EÖTVÖS LORÁND TUDOMÁNYEGYETEM

TERMÉSZETTUDOMÁNYI KAR

TÁVÉRZÉKELÉSI ADATOK FELDOLGOZÁSA MÉLYTANULÁSSAL

SZAKDOLGOZAT

Tóth Benjámin

Matematika BSc

Matematikai elemző szakirány

Témavezető:

Lukács András

Számítógéptudományi Tanszék

Budapest

2023

Köszönetnyilvánítás

Szeretném kifejezni köszönetemet konzulensemnek, Lukács Andrásnak, aki szakértelmével és segítőkészségével kiemelkedő iránymutatást nyújtott a szakdolgozatom elkészítése során. Szeretnék külön köszönetet mondani Nagy Eszternek is, amiért támogatott és segített a szakdolgozatom elkészítésében.

Tartalomjegyzék

Bevezető

Az adatbányászat egyik alapproblémája a klasszifikáció, amely adott objektumok vagy rekordok előre adott, különböző osztályokba történő csoportosításával foglalkozik. Ez a dolgozat mélytanulási módszerek hatékonyságát mutatja be műholdképek klasszifikációs problémáján.
A dolgozat első fejezetében bemutatom a felhasznált adathalmazt és annak jellegzetességeit. A második fejezet a projekt során alkalmazott neurális hálót mutatja be, illetve a használt tanító algoritmust és annak a teljesítmény mérésére szolgáló metrikákat. A harmadik és negyedik fejezetben a hiperparaméterek optimalizálását mutatom be. Először a klasszikus hiperparaméterekkel foglakozom, majd a továbbiakban az adathalmazon végzett augmentációt megvalósító transzformációk kiválasztásával. Az ötödik fejezetben kiértékelem a legjobb beállítással tanított modell teljesítményét, és egy további ötletet mutatok be a modell javítására. A hatodik fejezetben ismertetem a GAN modellt, és annak segítségével olyan szintetikus adatotot generálok, amely az eredeti adathalmaz statisztikáival rendelkezik. A hetedik fejezetben a generált adatokat felhasználom a klasszifikációs modell tanításához, majd az így kapott eredményeket is kiértékelem és összevetem a korábbiakkal. A nyolcadik fejezetben végül a legjobb modellt kipróbálom egy az eredeti adathalmaztól függetlenül összeállított teszt adathalmazon és megvizsgálom, hogy azon milyen teljesítmény érhető el.
A fejezetek a feladat megoldásának gyakorlati módszertanát követik, ezáltal nem csupán az elméleti háttere ismerhető meg egy klasszifikációs feladatnak, hanem egy ilyen probléma megoldási módszere is végigkövethető az olvasó számára.

A projekt során használt Python kód: https://github.com/BenjaminToth/BScThesis

1. fejezet

A feldolgozott adat

1.1. Az adathalmaz leírása

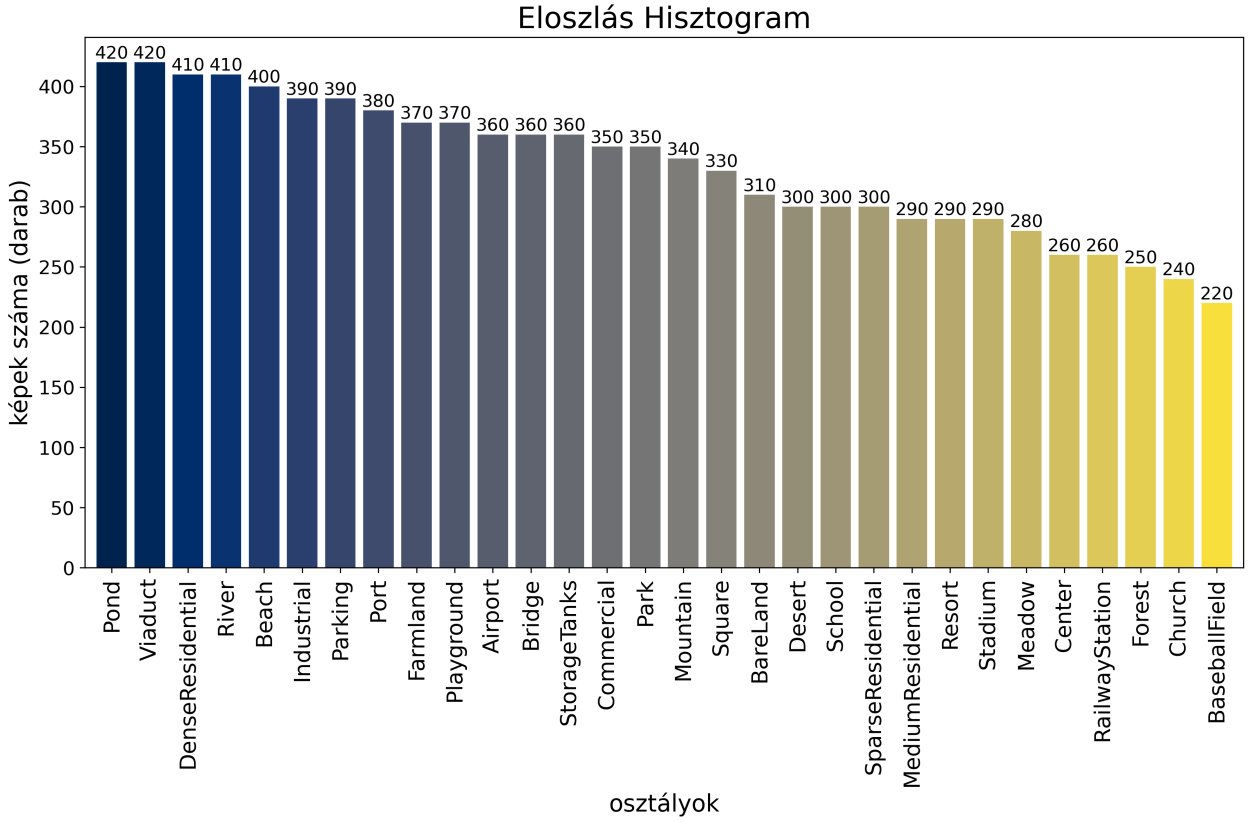
A szakdolgozatban tárgyalt adat forrása az AID adathalmaz (lásd a Függelékben). Ez egy nagyobb méretű képadatbázis, amely a Google Earth által használt felvételekből lett összeállítva. Az adathalmaz összeállításának célja klasszifikációs modellek tanítása és azokkal történő kísérletezés volt. AID készlete 30 különböző osztályban összesen 10000 RGB képet tartalmaz. Ezen színes képek mindegyike négyzet formájú, méretük többnyire 600×600 pixel, de előfordul pár kivétel ami 600 ± 10 pixeles oldalhosszúsággal rendelkezik.
A képek több forrásból származnak, mivel a Google Earth különböző távérzékelési szenzorokkal dolgozik. Földrajzi értelemben is több forrásról beszélhetünk, hiszen a Föld különböző részeiről válogatták a képeket, főként az Egyesült Államokból, Kínából, Angliából, Franciaországból, Olaszországból, Japánból és Németországból.

1.2. Osztályok vizualizálása

Az adathalmaz osztályai 30 légi tájkép kategóriát reprezentálnak. Ezek a kategóriák különböző környezeteket és helyszíneket fednek le, néhány osztály egy-egy nagyobb építményt reprezentál (stadion, templom, híd, stb), amíg más osztályok a terep milyenségét (mezőgazdaság, erdő, tengerpart, stb) vagy a lakóövezet sűrűségét (sűrű lakóterület, ritka lakóterület, stb) írják le. A fenti képen látható, hogy különbözik a nagyítás mértéke képenként. Ez többnyire osztályok közötti különbséget jelent, de valamennyire az osztályokon belül is érzékelhető. Megfigyelhető még az is, hogy az osztályt jellemző része a képeknek mindig középre van igazítva.

1.3. Osztályok eloszlása

Az adathalmazban szereplő 10000 kép nem egyenletesen oszlik el a 30 osztály között. A pontos adatokat a következő ábra mutatja:



Látható, hogy a legyakrabban előforduló osztály elemeinek a száma közel kétszerese a legritkábbnak. Azt is fontos megjegyezni, hogy habár az adathalmaz nem számít kis méretűnek, mégis osztályonként nem sok adatot tartalmaz. A későbbiekben ezekből a képmennyiségekből kell majd megtanulnia az adott modelleknek jó klasszifikációs teljesítményt nyújtani.

2. fejezet

Modell architektúra és metrika

2.1. Klasszifikációs háló

2.1.1. ResNet50

A projektem során használt neurális háló a ResNet50 modell volt. ResNet50 (Residual Network 50) egy konvolúciós neurális háló, amely kép alapú objektumfelismerési feladatokra alkalmazható, emellett napjaink egyik legtöbbet hivatkozott mélytanulási architektúrája.
Ez a modell 49 konvolúciós réteget és egy egy másodikként megjelenő max pooling réteget használ (így adódik az 50-es elnevezés), amely backbone végén egy average pooling réteg, és egy 1000 neuronos sűrű réteg softmax aktivációval található. Aktivációs rétegnek a modell a konvolúciós blokkok során ReLU-t alkalmaz. ResNet50-et a rétegek száma alapján a mélyebb neurális hálók közé soroljuk [2].
A ResNet innovációja a residual blokkok használata volt. A mélyebb hálók esetén gyakran felmerülő probléma az úgynevezett eltűnő illetve felrobbanó gradiens. Ezt a problémát próbálja a modell a residual blokkok használatával elkerülni. A modell bevezeti az ugrási kapcsolatokat (skip connection), amelyek úgy kapcsolnak össze egy korábbi és egy későbbi réteget, hogy közöttük kihagynak pár köztes réteget, ezzel egy residual blokkot hozva létre. Részletesebben, ha x a bemenet és y a kimenet egy ilyen blokknál, akkor az ugrási kapcsolat nélkül a blokk kimenete $y = f(x)$ lenne, ahol f a blokk rétegeihez tartozó függvény. Viszont az ugrási kapcsolat, amire jelen esetben identitás leképezésként érdemes gondolni, még hozzáad a kimenethez x-et, így végül $y = f(x) + x$ lesz. Ennek az egyszerű ötletnek az a haszna, hogy közelebb hozza a réteg által megtanulható input-output kapcsolatot.
Egy ilyen struktúrájú hálózat nyerte meg a ImageNet 2015-ben megrendezésre került versenyét.

2.1.2. Kimeneti réteg cseréje

Az eredeti ResNet50-nek a feladata az ImageNet 1000 osztályának a klasszifikációja volt, ezért az utolsó lineáris réteg kimenetének a mérete éppen 1000. Mivel viszont az általam használt adathalmaz osztályainak a száma 30, ezért a hálózat végét módosítanom kellett úgy, hogy csupán 30 neuron méretű legyen a kimenet. Ez a módosítás a tanítható paraméterek számának csökkenését is okozta, egészen pontosan 25 557 032 paraméter helyett a módosított hálóban már csak 23 569 502 található.

2.2. Veszteségfüggvény

A klasszifikációs modell teljesítményét mérő veszteségfüggvénynek a kereszt-entrópiát választottam. A modellben használt softmax aktivációs függvénnyel ez egy nagyon gyakran párosított veszteségfüggvény. A softmax függvény az inputként kapott 30 valós értékből a $p_1 \dots p_{30}$ valószínűségekkel tér vissza, és mi ennek az outputnak a pontosságára vagyunk kíváncsiak. Legyen $y_1 \dots y_{30} \in \{0, 1\}$ a valódi címkeértékek, a következők szerint:

$$y_i = \begin{cases} 1, & \text{ha az eredeti input az } i. \text{ osztályba tartozik.} \\ 0, & \text{ha nem.} \end{cases}$$

Ekkor kereszt-entrópia veszteségfüggvény által kapott érték

$$L = -\sum_{i=1}^{30} y_i \cdot \log_2 p_i \ .$$

Ha a megjósolt valószínűségek rosszul tükrözik az ismert címkeértékeket, akkor ez a érték nagy, ellenkező esetben pedig kicsi. Ha pedig teljesen pontos a predikció, akkor a felvett érték 0 lesz [6].

2.3. Optimalizáló algoritmus

A projekt során használt optimalizáló algoritmusnak az Adam-t illetve AdamW-t választottam. A hagyományos Adam-t a hiperparaméterek optimalizálásánál használtam, viszont a végleges tanítási fázisban már az AdamW-val dolgoztam.

2.3.1. Adam

Az Adam egy gradiens alapú optimalizáló algoritmus. Jelöljük $\mathbf{x}_t$-vel a t. ciklusban a modell súlyait tartalmazó vektort. A klasszikus esetben miután az adott ciklusban kiszámoltuk a gradiens vektort ∇f, az új súlyokat a következő módon kapjuk:

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \alpha \nabla f$$

ahol α a tanulási sebességet jelöli.
Az Adam algoritmus során számon tartjuk a korábbi gradiensek átlagának valamilyen formáját. Jelöljük a korábbi gradiensek mozgó átlagát $\mathbf{m}_t$-vel és a négyzetük mozgó átlagát $\mathbf{v}_t$-vel. Minden lépésnél a $\mathbf{g} = \nabla f(\mathbf{x}_{t-1})$ kiszámolása után a következő számításokat végezzük el:

$$\mathbf{m}_t = \beta_1 \cdot \mathbf{m}_{t-1} + (1 - \beta_1) \cdot \mathbf{g}_t$$

$$\mathbf{v}_t = \beta_2 \cdot \mathbf{v}_{t-1} + (1 - \beta_2) \cdot \mathbf{g}_t^2 \ ,$$

ahol $\mathbf{g}_t^2$-t elemenént kell érteni. β_1 és β_2 hiperparaméterek (alapértelmezett beállítás: $\beta_1 = 0.9$, $\beta_2 = 0.999$) azt szabályozzák, hogy mennyire gyorsan halványul el egy korábbi gradiens az átlagban, vagyis mennyire tekintünk a múltba a mozgó átlagok kiszámításánál.
Ezek után újra skálázzuk az átlagokat $1 - \beta^t$-el. Ez szükséges, hiszen $\mathbf{m}_0, \mathbf{v}_0 = 0$, ami azt eredményezi, hogy $\mathbf{m}_1 = 0.9 \cdot 0 + 0.1 \cdot \mathbf{g}_1 = 0.1 \cdot \mathbf{g}_1$, pedig $\mathbf{g}_1$-nek kellene lennie. Ezt úgy kapjuk, hogy $1 - 0.9 = 0.1$-el leosztunk, amire a további lépéseknél is szükségünk lesz a képlet szerint, így a következőt kapjuk:

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t}$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t}$$

Végül a súlyok megváltozását ezekből az értékekből származtatjuk:

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \alpha \cdot \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \varepsilon}$$

ε szintén egy hiperparaméter, amely a numerikus stabilitást biztosítja (alapértelmezett beállítás: 10^{-8}).
Láthatjuk, hogy az algoritmus ciklusonként eltérő lépéshosszakkal dolgozik. Ha nem változik sokat a gradiens, akkor nagyobb lépéseket teszünk, viszont nagyobb változások esetén kisebb, óvatosabb lépésekkel haladunk tovább [4].

2.3.2. AdamW

Mélytanulás során különböző regularizációs módszereket szokás használni, amely a túltanítás ellen is eredményes és a modell teljesítményét is növelheti. A hagyományos módszer az, hogy a gradiens vektor kiszámításánál végezzük el a regularizálást, viszont az Adam esetében ez kevésbé hatásos, mint más optimalizáló algoritmusok esetén, mivel a mozgó átlagokban is szerepelni fog a regularizálás, amit a képletben szereplő $\sqrt{\hat{\mathbf{v}}_t}$-vel leosztva normalizálunk. Az AdamW algoritmusa ezt a problémát próbálja megoldani azzal, hogy a mozgó átlagok kiszámítása utánra teszi a súlycsökkentést. A módosított egyenlet a következő:

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \alpha \cdot \left(\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \varepsilon} + w_t \cdot \mathbf{x}_{t-1} \right) .$$

w_t a t időben a súly csökkentésnek a mértéke. Így már természetesen nem jelenik meg a mozgó átlagokban ez a tényező, és csupán magukkal a súlyokkal lesz arányos. Az AdamW algoritmusa másban nem különbözik a hagyományos Adam-től [4].

2.4. A modell teljesítményének mérése

A modell teljesítményének mérését a tévesztési mátrixoból számolt pontosság, precizitás, szenzitivitás és F1 metrikákkal végeztem. Mivel a tárgyalt probléma a 30 osztállyon történő klasszifikálás, ezért a tévesztési mátrix mérete 30×30, jelöljük továbiakban C-vel. Egy $c_{ij} \in C$ elem reprezentálja azoknak az eseteknek a számát, amikor egy i osztálybeli elemet j-ből származónak prediktáltunk. Tehát a mátrix átlójában szerepelnek a helyesen klasszifikált esetek, azon kívül pedig a tévesztések.
A modell pontosságát a tévesztési mátrix segítségével a következő módon számolhatjuk:

$$\text{Pontosság} = \frac{\sum_{i=1}^{30} c_{ii}}{\sum_{i=1}^{30}\sum_{j=1}^{30} c_{ij}}$$

A másik fontos metrika az F1 mérőszám. Ennek a kiszámolásához először a precizitást és szenzitivitást érdemes vizsgálni. Nézzük meg ezeket az i. osztályra:

$$\text{Precizitás}_{(\text{osztály}=i)} = \frac{c_{ii}}{\sum_{j=1}^{30} c_{ji}}$$

$$\text{Szenzitivitás}_{(\text{osztály}=i)} = \frac{c_{ii}}{\sum_{j=1}^{30} c_{ij}}$$

Ezeknek a segítségével kiszámíthatjuk az i. osztályhoz tartozó F1 értéket

$$\text{F1}_{(\text{osztály}=i)} = \frac{2 \cdot \text{Precizitás}_{(\text{osztály}=i)} \cdot \text{Szenzitivitás}_{(\text{osztály}=i)}}{\text{Precizitás}_{(\text{osztály}=i)} + \text{Szenzitivitás}_{(\text{osztály}=i)}}$$

A projekt során makró F1-el dolgoztam [7], ami azt jelenti, hogy fent kapott definiált osztályonként vett F1 értékek átlagát vettem

$$\text{F1}_{(\text{makró})} = \frac{\sum_{i=1}^{30} \text{F1}_{(\text{osztály}=i)}}{30} \ .$$

Nem volt szükség súlyozott F1-et nézni, mert a tesztelő adathalmazban minden osztály elemszáma egyenlő. A későbbi mérések során pontosságot és F1 (makró) értéket vizsgálok.

3. fejezet

Hiperparaméterek optimalizálása

3.1. Az adathalamaz megvágása

Fontos jól megválasztani a tanító és tesztelő adathalmaz arányát. Úgy döntöttem, hogy véletlenszerű mintavételt alkalmazok azzal a megszorítással, hogy minden osztályból egyenlő számú adatot teszek a teszt adathalmazba. Habár nem mondható az osztályok szerint nagyon kiegyensúlyozatlannak az adat, mégis ez biztosítja, hogy a nagyobb osztályok nem nyomják el a kisebbek pontosságának mérését.
Megmértem, melyik vágás adja a legjobb eredményt. Osztályonként 20, 40, 60, 80 darab tanító adatot mintavételeztem, amely a következő eredményeket hozta:

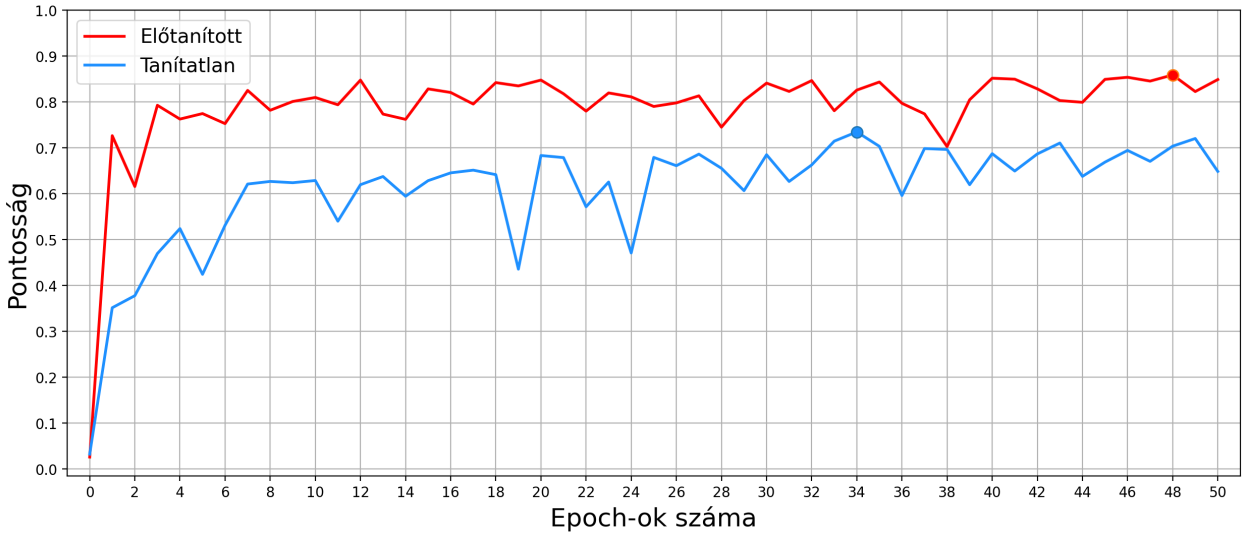
Vágás (tanító - tesztelő)	**Pontosság**	**Loss**
76% - 24%	0.7604	1.1148
82% - 18%	0.8126	0.8376
88% - 12%	0.8481	0.5902
94% - 6%	0.8592	0.5662

3.1. táblázat. A táblázatban a különböző beállításokkal tanított modellek legjobb pontossága és loss értéke szerepel. A tanítás beállításai: 50 epoch, lr = 0.001, batch = 32, előtanítást alkalmaztam.

Azt láthatjuk, hogy a legjobban teljesítő modell az volt, ami a legtöbb tanító adattal rendelkezett. Ez önmagában nem meglepő, viszont azt fontos megjegyezni, hogy egészen nagy különbségek voltak a teljesítményekben. Ezt az okozhatja, hogy egy olyan osztályból, amiben kevesebb mint 300 adat szerepel, elveszünk 80 adatot tesztelés céljából, azzal jelentősen csökkentjük tanulási lehetőségeket.
A későbbiekben ezt a 94% - 6% vágást alkalmaztam, viszont az így kiválasztott 600 tesztelésre szánt képet minden lehetséges módon megforgattam 90°-al, ezzel 2400-re növelve a teszt adathalmaz méretét.

3.2. Előtanítás

Neurális hálók tanításánál felmerülő kérdés, hogy hogyan inicializáljuk a tanítható paramétereket. Gyakran véletlenszerűen választunk értékeket, máskor viszont előtaníott paraméterekkel dolgozunk. Ez azt jelenti, hogy a hálót egy másik nagyobb adathalmazon (nehezebb feladaton) betanítjuk, majd az így megtanult paraméterekkel kezdjük el a saját tanításunkat. Ez gyakran jobb eredményeket hoz és gyorsabban tanul, mintha véletlen értékekből indultunk volna.
Én is alkalmaztam ezt a technikát. Az ImageNet adathalmazon megtanult súlyokkal inicializáltam a hálót. Maga az ImageNet egy milliós nagyságrendű adathalaz 1000 kategóriával. Fontos megjegyezni, hogy a kategóriák között nem szerepelnek műholdképek, mégis alap geometriai jellemzők jól megtanulhatók az előtanítás során, ezzel előnyt szerezve egy véletlen súlyozással szemben. Próbafuttatással vizsgáltam meg, hogy milyen különbség mérhető előtanított és tanítatlan háló között.



3.1. ábra. Az ábrán látható a két modell pontossága epoch-onként, ponttal kiemelve a legmagasabb felvett érték. A tanítás beállításai: 50 epoch, lr = 0.001, batch = 32.

Megfigyelhető, hogy az előtanított modell minden ciklusban a tanítatlannál nagyobb pontosságot ért el, és habár az 50 epoch alatt felvett maximumát később éri el, mégis mutat magas (a maximumához közeli) értékeket. Ez a kísérlet megerősíti, hogy a tárgyalt adathalmaz esetén érdemes előtanítást alkalmazni.

3.3. Batch méret

A batch méret az a paraméter, amely meghatározza, hogy egy adott iteráció során hány adatpontot (jelen esetben fényképet) használunk a modell paramétereinek frissítéséhez. Elmondhatjuk, hogy általánosan minél nagyobb batch mérettel dolgozunk, annál simább konvergenciára számíthatunk, hiszen több adatponton átlagolódnak a frissítések. Ellenben az a hátránya, hogy nagyobb memóriára van szükségünk hozzá, és gyakran nem tud annyival jobb eredményt hozni egy ilyen ciklus, mint amennyivel több időt vesz igénybe.
A következő táblázat mutatja a három különböző batch beállítással történt futtatás eredményeit:

Batch méret	**Pontosság**	**Loss**	**Futási idő (1 epoch)**
16	0.8493	0.6189	87 s
32	0.8592	0.5662	114 s
64	0.8601	0.5882	189 s

3.2. táblázat. A táblázatban a különböző beállításokkal tanított modellek legjobb pontossága, loss értéke és epoch-onként vett átlagos futási ideje szerepel. A tanítás beállításai: 50 epoch, lr = 0.001, előtanítást alkalmaztam.

Az eredmények alapján nem teljesen egyértelmű, hogy melyik beállítás a legideálisabb. Én végül a 32-es méretnél maradtam, mivel a 16-os hozzá képest, habár rövidebb futást biztosít, mégis 1%-kal rosszabb pontosságot mutat. A 64-es batch mérettel rendelkező modell az átlagos veszteségben alulmarad és a többlet 75 másodperc már nagy árnak bizonyul azért az extra 0.1% pontosságért.

3.4. Tanulási ráta

Fontos jól megválasztani azt az α (vagy lr.) paramétert, amely szabályozza a neurális hálózati súlyok módosítási lépéseinek méretét. Ez választható egy c konstansnak, amelyen a tanítás során nem módosítunk, vagy lehet egy változó is, amit valamilyen függvény szerint változtatunk. Ezen hiperparaméter optimalizálásánál öt különböző beállítással végeztem próbafuttatást. Három konstans beállítással ($\alpha = 0.01, 0.001, 0.0001$), és kettő tanulási ráta csökkentő módszerrel:

Lépcsőzetes csökkentés

A következő algoritmus szerint járunk el:

$\alpha \leftarrow 0.001$
Ha $i \equiv 0 \pmod 5$, **akkor**
$\alpha \leftarrow \alpha \cdot \sqrt[10]{0.1}$

ahol i az i. epoch-ot jelöli. Jól látszik, ez úgy lett megválasztva, hogy 50 epoch alatt csökken az α 0.0001-re.

Folytonos csökkentés

Itt annyiban változtatjuk meg az előző algoritmust, hogy minden epoch-ban elvégezzük a megfelelő szorzást, tehát most:

$\alpha \leftarrow \alpha \cdot \sqrt[50]{0.1}$

Itt $\sqrt[50]{0.1}$ szintén úgy lett választva, hogy 50 epoch alatt csökkenjen α a 0.1-szeresére. A próbafuttatások a következő eredményeket hozták:

Tanulási ráta beállítás	**Pontosság**	**Loss**
fix 0.01	0.7155	1.3203
fix 0.001	0.8592	0.5662
fix 0.0001	0.8410	0.5886
lépcsőzetes csökkentés	0.8671	0.5394
folytonos csökkentés	0.8666	0.5410

3.3. táblázat. A táblázatban a különböző beállításokkal tanított modellek legjobb pontossága és loss értéke szerepel. A tanítás beállításai: 50 epoch, batch = 32, előtanítást alkalmaztam.

A lépcsőzetes csökkentés módszere hozta a legjobb eredményt, de elmondható, hogy a folytonos módszer csupán nagyon kis különbséggel maradt le tőle mind pontosságban, mind átlagos veszteségben. A konstans beállítású modellek közül a $\alpha = 0.001$-es tanulási rátájú teljesített legjobban. A próbafuttatásoknál gyakran ezt a konstans beállítást használtam, viszont a hiperparaméterek optimalizálása után már a lépcsőzetes csökkentést alkalmaztam.

4. fejezet

Transzformációk az adathalmazon

4.1. Transzformációk

A neurális hálókkal történő képfeldolgozás során gyakran alkalmazunk különböző transzformációkat a tanító adatunkon. Ezek az augmentációk gyakran valamilyen módon véletlent tartalmazó, adatpontonként különböző műveletek. Fontos megjegyezni, hogy az adaton elvégzett augmentáció nem változtat a hozzá tartozó címkén. Transzformációk alkalmazása számos okból hasznos lehet számunkra. Bővíti és változatossá teszi az adatunkat, ami kis méretű adathalmaz esetén kulcsfontosságú. Jobban általánosítja a modellünk problémamegoldó képességét. Megnehezíti az eredeti képek pontos karakterisztikájának memorizálását, ezért a túltanulás ellen is hatásos eszköznek bizonyul. Természetesen a legjobb eredmény eléréséhez fontos jól megválasztanunk, hogy mely transzformációkat alkalmazzuk a tanítás során [3].
Ebben a fejezetben bemutatom a megfelelő transzformációkkal történt próbafuttatások eredményeit és megvizsgálom, hogy mennyivel jobb eredményt nyújtanak, mint egy transzformációk nélkül, de azonos hiperparaméterekkel tanított modell. Minden alfejezetben egy rögzített képen bemutatom az adott transzformációt öt különbözőképpen.

4.1.1. Geometriai transzformációk

A geometria transzformációk a térbeli karakterisztikáját változtatják meg a képeinknek, mint például a pozícióját, orientációját, méretét vagy perspektíváját. Három ilyen típusú transzformációt próbáltam ki, amelyek a következők voltak.

Véletlen forgatás

Ez a transzformáció elforgatja a kapott képet α szöggel, ahol α egy 0 és 360 közötti véletlen szám. Bár lehetséges leszűkíteni α tartományát, de a rövid kisérletezés után kiderül, hogy ez a beállítás adja a legjobb eredményt. Ennek természetesen van értelme, hiszen az adathalmazban szereplő légifelvételek tetszőleges forgatása egy ugyanolyan használható adatpont, mint az eredeti kép.

Eredmények:
Pontosság: 0.8592 $\longrightarrow$ 0.8994
Veszteség: 0.5662 $\longrightarrow$ 0.4982

Véletlen tükrözés

A véletlen tükrözés jelen esetben két különböző tükrözést jelent, egyet horizontálisan és egyet vertikálisan. A transzformáció során mindkettő tükrözést $p = 0.5$ valószínűséggel végezzük el. Ez természetesen nem biztosít akkora változatosságot, mint a véletlen forgatás, hiszen csupán négy lehetséges kimenete van egy adott képre nézve, de mégis a mérés alapján hasznos transzformációnak bizonyult.

Eredmények:
Pontosság: 0.8592 $\longrightarrow$ 0.8803
Veszteség: 0.5662 $\longrightarrow$ 0.4855

Véletlen kivágás

Ez a transzformáció két véletlenszerű döntést hoz. Az első, hogy hanyad része legyen a visszaadott kép az eredetinek. Ha az eredeti kép mérete T volt, akkor a visszaadott kép a T-nek x-szerese, ahol x egy 1 és 0.8 közötti véletlen szám. Tehát a kapott méret $T_c = x \cdot T$. Természetesen végig egy négyzetről beszélünk. A másik véletlen döntés, hogy mi legyen ennek az új kisebb képnek a középpontja, azzal a megkötéssel, hogy a visszaadott kép minden része rajta legyen az eredeti képen, tehát sehol ne lógjon le.

Eredmények:
Pontosság: 0.8592 $\longrightarrow$ 0.8872
Veszteség: 0.5662 $\longrightarrow$ 0.4934

4.1.2. Szín alapú transzformációk

A szín alapú transzformációk a képek színcsatornáit módosítják. Ezekkel a módszerekkel jól lehet az eredeti képek fényességét, kontrasztját, szaturációját, stb. megváltoztatni. A következő transzformációkat próbáltam ki:

Színzaj

Ez egy összetettebb transzformáció, mivel a kép fényességét, kontrasztját, szaturációját és hue értékét egyszerre módosítja (véletlenszerűen). A módosítások mértéke külön-külön nehezen mérhető, de a PyTorch eszközeit használva egy 0 és 1 közötti skálán könnyen szabályozható. Rövid kísérletezés után itt is kedvező eredmények születtek.

Eredmények:
Pontosság: 0.8592 ⟶ 0.8733
Veszteség: 0.5662 ⟶ 0.5097

Gauss Homály

Ez a transzformáció a kép homályosságáért felelős. Nem mindig előnyös a homályosítás, de kis mértékben javíthatja a teljesítményt, hiszen a képek különböző látási viszonyokat befolyásoló körülmények között készültek. Természetesen a homályosítás mértékéért egy véletlen változó felel.

Eredmények:
Pontosság: 0.8592 ⟶ 0.8643
Veszteség: 0.5662 ⟶ 0.5283

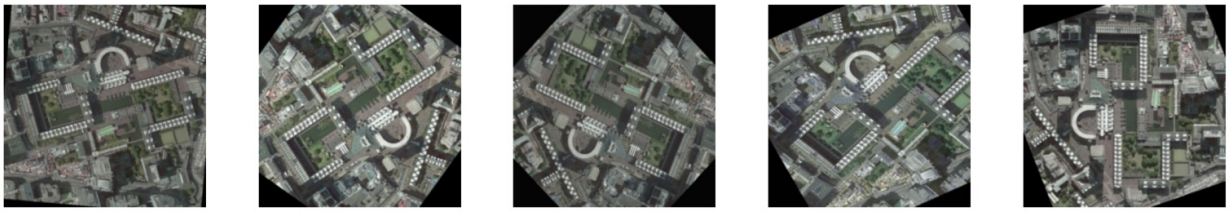
4.2. Transzformációk kiválasztása

A korábban megvizsgált transzformációkról most el kellett dönteni, hogy melyeket válasszuk ki a tanítási fázisra, illetve melyek azok, amiket nem használunk a későbbiekben [3]. Láttuk, hogy külön-külön mindegyik transzformáció javított a teljesítményen, de ebből nem következik, hogy az együttes használatuk adja a legjobb eredményt. Éppen ezért újabb próbafuttatásokra volt szükség, amik során kipróbáltam különböző kombinációit a transzformációknak.
Nem végeztem el minden lehetséges kombináció szerint próbatanítást, de a geometriai transzformációk összes kombinációját kipróbáltam, illetve az együttesükhöz hozzávettem a szín alapú transzformációk kombinációit, és azokkal is elvégeztem a megfelelő méréseket. Az alábbi táblázat mutatja, milyen eredmények születtek:

Transzformációk	**Pontosság**	**Loss**
VF, VT	0.9113	0.3579
VF, VK	0.9128	0.3675
VT, VK	0.9173	0.3372
VF, VT, VK	0.9198	0.3418
VF, VT, VK, SZ	0.9249	0.3131
VF, VT, VK, GH	0.9122	0.3375
VF, VT, VK, SZ, GH	0.9179	0.3236

4.1. táblázat. A táblázatban a különböző transzformációkkal tanított modellek legjobb pontossága és loss értéke szerepel. Jelölés: VF = véletlen forgatás, VT = véletlen tükrözés, VK = véletlen kivágás, SZ = színzaj, GH = gauss homály. A tanítás beállításai: 50 epoch, lr = 0.001, batch = 32, előtanítást alkalmaztam.

A legjobban teljesítő modell az lett, amelyik a gauss homály kivételével mindegyik transzformációt használta, pontossága is kiemelkedő, de veszteség terén is ez a beállítás hozta a legjobb eredményeket. Látható, hogy ez nem egy véletlen okozta apró különbség, ezért a későbbiekben ezzel a transzformáció csomaggal dolgoztam.
Az alábbi ábra szemlélteti, hogy ez a gyakorlatban hogyan néz ki:



A kiválasztott geometriai transzformációkat alkalmazva újra elvégeztem egy hiperparaméter optimalizálást, de az eredmények alapján nem volt szükség változtatásra.

5. fejezet

Tanítás és kiértékelés

5.1. Tanítás

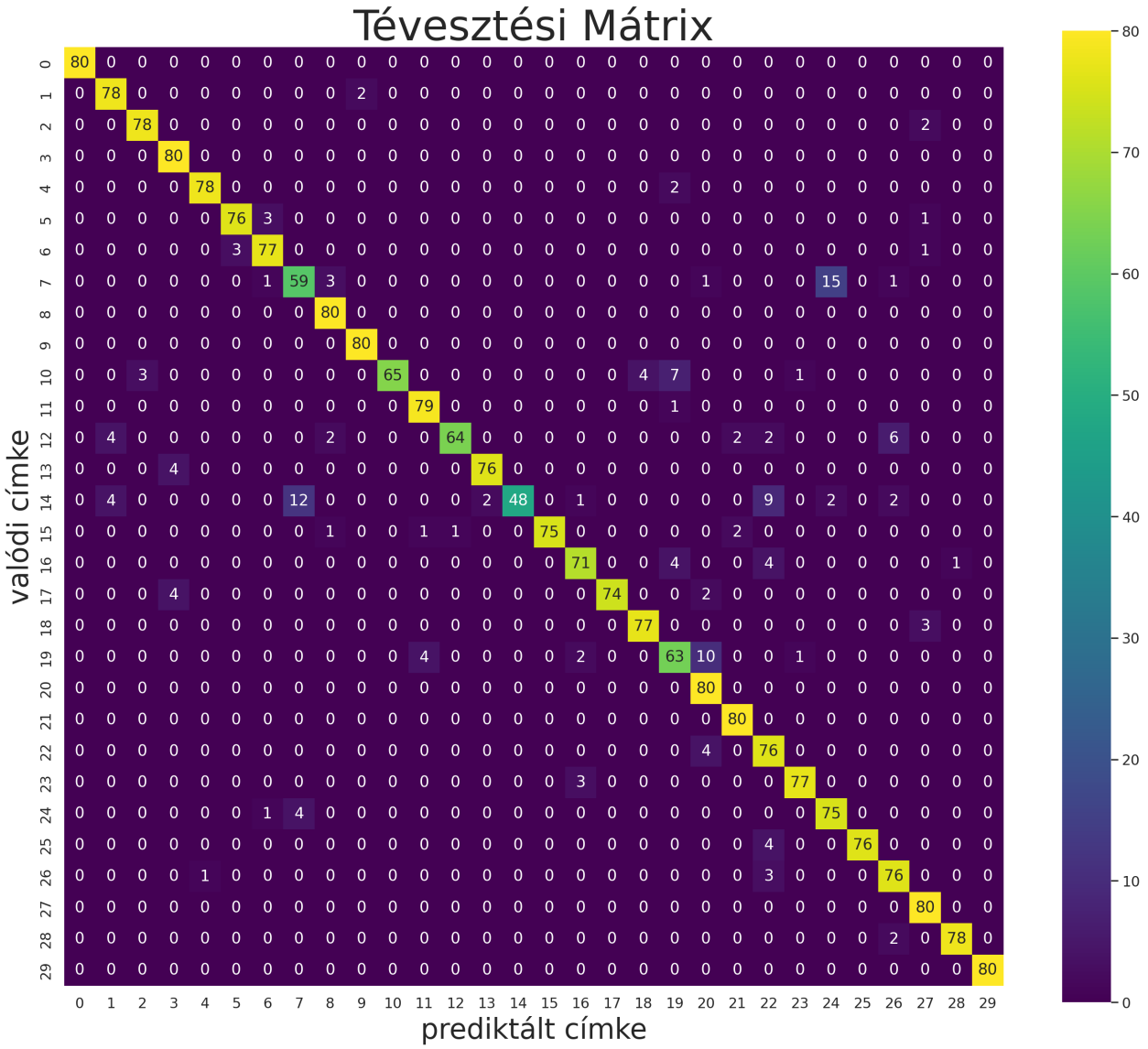
Az optimalizált hiperparaméterekkel és a kiválasztott transzformációkkal lefuttattam az első próbatanítást. A tanítás beállításai a következők voltak:

- Háló: ResNet50 (30 széles kimeneti réteggel)
- Háló súlyainak inicializálása: előtanítás
- Veszteségfüggvény: kereszt-entrópia
- Optimalizáló algoritmus: AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$)
- Tanulási ráta: lépcsőzetesen csökkenő (kezdetben $\alpha = 0.001$)
- Batch méret: 32
- Epoch-ok száma: 50
- Leállási feltétel: nem volt
- Adathalmaz mérete: 9600

A tanítás hossza 127 perc volt. A kód Python nyelven készült, a neurális háló megvalósítását a PyTorch csomag segítségével végeztem. Google Colab felületén dolgoztam és az ott elérhető NVIDIA Tesla T4 GPU-t használtam a tanítás során.

5.2. Eredmények kiértékelése

A tanítás során elért legjobb ereményt a modell a 34. epoch után produkálta. Ekkor a következő eredmények születtek:



5.1. ábra. A normalizálatlan tévesztési mátrixot láthatjuk. Az oszlopok és sorok indexének magyarázata a Függelékben található.

- Pontosság = 0.9312
- F1 = 0.9300

A tanítás eredményes volt. Mind az F1, mind a pontosság megfelelően magas értéket mutat. Érdemes megvizsgálni a tévesztési mátrixot. Az látható, hogy a legtöbb osztályra jól működik a modellünk, viszont van néhány, ami kiugróan alulteljesít a többihez képest, ilyen például a tavak osztálya (19) vagy a farmoké (10). Legtöbbször csupán egy-egy véletlenszerű félreklasszifikálás lehet az oka a modell tökéletlenségének, viszont észrevehető, hogy vannak visszatérő hibái, például a tavakat gyakran kikötőknek gondolja.
A modellünk teljesítményét azzal tudnánk a leghatékonyabban növelni, hogyha ezeknél a kritikus osztályoknál nagyobb pontosságot érnénk el úgy, hogy a többi osztályon a modell teljesítménye nem romlik. A paraméterek megváltoztatásával kísérleteztem, de nem vezetett jobb eredményre semelyik alternatíva. Az epoch-ok számának növelése sem jelentett megoldást.

5.3. Ötlet a modell fejlesztésére

A modell teljesítményének javítására az volt a megközelítés, hogy az adathalmaz méretét növeljük, egészen pontosan azon osztályok elemeinek a számát emeljük, amelyeken gyengén teljesített a háló.
Természetesen a legegyszerűbb az lett volna, ha egy másik adathalmaz képeit használom, de szimulálni szerettem volna, hogy csak az eredeti adathalmaz áll rendelkezésre. Olyan technikára volt tehát szükség, ami az adathalmazunk statisztikájával megegyező statisztikájú adatokat gyárt, és ezekkel bővíti a megfelelő osztályokat. Az egész művelet alapjául pedig csupán az eredeti adatunkat használhatjuk fel. Erre a megoldást a GAN-ok jelentették.

6. fejezet

Szintetikus adat generálása

6.1. GAN modell

A GAN (Generative Adversarial Networks) modell egy széles körben elterjedt generáló módszert hozott a mélytanulás világába. Gyakran használják képgenerálásra, felbontás növelésére, stílus átvitelre, képek színezésére és adathalmazok növelésére.

6.1.1. Alapötlet

A GAN modell alapötlete, hogy létrehozunk két különböző neurális hálót, az egyiket generátornak a másikat pedig diszkriminátornak nevezzük. A diszkriminátor feladata, hogy megállapítsa egy adott bemenetről, hogy az eredeti adathalmaz része vagy sem. A generátor célja pedig, hogy olyan kimenetet generáljon, amelyet inputként a diszkriminátornak adva az félreklasszifikálja, vagyis az eredeti adathalmazból származónak gondolja azt.
A két háló tanítása párhuzamosan történik egy (későbbiekben tárgyalt) összetett veszteségfüggvény segítségével. Ideális esetben a generátor megtanul az eredeti adathalmaz elemeitől megkülönböztethetetlen adatot gyártani.
A GAN-ok tanításának nehézségét az jelenti, hogy a modell érzékeny a hiperparaméterei beállítására, amelyek egyensúlyának megtalálása kihívást jelent. E mellett gyakran előforduló probléma, hogy a modell olyan állapotba áll be, amiben nagyon hasonló kimeneteket generál. Ez nem minden esetben jelent gondot, de a legtöbb feladat során inkább elkerülendő [1].

6.1.2. Generátor és diszkriminátor

Egy GAN struktúrának sokféle formája lehet. Én most egy általános képgeneráló modellt mutatok be, amelyhez hasonlót a későbbiekben használtam. Természetesen most is szükségünk van egy tanító adathalmazra. Jelöljük az elemeit d-vel.
A generátor, jelölje G, egy olyan háló, ami bemenetként valamilyen formában zajt kap,

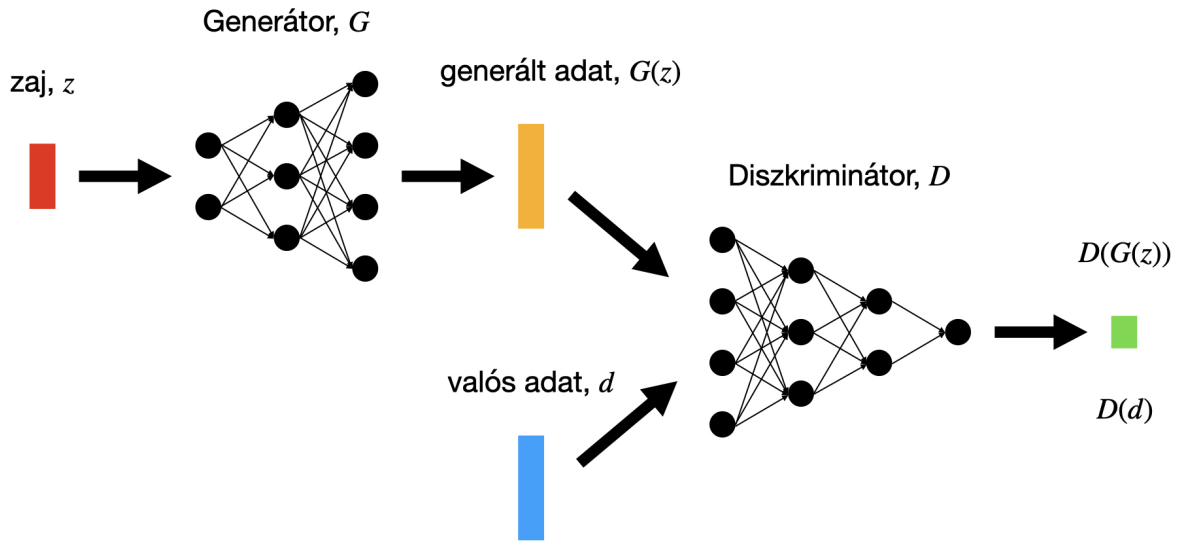
kimenetként pedig az adathalmazunk elemeivel megegyező formátumú generált adatot ad vissza. Ezt az úgy nevezett zajt tipikusan egy z vektorként kell elképzelni (természetesen lehet ez mátrix vagy magasabb dimenziós tenzor is), melynek elemei:

$$z_i \in [0,1] \text{ véletlen szám, } i = 1, \ldots, n \ ,$$

ahol n a vektor hosszát jelöli. A z érték minden generáló lépésben újrasorsolódik, ez biztosítja, hogy a generátor mindig más és más $G(z)$ kimenetet adjon.
A diszkriminátor, jelöljük D-vel, bemenetként megkaphatja egy d elemét az eredeti adathalmaznak, vagy pedig egy generátor által előállított $G(z)$ elemet. Feladata megálapítani, hogy az input valós vagy generált adat volt, ezért kimenet 0 és 1 közötti szám lesz. A diszkriminátorra nézve ideális eset, ha a visszaadott érték:

$$D(x) = \begin{cases} 0, & \text{ha } x = G(z), \\ 1, & \text{ha } x = d \ . \end{cases}$$

Közben persze a generátor célja, hogy a diszkriminátor félreklasszifikálja, vagyis $D(G(z)) = 1$ legyen [1].



6.1. ábra. Az ábrán egy GAN modell adatfolyama látható.

A két neurális hálózat nem csak egyszerű sűrű hálók lehetnek, hanem tetszőleges más struktúrák, amelyekre a fenti feltételek teljesülnek. Én a későbbiekben a DCGAN (Deep Convolutional GAN) modellt használtam, amelyben a generátor és diszkriminátor is egy konvolúciós neurális hálózat.

6.1.3. Veszteségfüggvény

A GAN modell során használt veszteségfüggvény a következő:

$$V(D,G) = \mathbb{E}_{x\sim p_d(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

Láthatjuk, hogy két részből áll a tárgyalt veszteségfüggvény. Az első reprezentálja a diszkriminátor predikcióját a valós adatokon, a második pedig a generált adatokon. $\mathbb{E}_{x\sim p_d(x)}[\log D(x)]$ jelenti a várható értékét a diszkriminátor kimenetének logaritmusának, amikor a bement a valós adathalmaz eloszlásából származik. Ezt úgy kell elképzelni, hogy ha adunk a diszkriminátornak egy adag valós adatot, akkor mi az átlaga a predikcióknak logaritmussal skálázva. $\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$-nél hasonló a helyzet, csak most az $1 - D(G(z))$ logaritmusát vesszük, ahol $D(G(z))$ a diszkriminátor predikciója a generált adaton [1]. A várható értéke persze most is felfogható mint az átlaga a predikcióknak, amikor zajt adunk a generátornak.
A diszkriminátor azt szeretné, hogy az első kifejezés értéke nagy legyen, hiszen ez azt jelenti, hogy képes jól megállapítani egy valós adatról, hogy tényleg valós. A egyenlet második fele szintén végiggondolható, hogy a diszkriminátor maximalizálni szeretné, viszont a generátor célja a minimalizálás, hiszen meg szeretné téveszteni a diszkriminátorunkat. Ebből adódik a következő optimalizálási feladat:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_d(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

Ez a függvény pontosan akkor éri el a minimumát, amikor a generált adatok eloszlása a valós adatok eloszlásával megegyezik [1].
Ezt a problémát megoldó tanító algoritmus egy ciklusa a következő:

for $1, \ldots, n$ **do**
  for $1, \ldots, k$ **do**
    `vegyünk` m `zaj mintát` $\{z_1, \ldots, z_m\}$ `és adjuk oda a generátornak`
    `vegyünk` m `valós adatot` $\{x_1, \ldots, x_m\}$
    `frissítsük a diszkriminátor súlyait a gradiensvektor hozzáadásával:`
    $\uparrow \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log(D(x_i) + \log(1 - D(G(z_i)))]$
  end for
  `vegyünk` m `zaj mintát` $\{z_1, \ldots, z_m\}$ `és adjuk oda a generátornak`
  `frissítsük a generátor súlyait a gradiensvektor kivonásával:`
  $\downarrow \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z_i)))$
end for

Itt n a tanító iterációk számát jelölim k pedig egy batch mérethez hasonló hiperparaméter.

6.2. A megépített GAN modell

A projekt során használt diszkriminátor egy hagyományosabb konvolúciós háló volt. Hét konvolúciós réteget tartalmazott, amelyeket az úgynevezett szivárgó ReLU (LeakyReLU) aktivációs függvény követett. Ezalól kivételt képez az utolsó réteg, amelynek a végén szigmoid található, hogy a korábban említett feltételeknek megfelelő kimenetet kapjunk. Ezek mellett még rétegenként végrehajtottam egy normalizálást.
Generátornak valamilyen értelemben a diszkriminátor tükörképét használtam. Ehhez hét dekonvolúciós réteget alkalmaztam, amelyeket a diszkriminátorral szemben itt egyszerű ReLU követett. A háló legvégén pedig hiperbolikus tangens (elemenként véve) biztosítja a kimenet elvárt formáját. A modell alapötletének forrása: [5].

Diszkriminátor:

- 7 konvolúciós réteg
- 5 batch normalizáló réteg
- 6 szivárgó ReLU
- 1 szigmoid aktivációs függvény

Paraméterek száma: 2 765 568

Generátor:

- 7 dekonvolúciós réteg
- 6 batch normalizáló réteg
- 6 ReLU
- 1 hiperbolikus tangens

Paraméterek száma: 3 576 704

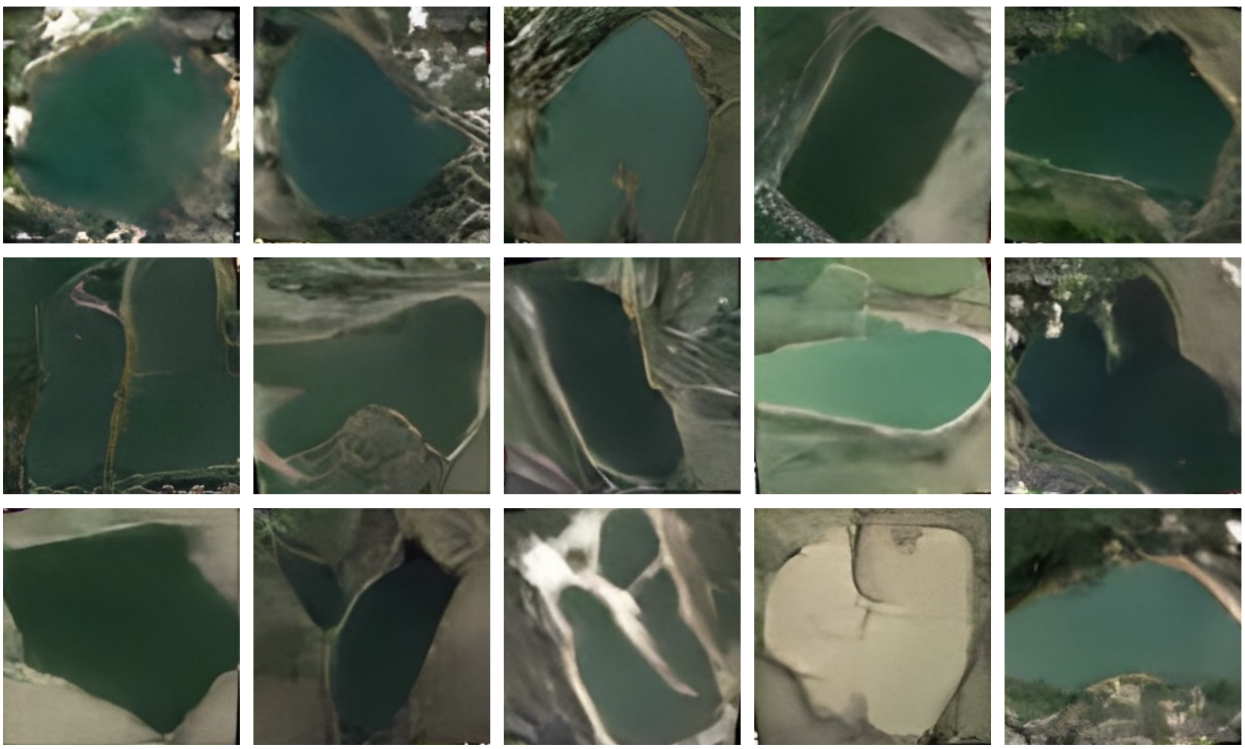
6.3. Generáló futtatások

A fent definiált modellen ötször hajtottam végre tanító futtatást. Minden tanítás alatt egy-egy olyan osztályt vettem tanító adathalmaznak, amelyen rosszabbul teljesített korábban a klasszifikációs modell. Így tehát a tanítás végére lett öt generátorom, amelyek mindegyike képes a megfelelő osztályhoz adatot gyártani. Mivel a tanító adathalmaz egy ilyen tanításnál nagyon kicsi, ezért most az epoch-ok számát nagyobbnak választottam. A tanítás során a hiperparaméterek optimalizálásánál kiválasztott transzformációkkal dolgoztam. A tanítás a következő beállításokkal történt:

- Háló: generátor és diszkriminátor
- Háló súlyainak inicializálása: véletlen
- Veszteségfüggvény: min-max GAN loss
- Optimalizáló algoritmus: Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e^{-8}$)
- Tanulási ráta: 0.0001
- Batch méret: 64
- Epoch-ok száma: 10000
- Leállási feltétel: nem volt

6.4. Eredmény és adatbővítés

A GAN modell teljesítményének mérésére a legalkalmasabb módszer, ha megvizsgáljuk, hogy a generátor milyen kimenettel tér vissza pár véletlen bemenet esetén. A következő ábrán látható pár próbafuttatás eredménye:



6.2. ábra. Tanítás után a tavak adathalmazához tartozó generátor háló által generált 15 kép.

Láthatjuk, hogy a tanítás sikeres volt. Habár nem hibátlanok a képek, mégis kivehető, hogy tavakat ábrázolnak. Minden generálás egy teljesen egyedi képet adott vissza, tehát nem futott bele a modell abba a GAN-okat jellemző hibába, hogy nem eléggé változatos a kimenet. Az eredeti adathalmazhoz hasonlóan itt is láthatunk természetesebb formájú tavakat vagy éppen szögletesebb, ember által formáltabbakat. Egyes képeken még akár olyan részletek is megjelennek, mint egy homokos partszakasz vagy éppen egy zátony.
Az adathalmaz megfelelő osztályait bővítettem generált képekkel. Mindegyik érintett osztályhoz 300 képet adtam hozzá, ezzel közelítőleg megkettőztem a méretüket. Az tanító adatmennyiség így 10 900 képre nőtt.

7. fejezet

A szintetikus adatokkal bővített tanítás és kiértékelése

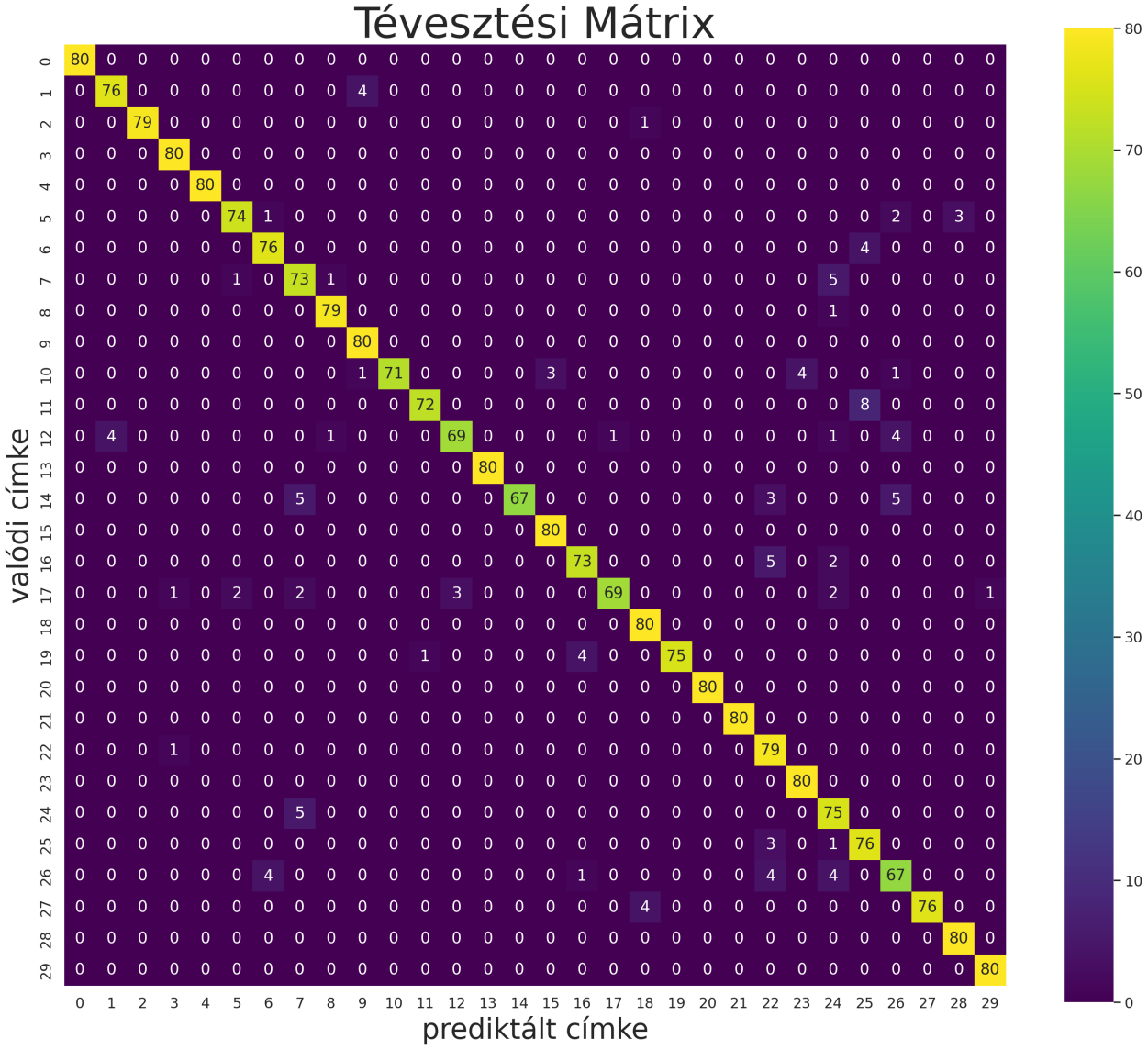
7.1. Tanítás

Miután bővítettem az adathalmazomat a generált képekkel, újra elvégeztem a modell tanítását. A beállítások kisebb változtatással a korábbihoz hasonlóak voltak:

- Háló: ResNet50 (30 hosszú kimeneti réteggel)
- Háló súlyainak inicializálása: előtanítás
- Veszteségfüggvény: kereszt-entrópia
- Optimalizáló algoritmus: AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e^{-8}$)
- Tanulási ráta: lépcsőzetesen csökkenő (kezdetben $\alpha = 0.001$)
- Batch méret: 32
- Epoch-ok száma: 50, 100
- Leállási feltétel: nem volt
- Adathalmaz mérete: 10 900

Csupán az adathalmaz méretében történt módosítás, ami hosszabb futási időt eredményezett, egészen pontosan 151 perc volt a Google Colab NVIDIA Tesla T4 GPU-ját használva. Elvégeztem egy újabb futtatást 50 helyett 100-as epoch számmal, de nem hozott jobb eredményeket, mivel a modell a teljesítményének maximumát az első 50 epoch alatt érte el.

7.2. Eredmény kiértékelése



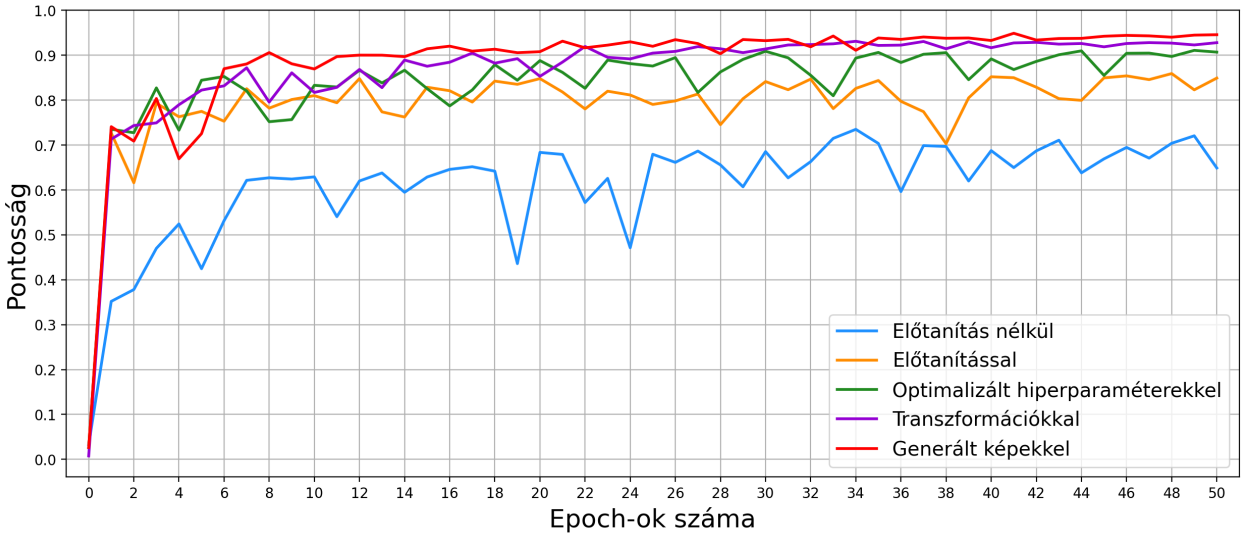
7.1. ábra. A normalizálatlan tévesztési mátrixot láthatjuk. Az oszlopok és sorok indexének magyarázata a Függelékben található.

- Pontosság = 0.9525
- F1 = 0.9525

Minden metrika szerint jobb teljesítményt nyújt az új modell, mint a korábbi. Több mint 2%-al javult a modell pontossága. Ez azzal magyarázható, hogy a korábbi modell bizonyos osztályokon jelentősen rosszabbul teljesített, mint a többin, azaz itt koncentrálódott a félreklasszifikált képek nagy része. Ezt sikerült orvosolni azzal, hogy a képgenerálás segítségével javítottuk ezen osztályok pontosságát. Megfigyelhető, hogy más osztályokon kissé romlott a modell teljesítménye, de összességében az egész tesztelő adathalmazt vizsgálva láthatjuk, hogy jobb eredmény született.
Korábbi tanítás után megfigyelhettünk gyakran előforduló hibákat, azaz olyan eseteket, amikor a modell az egyik osztálybeli elemeket tévesen egy másikba sorolta. Ezeket a tévesztési mátrix átlóján kívül eső nagy számok reprezentálják. Láthatjuk, hogy ilyen téren is javult az eredmény, hiszen kevesebb és kisebb ilyen tévesztés figyelhető meg.

7.3. Összehasonlítás korábbi modellekkel

Érdekes összevetnünk a projekt kölönböző stádiumában használt modellek tanulási görbéjét:



7.2. ábra. Az ábrán látható a projekt öt modell tanításának pontossága epoch-onként.

Amit a projekt során a pontosság mérésével eddig is láthattunk, hogy hogyan fejlődött a modell, azt most ezen az ábrán is megfigyelhetjük. Az első néhány epoch-ot leszámítva, ahol még nagy szórása van a modelleknek, és pár teljesítménybeli bezuhanástól eltekintve szépen látszik, hogy hogyan viszomyulnak egymáshoz az események.
Legrosszabbul teljesítő kipróbált modell az előtanítás nélkül futtatott volt. Lassan tanulta meg az adathalmaz jellegzetességeit, és alig tudott elérni 0.7-es pontosság feletti értéket. A legnagyobb teljesítménybeli ugrást az jelentette, amikor elkezdtem az előtanítás által kapott

súlyokkal inicializálni a hálót. Ez nem csak a modell legjobb pontosságát növelte, hanem a tanulás sebességén is gyorsítani tudott. További látványos javulást lehetett elérni a megfelelő hiperparaméterek megválasztásával. Ekkor már a modell pontossága 0.9 feletti értékeket is képes volt felvenni. A következő fejlesztést a megfelelő transzformációk kiválasztása jelentette, amiket a tanító adathalmazon alkalmazva a modell teljesítménye újabb ugrást mutatott. A projekt során mért legjobb eredményt végül szintetikus adat generálásával lehetett elérni, ekkor a modell stabilan 0.9 feletti pontosságot mutatott, olykor 0.95-ös értéket is elérve.

Az adott modellek legjobb teljesítményét a következő táblázatban láthatjuk:

Modell	**Pontosság**
Előtanítás nélkül	0.7350
Előtanítással	0.8592
Optimalizált hiperparaméterekkel	0.9098
Transzformációkkal	0.9300
Generált képekkel	0.9525

7.1. táblázat. A táblázatban a különböző beállításokkal tanított modellek legjobb pontossága szerepel.

8. fejezet

Modell tesztelése adathalmazon kívüli képeken

A legjobb eredményt elérő modellt teszteltem olyan valós képeken is, amelyek nem szerepelnek az eredeti adathalmazban. Összeállítottam egy 200 képből álló adathalmazt, amelyben hasonló felbontású négyzetes képek szerepelnek. A képeket a Google Maps segítségével gyűjtöttem. Ezek többsége Magyarország területéről származik, de hogy minden osztályt lefedjek más területekről is gyűjtöttem képeket, hiszen például a sivatag és a tengerpart osztályt is szerettem volna reprezentálni. Az így kapott adathalmaz minden elemére készítettem egy predikciót a háló segítségével.
Mivel ez egy kézzel összeállított adathalmaz, ezért előfordulnak olyan esetek, amikor nehezen lehetett eldönteni, hogy téved a modell vagy sem. Ez a kérdés főleg azoknál az osztályoknál került elő, amelyek a terület lakottságát írják le. Azt is érdemes megjegyezni, hogy két téves predikció között is van különbség, hiszen például egy ovális alakú bevásárlóközpontot stadionnak kategorizálni kisebb hiba, mint hegységnek.
Szigorúan véve ezen az adathalmazon a modell 156 képet klasszifikált sikeresen, ami a képek 78%-át jelenti. Ez rosszabb teljesítményt jelent, mint amit az eredeti adathalmazon nyújtott. Ilyenfajta romlást gyakran lehet tapasztalni, amikor egy másik adathalmazon teszteljük a modellünket. Ezt magyarázhatja az adathalmaz jellegzetességeinek túltanulása, adateloszlások közötti különbségek, a tanító adathalmaz túl kicsi mérete vagy valami egyéb környezetbeli különbség.
A következő ábrán látható az új adathalmaz néhány eleme és a hozzájuk tartozó predikciók:

8.1. ábra. 20 kézzel válogatott teszt kép és a modell hozzájuk tartozó predikciója.

Láthatjuk, hogy a természetesebb tájegységeket jól kategorizálja. Folyók, tavak, hegyek, erdők és mezőgazdasági területek hiba nélkül szerepelnek. A hibák többsége lakóövezetek és épületek félreklasszifikálásánál fordult elő.
A teszt eredménye alapján azt mondhatjuk, hogy habár a modell nem tudja ugyanazt a teljesítményt nyújtani, mint az eredeti adathalmazon, mégis egészen komplex mintázatok felismerését sikeresen megtanulta és ezekre képes pontos predikciót nyújtani.

Összefoglalás

A dolgozat célja egy műholdképeket tartalmazó adathalmaz által definiált klasszifikációs probléma minél magasabb szintű megoldása volt. Ezt a klasszikus adatbányászati feladatot a dolgozat a mélytanulás újszerű megoldásaival közelítette meg. Erre példa a ResNet50 és az AdamW használata, vagy az adatbővítési módszerek, transzformációk, valamint a generatív eljárások használata.

A fejezetek a neurális háló alapú modell fejlesztésének egy-egy lépését mutatták be. Ezek a lépések a hagyományos adatbányászati modellfejlesztő eljárás szerint épültek egymásra, és összeségében a jelenlegi legerősebb gyakorlat módszereit mutatták be és alkalmazták. A projekt során mérések segítségével láthattuk, hogy egy adott lépés milyen mértékben járult hozzá a modell teljesítményéhez. Egy az adatbányászatban gyakran előforduló problémát is tárgyalt a dolgozat, mégpedig a rendelkezésre álló adat elégtelenségét. Ismerteti az erre itt megoldást jelentő GAN modell működését és implementációját, illetve bemutatja ennek alkalmazását szintetikus adatok generálásával.

A dolgozat végére az elmélet megfelelő alkalmazásával és a paraméterek pontos beállításával létrejövő modell képes lett magas szintű teljesítményt nyújtani mind az eredeti teszt adathalmazon, mind az adathalmazon kívülről származó képek osztályozásán.

Az olvasót a dolgozat megismertette képi klasszifikációs feladatok egy lehetséges megoldásával, és azt is megmutatta, hogy a mélytanulási módszerek igen hatékony eszközt jelentenek számunkra egy ilyen komplex probléma kezelése során.

Függelék

- Az AID adathalmaz: https://captain-whu.github.io/AID/
- A projekt során használt Python kód: https://github.com/BenjaminToth/BScThesis
- Magyarázata a tévesztési mátrix oszlopainak és sorainak indexéhez: Airport = 0, BareLand = 1, BaseballField = 2, Beach = 3, Bridge = 4, Center = 5, Church = 6, Commercial = 7, DenseResidential = 8, Desert = 9, Farmland = 10, Forest = 11, Industrial = 12, Meadow = 13, MediumResidential = 14, Mountain = 15, Park = 16, Parking = 17, Playground = 18, Pond = 19, Port = 20, RailwayStation = 21, Resort = 22, River = 23, School = 24, SparseResidential = 25, Square = 26, Stadium = 27, StorageTanks = 28, Viaduct = 29.
- A dolgozatban szereplő képek Python nyelven készültek.

Irodalomjegyzék

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[4] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. 2018.

[5] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[6] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

[7] Ian H Witten, Eibe Frank, Mark A Hall, Christopher J Pal, and Data Mining. Practical machine learning tools and techniques. In *Data Mining*, volume 2, page 4. Morgan Kaufmann, 2016.