halmaznak automatikusan a modell végezte a tanulási folyamat megkezdése előtt, a képek 20 százaléka került minden osztályból a validációs halmazba. Annak érdekében, hogy minden egyes tanítás esetén ezek a halmazok változatlanok maradjanak, egy seed értéket adtam meg, ami esetemben 2024 volt.

3.2 Neurális háló architektúrák

A feladat megoldása során több saját, illetve mások által készített architektúrát is kipróbáltam, a lehető legpontosabb osztályozás elérésének érdekében. Értelemszerűen minden modell ugyan azt a klasszifikációs problémát próbálta megoldani, emiatt a tanító, és teszt halmazok megegyeztek minden modell esetén, viszont a bemeneti képek mérete már nem minden modellnél volt ugyan az. A legnagyobb különbségek a modellek között a rétegek típusában és számában fedezhető fel.

3.2.1 InceptionV3

Az InceptionV3 volt az első modell, amit kipróbáltam. Az InceptionV1 és V2 modellekre épül, viszont azoknál egy mélyebb, több rétegből álló architektúra. Eredetileg az ImageNet dataseten tanították, ahol 75% feletti pontosságot ért el. [16] Felépítése a következő:

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

3.3. ábra InceptionV3 felépítése [16]

Az táblázatban hivatkozott ábrák a hivatkozott forrásban [16] megtekinthetőek.

A modell eredetileg 1000 osztállyal lett tanítva, és 299x299 méretű RGB képeket vár bemenetként, viszont a feladatomban csak 5 osztály található, ezért az első, bemeneti réteget és az utolsó teljesen összekötött réteget le kellett cserélnem egy a számomra megfelelőre. A bemeneti réteg így 224x224 méretű képeket vár, míg az utolsó réteg egy 5 neuronból álló FC réteg lett. Az eredeti modell, és az általam tanított modell kimenete közé még elhelyeztem több FC réteget dropout alkalmazásával a túltanulás elkerülése érdekében, hiszen az eredeti 1000 osztály és az én 5 osztályom között nagy a különbség, és nem feltétlenül van szükség akkora komplexitásra a kimenet képzése során. A dropout értékét Hinton [17] ajánlása alapján 0.5 és a későbbiekben többek által ajánlott 0.2 értékekkel használtam.

```
x = inceptionv3.output
x = layers.GlobalAveragePooling2D()(x)
x = layers.Flatten()(x)
x = layers.Dense(512, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dropout(0.2)(x)
x = layers.Dense(64, activation='relu')(x)
x = layers.Dropout(0.2)(x)
predictions = layers.Dense(5, activation='softmax')(x)
```

3.4. ábra InceptionV3 implementációm

A tanítás hossza 100 epoch-ban volt maximalizálva, viszont a korai megállás miatt a 9. után megállt a tanulás. A modell statisztikái az adathalmazon a következőek:

Epoch	Tanítási pontosság	Validációs pontosság	Teszt pontosság	Tanítható paraméterek száma
9	87.15%	88.62%	66.6%	1,221,893

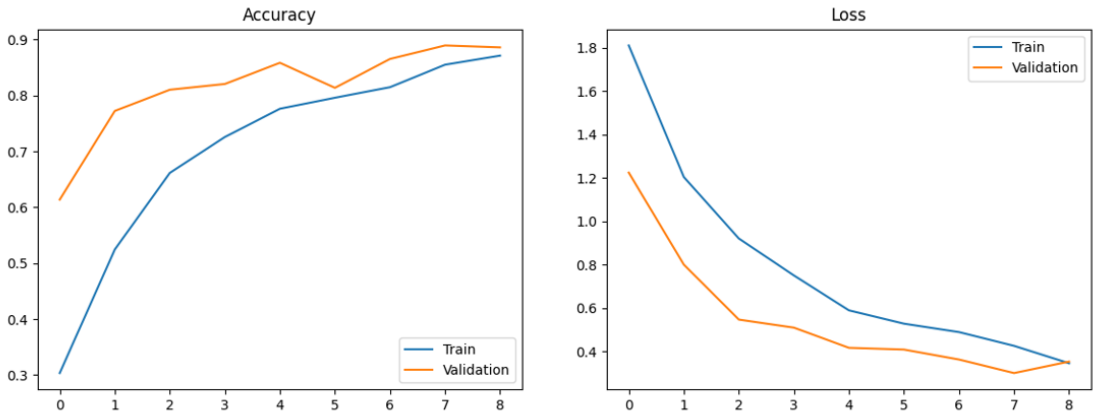
3.5. ábra InceptionV3 Statisztikái

	Amur	Busa	Harcsa	Ponty	Sügér
Amur	1	0	3	11	9
Busa	1	16	1	8	22
Harcsa	0	1	59	0	2
Ponty	2	0	0	28	10
Sügér	0	0	4	4	52

3.6. ábra InceptionV3 konfúziós mátrix

Látható, hogy az amurt klasszifikálta a legrosszabbul, mivel ebből a fajból állt rendelkezésemre a legkevesebb kép a tanításhoz. A további modellekre is ez a tendencia jellemző.

A pontosság és veszteség a következőképpen alakult az epoch-ok számának függvényében:



3.7. ábra InceptionV3 pontosság és veszteség

3.2.2 ResNet50

A ResNet50 hálózatot 2015-ben publikálták. Nevét az alkalmazott technológiáról kapta, Residual Network [18], az 50 pedig a hálózat mélységét, azaz a konvolúciós neurális háló rétegeinek számát jelenti. Eredetileg ez a modell is az ImageNet dataseten lett tanítva, bemenete 224x224 méretű képek. [18] Több ResNet hálózat készült el, melyek rendre 18, 34, 50, 101 és 152 rétegből állnak. A ResNet modellek felépítését a 3.8. ábra mutatja be.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

3.8. ábra ResNet hálózatok felépítése [18]

A bemeneti képek mérete, és a modell kimenetét követő teljesen összekötött rétegek megegyeznek az előző (InceptionV3) modell implementációjával, és a tanulás hossza is szintén 100 epoch-ban volt maximalizálva.

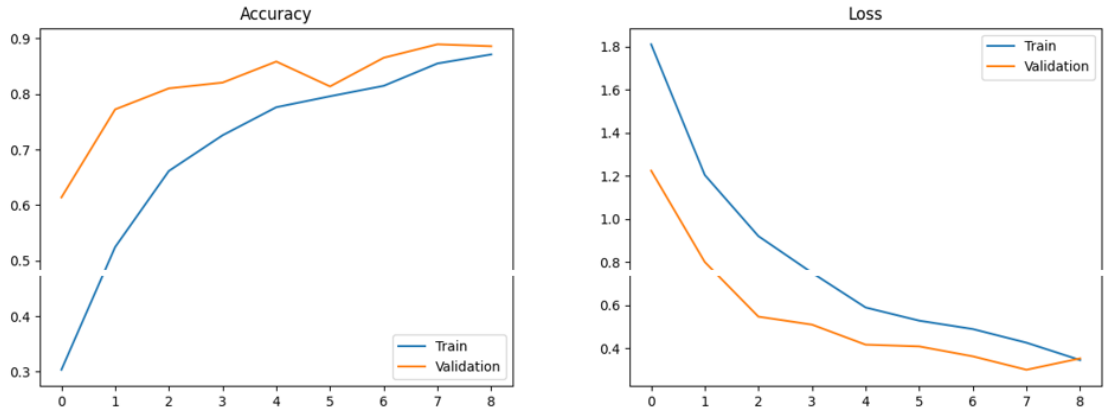
A modell statisztikái a következőek:

Epoch	Tanítási pontosság	Validációs pontosság	Teszt pontosság	Tanítható paraméterek száma
9	23.14%	23.45%	25.51%	1,221,893

3.9. ábra ResNet50 statisztikái

	Amur	Busa	Harcsa	Ponty	Sügér
Amur	0	24	0	0	0
Busa	0	48	0	0	0
Harcsa	0	62	0	0	0
Ponty	0	40	0	0	0
Sügér	0	60	0	0	0

3.10. ábra ResNet50 konfúziós mátrix



3.11. ábra ResNet50 pontosság és veszteség

A statisztikai adatokon is jól látható, hogy ez a modell rendkívül rosszul teljesített az adathalmazon, a konfúziós mátrixról leolvasható, hogy képtelen volt az osztályozásra.

3.2.3 Saját Architektúra 1

Több általam készített architektúrát is kipróbáltam, ezek közül a további fejezetekben hármat ismertetek.

Az első saját architektúra egy több, viszont kisebb rétegeket használó modell, amelynek a felépítése a következő:

A bemeneti réteg 32 darab 8x8 méretű kernelt használó konvolúciós réteg, ami 128x128 méretű RGB képeket vár, 0.2 dropout alkalmazásával. Ezt követi egy maximum-pooling réteg, 3x3 kernel mérettel. Ezt egy három konvolúciós rétegből álló blokk követi, melynek első rétege megegyezik a modell bemeneti rétegével, a második kettő pedig 16 darab 4x4 kernelt alkalmaz, az utolsó rétegen van droupot, 0.2 rátával. Ismét egy konvolúciós blokk következik, három darab réteggel, mind három 16 darab 2x2 kernelt tartalmaz. Eddig minden réteg aktivációja ReLu volt. Ezt a blokkot egy flatten réteg követi, majd a kimenet képzéséért felelős teljesen összekötött rétegek. Ebből 3 darabot tartalmaz a hálózat softmax aktivációs függvénnyel. Az első 256, majd 128 neuront tartalmaz. Az utolsó FC réteg neuronjainak száma megegyezik az osztályok számával, tehát 5. A modell elkészítéséhez tartozó kódrészlet a 3.12. ábra, a modell összefoglalása a 3.13. ábra, a modell sematikus struktúrája pedig a 3.14. ábrán látható.

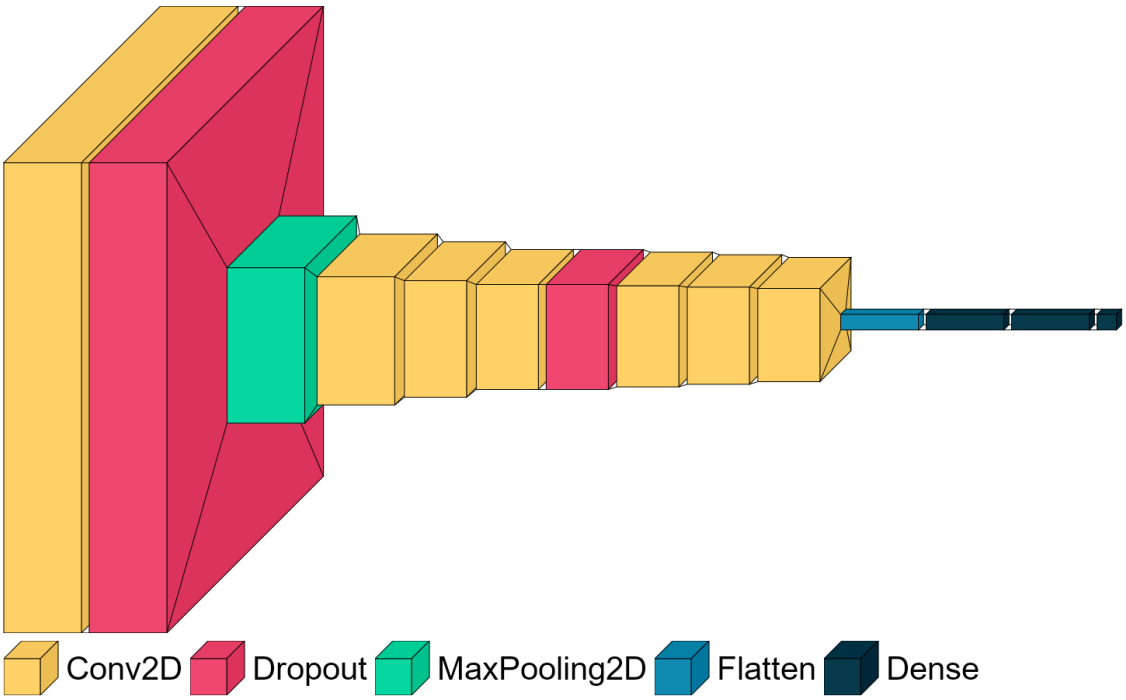
```
def create_model():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32, 8, activation='relu',input_shape=(128,128,3)),
        tf.keras.layers.Dropout(0.20),
        tf.keras.layers.MaxPool2D((3,3)),
        tf.keras.layers.Conv2D(32, 8, activation='relu',input_shape=(128,128,3)),
        tf.keras.layers.Conv2D(16, 4, activation='relu',input_shape=(128,128,3)),
        tf.keras.layers.Conv2D(16, 4, activation='relu',input_shape=(128,128,3)),
        tf.keras.layers.Dropout(0.20),
        tf.keras.layers.Conv2D(16, 2, activation='relu',input_shape=(128,128,3)),
        tf.keras.layers.Conv2D(16, 2, activation='relu',input_shape=(128,128,3)),
        tf.keras.layers.Conv2D(16, 2, activation='relu',input_shape=(128,128,3)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(256,activation = 'softmax'),
        tf.keras.layers.Dense(128,activation='softmax'),
        tf.keras.layers.Dense(num_classes,activation='softmax')
    ])

    return model
cnn_model = create_model()
optimizer = tf.keras.optimizers.SGD(learning_rate=0.001)
cnn_model.compile(optimizer=optimizer, loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics=['accuracy'])
if TRAIN:
    history = cnn_model.fit(train_dataset, epochs=100, validation_data=validation_dataset,
                            verbose=2,callbacks=[es,model_checkpoint_callback,model_checkpoint_callback_loss])
```

3.12. ábra A Saját Architektúra 1. CNN elkészítéséhez tartozó kódrészlet

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 121, 121, 32)	6176
dropout (Dropout)	(None, 121, 121, 32)	0
max_pooling2d (MaxPooling2D)	(None, 40, 40, 32)	0
conv2d_1 (Conv2D)	(None, 33, 33, 32)	65568
conv2d_2 (Conv2D)	(None, 30, 30, 16)	8208
conv2d_3 (Conv2D)	(None, 27, 27, 16)	4112
dropout_1 (Dropout)	(None, 27, 27, 16)	0
conv2d_4 (Conv2D)	(None, 26, 26, 16)	1040
conv2d_5 (Conv2D)	(None, 25, 25, 16)	1040
conv2d_6 (Conv2D)	(None, 24, 24, 16)	1040
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 256)	2359552
dense_1 (Dense)	(None, 128)	32896
dense_2 (Dense)	(None, 5)	645

Total params: 2,480,277
Trainable params: 2,480,277
Non-trainable params: 0

3.13. ábra A Saját Architektúra 1. CNN rétegei és azok paraméterei



3.14. ábra A Saját Architektúra 1. CNN sematikus struktúrája

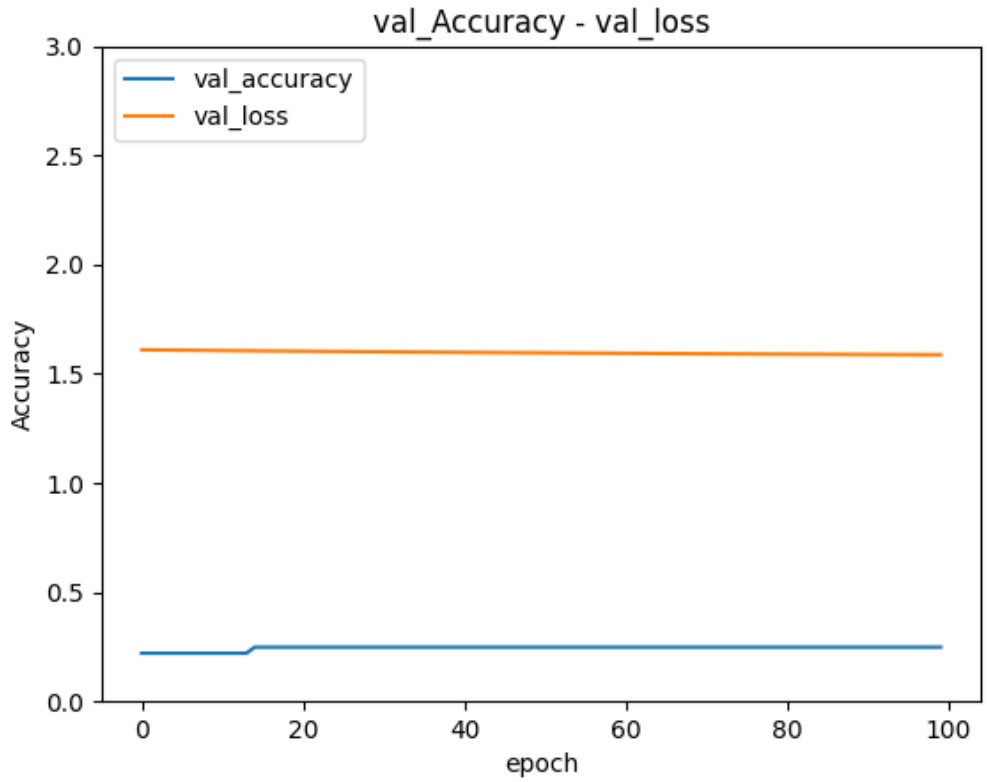
A modell statisztikáit a 3.15. ábra, konfúziós mátrixát a 3.16. ábra, validációs pontosságát és veszteségét a 3.17. ábra mutatja. Mindezekből látható, hogy a kis méretű rétegek alkalmazása nem célravezető, mivel lassan, és rosszul tanul a modell. A tanulás hossza 100 epoch-ban volt maximalizálva, amit nem került megszakításra korai megállással.

Epoch	Tanítási pontosság	Validációs pontosság	Teszt pontosság	Tanítható paraméterek száma
100	24.76%	24.83%	17%	2,480,277

3.15. ábra Első saját architektúra statisztikái

	Amur	Busa	Harcsa	Ponty	Sügér
Amur	0	0	0	24	0
Busa	0	0	0	48	0
Harcsa	0	0	0	62	0
Ponty	0	0	0	40	0
Sügér	0	0	0	60	0

3.16. ábra Első saját architektúra konfúziós mátrixa



3.17. ábra Első saját architektúra validációs adatai

3.2.4 Saját Architektúra 2

Az előző modellem sikertelensége alapján, egy komplexebb modellel próbálkoztam. Ez a modell az AlexNet [19] alapján készült.

Az első, bemeneti réteg 227x227 méretű képeket vár bemenetként és 96 darab 11x11 filtert tartalmaz, a stride értéke 4. Ezt követi egy batch normalization réteg, majd egy maximum-pooling 3x3 kernellel és 2 stride értékkel. A második konvolúciós réteg 256 darab 5x5 kernelből áll, stride értéke 1 és same paddinget alkalmaz. Ezt ismét a batch normalization és maximum pooling blokk követi, a fentebb már említett beállításokkal. Egy három rétegű konvolúciós blokk következik, melyben a kernelek száma rendre 512, 256, 128. A kernelek mérete 3x3, stride értéke 1, padding módja same. A blokkot egy maximum-pooling követi, 3x3 kernellel és 2 stride értékkel. Ezt egy flatten réteg követi, 0.2 dropout alkalmazásával. Az utolsó két réteg teljesen összekötött, rendre 2048 majd 1024 neuronnal, és 0.2 dropout értékkel. A kimenetért ismét egy FC réteg felelős, softmax aktivációval. A tanulás 41 epoch után korai megállással megszakításra került.

A modell elkészítéséhez tartozó kódrészlet a 3.18. ábraán, a modell összefoglalása a 3.19. ábraán, a modell sematikus struktúrája pedig a 3.20. ábraán látható.

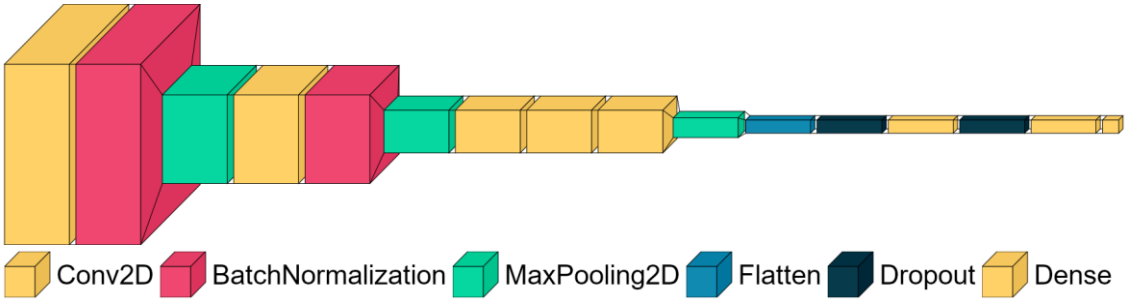
```
def create_model():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(filters=96, kernel_size = (11,11), strides= (4, 4), activation='relu',input_shape=(227,227,3)),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D(pool_size=(3, 3), strides= (2, 2)),
        tf.keras.layers.Conv2D(filters=256, kernel_size=(5, 5),
                               strides=(1, 1), activation="relu",
                               padding="same"),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D(pool_size=(3, 3), strides= (2, 2)),
        tf.keras.layers.Conv2D(filters=512, kernel_size=(3, 3),
                               strides=(1, 1), activation="relu",
                               padding="same"),
        tf.keras.layers.Conv2D(filters=256, kernel_size=(3, 3),
                               strides=(1, 1), activation="relu",
                               padding="same"),
        tf.keras.layers.Conv2D(filters=128, kernel_size=(3, 3),
                               strides=(1, 1), activation="relu",
                               padding="same"),
        tf.keras.layers.MaxPool2D(pool_size=(3, 3), strides= (2, 2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(2048, activation='relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(1024, activation='relu'),
        tf.keras.layers.Dense(num_classes,activation="softmax")
    ])

    return model
cnn_model = create_model()
optimizer = tf.keras.optimizers.SGD(learning_rate=0.001)
cnn_model.compile(optimizer=optimizer, loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics=['accuracy'])
if TRAIN:
    history = cnn_model.fit(train_dataset, epochs=100, validation_data=validation_dataset,
                            verbose=2,callbacks=[es,model_checkpoint_callback,model_checkpoint_callback_loss])
```

3.18. ábra A Saját Architektúra 2. CNN elkészítéséhez tartozó kódrészlet

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 55, 55, 96)	34944
batch_normalization (BatchNormalization)	(None, 55, 55, 96)	384
max_pooling2d (MaxPooling2D)	(None, 27, 27, 96)	0
conv2d_1 (Conv2D)	(None, 27, 27, 256)	614656
batch_normalization_1 (BatchNormalization)	(None, 27, 27, 256)	1024
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 256)	0
conv2d_2 (Conv2D)	(None, 13, 13, 512)	1180160
conv2d_3 (Conv2D)	(None, 13, 13, 256)	1179904
conv2d_4 (Conv2D)	(None, 13, 13, 128)	295040
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 128)	0
flatten (Flatten)	(None, 4608)	0
dropout (Dropout)	(None, 4608)	0
dense (Dense)	(None, 2048)	9439232
dropout_1 (Dropout)	(None, 2048)	0
dense_1 (Dense)	(None, 1024)	2098176
dense_2 (Dense)	(None, 5)	5125

Total params: 14,848,645
Trainable params: 14,847,941
Non-trainable params: 704

3.19. ábra A Saját Architektúra 2. CNN rétegei és azok paraméterei



3.20. ábra A Saját Architektúra 2. CNN sematikus struktúrája

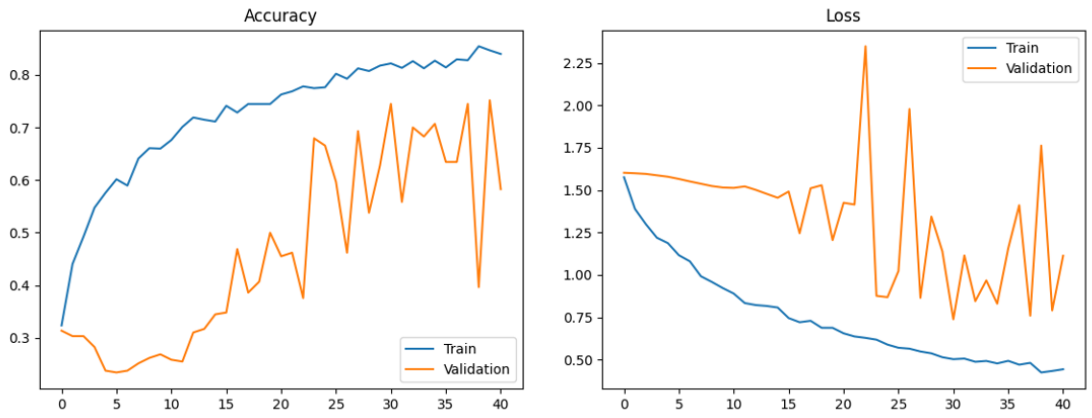
A modell statisztikáit a 3.21. ábra, konfúziós mátrixát a 3.22. ábra, validációs pontosságát és veszteségét ábra mutatja. A modell statisztikáin is látható, hogy ez jó teljesítményt ért el a teszt adatokon, a halak nagy részét a megfelelő osztályba sorolta. Az a probléma, hogy az amur osztályba tartozó halakat félreosztályozza a kevés tanuló minta miatt, továbbra is fennáll.

Epoch	Tanítási pontosság	Validációs pontosság	Teszt pontosság	Tanítható paraméterek száma
41	83.9%	58.28%	68.8%	14,848,645

3.21. ábra Második saját architektúra statisztikái

	Amur	Busa	Harcsa	Ponty	Sügér
Amur	1	8	6	7	2
Busa	0	36	0	12	0
Harcsa	0	5	50	1	6
Ponty	0	1	4	25	10
Sügér	0	3	3	5	49

3.22. ábra Második saját architektúra konfúziós mátrixa



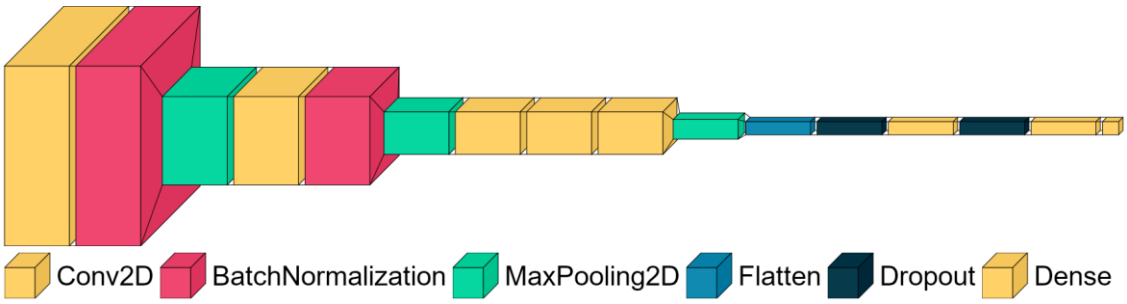
3.23. ábra Második saját architektúra pontossága és vesztesége

3.2.5 Saját architektúra 3

A harmadik általam készített architektúra a második továbbfejlesztése. Célja az volt, hogy megpróbáljon javítani a validációs és a teszt pontosságon jelentős számítási igény növekedés nélkül. Felépítése a következő:

A bemeneti konvolúciós réteg 128 darab 11x11 kernelt tartalamaz, a stride értéke 4. Ezt az előző hálózatban használt batch normalization és maximum pooling blokk követi. A második konvolúciós réteg 512 darab 5x5 kernelt tartalmaz, a stride értéke 1. Ezt ismételten normalizáció és pooling követi, majd egy három rétegű konvolúciós blokk, ami két 256 darab 3x3 méretű kernelt tartalmazó rétegből áll és egy 128 darab 3x3 kernelt tartalmazóból. Mindhárom rétegben a stride értéke 1, valamint a padding módja same. Ezt egy maximum pooling követi, majd egy flatten réteg után három teljesen összekötött réteg, 0.2 dropouttal, ezek rendre 2048, 1024 és 5 neuront tartalmaznak. A konvolúciós és teljesen összekötött rétegek ReLu aktivációt használnak, az utolsó FC réteg kivételével, ami softmax.

A modell elkészítéséhez tartozó kódrészlet a 3.18. ábraán, a modell összefoglalása a 3.19. ábraán, a modell sematikus struktúrája pedig a 3.20. ábraán látható.



3.24. ábra Harmadik saját modell felépítése

```
def create_model():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(filters=128, kernel_size = (11,11), strides= (4, 4), activation='relu',input_shape=(227,227,3)),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D(pool_size=(3, 3), strides= (2, 2)),
        tf.keras.layers.Conv2D(filters=512, kernel_size=(5, 5),
                               strides=(1, 1), activation="relu",
                               padding="same"),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D(pool_size=(3, 3), strides= (2, 2)),
        tf.keras.layers.Conv2D(filters=256, kernel_size=(3, 3),
                               strides=(1, 1), activation="relu",
                               padding="same"),
        tf.keras.layers.Conv2D(filters=256, kernel_size=(3, 3),
                               strides=(1, 1), activation="relu",
                               padding="same"),
        tf.keras.layers.Conv2D(filters=128, kernel_size=(3, 3),
                               strides=(1, 1), activation="relu",
                               padding="same"),
        tf.keras.layers.MaxPool2D(pool_size=(3, 3), strides= (2, 2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(2048, activation='relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(1024, activation='relu'),
        tf.keras.layers.Dense(num_classes,activation="softmax")
    ])

    return model
cnn_model = create_model()
optimizer = tf.keras.optimizers.SGD(learning_rate=0.001)
cnn_model.compile(optimizer=optimizer, loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics=['accuracy'])
if TRAIN:
    history = cnn_model.fit(train_dataset, epochs=100, validation_data=validation_dataset,
                            verbose=2,callbacks=[es,model_checkpoint_callback,model_checkpoint_callback_loss])
```

3.25. ábra A Saját Architektúra 3. CNN elkészítéséhez tartozó kódrészlet

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 55, 55, 128)	46592
batch_normalization (BatchNormalization)	(None, 55, 55, 128)	512
max_pooling2d (MaxPooling2D)	(None, 27, 27, 128)	0
conv2d_1 (Conv2D)	(None, 27, 27, 512)	1638912
batch_normalization_1 (BatchNormalization)	(None, 27, 27, 512)	2048
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 512)	0
conv2d_2 (Conv2D)	(None, 13, 13, 256)	1179904
conv2d_3 (Conv2D)	(None, 13, 13, 256)	590080
conv2d_4 (Conv2D)	(None, 13, 13, 128)	295040
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 128)	0
flatten (Flatten)	(None, 4608)	0
dropout (Dropout)	(None, 4608)	0
dense (Dense)	(None, 2048)	9439232
dropout_1 (Dropout)	(None, 2048)	0
dense_1 (Dense)	(None, 1024)	2098176
dense_2 (Dense)	(None, 5)	5125

Total params: 15,295,621
Trainable params: 15,294,341
Non-trainable params: 1,280

3.26. ábra A Saját Architektúra 3. CNN rétegei és azok paraméterei

A modell statisztikáit a 3.21. ábra, konfúziós mátrixát a 3.22. ábra, validációs pontosságát és veszteségét a 3.29. ábra Harmadik saját architektúra pontossága és vesztesége mutatja. A modell statisztikai adatain is látható, hogy az előző verzióhoz képest pontosságban kis

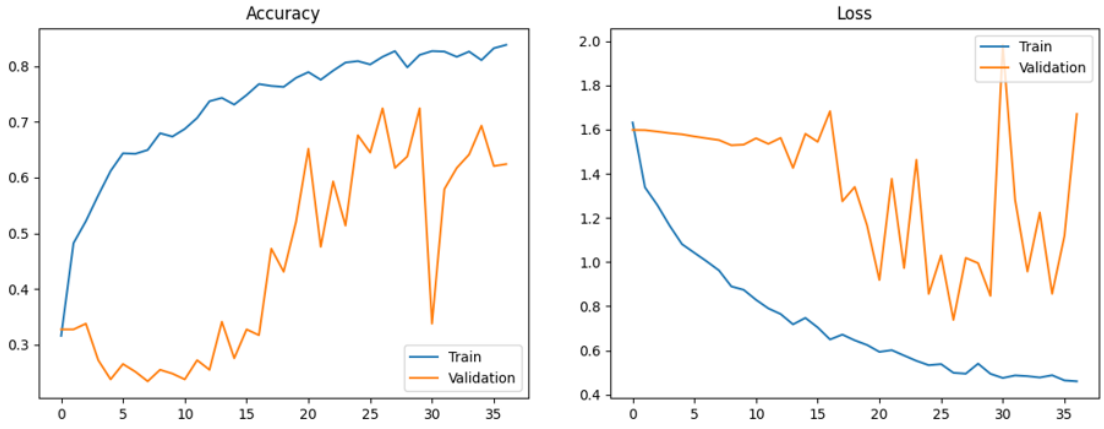
mértékű javulást hoztak a változtatások, viszont a tanítás hossza csökkent, annak ellenére, hogy több tanítható paramétert tartalmaz.

Epoch	Tanítási pontosság	Validációs pontosság	Teszt pontosság	Tanítható paraméterek száma
37	83.8%	62.41 %	69.23%	15,294,341

3.27. ábra Harmadik saját architektúra statisztikái

	Amur	Busa	Harcsa	Ponty	Sügér
Amur	2	8	11	3	0
Busa	0	42	0	6	0
Harcsa	1	4	55	1	1
Ponty	3	2	6	22	7
Sügér	0	4	21	5	41

3.28. ábra Harmadik saját architektúra konfúziós mátrixa



3.29. ábra Harmadik saját architektúra pontossága és vesztesége

3.30. ábra Saját Architektúra 3. CNN által predikált és valós osztályok

A 3.30. ábra látható néhány példa a modell predikciói közül. Látható, hogy a konfúziós mátrixhoz (3.28. ábra) hasonlóan az amur osztályba tartozó halakat nem tudta kellő pontossággal osztályozni, ez betudható a tanító halmaz kis méretének.

A tanító halmazban több olyan kép szerepel, amin a harcsákat egy sötét színű ponyván fényképezték le, ami egy bevett szokás a horgászok körében. Látható, hogy előfordult az, hogy egy ponty is ilyen ponyva felett lett fényképezve, így azt harcsának osztályozta a modell, ez által következtethetünk kis mértékű túltanulásra.

A 3.30. ábra látható képek között előfordul még néhány félreosztályozás, ami szabad szemmel is jól elkülöníthető halfajok között történt, ez arra utal, célszerű lenne a modellt

módosítani, esetleg új kép előkészítő technikák használatával, és több mintát használni a tanításhoz.

3.3 Modellek összehasonlítása

Modell	Epoch	Tanulás	Validáció	Teszt	Paraméterek
InceptionV3	9	87.15%	88.62%	66.6%	1,221,893
ResNet50	9	23.14%	23.45%	25.51%	1,221,893
Saját 1	100	24.76%	24.83%	24.83%	2,480,277
Saját 2	41	83.9%	58.28%	68.8%	14,848,645
Saját 3	37	83.8%	62.41 %	69.23%	15,294,341

3.31. ábra Modellek teljesítménye összesítve

A 3.31. ábra mutatja a modellek összehasonlítását, amelyből látható, hogy a legjobban teljesítő modell az általam készített Saját 3 nevű modell volt, míg a legrosszabbul teljesítő a Saját 1 modell volt. Látható, a tanulandó paraméterek száma növekedett a két modell között, viszont a tanításhoz szükséges epoch-ok száma csökkent, így nagyjából ugyanannyi számítási kapacitást igényelt mindkettő. Ez a jelenség látható a Saját 2 és Saját 3 modell között is, viszont ott a különbségek még kisebbek.

Az InceptionV3 modell a teszt minták nagy részét sikeresen osztályozta, viszont ennél a modellnél a tanítás bonyolult és időigényes, hiszen ez egy komplex modell. A tanulandó paraméterek alacsony számát az magyarázza, hogy amikor az én adathalmazaimon tanítottam a modellt, akkor csak az eredeti modell kimenetét követő teljesen összekötött rétegek tanultak, ez igaz a ResNet50 modellre is.

A táblázatból és a modellek jellemzésél látható konfúziós mátrixokról leolvasható az is, hogy a ResNet50 és a Saját 1 modell teljesen alkalmatlan ennek a problémának a megoldására, mivel az összes teszt példát egy osztályba sorolták.

3.4 Továbbfejlesztési lehetőségek

Feladatmegoldásom során a modell továbbfejlesztésén és egy telefonos alkalmazás elkészítésén is gondolkoztam, de ez idő hiányában nem került megvalósításra.

A modell könnyedén tovább fejleszthető új osztályok, azaz több halfaj bevonásával, ehhez viszont több képet kell gyűjteni, amik manuálisan ellenőrzöttek így, jó osztályokba kerülnek. Ez a probléma megoldható a tervezett telefonos alkalmazás segítségével, ahol a felhasználónak lehetősége lesz feltölteni a képeket. Elegendő mennyiségű új minta gyűjtése után a modell tovább tanítható, viszont amennyiben új osztályt kívánunk hozzáadni a modellhez, azt újra kell tanítani néhány elengedhetetlen módosítás után. Ilyenek például: Az osztályok számának átállítása, a kimeneti FC réteg neuronjainak száma. Ezeknek a módszereknek a folyamatos alkalmazásával elérhető az, hogy a neurális hálózat képes legyen akár a Magyarországon fellelhető összes halfaj helyes osztályozására.

A telefonos alkalmazás további funkciója az lenne, hogy a felhasználó az alkalmazás segítségével képet készít a felismerendő halról, amit az alkalmazás elküld egy szervernek vagy akár valamilyen felhőben futó szolgáltatásnak (például: AWS Lambda), ami válaszul visszaküldi a felhasználónak azt, hogy az adott hal melyik osztályba tartozik, és ennek a predikciónak a pontosságát. Hasznos funkció lehet a modell működésének elemzéséhez az, hogy a felhasználó értékeli a modellt, például egy félreosztályozás esetén elküldi a helyes osztályt, így statisztika készíthető arról, hogy mely halfajokat osztályozza félre a modell.

Az ebben a fejezetben levő technikákról és megvalósítási lehetőségekről csak gondolkodtam, részletes kutatást és tervezést igényel a hálózat teljesítményének további javítása és egy esetleges telefonos alkalmazás fejlesztése.

4. ÖSSZEFOGLALÓ

Szakdolgozatomban néhány gyakori magyar halfaj osztályozásához készítettem több neurális hálózatot, amely képes akár a kifogás helyszínén is beazonosítani az adott halat. Már a kutatómunka elején felismertem, hogy a feladat megoldásához szükséges a különböző gépi tanulási technikák, azon belül a felügyelt tanuláshoz használt módszerek ismerete.

Az egyszerűbb gépi tanulási módszerek, majd a neurális hálózatok, és azok különböző rétegjeinek működésének megismerése után, választásom a konvolúciós neurális hálózatokra (CNN) esett, mivel képek osztályozására ez bizonyult legmegfelelőbb módszernek.

A képek begyűjtése során problémát jelentett elegendő mennyiségű jó minőségű kép begyűjtése, így online elérhető forrásokból kellett a képek nagy részét beszereznem. A mennyiség még így sem volt elégséges, így különböző augmentációs technikákat kellett használnom a képek számának mesterséges növeléséhez. A tanító halmaz csak részben tartalmaz felügyelt körülmények között készült képeket, sok kép tartalmaz zavaró tárgyakat, különböző háttereket, ezért az elkészített modellek hajlamosak voltak a túltanulásra, aminek a mértékét különböző technikák segítségével próbáltam csökkenteni, viszont a teszt példáknál még így is előfordult ebből fakadó félreosztályozás.

A megoldás során kipróbáltam néhány más által készített és széles körben elismert képfelismerésre specializált neurális hálózatot, majd én is elkészítettem többet. Minden modellről készítettem különböző statisztikákat, majd értékeltem a kapott eredményeket. A statisztikák alapján a kipróbált neurális hálózatok közül a probléma megoldására a megvalósított saját architektúráim közül a harmadik, egyben utolsó megoldásom bizonyult a legjobbnak.

5. SUMMARY

In my thesis, I have developed several neural networks for the classification of some common Hungarian fish species, which are capable of identifying the fish even at the location of capture. At the beginning of the research, I recognized that knowledge of various machine learning techniques, particularly supervised learning methods, is necessary to solve the task.

After familiarizing myself with simpler machine learning methods, then neural networks and their different layers, I chose convolutional neural networks (CNNs) because they proved to be the most suitable method for image classification.

During the collection of images, it was a challenge to gather sufficient quantity of good-quality images; therefore, I had to obtain most of the images from online sources. Even then, the quantity was not sufficient, so I had to use various augmentation techniques to artificially increase the number of images. The training set only partially contained images taken under controlled conditions; many images contained distracting objects and different backgrounds, causing the trained model to overfit. I attempted to reduce the extent of overfitting using various techniques, but misclassification due to this issue still occurred in the test samples.

In the course of the solution, I tried several neural networks developed by others and widely recognized for image recognition, and then I developed several myself. I compiled various statistics for each model and evaluated the results. Based on the statistics, among the neural networks I tried, the third and final solution I implemented from my own architectures proved to be the most effective for solving the problem.

6. IRODALOMJEGYZÉK

[1] S. Russell és P. Norvig, Artifial Intelligence. A Modern Approach, Pearson; 4th edition, 2020.

[2] J. Han és M. Kamber, Adatbányászat. Koncepciók és technikák, Budapest: Panem, 2024.

[3] R. Farkas, „Gépi tanulás a gyakorlatban," [Online]. Available: https://www.inf.u-szeged.hu/~rfarkas/ML20/MLintro.html. [Hozzáférés dátuma: 17 03 2024].

[4] „ChatGPT," OpenAI, [Online]. Available: https://openai.com/chatgpt. [Hozzáférés dátuma: 15 április 2024].

[5] L. V. Ferreira, E. Kaszkurewicz és A. Bhaya, „Support vector classifiers via gradient systems with discontinuous righthand sides," *Neural Networks,* 19. kötet, 10. szám, pp. 1612-1623, 2006.

[6] I. Fazekas, „Neurális hálózatok," Debreceni Egyetem, 2013. [Online]. Available: https://gyires.inf.unideb.hu/GyBITT/19/index.html. [Hozzáférés dátuma: 12 11 2023].

[7] M. Altrichter, G. Horváth, B. Pataki, G. Strausz, G. Takács és J. Valyon, Neurális hálózatok, Budapest: Panem Könyvkiadó Kft., 2006.

[8] I. Moumene és N. Ouelaa, „Gears and bearings combined faults detection using optimized wavelet packet transform and pattern recognition neural networks," *The International Journal of Advanced Manufacturing Technology,* 120. kötet, pp. 1-20, 2022.

[9] R. Yamashita, M. Nishio, R. K. G. Do és K. Togashi, „Convolutional neural networks: an overview and application in radiology," *Insights into Imaging, 9.* kötet, 4. szám, pp. 611-629, 2018.

[10] „Computer Vision," Stanford Artificial Intelligence Laboratory, [Online]. Available: https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html. [Hozzáférés dátuma: 28 április 2024].

[11] J. Courtney, D. Burke és A. dePaor, „Application of Digital Image Processing to Marker-free Analysis of Human Gait," *Measurement Science Reviey,* 1. kötet, 1. szám, 2001.

[12] „Keras," [Online]. Available: https://keras.io/. [Hozzáférés dátuma: 28 április 2024].

[13] „Tensorflow," [Online]. Available: https://www.tensorflow.org/. [Hozzáférés dátuma: 28 április 2024].

[14] „Colab," Google, [Online]. Available: https://colab.research.google.com/. [Hozzáférés dátuma: 28 04 2024].

[15] „iNaturalist," 31. március 2024. [Online]. Available: https://hu.wikipedia.org/wiki/INaturalist. [Hozzáférés dátuma: 26 04 2024].

[16] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens és Z. Wojna, „Rethinking the Inception Architecture for Computer Vision," 2016.

[17] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever és R. Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research,* 15. kötet, pp. 1929-1958, 2014.

[18] K. He, X. Zhang, S. Ren és J. Sun, „Deep Residual Learning for Image Recognition," Microsoft Research, 2015.

[19] A. Krizhevsky, I. Sutskever és G. E. Hinton, „ImageNet Classification with Deep Convolutional Neural Network," in *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, 2012.

7. ÁBRAJEGYZÉK