



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

TDK DOLGOZAT

OE-NIK
2023

Hallgató neve:
Hallgató törzskönyvi száma:

Csamangó Bence
T/008489/FI12904/N

VALÓS TERÜLETEK GENERÁLÁSA ÉS FÁK DETEKTÁLÁSA

Csamangó Bence

Óbudai Egyetem

NEUMANN JÁNOS INFORMATIKAI KAR, BSc IV. évfolyam

Konzulens: Szénási Sándor, egyetemi tanár

A dolgozat a háromdimenziós területek generálásának és fák detektálásának lehetséges módszereit tárgyalja. A kutatás célja olyan módszerek és algoritmusok kidolgozása, amelyek lehetővé teszik a valósághű háromdimenziós környezeti fájlok létrehozását, valamint a fák pontos és hatékony felismerését és azonosítását. A bevezető rész ismerteti az eddig már bemutatott megoldásokat, és röviden leírja, hogy miként épülnek fel a magasság térképek és a műholdképek. A dolgozat első része bemutatja miként lehet magasság térképek segítségével olyan állományt generálni, ami megjeleníthető erre specifikus programokban. Ezt követően a fő témára tér át, ami bemutatja, hogy az ezekhez a területekhez tartozó műholdképeken miként lehetséges a fák detektálása. A dolgozat részletezi a lehetséges megoldásokat és emellett kitér a jelenleg legoptimálisabbnak tartott megoldásra. A dolgozatban bemutatott algoritmus képes megtalálni és felismerni a fákat a megadott műholdképek bemeneti adatai alapján. Az algoritmus hatékonyan kezeli a kihívásokat, mint például a fák ágai és levelei közötti átfedések, vagy a más tárgyakkal való összetéveszthetőség. Mivel lényegi részben képfeldolgozásra épül így, az ebből eredő módszereket is részletezi a tanulmány. A befejező részben bemutatja a teljes megoldást, annak pontosságát és természetesen annak a való életben lehetséges felhasználási területét, emellett összegzi a végeredményt és tárgyalja az esetleges jövőbeli fejlesztések lehetőségét is. Az eddigi kísérletek azt mutatják, hogy az itt kidolgozott módszerek és algoritmusok hatékonyak és egész pontosak a háromdimenziós területek generálásában és a fák detektálásában. Ezek az eredmények új lehetőségeket nyitnak a különböző alkalmazások számára, mint például a virtuális valóság, a szimulációk, vagy bármely más program, ami földrajzi adatokkal dolgozik akár háromdimenzióban.

TARTALOMJEGYZÉK

1	BEVEZETÉS	1
1.1	Előszó.	1
1.2	Hasonló fejlesztések	1
1.2.1	World Machine	1
1.2.2	L3DT	2
1.2.3	Unreal Engine	2
1.2.4	Unity Engine	2
2	TEREP RENDERELÉS	3
2.1	Level of detail (LOD)	3
2.1.1	Discrete LOD.	3
2.1.2	Continuous LOD	4
2.1.3	View-Dependent LOD	5
2.1.4	Hierarchical LOD	6
2.1.5	LOD-Sprite technika	7
2.2	Voxel modellek	7
2.3	Fájlkiterjesztések	8
2.3.1	obj. foramtum	8
2.3.2	COLLADA .dae.	9
2.3.3	ESRI Grid ASCII .asc	10
2.3.4	.3ds formátum	11
3	MŰHOLDKÉP FELDOLGOZÓ TECHNIKÁK.	13
3.1	Gépi tanulás	13
3.1.1	Felügyelt tanulás (Supervised learning).	14
3.1.2	Felügyelet nélküli tanulás (Unsupervised learning)	14
3.1.3	Fél-felügyeleti tanulás (Semi-supervised learning)	14

3.1.4	Megerősítéses tanulás (Reinforcement learning)	15
3.2	Gépi tanulás algoritmusok	15
3.2.1	Bináris döntési fa	15
3.2.2	Support Vector Machines (SVM)	16
3.2.3	K-legközelebbi szomszéd (KNN)	17
3.2.4	Zsákolás	17
3.2.5	Boosting	19
3.2.6	AdaBoost	19
3.2.7	LightGBM	20
3.2.8	FastTree	21
3.3	Mély gépi tanulás	22
3.3.1	Mesterséges neurális hálózatok	22
3.3.2	Konvolúciós neurális hálózatok	23
3.3.3	Rekurrens neurális hálózat	24
4	SZÍNMODELLEK	26
4.1	RGB Színmodell	26
4.2	HSV Színmodell	27
4.3	CIE Lab	27
4.4	CIE Luv	28
4.5	CIE XYZ Színmodell	28
4.6	Szürkeárnyaltos színmodell	29
4.6.1	Konverziók	29
5	ZAJSZŰRÉS ÉS EGYÉB JAVÍTÁSOK	31
5.1	Simítás	31
5.1.1	Beltrami szűrő	31
5.2	Gauss-szűrő	32
5.3	Gauss derivált-szűrő	33

6	SZOFTVERTERV	34
6.1	Magasságtérképek beolvasása és mentése.	34
6.2	Növényzet detektálása	34
7	MEGVALÓSÍTÁS	36
7.1	Magasságtérkép beolvasása	36
7.2	Exportálás	38
7.3	Növényzet detektálás	40
7.3.1	Feature értékek számítása	40
7.3.2	Label értékek számítása.	45
7.3.3	Tanítás	47
8	EREDMÉNYEK	50
8.1	Terület fájl generálás	50
8.2	Növények detektálása	51
8.3	Összehasonlítás	53
8.4	Felhasználás	55
8.5	Tapasztalatok	56
9	TERVEK A JÖVŐRE NÉZVE	56
10	Összegzés.	57
11	IRODALOMJEGYZÉK	58
12	ÁBRAJEGYZÉK.	60

1 BEVEZETÉS

1.1 Előszó

Maga az alap ötlet 3D területek generálása magasság térkép(heightmap)-ek segítségével, majd ezeken a térképeken objektumok (fák, bokrok és további növényzetek) elhelyezése műholdképek alapján. A cél egy olyan alkalmazás/szoftver megvalósítása, amely ezt képes kezelni, és a mai kornak megfelelő megoldást tud nyújtani a felhasználók számára. Ebben az esetben ez azt jelenti, hogy a beolvasott terület, egy arra alkalmas megjelenítőben helyesen rajzolódik ki, és jól reprezentálja az általa fedett valós területet. A fák detektálása esetén pedig a cél egy minél pontosabb és gyorsabb rendszer lefejlesztése, ami szintén megfelel a mai kor technológiai elvárásainak. A bementi kép alapján adjon vissza egy olyan képet/állományt, ami pontosan jelöli az adott képen a fák által lefedett területet. Magát a magasságtérképet és a műholdképet is egy bemenetként kapja meg szoftver. Először legenerálja a háromdimenziós térképhez tartozó kimeneti fájlt majd, ha ez elkészült akkor a megadott műholdkép alapján kijelöli, hogy mely területeken kell növényzetet/fákat elhelyezni. Ennek a megvalósításához többféle algoritmust és metódust is fel lehet használni, a következőkben ezeket is részletezni fogom.

1.2 Hasonló fejlesztések

1.2.1 World Machine

A World Machine egy olyan fejlesztésű szoftver, amivel képesek vagyunk 3D területek ábrázolására, szerkesztésére, textúrázására. A műveleteket valós időben tudjuk végrehajtani és a mai elvárásoknak megfelelően ezeket a lehető leggyorsabban teszi meg. Lehetőségünk van akár elektronikus rajztábla segítségével szerkeszteni, ami elég sokat tud segíteni. A program mindezt képes 8K felbontásban megjeleníteni. Ha esetleg valamit elrontanánk a program egy verzió követővel is fel van vértézve így, ha valamit elrontunk könnyedén vissza tudjuk állítani az ezt megelőző állapotot.

A szoftver lehetőséget nyújt saját, illetve véletlenszerűen generált térképek létrehozására. A saját térkép létrehozásához szintúgy, mint az általam elképzelt programban egy magasság térképre van szükség. A random generáláskor pedig meg kell adnunk, hogy mi a minimális, illetve a maximális magasság és hogy mekkora területet szeretnénk [1].

1.2.2 L3DT

Az L3DT egy Windows-alkalmazás tereptérképek és textúrák generálására. Célja, hogy segítse a játékfejlesztőket és a digitális alkotókat hatalmas, kiváló minőségű 3D világok létrehozásában. Előnye fentebb említett programmal szemben, hogy ingyenesen használható. Maga az alkalmazás támogatja a scripteket így könnyen személyre szabható a felhasználónak szánt interfész. Szintén lehetőségünk van választani saját, illetve random generált térkép megjelenítése között. E mellett valós időben szerkeszthetjük a 3D térképünket (Fontos, hogy akár 2D-s és 3D-s nézetben egyaránt) [2].

1.2.3 Unreal Engine

Az Unreal Engine a világ egyik legfejlettebb valós idejű 3D fejlesztő környezete. A játékfejlesztők és a különböző iparágakban dolgozó alkotók számára lehetővé teszi a következő generációs valós idejű 3D tartalmak (játékok, animációk, filmek) létrehozását nagyobb szabadsággal és rugalmassággal, mint valaha. Az Unreal Engine 5 a legújabb verziója ennek az eszköznek. Ez a verzió továbbfejlesztett funkciókat kínál, amelyek még nagyobb szabadságot és rugalmasságot biztosítanak a fejlesztők számára. Ilyen funkció például az új „NANITE” technológia, ami egy új geometriai renderelő megoldás, amely lehetővé teszi a natív, filmminőségű eszközök közvetlen beillesztését az adott jelenetekbe.

A NANITE technológia alapötlete az volt, hogy a játékterekbe a sokszor nagyon részletes modelleket és textúrákat a nélkül hozza létre, hogy a fejlesztőknek gondoskodniuk kelljen a modellek optimalizálásáról vagy a textúrák méretezéséről. A NANITE lehetővé teszi, hogy a játék motorja közvetlenül kezelje és megjelenítse az eredeti, nagyon magas felbontású modelleket és textúrákat, így a játékosok számára valóságosabb és részletgazdagabb vizuális élményt nyújthat [3].

1.2.4 Unity Engine

A Unity Engine egy erőteljes és népszerű játékfejlesztő motor, amit először 2005-ben mutattak be. Az Unity rendkívül sokoldalú, és lehetővé teszi játékok és interaktív tartalmak fejlesztését szinte minden platformra, beleértve a PC-t, mobil eszközöket, konzolokat és VR eszközöket is. A Unity egyszerűen kezelhető grafikus felülettel rendelkezik, ami segíti a fejlesztőket a játékok készítésében és a vizualizációban. A motor széleskörű támogatást nyújt a 2D és 3D grafikákhoz, fizika szimulációkhoz, hangkezeléshez és animációkhoz. Emellett rendelkezik egy gazdag ökoszisztémával, amely rengeteg eszközt, plugin-t és erőforrást kínál a fejlesztőknek [4].

2 TEREP RENDERELÉS

A projekt témájához erősen kapcsolódik maga a 3D megjelenítés is, például ha később saját megjelenítőt szeretnénk fejleszteni hozzá. Emiatt ebben a fejezetben a különböző 3D megjelenítési technikákat fogom részletezni egészen a régebbi technológiáktól kezdve a ma legjobban elterjedt megoldásokig.

2.1 Level of detail (LOD)

„Level of Details” másnéven részletességi szint. Jelenleg 4 fajtáját különböztetjük meg, ezek általában felhasználási módjukban térnek el egymástól. Ami mindben azonos az maga az alap koncepció. Ugyanis az alapelve, hogy egy modellt háromszögek segítségével írunk le. Minél több háromszögből áll fel annál részletesebb. Ezt optimalizálni is egész jól lehet hisz gondoljunk bele, ha az adott objektum messzebb van azt nem kell „teljesen” feldolgozni hisz nem látjuk részletesen mi magunk sem tehát kevesebb háromszöggel is megoldható így a távoli objektumok nem fogyasztanak feleslegesen erőforrást. Ezért is használják ezek algoritmusok egyikét valós idejű megjelenítésekhez.

Geometria: A geometriai LOD a 3D modellek geometriájának egyszerűsítését jelenti, ahogy azok a kamerától távolodnak. Ez az egyszerűsítés gyakran az objektum ábrázolásához használt poligonok számának csökkentésével történik. Például egy, a nézőhöz közeli fa több száz vagy ezer poligonon ábrázolható a bonyolult részletek megragadása érdekében, míg ugyanez a fa a távolban csak néhány tucat poligonon ábrázolható a számítási erőforrások megtakarítása érdekében.

Textúra: A textúra LOD a távoli objektumokhoz alacsonyabb felbontású textúrákat használ. A textúrák olyan képek, amelyeket 3D modellekre alkalmaznak, hogy színt, részletességet és felületi tulajdonságokat adjanak nekik. A textúrák LOD-ja azt jelenti, hogy a távolabbi objektumokhoz kisebb és kevésbé részletes textúrákat használunk, így kímélve a memóriát és a feldolgozási teljesítményt.

2.1.1 Discrete LOD

A „Discrete Level of Detail” magyarul „Diszkrét Részletességi szint” hagyományos megközelítést alkalmaz. Minden objektumhoz külön folyamatban hoz létre LOD-okat. Adott kritériumok alapján futás közben választja ki az adott objektum LOD-ját. Ezért is nevezik Diszkrétnek [5]. Jelentős előnye, hogy viszonylag egyszerűen leprogramozható, mai hardverekkel is jól alkalmazható és ami a legfontosabb, hogy viszonylag gyors. Cserébe viszont vannak hátrányai is. Nem alkalmazható túlságosan nagy objektumokon, illetve előfordulhat, hogy egyes poligonokat nem megfelelően helyez el

A diszkrét LOD legfontosabb alapelvei:

- **Fix szintek:** előre meghatározott részletességi szinteket használ, amelyek mindegyike a nézőtől mért távolságok meghatározott tartományához kapcsolódik. Ezeket a szinteket gondosan választják ki, hogy egyensúlyt teremtsenek a vizuális minőség és a számítási hatékonyság között.
- **Váltási mechanizmus:** Ahogy a nézőpont vagy a kamera elmozdul a jelenetben, a rendszer dinamikusan váltogat az előre meghatározott részletességi szintek között a néző objektumoktól való távolsága alapján. Ez a váltás hirtelen történik, ami lépcsőzetes változást okoz a részletességben.
- **Hatékonyság:** A diszkrét LOD optimalizálja a renderelést azáltal, hogy elkerüli a folyamatos számításokat, amelyek a LOD-technikák zökkenőmentes átmenetéhez szükségesek. Ez egyszerűsíti az erőforrás-kezelést és csökkenti a számítási igényt.

A diszkrét LOD számos előnnyel rendelkezik:

- **Kiszámítható teljesítmény:** Mivel a diszkrét LOD egy rögzített szintkészletre támaszkodik, kiszámítható és konzisztens teljesítményjellemzőket biztosít, ami alkalmasra teszi a valós idejű alkalmazásokhoz.
- **Erőforrás-kezelés:** A diszkrét kapcsolási mechanizmus egyszerűsíti az erőforrás-kezelést, lehetővé téve a számítási erőforrások, a memória és a tárolás hatékony használatát.
- **Vezérlés:** fejlesztők minden egyes LOD-szinten pontosan szabályozhatják az objektumok megjelenését, így biztosítva, hogy a vizuális minőség az egész jelenetben magas maradjon.
- **Kompatibilitás:** A diszkrét LOD a hardver- és szoftverplatformok széles skálájával kompatibilis, így jó választás lehet különböző grafikai fejlesztésekhez.

2.1.2 Continuous LOD

A „Continuous LOD”, avagy „Folyamatos/folytatólagos Részletességi szint” eltér a Diszkrét társától. A Diszkréttel szemben ez a verzió olyan adatstruktúrát használ, amiből a módosítani kívánt szint futás közben is kivehető. Általában ez hatékonyabb futásidőt eredményezhet. Előnye, hogy a részletességi szintek itt előre definiáltak tehát egy adott objektum nem használ fel többet a szükségesnél viszont mikor egy objektum részletessége adott pillanatban nem számottevő annak csökkenti a részletességi szintjét és egy nála nagyobb prioritást élvező objektum kapja meg. Emiatt jobb az erőforrás kihasználása és általánosságban elmondható, hogy részletesebben és valósághűbben adja vissza számunkra a modelleket [5].

A folytonos LOD fontosabb alapelvei:

- **Zökkenőmentes átmenet:** vizuálisan kellemes átmenetet biztosít a részletességi szintek között, elkerülve a diszkrét megközelítésekre jellemző hirtelen váltásokat. Ahogy az objektumok vagy jelenetek mozognak, a rendszer dinamikusan módosítja a részletesség szintjét a megfelelő vizuális élmény fenntartása érdekében.
- **Interpoláció:** zökkenőmentes átmenetek biztosítása érdekében a Continuous LOD gyakran alkalmaz interpolációs technikákat. Ezek a technikák összekeverik az adatokat és attribútumokat a szomszédos részletességi szintek között, lehetővé téve a zökkenőmentes vizuális változásokat.
- **Alkalmazkodóképesség:** részletesség szintje folyamatosan alkalmazkodik a néző objektumokhoz való közelsége alapján, ami valósághű vizuális élményt eredményez.

A folytonos LOD számos előnyt hoz magával:

- **Realizmus:** hozzájárul a vizuális jelenetek valósághűségéhez és elmélyüléséhez, lehetővé téve a zökkenőmentes átmeneteket és a vizuális hűség fenntartását az objektumok mozgása közben.
- **Irányítás:** a fejlesztők a LOD-átmenet minden egyes szakaszában finomra szabottan szabályozhatják a részletesség szintjét, biztosítva a kívánt vizuális minőség elérését.
- **Magával ragadó élmény:** az olyan alkalmazásokban, mint a játékok és a VR, fokozza a mélységérzetet azáltal, hogy zökkenőmentesen alkalmazkodik a néző perspektívájához.
- **Sokoldalúság:** a hardver- és szoftverplatformok széles skálájához igazítható, így alkalmazkodik a különböző grafikai fejlesztési foratókönyvekhez.

2.1.3 View-Dependent LOD

A „View-Dependent” azaz „Nézet függő Részletességi szint” alapelve, hogy a poligonok megjelenítése, illetve részletessége attól függ, hogy mi azt milyen szögből látjuk. Tehát egyetlen objektumnak több részletszintje is lehet (Amit látunk és közel van az teljes részletességében le van generálva míg mondjuk a másik oldala csak minimálisan, mert számunkra az felesleges, mivel nem látjuk). A közeli részeket nagy felbontásban jeleníti meg, a távolabbiakat pedig kevésbé részletesen. Előnyei közé tartozik, hogy alapjában véve jobb a szemcsézettsége, mint a folytonos LOD-é. Ez annak köszönhető, hogy az algoritmus oda osztja ki a legtöbb sokszöget, ahol a legnagyobb szükség van rá. Ha kicsit másképp tekintünk rá akkor akár felfoghatjuk optimalizációként is hisz így akár nagyobb modellek is könnyen megjeleníthetők.

A nézetfüggő LOD alapelvei a következőket foglalják magukban:

- **Nézőközpontú megközelítés:** a néző látómezejében lévő objektumokat helyezi előtérbe, több részletet és erőforrást rendelve ezekhez az objektumokhoz, miközben csökkenti a látómezőn kívüli objektumok részletességét.
- **Dinamikus adaptáció:** ahogy a néző mozog vagy változtatja a perspektíváját, a nézetfüggő LOD dinamikusan módosítja az objektumok részletzettségi szintjét a vizuális minőség fenntartása, és a renderelési teljesítmény optimalizálása érdekében.
- **Valós idejű interakció:** ez a technika különösen értékes az olyan alkalmazásokban, ahol a valós idejű interakció és a 3D-s jelenetek megléte elengedhetetlen, mint például a videojátékok, a virtuális valóság (VR).

A nézetfüggő LOD használatának előnyei:

- **Javított realitás:** a részletszintek dinamikusan beállításával a néző perspektívája alapján fokozza a realitásérzetet.
- **Hatékony erőforrás-felhasználás:** ez a technika optimalizálja az erőforrás-elosztást azáltal, hogy a renderelési erőforrásokat a nézőponton belüli objektumok számára prioritizálja, ezáltal javítva a teljesítményt.

2.1.4 Hierarchial LOD

A 4. ilyen részletezési szint a Hierarchikus. Őt a kisebb objektumokra fejlesztették ki mivel ez eddig említett társai kevésbé hatékonyan tudták modellezni. Az objektumokat úgynevezett „szerelvényekbe” szedi össze. Megfelelő távolság esetén egyszerűsíti ezeket a csoportokat, de fontos, hogy nem magukat az objektumokat. A hierarchikus verzió szépen összeegyeztethető a nézetfüggő megoldással. Jobb méretezhetőséget biztosít.

A hierarchikus LOD alapelvei a következők:

- **Hierarchikus szervezés:** az objektumok vagy jelenetek hierarchiába vannak szervezve, több részletességi szinten keresztül. Jellemzően a hierarchia magasabb szintjei durvább, míg az alacsonyabb szintek részletesebb ábrázolásokat tartalmaznak.
- **Erőforrás-hatékonyosság:** a hierarchikus LOD hatékonyan kezeli az erőforrásokat azáltal, hogy az objektumokat a megfelelő részletességű szinthez rendereli. Ez biztosítja a számítási erőforrások optimális elosztását, ami jobb teljesítményt eredményez.
- **Dinamikus alkalmazkodás:** bár nem olyan dinamikus, mint a nézetfüggő vagy a folyamatos LOD-technikák, a hierarchikus LOD mégis képes bizonyos mértékig alkalmazkodni olyan tényezőkhez, mint a nézőtávolság, a képernyő mérete vagy más renderelési korlátok.

A hierarchikus LOD előnyei a következők:

- **Erőforrás-kezelés:** a hierarchikus LOD az objektumok hierarchikus szervezésével és megfelelő részletességű megjelenítésével hatékonyan kezeli a számítási erőforrásokat, ami jobb teljesítményt eredményez.
- **Skálázhatóság:** ez a technika jól skálázható összetett környezetek esetén, és a hardverkonfigurációk széles skálájához igazítható, így alkalmas mind a „high-end”, mind az „low-end” rendszerekhez.
- **A részletesség egyensúlya:** a hierarchikus LOD egyensúlyt teremt a vizuális minőség és a teljesítményhatékonyság között, lehetővé téve a kiterjedt, részletes jelenetek ábrázolását.
- **Sokoldalúság:** a hierarchikus LOD számos grafikus alkalmazásban használható, így sokoldalú eszköz az összetett 3D környezetek renderelésének optimalizálásához.

2.1.5 LOD-Sprite technika

Ez a technika a „Level of Detail” és „Image-based modelling and rendering (IBMR)” összeépítésével jön létre. Az alap ötlet az adatkészlet nagy felbontású reprezentációjának generált nézetek gyorsítótárazása. Az algoritmus a későbbi képkockákat kisebb felbontásban jeleníti meg viszont alkalmazza textúraképként. Azokat a hibákat méri, amiket a nézőpont és a kulcsképkocka eltérése okoz mivel minden új nézet le van generálva. Ha ez a hiba túllép egy küszöbértéket akkor az algoritmus új kulcskockát generál [6].

2.2 Voxel modellek

A VOXEL modellek jól használhatók különféle, aránytalan adatbázisból származó adatok megjelenítésére/reprezentálására. Emiatt sok 3D-s objektum ábrázolásakor használták, valamint térképek ábrázolására is [7]. A VOXEL modellek képesek akár folytonos, „törésmentes” térképeket megjeleníteni. Ez annyit takar, hogy a domborzatokról elmondható, hogy nem tartalmaznak durva éleket, míg a „hálós” modellek esetén ez sűrűn előfordulhat.

Ezek után merül fel a kérdés, hogy akkor miért nem elterjedt például a videójáték iparban? Ez a kérdés természetesen jogos is. Előnyei mellett sajnos a VOXEL modelleknek vannak hátrányai is. Példának okáért felhozhatjuk a perspektívákat: Sajnos ezzel a megjelenítési eljárással olyan problémába ütközhetünk, hogy adott dőlésszögekből eldeformálódás figyelhető meg ami nem minden esetben megengedhető (Főleg, ha egy nagyon pontos térképre van szükségünk). A másik hatalmas probléma, hogy a mai hardverekhez párosítva elavult. Ugyanis a mai videógyorsítók poligonok feldolgozására lettek specializ-

zálva nem pedig „volexek”-re azaz a mai hardverekkel egy nagyobb objektum generálása jóval időigényesebb lenne, mint egy poligon alapú implementációval. Emiatt sem használják esetleges előnyei ellenére sem a videójáték iparban.

A voxelek kis téglalap alakú blokkok, melyeket magasságtérképből vagy DEM-fájlból hoztak létre. A téglalap méreteit x és y irányú lépések határozzák meg. Hasonlóan a pixelekhez a volexek is kezelik az adatokat csak éppen nem 2D-ben hanem 3D-ben. Közelebből több részletet látunk, távolabbról kevesebbet. Az éleket (zajokat) végpontok közötti interpolációval simítjuk.

2.3 Fájlkiterjesztések

A *terep renderelés* szekcióban még érdemesnek tartom az ehhez tartozó, különböző fájlformátumok bemutatását. Habár a fő téma nem a háromdimenzionális megjelenítés, 3D objektumokat eltároló fájlformátumra szükség lesz, hiszen a magasságtérképből kinyert adatokat valamilyen formátumban tárolni kell, ha azt egy erre alkalmas programban kívánjuk megjeleníteni. A 3D objektumok és terepadatok tárolására szolgáló általános fájlformátumok közé olyan formátumok tartoznak, mint például az OBJ, az STL és a COLLADA (DAE). Ezek a formátumok lehetővé teszik a 3D geometria, a textúrák és más releváns információk hatékony tárolását. Emellett képes még például a kapcsolódó földrajzi és térbeli adatok kezelésére. A „Geographic Information System” röviden GIS olyan formátumai is alkalmazhatók, mint a GeoTIFF vagy a Shapefile. A megfelelő fájlformátum kiválasztása nemcsak az adattárolás szempontjából lényeges, hanem a különböző szoftverekkel és platformokkal való kompatibilitás biztosítása szempontjából is.

2.3.1 obj. formátum

Az obj fájlformátum egy szövegalapú, nyílt fájlformátum, amelyet a Wavefront Technologies fejlesztett ki. A formátumot más 3D-s grafikai alkalmazások gyártói is átvették, és számos ilyen alkalmazás képes importálni/exportálni. Az obj fájlformátum több sorból áll, amelyek mindegyike egy kulcsot és különböző értékeket tartalmaz. A kulcs minden egyes sor a következő típusinformációt jelzi. Emiatt az obj fájlformátum nem igényel fejléct [8]. Az .obj fájlok következő tulajdonságokkal bírnak:

- **Geometriai adatok:** Az .obj fájl elsődleges célja az adott objektum vagy jelenet 3D geometriájára vonatkozó információk tárolása. Ide tartoznak a csúcsok (pontok a 3D térben), a csúcsnormálisok (a felületre merőleges irányok minden egyes csúcsnál) és a textúrakoordináták (UV-koordináták, amelyeket a modellek textúrázása során használnak).
- **Felületek:** Az .obj fájlok a 3D objektum felületeit határozzák meg a csúcspontokra

való hivatkozáshoz használt indexek segítségével. Egy felület lehet három vagy több csúccsal rendelkező sokszög.

- **Materiál:** Az .obj fájlok olyan információkat is tárolhatnak, mint például a modell különböző részeihez tartozó szín-, fényesség- és textúra-leképezések. Ezeket az információkat általában egy külön .mtl (Material Template Library) fájlban tárolják, amelyre az .obj fájl hivatkozik.
- **Simító csoportok:** definiálhatnak simító csoportokat, amelyek meghatározzák, hogy a normálisok hogyan interpolálódnak a szomszédos felületek között. A simító csoportok segítenek az ívelt felületek sima megjelenését létrehozni.
- **Csoportosítás és objektumhierarchiák:** Az .obj fájlok csoportokat és objektumokat határozhatnak meg a jeleneten belül. Ez lehetővé teszi a több komponensből álló összetett 3D modellek szervezését és strukturálását.
- **Megjegyzések:** megjegyzéseket is el lehet helyezni a sorok „#” szimbólummal történő előtagozásával. A megjegyzéseket a szoftver értelemszerűen figyelmen kívül hagyja.
- **Kompatibilitás:** számos 3D modellező és renderelő szoftvercsomag támogatja, így a különböző alkalmazások közötti 3D adatcseréhez gyakran használják. Ez azonban egy viszonylag egyszerű formátum, és nem feltétlenül támogatja a fejlettebb 3D fájlformátumok, például az .fbx vagy a .dae (Collada) összes funkcióját.

```
# This is a comment
v 0.0 0.0 0.0      # Vertex 1
v 1.0 0.0 0.0      # Vertex 2
v 0.0 1.0 0.0      # Vertex 3

f 1 2 3             # Triangle face
```

2.1. ábra: .obj Fájl felépítés

2.3.2 COLLADA .dae

A Collaborative Design Activity (COLLADA) egy nyílt XML-alapú fájlformátum, amelyet úgy terveztek, hogy lehetővé tegye a 3D digitális eszközök cseréjét különböző szoftverek és hardverek között [9]. A projektet 2004-ben kezdeményezte a Khronos Group, egy olyan vállalatokból és magánszemélyekből álló cégcsoport, amely a grafika és a multimédia nyílt szabványainak kidolgozására törekszik. A COLLADA-t azzal a céllal fejlesztették ki, hogy megoldja a 3D grafikai iparágat sújtó inkompatibilitási problémákat.

A fejlesztésben számos vállalat, köztük a 3D grafika és játékipar olyan jelentős szereplői is részt vettek, mint például az NVIDIA, a Sony Computer Entertainment és az Autodesk. Ez a közös munka egy olyan sokoldalú és bővíthető fájlformátum létrehozását eredményezte, amely képes a 3D geometria, animációk, textúrák, shaderek és sok más ábrázolására.

A COLLADA legfontosabb jellemzői és előnyei:

- **Átjárhatóság:** egyik elsődleges célja az volt, hogy lehetővé tegye a különböző 3D alkalmazások és platformok közötti átjárhatóságot. A COLLADA ezt úgy éri el, hogy olyan szabványosított formátumot biztosít, amelyet a szoftvereszközök széles köre többek között a 3D modellező szoftverek, animációs szoftverek, játékmotorok és renderelő alkalmazások tud olvasni és írni.
- **Bővíthetőség:** XML-alapú szerkezete lehetővé teszi a könnyű bővítést és testre szabást. Ez a rugalmasság létfontosságú a 3D grafikai ipar folyamatosan fejlődő igényeinek kielégítéséhez, ahol folyamatosan új funkciókat és technikákat fejlesztenek ki.
- **Platform-függetlenség:** nem kötődik egy adott operációs rendszerhez vagy hardverplatformhoz. Ez ideális választássá teszi a különböző eszközöket és platformokat célzó projekteken dolgozó fejlesztők számára, az asztali számítógépektől kezdve a játékkonzolokon át a mobileszközökig.
- **Átfogó adatrepresentáció:** a 3D adatok széles skáláját képes reprezentálni, beleértve a geometriai modelleket, animációkat, fizikai tulajdonságokat, anyagokat, árnyékolókat és még sok mást. Ez az átfogó representáció alkalmassá teszi a legkülönbözőbb alkalmazásokhoz, a videojátékoktól kezdve az építészeti vizualizációión át a filmgyártásig.

2.3.3 ESRI Grid ASCII .asc

A „Geographic Information Systems” (GIS) fontos szerepet játszik a modern térelemzésben, lehetővé téve számunkra a földrajzi adatok rögzítését, tárolását, kezelését és megjelenítését. Az ESRI Grid ASCII (*.asc) egy alapvető fájlformátum a térinformatikai rendszerek területén, amely lehetővé teszi a „rácsos” adatok megjelenítését. Gyakran csak ASCII Grid formátumként emlegetik, a GIS alkalmazásokban általánosan használt raszteres fájlformátum. Az „Environmental Systems Research Institute” (ESRI) által kifejlesztett formátumot a GIS-szoftverek széles körben támogatják, így jó választás lehet a térbeli adatok tárolására [10] [11].

Egy ESRI Grid ASCII fájl több összetevőből áll, amelyek együttesen egy raszteres adatkészletet határoznak meg:

- **Fejléc:** tartalmazza a rács metaadatait, beleértve a rács méreteit (sorok és oszlopok száma), a földrajzi kiterjedést (határoló koordináták), a cellaméretet és az adatok formátumát (egész vagy lebegőpontos értékek).
- **Adatértékek:** Az adatértékek alkotják a fájl elsődleges tartalmát, táblázatos formában elrendezve. Minden érték a rács egy cellájának felel meg, és soronként vannak rendezve. Ezek az értékek a térinformatikai információk széles skáláját képviselhetik, mint például a tengerszint feletti magasság, a hőmérséklet vagy a földhasználati besorolás.
- **Választható NODATA érték:** A fejlécben egy opcionális NODATA érték van megadva, amely az érvényes adatokkal nem rendelkező cellákat jelzi. Ez kulcsfontosságú a rácsháló hiányzó vagy nem meghatározott információinak kezeléséhez.

Az ESRI Grid ASCII formátum előnyei:

- **Átjárhatóság:** A GIS-szoftverek széles körben támogatják az ESRI Grid ASCII fájlokat, ami biztosítja a különböző platformok és alkalmazások közötti kompatibilitást és egyszerű adatcserét.
- **Olvashatóság:** Az ASCII formátum ember által olvasható, így a felhasználók számára hozzáférhetővé teszi az adatok közvetlen ellenőrzését, szerkesztését vagy módosítását szövegszerkesztőkkel.
- **Méret:** A bináris formátumokhoz képest az ASCII fájlok viszonylag kis méretűek, így alkalmasak e-mailben vagy webes letöltéseken keresztül történő megosztásra.
- **Metaadatok megőrzése:** Az ESRI Grid ASCII fájlok fejlécadatai alapvető metaadatokat tartalmaznak, biztosítva az adatok eredetét és értelmezhetőségét.
- **Sokoldalúság:** A formátum sokoldalú, és a térinformatikai adattípusok széles skáláját képes megjeleníteni, így különböző alkalmazások és iparágak számára alkalmas.

2.3.4 .3ds formátum

A 3ds fájlformátum az AutoDesk 3ds Max szoftverének elsődleges formátuma. Ez egy bináris formátum, amely különböző információkat tartalmazó részekből áll. Ezek a részek tartalmaznak egy azonosítót, amely jelzi, hogy milyen információt tárolnak, valamint a következő részhez képest való eltolást. Így azok a szoftverek, amelyek nem támogatnak bizonyos renderelési tulajdonságokat, egyszerűen figyelmen kívül hagyhatják azokat. A 3ds fájlformátum támogatja a következőket: geometria csúcsok/felületek, textúrák, fizikai anyagtulajdonságok, transzformációk, kamerainformációk és fények. A fájlformátum eredete az 1990-es évek elejére nyúlik vissza, amikor az Autodesk bemutatta a 3D Studio szoftver első verzióját. Ez a szoftver, amelyből végül 3ds Max lett, kulcsfontosságú szerepet játszott abban, hogy a 3D modellezés szélesebb körben is elérhetővé vált. A 3ds

fájlformátumot e szoftver alapértelmezett fájlformátumaként fejlesztették ki, és egyszerűsége és sokoldalúsága miatt vált népszerűvé.

A 3ds fájlformátum alapvetően egy bináris fájlformátum, amelyet a 3D-modelladatok hatékony tárolására terveztek. A formátum különböző darabokból áll, amelyek mindegyike meghatározott célt szolgál a 3D modell meghatározásában. A 3ds-fájl néhány kulcsfontosságú összetevője a következő:

- **Fejléc rész:** az alapvető információkat tartalmaz a fájlról, például a verzióját és a teljes fájl méretet.
- **Szerkesztői rész:** a fájl létrehozásához használt 3D Studio Max verzióra vonatkozó információkat tartalmaz.
- **Objektumdarabok:** a 3ds-fájl „szíve”, amelyek a jelenetben található 3D-objektumok adatait tartalmazzák. Minden objektumdarab az objektum geometriájára, anyagaira és transzformációira vonatkozó információkat tartalmaz.
- **Materiális részek:** a jelenetben használt anyagokkal kapcsolatos adatokat tárolja, beleértve a textúratérképeket, színeket és árnyékolókat.
- **Textúra:** az objektumokra alkalmazott textúrákra vonatkozó információkat tartalmazza, amelyek a renderelés és a megjelenítés szempontjából kulcsfontosságúak.
- **Animációs adatok:** az animációs adatok tárolására szolgál, lehetővé téve a kulcskockák, az animációvezérlők és a kapcsolódó paraméterek tárolását.

3 MŰHOLDKÉP FELDOLGOZÓ TECHNIKÁK

A programnak a specifikációk szerint képesnek kell lennie műholdképek alapján helyesen kijelölni, hogy mely területekre kell növényzetnek(fáknak) kerülnie. Ehhez többféle megoldás is használható, viszont érdemes figyelembe venni ezeknek a megoldásoknak az hatékonyságát is. Ennek fényében végig járom/bemutatom az ehhez felhasználható algoritmusokat, majd egy későbbi fejezetben taglalom a végül kiválasztott megoldási formát és annak kiválasztásának okát.

3.1 Gépi tanulás

A gépi tanulás (Machine learning - ML) a számítástechnika azon részterülete, amely Arthur Samuel 1959-es megfogalmazása szerint "a számítógépek képesek tanulni anélkül, hogy kifejezetten programoznák őket" [12]. A mesterséges intelligenciában a mintafelismerés és a számítógépes tanuláselmélet tanulmányozásából fejlődött ki, a gépi tanulás olyan algoritmusok tanulmányozását és felépítését vizsgálja, amelyek képesek adatokból tanulni és előrejelzéseket tenni, az ilyen algoritmusok a szigorúan statikus programutasítások követését meghaladva adat vezérelt előrejelzéseket vagy döntéseket hoznak, a minta bemenetekből történő modellépítés révén. A gépi tanulást számos olyan számítástechnikai feladatban alkalmazzák, ahol a jó teljesítményű explicit algoritmusok tervezése és programozása nehéz vagy kivitelezhetetlen.

A gépi tanulás sokféle alkalmazási területen használható, például képfelismerés, beszéd felismerés, termékajánlások, pénzügyi elemzések, autonóm járművek és még sok más területen. Az ML rendszerek képesek önállóan tanulni és alkalmazkodni az új adatokhoz, így nagyban hozzájárulnak az automatizáláshoz és a döntéshozatalhoz a valós időben. Fontos megjegyezni, hogy a gépi tanulásnak többféle típusa van, beleértve a felügyelt tanulást, a felügyelet nélküli tanulást és a megerősítéses tanulást, amelyek mindegyike más-más alkalmazási területekre és feladatokra alkalmazható.

A gépi tanulás két legelterjedtebb módszere a felügyelt tanulás és a felügyelet nélküli tanulás. A gépi tanulás mintegy 70 százaléka felügyelt tanulás, míg a felügyelet nélküli tanulás aránya 10-20 százalék. A fél-felügyelt és erősítéses tanulás két másik technológia, amelyet néha alkalmaznak.

3.1.1 Felügyelt tanulás (Supervised learning)

A gépi tanulás egyik legelterjedtebb típusa, amely során a rendszer előre meghatározott címkézett adatokon tanul. Ebben a módszerben a rendszernek ismert kimeneti értékekkel (címkékkel) rendelkező bemeneti adatokat mutatnak be, és a célja az, hogy olyan modellt hozzon létre, amely képes megjósolni vagy osztályozni a további, még nem látott adatokat. A felügyelt tanulás algoritmusokat a címkézett adatok segítségével tanítják meg. Az algoritmusok ezen adatokon keresztül próbálják megtanulni a bemenetek és a címkék közötti összefüggéseket, és ezáltal képesek lesznek az új bemeneti adatokra vonatkozó jóslásokra vagy osztályozásokra. A felügyelt tanulást széles körben alkalmazzák olyan területeken, ahol a cél a jövőbeli események vagy értékek előrejelzése a meglévő adatok alapján. Példák közé tartozik a képfelismerés, beszédfelismerés vagy akár a pénzügyi elemzés. A felügyelt tanulás során a modellt először tanítják a címkézett adatokon, majd kiértékelik a teljesítményét azáltal, hogy összehasonlítják a modell által előre jelzett értékeket a valós címkékkel. A kiértékelés alapján finom hangolhatják a modellt annak érdekében, hogy jobban teljesítsen.

A felügyelt tanulás az egyik leghasznosabb gépi tanulási típus, és számos alkalmazási területen sikeresen alkalmazzák az előrejelzések, osztályozások és döntéshozatal terén. A megbízható címkézett adatoknak azonban kulcsfontosságú szerepe van a tanulási folyamatban, és az algoritmusok minősége a rendelkezésre álló adatok minőségétől függ.

3.1.2 Felügyelet nélküli tanulás (Unsupervised learning)

Felügyelet nélküli tanulás esetén a gépi tanító rendszernek nincs előzetes ismerete a bemeneti adatok címkéiről vagy kimeneti értékeiről. Ez azt jelenti, hogy nincs felügyelet, vagyis nem állnak rendelkezésre címkézett adatok, amelyek alapján az algoritmus tanulhatna [13]. A cél itt az, hogy az algoritmus maga fedezze fel az adatokban rejlő mintázatokat és struktúrákat. A felügyelet nélküli tanulás rendkívül hasznos olyan esetekben, ahol a cél az adatok közötti természetes struktúra vagy csoportok feltárása. Példák közé tartozik a csoportosítás (clustering), ahol az algoritmus csoportokra osztja az adatokat anélkül, hogy tudná, mi a csoportok tartalma. Ezenkívül használható dimenziócsökkentésre is, amikor az adatok magas dimenziójúak és le szeretnénk egyszerűsíteni őket.

3.1.3 Fél-felügyeleti tanulás (Semi-supervised learning)

A fél-felügyeleti tanulás egy köztes megközelítés a felügyelet nélküli és a teljesen felügyelt tanulás között. Itt van néhány címkézett adat, de nagyobb mennyiségű címkézetlen adat. Az algoritmus először a címkézett adatokon tanul, majd próbálja kiterjeszteni ezt a

tudást a címkézetlen adatokra. A fél-felügyeleti tanulás akkor hasznos, amikor címkézett adatok megszerzése nehéz vagy költséges, például az egészségügyi vagy biológiai adatok esetében, ahol a címkézés szakértelmi tudást igényel.

3.1.4 Megerősítéses tanulás (Reinforcement learning)

A megerősítéses tanulás egy olyan gépi tanulási módszer, amelyben egy „ügynök” vagy döntéshozó interakcióba lép egy környezettel. Az ügynöknek különböző cselekedeteket lehet választani, és a környezet válasza jutalmak és büntetések formájában érkezik. Az ügynök célja a jutalmak maximalizálása, tehát a legjobb döntések megtalálása. A megerősítéses tanulást gyakran alkalmazzák olyan területeken, ahol az ügynöknek tanulnia kell a környezetében való navigáláshoz vagy cselekedetek megtételéhez. Például a robotika területén, ahol egy robotnak tanulnia kell, hogyan kerülje el az akadályokat és érje el a célt.

3.2 Gépi tanulás algoritmusok

3.2.1 Bináris döntési fa

A bináris döntési fa alapú műholdkép feldolgozás a gépi tanulás technikáján alapul. Csomópontokból és ágakból állnak, ahol a csomópontok az attribútumok vagy jellemzők alapján hozott döntéseket, az ágak pedig a döntések lehetséges kimeneteleit ábrázolják. A fák „bináris” jellege azt jelenti, hogy minden belső csomópontnak két ága van, amelyek két lehetséges eredményt jeleznek. A bináris döntési fák felépítése rekurzív „oszd meg és uralkodj” megközelítést követ, és a következő lépéseket foglalja magában:

- **Attribútum kiválasztása:** A leginformatívabb attribútum kiválasztása gyökércsomópontként, gyakran olyan kritériumok alapján, mint az információnyereség, a „Gini-tisztaság” vagy az entrópia.
- **Adatfelosztás:** Az adathalmaz felosztása részhalmazokra a kiválasztott attribútum értékei alapján.
- **Rekurzív felépítés:** A fenti lépések rekurzív alkalmazása minden részhalmazra, amíg egy megállási kritérium, például egy maximális mélység vagy egy levélcso-mópontban lévő adatpontok meghatározott száma nem teljesül.

A bináris döntési fák számos előnyt kínálnak, többek között átláthatóságot, értelmezhetőséget és egyszerű használatot. Kategorikus és numerikus adatok kezelésére egyaránt alkalmasak. Hajlamosak azonban a túlillesztésre, amikor túlságosan összetett modelleket hoznak létre, amelyek jól teljesítenek a képzési adatokon, de rosszul az új, nem látott adatokon. Erre adhat megoldást a „Random Forests”.

3.2.2 Support Vector Machines (SVM)

Az SVM-ek a felügyelt gépi tanulási algoritmusok osztályába tartoznak, amelyek hatalmas népszerűsége tette szert különböző területeken, beleértve a számítógépes látást, a természetes nyelvi feldolgozást és a bioinformatikát. Az SVM-ek különösen alkalmasak osztályozási és regressziós feladatokra, így a gépi tanulás hatékony eszközeinek nevezhetők. Az alap koncepció, hogy olyan hipersíkot találnak, amely a legjobban elválasztja a különböző osztályokhoz tartozó adatpontokat egy nagydimenziós térben [14]. Ezt a hipersíkot nevezzük döntési határnak. Az SVM-ek különösen jól működnek, ha az adatok lineárisan szeparálhatók, ami azt jelenti, hogy egy egyenes (két dimenzióban) vagy egy hipersík (magasabb dimenziókban) tisztán szétválaszthatja az adatokat különböző osztályokra.

A „támogató” vektorok a döntési határhoz legközelebb eső adatpontok. Ezek azok a kritikus adatpontok, amelyek meghatározzák a döntési határ helyzetét és irányát. Az SVM-ek onnan kapták a nevüket, hogy a döntések meghozatalához csak az adatpontok egy részhalmazára, a támogatóvektorokra támaszkodnak, ami számítási szempontból hatékonyra teszi őket. Az SVM-ek egyik fő célja a határérték maximalizálása, amely a döntési határérték és a legközelebbi támogatóvektorok közötti távolság. A nagyobb határérték nagyobb megbízhatóságot jelez az osztályozásban. Ez a margómaximalizálás segít az SVM-eknek abban is, hogy jól általánosíthassanak a nem látott adatokra, csökkentve a túlillesztés kockázatát.

Kétféle esetet különböztetünk meg:

Lineáris SVM: Lineárisan szeparálható adatok esetén a döntési határvonal lineáris egyenletként ábrázolható: $w^T \cdot x + b = 0$, ahol w a súlyvektor és b a torzító kifejezés. A cél az optimális w és b megtalálása, amelyek maximalizálják a margót, miközben biztosítják, hogy minden adatpont helyesen legyen osztályozva. Ez egy optimalizálási problémaként fogalmazható meg, amely a $\|w\|^2$ minimalizálását foglalja magában bizonyos korlátozások mellett.

Nem lineáris SVM: Számos valós környezetben az adatok nem lineárisan szeparálhatók. Az ilyen eseteket úgynevezett „kernel-trükk” technika segítségével tudják kezelni. Ahelyett, hogy az adatokat explicit módon egy magasabb dimenziós térbe képeznék le, egy kernelfüggvény segítségével kiszámítják az adatpontok közötti pontszorzatot ebben az átalakított térben. A népszerű kernelfüggvények közé tartozik a lineáris, a polinomiális, a radiális bázisfüggvény (RBF) és a szigmoid kernel, amelyek lehetővé teszik az SVM-ek számára, hogy összetett döntési határokat ragadjanak meg [15].

3.2.3 K-legközelebbi szomszéd (KNN)

A K-legközelebbi szomszéd, a példány alapú gépi tanulási technikák közül a legegyszerűbbek közé tartozik. Minden mintát/tanító adatot eltárol, és egy új pontot rendel az osztály szerinti K-legközelebbi szomszédjához az adott adathalmazban [16]. Az adatok közti távolság kiszámítására, vagy meghatározására általában euklideszi távolságot alkalmazunk. Az algoritmus képes arra, hogy meghatározza a szükséges függvény lokális közelítését, ami minden új osztályzásra került adatpont esetében eltérő lesz. Ez főleg akkor hasznos, ha az adott függvény összetett, de több kevésbé bonyolult függvény segítségével is felírható. A KNN alapvetően a hasonlóság elve alapján működik. Feltételezi, hogy a hasonló jellemzőkkel rendelkező adatpontok általában ugyanabba az osztályba tartoznak, vagy hasonló viselkedést mutatnak. Amikor egy új adatpont osztályozása a feladat, a KNN algoritmus kiszámítja a távolságot a pont és a képzési adathalmazban található k legközelebbi szomszédja között. A távolságok kiszámítása után az algoritmus kiválasztja a legkisebb távolsággal rendelkező „k” adatpontot, és hozzárendeli az osztálycímke (osztályozás esetén), vagy kiszámítja a „k” szomszédok átlagértékét (regresszió esetén). A legnagyobb számmal vagy a legközelebbi átlaggal rendelkező osztálycímke vagy regressziós érték lesz az új adatpont előrejelzése.

A k értéke a KNN-ben egy kritikus hiperparaméter, amely jelentősen befolyásolhatja az algoritmus teljesítményét. Egy kisebb k érték, például 1, a modellt nagyon érzékenyvé teszi az adatokban lévő zajra, ami potenciálisan túlilleszkedést eredményezhet. Ezzel szemben a nagyobb „k” érték, például 10 vagy 20, stabilabb előrejelzést biztosít, de alul illeszkedéshez vezethet. Ezért a megfelelő „k” érték kiválasztása gyakran olyan technikák segítségével történik, mint a „cross-validation” (keresztellenőrzés), hogy megtaláljuk a megfelelő egyensúlyt a torzítás és a szórás között.

Összefoglalva, a k-közelebbi szomszédok algoritmus, bár alapvetően egyszerű, a minapig használatos a gépi tanulás területén. Az adatpontok hasonlóságán alapuló előrejelzések készítésére való képessége miatt az alkalmazások széles körében alkalmazzák, az osztályozástól és a regressziótól kezdve az ajánlórendszerekig és az anomália-felismerésig. Bár a KNN-nek megvannak a maga korlátjai, rugalmassága és könnyű használhatósága egy jól és széleskörűen használható eszközzé teszi.

3.2.4 Zsákolás

A zsákolás vagy bagging a legegyszerűbb és legkorábbi technikák egyike, és a legalkalmasabb a kis képzési adatkészletekkel kapcsolatos problémák megoldására. Ez a módszer a „bootstrap” mintavételezési módszert alkalmazza az adatok véletlenszerű részhalmazainak meghatározására az eredeti modellkészlet cseréjével történő betanításához. A zsáko-

lás a gép pontosságának javítására használható, emellett pedig használják még regresszióban és osztályozásban is. Ez a megközelítés döntési fák használatával segíti a variancia levonását és a modellek robusztusságának fokozását ezért kiválóan alkalmas az eredeti gyenge modellekhez, amelyek nagy szórást mutatnak [17].

A zsákolás legfontosabb alapelvei:

- **Bootstrap mintavételezés:** A zsákolás az eredeti adathalmazból történő véletlenszerű mintavételezéssel kezdődik. Ez a folyamat több, az eredeti adatokkal nagyjából azonos méretű, de bizonyos fokú véletlenszerűséggel rendelkező gyakorlóhalmazt hoz létre. Egyes példányok minden egyes zsákban megismétlődhetnek, míg mások elhagyhatók.
- **Párhuzamos képzés:** A bagging lényege az alapmodellek párhuzamos képzése. Minden egyes részmintát egy különálló alapmodell képzésére használunk, egymástól függetlenül. Ezek az alapmodellek bármilyen típusúak lehetnek, de sokoldalúságuk és a túlillesztéssel szembeni ellenállásuk miatt általában döntési fákat alkalmaznak.
- **Összesítés:** Az alapmodellek képzése után a zsákolás egyesíti az egyes előrejelzéseiket, hogy egy végső előrejelzést készítsen. Az aggregálási módszer a probléma típusától függően változik; regressziós problémák esetén az előrejelzéseket gyakran átlagolják, míg osztályozás esetén többségi szavazást vagy súlyozott szavazást alkalmazhatnak.

A zsákolás előnyei:

- **Csökkentett eltérés:** A zsákolás hatékonyan csökkenti a modell varianciáját több alapmodell előrejelzéseinek átlagolásával. Ez a varianciacsökkentés stabilabb és megbízhatóbb előrejelzéseket eredményez.
- **Javított általánosítás:** Azáltal, hogy az alapmodelleket az adatok különböző részhalmazain képzik ki, a zsákolás segít a túlillesztés mérséklésében. Az „ensemble”-modell kisebb valószínűséggel jegyzi meg a képzési adatokban lévő zajt, és jobban képes általánosítani a nem látott példákra.
- **Megnövelt pontosság:** Sok esetben a zsákolás jelentős javulást eredményezhet az előrejelzési pontosságban, egyetlen alapmodellhez képest. Ez teszi hasznos és jól használható eszközzé különböző feladatok körében.

3.2.5 Boosting

A boosting technika célja, hogy minél pontosabb és emellett minél jobb teljesítményű algoritmust kapjunk. Az alap ötlet azon alapul, hogy szekvenciálisan adjuk hozzá az újabb modelleket. Ebben a csoportban a gyengébb modellek erősödnek/erősebbé válnak (Ezek általában a kiinduló modellek). Emiatt is nevezik „Boosting”-nak azaz erősítés/fokozás. A fokozást a döntési fák túlillesztésének megakadályozására alkalmazzák. Ennek köszönhetően csökken a szórás és a torzítás is. Az algoritmus többféle tanító készletből véletlenszerűen választ, majd arra cseréli az elsődleges tanító készletet. Modellek sorozatát (szekvencia) így tanítjuk fel. Először jön az első modell a betanítási adatok alapján, majd egy új modell jön létre az előző modell teljesítményének növelésére a megfelelő módosításokkal. Végül a megmaradt gyenge tanulókat súlyozással kombináljuk.

A Boosting alapelvei:

- **Szekvenciális tanulás:** A Boosting algoritmusok szekvenciálisan működnek, ahol minden egyes következő modell az előzőek által elkövetett hibákra összpontosít. Ezáltal adaptívan korrigálja a rosszul besorolt példányokat.
- **Súlyozott mintavételezés:** A képzés során a boosting minden egyes példányhoz súlyokat rendel, a rosszul besorolt példányok nagyobb súlyt kapnak. Ez előnyben részesíti a nehéz minták képzését.
- **Modell-aggregáció:** A gyenge tanulók kombinációja erős tanulót alkot. Ez az aggregációs folyamat gyakran súlyokat rendel az egyes modellekhez azok teljesítménye alapján.

3.2.6 AdaBoost

Az AdaBoost avagy Adaptive Boosting az előbb említett algoritmushoz hasonlóan működik. Az algoritmus a gyenge tanulókat úgy fejleszti, hogy legyőzi a gyengeségeit és egy iteratív megoldással kijavítja a gyenge tanulók hibáit. Ezeket a tanulókat a rendszer itt is szekvenciálisan adja hozzá és tanítja egy súlyozott tanítókészlet segítségével. Ezt követően a gyenge tanulókat kombinálja, hogy abból egy erősebbet hozzon létre.

Az AdaBoost előnyei:

- **Nagy pontosság:** Az AdaBoost arról ismert, hogy képes jelentősen javítani a gyenge tanulók pontosságát, gyakran felülmúlva más „ensemble” módszereket.
- **Robusztusság:** Kevésbé hajlamos a túlillesztésre, mivel a hibák kijavítására összpontosít, nem pedig a teljes képzési adat illesztésére.

- **Sokoldalúság:** Az AdaBoost a gépi tanulási feladatok széles körére alkalmazható, beleértve az osztályozást és a regressziót is.
- **Jellemzők kiválasztása:** Az AdaBoost implicit módon képes a jellemzők kiválasztására azáltal, hogy az informatív jellemzőknek nagyobb jelentőséget tulajdonít a képzési folyamat során.

3.2.7 LightGBM

A LightGBM egy gradiens növelő módszer (GB -> Gradient Boosting), amely döntési fákra alapul. Más döntési fa alapú módszerekhez hasonlóan a LightGBM is használható osztályozásra és regresszióra. Az algoritmus nagy teljesítményre optimalizált, akár elosztott rendszerekben is alkalmazható. Döntési fákat hoz létre, amelyeken levél elemek „nőnek”, ami azt takarja, hogy adott feltételek mellett csak egyetlen levél oszlik két ágra, annak erősségétől függően. A levél elemek miatt hajlamos a túltanulásra (overfitting) különösen kisebb adathalmaz esetén. Emiatt érdemes a fák mélységének korlátozása [18]. A LightGBM másik jellemzője, hogy olyan funkciókat kínál, amely segítségével az algoritmus, az úgynevezett „feature” értékeket kombinálja a dimenziók csökkentése érdekében, így gyorsabb és hatékonyabb a tanulási folyamat. Fontos megjegyezni hogy erre akkor van lehetőség, ha vannak olyan értékek, melyek a tanítás szempontjából elhanyagolhatóak.

A LightGBM fő jellemzői:

- **„Fokozatos” növelés (Gradient Boosting):** A többi boosting algoritmushoz hasonlóan a LightGBM is több gyenge tanuló kombinálásával épít erős tanulót, és minden egyes iterációnál a veszteségfüggvény gradiensének csökkentésére helyezi a hangsúlyt.
- **Hisztogram alapú tanulás:** A LightGBM egyedülálló hisztogram-alapú tanulási stratégiát alkalmaz, amely az adatokat diszkrét hisztogramokba rendezi, hogy csökkentse a memóriahasználatot és javítsa a képzés sebességét.
- **Gradiens alapú egyoldali mintavételezés (GOSS):** A GOSS a LightGBM-ben alkalmazott technika, amely a hatékonyság javítására szolgál azáltal, hogy csak a nagy gradienssekkel rendelkező adatpéldányok egy részhalmazát választja ki, így a képzés a leginformatívabb mintákra összpontosít.
- **Exkluzív jellemző-összekapcsolás (EFB):** Egy másik optimalizálás a LightGBM-ben, amely az exkluzív jellemzőket egybecsomagolja, csökkentve ezzel a memóriahasználatot és tovább növelve a sebességet.

A LightGBM működési alapelvei:

- **Levél alapú növekedés:** A LightGBM a hagyományos mélység szerinti növekedéstől eltérően, levél szerinti növekedést alkalmaz. A fát a legnagyobb veszteségcsökkentéssel rendelkező levél felosztásával növeszti, ami gyakran hatékonyabb és pontosabb modellekhez vezet.
- **Gradiens erősítés:** Minden egyes iterációban egy differenciálható veszteségfüggvény negatív gradiensét minimalizálja. Ez kevesebb iterációval jobb pontosságot eredményez.
- **Kategorikus jellemzők támogatása:** Natívan képes kezelni a kategorikus jellemzőket anélkül, hogy szükség lenne az egyponthoz való kódolásra, csökkentve ezzel az adathalmaz dimenzionalitását.

3.2.8 FastTree

A FastTree szintén egy gradiensnövelő algoritmus, ez egy gépi tanulási technika, amelyet regressziós és osztályozási feladatokra egyaránt használnak. Úgy működik, hogy több gyenge tanulót, jellemzően döntési fákat, erős „ensemble” modellé kombinál. A FastTree a gradiens boosting egy speciális implementációja, amely nagy teljesítmény elérésére összpontosít mind a képzési sebesség, mind az előrejelzési pontosság tekintetében [19] [20].

A FastTree fő jellemzői:

- **Döntési fák:** A FastTree elsősorban döntési fákat használ alaptanulóként. A döntési fák sokoldalú modellek, amelyek képesek az adatokon belüli összetett összefüggések megragadására, így a problémák széles körére alkalmasak. A mély és összetett fák építése azonban túlillesztéshez vezethet.
- **„Fokozatos” növelés (Gradient Boosting):** A gradiens boosting lényege, hogy a gyenge tanulók iteratív módon illeszkednek az előző tanulók hibáihoz. A FastTree ezt a megközelítést alkalmazza, és ezeket a gyenge modelleket egy erősebb modellé kombinálja.
- **Sebesség optimalizálás:** Mint a neve is mutatja, a FastTree a sebesség szempontjából kiemelkedik. Számos olyan optimalizációt valósít meg, amelyek jelentősen csökkentik a képzési időt, például hatékony adattárolást, mintavételi stratégiákat és párhuzamosítást.
- **Regularizáció:** A FastTree a túlillesztés elkerülése érdekében olyan regularizációs technikákat alkalmaz, mint a zsugorítás (tanulási sebesség) és az oszlopok almin-tavételezése. Ezek biztosítják, hogy a modell ne jegyezze meg a képzési adatokat, hanem olyan robusztus mintákat tanuljon, amelyek jól általánosíthatók a nem látott

adatokra.

- **Hiperparaméter-hangolás:** A többi gépi tanulási algoritmushoz hasonlóan a FastTree is gondos hiperparaméter-hangolást igényel, hogy optimalizálni lehessen a teljesítményét az adott feladathoz. A legfontosabb hiperparaméterek közé tartozik a tanulási sebesség, a fák száma, a fa mélysége és az almintavételi arány [19].

3.3 Mély gépi tanulás

A mélytanulás, avagy angolul deeplearning a mesterséges neurális hálózatok (ANN) és a kapcsolódó gépi tanulás algoritmusok egy csoportja, amelyek egynél több rejtett réteget tartalmaznak. A gépi tanulás családjába tartozik annak egy további ága. A következő tulajdonságokkal rendelkeznek:

- A rétegek nemlineáris feldolgozó egységeket használnak a jellemzők kinyerésére és átalakítására. Minden egyes egymást követő réteg az előző réteg kimenetét használja bemenetként. Az algoritmusok lehetnek felügyelték vagy felügyelet nélküliek, és a feladatuk közé tartozik a mintaelemzés és az osztályozás.
- Az adatok többszintű jellemzőinek vagy reprezentációinak tanulásán alapulnak. A magasabb szintű jellemzőket alacsonyabb szintű jellemzőkből származtatják, hogy hierarchikus reprezentációt alkossanak.

3.3.1 Mesterséges neurális hálózatok

A neurális hálózatok (Artificial Neural Networks, ANN) hasonlóan működnek, mint azt emberi tanulási folyamatok. Az ANN típusú osztályozás előnye, hogy könnyen beépíthetők a kiegészítő adatok a „felismerési” fázisba ezzel javítva az osztályozás pontosságát. Egyszerű feldolgozó egységekből állnak, amelyek szimulálják a biológiai neuronokat. Ezek a mesterséges neuronok, más néven csomópontok, egy vagy több rétegben helyezkednek el. Minden csomópont egy vagy több csomópontozatához kapcsolódik súlyozott kapcsolatokon keresztül, amelyek szimulálják a biológiai szinapszisokat [16]. Az ANN-architektúra egy bemeneti rétegből, egy vagy több rejtett rétegből és egy kimeneti rétegből áll. A bemeneti réteg a kezdeti adatok vagy jellemzők belépési pontjaként szolgál, míg a kimeneti réteg a végső előrejelzéseket vagy eredményeket állítja elő. A kettő között elhelyezkedő rejtett rétegek közbenső számításokat végeznek, és felelősek az adatokon belüli összetett minták megfejtéséért. Az ANN-ek működésében kulcsfontosságúak a neuronok közötti kapcsolatok és a hozzájuk tartozó súlyok. Minden egyes kapcsolat, az agyban lévő szinapszishoz hasonlóan, súlyokkal van ellátva. Ezek a súlyok kulcsfontosságúak, mivel ezek határozzák meg, hogy az egyik neuron kimenete milyen erősen befolyásolja a másik neuront. A tanítás során, ezeket a súlyokat úgy igazítják ki, hogy minimalizálják a

hálózat előrejelzései és a tényleges célértékek közötti eltérést.

Az ANN-ek tanulási folyamata az optimalizációs algoritmusokon, nevezetesen a gradiens süllyedésen és a „backpropagáción” alapul. Ezek a mechanizmusok megkönnyítik a súlyok iteratív finomítását, fokozatosan összehangolva a modell előrejelzéseit a valós megfigyelésekkel. A hatékony tanítás azonban megköveteli, hogy a hálózat hozzáférjen a címkézett adathalmazokhoz, ahol a hálózat összehasonlíthatja előrejelzéseit az alapadatokkal, és elvégezheti a szükséges korrigációkat.

A gyakorlati alkalmazásuk széles spektrumot ölel fel. A gépi látás területén megkönnyítik a képosztályozást és a különböző tárgyak felismerését. A szövegosztályozásban és a gépi fordításban jeleskednek. Az ANN-ek emellett fontos szerepet játszanak a beszédfelismerésben, az ajánlórendszerekben és az autonóm robotikában. Mindemellett pedig ma már az orvostudomány különböző területein is használják.

Azonban, mint ahogyan az életben sok más, az ANN-ek sem tökéletesek. A mély neurális hálózatok képzése számításigényes lehet, ami jelentős számítási erőforrásokat és kiterjedt címkézett adathalmazokat igényel. A túlillesztés/túltanulás (overfitting), amikor a modell a képzési adatokon kivételesen jól, a soha nem látott adatokon pedig rosszul teljesít, továbbra is gyakori probléma. Ennek a problémának az enyhítésére olyan technikákat alkalmaznak, mint a regularizáció. A mély hálózatok értelmezhetősége, amelyet gyakran „fekete doboz” problémaként jellemeznek, továbbra is aktív kutatási terület, mivel a belső működésük megértése elengedhetetlen az érzékeny területeken történő alkalmazásokhoz.

3.3.2 Konvolúciós neurális hálózatok

A konvolúciós neurális hálózatok (Convolutional Neural Networks, CNN), a mesterséges intelligencia területén kulcsfontosságú áttörést jelentenek, kifejezetten a vizuális adatok, köztük képek és videók feldolgozására és értelmezésére. Megjelenésük átforgalmazta a gépi látás területét, lehetővé téve, hogy példátlan pontossággal és hatékonysággal lehessen megoldani számos feladatot, például a képek osztályozását, a tárgyak felismerését és a képek szegmentálását [21]. A CNN-ek a vizuális adatok által támasztott egyedi kihívásokra adott válaszként jelentek meg. A képekben a pixelek térbeli elrendezése és a helyi minták jelentősége kiemelkedő fontosságú. A hagyományos neurális hálózatok korlátokba ütköztek a képek kezelésében a pixeladatok nagy dimenzióssága és a térbeli információk elvesztése miatt.

A CNN-ek meghatározó jellemzője a konvolúciós rétegekben rejlik, amelyek a tervezésük lényegét alkotják. Ezek a rétegek kis szűrőket (kerneleket) alkalmaznak, amelyek a bemeneti képen áthaladva vagy konvolúciót alkalmazva lokalizált jellemzőket vonnak ki. A szűrők a bejárás során elemeként szorzásokat végeznek a bemenettel, és olyan „jellem-

zótérképeket” hoznak létre, amelyek a megtanult jellemzőket, például éleket, textúrákat vagy bonyolult mintákat tartalmaznak. A bemenet különböző aspektusainak megragadására több szűrőt alkalmaznak, a szűrő paramétereit pedig a tanítás során tanulják meg. Sokszor pooling rétegeket is alkalmaznak, hogy a jellemzőtérképek térbeli dimenzióit csökkentsék, ami csökkenti a számítási komplexitást.

A CNN-ek képesek önállóan hierarchikus jellemzőreprezentációkat létrehozni. A korai rétegek olyan kezdetleges jellemzőket azonosítanak, mint az élek és a sarkok, míg a mélyebb rétegek ezeket a primitív jellemzőket integrálják a bonyolult objektumok felismeréséhez. Ez a hierarchikus megközelítés lehetővé teszi a CNN-ek számára, hogy az adatokon belül mind az alacsony szintű, mind a magas szintű jellemzőket ügyesen rögzítsék.

Összegezve, a konvolúciós neurális hálózatok hatalmas előrelépést jelentenek a számítógépes látás területén, mivel figyelemre méltó képességgel rendelkeznek a jellemzők automatikus kinyerésében és hierarchikus reprezentációk létrehozásában a különböző vizuális adatokból. Egyedülálló "képességük", hogy mind a helyi, mind a globális mintákat felismerjék, a kép- és videóelemzés számos alkalmazási területén nélkülözhetetlen eszközzé tette őket.

3.3.3 Rekurrens neurális hálózat

A rekurrens neurális hálózat (Recurrent Neural Network, RNN) alapvetően egy olyan mesterséges neurális hálózat, amelyet adatsorok feldolgozására terveztek [21]. A hagyományos előre csatolt neurális hálózatokkal ellentétben, amelyek rögzített méretű bemeneti vektorokkal működnek, az RNN-ek rendelkeznek egy olyan beépített memóriamechanizmussal, amely lehetővé teszi számukra, hogy egy rejtett állapotvektort tartsanak fenn, amely képes az előző lépésekből származó információkat rögzíteni. Ez a memóriakapacitás teszi az RNN-eket kivételesen alkalmassá a szekvenciális (egymást szorosan követő) adatokat tartalmazó feladatokra, például a természetes nyelvi feldolgozásra, a beszédfelismerésre vagy akár a különböző előrejelzésekre.

Az RNN legfontosabb összetevői:

- **Rejtett állapot:** A rejtett állapot, amelyet gyakran „h”-val jelölnek (az Anagol hidden szóból eredően), a hálózat memóriájaként szolgál. Minden egyes lépéskor frissül, és a sorozat korábbi lépéseinek információit őrzi.
- **Bemeneti szekvencia:** Az RNN-ek a bemeneti szekvenciákat egyesével veszik fel, például egy mondatot vagy egy idősorozatot. Minden egyes bemeneti elemet szekvenciálisan dolgoznak fel, lehetővé téve a hálózat számára, hogy figyelembe vegye az előző elemek kontextusát.

- **Súlymátrixok:** Más neurális hálózatokhoz hasonlóan az RNN-ek is rendelkeznek súlyokkal (weight) és torzításokkal (bias), amelyeket a tanítás során tanulnak meg. Ezek a súlyok határozzák meg, hogy a bemenet miként alakul át az egyes lépéseket követően.

A hagyományos RNN-ek egyik jelentős kihívása az „eltűnő” és „robbanó” gradiens problémája. Ahogy a szekvenciák hosszabbak lesznek, a gradiens túl kicsivé (eltűnés) zsugorodhatnak vagy túl nagyra (robbanás) nőhetnek a „backpropagáció” során, ami kihívást jelent a hálózat számára a hosszú távú függőségek megtanulása szempontjából. E probléma megoldására számos RNN-változatot fejlesztettek ki, például a LSTM-hálózatokat (Long Short-Term Memory) és a GRU-hálózatokat (Gated Recurrent Unit), amelyek speciális úgynevezett „kapuzási” (gating) mechanizmusokat alkalmaznak, hogy jobban szabályozzák az információ áramlását a hálózaton keresztül.

Összegezve rekurrens neurális hálózatok a neurális hálózatok egy hatékony és rugalmas osztályává fejlődtek, lehetővé téve a szekvenciális adatok modellezését különböző területeken. Az időbeli függőségek megragadásának képessége teszi őket a gépi tanulás arzenáljának egyik létfontosságú eszközévé. A kutatás további fejlődésével az RNN-ek valószínűleg továbbra is a mesterséges intelligencia innovációinak élvonalában maradnak, és többek között a természetes nyelvi felismerést, a beszédfelismerés és az autonóm rendszerek területén is előrelépést jelentenek.

4 SZÍNMODELLEK

A színmodell a színek mérésére szolgáló rendszer, amely képes emberi szem által érzékelhető színek kombinálására. A színmodellt úgy definiálhatjuk, mint a digitális ábrázolása a lehetséges színeknek. Jellemzően a színmodellek három vagy négy színkomponenssel rendelkeznek (RGB-> Red,Green,Blue). Különböző alkalmazásokhoz különböző szinterek állnak rendelkezésre. A színmodellek három kategóriába sorolhatók a képfeldolgozási alkalmazásuk szerint:

Eszközorientált színmodellek: Ezeket a modelleket széles körben használják számos olyan alkalmazásban, amelyek megkövetelik, hogy a szín összhangban legyen a használt hardvereszközökkel, például az emberi látás érzékeléséhez használt eszközökkel [22]. Ezek a modellek segítenek a színrendszerek konzisztens és pontos kezelésében különböző eszközökön és környezetekben. Ilyen például az RGB vagy a CMYK színmodell.

Felhasználóorientált színmodellek: Ezek a modellek lehetővé teszik a felhasználó számára, hogy leírja és megközelítse azt, amit a szín megjelenítéséből érzékel, mondhatjuk úgy is hogy a felhasználók igényeire és preferenciáira épülnek. A modellek az emberi színlátás és színmegértés alapján lettek kialakítva, és gyakran a személyre szabott színek kezelését és alkalmazását célozzák meg. Például a különböző paletták és színsémák.

Eszközfüggetlen színmodellek: Ezt a színmodellt nem befolyásolják az adott készüléktulajdonságok, és a paraméterekből ugyanaz a szín fog megjelenni, az eszközök teljes figyelembevétel nélkül. Ezek a modellek általában olyan absztrakt matematikai modellként működnek, amelyek segítenek a színek univerzális reprezentációjában és kezelésében különböző eszközökön és környezetekben. Ide sorolhatjuk az XYZ, CIE, vagy akár a LAB színcsoportokat is.

4.1 RGB Színmodell

Az RGB-modell színei három színnel írható le (R, G és B). Az RGB színtér egy háromdimenziós egységkockának tekinthető, amelynek minden tengelye az egyik alapszín jelképezi, színei a kocka koordinátái által meghatározott pontok a kockán belül. Az elsődleges színek tehát a következők: vörös, zöld és kék. Az RGB másodlagos színei a cián, a magenta és a sárga. Az RGB színrendszer jellege additív abban az értelemben, hogy a színek hozzáadása a képet világosabbá teszi. A fekete az origónál, a fehér pedig a sarokban van, ahol $R = G = B = 255$ [23].

4.2 HSV Színmodell

A HSV (vagy más néven HSB) -t, gyakran használják számítógépes grafikában, mert sokszor könnyebb egy színről árnyalat és telítettség szempontjából gondolkodni, mint additív vagy szubtraktív színtelepek szempontjából. Ez a modell közelebb áll az emberek színérzékeléséhez, mint az RGB. A festészetben például az árnyalat, a telítettség és az értékek a szín, az árnyékolás és a tónus kifejezése. A HSV-modelltér egy fordított hatszögletű piramisként írható le.

Hue (Színárnyalat): A színárnyalat azt mutatja meg, hogy egy adott szín milyen színű, és azt jelenti, hogy melyik színcsaládba tartozik. A színárnyalatot általában fokokban mérik, és egy kör alakú skálán helyezik el. Az HSV modellben a színárnyalat 0 és 360 fok közötti értéket vehet fel, ahol a 0 fok a vörös, a 120 fok a zöld, a 240 fok a kék színt reprezentálja.

Saturation (Telítettség): A telítettség azt mutatja meg, mennyire tiszta vagy élénk a szín. Minél magasabb a telítettség, annál élénkebb a szín, míg a telítetlenség esetén a szín halványabb lesz. A telítettség értéke a 0 és 100 között változhat, ahol a 0 egy teljesen szürke (telítetlen) színt jelent, míg a 100 egy teljesen tiszta és élénk színt.

Value (Érték): Az érték azt mutatja meg, mennyire világos vagy sötét a szín. Az érték skála 0 és 100 között mozog, ahol a 0 fekete színt jelent, míg a 100 fehér színt jelent.

4.3 CIE Lab

Más néven CIELAB színmodell a második egységes színtér, amelyet 1976-ban definiáltak, a CIE XYZ térből származik. Az Lab színmodell a színt a 3D színtérben elfoglalt helyétől függően határozza meg.

L (világosság): Ez a komponens a szín világosságát vagy sötétségét fejezi ki egy skálán, ahol a 0 a teljes sötétséget, a 100 pedig a teljes világosságot jelenti. A L^* értékek közötti távolság azt mutatja meg, mennyire eltérőek a színek világossága.

a (zöld-piros tengely): Az a^* komponens a zöld és piros színek közötti helyzetet mutatja meg a színtérben. Pozitív értékek pirosat, negatív értékek pedig zöldet jelentenek.

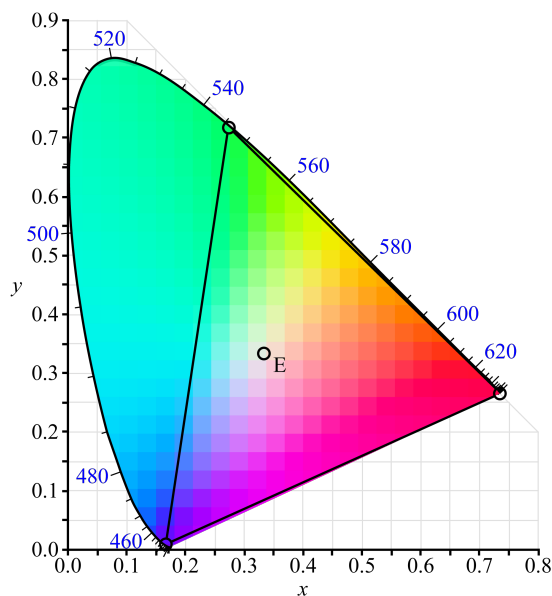
b (kék-sárga tengely): A b^* komponens a kék és sárga színek közötti helyzetet mutatja meg a színtérben. Pozitív értékek késsel, negatív értékek pedig sárgával vannak kapcsolatban. eszközfüggetlen, és nagyon fontosnak tekinthető az asztali színek esetében.

4.4 CIE Luv

Az első egységes színtér a CIE XYZ térből származtatott színtér fehér referenciaponttal, ahol a fehér referenciapont (X_n, Y_n, Z_n) a lineáris RGB (1, 1, 1) értékek. Az L^* a világosságot, az u^* és v^* pedig a színárnyalat és a színárnyalat korreláltumait jelöli [22]. Ennek a modellnek az előnye, hogy alkalmas a színek szimulációjára különböző megvilágítási körülmények között, és hasznos az olyan alkalmazásokban, ahol a színviszonyok vagy a színmegjelenítés változhat. A CIE Luv színmodell sokszor hasznos eszköz a színek precíz elemzéséhez és reprodukciójához olyan területeken, mint a színrendszerek közötti konverziók, illetve a színek adaptációja különböző környezetekhez vagy a színek értelmezése és vizsgálata a tudományos és ipari alkalmazásokban.

4.5 CIE XYZ Színmodell

Az egyik elsőként meghatározott színtér a CIE XYZ, más néven X, Y és Z tristimulus függvények, vagy akár a CIE színtérnek is szokás nevezni. Az XYZ-t a „International Commission on Illumination” (CIE) hozta létre 1931-ben. Ezeket a modelleket kézzel hozzák létre a vizualizáció és a megjelenések illesztésének emberi ítélőképessége segítségével, és a kiválasztott színtér ezen az illesztési eljárásen alapul. Az érzékenységi görbék X, Y és Z paramétereinek alakját bizonyos plauzibilis pontossággal mérik. Matematikailag a modell leírható Y fénysűrűségkomponenssel, valamint két X és Z színikoordinátával együtt, azonban néha az XYZ színmodellt az Y fénysűrűségparaméterével ábrázolják.



4.1. ábra: A CIE RGB alapszínek színskálája és az alapszínek elhelyezkedése a CIE 1931 xy színskáladiagramon.

4.6 Szürkeárnyaltos színmodell

A szürkeárnyaltos színtér a képfeldolgozás nélkülözhetetlen színtere, amelyet széles körben használnak a képelemzés és -manipuláció különböző aspektusainak egyszerűsítésére és javítására. Az RGB színtérrel ellentétben, amely a színeket vörös, zöld és kék csatornák kombinációjaként ábrázolja, a szürkeárnyaltos képek egyetlen csatornából állnak, amely a fényerő vagy intenzitás különböző szintjeit reprezentálja.

Szürkeárnyaltos színtér jellemzői:

- **Egyszerűség:** A szürkeárnyaltos képekkel egyszerűbb dolgozni, mint színes társakkal. Csak egy csatornát tartalmaznak, így könnyebben feldolgozhatók, tárolhatók és továbbíthatók. Ez az egyszerűség különösen értékes olyan helyzetekben, amikor az erőforrás-korlátozás vagy a feldolgozási sebesség fontos szempont.
- **Intenzitás információk:** A képeken az egyes képpontok értéke közvetlenül az intenzitást jelenti, a feketétől (0) a fehérig (255 egy 8 bites képen). Ez az egyszerűség hasznos az olyan alkalmazásokban, amelyek az intenzitásinformáció kinyerését vagy elemzését igénylik, mint például az éldetektálás, a kontrasztjavítás és a hisztogramelemzés.
- **Kisebbségi adatmérték:** A szürkeárnyaltos képek kisebb méretűek az RGB-képekhez képest, ami előnyös lehet, ha nagy adathalmazokkal vagy korlátozott tárolókapacitással dolgozunk. Az adatmérték csökkentése elősegíti a gyorsabb feldolgozást és a hatékony memóriahasználatot [24].

4.6.1 Konverziók

Magasságtérképek esetében fontos, hogy a beolvasott kép mindenképp szürkeárnyaltos színtérben legyen. Emiatt szükséges lesz színterek közötti konverzióra. Általában RGB csatornás fekete-fehér képek a „hibás” magasságtérképek, így az alábbi alfejezetben RGB és szürkeárnyaltos csatornák közötti konverziókat fogjuk elemezni.

Átlagolási módszer: Az első és egyben az egyik legegyszerűbb ilyen metódus az átlagolási módszer, mely egy képpont szürkeárnyaltos értékét a színcsatornák egyszerű átlagaként számítja ki. Ennél a módszernél minden színcsatorna (R, G, B) egyenlő mértékben járul hozzá a szürkeárnyalt-értékhez. Íme az átlagolási módszer képlete:

$$\text{Average} = \frac{R + G + B}{3} \quad (4.1)$$

Ez a módszer egyszerű és számítási szempontból hatékony. A színcsatornák kiegyensúlyozott ábrázolását biztosítja. Az átlagolási módszer nem feltétlenül tükrözi pontosan az érzékelt fényerőt, mivel minden színcsatornának azonos súlyt ad, ami nem biztos, hogy megfelel az emberi szem érzékelésnek. Emellett pedig a végeredmény zajossá válhat.

Világosság módszer: A „világosság” módszer a képpont szürkeárnyaltos értékét a maximális és minimális színcsatornaértékek átlagaként számítja ki. Ezt a módszert úgy tervezték, hogy megőrizze az eredeti színes kép észlelt fényerejét vagy fényességét. Íme a világossági módszer képlete:

$$\text{Lightness} = \frac{\max(R, G, B) + \min(R, G, B)}{2} \quad (4.2)$$

Ez a módszer általában megőrzi az eredeti kép általános kontrasztját és fényerejét, így vizuálisan kellemesebbé teszi azt. Előfordulhat, hogy nem tükrözi pontosan a valódi fényerőt, mivel a maximális és minimális csatornaértékektől függ, amelyeket a színtelítettség befolyásolhat.

Fényerősség módszer: A fényerősségi módszer úgy számítja ki egy képpont szürke árnyalat, hogy a színcsatornákra súlyozott együtthatókat alkalmaz, ami az emberi szem vizuális érzékeléséhez áll közel. A fényerősségi módszer képlete a következő:

$$\text{Luminosity} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \quad (4.3)$$

A fényerősségi módszer figyelembe veszi az emberi színérzékelést, és nagyobb súlyt ad a zöld csatornának, amely érzékenyebb a fényerősség változásaira. Ez olyan szürkeárnyaltos képet eredményez, amely gyakran nagyon közel áll az emberi szem által érzékelt fényerősséghez. Míg a fényerősségi módszer jobban megőrzi az érzékelt fényerőt, előfordulhat, hogy bizonyos színeket nem pontosan ábrázol. Például az erősen telített színek kevésbé élénknek tűnhetnek a szürkeárnyaltos képen.

Összefoglalva, az RGB képek szürkeárnyaltossá alakításának módszerét az adott alkalmazás és a kívánt eredmény függvénye. A világossági módszer előnyben részesíthető, ha fontos az általános kontraszt megőrzése, az átlagolási módszer egyszerű és hatékony, a fényerősségi módszert pedig gyakran választják, ha az érzékelt fényerősség pontos megjelenítése az elsődleges szempont. Mindegyik módszernek megvannak az erősségei és gyengeségei, és a választásnak összhangban kell lennie az adott képfeldolgozási feladat céljaival.

5 ZAJSZŰRÉS ÉS EGYÉB JAVÍTÁSOK

5.1 Simítás

Magasság térképek esetén előfordulhat, hogy a felhasználni kívánt kép nem elég zajmentes. Ebből következhet olyan probléma, hogy a legenerált felületek nem simák, hanem élesek, csúcsosak lesznek. Ezt a hibát simítási algoritmussal lehet kiküszöbölni.

5.1.1 Beltrami szűrő

Az algoritmus a szűrést a ponthalmaz minden pontja körül kiválaszt egy egységet, kiszámítja az áramlás léptetési idejét az adott egységen majd ennek megfelelően állítja be az aktuális pontot [25].

Algorithm: Beltrami filter

```
1: for iterations  $t = 1, 2, \dots, T$  do
2:   for  $i = 1, 2, \dots, N$  do
3:     Compute point neighborhood  $N_i$ .
4:     Estimate normal direction  $n_i$ .
5:     Use the projection points in  $N_i$  in the normal direction to obtain a local parametric surface representation  $I_i$ .
6:     Smooth the neighborhood's representation using the patch-space Beltrami flow.
7:     Update point  $i$  according to its change in  $I_i$  flow in the normal direction.
8:   end for
9: end for
```

Közvetlen szomszédság definiálása: konfiguráláskor nem áll rendelkezésre minden szomszédsági viszony. Ezért P_i pont N_i -vel való szomszédságát úgy definiáljuk, mint a legközelebbi K szomszédok halmazát, melyek R távolságon belül helyezkednek el. Itt a távolságot Euklideszi térben mérjük. N_i -t egy segédnek használja az algoritmus, amivel egy lokális keretet tud definiálni P_i köré. R értékének megválasztásakor figyelni kell, hogy ne legyen túl magas, illetve túl alacsony érték sem.

Esetleges problémák elkerülése végett bevezetésre kerül egy K_{min} azaz minimális számú legközelebbi szomszéd. Ezt követően megtörténik a bejárás majd egy becslés minden legközelebbi pontból minden szomszédos pontba. (K értékét R -hez képest állítjuk be, illetve K -t fentről korlátoznak határoljuk be.)

$$R = \max(5\sigma, r_{kmin}) \quad (5.1)$$

$$K = [k_{min} * \frac{R\Delta}{r^2 k_{min}}] \quad (5.2)$$

Keretbecslés és felületi interpoláció: Ebben a szakaszban P_i pont hoz N_i -pont segítségével vesszük, hogy megközelítsük a zajos régiót, amelyre P_i - kerül. A fentiekben ismertetett zajszűrőnek szüksége van egy úgynevezett rácsra, és interpolálni kel N_i -ben lévő pontokat egy olyan tartományon, amely körülveszi P_i érintő sík közelítését. Ehhez szükség lesz N_i normálvektorára. Itt használhatunk főkomponens elemzést (PCA).

Szegmensek normálmenti frissítése: Az interpolációt követően kapunk egy I_i -t. P_i frissítéséhez Euler-lépést kell végrehajtani I_i felett (egyszer). Ez után a módosult „köz-ponti” pixel δi -t kivonjuk mivel ez pontosan P_i -nek felel meg. i -pontot egy explicit lépéssel frissítjük.

$$p_i^{t+1} = p_i^t + \delta_i^t * n_i^t \quad (5.3)$$

Itt t a globális iterációs lépéseket jelöli. Fontos, hogy diszkrét áramlás számításakor az adott skálázási tényezőknek rögzítettnek kell lennie, amely arányos a rács méreteihez nézve, illetve fordítottan arányos R sugárral. A gyakorlatban viszont ez nem mindig van így emiatt a skálázási paraméter minimális szinten, de rugalmasan kezelhető. Jegyezzük meg azt is ennek az értéknek a módosítása esetén frissül a magasság térkép átlagos görbületi áramlása (Más fajta lesz, mint az eredeti).

Robosztus normálérték becslés: A módszer hatékonysága nagyban függ attól, hogy mennyire sikerül minden ponton jól megbecsülni a normálértéket. Az előbb ismertetett „PCA” metódus egész jónak mondható, viszont a Robosztus normálérték becslés egy jóval pontosabb megoldást ad eredményül.

Az eljárás lényege egy iteratív folyamat, ami hasonló az RANSAC megközelítéshez. Egy szomszédságból származó pontkészletet épít fel segédként és ezt felhasználva tesztel, hogy megtalálja a legkevésbé kiugró értékkel rendelkező normálvektort. Összeségében nézve hatékonyabb, mint a feljebb említett PCA metódus.

5.2 Gauss-szűrő

A Gauss-szűrő egy lineáris szűrő, amely a képen egy Gauss-görbe alakú kernel segítségével valósítja meg a simítást. A Gauss-függvény alakja a következő:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \quad (5.4)$$

A Gauss-szűrő alkalmazása során a kép minden pixelére egy kernelt alkalmazunk, amelynek elemei a Gauss-függvény értékei a különböző pozíciókban. A kernel mérete és a szórás értéke meghatározza, hogy milyen mértékben történik a simítás és a zaj ki-szűrése. A működése egyszerű: az egyes pixelek értékét átlagoljuk a környező pixelek értékeivel, súlyozva a Gauss-függvény értékeivel. Ez a simítás hatását eredményezi, mivel a szűrő átlagolja a közeli pixelek értékeit. A zajcsökkentés érdekében a Gauss-szűrő távolabb eső pixeleket is figyelembe veszi, hogy a környező zajos pixelek értékeit is kiegyenlítse. A szórás értéke a Gauss-függvény görbéjének szélességét határozza meg, ami meghatározza, hogy milyen messze terjedjen ki a szűrő hatása.

5.3 Gauss derivált-szűrő

A Gauss derivált szűrők az élek és a textúrák kiemelésére, illetve a képek éldetektálására és jellemzésére szolgálnak. A Gauss-szűrő alapján épülnek fel, differenciálás segítségével kiszámítják a kép változásainak gradiensét. Ennek eredményeként nem csak a képet simítják, hanem az élek irányát és erősségét is kiemelik.

Gauss-első deriváltja (Gaussian first derivative): Ez a szűrő a Gauss-függvény első deriváltjával számolja ki a kép gradiensét. Az éleket a képen lévő intenzitásváltozásokra érzékenyítve kiemeli, és azok irányát is meghatározza.

$$\frac{dG(x)}{dx} = -\frac{x}{\sigma^2} \cdot \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) \quad (5.5)$$

Gauss-második deriváltja (Gaussian second derivative): Ez a szűrő a Gauss-függvény második deriváltját alkalmazza a képre. Az éleket és a textúrákat érzékelve kiemeli a kép helyi intenzitás-csúcsait és mélységeit.

$$\frac{d^2G(x)}{dx^2} = \left(\frac{x^2 - \sigma^2}{\sigma^4}\right) \cdot \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) \quad (5.6)$$

Mindkét szűrőt gyakran használják a képfeldolgozásban, például az objektumok detektálására, képanalízisre és képfelismerésre. Lehetővé teszik az élek, textúrák és jellemzők kiemelését a képeken, amelyek segíthetnek az objektumok felismerésében és azok tulajdonságainak meghatározásában.

6 SZOFTVERTERV

Ebben a fejezetben az Algoritmusok fejezetben megismert megoldások alapján összegzem, hogy a saját szoftveremben mely megoldás/megoldások kerülnek implementálásra.

6.1 Magasságtérképek beolvasása és mentése

A magasságtérképek beolvasása viszonylag egyszerűen implementálható, ugyanis csak a pixelenkénti magasságra lesz szükségünk. Megjelenítőt külön ez esetben nem implementáltam, ez egy későbbi fejlesztési lehetőség a szoftver szempontjából. A beolvasáshoz szükség lesz a felhasználótól elkérni különféle bemeneteket. Kelleni fog a terület legmagasabb pontja méterben számolva, ehhez társítva a legalacsonyabb pont, illetve a lefedett terület pontos mérete és persze maga a magasságtérkép. Ezen adatok segítségével már jól kiszámolható az adott pixelre eső magasság (az ehhez tartozó számításokat a „Megvalósítás” című fejezetben lehet megtalálni).

A fejlesztői környezet kiválasztásakor többféle szempontot is figyelembe vettem. A felület legyen egyszerű a felhasználók számára és ezzel együtt biztosítsa a jövőbeli fejlesztések lehetőségét. Képes legyen megjeleníteni az éppen feldolgozott képet és frissíteni azt. Emiatt C# WPF keretrendszerben készült el a felhasználói felület kihasználva ennek minden előnyét (adatkötés, közvetlen frissítés). Ennek köszönhetően a későbbiekben is könnyen fejleszthető marad az alkalmazás és az esetleges új funkciók implementálása is egyszerűsödik.

6.2 Növényzet detektálása

Növényzet detektálásakor már sokkal nehezebb dolgunk van, mint az előbbi esetben. Ugyanis a műholdképeken még az emberi szem számára sem rögtön egyértelmű, hogy adott pixelhalmazok most éppen egy fát, bokrot, vagy csak egy zöld tavat vagy épületet reprezentálnak. Persze pár másodperc pásztázás után az ember jó eséllyel meg tudja mondani, hogy mi található az adott pixelcsoportban. (Természetesen itt szoba jöhet a műholdképek minősége is, ami egy újabb problémát vet fel, de tételezzük fel, hogy a mai modern korban ez a probléma nem áll fent.)

Ezen okok miatt a legjobb választás valamilyen fajta gépi tanulási módszer, hiszen vegyük észre, hogy egy szimpla klaszterezési eljárással jóval nehezebb dolgunk lenne már csak a színek miatt is (nem csak a növények lehetnek zöldek). A megfelelő algoritmus kiválasztását az alábbi szempontok alapján tettem meg:

- **Legyen a jelen kornak megfelelő megoldás**
- **A megoldás viszonylag gyors és effektív legyen**
- **Képes legyen nagy adathalmazok kezelésére**
- **Lehetőleg minél pontosabb megoldást adjon**
- **Mindezt a lehető legrövidebb futásidő alatt**

A fenti szempontok alapján szóba jöhetnének akár a „Support Vector Machine”-ok vagy röviden SVM-ek, viszont sajnos nagyobb adathalmazokkal már nem vagy pedig nehezen bírnak el, így ez esetünkben nem jó megoldás. Ezesetben szóba jöhetne az „AdaBoost”, mivel nagy adathalmazon is működik és viszonylag pontos megoldást is adhatna, viszont az idő már eljárt felette, hisz a 90-es években környékén mutatták be, mindemellett pedig a tanításhoz szükséges idő is hosszas lehet nagyobb adathalmaz esetén. Elsőre szintén jó gondolatnak tűnhet valamilyen mélyneurális hálózat (Deep Network) felépítése a kívánt cél, azaz a fák pontos detektálásának eléréséhez. Ezzel viszont megint az a probléma, hogy nagyobb adathalmaz esetén a tanítás nagyon sok időt vehet igénybe, és a későbbi modell által történő fa detektálás is hosszabb ideig tarthat. Az előbb említett „AdaBoost” alapötletnek nem rossz, hisz az „ensemble” módszer (gyenge tanulók kombinálásából alkot jobbakat) amit alkalmaz esetünkben is hasznos lehet. A probléma az „elavultsága” erre viszont megoldást jelenthet a „LightGBM” vagy pedig a „FastTree” algoritmus. Mindkét algoritmus „boosting” elven működik, mint ahogyan az „AdaBoost” is (ezeket részleteztem a harmadik szekcióban). Ez a két algoritmus egyaránt alkalmas bináris klasszifikációra, és viszonylag gyorsan és hatékonyan képesek nagyon jó megoldásokat adni, viszont a „FastTree” valamivel több hiperparaméterrel rendelkezik (például „pruning” ami törli azokat az ágakat amik nem járulnak hozzá a modell pontosságának növekedéséhez), ami a tanítási folyamat során hasznunkra válhat. Ezen okoknak köszönhetően bizonyult jobb választásnak tehát a „FastTree”.

Első próbálkozásaim még „LightGBM” algoritmussal készültek, viszont egy idő után a pontosság és a sebesség is stagnálni, egyes esetekben pedig romlani kezdett. Emiatt a későbbiekben „FastTree” modellre váltottam és meg is lett az eredménye, amit majd az „Eredmények” névvel fémjelzett nyolcas szekcióban mutatok be.

7 MEGVALÓSÍTÁS

Jelen formájában a program Csharp kódból épül fel, a felhasználói felület megjelenítéséhez „Windows Presentation Foundation”-t (WPF) használ. Emellett pedig számos külső hivatalos, vagy épp közösség által fejlesztett úgynevezett NuGet csomagokat alkalmaz. Ilyen csomagok például: ML.NET, OpenCVSharp. Ezek a csomagok főként a képek feldolgozásában, illetve a fák detektálásának egyszerűbb és optimálisabb megoldásában segítenek. Fontos megjegyezni, hogy ezek a későbbi fejlesztések alatt még változhatnak.

7.1 Magasságtérkép beolvasása

A beolvasás többféle módon is meg lehet valósítani, én próbáltam a lehető leghatékonyabb és egyben valamelyest egyszerű megoldást implementálni. Ebben az implementációban nem hagyatkozhatunk a beépített képfeldolgozó függvényekre mint például a `GetPixel()` és társai, ugyanis ezek sajnos drasztikusan képesek csökkenteni a programunk teljesítményét.

Maga a folyamat nem bonyolult, először is beolvassuk a képet. A könnyebb megoldás az lenne, hogy 2 darab for-ciklussal végig mennénk a képen és közben számoljuk egy adott képlet alapján (később lesz róla szó) a magasságot. Viszont a dupla-for miatt ez lassú. Ehelyett én egy for-ciklussal dolgozom, mivel az adatok a memóriában sorban következnek. A ciklus pedig magasság és szélesség szorzatának értékéig fut. Amire viszont érdemes figyelni, hogy ekkor a színcsatornákat fordítva kell értelmezni RGB képek esetén. Azaz BGR sorrendben lesznek (ARGB $\rightarrow$ BGRA).

```

public static double[] GetPixelValues(Bitmap b)
{
    BitmapData bmData = b.LockBits(
        new Rectangle(0, 0, b.Width, b.Height),
        ImageLockMode.ReadWrite, PixelFormat.Format32bppArgb);
    System.IntPtr Scan0 = bmData.Scan0;

    double[] pixels;
    int pixelSize=b.Height*b.Width;
    unsafe
    {
        byte* p = (byte*)(void*)Scan0;
        pixels = new double[pixelSize];
        for (int i = 0; i < pixelSize; ++i)
        {
            pixels[i] = (((p[0] * 114) + (p[1] * 587)
                + (p[2] * 299)) * 255) / 1024;
            p += 4;
        }
    }
    b.UnlockBits(bmData);
    return pixels;
}

```

7.1. ábra: Kép beolvasása

$$grayScale = 0.3 * R + 0.59 * G + 0.11 * B \quad (7.1)$$

Jól látszik a megfordított értékek alkalmazása, illetve még egy minimális optimalizáció. Ugyanis a tört értékek fel lettek szorozva 1000-rel mert így maga az szorzás folyamata gyorsabb mintha tört számokkal lenne, majd 1024-el osztjuk hogy kihasználjuk a „shiftelés” adta lehetőséget szintén a gyorsítás érdekében.

Felhasznált változók:

- p: Az adott pixelértékek pointere
- pixels[]: A beolvasott pixeleket tartalmazza átalakítva szürkeárnyalatossá
- bmData: Maga a kép
- pixelSize: Kép mérete (magasság szorozva a szélességgel)

7.2 Exportálás

Exportálni is hasonló módon lehetséges, itt most a .ASC fájlba való kiexportálást részletezném. Maga a kimenet pixelre pontosan megadja, hogy mekkora az adott magasság tehát egy képmagasság*képszélesség méretű táblázatot kell elképzelni, amiben ezek az adatok szerepelnek. Illetve a terület valós méretei.

1	ncols	4096
2	nrows	4096
3	xllcorner	0.000000
4	yllcorner	0.000000
5	cellsize	6.250000
6	NODATA_value	-9999
7	-9.62	-9.85 -9.93 -9.94 -9.82 -9.75 -9.81 -9.75 -9.81 -9.75 -9.81 -9.71 -9.55 -8.67
8	-9.69	-9.54 -9.44 -9.27 -8.99 -9.06 -8.99 -9.06 -8.99 -9.06 -8.99 -9.05 -8.91 -8.67
9	-9.82	-9.42 -9.10 -8.94 -8.77 -8.73 -8.77 -8.73 -8.77 -8.73 -8.77 -8.73 -8.74 -8.67
10	-9.92	-9.33 -8.99 -8.70 -8.68 -8.69 -8.68 -8.69 -8.68 -8.69 -8.68 -8.69 -8.68 -8.67
11	-9.94	-9.35 -8.91 -8.68 -8.68 -8.67 -8.68 -8.67 -8.68 -8.67 -8.68 -8.67 -8.68 -8.67
12	-9.97	-9.29 -8.90 -8.69 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67
13	-9.94	-9.25 -8.77 -8.68 -8.68 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67
14	-9.88	-9.06 -8.73 -8.69 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67
15	-9.81	-8.99 -8.77 -8.68 -8.68 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67
16	-9.75	-9.06 -8.73 -8.69 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67
17	-9.81	-8.99 -8.77 -8.68 -8.68 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67
18	-9.75	-9.06 -8.73 -8.69 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67
19	-9.81	-8.99 -8.77 -8.68 -8.68 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67
20	-9.74	-9.06 -8.73 -8.69 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67
21	-9.77	-8.97 -8.76 -8.68 -8.68 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67
22	-9.52	-8.94 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67
23	-8.67	-8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67
24	-8.67	-8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -9.78
25	-8.67	-8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -9.78 -9.78
26	-8.67	-8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -9.78 -9.78 -9.78
27	-8.67	-8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -9.78 -9.78 -9.78 -9.78
28	-8.67	-8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -8.67 -9.78 -9.78 -9.78 -9.78 -9.78

7.2. ábra: ASC fájl (részlet)

A fenti ábrán látható egy ilyen fájl felépítése. Felül a terület adatok láthatók, illetve a konvertált kép mérete.

Táblázat adatainak leírása:

- ncols: bemeneti kép szélessége („táblázat” oszlopai)
- nrows: bemeneti kép magassága („táblázat” sorai)
- xllcorner: X koordináta a programban való elhelyezésre
- yllcorner: Y koordináta a programban való elhelyezésre
- cellsize: Egy pixelre eső méret/távolság (az ábrán például egy pixel 6.25 métert reprezentál)

- NODATA value: Ha valahol nincs adatunk oda az itt meghatározott érték kerül

Egy ilyen fájl generálásához szükségünk lesz a felhasználó által kívánt minimum, illetve maximum magasságra, egy magasságtérképre, illetve, hogy a magasságtérkép mekkora területet reprezentál a valóságban. Ezek a felhasználó által megadott bemeneti értékek lesznek. A pixelenkénti magasság kiszámítása a következőképp alakul:

$$H_{min} + \left(\frac{data}{limit}\right) * (H_{max} - H_{min}) \quad (7.2)$$

- H_{min} = minimum magasság
- H_{max} = maximum magasság
- $data$ = az adott pixelhez tartozó érték (0-255)
- $limit$ = kép bitmélysége alapján, például 16bit $\rightarrow 2^{16}$

```
int newLine = 0;
int space = 0;
for (int i = 0; i < adim[0] * adim[1]; i++)
{
    streamWriter.Write("");
    if (newLine > adim[0])
    {
        streamWriter.WriteLine();
        newLine = -1;
    }
    if (space >= adim[1]-1)
    {
        space = 0;
    }
    streamWriter.Write(Math.Round((map.Hmin + (imgData[i]
/ limit) * (map.Hmax - map.Hmin)), 2));
    if (space < adim[1] - 1)
    {
        streamWriter.Write(" ");
    }
    space++;
    newLine++;
}
```

7.3. ábra: ASC exportálás

7.3 Növényzet detektálás

Mint ahogyan az fentebb említésre került, a növények felismeréséhez szükségünk lesz különböző „feature” és „label” értékekre. A „feature” értékek az éppen vizsgált pixel szín, textúra, entrópia(intenzitás) értékeiből adódik. A „label” értékek pedig az adott pixelre vonatkozóan annyit tárolnak, hogy fa e vagy pedig nem fa. Ezekre azért lesz szükség, hogy be tudjuk tanítani a FastTree algoritmust.

7.3.1 Feature értékek számítása

A **szín** értékek számításakor az RGB képet CIE Lab képpé konvertálok mert ebből jobban kivehetők a különböző elemek abban az esetben, ha e mellé még kiszámoljuk az adott pixelre vonatkozó fény/megvilágítási viszonyokat (fényességi adatok, illumination-invariant). Ez 6db „feature” értéket jelent(Ugyan úgy, mint az „RGB” képek esetében 3db R+G+B, itt pedig a „Lab” és „illumination-invariant” miatt 3+3). A következőkben pedig részletesen ismertetem az algoritmus működését.

```
private float[][] ComputeColorFeatures(Mat image)
{
    var features = new float[image.Width * image.Height][];
    Mat labImage = new Mat();
    image.ConvertTo(labImage, MatType.CV_32FC3, 1.0 / 255.0); // Convert to float and normalize to [0, 1]
    abImage = labImage.CvtColor(ColorConversionCodes.BGR2Lab); // Convert to LAB color space

    Parallel.For(0, image.Height, y =>
    {
        for (int x = 0; x < image.Width; x++)
        {
            int index = y * image.Width + x;
            features[index] = CalculateColorFeatures(labImage, image, x, y);
        }
    });

    return features;
}

private float[] CalculateColorFeatures(Mat labImage, Mat image, int x, int y)
{
    float[] colorFeatures = new float[6];

    var labPixel = labImage.At<Vec3f>(y, x);
    colorFeatures[0] = labPixel.Item0; // L channel
    colorFeatures[1] = labPixel.Item1; // a channel
    colorFeatures[2] = labPixel.Item2; // b channel

    var grayscale = 0.2126f * image.At<Vec3b>(y, x).Item0 +
        0.7152f * image.At<Vec3b>(y, x).Item1 +
        0.0722f * image.At<Vec3b>(y, x).Item2;

    colorFeatures[3] = grayscale; // grayscale value
    colorFeatures[4] = labPixel.Item0 - grayscale; // L minus grayscale
    colorFeatures[5] = labPixel.Item1 - labPixel.Item2; // a minus b

    return colorFeatures;
}
```

7.4. ábra: Szín értékek számítása

Ez a módszer egy Mat (mátrix) formában tárolt bemeneti képet vesz, és egy lebegőpontos álló tömböt ad vissza, amely a kép minden egyes pixelének színjellemzőit tartalmazza. Először létrehoz egy új, Features nevű float-tömböt, amelynek mérete megegyezik a bemeneti kép pixeleinek számával. Ez a tömb fogja tárolni az egyes pixellek színjellemzőit. Ezután a bemeneti képet az OpenCV könyvtár CvtColor függvényével LAB színtérbe konvertálja. Emellett lebegőpontos ábrázolásra konvertálja és a $[0, 1]$ intervallumon normalizálja az értékeket. A kód ezután egy párhuzamos ciklusba lép, ahol végig iterál a kép minden egyes pixelén. A cikluson belül a CalculateColorFeatures módszer meghívására kerül az egyes pixellek színjellemzőinek kiszámításához, és az eredményeket a Features tömbben tárolja. Végül az összes pixel színjellemzőit tartalmazó tömb kerül visszaadásra.

Az előzőekben bemutatott módszer segédfüggvénye a „CalculateColorFeatures” kiszámítja egy adott pixel színjellemzőit LAB színtérben. Bemenetként a kép LAB ábrázolását, az eredeti képet és azokat a pixelkoordinátákat (x, y) veszi fel, amelyekre a jellemzőket ki kell számítani. A függvény létrehoz egy colorFeatures nevű float-tömböt, amely az adott pixelre számított színjellemzők tárolására szolgál. A pixel LAB-értékeit (L, a, b) az `At<Vec3f>` módszerrel kapjuk meg a `labImage`-ből, és a `colorFeatures` tömbben tároljuk. Az `Item0`, `Item1` és `Item2` az L, a és b csatornát jelöli. Ugyanannak a pixelnek a szürkeárnyaltos értékét az eredeti kép három színcsatornájának (R, G, B) súlyozott összegével számoljuk ki. A súlyok megfelelnek az egyes csatornák fényerősségi értékeinek, és ezt az eredményt a `colorFeatures` tömb 3.indexén tároljuk. Az L-csatorna és a szürkeárnyaltos érték közötti különbséget a `colorFeatures` 4.indexén tároljuk. Ez a jellemző a szürkeárnyaltos túli fényerősségváltozásokat is megadja. Az a és a b csatorna közötti különbséget a `colorFeatures` 5.indexe tárolja. Ez a jellemző a zöld-magenta tengely mentén történő színváltozást rögzíti.

A **textúra** számítása is fontos tényező. A falevelek miatt főként, hisz elég sajátos jellemzőik vannak, amik nem csak szemmel láthatóak, hanem számokban is jól kifejezhetőek. Ez ahhoz is szükséges például, hogy az azonos színű ám de nem fa objektumokat meg tudja különböztetni az algoritmus. Ezen értékek számításához minden egyes L értéket (Lab értékekből az L) összevontam egy szűrő bankba. Szűrőnek pedig Gauss féle derivált szűrőt alkalmaztam. Első deriváltja X irányú a második pedig Y irányú. A szűrő 3 skálán 1, $\sqrt{2}$ és 2 értékekkel, illetve 6 orientációban került alkalmazásra így alkotva 18 „feature” értéket. Ez az algoritmus a következőképp épül fel:

```

private float [][] ComputeTextureFeatures(Mat labImage)
{
    // Convert the Lab image to grayscale
    Mat rgbImage = new Mat();
    Mat grayscaleImage = new Mat();
    Cv2.CvtColor(labImage, rgbImage, ColorConversionCodes.Lab2BGR);
    Cv2.CvtColor(rgbImage, grayscaleImage, ColorConversionCodes.BGR2GRAY);

    // Define filter parameters
    float[] scales = { 1.0f, (float)Math.Sqrt(2), 2.0f };
    int orientations = 6;
    int featureVectorLength = scales.Length * orientations;

    // Create a filter bank and apply filters
    List<Mat> filterResponses = new List<Mat>();

    foreach (float scale in scales)
    {
        for (int orientation = 0; orientation < orientations; orientation++)
        {
            float angle = (float)(orientation * Math.PI / orientations);
            float sigmaX = scale;
            float sigmaY = scale / 2.0f; // Second derivative in Y direction

            // Create Gaussian derivative filter
            Mat filterResponse = new Mat();
            Cv2.GaussianBlur(grayscaleImage, filterResponse, new OpenCvSharp.Size(0, 0), sigmaX);

            // Create Gaussian derivative filter in X direction
            Mat kernelX = new Mat(1, 3, MatType.CV_32F, new float[] { -1, 0, 1 });

            // Create Gaussian derivative filter in Y direction
            Mat kernelY = new Mat(3, 1, MatType.CV_32F, new float[] { -1, 0, 1 });

            // Apply the filters
            Cv2.Filter2D(filterResponse, filterResponse, MatType.CV_32F, kernelX);
            Cv2.Filter2D(filterResponse, filterResponse, MatType.CV_32F, kernelY);

            Cv2.GaussianBlur(filterResponse, filterResponse, new OpenCvSharp.Size(0, 0), sigmaY);

            // Store the filter response
            filterResponses.Add(filterResponse);
        }
    }

    // Extract texture features from the filter responses
    float [][] textureFeatures = new float[labImage.Rows * labImage.Cols][];
    int index = 0;

    for (int y = 0; y < labImage.Rows; y++)
    {
        for (int x = 0; x < labImage.Cols; x++)
        {
            float[] featureVector = new float[featureVectorLength];

            for (int i = 0; i < filterResponses.Count; i++)
            {
                // Calculate the feature value at this pixel and orientation
                featureVector[i] = (float)filterResponses[i].At<float>(y, x);
            }

            textureFeatures[index++] = featureVector;
        }
    }

    return textureFeatures;
}

```

7.5. ábra: Textúra értékek számítása

Ez a függvény egy Mat (mátrix) formában ábrázolt LAB színeképet vár bemenetként, és egy lebegőpontos értékeket tartalmazó (float)tömböt ad vissza, amely a kép minden egyes pixelének textúrajellemzőit tartalmazza. A LAB színeképet először RGB színeképpé, majd szürkeárnyaltos képpé konvertálja (a szürkeárnyaltos képet a textúraelemzéshez

használjuk). A szűrő paraméterei definiálva vannak, beleértve a szűrők különböző skáláit és orientációit. Az egyes skálákhoz és orientációkhoz tartozó szűrők számát a következő módon kell kiszámítani: skála mérete szorozva az orientációk számával. Ezt követően létrejön egy `filterResponses` nevű lista, amely az egyes skálákhoz és orientációkhoz tartozó szűrőket tárolja.

A továbbiakban a metódus minden egyes lehetséges méretarány- és tájolási kombináción végig iterál. Minden egyes kombináció esetében a következő lépéseket hajtja végre: Kiszámítja a tájolásnak megfelelő szöget radiánban. Beállítja a `sigmaX` és `sigmaY` értékeket, amelyek meghatározzák a Gauss derivált szűrők X és Y irányú standard eltérését. A szürkeárnyaltos képre a megadott `sigmaX` érték felhasználásával egy Gauss-féle szűrőt alkalmaz, létrehozva egy szűrőválaszt. Az X és Y irányban is létrehozza a Gauss derivált szűrőket. A szűrőválasz konvolválásra kerül mind az X, mind az Y derivált szűrőkkel. Az így kapott szűrőválaszra a `sigmaY` érték segítségével egy újabb Gauss-féle szűrőt alkalmaz. A végső szűrőválasz hozzáadásra kerül a `filterResponses` listához.

Az összes skála és orientáció szűrőválaszainak létrehozása után az algoritmus előkészíti a textúrajellemzők kinyerését. Létrejön egy `textureFeatures` nevű kétdimenziós tömb, amelynek mérete megegyezik a bemeneti kép sorainak és oszlopainak szorzatának számával. Ez a tömb tárolja az egyes pixelek textúrajellemzőit. A függvény ezután egy egymásba ágyazott ciklusba lép, és végig iterál a bemeneti kép minden egyes pixelén. Ebben a ciklusban minden egyes pixelhez létrehoz egy jellemzővektort. A jellemzővektor hosszát a szűrőválaszok száma határozza meg. A jellemzővektor minden egyes szűrőválaszhoz tárolja a szűrőválasz értékét a pixelre nézve. A pixelhez tartozó teljes jellemzővektor hozzáadódik a `textureFeatures` tömbhöz. A függvény visszatérési értéke egy lebegőpontos értékeket tartalmazó tömb, ami ebben az esetben a pixelek textúra jellemzőit tartalmazza.

Az **entrópia**, avagy „rendezetlenség” számítása az utolsó lépés. Ez azért fontos mert a fáknak és az árnyékoknak ez az értéke sokkal nagyobb, mint bármi más objektumé a képen. Ezt a pixelek `L` értékéből lehet számítani. A számítás során 5×5 , 9×9 és 17×17 méretű keretekkel megyek végig a kép `L` csatornáján. Ezt követően az értékeket egy 3 dimenziós tömbbe, mentjük, ami tartalmazza minden egyes pixel entrópia értékét. Ez az algoritmus a következőképp működik:

```

// Constants
private const int WindowRadius = 8;
private const int WindowSizeIncrement = 4;

private float [][] ComputeEntropyFeatures(Mat labImage)
{
    var entropyFeatures = new List<float []>();
    var bgrImage = new Mat();
    Cv2.CvtColor(labImage, bgrImage, ColorConversionCodes.Lab2RGB);
    var grayImage = new Mat();
    Cv2.CvtColor(bgrImage, grayImage, ColorConversionCodes.BGR2GRAY);

    // Precompute the logarithmic function lookup table
    var logTable = new float[256];
    for (int i = 1; i < 256; i++)
    {
        logTable[i] = (float)Math.Log(i);
    }

    for (int y = 0; y < labImage.Height; y++)
    {
        for (int x = 0; x < labImage.Width; x++)
        {
            var entropyValues = new float[3];
            var pixelValue = grayImage.At<byte>(y, x);
            var histogram = new int[256];

            // Compute the histogram for the pixel
            for (int j = -WindowRadius; j <= WindowRadius; j++)
            {
                for (int k = -WindowRadius; k <= WindowRadius; k++)
                {
                    var neighborX = x + k;
                    var neighborY = y + j;

                    if (IsValidPixel(neighborX, neighborY, labImage))
                    {
                        var neighborValue = grayImage.At<byte>(neighborY, neighborX);
                        histogram[neighborValue]++;
                    }
                }
            }

            // Compute the entropy for different window sizes
            for (int i = 0; i < 3; i++)
            {
                var searchWindowSize = (i + 1) * WindowSizeIncrement + 1;
                var entropy = 0f;
                var totalPixels = searchWindowSize * searchWindowSize;

                for (int j = 0; j < 256; j++)
                {
                    if (histogram[j] != 0)
                    {
                        var probability = (float)histogram[j] / totalPixels;
                        entropy += -probability * logTable[j];
                    }
                }

                entropyValues[i] = entropy;
            }

            entropyFeatures.Add(entropyValues);
        }
    }

    return entropyFeatures.ToArray();
}

private bool IsValidPixel(int x, int y, Mat image)
{
    return x >= 0 && x < image.Width && y >= 0 && y < image.Height;
}

```

7.6. ábra: Entrópia értékek számítása

Ez a függvény egy LAB színterű bemeneti képet vár, és entrópia-alapú textúrajellemzőket számol ki a kép minden egyes pixelére. A kiszámított jellemzők tárolására egy üres listát inicializál, amelynek neve `entropyFeatures`. A bemeneti képet először RGB képpé, majd szürkeárnyaltos képpé (`grayImage`) konvertálja az OpenCV könyvtár függvényei segítségével. Előre kiszámít egy logaritmikus függvénykereső táblázatot (`logTable`), amely az 1 és 255 közötti értékek logaritmusát tárolja. Ezt a táblázatot az entrópia számítások során fogjuk használni. A metódus ezután a bemeneti kép minden egyes pixelén végigmegy. Minden egyes pixelre különböző méretű úgynevezett „ablakokon” belül eső entrópia-alapú jellemzőket számol. Minden egyes pixelre három entrópiaértéket számítunk ki, amelyek három különböző ablakméretnek felelnek meg (a `WindowSizeIncrement` határozza meg). Az algoritmus kiszámítja a szürkeárnyaltos pixelértékek hisztogramját az aktuális pixel körüli ablakon belül. A hisztogram ez esetben egy 256 mezőt tartalmazó tömb (minden lehetséges szürkeárnyalt-értékhez egy mező). A hisztogram segítségével kiszámítjuk az entrópiát mindhárom ablakméretre, figyelembe véve az ablakon belül előforduló különböző szürkeárnyalt-értékek valószínűségét. A kiszámított entrópiaértékeket egy tömbben (`entropyValues`) tárolja, és ezt a tömböt hozzáadja az `entropyFeatures` listához. A metódus végrehajtja ezt a folyamatot a kép összes pixelén, végül az `entropyFeatures` listát kétdimenziós float tömbbé alakítja és visszaadja.

Az `IsValidPixel` egy segéd metódus, amely ellenőrzi, hogy egy adott pixelkoordináta (x, y) a bemeneti kép határain belül van-e. Igazat ad vissza, ha a pixel érvényes és a kép határain belül van, ellenkező esetben hamisat.

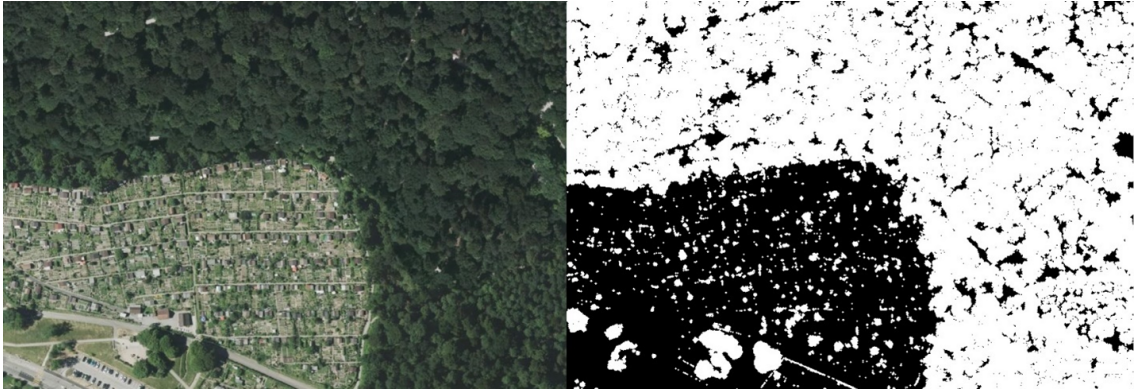
Végül ezeket az adatokat pixelenként konkatenálom egy tömbbe ezzel megkapva minden egyes pixelhez tartozó 27 db „feature” értéket [26].

7.3.2 Label értékek számítása

A „label” értékek számítása határozottan könnyebb. Először is szükség lesz olyan képekre, amik 2 színből állnak és az egyik szín a fákat jelöli a képen míg a másik azokat a területeket, ahol nem található fa. Általában ez a két szín a fekete és a fehér. Esetemben a fehér jelöli a fákat és a fekete pedig azokat a pixeleket, amik nem fához tartoznak. Ilyen képeket lehet kézzel is előállítani photoshop vagy gimp segítségével, vagy pedig letölteni. Az utóbbi picit nehezebb mert használható adatokat Európában nagyon kevés ország tesz ingyenesen hozzáférhetővé, de szerencsére Németországnak van ilyen ingyenesen hozzáférhető adathalmaz, én ezt használtam.

Az algoritmus maga pofon egyszerű, csak fogjuk a képeket és beolvassuk majd megnézzük, hogy adott pixel fehér-e, vagy fekete és ennek megfelelően egy tömbben eltárolunk igaz/hamis, vagy pedig 0 vagy 255 értékeket. Esetemben 0 vagy 255 értékek kerülnek a tömbbe, 0 jelenti azt a pixelt, ami nem fát tartalmaz és 255, ami fát. Ezeket

az értékeket kombinálva a „feature” értékekkel pedig képesek vagyunk erre megfelelő algoritmusokat betanítani.



7.7. ábra: Minták: balra a műholdkép, jobbra pedig a jelölt növényzet

```
public bool[] BuildResponseFromFilePath(string imgFilePath, byte treeVal = 255, byte nontreeVal = 0)
{
    using (var bmp = new Bitmap(imgFilePath))
    {
        var width = bmp.Width;
        var height = bmp.Height;

        // Check that the image is 8-bit grayscale
        if (bmp.PixelFormat != PixelFormat.Format8bppIndexed)
        {
            throw new ArgumentException(
                $"The response mask {imgFilePath} must be an 8-bit grayscale image"
            );
        }

        // Extract the image data
        var imageData = new byte[width * height];
        var data = bmp.LockBits(new System.Drawing.Rectangle(0, 0, width, height),
            ImageLockMode.ReadOnly, PixelFormat.Format8bppIndexed);
        Marshal.Copy(data.Scan0, imageData, 0, imageData.Length);
        bmp.UnlockBits(data);

        // Convert the image data to an int array
        var responseArr = new bool[width * height];
        for (var i = 0; i < imageData.Length; i++)
        {
            if (imageData[i] == treeVal)
            {
                responseArr[i] = true;
            }
            else if (imageData[i] == nontreeVal)
            {
                responseArr[i] = false;
            }
            else
            {
                throw new ArgumentException(
                    $"The response mask {imgFilePath} must consist of only " +
                    $"{treeVal} (tree) and {nontreeVal} (non-tree) " +
                    "pixel values"
                );
            }
        }
        return responseArr;
    }
}
```

7.8. ábra: Label értékek számítása

Ez a függvény a képfájl elérési útvonalát (imgFilePath) és két opcionális paramétert (treeVal és nontreeVal) vár bemenetként, amelyek megadják a „fa” és „nem fa” értékként kezelendő pixelértékeket. A módszer a(z) using utasítással biztosítja, hogy a Bitmap objektum a használat után törlődjön a memóriából. A kép szélességét és magasságát a Bitmap Width és Height tulajdonságainak használatával állítja be.

A metódus ellenőrzi, hogy a kép pixelformátuma 8 bites szürkeárnyaltos e. Ha a formátum nem 8 bites szürkeárnyaltos, az algoritmus hibát dob, jelezve, hogy a bemeneti képnek 8 bites szürkeárnyaltos képnek kell lennie. A kód a kép adatokat a kép bitjeinek a bmp.LockBits használatával történő zárolásával vonja ki. Ez lehetővé teszi a pixeladatok közvetlen hozzáférését feldolgozás közben. Létrehoz egy imageData nevű bájt tömböt a pixelértékek tárolására. A tömb méretét a kép szélességének és magasságának szorzata határozza meg.

A képadatokat ezután a responseArr nevű boolean tömbé konvertáljuk. Ez a tömb tartalmazza a megadott pixelértékek alapján a „fa” vagy „nem fa” értékeket. A függvény végigmegy az imageData minden egyes pixelértékén, és kiértékeli azt: Ha a pixelérték megegyezik a treeVal értékkel a responseArr megfelelő indexén lévő bool érték igaz lesz. Ha a pixelérték megegyezik a nontreeVal értékkel, akkor a responseArr megfelelő indexén lévő bool elem hamis értékű lesz. Ha a pixelérték egyik értéknek sem felel meg, akkor hibát dob, jelezve, hogy a válaszmátrixnak csak a megadott pixelértékekből kell állnia. Az eredményül kapott responseArr tömb a függvény végén visszaadásra kerül.

7.3.3 Tanítás

Miután elkészültek a szükséges „Feature” és „Label” értékek elkezdődhet a modell tanítása. Mint ahogyan az már korábban említettem, a tanítás ML.net keretrendszerben zajlik. Ezen belül pedig a „FastTree” algoritmussal. A következőkben az általam használt hiperparamétereket fogom bemutatni.

```
var pipeline = context.BinaryClassification.Trainers
    .FastTree(new FastTreeBinaryTrainer.Options()
    {
        LabelColumnName = @"Label",
        FeatureColumnName = @"Features",
        NumberOfLeaves = 10,
        MinimumExampleCountPerLeaf = 2,
        LearningRate = 0.0075,
        NumberOfTrees = 300,
        MaximumBinCountPerFeature = 550,
        MinimumStepSize = 0.025,
        EnablePruning = true,
        UnbalancedSets = true,
        OptimizationAlgorithm = BoostedTreeOptions.OptimizationAlgorithmType.GradientDescent
    });
```

7.9. ábra: Tanítási paraméterek

- **LabelColumnName:** Ez a paraméter az adatkészlet azon oszlopának nevét adja meg, amely a célcímkeket (label értékeket) tartalmazza. A tréner ezt az oszlopot fogja használni a modell betanításához, hogy a jellemzők alapján megjósolja a címkeket (pl. „fa”(true) vagy „nem fa”(false)).
- **FeatureColumnName:** Ez a tulajdonság az adatkészlet azon oszlopának nevét adja meg, amely a modell képzéséhez használt jellemzőket vagy bemeneti adatokat tartalmazza. A tréner ezt az oszlopot fogja használni a jellemzők és a célcímkek közötti kapcsolatok megtanulására (esetünkben ez a 27 darab feature per pixel).
- **NumberOfLeaves:** Ez az opció a FastTree bináris osztályozó által létrehozott egyes döntési fák leveleinek (végcsomópontjainak) számát szabályozza. A levelek kisebb száma egyszerűbb fákat eredményezhet, míg a nagyobb szám összetettebb modelleket eredményezhet.
- **MinimumExampleCountPerLeaf:** Ez a paraméter határozza meg a példák minimális számát, amelyet egy levélsomópontnak tartalmaznia kell. A kisebb értékek részletesebb modelleket eredményezhetnek, de túlillesztéshez vezethetnek.
- **LearningRate:** A tanulási ráta egy olyan hiperparaméter, amely a modell paramétereinek a tanítás során történő frissítéséhez szükséges lépésméretet szabályozza. A kisebb tanulási ráta stabilabb konvergenciához vezethet, de több iterációt igényelhet a tanítás (hosszabb tanulási idő).
- **NumberOfTrees:** Ez a tulajdonság a létrehozandó döntési fák számát adja meg. A boosting algoritmusok gyakran több fát alkalmaznak a predikciók elkészítéséhez, és ez a paraméter szabályozza az modellben lévő fák számát. Több fa részletesebb modellt képes felállítani, de ronthat a modell általánosító képességén és növeli a tanítási időt is.
- **MaximumBinCountPerFeature:** Ez az adattag határozza meg a numerikus jellemzők diszkrétizálásához használt minták vagy intervallumok maximális számát. Több minta esetén a részletesebb lehet a modell, de növelheti annak összetettségét is.
- **MinimumStepSize:** A minimális lépésméret egy olyan tulajdonság, amely befolyásolja a tanítás alatti konvergencia alakulását. A kisebb értékek pontosabb konvergenciát eredményezhetnek, de lassabb képzést is eredményezhetnek.
- **EnablePruning:** A metszés egy olyan technika, amely a döntési fák komplexitásának csökkentésére szolgál azáltal, hogy eltávolítja azokat az ágakat, amelyek nem járulnak hozzá a modell pontosságának javításához. A pontosság megállapításához szükséges validációs adathalmazt létrehozni.
- **UnbalancedSets:** Ez azt jelzi, hogy az adatkészlet kiegyensúlyozatlan, ami annyit takar, hogy a célcímkek között jelentős osztályegyenlőtlenség lehet (például több az igaz vagy a hamis értékek száma a tanító adathalmazban). Ha ez igazra van állítva az algoritmus alkalmazhat különböző technikákat a kiegyensúlyozatlan adat-

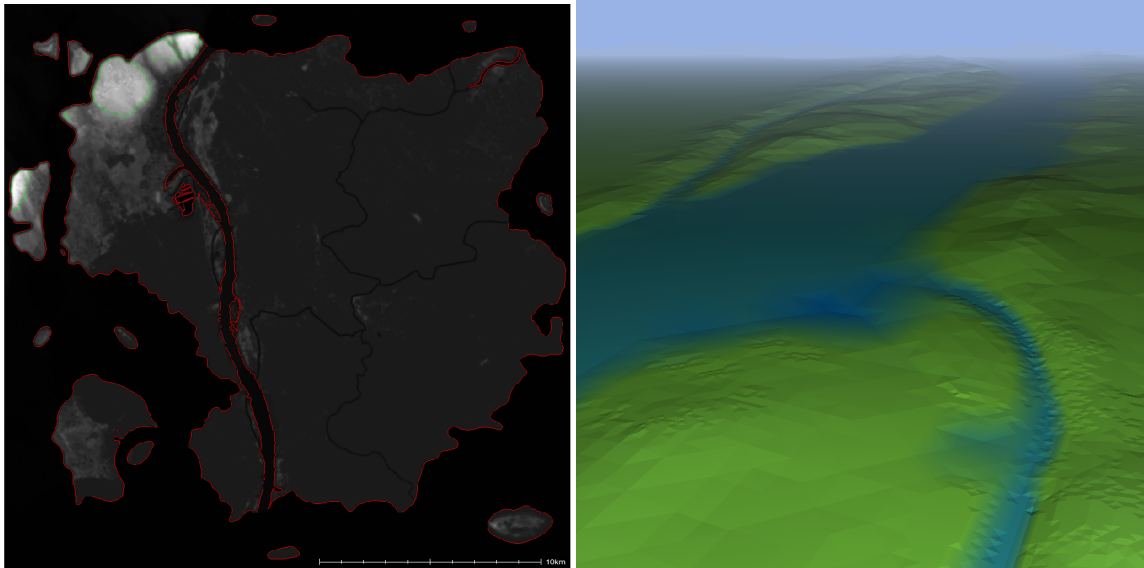
halmazok kezelésére. (Fák detektálása esetén szinte biztos a kiegyensúlyozatlan adathalmaz, így ezt szükségszerű alkalmazni)

- **OptimizationAlgorithm:** Ez a paraméter a képzés során használt optimalizációs algoritmust adja meg.

A fent szemléltetett paraméterleírásokból is jól látszik, hogy ritka az az eset mikor egy ilyen klasszifikációs problémára egzakt, valamilyen formában előre megadható megoldásunk van. Igazából egyik paraméter sem adható meg matematikai számítás alapján, sokkal inkább a tapasztalatokra lehet hagyatkozni. Nyilván vannak különböző keretek, ami között érdemes mozogni. Például a „learning rate” tipikusan 0 és 1 közötti érték és általában 0.1 és 000.1 között szokás alkalmazni. A levelek száma és a fák mérete a feladat komplexitásától függenek leginkább. Én a megoldásomban a tapasztalataimra, illetve a fent bemutatottakra hagyatkoztam és a próbálgatások során kijött eredmények alapján finomítottam a paramétereken, míg a végső megoldáshoz nem értem el.

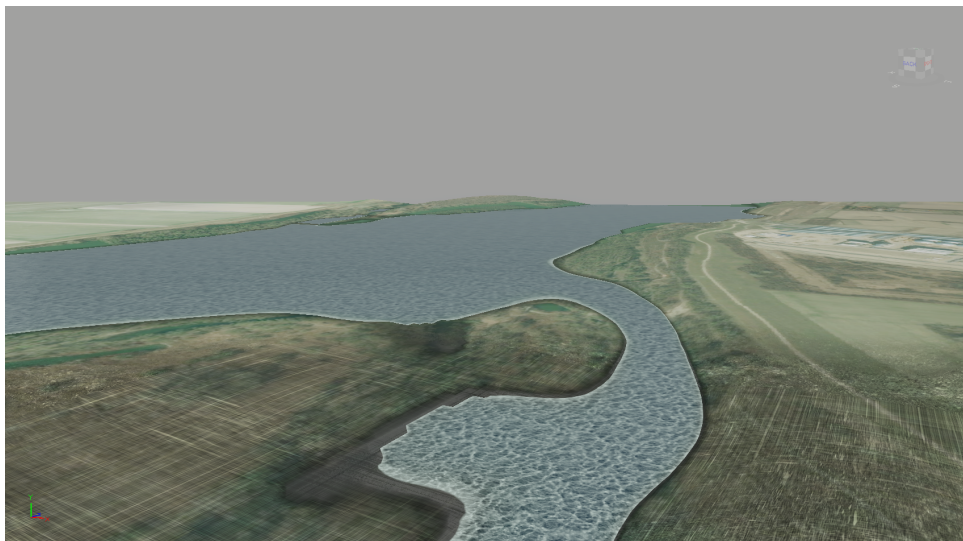
8 EREDMÉNYEK

8.1 Terület fájl generálás



8.1. ábra: Balra a bementi kép, jobbra pedig a 3D területből egy részlet

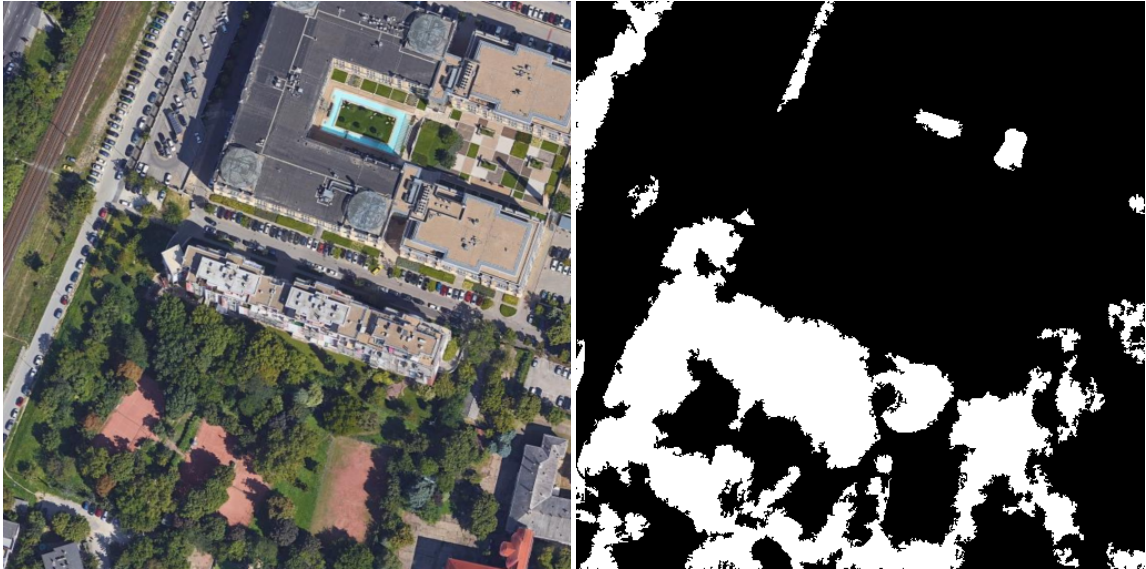
A képen jól látható, hogy a generálás egész pontosan hajtódott végre. A bal oldali képen látható nagyobb folyó jól kivehető a 3 dimenziós szerkesztő programban megjelenített jobb oldali képen. A képen látható élek a megjelenítő sajátosságai, nem pedig hiba (jobb futás érdekében kevesebb poligont használ). Ezt jól mutatja a következő kép ami már egy játék grafikus motorja segítségével készült, nevezetesen az „Arma Reforger”-ben.



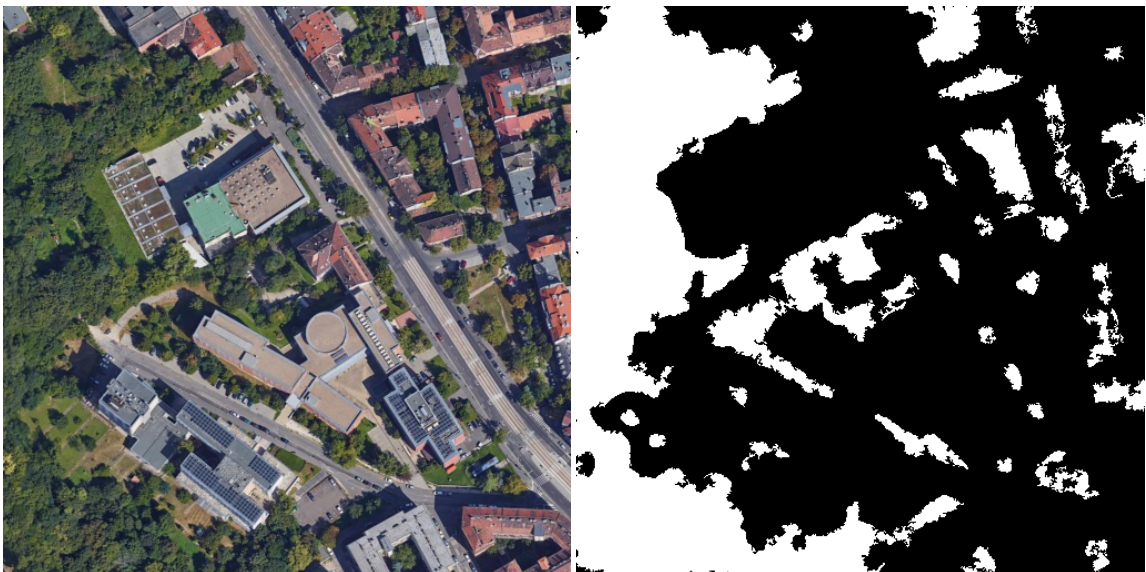
8.2. ábra: 3D terület részlete Arma Reforger grafikus motorban

8.2 Növények detektálása

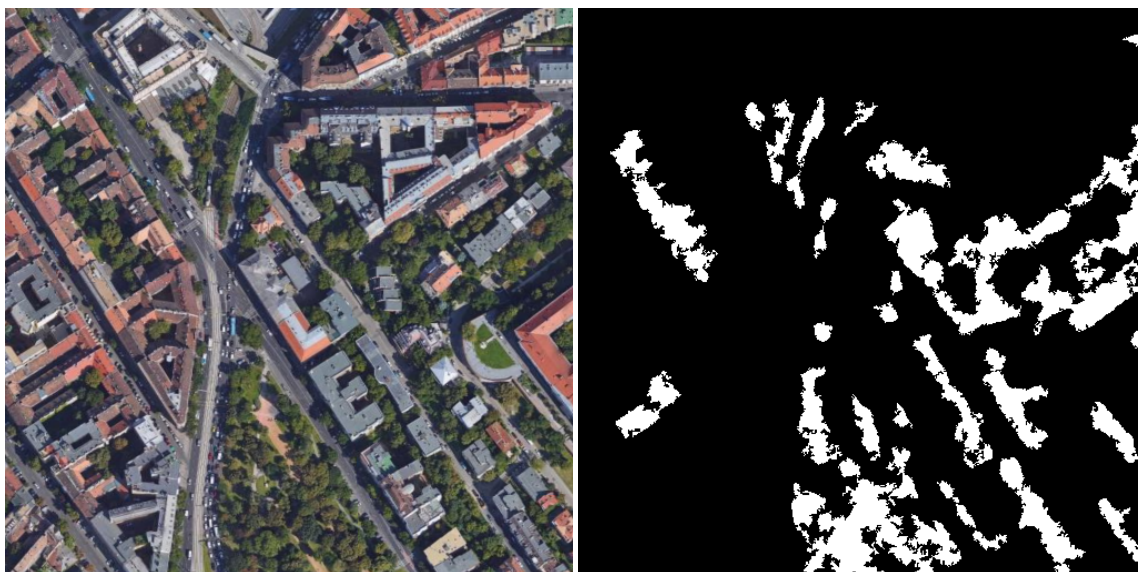
Minden kép esetén bal oldalt található a bemenet és jobb oldat pedig a modell által adott kimenet. Fekete színnel jelzi a "famentes" területeket és fehérrel a fákkal borított részeket.



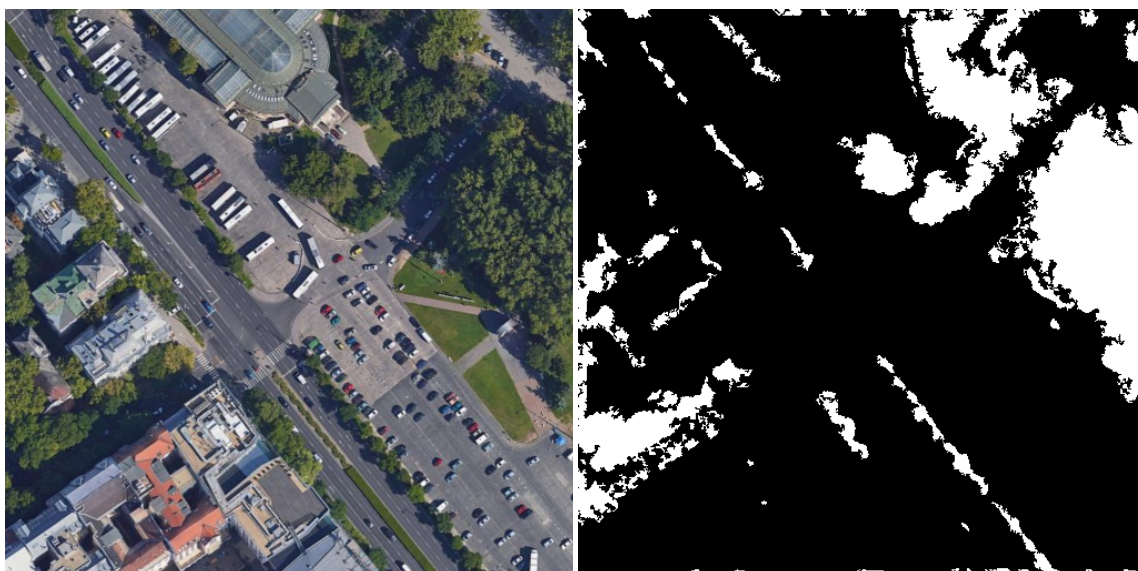
8.3. ábra: Buda, Dorottya udvar és környéke



8.4. ábra: Óbudai Egyetem NIK főépülete



8.5. ábra: Budai belvárosi környék



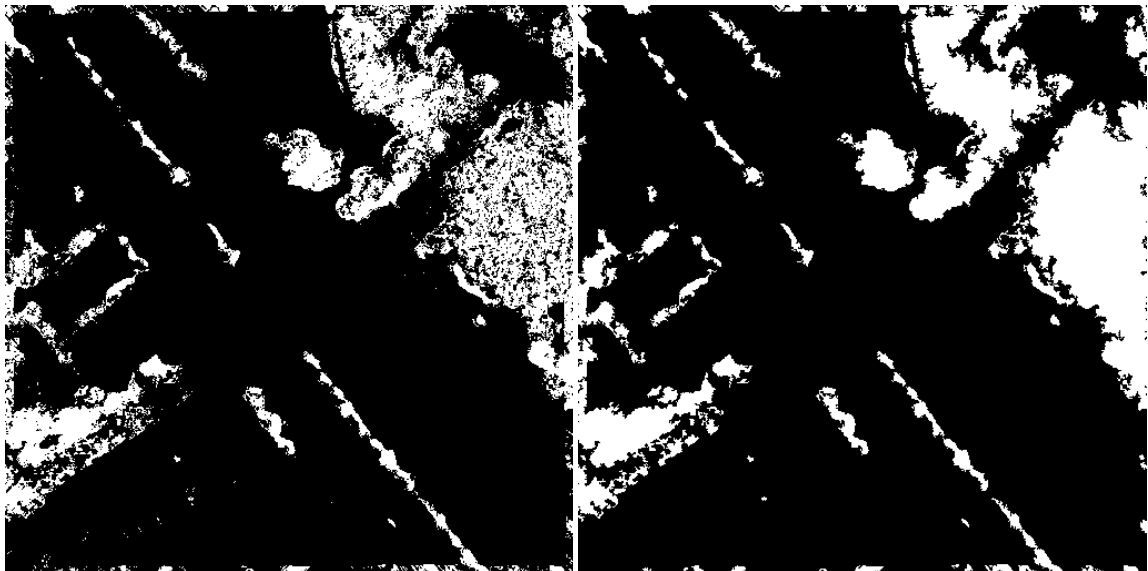
8.6. ábra: Pesti "parkosabb" környezet

A fenti bemutató képeken jól látható, hogy a modell egész kellemesen megtalálja a kívánt növényzetet bonyolult városi közegben. Ami a modellhez tartozó metrikákat, illeti az alábbi táblázatban megtalálható:

Pontosság	AUC	F1 Score	Tanítási idő
0.80	0.78	0.75	4 perc

8.1. táblázat: FastTree tanítási eredmények

Fontosnak tartom megemlíteni a metrikák alapján, hogy az átlag 80%-os pontosság a valóságban jobb, ahogyan az a képeken is látszik. Ennek a következő egyszerű oka van, fák detektálásakor nem szimpla klasszifikációt végzünk, mint például arcfelismeréskor hanem pixelenkénti avagy pixelszintű klasszifikációt hajtunk végre. Emiatt itt a pixelenkénti hibaarány torzítja a modell számtani eredményeit. Lássuk be, hogy például egy 512x512-es képen, ha akár több száz/ezer pixel össze vissza hibás akkor az nagyban befolyásolja a metrikák értékeit. Ez a valóságban a kimeneti képen viszont csak szimpla zajként jelentkezik, amit egyszerű, de annál hatékonyabb módszerekkel szüntethetünk meg. A következő kép ezt szemlélteti:



8.7. ábra: Zaj megszüntetése a kimeneti képen (balra a zajos, jobbra a szűrt kép)

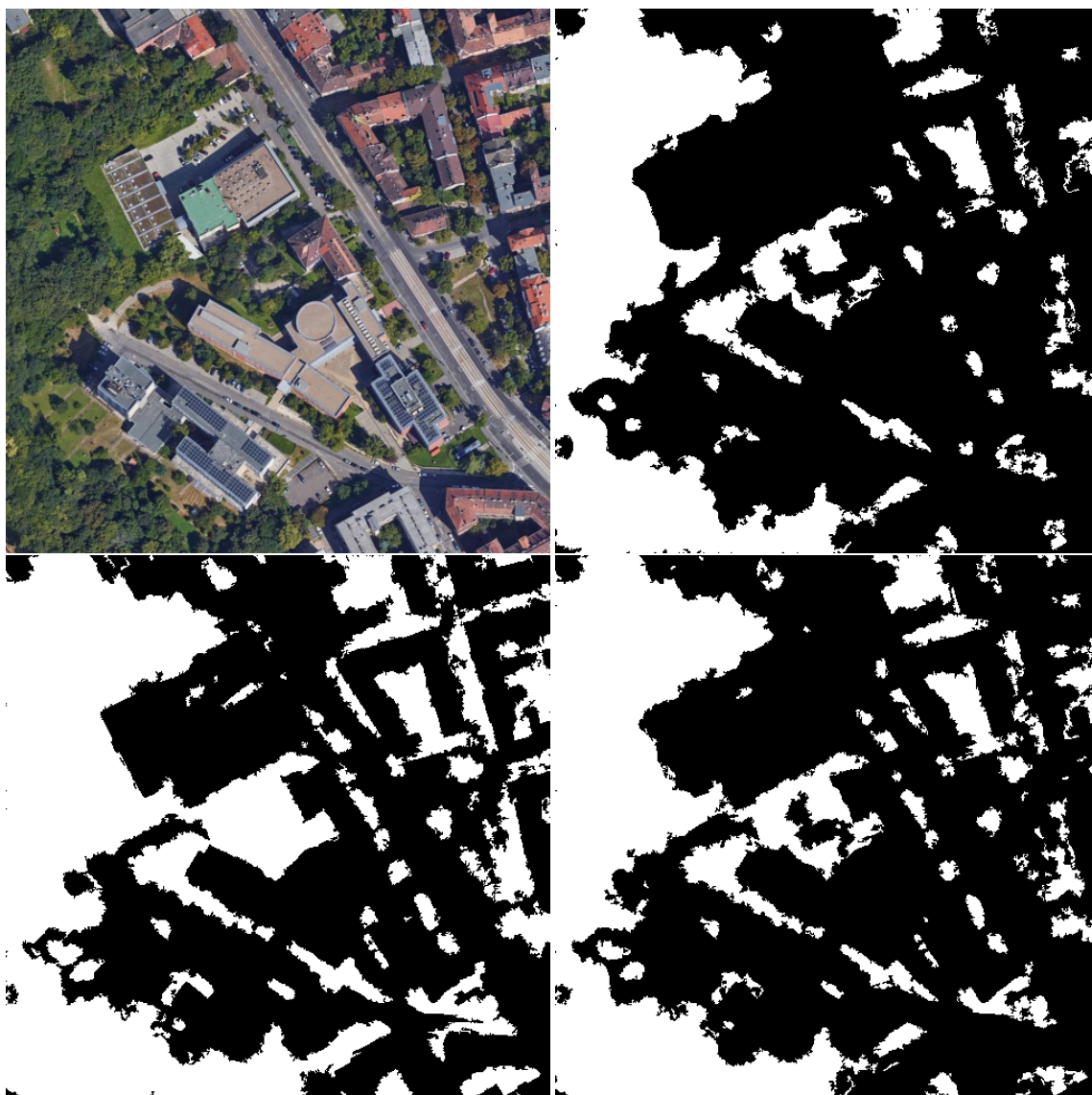
Jól látszik a bal oldali képen, hogy főként a fák között jelentkezik a modell hibáiból fakadó zaj. Zajszűrés után viszont már semmi probléma a kimeneti képpel. Ez a hiba annak tudható be, hogy egyes pixelek a fák között szimplán feketék vagy nagyon sötétek és a modell egyszerűen nem fának betudható pixelnek jelöli.

8.3 Összehasonlítás

Az összehasonlítást két modell bevonásának segítségével végeztem, az egyik modell az általam korábban használt LightGBM féle modell a másik pedig egy AdaBoost modell, ami egy Python „DetectTree” nevű könyvtárhoz köthető. Emiatt az AdaBoost modell esetében csak a dokumentációban megadott értékekre tudok hivatkozni, illetve amit a könyvtár használata alatt tapasztaltam.

Modell	Pontosság	AUC	F1 Score	Tanítási idő	Detektálási idő
FastTree	80%	0.78	0.75	4 perc	10 másodperc
LightGBM	72 – 75%	0.68	0.60	3 perc	10-15 másodperc
AdaBoost	85%	-	-	30 perc	15 másodperc

8.2. táblázat: Tanítási eredmények összehasonlítása különböző modelleken

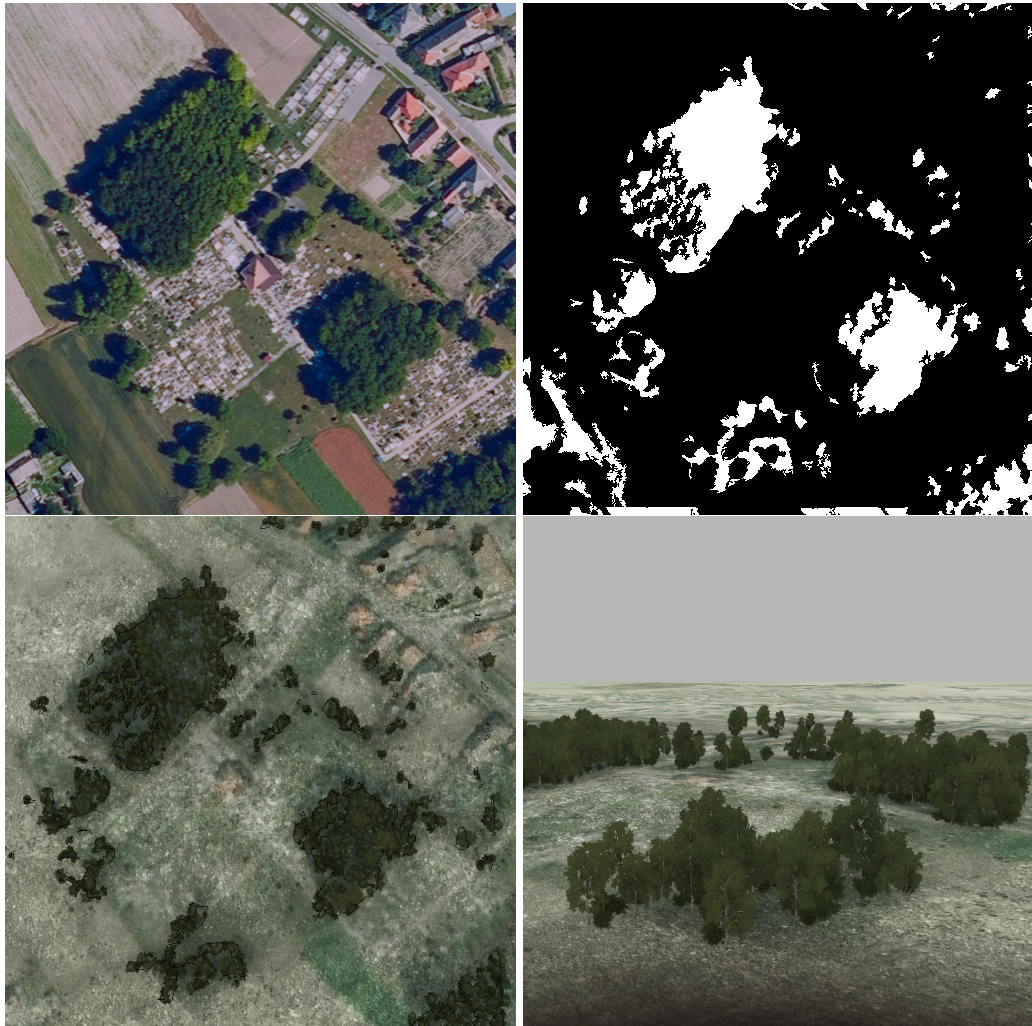


8.8. ábra: Összehasonlítás: 1.Kép a bemenet 2.FasTree kimenete 3.LightGBM kiemnete 4.Adaboost kimenete

Itt is jól látszik, amit már korábban is említettem, hogy a pontosság értékei itt máshogy verődnek vissza. Az AdaBoost (4.kép) és a FasTree (2.kép) közötti különbség szinte elenyésző, vannak részei a képnek, ahol az egyik van, ahol a másik teljesít jobban, de összességében az AdaBoost egy picivel jobb vagy legalábbis több helyen magabiztosabb. A LightGBM esete már egy picit más. Jól látszik, hogy sok helyen fals pozitívot mutat, ami probléma. Hiába a körülbelül 75 százalékos pontosság, ha sokszor a pozitívokkal hibázik. Ez volt ez egyik oka annak, hogy FastTree modellre váltottam.

8.4 Felhasználás

Miután megvan a háromdimenziós területünk ahová a fákat szeretnénk leképezni és megvan a fák pozíciói is, amit a modell adott vissza, akkor ezt a két adathalmazt „vegyítve” egy grafikus motorban tökéletesen vissza lehet adni mind a területet, mind a növényzetet is. Ez felhasználható szimulációkban, játékokban vagy akár a geoinformatika területein.



8.9. ábra: Felhasználhatóság: felül a valós terület alul pedig a szimulált

8.5 Tapasztalatok

A terület generálásakor különösebb problémába nem ütköztem, amit tapasztaltam, hogy a képminősége nagyon fontos, illetve maga kép mindenképp szürkeárnyaltos kell, hogy legyen különben nem lesz hatékony a feldolgozás. Emiatt is alakítom át a beolvasáskor, mert így nem jön elő ez a probléma. Amit még fontosnak tartok megemlíteni az a Get-Pixel() függvény. Célszerű elkerülni a használatát, ha fontos a „gyors” futásidő. Eleinte ezzel próbálkoztam és csak a beolvasás perceken át tartott egy 4096x4096 méretű kép esetében. A fentebb már részletezett megoldásom pedig 20-25 másodperc alatt végez az egész generálással.

A növények detektálása már egy szinttel nehezebb volt. A modell kiválasztásától egészen a megvalósításig. Ugyanis a 27 érték ellenére is vannak objektumok, amik nagyon hasonlóak a növényekhez műholdképről nézve, főleg, ha az adott kép minősége nem a legjobb. Emiatt próbálkoztam előre súlyozással, ami annyit takar, hogy mikor beolvasom a tanító mintát, csoportokba osztottam az LaB, Ill, Textúra és az Entrópia értékeket. Ezt követően különböző súlyokat adtam nekik mind addig míg egy egész jó eredményt nem ért el. Ennek köszönhetően sikerült javítani a pontosságon, de nem eléggé (ezt az ötletet a későbbiek során eldobtam mert inkább rontott mintsem javított a modellen). Emiatt is váltottam a már fentebb bemutatott FastTree modellre. Mindemellett pedig a 6 darab tanító képet kibővítettem 10 darabra. Ez a plusz 4 kép tartalmaz extrémebb eseteket mikor például vagy nagyon sok különböző, vagy épp nagyon kevés fa található. A képminőség természetesen itt is sokat jelentett, illetve az is, hogy milyen magasságból készült. Észrevételeim szerint olyan 90-100 méteres magasság a legoptimálisabb. Magasabbról már nagyon összefolynak a tárgyak, közelebből pedig már túl részletes a kép.

9 TERVEK A JÖVŐRE NÉZVE

A jövőben szeretném kicsit pontosítani a megoldást, illetve továbbfejleszteni. A pontosság és a stabilitás mellett pedig szeretném új funkciókkal kibővíteni a programot. Jó lenne, ha nem csak egy képet adna vissza a fák pozíciójáról, hanem valami pontosabb adatot, például koordináta értékek segítségével. Ebből pedig olyan fájlok lehetnének ki-generálhatóak, amik egész pontosan helyezik el a növényzetet. Mindezek mellett a terület generálást sem elhanyagolva hasznos lenne, ha lenne valamilyen megjelenítő, ami előnézet szerűen bemutatja, hogy a generálni kívánt terület miként nézne ki 3D-ben. Ezek után pedig a későbbiekben szívesen felvenném a kapcsolatot a „Bohemia Interactive” fejlesztő stúdióval, hogy az általam fejlesztett rendszert valahogy integrálhassuk a pálya szerkesztő szoftverjükbe, ez által segítve a különböző harmadik féltől származó úgynevezett „modok” elkészítését.

10 ÖSSZEGZÉS

A fentieket összefoglalva, jól látszik, hogy a dolgozat elején felvázolt tervek megvalósíthatóak. Első sorban több algoritmus a végig elemezve kiválasztásra kerültek azok melyek a későbbiekben felhasználásra kerültek. A területek generálása nem volt nehéz folyamat, hisz a bemeneti paraméterek alapján könnyen lehet számolni az adott pixelhez tartozó magasságot, majd ezt követően az összes többire is. Az ebből kapott .asc állomány pedig már importálható arra alkalmas alkalmazásokba, illetve könnyen bővíthető akár más állományokra is, hisz az .asc fájl jól konvertálható akár .obj vagy .3ds fájlá is.

A fák detektálása már jóval nehezebb problémát vetett fel. Ez esetben is több erre alkalmas algoritmust részletez a dolgozat, majd ezt követően bemutatja a választott megoldást és indokolja is azt. A választott algoritmus gépi tanuláson alapul, ezért ehhez kisebb-nagyobb előkészületekre volt szükség, mint például a tanítóminták megszerzése és azokból a „feature” illetve „label” értékek kinyerése. Ezt követően jött a tanítása, ahol a nagyobb kihívás a megfelelő paraméterek megadása volt. A tanító algoritmus konfigurálása után következett a tesztelés és az eredmények értékelése, ahol több modellel is összehasonlításra került. Természetesen minden modell ugyan azon az adathalmazon tanult, és ugyan azon a teszt adathalmazon is lett kiértékelve.

Összeségében én teljesen elégedett vagyok a kapott eredményekkel. Véleményem szerint sikerült egy olyan szoftvert elkészíteni, ami a mai kornak megfelelő technológiákon alapszik, és kellő szakmai körültekintés mellett valósult meg. Mint ahogyan az a dolgozatban említésre került, a generált területek szépen reprezentálják a valós bemeneti területet, és a detektált fák is túlnyomó részt megfelelő helyre kerülnek.

11 IRODALOMJEGYZÉK

- [1] W. M. S. LLC. „World Machine Features”. (2022), cím: <https://www.world-machine.com/features.php> (elérés dátuma 2023. 10. 20.).
- [2] A. Torpy. „Large 3D terrain generator”. (2023), cím: <https://www.bundysoft.com/L3DT/> (elérés dátuma 2023. 10. 20.).
- [3] E. Games. „Unreal Engine 5”. (2023), cím: <https://www.unrealengine.com/en-US/unreal-engine-5> (elérés dátuma 2023. 10. 20.).
- [4] U. Technologies. „Unity Engine”. (2023), cím: <https://unity.com/products/unity-engine> (elérés dátuma 2023. 10. 20.).
- [5] T. K. Heok és D. Daman, „A review on level of detail”, *Proceedings. International Conference on Computer Graphics, Imaging and Visualization, 2004. CGIV 2004.*, IEEE, 2004, 1–6. old. DOI: 10.1109/CGIV.2004.1323963.
- [6] B. Chen, J. E. S. II, E. Kuo és A. Kaufman, „LOD-Sprite Technique for Accelerated Terrain Rendering”, *Proceedings Visualization '99*, IEEE, 1999, 1–7. old. DOI: 10.1109/VISUAL.1999.809900.
- [7] M. Fan, M. Tang és J. Dong, „A Review of Real-time Terrain Rendering Techniques”, 8. sz., 685–691. old., 2004.
- [8] K. McHenry és P. Bajcsy, „An Overview of 3D Data Content, File Formats and Viewers”, 2008.
- [9] M. Lundgren, „A comparison of 3D file formats”, dipl., Linköping Institute of Technology, 2011.
- [10] D. Hu, S. Ma, F. Guo, G. Lu és J. Liu, „Describing data formats of geographical models”, *Environ Earth Sci*, Springer-Verlag Berlin Heidelberg, 2015, 7101–7115. old. DOI: 10.1007/s12665-015-4737-4.
- [11] W. Dijak, „Landscape Builder: Software for the creation of initial landscapes for LANDIS from FIA data”, *International Academy of Ecology and Environmental Science*, 17–25. old., 2013.
- [12] P. Ongsulee, „Artificial Intelligence, Machine Learning and Deep Learning”, IEEE, 2017. DOI: 10.1109/ICTKE.2017.8259629.
- [13] S. A. PhD és G. S. Babu, „Satellite Image Classification Methods”, *International Journal of Computer Applications*, 8. évf., 9. sz., 20–25. old., 2015.
- [14] M. A. Hearst, S. Dumais, E. Osuna, J. C. Platt és B. Schölkopf, „Support vector machines”, *IEEE Intelligent Systems and their Applications*, IEEE, 1998, 18–28. old. DOI: 10.1109/5254.708428.

- [15] H. Bhavsar és M. H. Panchal, „A Review on Support Vector Machine for Data Classification”, *International Journal of Advanced Research in Computer Engineering and Technology*, 1. évf., 10. sz., 185–189. old., 2012, ISSN: 2278-1323.
- [16] L. A. C., J. L. F.O., S. M. F. és tsai., „Comparing machine learning classifiers in potential distribution modelling”, *Expert Systems with Applications*, 38. sz., 5268–5275. old., 2011.
- [17] M. Zounemat-Kermani, O. Batelaan, M. Fadaee és R. Hinkelmann, „Ensemble machine learning paradigms in hydrology”, *Journal of Hydrology*, 598. évf., 1–13. old., 2021.
- [18] Microsoft. „LightGBM Features”. (2023. szept. 12.), cím: <https://lightgbm.readthedocs.io/en/latest/Features.html> (elérés dátuma 2023. 10. 20.).
- [19] K. V. Rashmi és R. Gilad-Bachrach, „Fast Decision Tree Algorithm”, *Advances in Electrical and Computer Engineering*, 14. évf., 1. sz., 65–68. old., 2014.
- [20] Microsoft. „FastTree Binary Trainer”. (2020. febr. 1.), cím: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.fasttree.fasttreebinarytrainer?view=ml-dotnet> (elérés dátuma 2023. 10. 25.).
- [21] W. Zhang, H. Li, Y. Li, H. Liu, Y. Chen és X. Ding, „Application of deep learning algorithms in geotechnical engineering: a short critical review”, *Artificial Intelligence Review*, Springer-Verlag Berlin Heidelberg, 2021, 5635–5673. old. DOI: 10.1007/s10462-021-09967-1.
- [22] N. A. Ibraheem, M. M. Hasan, R. Z. Khan és P. K. Mishra, „Understanding Color Models: A Review”, *ARPN Journal of Science and Technology*, 2. évf., 3. sz., 265–275. old., 2011-2012, ISSN: 2225-7217.
- [23] M. O. Al-Dwairi, Z. A. Alqadi, A. A. AbuJazar és R. A. Zneit, „Optimized True-Color Image Processing”, *World Applied Sciences Journal*, IDOSI Publications, 2010, 1175–1182. old. DOI: 10.1007/s10462-021-09967-1.
- [24] P. C. Saravanan, „Color Image to Grayscale Image Conversion”, *Second International Conference on Computer Engineering and Applications*, IEEE, 2010, 196–199. old. DOI: 10.1109/ICCEA.2010.192.
- [25] A. Wetzler, G. Rosman és R. Kimmel, „Patch-space Beltrami denoising of 3D point clouds”, *Patch-space Beltrami denoising of 3D point clouds*, IEEE, 2012, 1–5. old., ISBN: 978-1-4673-4681-8.
- [26] L. Yang, X. Wu, E. Praun és X. Ma, „Tree detection from aerial imagery”, *Tree detection from aerial imagery*, Association for Computing Machinery, 2009, 131–137. old. DOI: 10.1145/1653771.1653792.

12 ÁBRAJEGYZÉK

2.1. .obj Fájl felépítés	9
4.1. A CIE RGB alapszínek színskálája és az alapszínek elhelyezkedése a CIE 1931 xy színskáladiagramon.	28
7.1. Kép beolvasása	37
7.2. ASC fájl (részlet)	38
7.3. ASC exportálás	39
7.4. Szín értékek számítása	40
7.5. Textúra értékek számítása	42
7.6. Entrópia értékek számítása	44
7.7. Minták: balra a műholdkép, jobbra pedig a jelölt növényzet	46
7.8. Label értékek számítása	46
7.9. Tanítási paraméterek	47
8.1. Balra a bementi kép, jobbra pedig a 3D területből egy részlet	50
8.2. 3D terület részlete Arma Reforger grafikus motorban	50
8.3. Buda, Dorottya udvar és környéke	51
8.4. Óbudai Egyetem NIK főépülete	51
8.5. Budai belvárosi környék	52
8.6. Pesti "parkosabb" környezet	52
8.7. Zaj megszüntetése a kimeneti képen (balra a zajos, jobbra a szűrt kép) . .	53
8.8. Összehasonlítás: 1.Kép a bemenet 2.FasTree kimenete 3.LightGBM ki- emnete 4.Adaboost kimenete	54
8.9. Felhasználhatóság: felül a valós terület alul pedig a szimulált	55