Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Zsámboki Richárd

TDK DOLGOZAT

Városi Lidar pontfelhő objektumok mély tanulás alapú osztályozása

KONZULENS

Dr. Benedek Csaba

Börcs Attila

BUDAPEST, 2016

Tartalomjegyzék

1 Bevezetés

A mesterséges intelligencia alapú módszerek számos mérnöki területen töretlen lendülettel terjeszkednek. Kiemelkedő fontosságú alkalmazási lehetőségük az objektumfelismerő rendszerekben történő alkalmazásuk. Az utóbbi 1-2 évtizedben hatalmas fejlődésen ment keresztül ez a tudományterület, a folyamatos haladást motiválja egyrészt a széleskörű felhasználhatóság, másrészt az informatikai hardverek nagy léptékű fejlődése, mely korábban megvalósíthatatlannak tartott módszerek implementálását tette lehetővé.

Napjaink egyik kiemelten fejlesztett technológiája az autonóm autó. Számos gyártó dolgozik különböző elgondolásokon, különböző automatizálási szinteken. Több komponens közül azonban legalább egy közös, az egyszerű sávtartó rendszertől kezdve a teljesen vezető nélküli közlekedést megvalósító komplex algoritmusig szükség van objektumfelismerést végző rendszerekre. Szenzorok tekintetében radarokat, kamerákat, ultrahangos eszközöket alkalmaznak általában, egyszerre többet is a biztos működés miatt. A legújabb, talán legösszetettebb képalkotó eszköz, amelyet autók a környezetük feltérképezésére használhatnak, a LIDAR szenzor, ami a környezet háromdimenziós leképezését végzi el.

A két említett terület, azaz a mesterséges intelligencia alapú algoritmusok, és az autók objektumfelismerő rendszereinek találkozása ígéretes, régóta fejlesztett módszer. Bevett szokás, hogy egy tanuló algoritmust még a gyártás előtt betanítanak, és már egy végleges verziót telepítenek az elkészült autókba. Fontos fejlesztési irány a jelenleg elérhető felismerő rendszerek számának növelése, a stabilan működő alacsonyabb szintű algoritmusok magasabb szintűekkel történő támogatása. Egy konkrét elképzelés a már említett LIDAR szenzorokon alapuló komplex objektum osztályozó rendszer megvalósítása.

A nyers háromdimenziós adatok egyik hátránya, hogy a hagyományos mesterséges intelligencia alapú algoritmusok velük történő használata, bár vannak működő eljárások, még nincs kiforrott megoldás. Bevett módszer, hogy az adatokat csak előfeldolgozás után adják oda ezeknek az algoritmusoknak. Egy hasonló elven működő objektumfelismerő rendszert valósítottam meg projektem előző szakaszában. A gépi tanulás új eredményei azonban lehetőséget nyújtanak az adatok nyers feldolgozására.

A mesterséges intelligencia kutatás egyik jelenleg legfejlettebb technológiája a mély tanulás. A módszer a neurális hálók elméletére épít, de jóval túlmutat azon, ez az eddig elért eredményeken is megmutatkozik. A mély tanulás egy remek jelölt a LIDAR szenzorokból származó nyers háromdimenziós információkra alapozott objektumfelismerő rendszer központi algoritmusára. Többek között az ACFR (Australian Center for Field Robotics) kutatói is kidolgoztak egy mély tanulás alapú városi LIDAR pontfelhőkön objektumokat osztályozni képes rendszert [1]. A kutatásuk során összegyűjtött felcímkézett háromdimenziós objektumokat elérhetővé tették, ezek képezik munkám során a teszteléshez használt adathalmazt.

Jelen dolgozatban a szükséges technológiai elvek és a rendszer fejlesztése során használt szoftverkönyvtárak bemutatása után vázolom az általam megtervezett objektumfelismerő algoritmus működésének lényegét. Tárgyalom a konkrét implementációs lépéseket és prezentálom az eddigi eredményeket.

2 Irodalomkutatás

A kitűzött célok eléréséhez számos kevésbé ismert technológia és elmélet megértése volt szükséges. A következőkben a legfontosabbakat röviden tárgyalom.

2.1 Pontfelhők

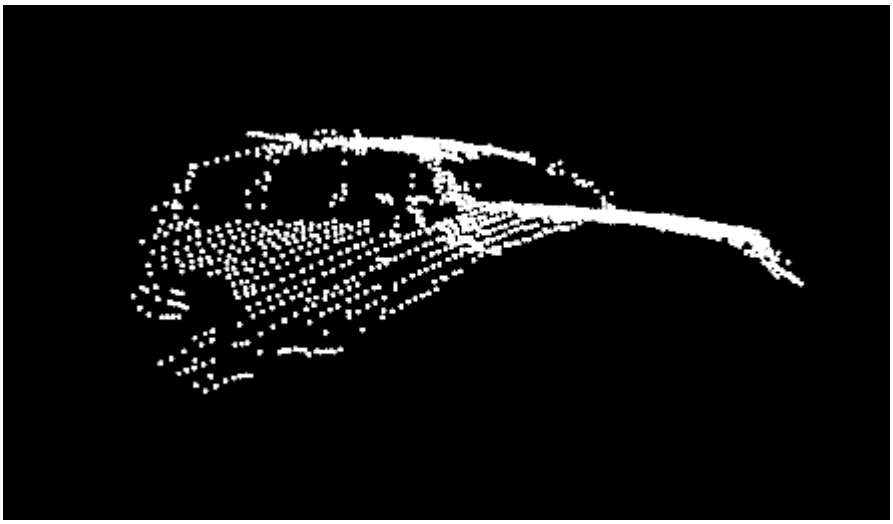
Több modern látórendszerekkel kapcsolatos technológia fejlesztése során megjelent az igény az informatikában környezetünk közvetlenebb leképezését reprezentáló adatstruktúrára, a pontfelhőkre. Ezek többdimenziós pontok halmazai, ahol a pontokhoz az x, y, z, térbeli Descartes-koordináták mellett hozzárendelhetünk szín- vagy intenzitás információt [2].

A több dimenziós információ kezelésének, feldolgozásának hatékonysága kritikus fontosságú. Pontfelhőkkel történő műveleteknél figyelembe kell venni, hogy az ilyen típusú fájlok mérete miatt már komoly számítási kapacitásra van szükség. Az alkalmazások fejlesztésében sokat segítenek már létező programozási segédkönyvtárak.

A pontfelhők közvetlen megjelenítésére és feldolgozására számos lehetőség adódik. Jellemzően a pontfelhőket átalakítják poligon (általában háromszög) hálóvá, NURBS felületmodellé vagy CAD modellé egy úgynevezett felület rekonstrukciós eljárás segítségével. A pontfelhők háromdimenziós felületté alakításának számos technikája létezik. Egyes megoldások háromszögeket feszítenek a pontfelhők meglévő pontjaira [3], mások távolság térképpé alakítják a pontfelhőt, majd ebből a térképből hozzák létre a felületeket.

A pontfelhők tárolására számos fájlformátum létezik, nincs egy általánosan használt megoldás. A jelenleg elérhető formátumokat is főleg a számítógépes grafikával foglalkozó közösség hozta létre, gyakran kifejezetten az adott kor és felhasználási cél igényeire szabva [4]. A teljesség igénye nélkül a legelterjedtebbek ezek közül: a PLY poligon alapú formátum, amit a Stanford egyetemen fejlesztettek [5], a sztereolitográfiában használt STL, a geometriát leíró OBJ, valamint az ISO sztenderd XML alapú X3D fájlformátum.

A munkám során egy viszonylag új formátum, a PCD (Point Cloud Data) mellett döntöttem. A választás alapja a használt pontfelhőkezelő szoftverkönyvtár, amely ezt a pontfelhő reprezentációt kéri metódusaiban. Egy PCD fájl egy fejlécből és a pontfelhő pontjait leíró adatsorokból áll. A fejléc határozza meg a pontok lehetséges mezőit (intenzitás, szín stb.), a pontfelhő méreteit és pontjainak számát és egyéb jellemzőket. Az adatok leírására jelenleg két adattípus is támogatott, az ASCII és a bináris. Megalkotásának fő motivációja az volt, hogy így az őt használó szoftver metódusaihoz nem kellett átalakító lépéseket írni, a saját fájlformátumot kihasználva gyors és effektív algoritmusokat tudtak írni. Ezen kívül a PCD fájlok rugalmasok, tudnak tárolni rendezett pontfelhőket, különböző adattípusokat [4]. Egy, a munkám során használt, PCD fájlban eltárolt pontfelhő megjelenítése látható a 2.1. ábraán.



2.1. ábra: Egy PCD formátumú pontfelhő megjelenítve

Érdemes említést tenni a legelterjedtebb, pontfelhők létrehozásánál használt képalkotó szenzorokról. Talán a legismertebb, sokak által használt eszköz a Microsoft Kinect eszköze, amelyben egy RGB videokamera és egy mélységszenzor működik együtt. Előnye, hogy a nagyközönség számára elérhető, kutatási projektekben is könnyen felhasználható. Kiegészítő eszközökkel, szoftverekkel komplexebb leképezési feladatokra is alkalmassá tehető. Egy másik elterjedt szenzorcsoport a Time of Flight (ToF) kamerák, melyek aktív fényforrással rendelkeznek, és az objektumról vagy felületről visszaverődő fény utazási idejéből számítják ki az objektum pontjainak távolságát. Részletes tárgyalásra kerül a következő fejezetben a LIDAR technológia.

2.2 LIDAR technológia

A LIDAR (Light Detection and Ranging) az 1960-as évek elején kifejlesztett technológia, melyben egyesül a lézer pontos fókuszálhatósága a radar távolság-meghatározó képességével [6]. A működés alapja az eszközről kibocsátott lézernyalábok detektálása az objektumokról való visszaverődés után. A ToF eszközökhöz hasonlóan itt is az utazási időből számíthatók a távolságok.

Egy LIDAR eszköz jellemzően egy lézerből, egy szkennerből, fotodetektorból és egy speciális GPS vevőből áll. Amikor egy lézer a vizsgálandó területre mutat, a fénysugarat az útjába kerülő felület visszaveri. A lézer mellett elhelyezett szenzor ezt a visszavert fényt érzékeli és így méri a távolságot. A szkenner dolga a lézert és a hozzá tartozó optikát úgy mozgatni, hogy a célnak megfelelően fel tudják térképezni a környezetet. Amikor a lézeres távolságméréseket fúzionálják a GPS és IMU (inerciális mérőegység) által szolgáltatott pozíció és orientáció adatokkal, az eredmény egy sűrű, részletekben gazdag pontfelhő [7].

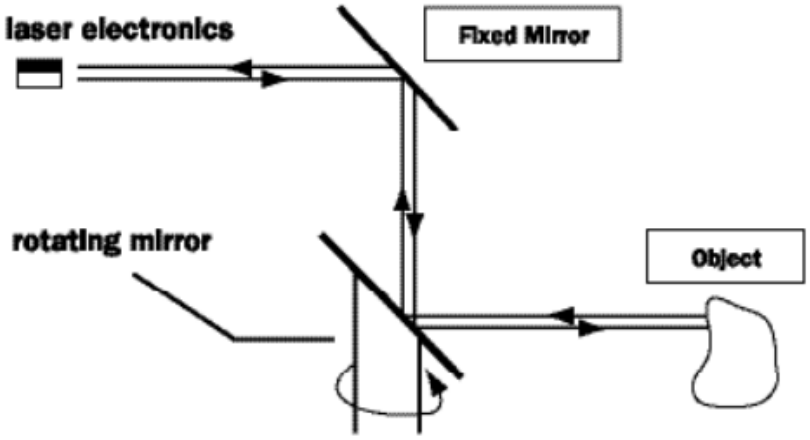
A LIDAR eszközöknek számos fő típusa létezik [8] alapján. Leggyakoribb az előzőekben is leírt visszaverődés alapú LIDAR, ami a visszavert fénysugár érzékeléséből számítja a felület távolságát. A DIAL (DIfferential Absorption Lidar – differenciális abszorbciós LIDAR) kémiai elemek koncentrációjának mérésére használható. A Raman LIDAR a Raman-szórás (rugalmatlan fotonszóródás egy esete) jelenségét kihasználva detektálja a választott molekulát. Magas spektrális felbontású LIDAR (HSRL – High Spectral Resolution LIDAR) alkalmazható gázok érzékelésére az atmoszférában, az utóbbi optikai jellemzőinek mérésén keresztül. Végül, de nem utolsó sorban a Doppler Lidar segítségével a célpont sebessége mérhető a Doppler effektust kihasználva.

A LIDAR első dokumentált alkalmazása a meteorológiában történt felhők mérésére [9]. A pontossága és hasznossága 1971-ben, az Apollo 15 küldetés alatt vált közismertté, amikor is ezzel a technológiával térképezték fel a hold felületét. Azóta számos alkalmazását ismerjük, melyetek két fő csoportra lehet osztani, légiekre és földiekre [10]. Előző esetben egy légi jármű, például drón térképezi fel az alatta elterülő tájat, utóbbi esetben a LIDAR szenzor egy földi járművön helyezkedik el. Használnak LIDAR-t a mezőgazdaságban, a régészetben, geológiában, meteorológiában, űrkutatásban, robotikában, bűnüldözésben (gyorshajtás érzékelése [11]), és feltételezhetően a haditechnikában is, bár innen kevés a nyilvánosságra is hozott példa.

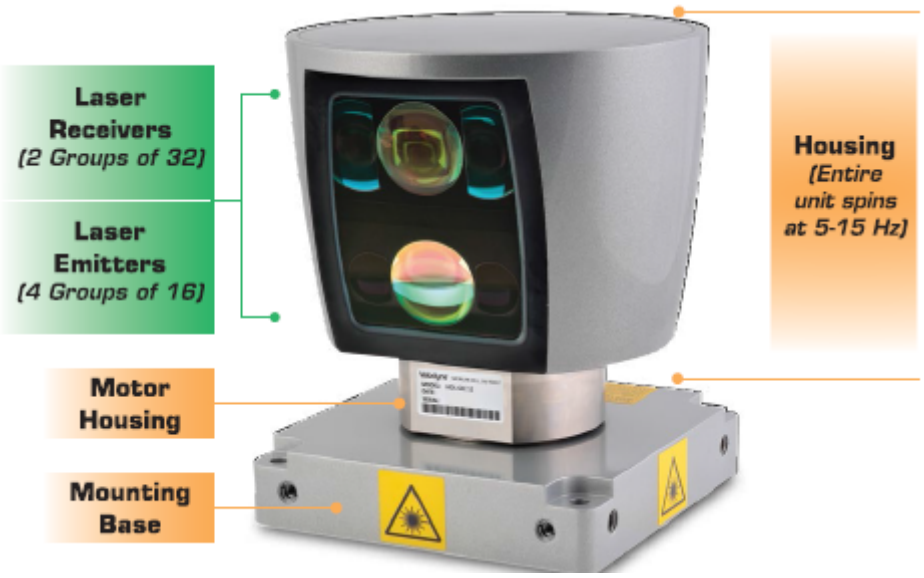
A projektem során LIDAR technológia segítségével rögzített pontfelhők adatait használtam fel. Ezeket korábban az ACFR munkatársai egy Velodyne HDL-64E LIDAR szenzorral készítették. A projekt egy korábbi szakaszában dolgoztam az MTA-SZTAKI által készített felvételeken is, az azokat készítő eszközt mutatja a 2.2. ábra.

2.2. ábra: Az MTA SZTAKI Velodyne szenzora

Az eszköz 360°-os horizontális és 26,8°-os vertikális látószöggel rendelkezik, 5-15 Hz-es szkennelési sebességre képes és 1,3 millió pontot rögzít másodpercenként [12]. A szenzor felépítését szemlélteti a 2.4. ábra. A szenzor fején 64-64 db lézer adó és vevő egység található, így minden körbefordulással egy teljes értékű háromdimenziós szkennelést hajt végre. Az alkalmazott forgótükrös konstrukció működéséről készült sematikus rajzot láthatjuk a 2.3. ábraán.

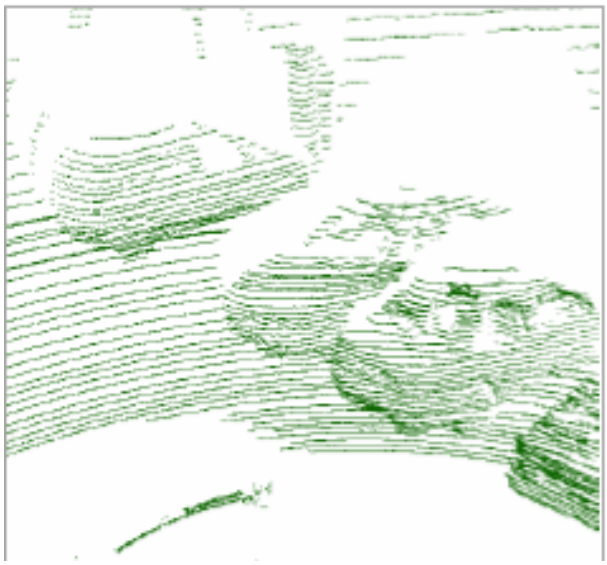


2.3. ábra: Egy emitter/detektor párból álló forgótükrös LIDAR működése [13]



2.4. ábra: Velodyne HDL-64E Lidar szenzor felépítése [12]

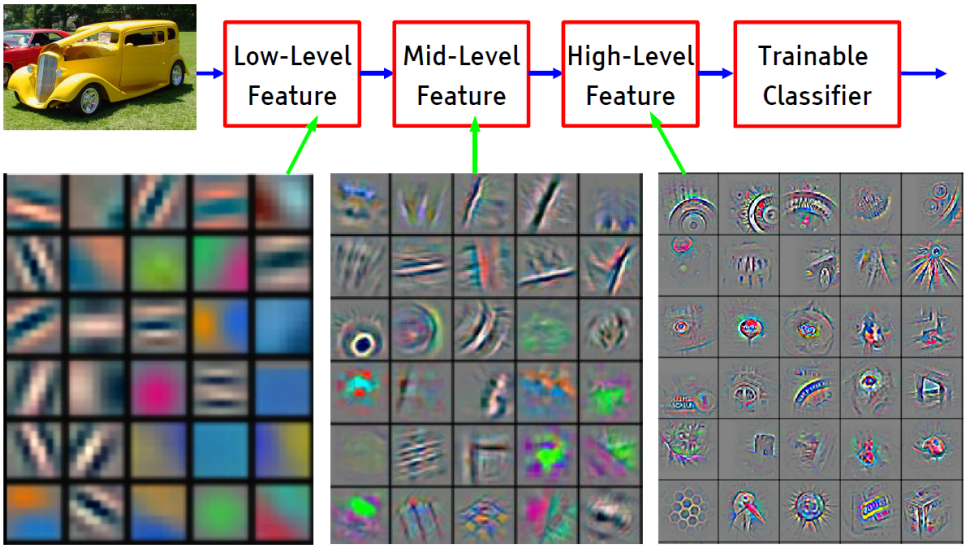
A szenzort maximális forgatási teljesítményen használva 60-100 ezer pontot tartalmazó pontfelhőkből álló idősorozatot hoz létre. Ezzel a sebességgel már lehetséges a környezet dinamikai változásainak háromdimenziós megfigyelése. Ennél a nagy sebességnél is pontos (< 2cm) pontfelhőt képes előállítani, példát szemléltet a 2.5. ábra. Az eszköz hátránya az hagyományosan magas ára, azonban már elérhetőek olcsóbb kompaktabb változatok is. A leképezés pontossága, valamint a mérés fényviszonyokkal szembeni invarianciája miatt az eredmények megbízhatósága elég magas ahhoz, hogy megérje használni.



2.5. ábra: Velodyne HDL-64E által alkotott pontfelhő-részlet

2.3 Mély tanulás

A mintafelismerés tradicionális modellje az 50-es évek óta a fix, jól meghatározott sajátosságok kinyerése és ezekre a sajátosságokra ráállított tanítható osztályozó. A gond ezzel a módszerrel többek között az, hogy a sajátosság kinyerésére szolgáló eszközök, rendszerelemek megtervezése szakértelmet és sok erőforrást igényel [14]. Elméleti eredmények alapján ahhoz, hogy olyan bonyolult függvényeket tanítsunk meg gépekkel, melyek magas szintű absztrakciót reprezentálnak, mély struktúrákra van szükség [15]. Az ilyen architektúrák több szintű nemlineáris műveletekből állnak, úgynevezett „elejétől-végéig" tanulást valósítanak meg. Ez azt jelenti, hogy már maga a sajátosság kinyerő is tanítható, valamint az arra épülő osztályozó is. Egy képi objektum-felismerésben használt ilyen struktúrát vizualizál a 2.6. ábra. Az ilyen struktúrákat, architektúrákat alkalmazó gépi tanulási módszereket nevezzük összefoglaló néven mély tanulásnak.



2.6. ábra: Elejétől végéig tanulás sajátosságainak vizualizálása [14]

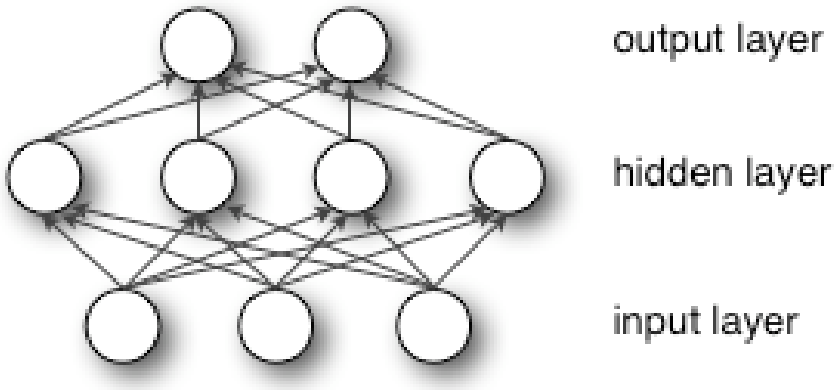
A mély tanulás lényege tehát, hogy több szintű reprezentációk, absztrakciók megtanulására képes, ezzel ismerve fel komplexebb mintákat, objektumokat képeken, hangokon és szövegeken. Több cikk is szól az elért jelentős eredményekről, viszont érdekesség, hogy a működésének sikerességét nehéz elméleti módszerekkel igazolni. Ez abban is megmutatkozik, hogy a cikkek egy része (például [16]) fel is hívja a figyelmet arra, hogy struktúrájuk bizonyos paramétereit ad hoc módszerekkel állították be.

A mély tanulás körébe számos architektúra tartozik. Ilyenek például a mély neurális hálók, vagy más néven többrétegű perceptronok, a konvolúciós neurális hálók, mély Boltzmann gépek, zajtalanító auto enkóderek stb.

2.3.1 Többrétegű perceptron

A többrétegű perceptron egy regressziós osztályozó, ahol a bemenetet egy megtanult nemlineáris transzformációval egy olyan térbe vetítjük, ahol már lineárisan szeparálható a különböző osztályokra. A leggyakrabban használt nemlinearitások a sigmoid és a tanh függvények, melyek közül [17] empirikusan azt figyelte meg, hogy az utóbbi jobb konvergencia tulajdonságokat mutat.

A bemeneti réteg neuronszáma az osztályozandó elemek méretétől, a kimeneti rétegé a megkülönböztetni kívánt osztályok számától függ. A közbenső rétegeket nevezzük rejtett rétegnek, melyből egy is elegendő egy univerzális approximátor készítéséhez (ezt a struktúrát ábrázolja az 2.7. ábra). A rejtett réteg(ek)ben található neuronok száma a rendszer egyik beállítandó paramétere. Általános szabály, hogy a neuronok számát a rendelkezésre álló adathalmaz mérete alapján válasszuk meg. Minél több egyed áll rendelkezésre a tanításhoz, annál több elemű rétegeket érdemes használni.



2.7. ábra: Többrétegű perceptron egy rejtett réteggel [17]

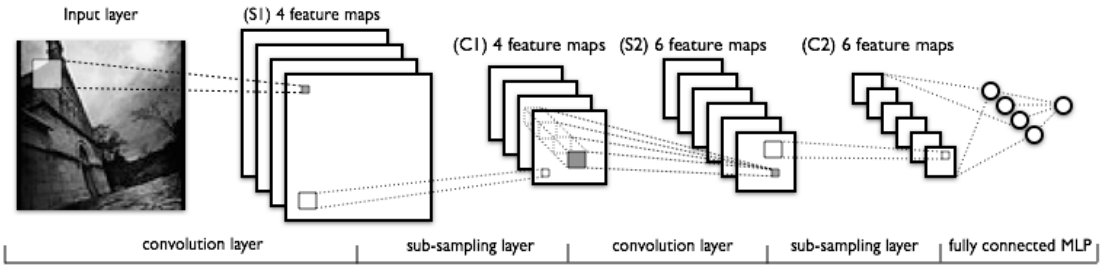
Az ilyen típusú rendszerek nagy előnye a kevés beállítandó paraméter és azok optimalizálásának egyszerűsége, valamint az, hogy több más mély tanítási struktúrával ellentétben kis adathalmazokra is taníthatók megfelelő eredményekkel. Hátrányuk, hogy hajlamosak a túltanulásra, és hogy nagy réteg vagy elemszám esetén nagy a számítási idejük. A túltanulás való hajlandóság magyarázható a hozzáadott absztrakt rétegekkel, amik lehetővé teszik a tanító adathalmaz ritkán előfordulói jelenségeinek modellezését.

Több különböző megoldás is született az említett túltanulás leküzdésére, ezek közül az egyik legeredményesebb nem túl régi módszer a dropout regularizáció [18]. Az alapötlet, hogy véletlenszerűen dobjunk el neuronokat kapcsolataival együtt a hálózatból

tanítás során. Ez megelőzi a neuronok túlságos ko-adaptációját, így növelik a neurális hálózatok teljesítményét felügyelt tanulás során.

2.3.2 Konvolúciós neurális háló

Konvolúciós neurális hálóknak nevezzük a többrétegű perceptron biológiai látórendszerek által inspirált változatát [15]. Tudósok már korábban rájöttek, hogy például a macskák látórendszere sejtek komplex rendszeréből áll, a látóteret több alrégió fedi le, melyek lokálisan szűrik a bemeneti teret. Mivel az állatok látórendszere a legerősebb vizuális feldolgozó rendszer, amit ismerünk [17], kézzel fekvőnek tűnik működését emulálni mérnöki rendszerekben. Számos különböző konstrukcióval rendelkező konvolúciós hálózatot használnak a terület szakértői, többnek megkülönböztető neve is van, mint például a LeNet-nek (2.8. ábra), ami az első sikeres hálózat [19], de nagy jelentőségű az AlexNet és a GoogLeNet is [20]. Ezen hálózatok felépítésének tanulmányozásával jelöltem ki munkám során a tesztelésre jelölt hálózatstruktúrákat.

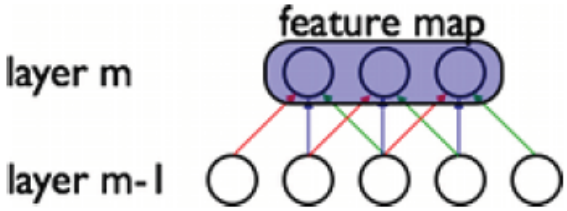


2.8. ábra: LeNet konvolúciós hálózat struktúrája [17]

A többrétegű perceptronoktól három fő tulajdonság különbözteti meg a konvolúciós neurális hálózatokat [17]. Egyik a rétegek háromdimenziós rendezése, ezek a magasság, szélesség és mélység. Másik fontos tulajdonság a lokális kapcsolatok, vagyis a szomszédos rétegek neuronjai között létrehozott kapcsolati minta. Ez utóbbi miatt kapunk nemlineáris, globális szűrőket. Végül, de nem utolsó sorban a közös súlyok rendszere, ami jellemezi a konvolúciós neurális hálózatokat. Ez röviden azt jelenti, hogy minden egyes szűrőt a teljes mintán alkalmazzuk ugyanazokkal a paraméterekkel. Így az egyes sajátosságok detektálása független lesz azok pozíciójától (például egy arc észrevétele egy képen).

A közös, osztott súlyok használatának egyéb hozadéka, hogy csökkennek a megtanulandó szabad paraméterek, így növekedni fog a tanulás hatékonysága. Egy ilyen

osztott súlyokkal rendelkező szűrők összességét sajátosság térképnek (feature map) hívunk, sematikus ábrázolása látható a 2.9. ábraán. Itt 3 rejtett elem tartozik az *m*. rétegben egy sajátossághoz, a közös súlyokat pedig az azonos színű kapcsolatok jelképezik. A tanulás közben az azonos színnel jelzett élek súlyozására egyenlő értéket tartó korlátozást vezetünk be.



2.9. ábra: Sajátossági térkép egy konvolúciós neurális hálózatban

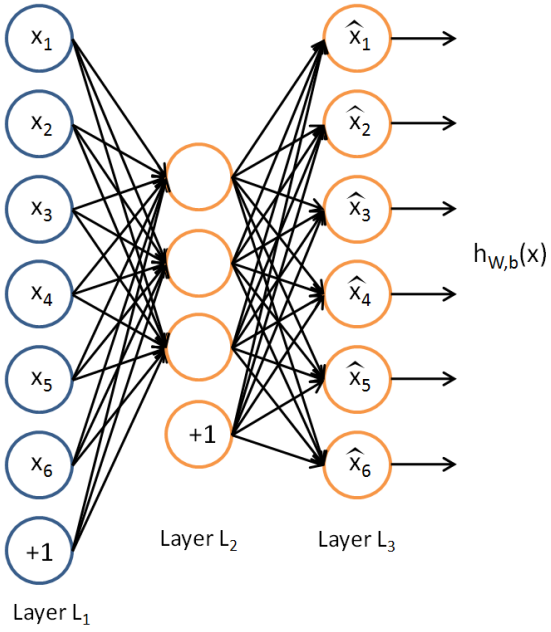
A használt architektúra kétféle rétegből épül fel, konvolúciós és összevonó (angolul pooling, néhol subsampling) rétegekből. Minden réteg rendelkezik egy topográfiai struktúrával, amely meghatározza, hogy egy adott neuron a bemenet mely „pozíciójához" van rendelve. Minden rétegben a neuronok súlyai az előző réteg egy téglalap alakú területén elhelyezkedő neuronjaihoz rendeltek. Ugyanazok a súlyok, de különböző területelemek tartoznak a különböző pozíciókban elhelyezkedő neuronokhoz [15]. A konvolúciós rétegekben a konvolúciós művelet egy tanult kernellel (szűrővel) történik, az összevonás pedig a minta nemlineáris méretcsökkentésének egy fajtája [17]. Utóbbi esetében szokás például a maximum összevonás, azaz a bemeneti adatokra, például képre nem átlapozódó négyszögeket illesztünk, majd kimenetként csak e négyszögekhez tartozó értékek maximumát adjuk ki. Így csökkentettük a minta méretét, amivel együtt csökkentettük a tanulandó paraméterek számát, így a túltanulás esélyét is. Egyéb rétegtípusok is használatosak a konvolúciós neurális hálózatokban, például ReLU (egyenirányító lineáris egység), teljesen csatlakoztatott (hagyományos neurális hálóhóz hasonló) és hibaképző rétegek.

A konvolúciós neurális hálózatok nagy előnye hagyományos neurális hálókhoz képest könnyebb tanítás. Ezen kívül jóval kevesebb paramétert kell becsülnie, ezért is terjed el egyre jobban a használata [21]. Alkalmazásának egy tipikus példája a számítógépes látás területe, ide tartozik a Google által létrehozott DeepDream szoftver is [22].

2.3.3 Auto enkóderek

Az auto enkóderek a mély tanulási módszerek családjába tartozó felügyelet nélküli tanító algoritmusok. Implementálásuk és tanításuk a többi hasonló metódushoz képest egyszerűbb, így gyakran alkalmazzák mély neurális hálózatok építőelemeiként, ahol minden egyes szint egy önállóan tanítható auto enkóder.

Az auto enkóder lényege, hogy megtanulja a bemenetet egy olyan reprezentációra kódolni, amiből később a kezdeti bemenet visszaállítható [23]. Tehát az ideális kimenet egyben a bemenet is. Ha a rejtett rétegünk lineáris, akkor hasonló működést tapasztalunk, mint PCA esetén. Nemlineáris rejtett réteg esetén azonban többmódusú jellemzőket is kinyerhetünk auto enkóder segítségével.



2.10. ábra: Auto enkóder felépítése [21]

Egy auto enkóder tipikusan áll egy bementi rétegből, ami az eredeti adatokat, vagy leíró vektorokat reprezentálja, egy vagy több rejtett rétegből, amik a transzformált, kódolt sajátosságot reprezentálják, végül pedig egy kimeneti rétegből, ami segítségével összehasonlítjuk az eredeti bemenetet a visszaállított bemenettel. Egy jellemző reprezentációt mutat a 2.10. ábra. Ha több mint egy rejtett réteget tartalmaz, akkor az auto enkódert mélynek nevezzük. A rejtett rétegek dimenziója lehet kisebb (ha tömöríteni akarunk), vagy nagyobb (ha magasabb térbe akarunk letérképezni), mint a bemeneti dimenzió. Az auto enkódereket gyakran a számos back-propagation algoritmus egyikével tanítjuk, tipikusan sztochasztikus gradiens metódussal.

Az egyik probléma a megközelítéssel, hogy minden egyéb megkötés nélkül az auto enkóder „n” dimenziós bemenettel és legalább „n” dimenziós kódolással potenciálisan csak megtanulja az identitást, lemásolja a bementet, ami haszontalan. [23] Meglepő módon a gyakorlat azt mutatja, hogy a nemlineáris auto enkóderek sztochasztikus gradiens alapú optimalizálással és a bemenetinél nagyobb dimenziójú kódolással hasznos reprezentációkat tanulnak meg, amik kiaknázzák a statisztikai szabályszerűségeket a tanító adathalmazban, nem csak megtanulják az egyszerű identitást.

Az előző megállapításoktól függetlenül léteznek módszerek arra, hogy egy auto enkóder esetében megakadályozzuk az identitás megtanulását úgy, hogy továbbra is a bemenet hasznos reprezentációját kapjuk. Az egyik elterjedt módszer a zajtalanító auto enkóder, ami a bemenetet egy sztochasztikusan szennyezett transzformáltjából próbálja visszaállítani. Ezzel a működéssel az enkóder több szempontból is hatékonyabban működik [14]. Először is, mivel a kimenetnek az eredeti, zaj nélküli bemenettel kell megegyeznie, a modell el fogja kerülni az identitás megtanulását. Másodszor: a véletlenszerű zajok miatt a megtanult reprezentáció robosztus lesz a tesztminta hasonló zavaraira. Harmadszor: mivel minden egyes zajjal ellátott bementi minta különböző lesz, nagyban megnöveljük a tanító adathalmaz méretét, így csökkentve a túltanulás esélyét.

A zajtalanító auto enkódereket lehet egymásra halmozni, az egyes rétegek kimeneti kódjait a következő enkóder bemenetére kötni, így létrehozva egy mély hálózatot. Ekkor a szokásos felügyelt tanulást megelőzően felügyelet nélküli előtanulás végzendő el, mégpedig rétegenként. A következő fázis a finomhangolás, amikor is az auto enkóderek elé egy regressziós réteget csatlakoztatunk, és az így kapott - már megismert - felügyelt tanítási módszerekkel, többrétegű perceptronként tanítjuk. Több cikkben, például [24]-ben olvasható, hogy ez a módszer nagyban növeli az osztályozás pontosságát a felügyelet nélküli előtanítás nélküli alkalmazásokhoz képest.

2.3.4 Egyéb architektúrák

A munkámban az eddig tárgyalt három struktúrájú neurális hálózatokat vizsgáltam, de fontos megemlíteni, hogy a mély tanulás területén számos egyéb technika is létezik sőt, ez a szám folyamatosan bővül is. A következőkben tömören beszámolok a munkám során részletesen nem elemzett architektúrákról.

Boltzmann gép (Boltzmann Machine, BM): Szimmetrikusan kapcsolat neuronhoz hasonló egységek hálózata, ahol az egységek sztochasztikus döntést hoznak arról, hogy be vagy ki legyenek kapcsolva [25].

Korlátozott Boltzmann gép (Restricted Boltzmann Machine, RBM): Egy speciális Boltzmann gép, ami egy réteg látható és egy réteg rejtett egységből áll, ahol az egységek között nem lehet látható-látható, vagy rejtett-rejtett kapcsolat [25].

Mély Boltzmann gép (Deep Boltzmann Machine, DBM): Egy speciális korlátozott Boltzmann gép, ahol a rejtett egységek mély hálózati struktúrába vannak rendezve, és csak a szomszédos rétegek között lehet kapcsolat [25].

Deep belief network (DBN): Sztochasztikus, rejtett változókból álló többrétegű valószínűségi modell. A legfelső két réteg szimmetrikus kapcsolattal rendelkezik, az alsóbb rétegek mindig csak a felettük lévő rétegtől kapnak információkat [25]. Egy DBN tekinthető egyszerű tanuló modulok összeségének. Használható mély neurális hálók előtanítására úgy, hogy egy betanult DBN súlyait használjuk a neurális hálónk kezdeti súlyaiként [26]. Egy DBN tanítható felügyelet nélküli módon rétegről rétegre, ahol a rétegek tipikusan korlátozott Boltzmann gépekből áll.

Konvolúciós DBN: Egy viszonylag friss eredmény a mély tanulás területén. A struktúra a konvolúciós neurális hálóhoz hasonló, a tanítás módja pedig a DBN-hez. Így egyszerre használja ki a konvolúciós neurális hálókhoz hasonlóan a képek 2 dimenziós struktúráját, és a DBN előtanítási képességét [27].

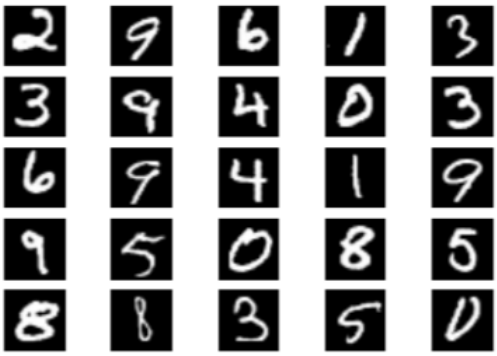
2.3.5 Eredmények összehasonlítása

A munkám során a kutatásomat a következő mély tanulási technológiákra szűkítettem: mély neurális hálók, azaz többrétegű perceptronok, konvolúciós neurális hálók és zajtalanító auto-enkóderek. A legutóbbit csak a többi előtanítási fázisaként alkalmaztam, így jelen fejezetben a különböző típusú mély neurális hálók és konvolúciós neurális hálók eddig elért eredményeit taglalom.

Ahogy az egyes típusok részletes kifejtésénél is írtam, mindkettőjüknek megvannak a felépítésükből adódó előnyeik és hátrányaik. A mély neurális hálók előnye az egyszerű felépítésük és a kis adathalmazokon való alkalmazhatóságuk. Hátrányuk azonban a nagy méreteknél előkerülő túltanulás jelensége és a nagy számítási igény. A konvolúciós hálózat ezzel ellentétben előnyként tudhatja magáénak a túltanulás leküzdését, valamint a hatékony tanulásból következő gyorsabb számítást. Hátrányuk azonban a komplex struktúra, ami komoly és hosszadalmas tervezési procedúrát követel meg egy konvolúciós hálózat megalkotásakor.

Míg a mély neurális hálók már több évtizede alkalmazás alatt állnak, a konvolúciós neurális hálók kutatása csak az utóbbi 10 évben erősödött meg. Épp emiatt az utóbbi évek nagy áttöréseit ez utóbbi hálózattal érték el. Sikeresen valósítottak meg számítógépes játékot megtanuló rendszert [28], értek el rekorderedményt az ImageNet nemzetközi kép osztályozó versenyén [16] vagy alkottak az emberi kézírást 0,23%-os hibával felismerő programot [29].

A kép osztályozás egy megszokott összehasonlításra használt tanító adathalmaza a több mint 70000 kézzel írt számokból álló MNIST adathalmaz [29]. A minták egy kivonatát szemlélteti a 2.11. ábra.



2.11. ábra: MNIST adathalmaz egy véletlen mintavétele

Az MNIST hivatalos oldalán elérhetőek a különböző típusú hálókkal elért eredmények, a munkám számára releváns típusok eredményeiből mutatok be most egy kivonatot a 2-1. táblázatban. Felsorolok pár hagyományos osztályozó algoritmushoz tartozó eredményt is, hogy megmutassam a mély tanulás általános előnyét velük szemben.

2-1. táblázat: Osztályozó algoritmusok MNIST adathalmazon elért eredményei [29]

Típus	Felépítés	Hibaráta (%)
Lineáris osztályozó	-	12
K-legközelebbi szomszéd	-	5
Support Vecot Machine	-	1,4
Neurális háló	2 réteg, 784-800-10 neuron	1,6
Mély neurális háló	6 réteg, 784-2500-2000-1500-1000-500-10 neuron	0,35
Konvolúciós neurális háló	LeNet-5	0,95
Konvolúciós neurális háló	35 konvolúciós háló összesítése [30]	0,23

Látható, hogy míg a hagyományos neurális hálózat hibarátája hasonló nagyságrendben mozog, mint neurális hálózatot nélkülöző klasszikus gépi tanulási módszerek, mint például az SVM, a mély tanuláshoz tartozó struktúrák közül már az egyszerűbbek is felülmúlják azok eredményeit. A mély neurális háló és a konvolúciós háló közötti verseny kimenetele nem eldönthető pusztán a hibaráta alapján, de az megállapítható, hogy helyesen megtervezett konvolúciós hálózat hasonló eredményt tud elérni, mint a nála több nagyságrenddel több neuronnal, és így paraméterrel rendelkező mély neurális hálózat. Így, ha összevetjük a kapott eredményt a szükséges számítási kapacitással, azt kapjuk, hogy a konvolúciós hálózat általánosan jobb választás, mint a mély neurális hálózat. A munkám egy későbbi szakaszában, az általam felvetett probléma megoldásánál is ezt szeretném igazolni.

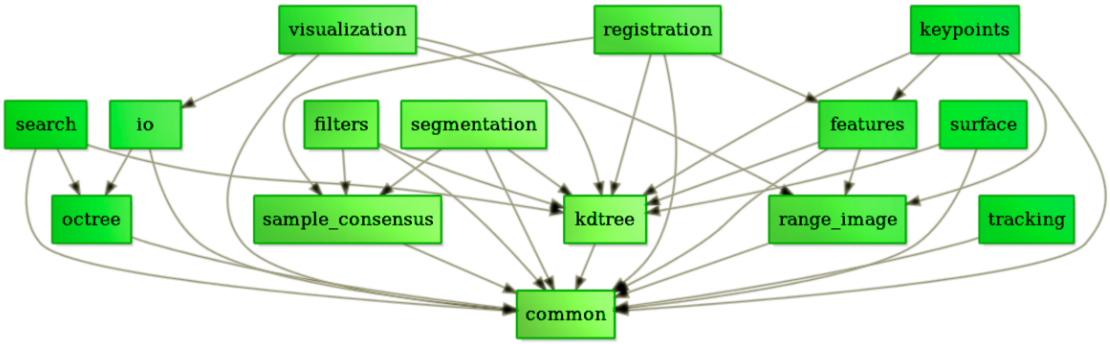
3 Szoftveres előkészületek

A munkám során fő célom egy összetett, pontfelhő reprezentációjú utcai objektumok osztályozására képes szoftver megalkotása. Egy ilyen programot kutatói vagy céges környezetben sok ember fejleszt a számos bonyolult modul miatt. A megvalósíthatóságot szem előtt tartva úgy döntöttem, hogy a saját felelősségemnek az osztályozó program fő felépítését és tesztelését hagyom meg, az alacsonyabb szintű műveletekhez, így a pontfelhők kezeléséhez és a mély tanulás struktúráinak megalkotásához szabadon letölthető és felhasználható programkönyvtárakat használok. Négy fő forrás segítségét vettem igénybe, a pontfelhő alapú műveletekhez PCL [31], a képfeldolgozáshoz OpenCV programkönyvtárat, a mély tanuláson belül a mély neurális hálókhoz és a konvolúciós neurális hálókhoz [32]-et, a zajtalanító auto enkóderhez pedig [33]-at. Ezen szoftverek részletes tárgyalását, a főbb használt metódusok bemutatását végzem el a következőkben.

3.1 PCL programkönyvtár

3.1. ábra: A Point Cloud Library logója [34]

A PCL (3.1. ábra) egy C++ nyílt függvény könyvtár, amelyet pontfelhők kezelésére hoztak létre. Számos alapvető adatstruktúrát, algoritmust tartalmaz különböző feladatokra [6]. A PCL keretrendszerébe tartoznak például szűréshez, sajátosság kinyeréshez, felület visszaállításhoz, modellillesztéshez vagy szegmentációhoz szükséges műveletek [34]. Ezek az algoritmusok több különböző alkalmazáshoz használhatóak, például zajos adatban a kívülálló elemek szűrése, a kép releváns részleteinek kiemelése, objektumfelismeréshez szükséges leírók kinyerése. A fejlesztést egyszerűsítendő, a PCL kisebb könyvtárakra van osztva, melyekben egy-egy tipikus felhasználhatóságú algoritmuscsoport kap helyet [35]. A PCL főbb könyvtárak által alkotott kapcsolati hálóját szemlélteti a 3.2. ábra.



3.2. ábra: A PCL felépítése [35]

A munkám során a PCL-t az ASCII formátumban rendelkezésemre álló nyers pontfelhők konvertálására, elmentésére, megjelenítésére használom. A megjelenítőn keresztül készítek kétdimenziós projekciókat.

3.1.1 I/O eszköztár

A pcl_io könyvtár [36] metódusai két fő funkcióra oszthatók. Egyik a pontfelhő fájlok (PCD formátumban) olvasása és írása, másik pedig a különböző érzékelők segítségével történő pontfelhő felvétel. Több kamerát, a jól ismert Microsoft Kinect-et, és a korábban részletesen tárgyalt Velodyne szenzorokat is tudják kezelni a PCL ide tartozó moduljai.

Mivel a rendelkezésemre álló adatok nem PCD fájlban tároltak, ezért az olvasó metódusokra nem volt szükségem, azonban a későbbi felhasználhatóság miatt az átkonvertált pontfelhőket el kellett mentenem.

3.1.2 Megjelenítés eszköztár

A pcl_visualiztion [37] könyvtár a pontfelhőkön dolgozó algoritmusok eredményeinek gyors és egyszerű megjelenítésére szolgál. A funkcionalitás része többek között a különböző típusú pontfelhők megjelenítése tetszőleges vizuális paraméterekkel (szín, pontméret, átlátszóság stb.), egyéb alakzatok (gömbök, vonalak, síkok stb.) képernyőre rajzolása és hisztogram megjelenítése.

A számos megjelenítő típus közül én a PCLVisualizer-t [38] használom, ami a PCL megjelenítők közül az egyik legösszetettebb és legerősebb. Számos hasznos kiegészítő funkciót ajánl fel a programozónak, például a kamera szabad beállítását, alakzatok rajzolását, és pillanatkép készítését az aktuális kameranézetről. Ez utóbbi műveletnek nagy jelentősége lesz a későbbi munkám során.

3.2 OpenCV

Az OpenCV (3.3. ábra) egy képkezelő- és feldolgozó függvényeket tartalmazó könyvtár, melyet az Intel orosz kutató központjában fejlesztettek ki, a Willow Garage és az Itseez által támogatott [39]. Nyílt forráskódú, a BSD licenc [40] keretein belül szabadon használható. A projekt 1999-ben indult, kezdeti célja optimalizált és nyílt képfeldolgozó programok fejlesztése, melyekre a fejlesztők építkezhetnek, így sokkal érthetőbb és feldolgozhatóbb kódok születnének. Eredetileg C interface-t használt, azonban az OpenCV 2.0. (2009) óta a fő fejlesztési irány a C++ interface.

3.3. ábra: Az OpenCV logója

A könyvtár a képfeldolgozás számos területén alkalmazható, ilyen többek között a robotika, de fontos megemlíteni az arcfelismerést, az objektum felismerést, szegmentációt, mozgáskövetést stb. Az egyes alkalmazások között természetesen nincs éles határvonal, a könyvtár elemei rendkívül rugalmasan alkalmazhatóak. Nagy előnye még, hogy számos platformot támogat (c++, Java, Python stb.).

3.2.1 A Mat típus

Egy digitális képet általában mátrix formájában tárolunk a memóriában, ezért egy képfeldolgozáshoz fejlesztett könyvtárnak, mint az OpenCV-nek tartalmaznia kell egy mátrixos adattárolási megoldást. A kezdeti C interface mellett nagy gondot jelentett a manuális memóriakezelés az OpenCV alkalmazásokban. Ezt küszöbölték ki az OpenCV 2.0.-ban, ahol is a c++ interface mellé megjelent a Mat típus. Ez gyakorlatilag egy osztály, amely két részből áll. Egy fix méretű header tárolja az információkat, a mátrix méretét, tárolási típusát, és a mátrix memóriacímét, és egy pointer, ami a pixel értékeket tartalmazó mátrixra mutat [41] [42]. A mátrix dimenziója a tárolási módszertől függ, mérete pedig képről képre más, általában nagyságrendekkel nagyobb, mint a header mérete. A tárolási típusnál megadható a használt színrendszer és adattípus.

Egy átlagos képfeldolgozási feladat megoldása során az OpenCV több függvényét is használni kell, és ezek között gyakori, hogy képeket adnak tovább egymásnak. Ezek az algoritmusok elég számításigényesek is lehetnek, ezért nem szabad a gyakran nagyméretű képek felesleges másolásával tovább csökkentenünk a programunk sebességét [41]. Ennek a problémának a megoldására az OpenCV referencia számláló rendszert alkalmaz. Ez azt jelenti, hogy minden Mat objektum saját headerrel rendelkezik, de a mátrixon osztozhatnak úgy, hogy pointereik ugyanarra a címre mutatnak. Sőt, a másolás operátor is csak a headert és a pointert másolja le, magát az adatot nem.

3.2.2 Szűrő funkciók

Az OpenCV számos, a zajos képek szűrésére használható metódust kínál a programozónak. Az egyik fő csoportjuk a konvolúciós műveleteken alapuló szűrők. Digitális képfeldolgozás során gyakran végzünk olyan, úgynevezett konvolúciós műveletet a képpontokon, amely műveletek felhasználják a képpont környezetében lévő pixelek értékét. Legegyszerűbb példa az átlag és a medián szűrő, amely minden pixelt helyettesít egy meghatározott környezetének átlag vagy medián értékével. Az átlagszűrőnél komplexebb algoritmus alapján működik a Gauss-szűrő, ami, habár nem a leggyorsabb, de [43] szerint talán a leghasznosabb szűrő. A szűrő kernelének súlyait normál eloszlás alapján számítjuk.

A szűrést megvalósító másik technika a morfológiai műveletek. A morfológia szorosan kapcsolódik az előzőleg tárgyalt konvolúcióhoz, de itt főleg logikai, és nem aritmetikai műveletekről van szó. Alkalmazása főleg bináris képeken történik (képvágás után), amikor is a képvágás műveletének hiányosságait tudjuk vele pótolni, például megmaradt zajokat tudunk eltüntetni. A morfológia alapja Hit és Fit műveletek, amik egy logikai strukturáló elem segítségével egy adott pixel értékét 0-ra vagy 1-re állítja. Ha egy kép minden pixelére Hit-et alkalmazunk, akkor dilatációt végzünk el rajta, ha Fit-et, akkor pedig erodáljuk a képet. Az utóbbi két műveletet kombinálva további, jól használható összetett műveleteket kapunk, mint például a nyitás és a zárás művelete, vagy az utóbbi kettő kombinációját, a határoló vonal felismerést.

A munkám során a LIDAR által felvett pontfelhők kétdimenziós projekcióin használom az előbb felsorolt szűrő műveleteket.

3.3 Mély tanulás programkönyvtár

A mesterséges intelligencia alapú tanuló algoritmusok optimális és hibamentes megvalósítása bizonyos programkörnyezetekben körülményes, különösen igaz ez a mély tanulás területéhez tartozó algoritmusokra. Jelenleg több elérhető szoftvercsomag közül lehet választani, legismertebbek a caffe, TensorFlow és a Theano, de MATLAB környezetben írt mély tanulás függvénycsomagot is lehet találni.

Konzulensi tanácsra tanulmányozni kezdtem egy online elérhető, nyílt forráskódú programkönyvtárat, melyet pontosan erre a célra fejlesztettek [32]. Viszonylag kevés munkával előkészíthető és futtatható a mintaprogram, mely az emberi kézírás felcímkézett adatbázisán (MNIST adatbázis [29]) végzett klasszifikáció során mutatja be a függvények működését. Három fő tanulási módszer implementálásában segít a minta, a konvolúciós háló, többrétegű perceptron és a dropout tanulás használata mind elsajátítható. A szerzők a következő referenciákat adták meg munkájuk alapjául: [19], [44], [21].

A programkönyvtár weboldalán szerepel egy összehasonlítás a caffee, TensorFlow és a Theano szoftverekkel. Előnyként sorolja fel, hogy nincs kötelezően előre telepítendő függősége, valamint a szóban forgó szoftvercsomagot nem is kell telepíteni. Hátrány azonban a GPU támogatás hiánya, és a c++ nyelv, ami a többiek Python nyelvéhez képest nehezebben használható több fórum szerint is. Számomra fontos előny volt még ezen kívül a Windows támogatás, mivel a munkám során Microsoft Visual Studio fejlesztői környezetben akartam dolgozni.

Az alkotók állítása alapján [32] a biztosított mintaprogram gyors, még GPU használata nélkül is, kevés függőséggel rendelkezik és így egyszerűen implementálható, valamint pontos osztályozásra képes. A biztosított MNIST tanulóhalmazon 13 perces tanulás után 98,8%-os pontosságú klasszifikációt képes elvégezni. Támogatja a legfőbb, konvolúciós hálózat elemeiként használható hálótípusokat, az algoritmusok során használt függvények is változtathatóak, az aktivációs, a veszteségi és az optimalizációs függvényeknél is több módszer közül választhatunk.

Jobban tanulmányozva az egyes függvényeket rájöttem, hogy sok helyen specifikusan a már korábban említett MNIST adatbázis mintáira, különösen az alkalmazott speciális fájlformátumra írták a programkönyvtárat. A későbbi feladatok során ezen függvények átírását meg kellett valósítanom.

3.3.1 Funkcionalitás

A használt programkönyvtár ideális különböző felépítésű mély neurális és konvolúciós neurális hálózat felépítésére és betanítására. Fontos azonban, hogy a szoftver nem nyújt segítséget a hálózat megtervezésében, csak futtatásra alkalmatlan paraméterezéssel rendelkező hálózatok tanításakor jelez hibaüzenettel (például szomszédos rétegek dimenziói nem kompatibilisek) a problémáról. Tehát a kívánt hálózatot a felhasználónak kell helyesen előre megterveznie, majd a program segítségével felépítenie. Az így megalkotott hálózat már tanítható és tesztelhető. A megfelelő formátumú tanító és teszt adatokról is a felhasználónak kell gondoskodnia, hacsak nem elegendő a programhoz mellékelt MNIST adathalmaz.

A mély és konvolúciós neurális hálózat felépítésénél a rétegeket használhatjuk, mint építőelemeket. Használhatunk teljesen csatlakoztatott, konvolúciós, átlagoló vagy maximum alapú összevonó réteget, de elérhetőek az olyan speciális rétegtípusok, mint a dropout vagy lineáris műveleti réteg.

Egy neurális hálózat működését nagyban befolyásolják a használt neuronok aktivációs függvényei. A használt szoftver számos beállítható függvényt kínál fel, ilyenek például a közismert tanh vagy sigmoid függvények, de használhatunk softmax, identitás, vagy lineáris egyenirányító aktiválófüggvénnyel rendelkező neuront is.

A tanulás során használt optimalizáló algoritmus terén is ad választási lehetőséget a program. Használhatjuk az alapbeállításnak értelmezett és széles körben elterjedt sztochasztikus gradiens eljárást, de kipróbálhatjuk a hálózatunk működését adagrad vagy rmsprop eljárás alkalmazása mellett is.

A megtervezett, felépített és betanított neurális hálózatok teszteléséhez is több hasznos segítséget nyújt a szoftver. Egyrészt minden tanítási ciklus végén rendelkezésünkre áll az aktuális hibaráta, így tudjuk vizsgálni egy hálózat tanulási folyamatát, meg tudjuk figyelni a hiba változásának tendenciáit. Külön függvények érhetőek el a már betanított hálózatok hatékonyságának mérésére. Futtatva a beépített tesztmetódust, részletes kimutatást kérhetünk az osztályozási pontosságról. Elérhető az a klasszifikációs mátrix, amiről leolvasható az összes helyes és hibás osztályozás száma osztályokra bontva. Nincs egy általános dokumentáló metódus, az eddig ismertetett függvények segítségével a felhasználónak kell megvalósítani a teszteredmények megfelelő naplózását.

3.4 Kiegészítő mély tanulás programkönyvtár

Az előző fejezetben bemutatott szoftver egyik nagy hátránya, hogy a mély és konvolúciós neurális hálózaton kívül nem támogat más mély tanulás területéhez tartozó struktúrát. Más programkönyvtárak viszont nem tartalmazzák azt a szintű támogatást tesztelési funkciók és mintaprogramok tekintetében, mint az általam használt. Megoldásként egy több mély tanulási technológiát több programozási nyelven megvalósító szoftvercsomag megfelelő részeit illesztettem össze a másik program keretrendszerével.

A választott kiegészítő könyvtár [33] tartalmazza az általam eddig hiányolt mély tanulás alapú technológiákat. Egyik hátránya, hogy c++ nyelven nem található meg benne a mély és konvolúciós neurális hálózat implementációja, de szerencsére ezekkel már rendelkezem. Elérhetővé vált viszont így számomra a DBN (Deep Belief Network), korlátozott Boltzmann gépek és zajtalanító auto enkóderek és egymásra halmozott zajtalanító auto enkóderek használata. Az általam használt c++ nyelven kívül c, java, python, scala és go programozási nyelveken érhetők még el a szoftver funkciói, azonban nem mindenhol ugyanaz a választék.

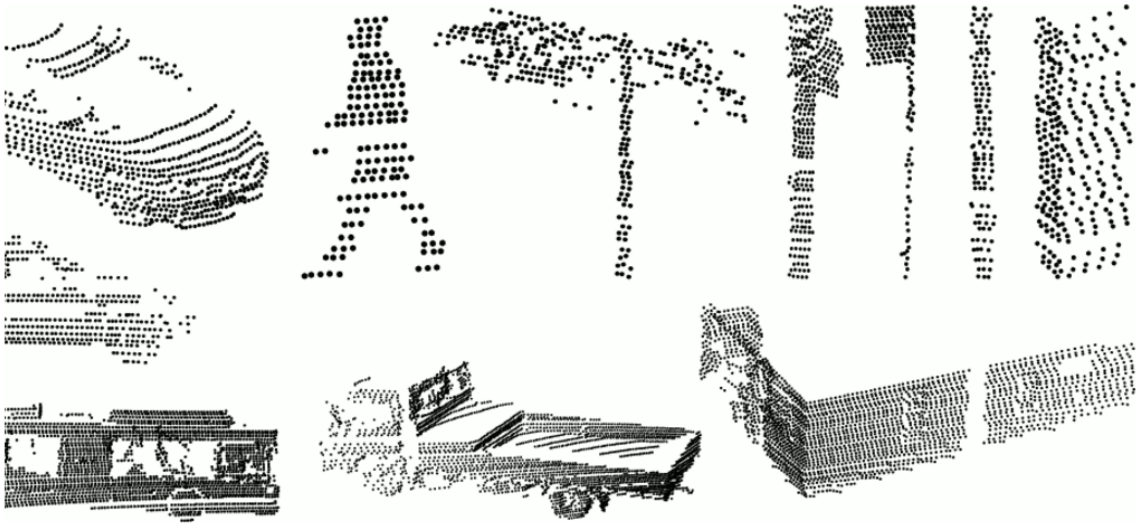
A mély és konvolúciós hálózatot támogató programkönyvtárral ellentétben sokkal egyszerűbb a kód szerkezete, minden algoritmushoz egy header és egy forrásfájl tartozik. A szoftvercsomag e tulajdonsága ideálissá tette a már meglévő nagyobb projektembe történő integrálásra. Természetesen a használt adatstruktúrák, és a felépítés logikája különbözik részben a korábbi programban használtakétól, de mivel nem bonyolult a kód szerkezete, ezeket a kompatibilitási problémákat kisebb energiaráfordítással orvosolni lehet.

3.5 Pontfelhő objektumkészlet

Az utcai objektumok mély tanulás alapú osztályozásához szükség van olyan utcai objektumokra, amiket valós LIDAR eszközzel vettek fel, szegmentáltak és megjelölték az objektum típusát, azaz felcímkézték őket. A projekt korábbi szakaszában az MTA SZTAKI biztosított számomra adatokat, azok azonban nem nyers pontfelhők, hanem már csak az azokból kinyert sajátosságértékek voltak. Így csak magát a mély tanulást tudtam implementálni és tesztelni, nem volt lehetőségem közvetlen megállapításokat tenni a mély tanulás pontfelhőkön történő alkalmazásáról.

Az elérhető és szabadan felhasználható kutatási anyagok közül az Ausztrál Robotikai központ egy projektjét [1] választottam fő forrásomnak, amely keretei között hasonló témával foglalkoztak, mint én, azaz kültéri háromdimenziós objektumok osztályozásával. A konkrét céljaik és módszereik merőben mások voltak, azonban honlapjukon elérhetővé tették a kutatás során használt adatokat [45]. A feltöltött pontfelhők mennyisége és minősége miatt úgy döntöttem, hogy ezeket fogom a munkám során fejlesztett mély tanulási algoritmusok tanító és teszt adatainak alkalmazni.

Az adathalmaz hétköznapi utcai objektumok pontfelhőiből áll, melyeket Velodyne HDL-64E LIDAR eszközzel vettek fel Sydneyben. Összesen 631 egyedet számlál a minta számos típusú objektumot lefedve autóktól kezdve a gyalogosokon át a fákig és táblákig (néhány objektum látható a 3.4. ábraán).



3.4. ábra: A használt ACFR pontfelhő adatbázis néhány objektuma [45]

Az ACFR munkatársai kifejezetten azzal a céllal készítették és osztották meg az adatbázist, hogy azzal osztályozó algoritmusokat lehessen tesztelni. A minták tulajdonságaiban megjelenítik azt a nem ideális érzékelést, ami egy valós városi feltérképező alkalmazásra, például egy autonóm autó LIDAR alapú leképzésére jellemző. A legtöbb objektumtípusról több különböző minőségű pontfelhő található különböző nézőpontokból. A fájlok több formátumban is elérhetők, én az egyszerű .csv fájlban tárolt ASCII változatot használtam, mert ezt a legegyszerűbb PCD pontfelhő fájllá konvertálni.

Minden objektum esetén pontonként tárolt a pont lemérésének pontos ideje, a visszatérő érzékelt lézerfény intenzitása, a lézer azonosítószáma (a használt Velodyne eszköznek ugyanis 64 lézere van), a pont háromdimenziós mért koordinátái, a vízszintes irányszög radiánban, a pont mért távolsága, a pont származási helyeként szolgáló teljes 360 fokos letérképezés azonosító száma.

Az adatbázisban számos olyan típusú objektum is szerepel, amit csak nagyon kevés egyed reprezentál. Ezeket a feltöltők nem ajánlják objektumfelismerő algoritmusok tesztelésére, de kihívásként belevehetők a tanító adathalmazba. Az összes minta 4 egyenlő részre lett osztva egyed duplikációk nélkül. A 3-1. táblázat tartalmazza a főbb osztályokat és a hozzájuk tartozó egyedszámot.

3-1. táblázat: ACFR pontfelhő adatbázis osztályai és a hozzájuk tartozó egyedszámok

Osztály	**Egyedszám**
gyalogos	152
autó	88
közlekedési tábla	51
közlekedési lámpa	47
teherautó	35
fa	34
oszlop	21
épület	20
busz	16

4 Tervezés

A munkám végső célja, vagyis egy mély tanulás alapú, városi LIDAR pontfelhőkön objektumfelismerést elvégezni képes szoftver implementációja nagyfokú előkészítést igényel. Jelen fejezet témája a probléma megoldásának lehetséges lehetőségeinek taglalása, a tervezett működés, a keretrendszer bemutatása, valamint a későbbi validációs lehetőségek elemzése.

4.1 Opciók elemzése

A tervezés alapjának egy korábbi munkámat [46] vettem, ahol megvalósítottam egy mély tanulás alapú objektumosztályozó programot. Tanító és tesztadatként ott nem pontfelhőket, hanem azokból előzetes feldolgozás során kinyert sajátosságvektorokat alkalmaztam. A mély tanulás alapú algoritmusok rugalmassága és a program szerkezete miatt azonban így is támpontként tudtam használni a korábbi tapasztalataim, munkáim.

A cél tehát tömören összefoglalva az, hogy az ACFR adatbázis [45] pontfelhőit, mint az objektumfelismerő program bemeneteit a megfelelő osztályba soroljuk. A megvalósításnak két jól elkülöníthető kihívását emelném ki, véleményem szerint ezekre kell nagyobb hangsúlyt fektetnem. Az egyértelműbben megoldható a különböző tesztelésre kerülő mély tanulás alapú algoritmusok tervezési paramétereinek hangolása. Erre [46]-ban felvázoltam egy módszert, és ugyanezt tervezem jelen munka esetében is alkalmazni. A másik, komolyabb probléma a bemeneti adatok és a mély tanulás implementációját segítő programkönyvtár [32] inkompatibilitása.

A rendelkezésre álló pontfelhők adatszerkezete háromdimenziós, azonban a mély tanuláson alapuló algoritmusok elméleti leírásai, és főleg a szoftveres megvalósításai a legfeljebb kétdimenziós bemeneti adatokra fókuszálnak. Az irodalomkutatás során is megfigyeltem, hogy a nagy mennyiségű publikáció szól a kép- és hangfelismerésben elért eredményekről, a háromdimenziós adatokról csak néhány. Három lehetséges megoldási módot vetettem fel, a mély tanulás programkönyvtár teljes szintű módosítását, pontfelhők mélységi kép reprezentációjának, vagy a pontfelhőkről képzett hengeres projekciók használata a nyers pontfelhők helyett. A következő fejezetben részletesen elemzem ezeket a lehetőségeket, és indoklom a végső választásomat, a hengeres projekciók használatát.

4.1.1 Mély tanulás programkönyvtár módosítása

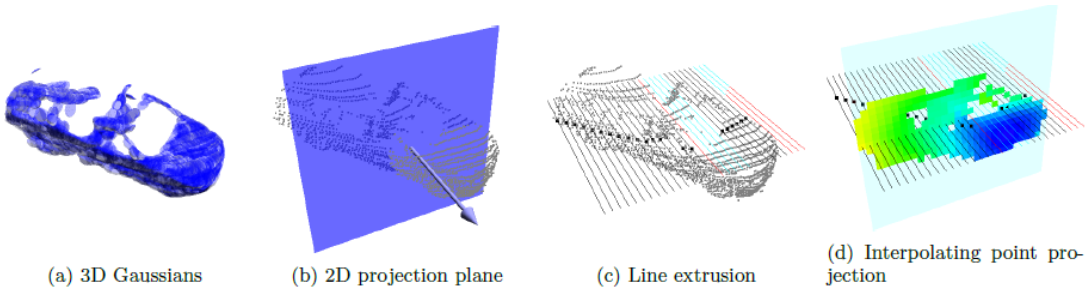
Első ötletem a mély és konvolúciós neurális hálók összeállítására és tanítására használt szoftver olyan szintű módosítása, hogy képes legyen háromdimenziós tömbként kapott bemenetek kezelésére is.

Ha megvalósulna, ez a megoldás lenne az ideális, hiszen a mély tanulás egyik fő előnye a sajátosságok megtanulása, így nagyon jó eredményeket produkál nyers adatok alkalmazásakor. A másik két megoldási módban már feltételezek egy előfeldolgozást, mely során mindenképpen történik információvesztés, ami az osztályozási hiba növekedését okozhatja. A nyers pontfelhőkön történő objektumfelismerés során, ha megvizsgálnánk a köztes absztrakt sajátosságokat, hasznos megállapításokat tudnánk tenni a pontfelhők feldolgozásával kapcsolatban.

Részletesen megvizsgálva az ötletet két hátráltató tényezőt állapítottam meg, ami miatt végül elvetettem a megvalósítását. Az egyik a szoftver módosításának munkaigénye. Hasonló típusú és skálájú szoftveres munka nem egy egyszemélyes projekt, számos hibalehetőséget tartogat, így ha eltekintünk minden más nehézségtől, akkor se valószínű hogy önerőből sikerülne a módosítás véghezvitele. A másik akadály a mély és konvolúciós neurális hálók során alkalmazott matematikai műveletek átfogalmazása háromdimenziós adatokra. Ez a lépés néhol egyértelmű ugyan, de több helyen, például a konvolúciós réteg főbb műveleteinél olyan átalakításokra lenne szükség, ami szintén túlmutat a jelen projekten.

4.1.2 Mélységi képek használata

A pontfelhők helyettesítése azok mélységi kép reprezentációjával az objektumfelismerő program bemeneteként a használt ACFR adatbázis egyik fő referenciaként megadott cikkéből [1] származó ötlet. Lényege, hogy a bementi pontfelhőről egy előre definiált algoritmussal mélységi képet készítenek, ami már egy kétdimenziós adatstruktúra, így a későbbi objektumfelismerésnél a számos jól bevált algoritmus gond nélkül használható. A folyamat lépéseit mutatja a 4.1. ábra.



4.1. ábra Mélységi kép készítése pontfelhőről [1]

A folyamat során [47] alapján a pontfelhő lokális régióit, mint Gauss eloszlásokat kezelik (4.1. ábra – (a)). Ezeket a régiókat egy megfelelően választott síkra vetítik (4.1. ábra – (b)). A síkot az objektum tömegközéppontjának és a kamera középpontjának helyzete határozza meg, a választott sík normálisa ugyanis az e két pontot összekötő vektor. A vetítés utáni kép pixeleinek az értékét úgy határozzák meg, hogy az eredeti háromdimenziós térben egy merőlegest állítanak az adott pixelre vetítik (4.1. ábra – (c)), és vizsgálják a Gauss eloszlás értékét és az kapott mélységi értéket eltárolják. Egy tipikus végeredmény látható a 4.1. ábra/d-n.

A vázolt megoldás a cikk alapján eredményes volt, sikeresen betanítottak több objektumfelismerő modellt, az elért legjobb általános osztályozási pontosságok 60% felett voltak. Fontos kiemelni, hogy nem használtak mély tanulás alapú algoritmust, hanem hagyományos (k legközelebbi szomszéd és SVM) módszerekkel tesztelték a pontfelhők mélységi kép reprezentációinak relevanciáját az objektumfelismerésben. A munkám során tehát az ő általuk meghatározott előfeldolgozást használva igazolhatnám, hogy a mély és konvolúciós neurális hálók esetén is működőképes a módszer, esetleg jobb osztályozási pontosság is elérhető.

Más megfontolásból azonban elvetettem a mélységi képek használatát. A fő indokom éppen az, hogy ha nem is mély tanulási módszerekkel, de már igazolva lett a módszer hasznossága. Mivel feladatom egy teljes, pontfelhő objektumokat osztályozni képes szoftver megvalósítása, úgy határoztam, hogy a pontfelhők előfeldolgozásánál saját, vagy legalábbis konkrétan még nem tesztelt módszert használok. Ezáltal nem csak pusztán azt tudnám igazolni, hogy a mély tanulás eredményesen használható a pontfelhők osztályozásában, hanem egy újfajta pontfelhő reprezentáció objektumfelismerésben való használhatóságát is vizsgálnám.

4.1.3 Hengeres projekció használata

Az előző fejezetben leírt mélységi kép reprezentációs módszer adta azt az ötletet, hogy célszerű lenne egy jól definiált projekciós eljárást használni, hogy kihasználhassam a kétdimenziós adatstruktúrák előnyeit. Mivel a konkrét megvalósítást nem akartam átvenni, saját megoldáson kezdtem gondolkodni.

A projekció meghatározásakor a legfőbb feltételem az volt, hogy a lehető legtöbb információt tartson meg a pontfelhő objektumokról függetlenül azok típusától, méretétől vagy minőségétől. Hasonlóan fontos volt a globális megvalósíthatóság, azaz olyan vetítési eljárásra van szükség, ami a használt adatbázis összes pontfelhőjén egyértelműen és automatikusan elvégezhető. Ezen kívül törekedtem a túlságosan komplex algoritmusok elkerülésére, mivel azok esetleges hiba vagy pontatlanság esetén nehezen felülvizsgálhatók és módosíthatók.

Az általam javasolt megoldás a röviden csak hengeres projekciónak nevezett eljárás használata. A művelet során az objektum tömegközéppontján átmenő függőleges tengely körül elforgatjuk a pontfelhőt, bizonyos lépésenként levetítjük a pontokat egy függőleges síkra, majd az így kapott vetületi képeket összeolvasztva megkapjuk a kívánt projekciót.

A hengeres projekció több szempontból is teljesíti a korábban leírt kívánalmakat. Először is, mivel a legtöbb utcai objektum szimmetrikus, viszonylag kevés részletinformáció fog eltűnni. A forgatások és vetítések után minden objektum képe egy szimmetrikus alakzat lesz, így az extrém alakbéli különbségek egy osztályon belül véleményem szerint enyhülnek, jobban szeparálhatók lesznek az objektumtípusok. Implementációs oldalról megvizsgálva az algoritmust, nem láttam nehézséget abban, hogy egyértelműen meg lehessen valósítani ugyanazt az eljárást minden egyes objektumra. A használt pontfelhőkönyvtár elérhető metódusait vizsgálva hamar találtam több olyat, ami potenciálisan felhasználható a hengeres projekciós eljárás során. Egy esetleges módosítás, például a felvételek száma, vagy az összeolvasztás módja, esetleg utólagos szűrések beiktatása is egyszerűen megvalósítható.

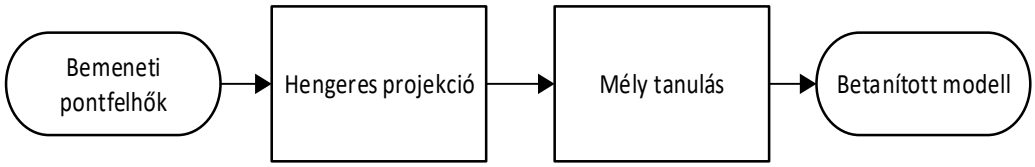
Összefoglalva tehát, megvizsgálásra került a pontfelhőobjektumok és a mély tanulás algoritmusok kompatibilitásproblémái megoldásának három lehetséges módszere, melyek közül kiválasztottam egy saját tervezésűt.

4.2 Keretrendszer tervezete

Jelen fejezetben bemutatom a választott projekciós eljárásból és a mély tanulás alapú objektumfelismerésből összeállítandó szoftver tervezésének menetét, a programelemek kapcsolatát, a működés folyamatának leírását.

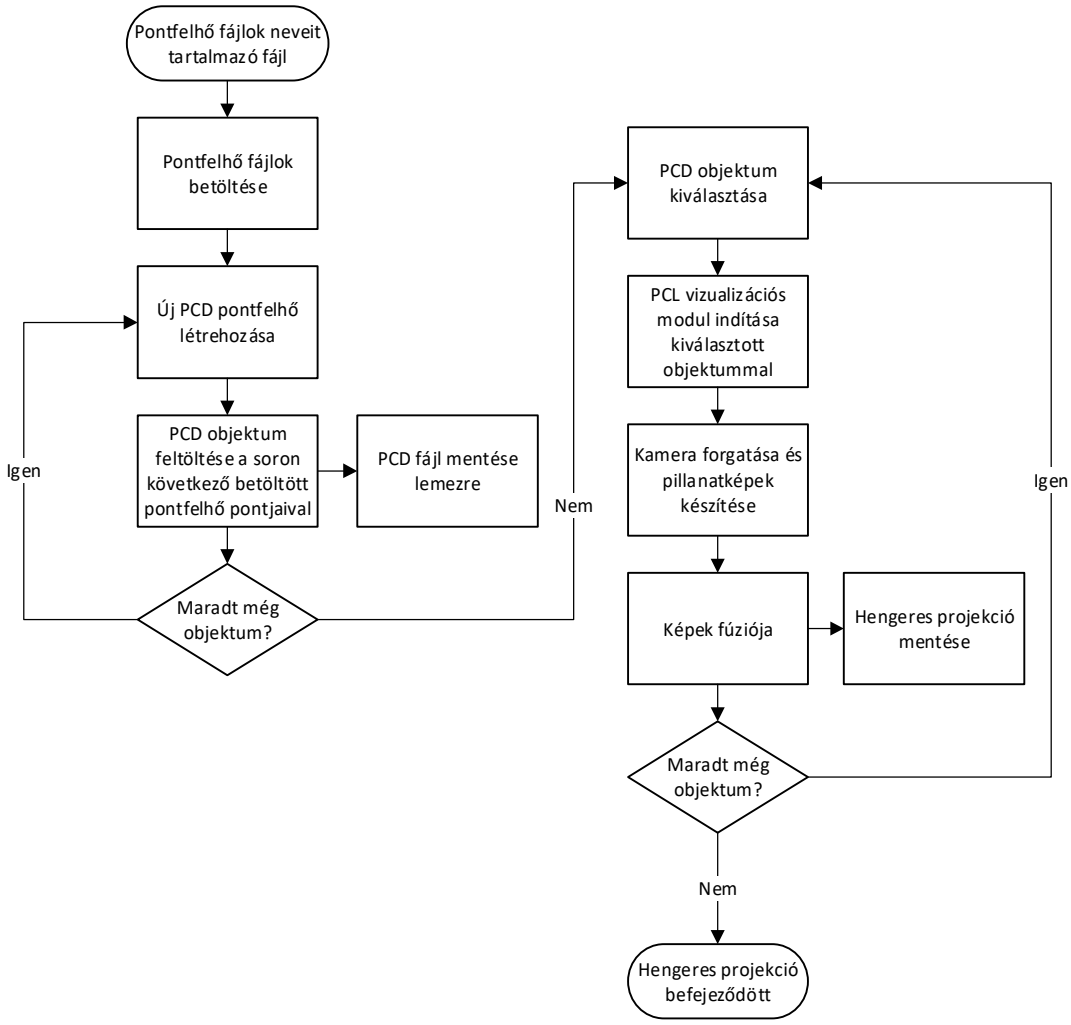
Mint minden gépi tanulásra alkalmas szoftver, az enyém is a működési cél alapján két fő részre osztható, a pontfelhő adatbázis alapján az osztályozást megtanuló, és az osztályozást a betanult modell alapján elvégző program. Mindkettő hasonló elemekből építkezik, természetesen az előbbi jóval komplexebb, és működésének sikeressége meghatározza az utóbbi rész használhatóságát, így munkám során a tanuló algoritmust megvalósító szoftverrész implementálására helyezem a hangsúlyt.

A mély tanulás alapú objektumfelismerés felcímkézett adatbázissal történő betanítását két főbb részre különítettem el. A bemeneti adatok, vagyis az ASCII formátumban kapott pontfelhők először egy hengeres projekciót megvalósító programrészben kerülnek feldolgozásra, ahonnan a kapott képi adatok tanító és teszt adatokra bontva címkéikkel együtt a tanító programba kerülnek. A folyamat kimenete egy betanított és letesztelt mély tanulás modell paraméterei, valamint a tesztelés során elért eredményei, melyeket később a közvetlen objektumfelismerésnél használni lehet. A folyamatot szemlélteti a 4.2. ábra.



4.2. ábra: Mély tanulás alapú objektumfelismerés betanítás folyamata

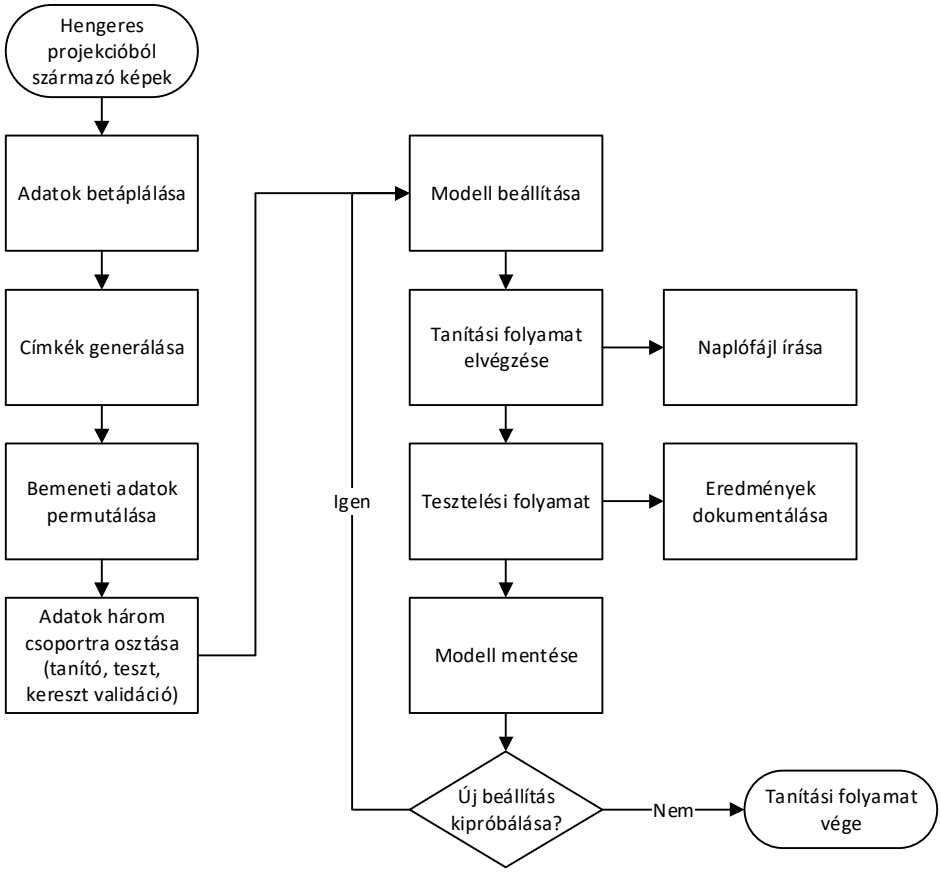
A hengeres projekció esetén a nyers pontfelhőadatokon számos különböző műveletet kell elvégezni az adatok beolvasásától a projekciós kép elkészítéséig. A teljes folyamatot szemlélteti a 4.3. ábra. Első lépésként betöltjük a kiválasztott pontfelhőfájlok neveit. Itt dől el, hogy a négy részre osztott adatbázis mekkora részére vagyunk kíváncsiak. A betöltött fájlnevekhez tartozó pontfelhőfájlokat beolvassuk, a folyamat közben minden egyes objektum esetén létrehozunk egy új PCD pontfelhőt, amit pontonként feltöltünk az aktuális objektum pontjaival. Miután minden pontfelhővel végeztünk, kiegészítő műveletként elmentjük a kapott pontfelhőket.



4.3. ábra: Hengeres projekció folyamata

A rendelkezésre álló, most már a használt PCL programkönyvtár metódusaival teljesen kompatibilis formátumban lévő pontfelhőkön ismét egyesével megyünk végig, és valósítjuk meg a hengeres projekcióhoz szükséges síkra vetítéseket. Először elindítjuk a PCL megjelenítő modulját, ahová betöltjük az aktuális pontfelhőt. Kihasználjuk a megjelenítő azon tulajdonságát, hogy inicializáláskor a kamera pozícióját a pontfelhő objektum tömegközéppontja felé helyezi, és le is méri az objektum magassági méreteit. Ezt kihasználva már tudunk definiálni egy algoritmust, amely segítségével a kamerát körbe tudjuk forgatni az objektum körül megfelelő távolságra úgy, hogy közben mindig a tömegközéppont felé néz. A forgatás közben elkészítjük a képeket, amiket ideiglenes helyen tárolunk. Az elkészült felvételeket ezután egy súlyozó összeadással, OpenCV metódusok segítségével egy képpé fuzionáljuk, majd elvégezzük az esetleg szükséges utómunkákat, végül elmentjük a kapott képeket.

A mély tanulást elvégző programrészt úgy tervezem, hogy több különböző algoritmus (legalább a mély és konvolúciós neurális hálózat) alapján tudjon működni. A tanulás tervezett folyamatát szemlélteti a 4.4. ábra. Az első művelet minden esetben az adatok betáplálása és a címkelista összeállítása. Rendelkezésre állnak kezdetben az előző modul végeredményeként elmentett képek. A fájlok nevei tartalmazzák a helyes címkét, azokat egy külön vektorban sorban eltároljuk. A tanulás robusztus végkimenetelét garantálandó, szükség van a bemeneti adatok véletlenszerű permutálására.



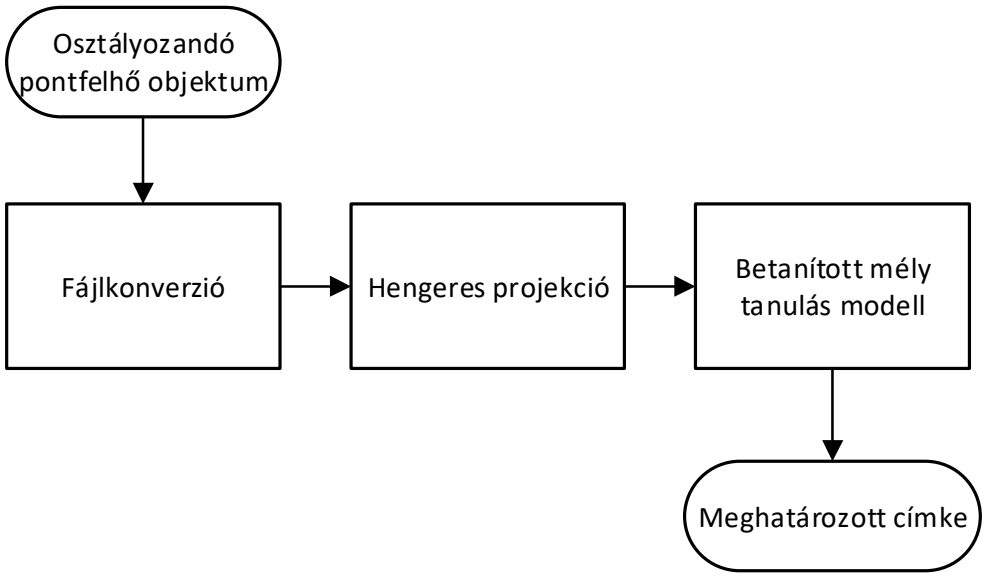
4.4. ábra: Mély tanulás alapú objektumfelismerés betanítási folyamata

A megkevert adatokat három különböző méretű részre osztjuk. Egy nagyobb méretű részt használunk fel a mély tanulás alapú modell tanítására, egy kisebbet a modell tanítási cikluson belüli kiértékelésére, végül egy szintén kisméretű részt a kereszt validációra. Utóbbira azért van szükség, mert a tanító program nemcsak a modell betanítását, hanem a választott mély tanulás alapú módszer szerkezeti paramétereit (például a rejtett elemek száma egy rétegben, vagy a konvolúciós hálózat felépítésének módja) is folyamatosan változtatja és kiértékeli, így a folyamat végére egy optimálisan beállítottnak mondható tanító algoritmus által tanított és tesztelt modellt kapunk. Ezt a módszert egy korábbi munkám során [46] sikeresen alkalmaztam.

A tanítás fő részét képezi maga a tanító algoritmus futtatása. Itt használom ki legjobban a mély tanulás programkönyvtárak ajánlott metódusait. Az előzetesen megtervezett és beállított modellen a megfelelő paraméterezéssel megívom annak betanító műveletét a korábban létrehozott tanító adatbázissal. A folyamat bizonyos számú iteráció után sikeresen leáll, vagy hibával kilép, ha gondba ütközik a tanítási folyamat során.

A tanítás befejező műveletei a dokumentálást végzik el. A tanítás egyes ciklusainak eredményeit is folyamatosan mentjük egy naplófájlba, bár ennek tényleges felhasználása, kiértékelése nagyon nehézkes lenne. Ezért minden egyes különbözően beállított modell betanítása után elmentjük a kapott modellt, és a korábban elkülönített kereszt validációs adatokon is mérjük az objektumfelismerés teljesítményét. Az itt kapott eredményeket részletesen kiértékeljük, és elmentjük későbbi feldolgozásra.

A betanított modellt felhasználó egyszerű objektum beosztályozó algoritmus működése egyértelmű. A bemeneti pontfelhőt, ha szükséges, PCD formátumra konvertáljuk, majd elvégezzük a hengeres projekciót. A kapott vetületi képet a modellünk bemenetére adjuk és kimenetként megkapjuk a meghatározott osztály sorszámát. A folyamatot a 4.5. ábra szemlélteti.



4.5. ábra: Mély tanulás alapú objektum osztályozás folyamata

4.3 Validációs lehetőségek

A munkám végső célja annak igazolása, hogy a mély tanulás nagy megbízhatósággal használható utcai LIDAR pontfelhő objektumok osztályozására. A feltevésem belátásához nem elegendő csak egy megtervezett mély vagy konvolúciós neurális hálózaton a teljes ACFR adatbázis alapján tanítást majd tesztelést végezni. Előre meghatározok tehát olyan követendő kiértékelési, összehasonlítási lehetőségeket, melynek segítségével már nagy megbízhatósággal meghatározható a mély tanulás alapú algoritmusok várható osztályozási pontossága a meghatározott feladat során.

Fontos tisztázni, hogy az ACFR pontfelhő adatbázis számos olyan osztályból is tartalmaz mintát, amik túlzottan alulreprezentáltak a mintában. Az első szintű kiértékeléskor ezeket érdemes kihagyni, és csak a legtöbb egyeddel rendelkező, maximum 5-6 osztállyal dolgozni, a többi mintát pedig egyszerűen negatív mintának címkézni. Ha a legtöbb egyeddel rendelkező osztályok körében már igazolni tudnám a mély tanulás sikerességét, folyamatosan lehetne emelni a vizsgált osztályok számát.

A különböző beállítású mély tanulási algoritmusok osztályozási pontosságát többféleképpen is össze lehet vetni. Használható az egyszerű összesített pontosság, azaz a helyesen osztályozott minták hányada. Ez első közelítésben adhat is helyes sorrendet a különböző modellek között, azonban bizonyos esetekben nem nyújt pontos képet az algoritmusról. Elképzelhető olyan eset, amikor a gyakori, és a valós alkalmazásokban kiemelten fontos osztályokat (autó, gyalogos stb.) kiemelkedően nagy arányban felismeri, azonban egyéb osztályoknál alul teljesít. Értelemszerűen egy ilyen modell jobb, mint egy olyan, aki átlagos pontosságban megelőzi, azonban a lényeges osztályokban jóval rosszabb felismerésiarányt produkál.

Ez utóbbi problémát az egyes objektum-csoportok osztályozásának F1 értékének kiszámításával oldottam meg, amelyet a precízió és recall jellemzőinek harmonikus átlagaként határozunk meg. A kiszámításukat a 4.1, 4.2 és 4.3 egyenlet mutatja, ahol *tp* a helyes pozitív, *fp* a hibás pozitív, *fn* pedig a hibás negatív predikciók számát jelöli. Ahhoz, hogy minden osztályra ki tudjam számítani ezeket az értékeket, osztályonként bináris osztályozási problémának kell kezelni az értékelést, mert a fenti jellemzők definícióból adódóan csak így kiszámíthatók.

$$precision = \frac{tp}{\mathrm{tp} + fp} \tag{4.1.}$$

$$recall = \frac{tp}{\mathrm{tp} + fn} \tag{4.2.}$$

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{4.3.}$$

A kapott eredmények alapján már ellenőrizni lehet, hogy a megfelelő összesített pontosságon túl az algoritmus sikeresen osztályozni tudta-e a különböző arányban reprezentált és különböző fontosságú objektumokat is. A különböző osztályozási pontosságot reprezentáló értékek súlyozott összegzésével meghatározható egy olyan, általános osztályozási teljesítményt meghatározó mérőszám, mely segítségével a vizsgált mély tanulási algoritmusok sorba rendezhetők, így a megvalósítandó feladatra ideális jelöltek választhatók.

5 Implementáció

A következőkben röviden ismertetem az előző fejezetben megtervezett szoftver főbb, konkrét implementációs döntéseit, részleteit, különös tekintettel a mély tanulás paramétereinek a hangolását segítő keretrendszerre. A megalkotott kódok többségében saját megoldások, vagy korábbi projektjeim alapján készültek, néhány ötletet ismert programozói fórumokból merítettem.

5.1 Hengeres projekció

A szoftvert fejlesztési fázisban két részre osztottam, azaz egy pontfelhő kezelő programmal az összes rendelkezésre álló pontfelhőn elvégeztem a hengeres projekciót, majd a kapott képeken tanítom a mély neurális modellt. A projekció során tehát felhasználom bemenetként az összes fájlnevet, amit az ACFR adatbázisból a szerzők ajánlanak. Az így feldolgozandó 588 objektum a teljes adathalmaz azon részhalmazát alkotják, melyek a legnépesebb 14 osztály elemeiből kerülnek ki, és a felvétel minőségét megfelelőnek ítélték meg.

A hengeres projekciós szoftver implementálásánál nem használtam fel meglévő keretet, mintaprogramot, mint a mély tanulás esetén, hanem saját tervezésű struktúrát használtam. Egy fájlnévolvasó modul beolvassa a neveket, amelyet átad egy ASCII adatok beolvasására szolgáló osztálynak. A pontok koordinátáit tartalmazó vektort kiszámítva és egy PCL kompatibilis pontfelhő generálására szolgáló programrésznek átadva megkapjuk az egyes objektumok pontfelhőit.

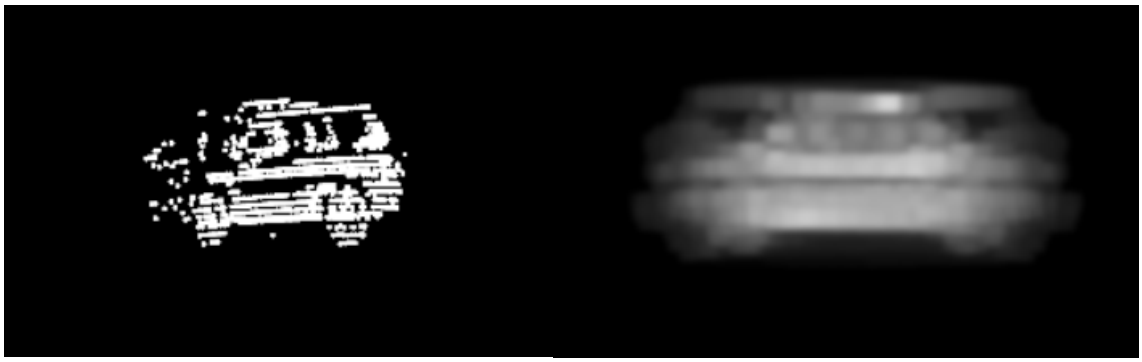
Az objektumok PCD formátumú pontfelhővé alakítására azért van szükség, mert a hengeres projekció elvégzéséhez olyan függvényeket választottam, amelyek csak ilyen típusú adatokon használhatóak. Konkrétan létrehoztam [38] alapján egy PCL megjelenítő modult, amit kiegészítettem egy olyan funkcióval, ami elvégzi a hengeres projekcióhoz szükséges felvételek elkészítését. A projekciók elkészítésének két paraméterét határoztam meg: Az első a felvételek száma, azaz a 360 fokos körülfordulás alatt hány leképezést kívánunk kinyerni. A második paraméter a projekciós henger sugara, azaz a felvételeket készítő képzeletbeli kamera objektum középpontjától vett távolsága. Egy tipikus beállításként használtam például a 60-as felvételszámot 20 méteres sugárral.

A kapott képek egy ideiglenes mappában tárolódnak, és egy OpenCV függvényeket felhasználó modul segítségével összeolvasztva adják a hengeres projekció végeredményét. Az első kísérletek során megfigyeltem, hogy az egyes, a szegmentált objektum körül található zajból eredő pixelek a projekció során jelentősen tudják torzítani a kapott képet, vékony csíkokként jelennek meg. Ezen kívül a Velodyne Lidar szenzorral felvett pontfelhők vízszintes csíkozottsága is olyan nem kívánt mintázatot visznek bele a folyamat végeredményébe, ami a későbbi mély tanuláskor felesleges alaksajátosság megtanulására ösztönözné a hálót, így csökkentve a hasznos reprezentációk megtanulásának mértékét. Az előbbiek miatt döntöttem úgy, hogy az egyes felvételek súlyozott összegének kiszámítása után az eredményen dilatációs és Gauss élsimítást végzek viszonylag nagy szűrűmérettel, így szabadulva meg a felesleges részletektől.

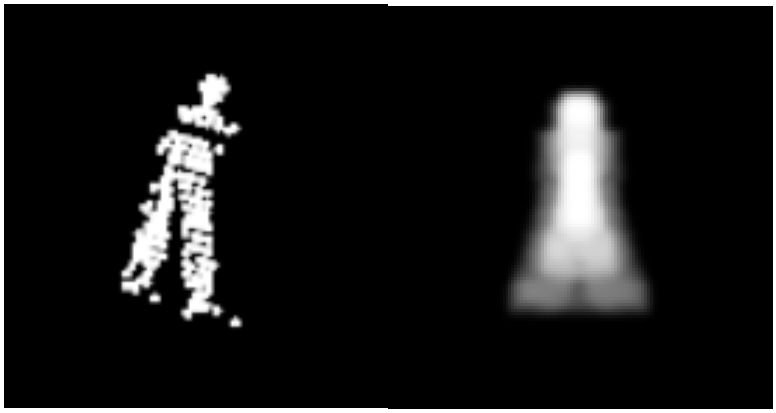
A további vizsgálódásokhoz több különböző paraméterbeállítás mellett legeneráltam az összes pontfelhő projekcióját. A beállítások felsorolása található az 5-1. Táblázat táblázatban, néhány vizualizációt különböző objektumok leképzéséről mutatnak az 5.1. ábra5.2. ábra5.3. ábra5.4. ábraábrák. A későbbiekben várható más beállítások tesztelése is.

5-1. Táblázat: Hengeres projekció különböző verzióinak paraméterei

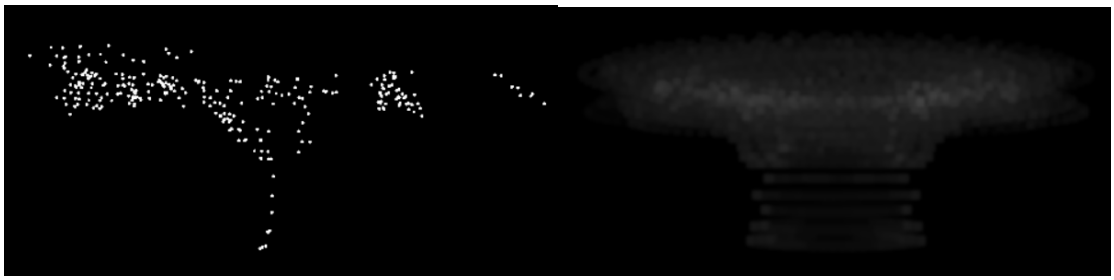
Verzió	Projekciók száma	Projekció sugara	Dilatációs méret	Gauss szűrű mérete
1	60	10	3x3	3x3
2	60	10	7x7	7x7
3	60	20	3x3	3x3
4	60	20	7x7	7x7



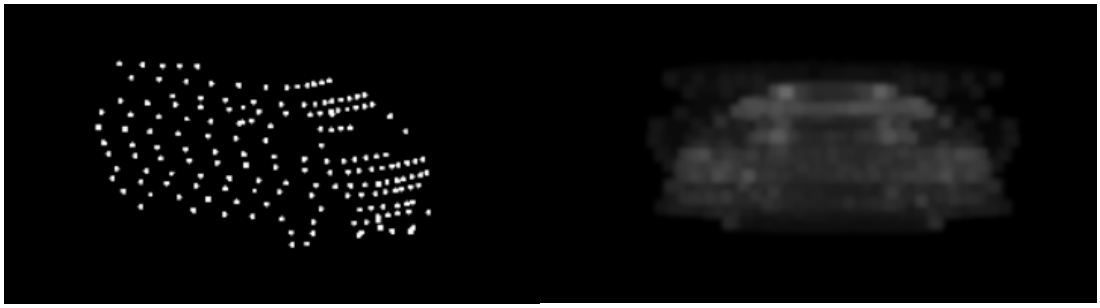
5.1. ábra: Autó pontfelhő objektumról készült hengeres projekció



5.2. ábra: Gyalogos pontfelhő objektumról készült hengeres projekció



5.3. ábra: Fa pontfelhő objektumról készült hengeres projekció



5.4. ábra: Kisteherautó pontfelhő objektumról készült hengeres projekció

5.2 Mély tanulás

A rendelkezésre álló mély tanulás programkönyvtárt felhasználva, korábbi tapasztalataimra hagyatkozva végeztem el a pontfelhőobjektumok osztályozására képes mély neurális modell tanításának optimalizálását elvégző keretrendszer implementálását. A továbbiakban két jelentős feladatot emelek ki és részletezek, az egyik a hengeres projekció kimeneteként kapott képi adatok tanítást megelőző feldolgozása, a másik a különböző architektúrák és hangolandó tanulási paramétereik kiválasztása.

5.2.1 Adatok előzetes kezelése

A mély tanulás, és általában a gépi tanulás egy alapvető fontosságú lépése az adatok előkészítése a tanításhoz. Ugyanazon komplexitású neurális modell tud jelentősen más teljesítményt produkálni ugyanazon bemeneten az előfeldolgozási folyamatoktól függően. Esetemben a tanítást nehezíti a kevés minta, így ez a lépés kritikus fontosságú az elérhető eredmény szempontjából.

Első lépés a képek tömörítése, amivel a hengeres projekció során nem foglalkozok. A tömörítés mértéke befolyásolja a tanulás számítási igényét és az elérhető pontosságot. A belátható idejű futtatást szem előtt tartva és a részletek túlzott elvesztését elkerülendő a 32x32 és a 64x64 pixelesre skálázás mellett döntöttem. Az előbbi méretet a tanulási paraméterek terének minél széleskörűbb bejárása során használtam, a nagyobb formátumú képekkel értelemszerűen csak kevesebb paraméterbeállítás tesztelésére van időm.

A képek tömörítése után az adathalmazt fel kell osztani. A minták számának limitáltsága miatt úgy döntöttem, hogy validációs adatok használata helyett az [1]-ben is használt tesztelési metodológiát követem. A 14 legnépesebb osztályból származó 588 pontfelhő objektumból álló adathalmaz eleve négy egyenlő részre van osztva úgy, hogy az egy objektumról készült felvétel nem szerepel két részben. Egy betanított klasszifikáló modell osztályozási pontosságát úgy határozták meg, hogy a teljes tanító-teszt folyamatot négyszer ismételték meg úgy, hogy a négy részből három alkotta a tanító, egy pedig a teszt adatokat. Én is így jártam tehát el, a tanító keretrendszernek paraméterként lehet megadni, hogy melyik csoportot használja tesztelésre. A validációs adatok használatát ki lehet váltani ezzel a módszerrel, hiszen elkerülhető a tanítási paraméterek egy konkrét teszthalmazra történő optimalizálása, a modellnek a teljes adathalmazon kell végső soron jól teljesítenie.

A következő lépés az adatok véletlenszerű megkeverése, majd normalizálása. Több gépi tanulással foglalkozó irodalom, online kurzus (például [48]) szentel nagy hangsúlyt a tanító algoritmus bemeneteként szolgáló adatok normálására. Ennek fő oka az ilyen algoritmusokban használt gradiens alapú optimalizáció, mely igazoltan jobb eredményeket ér el, ha a bemeneti sajátosságok, képek esetében pixelértékek, azonos nagyságrendűek, ugyanis ekkor a sajátosságok által alkotott tér szabályos alakú (két dimenziós esetben kör), ahol az említett gradiens módszerek gyorsabban és pontosabban találják meg az optimumot.

Többféle elterjedt megoldás is ismert, ezek közül én többek között az átlag alapú normalizálás két típusát teszteltem, az egyik az egyes pixelek értékeinek terjedelmét, a másik a szórását használja fel. Az előbbi leképezést a 5.1. egyenlet, utóbbit a 5.2. egyenlet adja meg. A fellelhető eredmények alapján a szórás alapú normalizálásnak tulajdonítottam a legnagyobb potenciált.

$$x' = \frac{x - \bar{x}}{\max(x) - \min(x)} \tag{5.1.}$$

$$x' = \frac{x - \bar{x}}{\sigma} \tag{5.2.}$$

A fentieken kívül teszteltem a lineáris skálázást, mint normalizációt. Ebben az esetben nem a pixelértékek átlagát, hanem azok minimumát vonom ki a bemeneti értékekből, majd elosztom a kapott eredményt a terjedelemmel. Egy skalárral való szorzással így beállítható a kapott adathalmazra jellemző terjedelem, ez alap esetben [0,1]. A leképezést az 5.3. egyenlet írja le. Kipróbáltam egy olyan módosítást is, amikor az adott pixel saját terjedelme helyett a globális maximummal és minimummal számítom a terjedelmet.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{5.3.}$$

5.2.2 Kijelölt architektúrák és tanítási paramétereik

Vizsgálódásom során két fő mély neurális modellre támaszkodtam, a többrétegű perceptronra és konvolúciós hálózatra. Az előbbi kiegészítéseként foglalkoztam az úgynevezett dropout módszer alkalmazásával is. Az egyes architektúrák tanítása során fontos paramétereket és jellemző értéktartományaikat tartalmazza az 5-2. Táblázattáblázat. Ezeken kívül általános, beállítás jellegű paraméterek a használt projekció verzió, valamint az alkalmazott adat normalizáció.

5-2. Táblázat: Tanítási paraméterek és jellemző értéktartományaik

Paraméter neve	**Jellemző értéktartomány**
Tanulási ráta (α)	0,001-0,5
Rejtett elemek száma (többrétegű perceptron és dropout esetén)	100-1500
Tanulási ráta csökkentési tényezője	0,95-1
Kötegméret	1-50
Iterációk száma	100-150
Dropout ráta (Dropout esetén)	0.2-0.8
Konvolúciós szerkezet (Konvolúciós hálózat esetén)	komplex paraméter

Az egyes paramétereik leírását, szerepét sorolom a továbbiakban:

Többrétegű perceptron paraméterek:

- **Rejtett elemek száma:**

A többrétegű perceptron struktúráját kizárólagosan meghatározó paraméter, mivel a bemeneti és kimeneti méret adottak. Méretének növelésével bonyolultabb reprezentációkat is képes megtanulni a hálózat, viszont növekszik a tanítás számítási igénye is. Értékét 100 és 1500 között vizsgáltam, ez alatt nem elegendő a reprezentációs képesség, nagyobb elemszám esetén a túl bonyolult számítás jelentősen lassítja a tanulási folyamatot.

- **Tanulási tényező (α):**

Gradiens alapú optimalizációs módszerek tanulási tényezője meghatározza azt a súlyt, amivel az adott tanítási ciklus hibáját figyelembe vesszük a hálózat paramétereinek frissítésekor. Általánosan megállapítható, hogy a legfontosabb tanulási paraméter minden mély neurális modell esetében, optimális értékének megtalálása kritikus fontosságú. Nagy értéke gyors tanítást okoz, azonban bizonyos érték felett a tanítás divergál, elszáll. Kis értéke pontosabb tanítást eredményez, azonban az optimumhoz lassabban konvergál. Mivel az ideális tanítás gyorsan és pontosan konvergál a minimális hibához, a tanulási tényező értékét célszerű nagyobb értékről indítani, és ciklusonként csökkenteni. A paraméterhangolás során a tanulási tényező kezdeti értékét optimalizáltam, a vizsgált értéktartományt a többrétegű perceptron esetén 0,001 és 0,5 közöttinek választottam.

- **Hanyatlási tényező:**

A tanulási tényezőnél említett iterációnkénti értékcsökkentés mértékét meghatározó tényező a hanyatlási tényező. Értéke nem túl kicsi, hiszen akkor túl gyorsan kis tanulási tényezőnk lenne, ami a már ismertetett lassúsággal jár. A hanyatlási tényező értékétől függetlenül minden algoritmus rendelkezik egy minimális tanulási tényező értékkel, hogy sok iteráció esetén se degradálódjon hatástalanná a tanulás. A hanyatlási tényezőt 0,95 és 1 között teszteltem.

- **Iterációk száma:**

A tanítás ciklusainak száma, ezt konstans 150-nak állítottam be, mert így minden paraméter-beállítás esetén biztosan elérte az adott hibaminimumot a tanuló algoritmus.

- **Kötegméret**

A tanítások során nem minden esetben használtam sztochasztikus gradiens módszer alapú optimalizációt, hanem kötegelt gradiens módszerrel is kísérleteztem. Ennek lényege, hogy a bemeneti adatokat kötegekbe rendezi, és ezeken a csoportokon egyszerre hajtja végre a tanítás egy ciklusát, nagyban gyorsítva így a tanulást. A kötegmérettel e csoportok méretét hangoltam. Értékét 1 és 50 között teszteltem. Az 1-es kötegméret megegyezik a sztochasztikus gradiens alapú optimalizációval.

Konvolúciós neurális háló paraméterek:

A konvolúciós neurális hálók tanítása mindössze egy tekintetben különbözik a többrétegű perceptronétól, ez pedig a hálózati struktúra.

- **Hálózati struktúra**

A konvolúciós neurális hálókat három különböző típusú rétegből állítottam össze. Elméletben nincsen felső korlátja a kombinációknak, azonban bizonyos szempontok alapján redukálni kell a struktúrák számát. Korábbi munkáim során már leírtam számos variánst, azonban a jelen dolgozatban használt minták dimenziója eltér a korábbitól, így a rendelkezésemre álló struktúrákat refaktorálnom kell. Az alapötlet mindig hasonló, a konvolúciós és átlagoló merítő rétegeket váltakozva kell használni, figyelve a csatlakoztatott rétegek kompatibilitására a ki-és bemeneti dimenziók beállításával.

Dropout paraméterek:

A dropout módszer tesztelése többrétegű perceptron hálózaton keresztül történik, így paraméterek tekintetében osztozik a két tanítás. Egyetlen új paraméter van, mégpedig a dropout tényező.

- **Dropout tényező**

A dropout módszer lényege, hogy a neurális hálózat rejtett rétege után az aktivációk értékeit véletlenszerűen eldobjuk tanítás közben, így kényszerítve a hálózatot arra, hogy minél jobban tanuljon megáltalánosítani. Tesztelési időben nincs dropout, a tanítás alatti hatását egy skálázással kell helyettesíteni, hogy átlagosan jó aktivációs értékeket kapjunk. Az értékek eldobásának valószínűségét, és az említett skálázás értékét a dropout tényező adja meg. Értékét 0,2 és 0,8 között teszteltem.

6 Eredmények

A megalkotott keretrendszer segítségével már kiértékelhető az egyes architektúrák teljesítménye, a beállítások és tanulási paraméterek hatása az osztályozás pontosságára. A tesztelés során az optimális paraméterek megtalálására két módszer van, a szisztematikus és a véletlenszerű keresés. Előbbinél a paraméterek által alkotott teret rácspontokra osztjuk, majd minden esetben lefuttatjuk a tanító algoritmust és kiértékeljük az osztályozási pontosságot. Véletlenszerű keresésnél csak a paraméterek értéktartományát adjuk meg, mindent véletlenszerűen mintavételezünk ezekből a tartományokból. Az optimumot mindkét módszer képes megtalálni, én a szisztematikus keresést választottam. Ez a módszer jóval lassabb, de az utólagos, részletes kiértékésre, az egyes paraméterek hatásainak analizálására jóval hatékonyabb részeredményeket produkál.

A hálózatok közül a dolgozat beadásakor még csak a többrétegű perceptron elemzését kezdtem el, a dropout módszer és a konvolúciós hálózatok tesztelése a projekt későbbi szakaszának feladatai.

6.1 Többrétegű perceptron

A többrétegű perceptron kiértékelését még nem végeztem el teljes mértékben, egyelőre csak egy, általam legnagyobb potenciálónak vélt hengeres projekció verziót és a szórás alapú átlag normalizációt használtam. Elvégeztem a paramétertérben történő optimumkeresést, a legjobb elért eredményeket, és a hozzájuk tartozó paramétereket tartalmazza a táblázat. Kiemelném, hogy az elért pontosság minden esetben nagyobb, mint a dolgozatom alapötletét adó cikkben ([1]) elért maximális, 67%-os eredmény. Feltehető, hogy a jobb reprezentációs képességgel rendelkező konvolúciós hálózat alkalmazása esetén ennél is jobb eredmény érhető el.

Az eredményekhez hozzá kell tennem, hogy azok nem a korábban vázolt validációs módszerrel számítódtak, hanem mindig ugyanazt az adathalmazt használtam tesztelésre. Ez hibás eredményre vezethetett, így a projekt következő lépése a folyamat újbóli elvégzése a korrekt tesztelési metódussal. Néhány esetet vizsgálva megállapítottam, hogy ezzel a módszerrel is elérhető, sőt meghaladható az ACFR cikkben elért eredmény.

6-1. táblázat: Többrétegű perceptron modellel elért osztályozási pontosság

Rejtett elemek száma	**Tanulási tényező**	**Tanulási tényező csökkentési rátája**	**Kötegméret**	**Globális pontosság**
1000	0,01	1	20	**74,83**
600	0,06	1	30	**73,55**
1500	0,01	1	50	**72,90**
800	0,06	1	30	**72,26**

7 Befejezés

Az irodalomkutatás keretei között bemutattam a témához kapcsolódó hardver, a LIDAR eszköz érzékelési elvét, a LIDAR szenzorok főbb típusait és alkalmazásait. Ugyanitt vizsgáltam különböző mély tanulás alapú algoritmusok elméleti alapjait, objektumfelismerésben elért eddigi eredményeiket. A különböző módszerek előnyeit és hátrányait összevetettem, kiválasztottam a munkám során használandókat.

A feladat megvalósítása egy összetett szoftver implementálását követeli meg, ehhez segítségként szabadon felhasználható programkönyvtárakat kerestem. A felhasználásra jelölt szoftvereket bemutattam, vizsgáltam azok elérhető, és számomra hasznosnak tűnő metódusait.

Megterveztem a teljes program keretrendszerét, az osztályozási folyamat főbb részeit. A nagy problémát okozó pontfelhő adatok és mély tanulási algoritmusok bemenetének kompatibilitási problémáinak több lehetséges megoldását is taglaltam, azok előnyeit és hátrányait összevetettem és választottam közülük. Kidolgoztam egy saját eljárást a nyers pontfelhő adatok kétdimenziós reprezentálására, egy hengeres projekciós eljárást. A program befejezése utáni tesztelési folyamat keretét is megadtam, kijelöltem azokat a módszereket és mérőszámokat, amelyek alapján a különböző algoritmusokat összehasonlíthatom.

Elvégeztem az objektumok osztályozására képes keretrendszer teljes implementációját, különösen figyelve a rugalmas és átlátható tesztelhetőségre. Elkezdtem a különböző paraméterek mellet futtatott tanítások eredményeinek kiértékelését.

Összevetve, jelen dolgozatban sikerült egy összetett, a mély tanulás városi LIDAR pontfelhő objektumok osztályozásában elérhető potenciáljának meghatározására szolgáló program elméleti alapjait lefektetni, egy lehetséges működését megtervezni, valamint egy implementációját elkészíteni. A projekt következő lépése tanítási paraméterek változtatása melletti minél több tesztelés, és azok eredményeinek kiértékelése. Hátravan még a megalkotott tanító program alapján egy valós helyzetben is használható, objektum meghatározó szoftver készítése, annak használhatóságának kiértékelése.

8 Irodalomjegyzék

1] M. De Deuge, A. Quadros, C. Hung és B. Douillard, „Unsupervised Feature Learning for Classification of Outdoor 3D Scans," in *University of New South Wales*, Sydnes, Australia, 2013.

2] C. Benedek és O. Józsa, „Reconstruction of 3D Urban Scenes Using a Moving Lidar Sensor," MTA SZTAKI, Budapest, 2013.

3] P. Cignoni, „Meshing Point Clouds," Meshlab, 7. szeptember 2009. [Online]. Available: http://meshlabstuff.blogspot.hu/2009/09/meshing-point-clouds.html. [Hozzáférés dátuma: 16. május 2016.].

4] „The PCD (Point CLoud Data) file format," pcl, [Online]. Available: http://pointclouds.org/documentation/tutorials/pcd_file_format.php#pcd-file-format. [Hozzáférés dátuma: 16. május 2016.].

5] „The Stanford 3D Scanning Repository," [Online]. Available: http://graphics.stanford.edu/data/3Dscanrep/. [Hozzáférés dátuma: 16. május 2016.].

6] B. Nagy, „Utcai objektumok felismerése Lidar pontfelhősorozatokon - szakdolgozat," PPKE, 2014.

7] „What is LIDAR?," NOAA, [Online]. Available: http://oceanservice.noaa.gov/facts/lidar.html. [Hozzáférés dátuma: 16. május 2016.].

8] laser-distance-measurer, „Lecture 14: Principles of active remote sensing: Lidars and lidar sensing of aerosols, gases and clouds," 2015.

9] G. Goyer és R. Watson, „The Laser and its Application to Meteorology," *Bulletin of the AMerican Meteorolgical Society,* %1. kötet44, pp. 564-575, 1963..

10] G. Vosselman és H.-G. Maas, Airborne and terrestrial laser scanning, Whittles Publishing, 2012.

M. Gonglach, „How Police Laser Guns Work,” 17. december 2012.. 11] [Online]. Available: http://www.laserjammer.net/2012/how-police-laser-guns-work/. [Hozzáférés dátuma: 16. május 2016.].

„HDL-64E Data Sheet,” Velodyne, 2014.
12]

Velodyne, „Velodyne's HDL-64E: A High definition LIDAR sensor for 13] 3-D applications,” Velodyne, 2007.

Y. LeCun és M. Ranzato, „Deep Learning Tutorial,” Atlanta, 2013.
14]

Y. Bengio, Learning Deep Architectures for AI, 2009.
15]

A. Krizhevsky, I. Sutskever és G. E. Hinton, „ImageNet Classification 16] with Deep Convolutional Neural Networks,” 2013.

LISA lab, „Deep Learning Tutorials,” University of Montreal, 2015.
17]

N. Srivastava, G. Hinton, A. Krizehvsky, I. Sutskever és R. 18] Salakhutdinov, „Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machnine Learning Research,* %1. kötet15, pp. 1929-1928, 2014.

Y. LeCun, L. Bottout, Y. Bengio és P. Haffner, „Gradient-Based Learning 19] Applied to Document Recognition,” IEEE, 1998.

A. Karpathy és F.-F. Li, „Convolutional Neural Networks 20] (CNNs/ConvNets),” 2015. [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Hozzáférés dátuma: 4. december 2015.].

A. Ng, J. Ngiam, C. Yu Foo, Y. Mai és C. Suen, „UFLDL Tutorial - 21] Autoencoders,” április 2013. [Online]. Available: http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/. [Hozzáférés dátuma: 4. december 2015.].

A. Mordvintsev, C. Olah és M. Tyka, „DeepDream - a code example for
22] visualizing Neural Networks,” Google, 2015.

L. Deng és D. Yu, Deep Learninig Methods and Applications, Foundation
23] and Trends, 2014.

P. Vincent, H. Larochelle, I. Lajolie, Y. Bengio és P.-A. Manzagol,
24] „Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion,” *The Journal of Machine Learning Research,* %1. kötet11, pp. 3371-3408, 2010.

L. Deng, „Three Classes of Deep Learning Architectures and Their
25] Applications: A Tutorial Survey,” Microsoft Research, Redmond, USA, 2014.

H. Larochelle, „An empirical evaluation of deep architectures on
26] problems with many factors of variation,” *Proc. 24th Int. Conf. Machine Learning,* pp. 473-480, 2007.

A. Krizhevsky, „Convolutional Deep Belief Networks on CIFAR-10,”
27] Toronto, 2014.

V. Mnih, K. Kavukcouglu, D. Silver, A. Graves, I. Antonoglou, D.
28] Wierstra és M. Riedmiller, „Playing Atari with Deep Reinforcement Learning,” 2013.

Y. LeCun, C. Cortes és C. Burges, „The MNIST database,” [Online].
29] Available: http://yann.lecun.com/exdb/mnist/. [Hozzáférés dátuma: 4. május 2015.].

D. Ciresan, U. Meier és J. Schmidhuber, „Multi-column Deep Neural
30] Networks for Image Classification,” *Computer Vision and Pattern Recognition,* pp. 3642-3649, 2012.

„PCL Tutorials,” [Online]. Available:
31] http://pointclouds.org/documentation/tutorials/. [Hozzáférés dátuma: 30. április 2015].

T. Nomi, „tiny-cnn: A C++11 implementation of deep learning
32] (convolutional neural networks),” 2013. [Online]. Available: https://github.com/nyanp/tiny-cnn. [Hozzáférés dátuma: 30. április 2015.].

Y. Sugomori, „Github - Deep Learning,” 2013.. [Online]. Available:
33] https://github.com/yusugomori/DeepLearning. [Hozzáférés dátuma: 8. december 2015.].

R. Bogdan Rusu és S. Cousins, „3D is here: Point Cloud Library (PCL),”
34] USA, 2011.

PCL, „What is PCL?,” PCL, [Online]. Available:
35] http://pointclouds.org/about/. [Hozzáférés dátuma: 17. május 2016.].

PCL, „Module io,” PCL, [Online]. Available:
36] http://docs.pointclouds.org/trunk/group__io.html. [Hozzáférés dátuma: 17. május 2016-].

PCL, „Module visualization,” PCL, [Online]. Available:
37] http://docs.pointclouds.org/trunk/group__visualization.html. [Hozzáférés dátuma: 17. május 2016.].

PCL, „PCLVisualizer,” PCL, [Online]. Available:
38] http://pointclouds.org/documentation/tutorials/pcl_visualizer.php#pcl-visualizer. [Hozzáférés dátuma: 17. május 2016.].

„Wikipedia - OpenCV,” [Online]. Available:
39] http://en.wikipedia.org/wiki/OpenCV. [Hozzáférés dátuma: 30. október 2014.].

The Linux Information Project, „BSD License Definition,” 19. április
40] 2004. [Online]. Available: http://www.linfo.org/bsdlicense.html. [Hozzáférés dátuma: 3. november 2014.].

G. Bernát, „Mat - The Basic Image Container,” [Online]. Available:
41] http://docs.opencv.org/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html#matthebasicimagecontainer. [Hozzáférés dátuma: 30. október 2014.].

P. Tamás, „Képfeldolgozás előadásvázlat,” BME, 2014.

42]

„OpenCV online documentation - Smoothing,” [Online]. Available:
43] http://docs.opencv.org/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html#smoothing. [Hozzáférés dátuma: 4. november 2014.].

Y. Bengio, „Practical Recommendations for Gradient-Based Training of
44] Deep Architectures,” 2012.

A. Quadros, J. Underwood és B. Douillard, „Sydney urban objects
45] dataset,” ACFR, 2013. [Online]. Available: http://www.acfr.usyd.edu.au/papers/SydneyUrbanObjectsDataset.shtml. [Hozzáférés dátuma: 18. május 2016.].

R. Zsámboki, „Objektumfelismerés városi Lidar pontfelhőkön,” 2015.

46]

A. Quadros, J. Underwood és B. Douillard, „An occlusion-aware feature
47] for range images,” *Robotics and Automation,,* pp. 4428-4435, 2012.

A. Ng, „Coursera - Machine Learning,” Stanford, 2014. [Online].
48] Available: https://www.coursera.org/learn/machine-learning. [Hozzáférés dátuma: 09. december 2015].