

TDK-dolgozat

Szerző:

Bódi Vencel

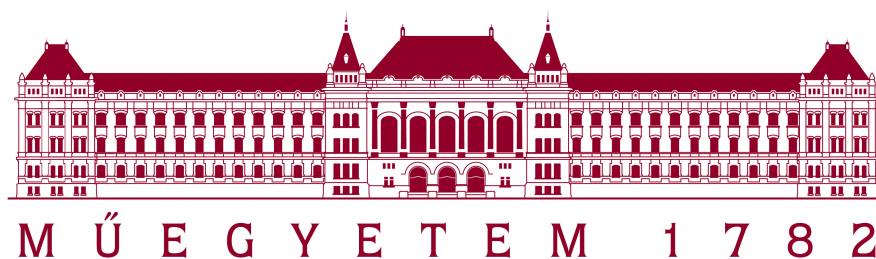
MCTS alapú batch-sorrend optimalizálás számítógépes látás támogatásához

Szerző:

Bódi Vencel

Másodéves BSc hallgató

2023/24-es évfolyam



Budapesti Műszaki és Gazdaságtudományi Egyetem

Közlekedésmérnöki és Járműmérnöki Kar

Közlekedés- és Járműirányítási Tanszék

Konzulens:

Kővári Bálint

Tudományos Segédmunkatárs

Országos Tudományos Diákköri Konferencia

Tartalomjegyzék

1. Motiváció	2
2. Monte Carlo Tree Search	4
2.1. Döntéselmélet	4
2.2. Döntési tér konstrukciója	6
2.3. Klasszikus fakesési algoritmusok	7
2.4. Áttekintés	8
2.5. Az algoritmus lépései	9
2.5.1. Selection - UCT Tree Policy	9
2.5.2. Expansion	11
2.5.3. Rollout - Default Policy	12
2.5.4. Backpropagation	13
3. Deep Learning modellek	14
3.1. Gépi tanulás, mélytanulás	14
3.2. Konvolúciós Neurális Hálózatok	16
3.3. A felügyelt tanulási folyamat	20
4. Környezet	23
4.1. Python	23
4.2. Pytorch	23
4.3. MNIST adatbázis	24
5. MCTS-alapú optimalizáció	25
5.1. MCTS alkalmazása	26
5.2. Metodológia	28
6. Eredmények	31
7. Konklúzió, kitekintés	34

1. fejezet

Motiváció

A számítógépes látásra épülő rendszerek egyre szélesebb körű alkalmazásai napjaink technológiai fejlődésének egyik legmeghatározóbb területévé váltak, melyek nem csupán az ipari automatizálásban, hanem az orvosi diagnosztikában, a közlekedésben és a biztonságtechnológiában is komoly előrelépéseket hoztak. A *Computer Vision* rendszerei az emberi látás funkcionalitását utánózva képesek értelmezni és feldolgozni a vizuális információkat, egyre pontosabb döntéseket hozva. Ez a fejlődés a mesterséges intelligencia és azon belül a *Deep Learning* (DL, mélytanulás) gyors ütemű innovációira vezethető vissza, különösen a *Konvolúciós Neurális Hálózatok* (CNN) alkalmazására, amelyek kiválóan alkalmasak képi adatok feldolgozására és különböző feladatok megoldására.

Az 1980-as évek végén, 1990-es évek elején megjelent CNN hálózatok [1] továbbra is a leginkább meghatározó technológiának számítanak, különösen a képfeldolgozás területén. Ezek a modellek az adatok térbeli hierarchikus szerkezetét hasznosítva képesek az alacsony szintű jellemzők felismerésétől kezdve az egyre bonyolultabb mintázatok és objektumok azonosítására. A CNN-ek eredményességét olyan modellek bizonyítják akár korlátozott elérhető számítási kapacitás esetén is, mint a *ResNet* és a *MobileNet*, amelyek hatékony architektúrákat kínálnak, például mobil eszközökön történő alkalmazásokhoz. Az olyan modellek, mint a *MobileNetV3* vagy a *ResNet50*, az elmúlt évek meghatározó mérföldkövei voltak a képfeldolgozás fejlődésének.

Azonban a CNN-alapú architektúrákra építve újabb deep learning modellek is nagy teret nyertek. A legújabb technológiai innovációk közé tartozik a *YOLO* (*You Only Look Once*) család, amely éveken keresztül rekordtartó a valós idejű objektumdetektálás területén [2]. Legújabb tagja, a *YOLOv10* [3], mára az egyik legkifinomultabb és leghatékonyabb detekcióra kifejlesztett modellé vált. A YOLO modellek különlegessége abban rejlik, hogy valós időben képesek objektumokat detektálni és osztályozni

egyetlen futás során [4], így késleltetésük elhanyagolható, ami rendkívül fontos az autonóm járművek és más valós idejű alkalmazások számára.

A hagyományos CNN-ek mellett egyre nagyobb figyelmet kapnak az új generációs modellek, mint például a *ConvNeXT*, amely továbbfejleszti a konvolúciós hálózatok hatékonyságát és pontosabb eredményeket kínál a képfeldolgozási feladatok során [5]. A ConvNeXT architektúra a CNN-ek evolúcióját jelképezi, ahol az újratervezett konvolúciós blokkok nagyobb rugalmasságot és teljesítményt biztosítanak, különösen nagy felbontású képek feldolgozásakor.

Ezzel párhuzamosan egy másik forradalmi megközelítés, a *Vision Transformer* (ViT), új irányokat nyitott a képfeldolgozásban [6]. A transformer-alapú modellek eddig főként a természetes nyelvfeldolgozásban mutattak kiemelkedő teljesítményt, ám a ViT megmutatta, hogy ezek a modellek a képi adatok feldolgozásában is igen kompetensek. A CNN-ekkel ellentétben, amelyek lokális jellemzők felismerésére fókuszálnak, a Vision Transformaterek a globális összefüggéseket is figyelembe veszik, ezáltal javítva a modell teljesítményét különböző feladatoknál. Bár a ViT modellek számítási igényei nagyobbak, skálázhatóságuk és általánosítási képességük figyelemre méltó.

Az ilyen modellek fejlődése, mint a YOLOv10, ConvNeXT és a Vision Transformer, jelentős áttöréseket hozott a computer vision és képfeldolgozás terén. Ezek a modellek mind a pontosság, mind a hatékonyság terén új rekordokat érnek el, ám gyakori korlátozó tényezőjük továbbra is a nagy adatigény és az időigényes tanítási folyamat. A modellek tanulása és finomhangolása általában hatalmas adatbázisokat és számítási erőforrásokat igényel, ami korlátot szab a rendszer további fejlesztésének és skálázhatóságának.

Jelen dolgozat célja, hogy egy olyan hiperparaméter-optimalizálási megközelítést mutasson be, amely lehetővé teszi a ML modellek, különösen a képfeldolgozási rendszerek hatékonyabb betanítását, ezáltal jelentősen csökkentve az adatigényt és javítva a modellek általános teljesítményét. Az alkalmazott módszer a Monte Carlo Tree Search használatával nemcsak a hálózatok pontosságát, hanem azok konvergenciájának sebességét is jelentősen megnövelik.

2. fejezet

Monte Carlo Tree Search

2.1. Döntéselmélet

A *döntéselmélet* egy olyan matematikai illetve logikai eszköztár, amely az egyének (ágensek) döntési folyamatait vizsgálja. Különösen arra összpontosít, hogy hogyan található meg az optimális döntések olyan helyzetekben, ahol különböző választások különböző kimenetekkel járhatnak. Az ágensek célja ebben a kontextusban a legjobb döntések meghozatala a rendelkezésre álló információk és lehetséges kimenetek ismerete alapján. Egy döntési probléma a következő paraméterekkel rendelkező keretrendszerrel modellezhető:

- S : az állapotok halmaza;
- s_0 : kezdő állapot;
- S_T : a terminális állapotok halmaza;
- A : az akciók halmaza;
- $A(s)$: azon akciók halmaza, amelyek az s állapotból elérhetők;
- $T(s, a) \rightarrow s'$: átmeneti függvény, s állapotból a akció hatására s' állapotba visz;
- $R(s) \rightarrow \mathbb{R}$: az állapotkiértékelési (jutalom) függvény.

Ezek alapján egy döntési probléma modellje az S állapotok halmazából felépülő térként írható le, amiben a kiindulási állapot s_0 , az A halmaz az összes akciók halmaza, míg az $A(s)$ halmaz kifejezetten az s állapotból elérhető akciókat jelöli. Az egyes állapotok közötti átmeneteket a $T(s, a) \rightarrow s'$ átmeneti függvény határozza meg, amely előírja, hogy egy adott s állapotból egy a akció végrehajtásával mely s' állapotba juthatunk el. A döntési problémának akkor lesz vége, ha az ágens az S_T terminális állapotok

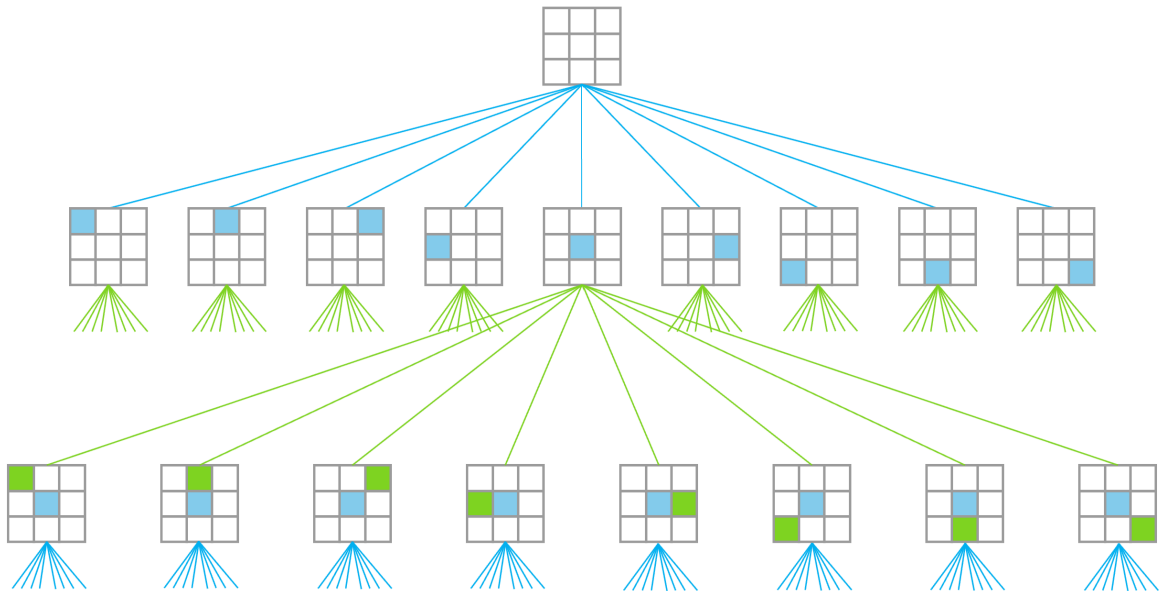
valamelyikébe eljut. Minden állapot kiértékelhető egy $R(S)$ függvénnyel, így lehetséges priorizálni a különböző állapotok között[7].

Ennek megfelelően kialakulnak szomszédos állapotok, amelyek közvetlenül elérhetők egy-egy akción keresztül. Kézenfekvő tehát, hogy ezt a teret gráfokkal vagy fastruktúrákkal (döntési fákkal) írjuk le.

Ebben az esetben a döntési fa csomópontjai az állapotokat, az élek pedig az akciókat jelképezik, amelyek az egyik állapotból a másikba visznek. A fa szerkezete lehetőséget biztosít arra, hogy hierarchikusan ábrázoljuk a döntési folyamatot, ahol az ágak az egyes döntések lehetséges kimeneteleihez, újabb állapotokhoz vezetnek.

A döntési fa első csomópontja (gyökér, root) a kezdő állapot (s_0), innen pedig az ágak az állapotok közötti átmeneteket (akciókat) reprezentálják. Az egyes csomópontokhoz rendelt állapotértékelési függvény ($R(S)$) segítségével megállapíthatjuk, hogy melyik állapot mennyire kedvező, és ezáltal priorizálhatjuk a kedvezőbb állapotokhoz vezető útvonalakat.

A fa leveleit a terminális állapotokkal (S_T) azonosíthatjuk. A fastruktúra további nagy előnye, hogy segít logikusan felépíteni a döntési folyamatot, és szemlélteti, hogy a döntéshozó (ágens) milyen lépéseken keresztül juthat el a kívánt végkimenetelig. Így, amennyiben célunk, hogy a legkedvezőbb terminális állapotot érnük el a fában, egyszerű fakeselési algoritmusokat hívhatunk segítségünkre.

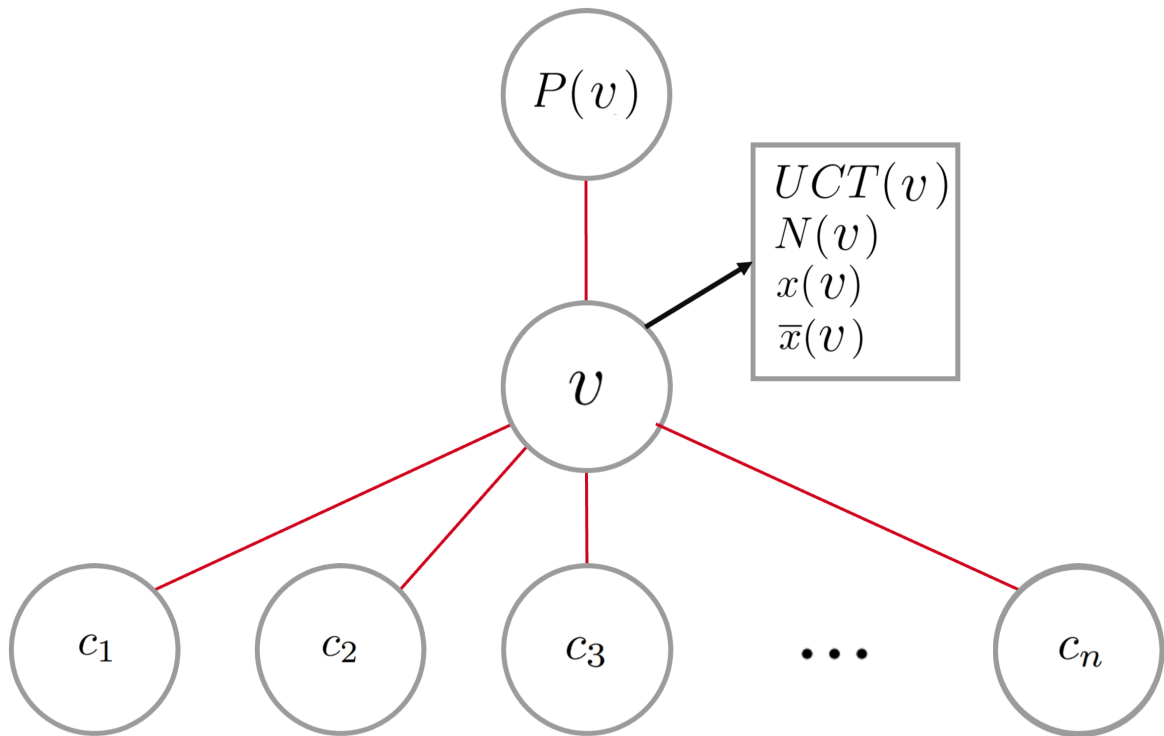


2.1. ábra. Az amőba döntési fájának (játékfájának) részlete

2.2. Döntési tér konstrukciója

A Monte Carlo Tree Search algoritmus a döntési tér különböző helyzeteit, azaz állapotait vizsgálja, majd az ezekben elérhető akciókat használja fel a döntési fa felépítéséhez, amelyben minden csomópont egy-egy állapotot reprezentál. Az állapotokból elérhető akciók szabják meg, hogy hogyan tudunk továbbjutni az adott állapotból egy újabb állapotba. Bár az állapotok és csúcsok lényegileg ugyan azt írják le, mégis érdemes különböző jelöléseket bevezetni a két struktúrára az áttekinthetőség kedvéért.

Az algoritmus működése során épített fa csomópontjaiban több adatot is tárolunk, ami segít leírni magát a reprezentált állapotot. A fa csomópontjait, amelyek egy állapotot reprezentálnak v -vel, így a gyökeret v_0 -lal, a v csúcsához társított állapotot pedig s_v -vel jelöljük. v látogatottsági száma $N(v)$, teljes pontszáma $x(v)$. Ezekből meghatározható $\bar{x}(v)$, a csúcs átlagos pontszáma. Egy csúcs által reprezentált állapotból elérhető akciók jelölése megmarad a korábbiakban ismertetett $A(v)$, az átmeneti függvény pedig $T(v, a)$, vizsgált csúcsba közvetlenül vezető akció $a(v)$. A csúcsok gyerekeinek halmazát $C(v)$ -vel, ebből az i -edik gyermeket c_i -vel jelöljük, a vizsgált csúcs szülő-csomópontja $P(v)$. A csomópontok későbbiekben bemutatott UCT pontszáma $UCT(v)$.



2.2. ábra. Csomóponti jelölések

2.3. Klasszikus fakeselési algoritmusok

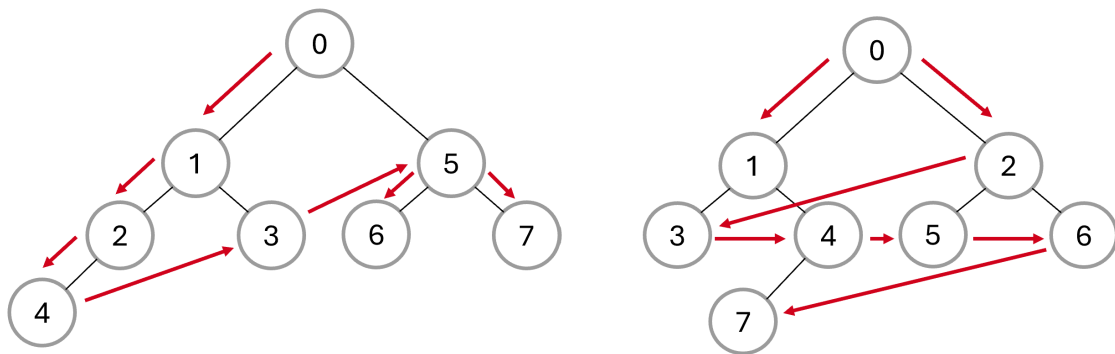
A fakeselési algoritmusok között két alapvető megközelítést különböztetünk meg: a *BFS* (Breadth-First Search, szélességi keresés) és a *DFS* (Depth-First Search, mélységi keresés) algoritmusokat. Ezek az algoritmusok alapvető szerepet játszanak a gráfok és fastruktúrák bejárásában, és mindkettőnek megvannak a maga előnyei és hátrányai.

BFS – Szélességi keresés

A BFS algoritmus a gráf csomópontjait rétegesen járja be, először a kezdőcsomópontához közvetlenül kapcsolódó csomópontokat vizsgálva, majd fokozatosan az egyre távolabbi csomópontokat. Ez az algoritmus különösen hasznos, ha a legrövidebb utat keressük egy adott csomóponthoz egy gráfban. A keresési algoritmusok között ez a leginkább memória-, illetve számításigényes, ugyanis a gráf összes csomópontját feltérképezi.

DFS – Mélységi keresés

A DFS ezzel szemben a gráf csomópontjait mélységi alapon járja be, azaz egy útvonalon halad addig, amíg el nem éri egy levélcsomópontot (vagy egy már látogatott csomópontot), majd visszalép, és egy másik elágazást próbál ki. Ez az algoritmus mélyebb keresésre fókuszál, és gyakran hasznos olyan problémák esetében, a leghosszabb út keresése a cél.

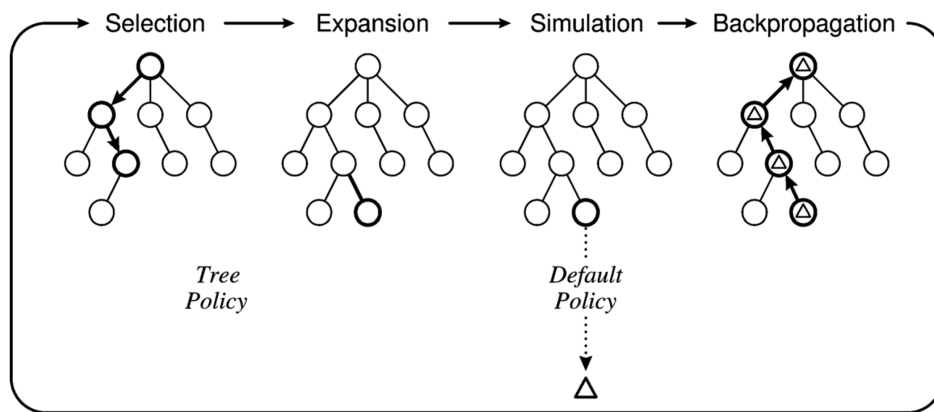


2.3. ábra. A BFS és DFS algoritmusok által előállított csomópont-sorrend

2.4. Áttekintés

A **Monte Carlo Tree Search** (MCTS) algoritmus egy iteratív döntésoptimalizációs módszer, amelyet gyakran alkalmaznak összetett döntési problémák és játékok megoldására. Az algoritmus kifejezetten jól teljesít olyan problémák esetén, ahol a döntési tér nagy, vagy komplex. Ezt úgy éri el, hogy egy döntési fát (gráfot) épít fel véletlenszerű szimulációk futtatásával, majd az egyes csúcsok kiértékelésével a következő iterációban a legkedvezőbb állapotok felé folytatja a keresést. Az MCTS négy fő lépésből (2.4) áll, amelyek minden iterációban végrehajtásra kerülnek:

- **Selection** (Kiválasztás)
- **Expansion** (Bővítés)
- **Rollout** (Simulation, Szimuláció)
- **Backpropagation** (Visszaterjesztés)



2.4. ábra. Az Monte Carlo Tree Search algoritmus lépései[8]

Ezzel a négy egyszerű lépéssel az MCTS fokozatosan egy asszimetrikus fát készít. Az algoritmus minden iterációjában egy *Tree Policy*-t követ, amely alapján mérlegel az *Exploration* (felfedezés, azoknak a területeknek, ágaknak a vizsgálata, amelyekből még nem vett elég mintát) és *Exploitation* (kiaknázás, azoknak a területeknek a vizsgálata, amelyeket ígéretesnek tart) között. Ezzel a szabállyal kiválasztja a jelenlegi fában a legígéretesebb csúcsot. Amennyiben a kiválasztott csúcsnak még nincsenek gyermekei, úgy az onnan végrehajtható akciók által elérhető állapotoknak megfelelően létrehozza a csúcs gyermekeit. Amennyiben a csúcsnak vannak gyermekei (vagy az előbb leírtak alapján az algoritmus létrehozta azokat), úgy azok közül a legígéretesebbet kiválasztja, és a *Default Policy* alapján (alapesetben véletlenszerű döntéseket hozva) egy végső állapotig egy *Rollout*-ot (végigjátszást, szimulációt) hajt végre. Ennek eredményét

visszaterjeszti a fában (*Backpropagation*), azaz a Rollout eredményét a vizsgált gyermek és minden szülő pontszámához hozzáadja, illetve azok látogatottsági számát eggyel megnöveli. Mikor az algoritmus a kívánt iteráción át futott megkaphatjuk a kezdőpozícióból (*Root*) elérhető legjobb döntést, döntéssorozatot. Az alábbiakban (1) látható az MCTS pszeudokódja:

1. Algorithm A Monte Carlo Tree Search egy iterációja

```

function MCTS( $s_0$ )
   $v_0 \leftarrow v : s_v = s_0$ 
  while in computational budget do
     $v' \leftarrow \text{SELECTION}(v_0)$ 
    if  $n(v') \neq 0$  then
       $v' \leftarrow \text{EXPAND}(v')$ 
     $p \leftarrow \text{ROLLOUT}(s_{v'})$ 
     $\text{BACKPROPAGATION}(v', p)$ 
  return a BESTCHILD( $v_0$ )

```

Ezzel az MCTS algoritmus hatékonyan képes a legkedvezőbb állapotok megtalálására akár igen nagy, illetve komplex állapotterekben. Egyik legnagyobb előnye, hogy alkalmazása esetén nem szükséges minden köztes (nem terminális) állapotot megvizsgálunk, mint például a mélységében maximalizált *Minimax* algoritmus esetén, így nagyban lecsökkenti a legkedvezőbb állapot megtalálásához szükséges számítási igényt.

Mára a Monte Carlo Tree Search a Machine Learning egyik kulcsfontosságú algoritmusává vált a többszemélyes játékok és döntési problémák terén. Előszeretettel alkalmazzák például a *Computer GO* területén, ugyanis a Go egy kifejezetten nagy állapotterrel rendelkező játék, azaz egy állapotból közvetlenül elérhető további állapotok száma nagy. Ezek miatt az MCTS jó eredményeket ér el Deep Learning használata mellett [9].

2.5. Az algoritmus lépései

2.5.1. Selection - UCT Tree Policy

A Selection lépés célja, hogy a döntési fában megtalálja a legígéretesebb csomópontot az aktuális iteráció során. Ebben a fázisban az algoritmus a gyökérpontból indul, majd lefelé haladva minden csúcsban kiválaszt egy gyermeket egy előre meghatározott szabály, a Tree Policy alapján. A Tree Policy segítségével az adott problémának megfelelően súlyozhatjuk az exploitációs és explorációs tagokat. A Selection algoritmus a következőképpen írható le:

2. Algorithm Selection algoritmus

```

function SELECTION( $v_0$ )
   $v \leftarrow v_0$ 
  while  $C(v) \neq \emptyset$  do
     $v \leftarrow \arg \max_{c_j \in C(v)} UCT(c_j)$ 

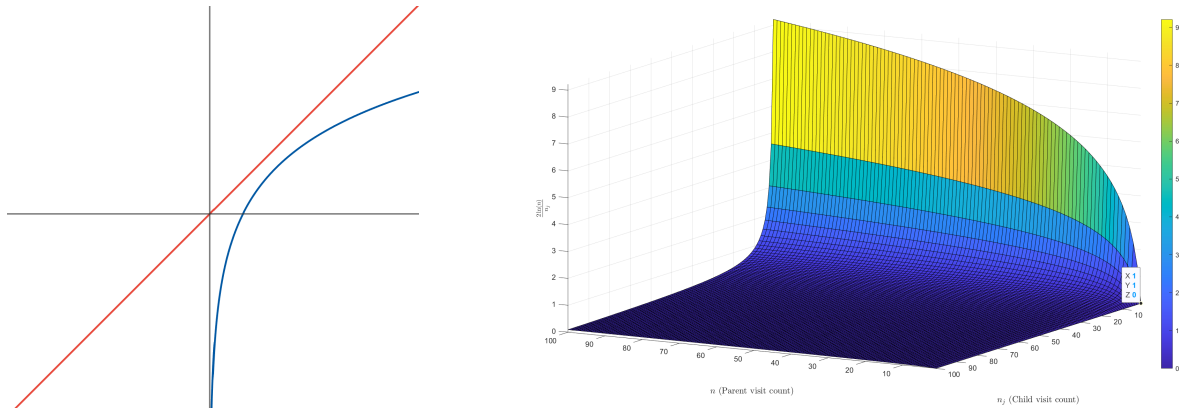
  return  $v$ 

```

Az egyik legelterjedtebb Tree Policy az úgy nevezett *UCT* (Upper Confidence Bound 1 applied for Trees), amelyet Kocsis Levente és Szepesvári Csaba mutattak be [10]. Ez matematikailag a következőképpen írható le:

$$UCT(v) = \bar{x}(v) + C_p \cdot \sqrt{\frac{2 \cdot \ln N(P(v))}{N(v)}}, \quad (2.1)$$

ahol C_p , konstans paraméter, amellyel az explorációs tag súlya állítható. A fenti kifejezésből látható, hogy az UCT Tree Policy figyelembe veszi a csúcs átlagos pontszámát, amely a visszaterjesztések során gyűlik össze, valamint a csúcs látogatottságát a szülőhöz viszonyítva (amennyiben $N(v)$ nulla, úgy az explorációs tag értékét végtelennek tekintjük).



2.5. ábra. A $2\ln N(P(v))$ és $N(v)$ tagok viszonya egymáshoz

Ahogy az UCT képletéből (2.1) és a fenti ábrákon (2.5) is látható, az explorációs tag értéke a szülő és az adott gyermek látogatottságának arányától függ. Ha a gyermek látogatottsága közel azonos a szülőéhez, akkor a tört kisebb értéket vesz fel, míg nagyobb különbség esetén a tört értéke növekszik. Ezek alapján belátható, hogy az UCT politika előnyben részesíti a kevésbé látogatott csúcsokat a fában. Az explorációs tag C_p szorzótényezőjének bevezetésével dinamikusan állíthatóvá tesszük a kevésbé látogatott csúcsok preferenciájának mértékét. Ennek következtében lehetőségünk nyílik az UCT Tree Policy finomhangolására az adott problémának megfelelően. Amennyiben

egy probléma a csúcsok alaposabb megvizsgálását követeli meg, úgy ez megtehető a C_p konstans növelésével, amennyiben az ellenkezőre van szükség, úgy annak csökkentésével.

Továbbá érdemes megjegyezni, hogy az UCT plolitika képes átmenetet teremteni az előző fejezetekben bemutatott BFS és DFS algoritmusok között az exploitációs és explorációs tagok súlyozásával. Amennyiben csak az exploitációs tagot vesszük figyelembe, úgy könnyen belátható, hogy a DFS algoritmushoz hasonló működést kapunk, a döntési folyamatok során csupán az egyes csúcsok pontszáma lesz behatással a választásra. Így mindig az elsőként létrehozott gyermeket fogja figyelembe venni az algoritmus, így célja az lesz, hogy minél hamarabb a lehető legmélyebbre jusson a fában. Amennyiben az explorációs tag kerül előtérbe, az algoritmus a BFS-re emlékeztet, amely minden csúcsot fel akar térképezni a fa gyökérpontjától lefelé, a teljes állapotteret lefedve.

A Tree Policy a Monte Carlo Tree Search egyik legfontosabb eleme, amely lehetővé teszi az algoritmust, hogy a problémának megfelelő tökéletes átmenetet teremtsen meg a két alapvető fakeseresési algoritmus, a DFS és BFS között, ezáltal a döntési tér kiaknázása és felfedezése között. Ezáltal az algoritmus egyaránt kiválóan alkalmazható olyan problémákra is, amelyek vagy inkább a mélységi, vagy inkább a szélességi keresést igénylik.

2.5.2. Expansion

Az *Expansion* lépés során a Monte Carlo Tree Search algoritmus a döntési fa bővítését végzi. Miután a Selection fázis során az algoritmus kiválasztotta a legígéretesebb csúcsot, úgy a fát bővíteni kell ¹. Ilyenkor a lehetséges akciókat megvizsgálva megállapítjuk, hogy milyen állapotokba juthatunk el, majd létrehozuk az azoknak megfelelő gyermek csúcsokat².

Az egyes csúcsokból elérhető gyermekek maximális számát az úgynevezett *branching factor* (elágazási tényező) határozza meg. Ez azt mutatja meg, hogy az adott csomópontból hány különböző akció által érhetünk el új állapotokat (a csúcsok maximális fokszáma). Amennyiben a branching factor nem maximális (nem egyezik az elérhető akciók számával), úgy az akciókat véletlenszerűen választjuk ki a lehetséges akciók közül. A magas branching factor nagyobb keresési teret eredményez, ugyanis így minden egyes csomópont gyermekeinek számát megnöveljük, és így több lehetséges állapotot

¹Amennyiben az nem terminális állapot, illetve nem rendelkezik gyermek csomópontokkal.

²A gyakorlatban az is előfordul, hogy egyszerre csak egy gyermeket hozunk létre, majd a következő iterációban - amennyiben az algoritmus szerint továbbra is ez a legígéretesebb csús - a következőt

3. Algorithm Expansion algoritmus

```

function EXPANSION( $v$ )
  pick  $a \in A(v)$  that has not been used before
  add new child  $c$  to  $C(v)$  with:
     $s_c \leftarrow T(s_v, a)$ 
     $A(c) \leftarrow A(v) \setminus a$ 
     $a(c) \leftarrow a$ 
     $P(c) \leftarrow v$ 
  return  $c$ 

```

kell végigjátszani, majd kiértékelni. Ez jelentős számítási igény növekedéshez vezet, ugyanakkor igen hatásosan javítani tudja az algoritmus teljesítményét. Alacsonyabb branching factor esetén az algoritmus hamarabb eljuthat egy terminális állapotban, ugyanakkor csökken az esélye, hogy megtalálja az optimális megoldást a problémára.

A branching factor értéke az adott probléma jellegétől függ, és alapvetően befolyásolja az MCTS hatékonyságát. Alacsonyabb elágazási tényezőnél az algoritmus gyorsabban elérheti a terminális állapotokat, míg nagyobb mellett fajlagosan több explorációra van szüksége. Ugyanakkor ez nem jelenti azt, hogy a magasabb branching factor hátrányos lenne, az algoritmus az exploitation és exploration tagok megfelelő hangolásával nagy és komplex keresési térben is képes hatékonyan működni.

2.5.3. Rollout - Default Policy

A Monte Carlo Tree Search *Rollout* fázisa egy szimulációs folyamat, amelyben az algoritmus a használt Tree Policy alapján a legígéretesebbnek tartott csúcsból (amely nem egy terminális állapotot jelképez) egy Default Policy alapján döntéseket hozva eljut egy terminális állapotba, vagy egy meghatározott mélységig folytatja a keresést.

A leggyakrabban alkalmazott Default Policy véletlenszerű döntésekből áll: a rollout során a kiválasztott csúcsból az algoritmus addig hoz véletlenszerű döntéseket, amíg egy terminális állapotot el nem ér. Ezek után ez a terminális állapot kiértékelésre kerül, majd ez fogja megadni a kiválasztott csúcs pontszámát, amelyet a *Backpropagation* fázisban visszaterjeszt a fában. A rollout fázis kiemelt szerepet játszik az MCTS hatékonyságában, mivel lehetőséget biztosít arra, hogy az algoritmus becslést készítsen a fa adott csúcsa alatt található döntési utak várható eredményéről. Ezáltal a rollout szimulációi heurisztikus úton nyújtanak fontos információkat az adott csúcs hosszú távú várható eredményeiről. Fontos kiemelni, hogy a fa struktúráját a rollout alatt nem változtatjuk, azaz a véletlenszerűen kiválasztott akciók által elért csomópontokat "elfelejtjük" a rollout után, nem adjuk hozzá a fához, csupán az eredményt jegyezzük meg.

4. Algorithm Rollout algoritmus

```

function ROLLOUT( $v$ )
  while  $s_v \notin S_T$  do
     $a \leftarrow$  random choice from  $A(v)$ 
     $v \leftarrow T(v, a)$ 
  return  $R(s_v)$ 

```

A szimuláció során alkalmazott Default Policy erősen befolyásolja a rollout megbízhatóságát. A véletlenszerű döntések használata egy egyszerű, általános, ugyanakkor kimondottan jó eredményre vezető módszer. Néhány döntési tér esetén ugyanakkor érdemes probléma-specifikus Default Policy-t készíteni a hatékonyabb működésért. Ilyen például a fentebb említett Go játék, ahol gyakran előfordulnak visszatérő lépéssorozatok, így érdemes előtérbe helyezni ezeket a mozgási mintákat.

2.5.4. Backpropagation

A *Backpropagation* Monte Carlo Tree Search utolsó lépése egy iteráció során. Célja, hogy a rollout során megszerzett információkat a gyökér felé az egész fában terjessze, hogy a következő iterációban a selection algoritmus a friss információk alapján legyen képes megállapítani a legígéretesebb csomópontot. Ennek értelmében az érintett csúcsok pontszámához hozzáadjuk a rollout eredményét, majd látogatottsági számukat eggyel megnöveljük.

5. Algorithm Backpropagation algoritmus

```

function BACKPROPAGATION( $v : s_v \in S_T, R(s_v)$ )
   $S \leftarrow s_v$ 
  while  $P(v) = \emptyset$  do
     $x(v) \leftarrow x(v) + R(S)$ 
     $N(v) \leftarrow N(v) + 1$ 
    BACKPROPAGATION( $P(v), R(S)$ )
  return

```

Érdemes megfigyelni, hogy a backpropagation egy rekurzív eljárás, amely a rollout kiindulási csomópontjából a gyökérig minden olyan csúcsot érint, amelyek szülő-gyermek viszonyban vannak egymással, ezzel minden iterációban egy-egy ágat érint a fában.

3. fejezet

Deep Learning modellek

3.1. Gépi tanulás, mélytanulás

A mesterséges intelligencia (MI, Artificial Intelligence, AI) egy átfogó tudományterület, amelynek célja olyan rendszerek és gépek fejlesztése, amelyek képesek modellezni és utánozni az emberi gondolkodást, döntéshozatali folyamatokat, valamint az absztrakt gondolkodást. Az MI magában foglalja azokat a technológiákat és módszereket, amelyek lehetővé teszik, hogy ezek a rendszerek képesek legyenek összetett problémák megoldására[11].

A mesterséges intelligencia egyik legmeghatározóbb részterülete a *Machine Learning* (ML, gépi tanulás), amelynek célja, hogy a rendszerek képesek legyenek önállóan tanulni a bemenetként szolgáló adatokból és a korábbi tapasztalatokból. A hagyományos, szabályalapú rendszerekkel szemben a machine learning algoritmusok nem előre meghatározott szabályokból építkeznek, hanem az adatok közötti mintázatok felismerésére, elemzésére és ezen mintázatok alapján történő predikciók készítésére képesek. Az ML-modellek a folyamatos tanulás révén optimalizálják a teljesítményüket, és egyre pontosabb eredményeket nyújtanak.

Az elmúlt évtizedek technológiai fejlődése és a számítási kapacitás növekedése jelentősen elősegítette a Deep Learning (DL, mélytanulás) térnyerését, amely a gépi tanulás egy specifikus ága. A DL több rétegű, úgynevezett mély neurális hálózatokat (Deep Neural Networks, DNN) alkalmaz, amelyek képesek nagy mennyiségű, összetett adat feldolgozására és elemzésére. A DNN modellek rétegei hierarchikusan rendezettek, ami lehetővé teszi a bonyolult mintázatok és összefüggések felismerését az adathalmazban. Ezek a modellek különösen hatékonyak olyan területeken, mint a képfelismerés, környezettel való interakció, beszédfelismerés és a természetes nyelvfeldolgozás.

A deep learning keretein belül sok stratégia létezik a modellek tanítására, azok strukturalására. Napjainkban igen elterjedt a megerősítéses tanulás (RL, Reinforcement Learning). Ez egy olyan gépi tanulási paradigma, amely során a modell a környezetével való interakciói alapján tanul. A modell a kapott jutalmak és büntetések segítségével optimalizálja döntéseit, cselekvéseit [12]. A Q-learning az RL egyik legismertebb algoritmus, amely hatékonyan alkalmazható különböző döntéshozatali és vezérlési feladatok megoldására. Emellett a Deep Q-Network (DQN) egy olyan technika, amely ötvözi a megerősítéses tanulás és a deep learning előnyeit, így képes még összetettebb problémák megoldására.

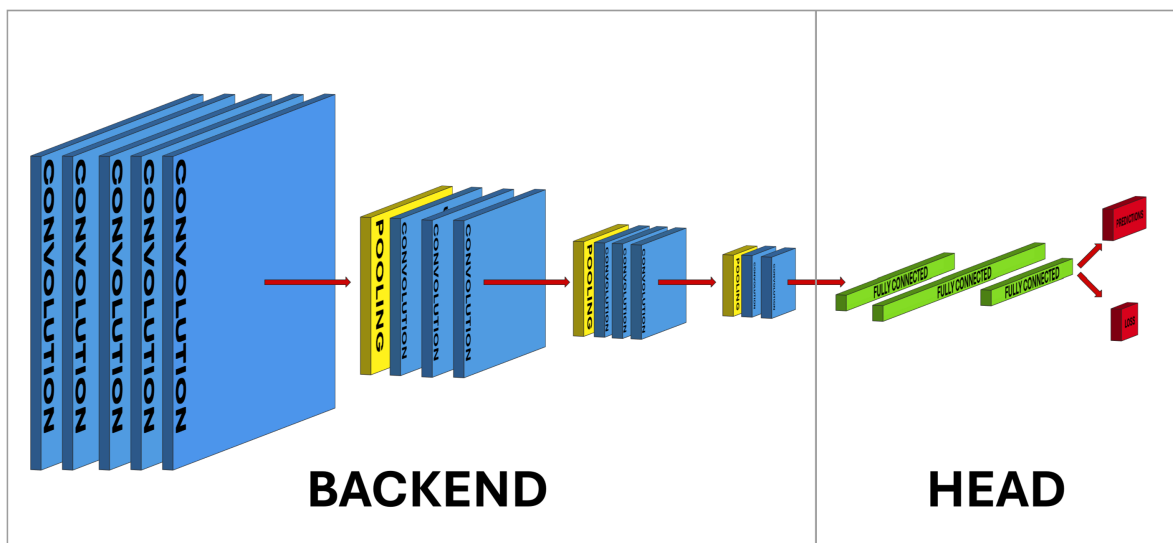
A felügyelet nélküli tanulás (USL, Unsupervised Learning) címkézetlen adatokkal dolgozik, és célja az adatok közötti mintázatok, struktúrák és rejtett összefüggések felfedezése. Ezt a módszert akkor alkalmazzák, amikor az adatokhoz nem tartoznak előre meghatározott címkék [13]. A klaszterezési módszerek, mint például a K-means algoritmus, az USL egyik leggyakoribb felhasználási területe, amely az adatok csoportosítását végzi. Továbbá, a *Principal Component Analysis* (PCA, Főkomponens-Analízis) egy másik népszerű módszer a felügyelet nélküli tanulás területén, amely az adatok dimenziócsökkentésére szolgál, miközben megtartja azok leglényegesebb jellemzőit.

A félig felügyelt tanulás (SSL, Semi-Supervised Learning) a felügyelt és a felügyelet nélküli tanulás közötti átmenetet képviseli. Ebben a módszerben a modell egyaránt használ címkézett és címkézetlen adatokat a tanuláshoz, ami különösen hasznos, ha a címkézett adatok drágán vagy nehezen előállíthatók [14]. A generatív modellek és a *Label Propagation* technikák a félig felügyelt tanulás tipikus alkalmazásai, amelyek a címkézetlen adatok felhasználásával növelik a modell hatékonyságát és pontosságát.

Napjainkban az ipari környezetben talán leginkább elterjed tanítási stratégia a felügyelt tanulás (SL, Supervised Learning), amely egy olyan módszer, amelyben az algoritmusok címkézett adathalmaz alapján tanulnak. A felügyelt tanulás során a modell megtanulja összekapcsolni az adatokhoz tartozó címkéket az adatok jellemzőivel, majd az új, címkézetlen adatok alapján képes lesz predikciókat készíteni. A felügyelt tanulás széles körben alkalmazott módszerei közé tartoznak a lineáris regresszió, logisztikus regresszió, Support Vector Machine, valamint a döntési fák, amelyeket gyakran használnak osztályozási és regressziós problémák megoldására. A következő fejezetben tárgyalt CNN-ek a felügyelt tanulás keretében is kiemelkedő szerepet játszanak a képosztályozási, szegmentációs és detekciós feladatok során.

3.2. Konvolúciós Neurális Hálózatok

A Konvolúciós Neurális Hálózatok (CNN, Convolutional Neural Network) legnagyobb előnye a hagyományos, *Fully Connected* neurális hálózatokkal (FC, sorosan kapcsolt neurális hálózatok) szemben, hogy képesek a képek térbeli, lokális jellemzői alapján információkat kinyerni. Ezzel lehetővé válik számukra, hogy bonyolultabb mintázatokat és összefüggéseket tanuljanak meg, ami jelentős előnyt biztosít a képfeldolgozásban. A CNN-ek kevésbé érzékenyek az invarianciákra, például a különböző méretű, elforgatott vagy eltolott mintázatokra, és jobb általánosító képességgel rendelkeznek, mivel hatékonyan felismerik az azonos mintázatokat eltérő kontextusokban is.



3.1. ábra. Egy egyszerű CNN tipikus felépítése

A konvolúciós hálózatok sikeressége jelentős mértékben tulajdonítható a *Receptive Field* (látómező) koncepciónak, amely lehetővé teszi a modellek számára, hogy a képeken egyszerre kisebb területeket tudjanak megfigyelni, majd ezeket a lokális mintázatokat össze tudják kapcsolni globális struktúrákká. A hagyományos fully connected hálózatok erre nem, vagy csak korlátozottan képesek, ugyanis itt általánosságban minden bemenetnek a kép egy-egy pixele felel meg, amelyek így elvesztik a térbeli összefüggéseiket.

A CNN-ek további előnye a *Parameter Sharing* (paramétermegosztás), amely a konvolúciós szűrők alkalmazásának módjából adódik. A konvolúciós szűrőket a kép egészén alkalmazzuk, így ugyan azokat a súlyokat többször felhasználjuk. Így ahelyett, hogy minden lokális egységhez külön szűrőt kellene megtanulnia, ugyan azt a szűrőt tudja használni a képek egészén. Ennek okán a hálónak sokkal kevesebb súlyt kell megtanulnia, amivel jelentősen lecsökken memóriaigénye [15].

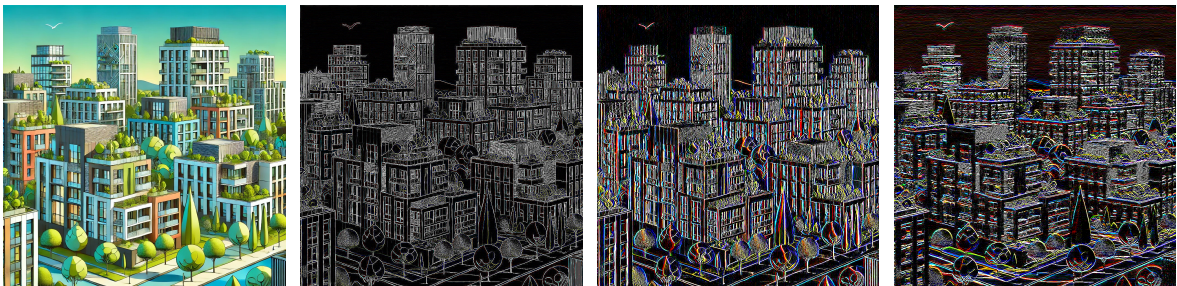
A modern konvolúciós neurális hálózatok, mint a Resnet, DenseNet vagy Inception még mélyebb hálózatok létrehozását tették lehetővé olyan technikákkal, amelyek megoldják a deep learning tipikus problémáit, például a *Vanishing Gradient* (eltűnő gradiens) problémája. Ezeket a hálózatokat *Residual Blokk*-okkal, *Skip Connection*-ökkel és egyéb fejlett technikákkal látták el, hogy megkönnyítsék az információáramlást a mélyebb rétegek felé. Így ezek a hálózatok már nemcsak képek osztályozására képesek, akár szegmentációra, detekcióra, vagy élő videók feldolgozására is használhatóak.

A konvolúciós neurális hálózatok a legtöbb esetben két részből épülnek fel: egy *Backend* és egy *Head* részből. A backend részben találhatóak a *konvolúciós* és *pooling* rétegek, amelyek feladata a képeken található térbeli kapcsolatok felfedése és az azokon található információk kinyerése. A konvolúciós rétegek a kép lokális információinak kinyerését végzik, mint például az élek, textúrák és színek felismerése. A pooling rétegek a képek dimenzióját csökkentik a könnyebb és gyorsabb feldolgozhatóság érdekében, illetve a modell tanulását hatékonyabbá teszik az azonos mintázatok közötti különböző invarianciák kezelésében, amely például a különböző orientációjú, pozíciójú és méretű, de összességében hasonló mintázatok absztrakciójában segít.

Ezekből az információkból (illetve más a, modellel közölt információk alapján) hozza meg a döntést a modell head része, ami többnyire fully connected rétegekből áll. A head a backend által összegyűjtött jellemzőket kilapított (*flattened*) formában kapja meg, majd ezen információk alapján hozza meg a konkrét döntést, amely klasszifikáció esetén például a kimeneten elhelyezett *Softmax* függvényvel valósulhat meg.

Konvolúciós rétegek

A konvolúciós rétegek a CNN-ek alapjait képezik, amelyek lehetővé teszik a hálózatokat a képadatok hatékony értelmezésére. A hagyományos neurális hálózatoktól eltérően, ahol az minden bemeneti neuronnak közvetlen kapcsolata van a következő réteggel, itt az egyes konvolúciós rétegek lokális kapcsolatokat építenek ki. Egy adott konvolúciós szűrő (*Kernel*) egyszerre a kép csak egy részét látja.



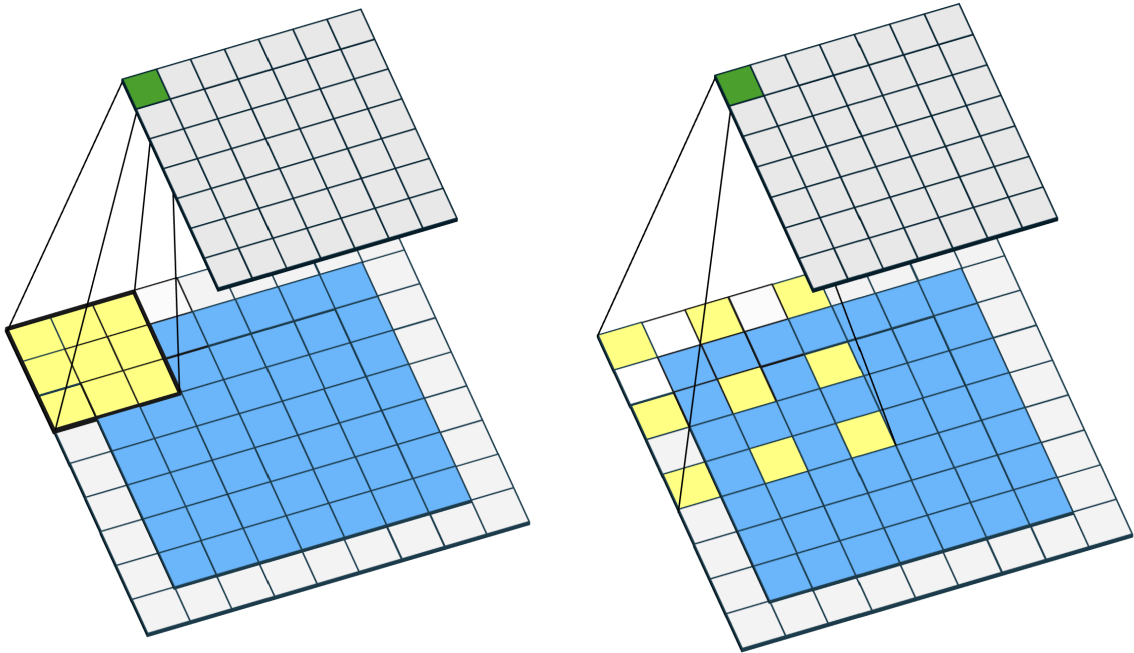
3.2. ábra. Különböző nevezetes szűrők alkalmazása egy egyszerű képen; balról jobbra: eredeti kép, Diszkrét Laplace operátor, Scharr-szűrő (y), Scharr-szűrő (x)

Ezt a szűrőt a kép különböző pontjaira alkalmazzák, azaz a kernel és a kép adott elemeit diszkrét konvolúcióval kombinálják. Ezzel kapjuk meg az adott szűrőhöz tartozó *Feature Map*-et (jellemzőtérkép). Ezen a képen már a szűrő által vizsgált lokális tulajdonságok (élek, színek, stb.) kiemelten szerepelnek. A diszkrét konvolúció ebben az esetben kétdimenziós mátrixokra:

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a, j+b} K_{a,b} \quad (3.1)$$

ahol S_{ij} a kimeneti mátrix (kép) i -edik sorának és j -edik oszlopának értéke, I a bemeneti kép, K pedig a használt kernel.

A konvolúció művelete a képek méretének csökkenésével jár, így bevett technika a *Padding*, amely során a kép kerületén nulla intenzitású pixeleket helyeznek el, amelyek ilyen módon nem befolyásolják a kialakuló feature map-et, ugyanakkor a kép méretének csökkenését megakadályozzák. A szűrőket az egész képen végigcsúsztatják bizonyos lépésközzel (*Stride*) és dilatációval (*Dilation*, 3.3). Az egyes szűrőket a méretükkel, a hozzájuk tartozó stride és dialtion értékekkel szokták jellemezni, ezek előre beállítottak, a hálózat nem tanulható paraméterei közé tartoznak.

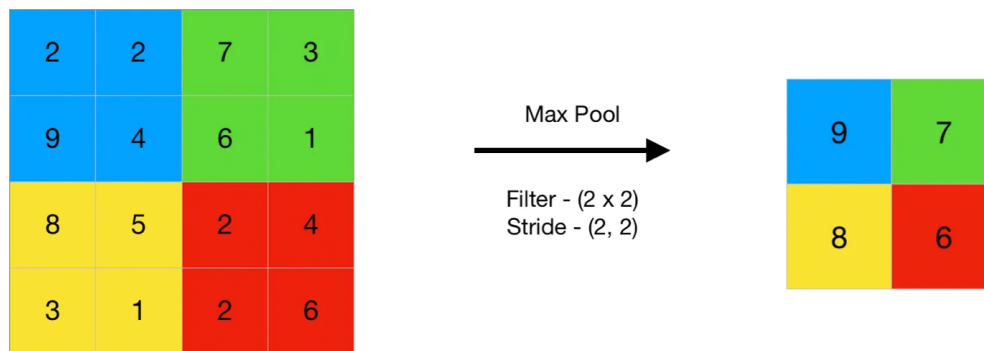


3.3. ábra. Dilation hatása a konvolúciós rétegre.

Pooling

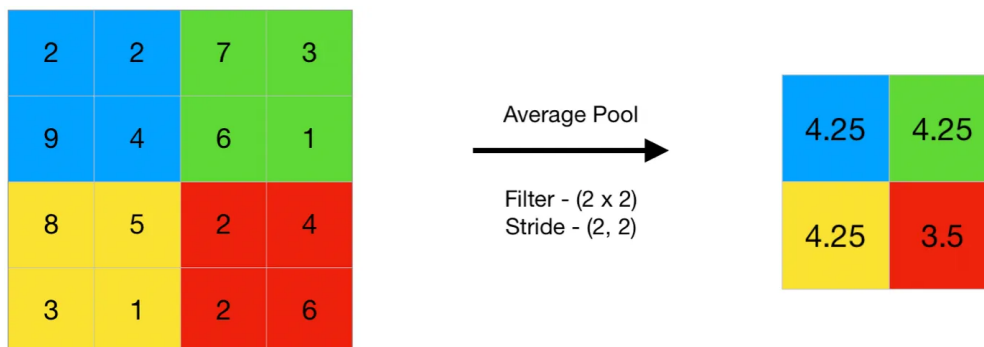
A pooling rétegek általában a konvolúciós rétegek után helyezkednek el, a feature-map felbontásának csökkentésére használják őket. Céljuk, hogy az adatok térbeli dimenzióját csökkentsék a lényegi jellemzők megtartása mellett. Ez nagyban csökkenti a modell számítási igényét, illetve további invarianciát visz a rendszerbe, amivel segíti a modell általánosító képességét, ezzel az overfitting elkerülését.

A pooling rétegek működése során a bemenő feature-map-et kis, nem átfedő ablakokra bontják, majd ezek mindegyikén végrehajtják a pooling műveletet, aminek több fajtája ismert [16]. Leggyakrabban alkalmazott az úgynevezett *max pooling* (3.4), amely során az ablakban található értékek közül a legnagyobbat választják ki. Ez a megoldás előnyös, ha a kép leglényegesebb struktúráit szeretnénk megtartani.



3.4. ábra. Max pooling működése [17]

Egy másik, kevésbé elterjedt pooling technika az *average pooling* (3.5), amely során az új feature-map-re az ablakban lévő "pixelek" értékének átlaga kerül. Ez segít a zaj szűrésében, ugyanakkor a kép elmosódottabb lesz alkalmazása után.



3.5. ábra. Average pooling működése [17]

3.3. A felügyelt tanulási folyamat

A felügyelt tanulás napjaink egyik legmeghatározóbb mélytanulási módszere, amely címkézett adatok használatára épül. A folyamat során a modell olyan adatokat kap bemenetként, amelyek rendelkeznek az elvárt kimeneti értékekkel, ezek alapján tanulja meg az adatok és a címkék közötti összefüggéseket. Így a modell képessé válik új, nem látott adatok alapján pontos predikció meghozására.

A modell tanítása több, egymás utáni lépésben történik. Először a bemenetekből készít egy predikciót a modell (*Forward pass*), majd a valós címke és a predikció összevetésével megállapítjuk a kettő közötti eltérést (*Loss*). A loss-t visszaterjesztjük a hálózatban (*Backpropagation*), ezzel megkapjuk az egyes súlyok szerepét az aktuális predikcióban, majd ezeknek megfelelően egy optimalizálási algoritmussal frissítjük a súlyokat[18]. Ennek a ciklusnak az ismétlésével érhetjük el, hogy modellünk megtanulja a teszt adathalmaz jellemzőit, és absztrakciókat építsen ki.

A tanítás lépései

A **forward pass** (előterjesztés, forward propagation) során a modell a bemeneti adatok alapján a rétegek közötti összefüggések (súlyok és *bias* értékek) felhasználásával egy kimeneti predikciót készít. A rétegek közötti aktivációs függvények biztosítják a modell számára a nemlinearitást, így képes komplex problémák megoldására is[19].

Amennyiben a modell feladata klasszifikáció, úgy minden osztályhoz a *Softmax* függvény segítségével rendel egy valószínűséget. A Softmax függvény a kimeneti neurális réteg aktivációit alakítja valószínűségi eloszlássá, így ezek összegét 1-nek állítja be. A modell ez alapján kiválasztja a legnagyobb valószínűséggel rendelkező kimenethez tartozó osztályt, mint végleges predikciót. Így nemcsak egy egyszerű predikciót kapunk, hanem megállapíthatjuk azt is, hogy a modell mennyire biztos döntésében. A Softmax függvény a következőképpen írható le:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^K e^{x_j}}, \quad (3.2)$$

ahol x_i az i -edik osztály kimeneti aktivációja és K az összes osztályok száma.

A modell forward pass során előállított predikcióját összehasonlítjuk a valódi címke értékével, majd ez alapján meghatározzuk a modell teljesítményét. Ez többnyire egy előre meghatározott **hibafüggvény** segítségével történik, amelynek célja, hogy számússítsa a modell hibáját, amelyet a későbbiekben minimalizálni szeretnénk.

Osztályozási problémák esetén a leggyakrabban használt hibafüggvény a *Cross-Entropy loss*, amely figyelembe veszi az összes osztályhoz tartozó kimenet aktivációját, így számol a modell predikciójának *confidence* (magabiztosság) értékeivel. a Cross-Entropy loss a következő módon írható le:

$$\text{Cross-Entropy} = - \sum_{i=1}^K y_i \log(\hat{y}_i), \quad (3.3)$$

ahol K az osztályok száma, y_i a valós osztály címkéje, és $\hat{y}_i$ a modell által adott valószínűség az i -edik osztályhoz.

A **backpropagation** során a modell forward pass-ban előállított predikciójának hibáját terjesztjük vissza a hálózatban valamilyen gradiens alapú módszer használatával. A gradiens alapú módszerekkel egyszerűen meghatározhatjuk, hogy az adott súlyt milyen irányba és mekkora értékkel kell módosítanunk, hogy a hálózat predikciós hibáját csökkentsük, ezzel teljesítményét megnövelhessük.

Első lépésként ki kell számítanunk a hibafüggvény gradiensét a kimeneti rétegre. Legyen a hibafüggvény L , a modell által prediktált érték $\hat{y}_i$, a címke pedig y . A célunk az, hogy minimalizáljuk L -t. A hibafüggvény kimeneti réteg aktivációja szerinti deriváltja meghatározza, hogy a kimeneti réteg aktivációjának növelése vagy csökkentése hogyan befolyásolja a veszteséget, így az arra vonatkozó gradiens a következő módon alakul:

$$\frac{\partial L}{\partial \hat{y}_i} \quad (3.4)$$

A kimeneti réteg után a gradiens visszaterjesztése következik minden korábbi rétegre a láncszabály alapján. Minden rétegnél kiszámoljuk a hibának az adott réteg aktivációjára gyakorolt hatását, majd továbbítjuk azt az előző rétegek felé. Egy-egy neuron gradiensét az alábbiak szerint kaphatjuk meg:

$$\frac{\partial L}{\partial x_j} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial x_j}, \quad (3.5)$$

ahol x_j a rejtett réteg j -edik neuronjának bemenete. Ezzel a láncszabály lehetővé teszi számunkra, hogy minden egyes neuronról megállapíthassuk, hogy mekkora mértékben kell növelni vagy csökkenteni a háló teljesítményének optimalizálásához.

Miután kiszámítottuk az egyes neuronokhoz tartozó gradienseket, egy **gradiens alapú optimalizálási algoritmus** segítségével módosítanunk kell a háló súlyait annak optimalizálásához. Ezek az algoritmusok felelősek a súlyok megfelelő mértékű frissítéséért.

Az egyik legegyszerűbb algoritmus a *Gradient Descent*, amely a súlyokat a hibafüggvényének gradiensével ellentétes irányba mozditja el, így csökkentve a hibát. Az i -edik réteg j -edik neuronjához kapcsolódó w_{ij} súly frissítését a következő módon írhatjuk fel:

$$w_{ij} \leftarrow w_{ij} - \alpha \cdot \frac{\partial L}{\partial w_{ij}}, \quad (3.6)$$

ahol α a tanulási ráta, amely a súlyok frissítésének lépésméretét szabályozza.

Ezek a tanítási folyamat egy iterációját adják. Így a tanítási folyamatot több egységre bonthatjuk: a tanító adathalmazt kisebb részekre, úgynevezett *batch*-ekre választjuk szét. Egy *batch* a tanító adathalmaz egy részhalmazát jelenti, amelyen egyszerre, párhuzamosan számítjuk ki a gradienst és a súlyok frissített értékét. Egy *epizód* (epoch) alatt az összes *batch*-et feldolgozzuk, azaz a teljes tanító adathalmazon végigmegyünk. Az epizódokon belül a *batch*-eket véletlenszerű sorrendben adagoljuk a hálózatnak, majd azok alapján fokozatosan módosítjuk a súlyokat az optimalizálási algoritmusnak megfelelően.

4. fejezet

Környezet

4.1. Python

Az tanítási algoritmus implementációja Python programozási nyelven történt, amely széles körben elterjedt a gépi tanulás, mélytanulás és egyéb tudományos területek kutatásában. A Python napjainkban a kutatási projektek és a machine learning közösségek körében is standard eszköznek számít, mivel magas szintű, interpretált programnyelvként egyszerű és gyors fejlesztési folyamatot biztosít. Az olvasható szintaxisa és rugalmassága hozzájárul a gyors prototípus-fejlesztéshez és a modellkarbantartás megkönnyítéséhez, ami különösen fontos a kutatás során gyakran szükséges iterációkhoz. A Pythonhoz elérhető széles körű könyvtár- és keretrendszer-támogatás, mint például a *NumPy*, *Pandas*, *Matplotlib*, *Scikit-learn*, valamint deep learning keretrendszerek, mint a *TensorFlow* és *PyTorch*, lehetővé teszik a hatékony adatfeldolgozást, a modellépítést és a teljesítmény optimalizálását.

4.2. Pytorch

A kutatás során a *PyTorch* deep learning keretrendszert használtam, amely széles körben elterjedt eszköz a gépi tanulás, mélytanulás területén. A PyTorch dinamikus számítási gráfja lehetővé teszi a modellek futás közbeni módosítását és tesztelését, ami nagy rugalmasságot biztosít a fejlesztési folyamat során. Az implementáció során nem magukat a neurális hálózati architektúrákat fejlesztettem, hanem a PyTorch beépített modelljeit használtam, mint például a *MobileNetV3* small és large verzió. Ezek a modellek jól ismertek hatékony és optimalizált architektúráikról, amelyeket különböző képfeldolgozási feladatokra fejlesztettek ki.

Az általam írt algoritmus elsősorban a modellek tanítási folyamatának módosítására fókuszált. Az modellek betanítása, illetve újratanítása (fine-tuning) során a Mobile-

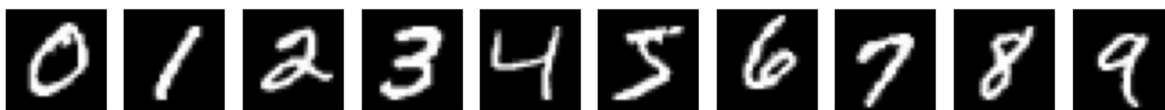
NetV3 small hálózatot vettem alapul, amely hatékonyan működik különböző képfeldolgozási feladatokon, így megfelelő alapot biztosított az általam alkalmazott technikák tesztelésére.

A PyTorch natív támogatása a GPU-alapú számításokhoz kulcsfontosságú volt az adathalmazok hatékony feldolgozásához, különösen a mély neurális hálózatok tanításának felgyorsításában. A keretrendszer egyszerű integrációja a *CUDA* technológiával jelentős mértékben csökkentette a tanítási időt, így lehetővé tette az algoritmus tesztelésének iterálását és finomhangolását.

A kutatásban való széles körű elterjedtségén túl a PyTorch egyre inkább elfogadottá válik ipari projektekben is, különösen olyan alkalmazásokban, ahol gyors iterációra van szükség a modellfejlesztési folyamatok során. Az aktív közösségi támogatás és a folyamatos fejlesztések biztosítják, hogy a legújabb technológiák és eszközök mindig elérhetők legyenek a kutatók számára.

4.3. MNIST adatbázis

Az MNIST (Modified National Institute of Standards and Technology) adatbázis a machine-, illetve deep learning kutatásában széles körben alkalmazott, alapvető képadatbázis. A készlet 70,000 kézzel írt számjegyet tartalmaz (0-tól 9-ig), amelyek 28x28 pixeles, fekete-fehér képekként jelennek meg. Az adatbázis 60,000 képe tanításra, míg 10,000 tesztelésre szolgál. Az egyes képek pixelei 0 és 255 közötti intenzitásértéket kapnak, ahol a 0 a fekete, a 255 a fehér színnek felel meg. Az MNIST-et általában különféle machine learning algoritmusok, különösen osztályozást végző modellek kiértékelésére használják, ahol a cél a számjegyek pontos felismerése és kategorizálása [20].



4.1. ábra. Részlet az MNIST adatbázisból, számjegyek nullától tízig

Ez az adatbázis egy szabványos benchmarkként szolgál a machine learning modellek teljesítményének mérésére és összehasonlítására, mivel egyszerű szerkezete ideális különféle algoritmusok, mint például a Supervised Learning (SL, felügyelt tanítás) alapú modellek tesztelésére. Bár az MNIST viszonylag könnyű problémát kínál, elég kihívást jelent ahhoz, hogy megbízhatóan értékelhetőek legyenek rajta a modellek képességei és általánosítási teljesítményei.

5. fejezet

MCTS-alapú optimalizáció

Kutatásom középpontjában a Monte Carlo Tree Search algoritmus alkalmazása áll Supervised Learning tanítási folyamat optimalizálására.

Az általam felvetett probléma megértéséhez érdemes az alábbiakba belegondolni: a modell állapotára tekinthetünk úgy, mint az optimalizációs térben egy adott pontra, amely egy n -dimenziós térben helyezkedik el, ahol n a hálózat tanulható paramétereinek száma. Ennek értelmében minden egyes paraméter (például egy-egy súly vagy bias) egy-egy dimenziót képvisel ebben a térben, így a modell súlykészletének állapota egyértelműen meghatároz egy konkrét pozíciót.[21].

Ebben a nézőpontban a hálózat teljesítménye - legyen az a validációs pontosság, hibaérték, vagy egyéb mérőszáma a teljesítménynek - egy olyan sokváltozós függvényként értelmezhető, amely ezen az n -dimenziós téren definiált. A tanítási folyamat célja, hogy a modell a lehető legjobb pontosságot érje el, azaz a modell állapota egy olyan pontra konvergáljon ebben a térben, ahol a pontosság maximális - vagy a veszteség minimális[22]. Így feladatunk egyértelművé vált: egy olyan trajektóriát kell terveznünk, amely a lehető leggyorsabban eljut a globális optimumba.

A tanítás során minden batch után a kapott hibaértékből számolt gradienseket visszaterjesztjük a neurális hálózatban, majd az egyes neuronok gradiense szerint módosítjuk a neuronokhoz tartozó súlyokat. Ezzel egy lépést teszünk az optimalizációs térben, így tudunk batch-enként haladni az optimum felé - vagy éppen attól eltérő irányba[23].

Klasszikus esetben a batch-ek sorrendje ugyan olyan marad, vagy véletlenszerűen változik az epizódok között, ami zavaró tényezőként hathat a modell konvergenciájára[24]. A modell mozgása így kaotikussá válhat: nem mindig a legkedvezőbb batch-et - *lépést az optimalizációs térben* - dolgozza fel, így lassulhat a konvergencia, vagy rosszabb esetben megrekedhet egy lokális optimumban. Ezzel a technikával a modell nem feltétlenül képes eljutni az *epizód alatt elérhető legjobb állapotba*.

Ezt a jelenséget többen is vizsgálták már. Keskar és társai (2016) [25] kimutatták, hogy nagyobb batch-méretek mellett a modell általánosító képessége jelentősen gyengébb, ugyanis hajlamos megakadni az optimalizációs tér "élesebb" lokális minimumaiban, nehezebben lépnek ki ezekből, míg kisebb batch-méretek mellett a modell a "laposabb" optimumok felé konvergál.

Bengio és társai "*Curriculum learning*" (2009) [26] című cikkükben mutatták ki, hogy neurális hálózatok tanítási folyamatát is segítheti, ha a modell fokozatosan egyre nehezebb mintákon tanul. Ennek értelmében, ha a mintákat aktívan választják ki, akkor a modell konvergenciája felgyorsul, absztrakciós képessége megnövekszik.

Kumar és társai 2010-ben adták ki "*Self-paced Learning for Latent Variable Models*" [27] című cikküket, amiben bemutatták a self-paced learning folyamatát, amiben a modell először a könnyebb példákon tanul, majd fokozatosan bevonja a tanulásba a nehezebb példákat is. Ez annyiban különbözik a fentebb bemutatott curriculum learning-től, hogy a minták választása nem egy előre meghatározott sorrend szerint történik, hanem dinamikusan választják ki a tanuláshoz szükséges mintákat a modell aktuális állapotához igazodva. Ezzel még a curriculum learning-hez képest is jobb eredményeket lehet elérni.

Ezek alapján feltehetjük a kérdést: vajon létezik olyan módszer, amivel teljes mértékben optimalizálni lehet a minták sorrendjét? Dolgozatomban erre a kérdésre keresem a választ. Az általam javasolt módszer alapján segítségül hívhatjuk a Monte Carlo Tree Search algoritmust, amivel jelentősen javíthatjuk a modell konvergenciáját, és felállíthatunk minden epizódban egy olyan batch-sorrendet - tervezhetünk egy olyan *trajektóriát* - amely közel áll az optimálishoz.

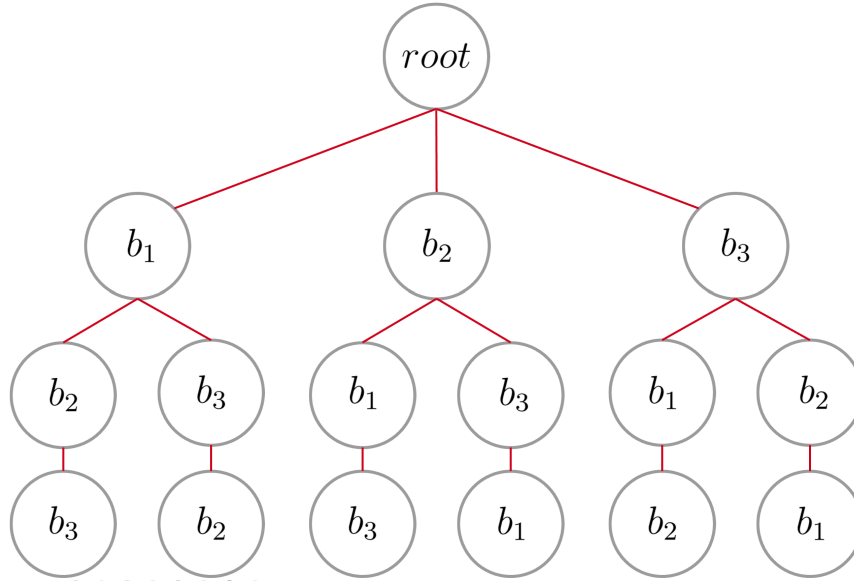
5.1. MCTS alkalmazása

Fa felépítése

Az általam fejlesztett batch-sorrend optimalizált tanítás az MCTS algoritmus használatán alapul, amely a batch-ek fastruktúrába szervezésével határozza meg a legjobb tanulási útvonalat. Ebben a fastruktúrában minden csúcs egy-egy állapotot képvisel, amely az addig feldolgozott batch-ek tanítása utáni modell állapotát jelenti. Egy csúcs „gyermekai” pedig azok a lehetséges következő állapotok, amelyek az adott állapotból kiindulva, a még fel nem használt batch-ek valamelyikének betanításával érhetők el. Az egy csomópontból maximálisan létrehozható gyermekek számát a fa branching factora írja le.

A folyamat az epizód elején az MCTS gyökércsúcsával kezdődik, amely a modell kezdeti állapotát jelenti, vagyis azt az állapotot, amelyet az előző epizód végén kaptunk. Ekkor a modell még nem tanult az aktuális epizód egyetlen batch-én sem.

A csúcsokban elérhető akciók megfelelnek a fennmaradó, fel nem használt batch-ek betanításával, így minden új csúcs létrehozása egy batch betanítását jelenti a jelenlegi állapotról, amellyel az optimalizációs térben új pozíciót érünk el. Ezáltal a $T(s, a)$ átmeneti függvény megfeleltethető egy-egy batch betanításával.



5.1. ábra. Lehetséges batch-sorrendek fája három batch (b_1, b_2, b_3) esetén

Azokat az állapotokat tekinthetjük terminális állapotoknak, amelyekben az epizód során az összes batch meg lett tanítva a modellnek. Az ezek kiértékeléséhez használatos $R(s)$ állapotkiértékelési függvény megfelel a modell pontosságának validálásával. Tesztjeim során a modell teljesítményét a validációs pontossággal mértem, ennek maximalizálására fektettem a hangsúlyt.

A rollout során a fennmaradó batch-eket véletlenszerű sorrendben megtanítjuk a hálózatnak, majd az úgy kapott modellt kiértékeljük. Ezzel egy jó előrejelzést kaphatunk az egyes batch-sorrendek által kapott teljesítményről.

Belátható, hogy a klasszikus tanítási módszer ¹ esetén nem csinálunk mást, mint a felállított tanítási fából egy véletlenszerű ágat kiválasztunk.

¹Klasszikus módszer alatt a véletlenszerű batch-sorrend értendő.

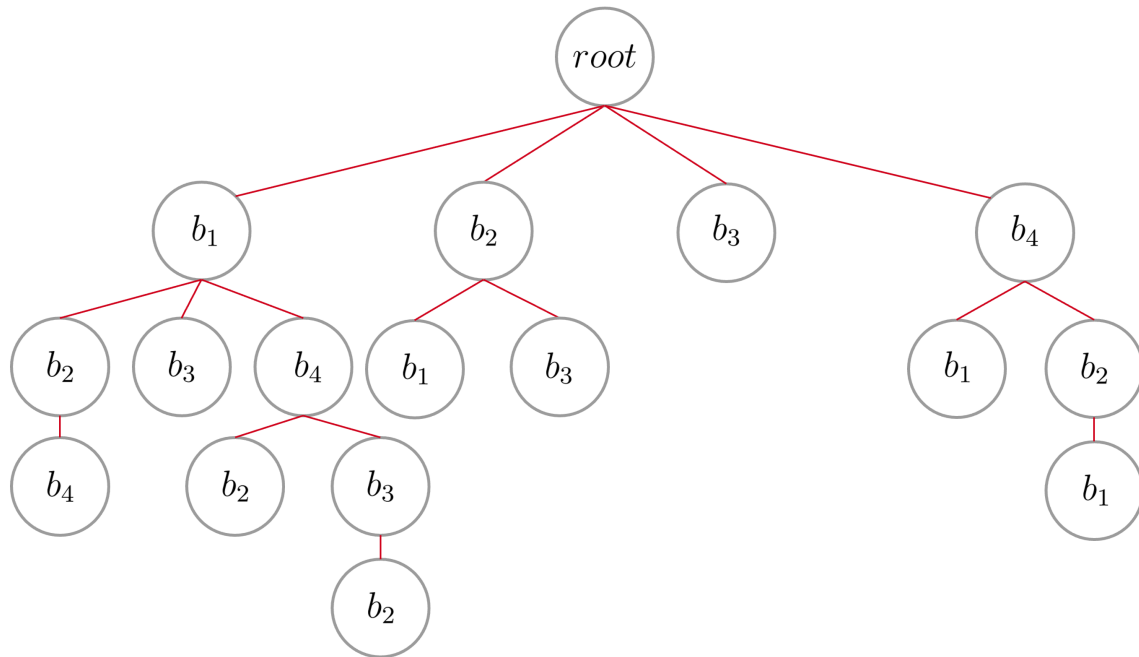
5.2. Metodológia

Kísérleteim során az MCTS alapú batch-sorrendel történő tanítás-optimalizálást vizsgáltam, amelyhez az MNIST adatbázist használtam, ami 60000 tanító adatot, és 10000 tesztelő illetve validációs adatot tartalmaz. Ezen futtattam tesztek különböző adatmennyiségek használata mellett tanítottam a MobileNetV3 small modellt, majd összehasonlítottam ezeket a klasszikus módszerrel kapott eredményekkel.

A következő paraméterekkel kezdtem el a tesztek futtatását: az **epizódok száma** (e), a **tanulási ráta** függvénye, a **felhasznált adatok mennyisége** (D), a **batch-ek mérete** (n) illetve az ezekből kapott a **batch-ek száma** (N). Az egyes batch-ek között nem tettem különbséget. A fenti adatok alapján egy tanítási fa mérete (N_v) (a csúcsok teljes száma) a következő összefüggéssel adható meg:

$$N_v = 1 + \sum_{i=1}^N \frac{N!}{(i-1)!} \quad (5.1)$$

Ahogy a fenti egyenletből is látszik, a batch-ek számának növelése az exponenciális mértéknél is gyorsabban növeli a tanítási fa méretét, ami komoly kihívást jelent számítási kapacitás szempontjából. Minden egyes batch tanítása után újabb gyermekek jönnek létre a fában, ami jelentősen bonyolultabbá teszi az optimalizációs feladatot. Erre a problémára nyújt kiváló megoldást a Monte Carlo Tree Search algoritmus, amellyel effektíven navigálhatunk a fában a legjobb terminális állapotot keresve.



5.2. ábra. MCTS által épített aszimmetrikus fa

A csúcsokhoz kapcsolt validációs pontosságot a $[0, 1]$ intervallumon tartottam, így a nagy optimalizációs tér mellett az MCTS dolga tovább nehezült azzal, hogy a tanítás végén a különböző csúcsokban található modell-állapotok közötti validációs pontosság különbségek lecsökkentek. Ezzel a csúcsok közötti UCT érték-különbségek is lecsökkentek, az explorációs tag nagyobb teret nyert, és sokkal több csúcsot kellett meglátogatnia az algoritmusnak, még olyanokat is, amelyek nem értek el érdemi javulást az előző állapothoz képest. Ennek a hibának kiküszöbölésére vezettem be egy K exploitációs tényezőt, amellyel az exploitációs tagot tudom tovább súlyozni². Így a módosított UCT képlete a következő³:

$$UCT(v) = K \cdot \bar{x}(v) + C_p \cdot \sqrt{\frac{2 \cdot \ln N(P(v))}{N(v)}} \quad (5.2)$$

A K exploitációs tényezőt a tanítás során lineárisan növeltem, így biztosítva, hogy az algoritmus könnyebben különbséget tudjon tenni a közeli UCT értékű csúcsok között, és ki tudja választani a legígéretesebbet közülük a későbbi epizódok alatt is, ahol ezek az eltérések többedik tizedesjegynél jelentkezhetnek. A lineáris növelés egy jó megoldásnak bizonyult, ugyanakkor a módszer finomhangolásra szorul, további vizsgálatok szükségesek az optimális paraméterezés megtalálására.

A tesztelések során azonos adatmennyiséggel tanított modelleket hasonlítottam össze, hogy pontosan fel tudjam mérni az MCTS-alapú és a klasszikus megközelítések teljesítménykülönbségét. A probléma jellegéből adódóan a körük épített paraméterek finomhangolása külön kísérleteket igényeltek. Egy-egy paraméterezés csak egy adatmennyiségre volt megfelelő, így minden egyes adatmennyiségre irányuló kísérletnél ezeket a paramétereket finomhangolni kellett.

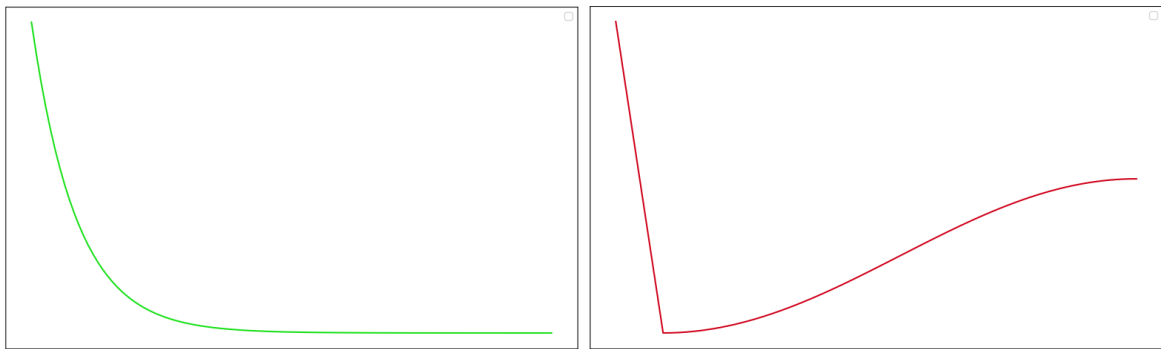
Az egyes adatbeállításoknál a tanulási ráta függvénye és az exploitációs tényező finomhangolása lényegesen hozzájárult a fejlesztett módszer sikerességéhez. Ennek megfelelően minden adatbeállításnál külön figyelmet igényelt ezen paraméterek helyes beállítása, ami akár iteratíván, több kísérleten keresztül zajlott. Az MCTS alapú optimalizáció több helyen meglepő eredményt hozott, az algoritmus eredményessége céljából több újszerű paraméterezést kellett bevezetni. A tanulási ráta szokatlan megválasztása a legszemléletesebb ezek közül. Az algoritmus által elért szokatlanul gyors konvergencia miatt a szokványos eljárások nem mindig mutattak megfelelő működést, körvonalazódott, hogy az algoritmusnak ezen téren különleges igényei vannak. Túl alacsony

²Természetesen az is megoldás lett volna, ha a C_p konstans csökken a tanítás alatt

³A 2.1 és 2.2 bekezdésekben leírt jelölések szerint

kezdeti tanulási ráta mellett a gyors konvergenciából jelentősen veszít az algoritmus, és a klasszikus módszerre hasonlító eredményeket kaptunk - megemelkedett számítási igénnyel. Túl magas tanulási ráta esetén hasonló problémába ütközünk, mint a klasszikus esetben, a modell nem képes hatékonyan az optimum felé konvergálni. Az optimalizáció során ezen paraméterek egyensúlyozása volt a legfőbb kihívás.

A tanulási ráta optimális időbeli csökkenése jelentette a legnagyobb problémát. Itt két megoldás került ki legeredményesebbként, egy exponenciálisan csökkenő tanulási ráta, és egy módosított, *Inverse Warmup Cosine Annealing* ráta. Ezek az alábbi ábrán (5.3) láthatóak:



5.3. ábra. Exponenciálisan lecsengő és Inverse Warmup Cosine Annealing tanulási ráta

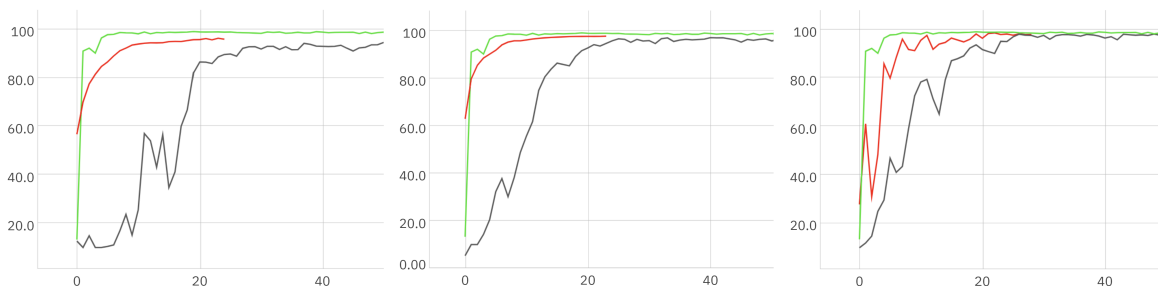
Ez az empirikusan kitapasztalt megközelítés meglepően hatékonyan segítette a hálózatot a gyors konvergenciához, ugyanakkor irodalmi említést nem találtam rá. Működési elve és hatása egyelőre nem tisztázott, matematikai indoklása további kutatás tárgyát képezheti.

6. fejezet

Eredmények

A MCTS-optimalizált batch-sorrendekkel történő tanítás lehetővé tette nagyobb tanulási ráták alkalmazása mellett a modell konvergenciájának stabilan tartását, így jelentős mértékben megnövelte annak sebességét a globális optimum felé. Nagy mértékben csökkentette a modell tanításához szükséges epizódok számát, és elősegítette a modell végső teljesítményének szignifikáns növelését a klasszikus módszer alkalmazásához képest, ugyanakkor, vagy akár kisebb adathalmaz használata mellett. Tesztjeim során az alapvető paraméterek, mint a batch-ek száma, a kiindulási tanulási ráta és a futáshoz használt seed érték a klasszikus és MCTS-optimalizált tanítások során az egyes adathalmazokon belüli vizsgálatoknál ugyan azok maradtak az összehasonlíthatóság kedvéért.

Kísérleteim során több megközelítést alkalmaztam: megvizsgáltam azt az esetet, amikor a modell teljes tanítása során optimalizáltam a batch-ek sorrendjét, a tanítás elejétől kezdve. Ez jelentős számítási kapacitást igényelt, ugyanakkor nem maradt eredmények nélkül. A modell konvergenciájának sebessége jóval meghaladta a klasszikus módszer által elérhetőt, ez az alábbi ábrákon (6.1) látható:



6.1. ábra. Modell konvergenciája különböző adatmennyiségek mellett (1000, 5000 és 15000 minta)

A grafikonokon pirossal az MCTS-optimalizált, szürkével a klasszikus módszerrel, zölddel a 60000 képpel tanított modell konvergenciája látható az epizódok függvényében. A

grafikonokon látható, hogy az MCTS-optimalizált tanítási módszer alkalmazása esetén a modellek konvergenciájának sebessége jelentősen megnőtt. Az 1000 és 5000 adattal tanított modellek esetén az **első epizód alatt már 60% körüli teljesítményt érhetünk el**.¹

Mindemellett a legnagyobb elért validációs pontosság tekintetében is sikerült felülmúlni a klasszikus módszer által nyújtott teljesítményt. A klasszikus módszernél 300 epizódon keresztül tanítottam a modelleket, míg az MCTS-optimalizált modelleknél 25 vagy 30 epizódot használtam. Az alábbi táblázatban (6.1) ennek összefoglalása látható:

Adatmennyiség	Klasszikus módszer	MCTS-optimalizált módszer
1000	95.88%	96.37%
5000	97.69%	97.74%
15000	98.53%	98.94%
60000	99.15%	-

6.1. táblázat. MCTS-optimalizált eredmények összehasonlítása a klasszikus módszerrel teljes tanítás esetén, több adathalmazra

A fenti táblázat alapján (6.1) látható, hogy az MCTS-optimalizált tanítási módszer jelentős előnyt biztosított a klasszikus megközelítéssel szemben. 1000 minta mellett **0,49%-os**, 5000 adat esetén **0,05%-os**, 15000 adat esetén pedig **0,41%-os** növekedést érthetünk el a klasszikus módszerhez képest. Látható, hogy több adatmennyiség esetén is kiváló eredményekkel működik az algoritmus, ugyanakkor a finomhangolás terén látható különbség van az 5000 mintával futtatott teszt, és a másik két teszt között, az utóbbiak javára. Az eredmények megmutatják, hogy a 15000 adatos MCTS-optimalizált tanítás versenyképessé vált a 60000 adatos klasszikus tanítással szemben is.

A tanítás egészének optimalizálása mellett megvizsgáltam a modellek továbbtanításának két módszerét is, bár a módszer számításigénye miatt csak korlátozott számú tesztet futtattam le. Az első módszer szerint a modelleket klasszikus módszerrel tanítottam, majd ugyanazon adathalmazon MCTS-optimalizált módszerrel finomhangoltam teljesítményüket. Ezen eredmények az alábbi táblázatban (6.2) láthatók:

2500 adat	Validációs pontosság	Epizódok száma
Klasszikus módszer	95.91%	300
MCTS-finomhangolás	96.93%	20

6.2. táblázat. Modell továbbtanítása azonos adathalmazon

¹A 15000 adattal tanított modell esetében még egy korábbi megközelítést alkalmaztam az algoritmus implementálására, a konvergencia stabilitása ezért mutat eltérő trendeket

A 6.2 táblázatban bemutatott eredmények alapján egyértelműen látható, hogy az MCTS-alapú finomhangolás jelentős mértékben javította a modell teljesítményét. 2500 minta használata esetén a klasszikus módszerrel **95,91%**-os maximális validációs pontosságot sikerült elérni, majd ezt a modellt továbbtanítva további **1,02%**-ot sikerült javítani ezen. Ezek alapján látható, hogy az algoritmus növelni tudja a címkézett adatok kihasználtságát, javítani tudja a modell hatásfokát az adatkihasználás tekintetében.

Egy másik módszer szerint a modellt a klasszikus módszerrel tanítottam az adathalmaz egy részével, majd az adathalmaz fennmaradó részén folytattam tanítását az MCTS-optimalizált módszerrel. Ezek az eredmények az alábbi táblázatban (6.3) találhatók:

Módszer	Validációs pontosság	Epizódok száma
Klasszikus tanítás 50000 adattal	99.12%	300
MCTS továbbtanítás 10000 adattal	99.20%	1
Klasszikus tanítás 60000 adattal	99.15%	300

6.3. táblázat. Modell továbbtanítása új adatokkal

Ez az eset azt az ipari módszert hivatott modellezni, amikor egy modell már használatra kész, ugyanakkor az időközben összegyűjtött adatmennyiséggel szeretnék javítani teljesítményét, frissíteni tudását a felhasználási környezetének változásaihoz alkalmazkodása végett. Látható, hogy az 50000 adattal klasszikus tanítás esetén 300 epizód alatt elért **99,12%**-os validációs pontosságot csupán egy epizód alatt **99,20%**-ra emelte az algoritmussal való tanítás, ami javulást mutat a 60000 adattal 300 epizódon keresztül tanított modell **99,15%**-os eredményéhez képest is. A dolgozat írásakor a legnagyobb validációs pontossággal működő modell (*"Branching/Merging CNN + Homogeneous Vector Capsules"* [28]) az MNSIT adatbázison 99,87%-os pontosságot ér el. Tekintve a modellek mérete, technológiai fejlettsége és specifikussága közötti különbséget, ez a növekedés igen jelentősnek számít.

7. fejezet

Konklúzió, kitekintés

A bemutatott eredmények alapján világosan látszik, hogy az MCTS-alapú batch-sorrend optimalizáció egy olyan innovatív megközelítést kínál a tanítási stratégiák között, amely rendkívüli sebességet mutat a tanulási konvergenciát illetően, illetve a hálózat végső teljesítményét is jelentős mértékben képes növelni. Alkalmazási lehetőségeit tekintve megállja a helyét mind a hálózat teljes tanítása esetén, mind annak finomhangolása, vagy továbbtanítása esetén.

Ugyan az algoritmus működése megkérdőjelezhetetlen, azonban számos kérdés és kihívás maradt megválaszolatlanul. A módszer jelentős számítási igénye további optimalizációs módszerek bevezetésére hívja fel a figyelmet, illetve a tanulási ráták és egyéb hiperparaméterek finomhangolása kulcsfontosságú az algoritmus tökéletesítéséhez.

További kutatások során érdemes lenne megvizsgálni az algoritmus skálázhatóságát nagyobb, komplexebb adathalmazokon és hálózatokon. Szükség lenne a módszer automatizált finomhangolásának feltérképezésére, illetve adaptív paraméterbeállítási stratégiák kidolgozására, amelyek akár jelentősen eltérhetnek a megszokott, jól ismert eljárásoktól. Ezek nagyban hozzájárulhatnak az MCTS-alapú batch-sorrend optimalizált tanítási metódus hatékonyabbá, robusztusabbá tételéhez. Ezzel egy új tanítási paradigma alakulhat ki, amely az optimalizációs eljárások egy új szintjét hozhatja el, lehetővé téve a modellek még gyorsabb és megbízhatóbb fejlődését.

Köszönetnyilvánítás

Ezúton szeretném megköszönni konzulensemnek, Kővári Bálintnak, hogy lehetővé tette ennek a dolgozatnak megszületését segítségével és mentorálásával.

Az MCTS-alapú batch-sorrend optimalizálás számítógépes látáshoz projekt nevében köszönetet mondok a HUN-REN Cloud (lásd: Héder et al. 2022; [29]; <https://science-cloud.hu/>) használatáért, ami hozzájárult az eredmények eléréséhez.

Ábrák jegyzéke

2.1.	Az amőba döntési fájának (játékfájának) részlete	5
2.2.	Csomóponti jelölések	6
2.3.	A BFS és DFS algoritmusok által előállított csomópont-sorrend	7
2.4.	Az Monte Carlo Tree Search algoritmus lépései[8]	8
2.5.	A $2\ln N(P(v))$ és $N(v)$ tagok viszonya egymáshoz	10
3.1.	Egy egyszerű CNN tipikus felépítése	16
3.2.	Különböző nevezetes szűrők alkalmazása egy egyszerű képen; balról jobbra: eredeti kép, Diszkrét Laplace operátor, Scharr-szűrő (y), Scharr-szűrő (x)	17
3.3.	Dilation hatása a konvolúciós rétegre.	18
3.4.	Max pooling működése [17]	19
3.5.	Average pooling működése [17]	19
4.1.	Részlet az MNIST adatbázisból, számjegyek nullától tízig	24
5.1.	Lehetséges batch-sorrendek fája három batch (b_1, b_2, b_3) esetén	27
5.2.	MCTS által épített aszimmetrikus fa	28
5.3.	Exponenciálisan lecsengő és Inverse Warmup Cosine Annealing tanulási ráta	30
6.1.	Modell konvergenciája különböző adatmennyiségek mellett (1000, 5000 és 15000 minta)	31

Táblázatok jegyzéke

6.1. MCTS-optimalizált eredmények összehasonlítása a klasszikus módszerrel teljes tanítás esetén, több adathalmazra	32
6.2. Modell továbbtanítása azonos adathalmazon	32
6.3. Modell továbbtanítása új adatokkal	33

Irodalomjegyzék

- [1] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [2] Ranjan Sapkota, Rizwan Qureshi, Marco Flores Calero, Chetan Badjugar, Upesh Nepal, Alwin Poullose, Peter Zeno, Uday Bhanu Prakash Vaddevolu, Sheheryar Khan, Maged Shoman, Hong Yan, and Manoj Karkee. Yolov10 to its genesis: A decadal and comprehensive review of the you only look once (yolo) series, 2024.
- [3] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. Yolov10: Real-time end-to-end object detection, 2024.
- [4] J Redmon. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.
- [5] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.
- [6] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [7] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.
- [8] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.
- [9] Bálint Kővári, Bálint Pelenczei, István Gellért Knáb, and Tamás Bécsi. Beyond trial and error: Lane keeping with monte carlo tree search-driven optimization of reinforcement learning. *Electronics*, 13(11):2058, 2024.

- [10] Levente Kocsis, Csaba Szepesvári, and Jan Willemson. Improved monte-carlo search. *Univ. Tartu, Estonia, Tech. Rep*, 1:1–22, 2006.
- [11] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.
- [12] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [13] Samreen Naeem, Aqib Ali, Sania Anam, and Muhammad Munawar Ahmed. An unsupervised machine learning algorithms: Comprehensive review. *International Journal of Computing and Digital Systems*, 2023.
- [14] Yanwen Chong, Yun Ding, Qing Yan, and Shaoming Pan. Graph-based semi-supervised learning: A review. *Neurocomputing*, 408:216–230, 2020.
- [15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2283–2286, 1998.
- [16] Afia Zafar, Muhammad Aamir, Nazri Mohd Nawi, Ali Arshad, Saman Riaz, Abdulrahman Alruban, Ashit Kumar Dutta, and Sultan Almotairi. A comparison of pooling methods for convolutional neural networks. *Applied Sciences*, 12(17):8643, 2022.
- [17] Abhishek Jain. Pooling and their types in cnn. Medium, 2024. Accessed: 2024-11-04.
- [18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [19] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.
- [20] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.
- [21] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.
- [22] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM review*, 60(2):223–311, 2018.

- [23] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gerard Ben Arous, and Yann LeCun. The Loss Surfaces of Multilayer Networks. In Guy Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 192–204, San Diego, California, USA, 09–12 May 2015. PMLR.
- [24] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002.
- [25] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.
- [26] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.
- [27] M Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. *Advances in neural information processing systems*, 23, 2010.
- [28] Adam Byerly, Tatiana Kalganova, and Ian Dear. No routing needed between capsules. *Neurocomputing*, 463:545–553, 2021.
- [29] Mihály Héder, Ernő Rigó, Dorottya Medgyesi, Róbert Lovas, Szabolcs Tenczer, Ferenc Török, Attila Farkas, Márk Emódi, József Kadlecsek, György Mező, Ádám Pintér, and Péter Kacsuk. The past, present and future of the ELKH cloud. *Információs Társadalom*, 22(2):128, aug 2022.