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

Mélységbecslő- és szemantikus szegmentáló mély neurális hálók fejlesztése és alkalmazása

TDK dolgozat

Készítette:

Bogár György Richárd

Konzulens:

Szántó Mátyás

2021

Tartalomjegyzék

Kivonat **i**

Abstract **ii**

1. Bevezetés **1**

1.1. Feladat háttere 1

1.2. Feladat és annak lehetséges megoldásai 2

1.2.1. Mélységképek előállítási módjai 2

1.2.2. Neurális hálózatok 3

1.3. A dolgozat témája 6

1.4. A dolgozat felépítése 7

2. Az AdaBins mélységbecslő **8**

2.1. Az AdaBins felépítése 8

2.1.1. A U-net modul 9

2.1.2. Az AdaBins modul 9

2.1.3. Tanítási hiperparaméterek és konfiguráció 11

3. Az AdaBins háló fejlesztése és tanítása **13**

3.1. U-net modul 13

3.1.1. Eredmények értékelése 14

3.2. Teljes AdaBins háló 15

3.2.1. Fejlesztés 15

3.2.2. Tanítás NYU Depth Dataset V2 adathalmazon 16

3.2.3. NYU Depth Dataset V2 tanítás eredményeinek értékelése . . . 17

3.2.4. Tanítás KITTI depth adathalmazon 18

3.2.5. KITTI depth tanítás eredményeinek értékelése 18

4. Szegmentáló és hibrid megoldások az AdaBins architektúrájából kiindulva 20

4.1. Attention Segmentation háló 20

4.1.1. Kialakítás 20

4.1.2. Kezdetleges tanítási eredmények KITTI segmentation adathalmazon 21

4.2. AdaBins segmentetion hibrid architektúra 21

4.2.1. Kialakítás 22

4.2.2. Tanítási eredmények Apolloscape Scene Parsing adathalmazon 22

5. Összefoglalás és konklúzió 25

5.1. Összefoglalás 25

5.2. Adathalmazok összehasonlítása tanítás szempontjából 26

Irodalomjegyzék 28

Kivonat

Dolgozatom egyik fő témája egy monokuláris mélységbecslő neurális háló fejlesztése, illetve tanítása. Ennél a feladatnál több tanító adatbázis is alkalmazásra került, melyek különböző technikákkal készültek, és ebből kifolyólag eltérő jellemzőkkel is bírnak. Míg a beltéri területeket megjelenítő képek sűrűn tartalmaznak mélységadatokat, addig van olyan kültéri adatbázis, melyhez ritkán és jellemzően csak a kép bizonyos részén találhatóak érvényes adatok. Munkám során összehasonlítom ezen különbségeket a tanítások során elért hibafüggvények alakulása és a kimenet kvalitatív minősége alapján, illetve az egyik gyakran használt mélységi adathalmazon végrehajtott tanítás eredményeit több metrika szerint is kiértékelem.

A mélységbecslő architektúrájából kiindulva egy, a szemantikus szegmentáció problémáját megoldani célzó hálót is terveztem. Ennek kezdeti, kevés képen történő tanításait követően azt a következtetést vontam le, hogy a háló nagy valószínűséggel képes lenne a generalizálásra, ezért a nagyobb adathalmazon történő tanítás mellett döntöttem, melynek eredményeit szintén összevetem más algoritmusok teljesítményével.

A mélységbecslő és a szegmentáló hálók közös alapjából kifolyólag egy olyan architektúra kialakítására és tanítására is kísérletet teszek, mely egyszerre igyekszik megbecsülni egy bemeneti képből a mélységet, illetve a pixelekhez tartozó szemantikus osztályokat. Ennek tanításához egy általam kigondolt, egyszerű hibrid hibafüggvényt alkalmazok annak érdekében, hogy a háló mindkét részfeladat megoldását a lehető legoptimálisabban tanulja meg.

Abstract

One of the main subjects of my work is the development and training of a monocular depth estimator neural network. In this task multiple training datasets were being used which were made with different techniques therefore having different attributes. While one indoor dataset includes dense depth values in the maps, another one with outdoor images has sparse depth maps and is valid even on a specific area of the image. During my work I compare the different outcomes of these trainings using the saved loss function diagrams and the qalitative comparison. Besides I implement multiple evaluation metrics to compare my results with other implementations in a qantitative way.

Using the depth estimator architekture as a starting point, I designed a neural network which aims to solve the problem of semantic segmentation. After training this architecture I came tot he conclusion that this structure can be a promising one and further training is needed on larger number of training data. The results of these trainings are also subjects of comparison with other solutions in the literature.

Using the common base of the depth estimator and the semantic segmentation networks I design and train an architecture which solves the two problems simultaneously. To achives the best training I present a hybrid loss function which aims to achive the best training results in both tasks.

1. fejezet

Bevezetés

1.1. Feladat háttere

Napjainkban a képfeldolgozással kapcsolatos aktívan kutatott területek egyike a Visual Simultaneous Localization And Mapping (VSLAM), mely a kamera helyzetének meghatározásával és az adott lokális környezet feltérképezésével foglalkozik. Ezt a problémakört többféleképpen is feloszthatjuk. Az egyik általános osztályozási szempont az információ kinyerésének módja a képekből.

A ritka típusú Visual SLAM eljárások esetén a képeken olyan egyedi képjellemzőket keresünk, melyeket képesek lehetünk követni több egymást követő képen is, illetve elmozdulásuk alapján információt szerezni a kamera elmozdulásáról a két kép között. Ezek az úgynevezett kulcspontok általában valamilyen sarokszerű tulajdonsággal rendelkező pontok, melyek a kép jól textúrált területeiről kerülnek kiválasztásra. Ebben a kategóriában egy széles körben elfogadott és használt robusztus megoldás az úgynevezett ORB-SLAM [9], mely a képeken ORB leírókat [14] használ a kulcspontok megállapítására. A képek kódolt tárolásához (mapping) és a lokalizációhoz az ilyen SLAM algoritmusoknál egy eredményes megoldás lehet a Bag of Visual Words [7] alkalmazása.

A másik csoport ezen az osztályozáson belül a sűrű SLAM megoldások, melyek az egész képet felhasználják annak érdekében, hogy megoldják a SLAM feladatot. Itt általában fotometrikus hibák minimalizálásával határozzák meg a képek közti elmozdulásokat. A fotometrikus hibák mellett bizonyos esetekben a geometriai rekonstrukció hibájának minimalizálása is megjelenik. Erre példa lehet az úgynevezett CodeSLAM [3], mely a kamera képei mellett mélységképeket is felhasznál arra, hogy

a térképkészítéshez kiválasztott úgynevezett kulcsképeket jól optimalizált kódokká alakítsa, melyek a lokalizáción kívül a geometriai jellemzők rekonstruálására is hasznosíthatóak.

Bár az elnevezés közvetlenül nem tartalmazza, a SLAM eljárások egyik legfontosabb része az odometria, melynek során az ágens haladása közben követésre kerül annak relatív elmozdulása az idő függvényében. A vizuális odometria megoldható sűrű eljárás esetén a sztereó probléma olyan módú átalakításával, melynek során a két, időben egymás után készült képet vesszük, mint sztereó képpár és a két kép közti fotometrikus hiba minimalizálásával igyekszünk megállapítani a kamera elmozdulását. Egy másik megközelítés, amikor a képek közti optikai áramlási mező meghatározásával információt szerzünk a pixelek eltolódásáról a mintavételek közt, és ezt próbáljuk visszavezetni a kamera póz megváltozására. [15] A megoldás javítására a vizuális odometria esetében is alkalmazhatnak mélységbecslő eljárásokat, így a becsült elmozdulásértékeket geometriai rekonstrukció szempontjából is képes optimalizálni. Ezek alapján a SLAM egyes komponenseit úgy is megválaszthatjuk, hogy a mélységbecslést végző modul több helyen is felhasználható legyen, ezáltal növelve annak kihasználtságát.

1.2. Feladat és annak lehetséges megoldásai

1.2.1. Mélységképek előállítási módjai

Mint az a fentiek alapján látható, a mélységképek a SLAM problémakörön belül maradva is sokoldalúan felhasználható információt hordoznak a kamera által megfigyelt lokális környezet geometriájáról. Ezen képeket több csoportba sorolhatjuk azok generálási módja szerint.

Az egyik meglehetősen robusztus működést mutató megoldás a sztereó alapú mélységkép generálás, melyeknél két, egymástól előre meghatározott pozícióban és orientációban elhelyezett kamerát alkalmaznak, hogy a köztük meghatározott távolság és az általuk rögzített képek segítségével megállapítsák a képeken látható pontok távolságát az elrendezéstől. Ezeknek egyik hátránya, hogy a sztereó elrendezésből adódóan a kamerához túl közel kerülő objektumokon már esetenként nehezebben tudják megállapítani az összetartozó képpontokat, mivel közelebb már az objektum más más részei jobban látszódnak a két kamerán. A másik ehhez hasonló probléma,

amikor az egyik kamerát eltakarja valami, aminek következtében a rendszer részben vagy egészében nem képes a mélység meghatározására.

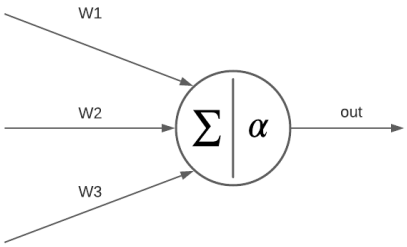
Egy másik megoldás, amikor a mélységképet meghatározó rendszer aktív módon bizonyos jeleket bocsát ki, és azok visszaverődése alapján prónálja meghatározni a mélységeket. Erre a módszerre példák az infra alapú mélységkamrák, illetve a LIDAR (Light Detection and Ranging) mélységkamerák.

A harmadik módszer a monokuláris mélységbecslés. Itt az egyik megoldás, amikor az időbeli sztereó probléma megoldásával igyekeznek mélységképek megállapítására pl. valószínűség alapú optimalizációs eljárásokkal. A másik lehetőség monokuláris esetben, amely napjainkban a mélységbecslés egy aktívan kutatott területe, amikor ténylegesen egy bemeneti kép alapján igyekszünk megbecsülni a mélységértékeket. Itt hagyományos módszerekkel alkalmazhatunk olyan a priori ismereteket, mint a látott környezeten felismert geometriai alakzatok felismerése és használata a mélység megállapításához, ez azonban általában erős korlátokba tud ütközni. Másik megoldásként használhatjuk a többek közt a képfeldolgozás területén is fontos iránynak számító mély neurális hálók alkalmazását, melyek bizonyítottan képesek jó eredményeket elérni a monokuláris mélységbecslés- és az ahhoz hasonló bonyolultságú feladatok megoldásánál. Ennek egyik előnye, hogy felügyelt tanulás esetén képesek lehetünk egy adott háló struktúrával megtanulni sokkal drágább, illetve bonyolultabb műszerek kimeneteinek előállítását alacsonyabb lokális hardverigény mellett. Erre példa, hogy a megfelelő működési körülmények közt jó mélységértékeket szolgáltató LIDAR kimenetéhez hasonló értékeket is képesek lehetünk tanulással előállítani egyetlen bemeneti kép alapján.

1.2.2. Neurális hálózatok

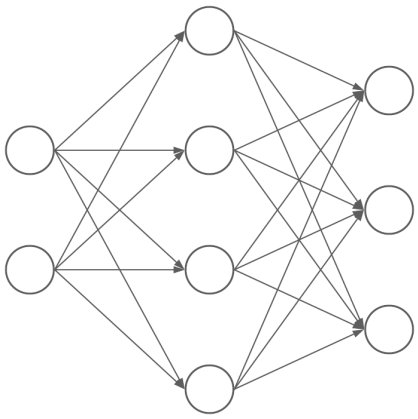
A neurális hálók működésének alapeleme a neuron. A legegyszerűbb esetet vizsgálva egy neuron tartalmaz egy összeadót, illetve egy nemlineáris elemet.

A bemenetére érkező értékeket az azokhoz súlyokkal megszorozva adja össze. Ezen súlyok kiemelt jelentőségűek a neurális hálók vizsgálatánál, ugyanis ezeket hangoljuk az úgynevezett tanítási szakaszban. A nemlineáris elemek közt megemlíthetőek például a szigmoid, illetve a tanh függvények, azonban az egyik napjainkban igen népszerű nemlinearitás az úgynevezett ReLU (Rectified Linear Unit). Ez utóbbi egy igen egyszeű függvény, amely nulla vagy annál kisebb bemenetre nullát, annál nagyobb bemenetre pedig a bemenetet adja vissza eredményként. A nemlineáris elemek szintén kiemelt jelentőségűek, mivel ezek hiányában nem lennénk képesek nagy



1.1. ábra. Egy neuron. Belül az összeadás és a nemlineáris függvény található

és komplex többrétegű hálók kialakítására, hiszen nélkülük a csupán egymás utáni szorzások és összeadások a bemenet és a kimenet között egy eredő szorzásra és összeadásra redukálhatóak, vagyis csak egy lineáris összefüggés tudnánk velük modellezni. Az egy elemi neuronnál bonyolultabb hálókat úgy alakítják ki, hogy több ilyen neuront helyeznek egymás fölé, melyeknek a bemenetei ugyanazok, azonban mind egyedi saját súlyaival szorozza ezeket a bemeneteket. Több ilyen neuron egymás felett vertikálisan elhelyezve (ha a bemenet a bal oldalon érkezik a hálóba és a kimenet a jobb oldalon érhető el) tesz ki egy neurális háló réteget.



1.2. ábra. Három neurális háló réteg

Több ilyen réteget egymás után elhelyezve a nemlinearitásoknak köszönhetően egy komplex függvénykapcsolat írható le a bemenetek és a kiemenet(ek) között. Ezen az alap neuronon alapuló struktúrán kívül a mai megoldások számos bonyolultabb és feladatspecifikusabb rétegeket is tartalmaznak, melyek segítségével jobban kiszűrhetőek képi tulajdonságok, vagy éppen az időbeliség reprezentációját teszik lehetővé. Ahhoz, hogy a neurális hálózatok világában is hatékonyan alkalmazhatóak legyenek a mélytanuló algoritmusok, nagyban hozzájárult a konvolúciós rétegek kialakítása, melyek a képfeldolgozásban már megszokott és gyakran használt konvolúciós kernelek súlyait igyekszik a feladat szempontjából legjobban optimalizálni. A neurális

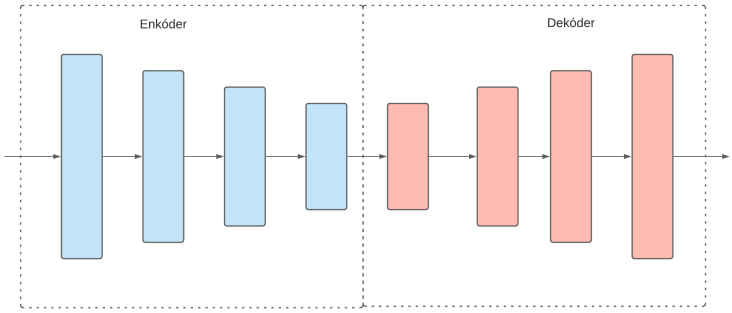
hálózatokról általánosan elmondható, hogy nagy mennyiségű tanító adatra van szükséges ahhoz, hogy ezen tanítható paramétereket a feladat megoldására a legjobban optimalizálni tudjuk. Ezen kívül fontos jellemzője ezeknek a megoldásoknak, hogy a feladatspecifikus modellezés helyett egy komplex, de általános struktúrát alakítunk ki, amely a tanítóadatokra és a feladatra nézve a legoptimálisabb megoldást igyekszik elérni. Ezzel képes lehet a háló a tanulás folyamán olyan rejtett információkat is kinyerni a bemenetből, amelyre az ember alkotta problémamodell nem terjed ki, ami esélyt ad a megnövekedett pontosság, illetve robosztusság elérésére.

A neurális háló alapú mélytanuló algoritmusokat a megoldandó probléma szempontjából két nagy csoportra szoktuk osztani. Az első az osztályozás, amely során a bemeneti adatokat valamilyen előre meghatározott osztályba próbáljuk sorolni. Erre az egyik legegyszerűbb példa kézzel írott számok felismerése és osztályozása 0-9 értéktartományban [4]. A másik, jóval általánosabbnak mondható feladat a regresszió, amikor egy folytonos kimeneti értéket szeretnénk megbecsülni a bemenet alapján. Klasszikusan a mélységbecslés is ebbe a feladatkörbe sorolható.

Bemeneti adatok alapján két fontosabb eljárást különböztethetünk meg.

1. Az első és egyszerűbb módszer, amikor ismerjük az elvárt kimenetet, ennek az előállítását szeretnénk megtanítani a neurális hálónknak. Ezt felügyelt tanításnak (supervised learning) nevezzük. Felügyelt tanulás lehet például a mélységbecslés is, amennyiben egy másik módszer (pl. LIDAR) eredményeihez szeretnénk közelíteni a hálónkkal, és ezeket az eredményeket használjuk fel a tanításhoz.

2. A másik nagyobb kategória a felügyelet nélküli tanulás, amely során az elvárt ideális kimenet nincs előre meghatározva, csak valamilyen hibafüggvény alapján ellenőrizzük a háló működését. Ez a tanítási módszer azért lehet előnyös a felügyelt módszerhez képest, mivel itt nincs szükség tanító adathalmazra, csupán bemeneti adatok és egy megfelelő hibafüggvény kell ahhoz, hogy különféle környezetekben elő tudjuk állítani az optimális eredményt. A felügyelet nélküli tanítás témaköréhez kapcsolható például az úgynevezett autóenkóder architektúra, ami egy enkóder és egy dekóder kaszkád kapcsolásából tevődik össze, és egy jellemző feladata lehet, hogy a kimenetén minél jobban képes legyen a bemenet rekunstrukciójára. Ennek hasznossága abban nyilvánul meg, hogy sikeres tanítás esetén a középen található bottleneck rétegeknél az adathalmaz minden képére egy tömör, redukált reprezentáció előállítására lesz alkalmas a háló.



1.3. ábra. Autóenkóder architektúra

3. Érdemes megemlíteni egy harmadik, az előző kettőhöz kevésbé szorosan kapcsolódó mélytanuló eljárást, a megerősítéses tanulást. Ezt olyan feladatkörökben használják általában, mint döntési, irányítási problémák megoldása. Tanítása jutalmazással történik és az algoritmus ennek a jutalomnak a maximalizálására törekszik. [11]

1.3. A dolgozat témája

Munkám fő fókuszában egy mélységbecslő mély neurális háló tanulmányozása és tanítása áll. Az itt felmerülő nehézségek megoldása komoly tapasztalatszerzési lehetőségként szolgált a mély neurális hálók fejlesztésének és tanításának területén.

Egy másik fontos képfeldolgozási terület, melyben szintén jelennek meg deep learning alapú megoldások, a szemantikus szegmentáció problémaköréhez kapcsolható. Publikációkban olvasható tapasztalatok [17], illetve a struktúra alkalmasságának megfontolását követően módosítottam a tanulmányozott mélységbecslő struktúrát, hogy minél jobban képes legyen a szegmentáció problémájának megoldására. Az eközben szerzett tapasztalatok, eredmények bemutatása szintén a dolgozat tárgyát képezi.

Végül a mélységbecslő és a szegmentáló háló kombinálásaként kísérletet tettem egy két kimenetű háló tanítására azzal a megfontolással, hogy a két, egymástól nem távol álló feladatkör eggyüttes megoldása javíthatja az eredményeket mindkét területen.

1.4. A dolgozat felépítése

A dolgozat elején bemutatásra kerül a tanulmányozott mélységbecslő, illetve az annak fejlesztési- és tanítási folyamatai során nyert tapasztalatok. Ezt követően az eredmények prezentálása következik különböző körülmények között. A dolgozat második felében a szegmentáló-, majd a hibrid mélységbecslő-szegmentáló hálók kerülnek bemutatásra a fenti szempontok alapján.

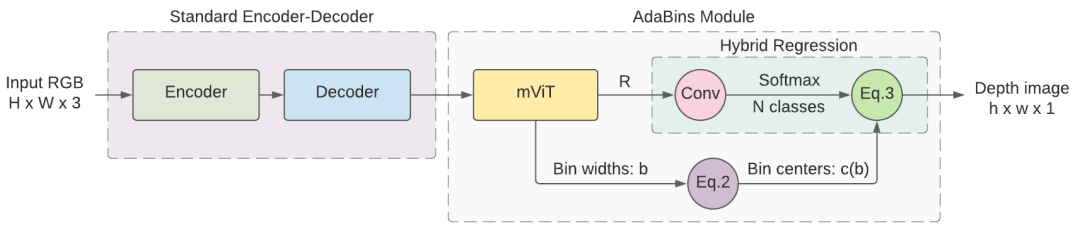
2. fejezet

Az AdaBins mélységbecslő

Munkám első részében egy olyan mélységbecslő mély neurális háló fejlesztésével, illetve tanításával foglalkoztam, melynek felépítése és publikációja alapján elmondható, hogy érdekes és továbbgondolható struktúráról van szó. Ez a háló az AdaBins [2], melynek az egyik legnagyobb különlegessége, hogy a mélységbecslés klasszikusan regressziónak tekinthető feladatát osztályozási problémára vezetik vissza. Ezt úgy teszik, hogy az adott adathalmaz által reprezentált mélységtartományt úgynevezett binekre osztják. A publikáció alapján ezen binek legoptimálisabb eloszlását úgy érjük el, ha azt az adott képhez adapítvan határozzuk meg.

2.1. Az AdaBins felépítése

A következő pontokban bemutatom a tanulmányozott, fejlesztett és tanított AdaBins mélységbecslő felépítését. Az AdaBins alapját egy U-net képzi. Ennek kimene-



2.1. ábra. Az AdaBins mélységbecslő felépítése

tén hajtanak végre további feldolgozást egy speciális AdaBins modullal, hogy state of the art eredményeket érjenek el a monokuláris mélységbecslés területén.

2.1.1. A U-net modul

A U-net modul egy olyan autoenkóderre hasonlító struktúra, melyben nem csak az enkóder és a dekóder blokkjain belül jelennek meg közvetlen összeköttetések, hanem az enkóder és a dekóder azonos felbontású rétegei közt is. Ezzel a U-net kimenete nem csak az alacsony felbontású, jól letömörített enkóder kimenet alapján kerül kiszámításra, hanem ebben részt vesznek nagyobb felbontású, a bemenethez közelebbi rétegek is, mely segíthet abban, hogy több részlet jelenjen meg az adott mélységképen. Enkódernek egy előre betanított EfficientNet B5 [16] hálót választottak, míg a dekóder egy általánosnak tekinthető Unet dekóder, amely konkatenálást és bilineáris interpolációval való felskálázást, illetve konvolúciós rétegeket alkalmaz.

2.1.2. Az AdaBins modul

Az AdaBins mélységbecslő újdonságát a Visual Transformer blokkra építő AdaBins modul adja. Ennek a lényege, hogy egyrészt ötvözi a U-net kimenetén található lokális- és a Transformer által meghatározott globális mélységinformációkat. A Visual Transformer [5] működése hasonló a nyelvi feldolgozásban is ismert szekvenciális adatfelolgozásra alkalmazott, figyelmi mechanizmusra épülő Transformer blokkhoz. A lényegi különbséget az adja, hogy képek feldolgozása esetén meg kell oldanunk, hogy a Transformer valamilyen szekvenciát kapjon a bemenetén. Ennek megoldására a képet kisebb, úgynevezett foltokra bontják fel, majd ezeket a foltokat kódolják el úgy, hogy a transformer számára értelmezhető vektor szekvencia alakúak legyenek. Fontos még, hogy az egyes kódok valamilyen formában tartalmazanak sorrendiséget reprezentáló tagot, amelyről szintén gondoskodni kell, mielőtt átadjuk az átalakított képet a transformernek. Értelmezés szempontjából fontos kiemelni a különbséget a nyelvfeldolgozásnál és képfeldolgozásnál használt transformerek között. Míg előbbinél az egyes elemek (pl. fordítási feladat megoldásánál ilyenek lehetnek a szavak egy mondatban) sorrendisége a nyelvtani szerekeztben, illetve az információáramlás szempontjából időben is értelmezhető, addig a képeknél kialakított szekvencia a kép egyes részeit jelenti. Úgy, ahogy a nyelvi esetben a transformer igyekszik globális és lokális összefüggéseket keresni a kódok közt, úgy a képi esetben is hasonlóan képesek lehetünk a U-netnél globálisabb információkat kinyerni a képről.

A transformer kimenetei közül az első az adaptív bin számításra használt MLP-hez csatlakozik, az azt követő C darab kimeneti kód pedig az úgynevezett range attention maps kiszámításához kellenek. A range attention maps a transformer globális és a U-net lokális információinak ötvözéséből adódik. Ezek egyesítéséhez a transfor-

mer C db kimeneti kódját úgy használják, mint 1x1 méretű kernelek, amikkel a U-net kimenetén hajtanak végre konvolúciót. Ez a gyakorlatban megfeleltethető egy pixelenkénti pont szorzásnak. A pontszorzásos konvolúció és a U-net kimenete közt egy 3x3-as kernelt alkalmazó konvolúciós réteg is elhelyezkedik. Ezek fogják meghatározni, hogy az egyes pixelekhez mekkora súllyal társíthatóak az adott binek által reprezentált mélyésgértékek. A range attention maps további konvolúciós és Softmax rétegeken mennek keresztül mielőtt a binek segítségével mélyésgeket határoznánk meg belőlük. A bineknél egy normalizációt, illetve bin szélességből bin közepekre történő transzformációt alkalmazunk a végső felhasználásuk előtt.

Az említett bin normalizáció fő motivációja, hogy a bin szélességek egy 1-re normált, relatív skálájú eloszlást adjanak, amit később felhasználhatunk az adathalmaz esetén ismert mélységtartományra vetítéshez. A normalizáció az alábbi képlettel történik:

$$b_i = \frac{b'_i + \epsilon}{\sum_{j=1}^{N} (b'_j + \epsilon)} \tag{2.1}$$

Ahol ϵ egy kis pozitív szám, amely ahhoz kell, hogy bármilyen kis bin értékeket kapunk is a normalizáció kimenetén, minden bin szélesség pozitív értékű legyen. Számszerűen $\epsilon = 10^{-3}$.

A relatív bin szélességek vetítése az adathalmaz mélységtartományára, és egyben a bin szélességek és bin közepek közti konverzió az alábbi módon történik:

$$c(b_i) = d_{min} + (d_{max} - d_{min})(b_i/2 + \sum_{j=1}^{i-1} b_j) \tag{2.2}$$

ahol $c(b_i)$ az i-edik binhez tartozó bin közép érték, d_{max} és d_{min} az adathalmazra jellemző minimum és maximum mélységértékek. A kiszámított range-attention maps és bin közepek alapján a kimeneten megjelenő mélységértékek az alábbi, súlyozott összegzés alapján kerülnek előállításra, melyet a következő képlet ír le:

$$\tilde{d} = \sum_{k=1}^{N} c(b_k) p_k \tag{2.3}$$

ahol $\tilde{d}$ a becsült mélységérték p_k a range attention maps adott bin középhez becsült valószínűsége. Ezen művelet segít abban, hogy a pusztán oszályozási problémára visszavezetett, és ezáltal kvantált értékkészletű mélységbecslés helyett finomabb, a számábrázolás és a bin eloszlás korlátai mellett lényegében folytonos értékkészlettel rendelkezzen a kimeneti mélységkép.

2.1.3. Tanítási hiperparaméterek és konfiguráció

A tanításhoz az eredeti publikáció egy összetett hibafüggvényt alkalmaz, ami egyrészt a mélységek logaritmikus különbségét veszi figyelembe, másrészt azt figyeli, hogy a binek eloszlása minél jobban közelítse a tanító mélységképek eloszlását. A mélység alapú tag képlete:

$$\mathcal{L}_{pixel} = \alpha \sqrt{\frac{1}{T}\sum_i g_i^2 - \frac{\lambda}{T^2}(\sum_i g_i)^2} \tag{2.4}$$

ahol

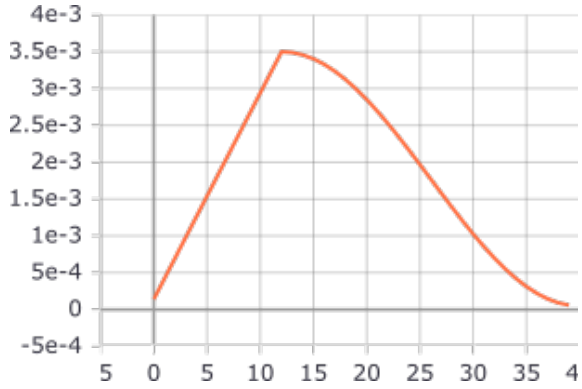
$$g_i = \log \tilde{d}_i - \log d_i \tag{2.5}$$

illetve d_i jelölje a ground turth adatot, $\tilde{d}_i$ a becsült mélységet, T pedig azon pixelek számát, ahol érvényes mélységadatok találhatóak. A publikáció által közölt paraméterek $\lambda = 0.85$, illetve $\alpha = 10$.

A tanításhoz az AdamW optimalizációs eljárást választották, ami az Adam optimalizátor képességein felül egy weight decay-nek nevezett regularizációs tagot is tartalmaz, ami lényegében az L2 regularizációval megfelelő működést képes biztosítani. Ennek lényege, hogy a háló súlyainak négyzetösszegét bűntető tagként jelenteti meg a mélység hibafüggvénye mellett. A súlyok nagyságának korlátozása azért előnyös, mert ezzel megakadályozhatjuk, hogy a rétegekeben bizonyos súlyok túl nagy jelentőséggel szóljanak bele a réteg kimenetébe, míg más súlyokat elnyomnak. Ez nem vezetne optimális működéshez, mivel a túl nagy elnyomó súlyok túltanulást eredményezhetnek. Azonban, ha képesek vagyunk a hálót annak tanítása során olyan irányba terelni, hogy minél több súly részt vegyen az eredmények generálásában, az jobb generalizációhoz vezethet, így némileg megelőzhető a look up table-szerű működés. A weight decay értéke 10^{-2}.

A tanulási ráta segítségével szabályozhatjuk, hogy a háló súlyai tanítás közben milyen erősítéssel legyenek módosítva a hibafüggvény és a gradiensek szerint.

A legegyszerűbb alkalmazása a konstans tanulási ráta használata az egész tanítás alatt, azonban léteznek ennél szofisztikáltabb megoldások is, amelyek jobban tudják segíteni a tanulás hatékonyságát. Az AdaBins esetében a 1-cycle policy-t követve egy felfutó, majd a tanítás végéig lecsengő nagyságú tanulási rátát használnak. A felfutó szakasz az iterációk első 30%-át teszi ki, és lineárisan emelkedik max_lr/25-ről max_lr értékre, ahol max_lr a tanulási ráta maximuma. Számszerűen $max\_lr = 3.5 \times 10^{-4}$. A felutást koszinuszos lecsengés követi max_lr/75 értékre.



2.2. ábra. Learning rate alakulása az idő függvényében

3. fejezet

Az AdaBins háló fejlesztése és tanítása

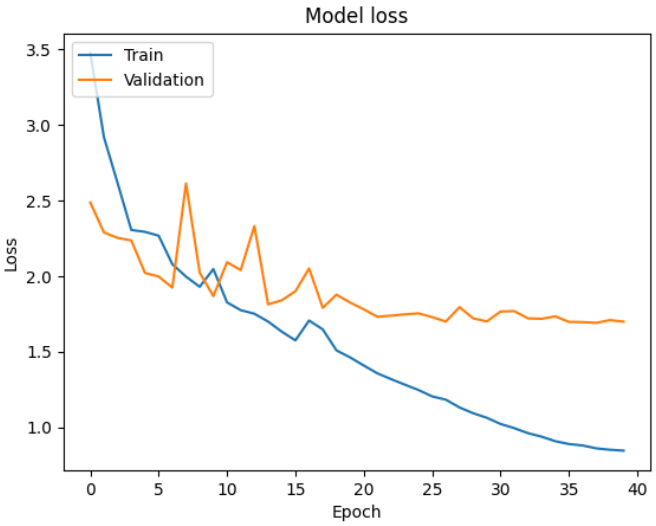
Az AdaBins mélységbecslőnek elérhető volt a publikáció szerző által készített hivatalos implementációja, amely a PyTorch Deep Learning könyvtárat használja. A mélységbecslő általam készített megvalósítása követte ennek felépítését, azonban a felhasznált keretrendszer a TensorFlow és Keras voltak. A saját implementáció készítésének oka többek közt az volt, hogy mélyebben megismerhessem egy napjainkban state of the art-nak tekinthető háló fejlesztési folyamatát és kihívásait. A másik ok ezen keretrendszer mellett az abban szerzett előzetes gyakorlat az egyszerű neurális háló struktúrák definiálásában és tanításában.

3.1. U-net modul

Elsőként a háló architektúrájának megsimerése mellett implementáltam a U-net modult [12], melyen tanításokat is elvégeztem az NYU Depth Dataset V2 [10] mélységképeit használva. Ezen mélységképekről alapvetően elmomndható, hogy sűrűn tartalmaz érvényes értékeket, a kép belső tartományában ritkák az érénytelen területek, inkább a széleken tapasztalhatóak ilyesfajta pixeltartományok. A U-net tanítása során elsődlegesen a keretek által okozott invalid mélységértékek figyelmen kívül hagyásával igyekeztem a háló konvergenciáját és generalizálását javítani. Ezt követően az alapvetően zérusnak tekinthető értékeket felvevő érvénytelen adatok közül nem csak a keret környezetében találhatóakat, hanem a képen belüli régiókat is igyekeztem figyelmen kívül hagyni. Ennek az egyik hasznos következménye, hogy a hálónak ezzel van lehetősége ezeken az érvényes adattal nem rendelkező területeken

egy, a helyeshez jobban közelítő eredményt elérni. A másik fontos pozitívum, hogy a rossz adatok kiszűrésével nem tanítjuk meg a hálónak a hibás eredmények reprodukálását, így a tanult súlyok nem állnak rá arra, hogy a dataset készítéséhez használt RGBD kamera korlátait örökölje.

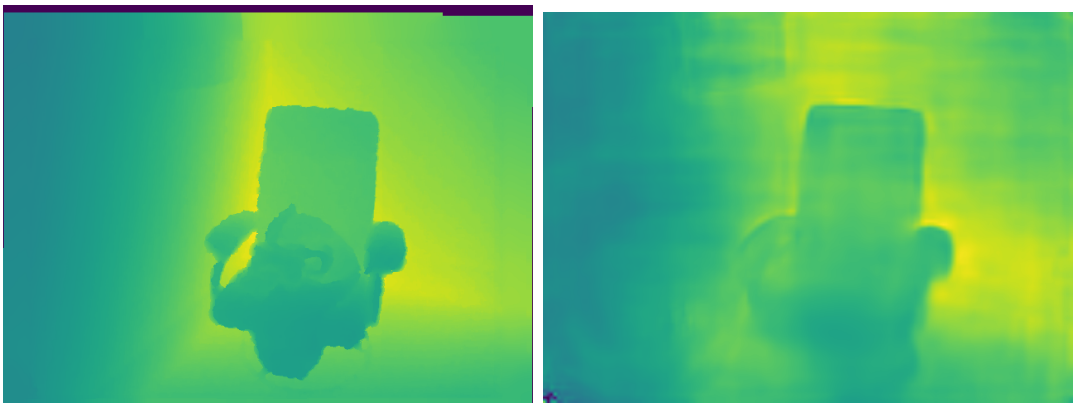
3.1.1. Eredmények értékelése



3.1. ábra. Hibafüggvény alakulása a U-net tanítása során

A U-net tanítása során mentésre került a hibafüggvény alakulása az idő függvényében. Ennek vizsgálatából megállapítható egyrészt, hogy a tanító adatokra a hál képes tanulni, a konvergenciával ilyen tekintetben nincsenek problémák. A validációs halmaz hibájának alakulása alapján túltanulásra következtethetünk, mivel megközelíthetőleg a 15. epochtól kedzve a validációs- és tanítási hiba közti különbség egyre nagyobb az idő előrehaladtával.

Az általánosan elért (tanításnál nem használt bemenet hatására kapott) eredményre példa a 3.2 ábrán látható. Itt megfigyelhető, hogy globális szinten a lassú mélységgradienssel rendelkező területek jól követik az elvárt értéket, azonban nem olyan "simák" az átmenetek, mint a grount truth mélységképen. A másik javítandó részlet a képen, hogy az élszerű, hírtelen mélységváltozásoknál a kimenet nem követi jól, illetve kellő felbontással az elvárt értékeket. Bár mind ebben, mind pedig a teljes AdaBins tanítás esetén elmondható, hogy a háló kimenetén a bemeneti kép $H \times W$ dimenziói helyett a számításigények csökkentése céljából csupán $h \times w$ méretű kimenetet állít elő a háló, ahol $h = \frac{H}{2}$ és $w = \frac{W}{2}$

!ht] 

3.2. ábra. AdaBins U-net modul tanításához használt ground truth mélységkép (bal) és a háló kimenete (jobb)

3.2. Teljes AdaBins háló

3.2.1. Fejlesztés

A későbbiekben [13] során kiegészítettem a meglévő AdaBins U-net modult a teljes (AdaBins modult is tartalmazó) struktúrává, hogy a publikációhoz legjobban hasonlító megoldást próbálhassak meg tanítani, így egy várhatóan közel state of the art mélységbecslőt elérve. Ehhez szempont volt a publikációban [2] közölt AdamW optimalizátor alkalmazásának lehetővé tétele. Ez a feladat annak a korábbi problémának a megoldását is aktualizálta, hogy a tanulási rátát ne csak epochonként, hanem minden iterációban képesek legyünk változtatni, ami segít a tanítás finomabb szabályozásában. Az AdamW alkalmazásának nehézsége abban rejlett, hogy ez az optimalizátor még csak a Tensorflow-addons könyvtárban érhető el, és nem alkalmazható vele minden olyan tanulási ráta ütemezés, mint a core TensorFlow könyvtár optimalizációs algoritmusainál, ahol egy egyszerű callbackes módszerrel tudtuk változtatni a learning ratet az epochok között. Hogy mégis működtetni tudjuk az AdamW-t, egy másik, némileg bonyolultabb scheduling módszert kellett alkalmazni. Itt subclassingot kellett alkalmazni a tf.keras.optimizers.schedules.LearningRateSchedule osztályra, hogy az ütemezőnket át tudjuk adni a tanítást futtató metódusnak paraméterként.

Ezen kívül az AdaBins_TensorFlow úgy került kialakításra, hogy a hiperparaméterek, és egyéb konfigurációs beállítások egy helyről legyenek kezelhetőek. Ehhez a korábban már létrehozott, és alapvetően a háló működtetéséhez és egyszerű használatához készített segédfüggvényeket és osztályokat tartalmazó helpers állományban létrehozásra kerültek a Hyperparameters és a Configuration osztályok. Míg előbbi

a háló és annak tanításához használt hiperparamétereket, az alkalmazott GPU-k számát és ezekhez hasonló információkat tartalmazott, addig a Configuration osztály inkább a háló struktúrájának megváltoztatására, illetve az Adam és AdamW közötti kiválasztásra volt alkalmas. Szintén itt lehet beállítani, hogy a háló mekkora részletet vágjon ki a bemeneti képből, hogy utána ezen a képrészleten végezze el a mélységbecslést. Ezek az osztályok nem tartalmaznak metódusokat, csupán ezeket az értékeket tárolják. Egy fejlesztési lehetőség ezen a területen, hogy a GPU-k száma is a Configuration osztályban kerüljön eltárolásban, ugyanis inkább a rendszer konfigurációját jellemzi, és kevésbé tekinthető hiperparaméternek.

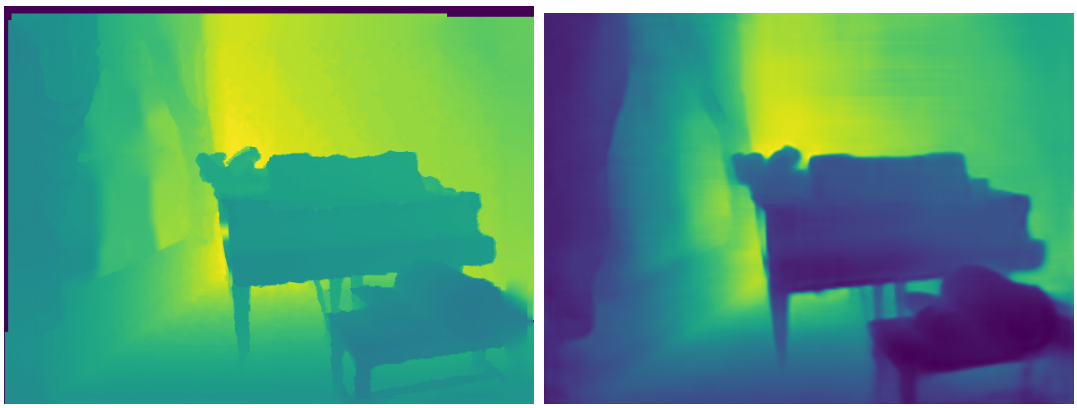
A tanítás szempontjából lényeges lehet, hogy a hálóban több helyen is He normal inicializációt alkalmaztam, mivel ez egy jó kiinduló értéket ad azoknál a súlyoknál, amik relu aktivációt megelőző hálóhoz tartoznak. Ahhoz, hogy a háló jobban a generalizációra tanuljon az egyszerű lookup table-szerű viselkedés helyett, adatkiterjesztő technikákat is alkalmaztam. Az egyik ilyen a háló elején elhelyezett Gauss zaj réteg, amely a bemeneti képeket zajosítja tanítás időben, azonban inference módban kikapcsolásra kerül. A másik a bemeneti képeken alkalmazott véletlenszerű horizontális tükrözés, amely szintén segít, hogy az adatokat változatosabbá tegyük. Ezen a területen egy további lehetőségként a későbbiekben még implementálható egy random crop, amely a kép közepéről véletlenszerű helyről, azonban megadott méretű kivágást tudna elvégezni. Ez egyrészt szintén az adatok változatosságát tudja növelni, másrészt a hálót bizonyos szinten segíti abban, hogy invariánsabb legyen a kamera paraméterei okozta torzításokra, mivel a képek az mindig egy másik képrészletről kerülnek ki, ezáltal némileg másabbak lesznek a torzítások a bemeneteken.

3.2.2. Tanítás NYU Depth Dataset V2 adathalmazon

Az elsőként alkalmazott adathalmaz az NYU Depth Dataset V2 volt. Ennek egyik oka, hogy a publikáció szerzői is írnak ezzel az adathalmazzal végzett tanításaik eredményeiről, a másik pedig, hogy az adathalmaz kezelése könnyű TensorFlow keretrendszerben, mivel a TensorFlow datasets beépített függvényeivel egyszerűen letölthető és beolvasható a háttértárról.

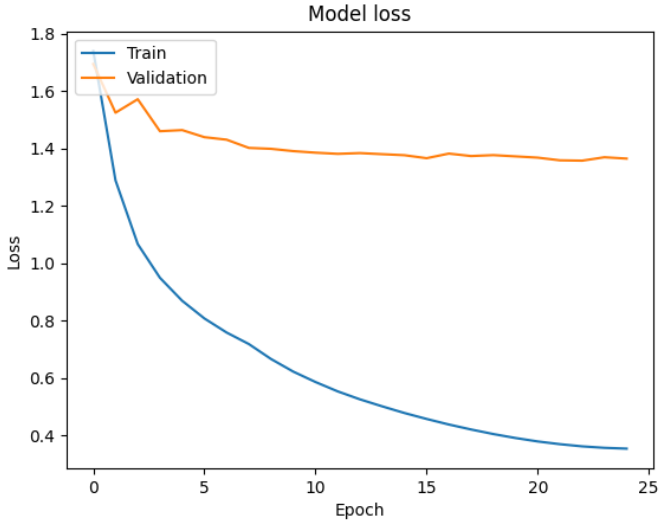
Az adathalmazra történő tanításkor az AdaBins modul kisebb eltérésekkel rendelkezett az eredeti, publikált architektúrához képest, azonban a tanítás eredményei kvalitatív vizsgálat során további használatra alkalmasnak tűntek.

3.2.3. NYU Depth Dataset V2 tanítás eredményeinek értékelése



3.3. ábra. AdaBins tanításához használt ground truth mélységkép (bal) és a háló kimenete (jobb)

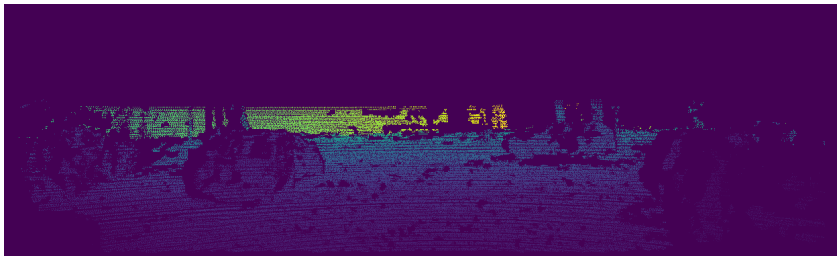
A tanítás során rögzítésre került a hibafüggvény értéke. Ennek vizsgálatával megállapítható, hogy nagyjából az 5. epoch után nem javul a validációs hiba, ami azt jelenti, hogy a maradék időben a háló nem generalizálni tanul meg, inkább csak túltanítás történik a tanító halmazra. Ennek egyik oka lehet, hogy ennél az adathalmaznál még nem került alkalmazásra olyan, a túl nagy súlyokat bűntető regularizációs technika, mint az L2 regularizáció vagy az AdamW optimalizátor. Ezen kívül itt még a Gauss zaj, illetve tükrözéses adat kiterjesztés sem volt használva, amik tovább segíthették volna a generalizációt.



3.4. ábra. AdaBins hibafüggvényének alakulása NYU Depth Dataset V2-vel

3.2.4. Tanítás KITTI depth adathalmazon

A KITTI mélységkép adathalmazára [18] történő tanítás során a háló struktúrája jobban közelítette a publikációban ismertetett architektúrát, csak kisebb eltérések szerepeltek benne. Ennél a tanításnál már alkalmazásra került az AdamW optimalizátor, amihez az újfajta tanulási ráta ütemezőt használtam. A KITTI tanítás egyik nehézsége az volt, hogy az adathalmaz LIDAR szenzorral készült, ami miatt a ground truth mélységképen egy bizonyos magasság fölött nagyon kevés esetben szerepeltek érvényes értékek. Erre példa az alábbi kép:

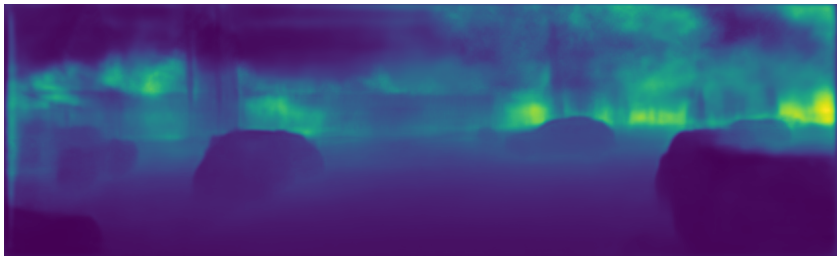


3.5. ábra. KITTI ground truth mélységkép

A KITTI tanítás egy további nehézsége volt még, hogy ezen adathalmazhoz nem található a TensorFlow által szolgáltatott egyszerű API hivás, mely menedzselné az adatok letöltését és szerializációját, tehát ezen funkciókkal még ki kellett egészíteni az AdaBins_TensorFlow projektet

3.2.5. KITTI depth tanítás eredményeinek értékelése

A KITTI tanítás után a háló az alábbi kimenetet produkálta az adathalmazból választott bemenetre. A háló későbbi használatából látható volt, hogy a háló más hasonló bemeneti adatok esetében is képes volt ezt megközelítő minőségű mélységképek előállítására.



3.6. ábra. KITTI becsült mélységkép

A háló tanításának kiértékelése szempontjából itt a hiba alakulása és a kvalitatív ellenőrzés mellett az AdaBins publikációban is publikált kiértékelési metrikákat is implementáltam, hogy a háló performanciájának számszerű értékelését is lehetővé tegyem. A alábbi kiértékelési metrikákat implementáltam és használtam az összehasonlításhoz:

Threshold accuracy (δ_i): Azt fejezi ki, hogy a pixelek hány százaléka esetén kisebb a δ érték, mint a threshold szint. A δ meghatározása az alábbi módon történik: $\delta = max(\frac{y_p}{\hat{y}_p}, \frac{\hat{y}_p}{y_p})$ Average Relative error (REL): $\frac{1}{n}\sum_p^n \frac{|y_p - \hat{y}_p|}{y}$, Squared Relative Difference (Sq Rel): $\frac{1}{n}\sum_p^n \frac{||y_p - \hat{y}_p||^2}{y}$, Root Mean Squared error (RMS): $\sqrt{\frac{1}{n}\sum_p^n (y_p - \hat{y}_p)^2}$, RMS log: $\sqrt{\frac{1}{n}\sum_p^n ||(log(y_p) - log(\hat{y}_p)||^2}$

Metrika	AdaBins_TensorFlow	Eredeti AdaBins	Eigen et al. [6]	Liu et al. [8]
δ_1	0.884	0.964	0.702	0.680
δ_2	0.984	0.995	0.898	0.898
δ_2	0.997	0.999	0.967	0.967
REL	0.09616561	0.058	0.203	0.201
Sq Rel	0.544	0.190	1.548	1.584
RMS	4.921754	2.360	6.307	6.471
RMS log	0.149	0.088	0.282	0.273

3.1. táblázat. AdaBins_TensorFlow KITTI tanítás összhasonlítása az eredeti publikáció eredményeivel kiértékelési metrikák alapján

Ezen kívül még egy, a publikációban leírt metrikát is használtam, ami az average log_10 error volt. Ennek formulája: $\frac{1}{n}\sum_p^n |\log_{10}(y_p) - log_{10}(y_p)|$ Bár a publikáció ide vonatkozó táblázatában nem található ennek megfelelő oszlop a KITTI tanítás esetén, ezáltal nincs összehasonlítási alap, azonban a kiértékeléskor ennek az értékét szintén kiszámítottam. Ez $log_{10} = 0.103$

A kvantitatív kiértékelés relatíve nagy eltéréssel rendelkezik az eredeti megvalósításhoz képest, azonban más, a publikációban szereplő megoldássokkal összevetve elmondható, hogy abszolút eltérésben nem esik távol tőle. Mivel általánosan elmondható, hogy a háló a tanítóadatokra még mindig képes jobb eredményeket elérni, mint a validációs halmazra, ezért az eredmények javulását okozhatná például nagyobb adathalmazra történő tanítás vagy bonyolultabb adatkiterjesztés alkalmazása.

4. fejezet

Szegmentáló és hibrid megoldások az AdaBins architektúrájából kiindulva

4.1. Attention Segmentation háló

A mélységbecslők mellett egy másik hasznos funkció lehet például akár egy SLAM feladat megoldásánál a szemantikus információk kinyerése a bemeneti képből. A szemantikus szegmentáció során a képen igyekszünk minden pixel által reprezentált valós térbeli pontot egy kategóriába, osztályba sorolni. Az ide tartozó irodalomból [17] kiindulva feltételeztem, hogy egy mélységbecslő architektúra (esetlegesen némi modifikációt követően) alkalmas lehet a szemantikus szegmentáció problémájának megoldására is. Erre végeztem kísérletet az AdaBins struktúrával.

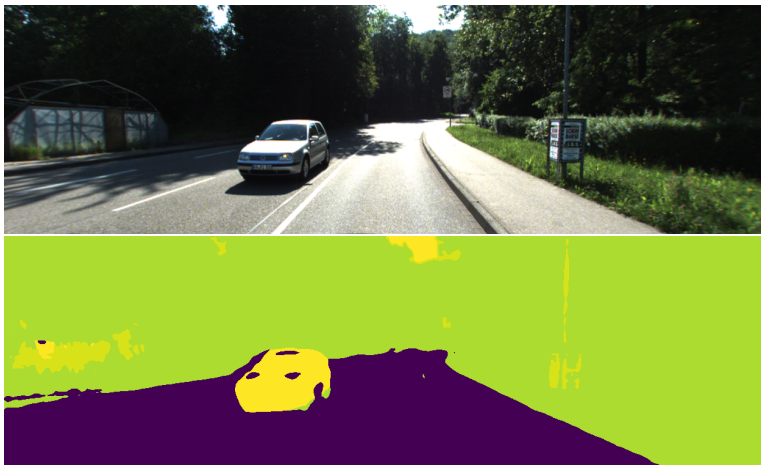
4.1.1. Kialakítás

A háló kiindulási alapja az AdaBins mélységbecslő, melyen belül már található egy olyan mechanizmus, amely osztály jóságokat határoz meg az egyes pixel értékekhez. A szegmentáló háló ezt használja fel kimenetéül, így azok a rétegek, amelyek az AdaBins hálóban ezután találhatóak, nem képezik a szegmentáló részét. Ezen kívül eltérés az is, hogy mivel nincs szükség az osztályozás utáni rétegekre, ezért átalakítható maga az mViT kimenete is. Az AdaBins esetében a transzformer első kimeneti kódját arra használjuk, hogy előállítsuk az adott képre adaptív bin közép eloszlást,

és csak az ezutáni kimeneti kódokból került előállításra a valószínűségek tenzora. Mivel a szegmentáló hálóban nincs szükség binekre, ezért itt ez a rész kihagyható, a valószínűségeket előállíthatjuk az első kódtól kezdődően.

4.1.2. Kezdetleges tanítási eredmények KITTI segmentation adathalmazon

A háló tanítása során a KITTI szegmentációs adathalmaz [1] szolgáltatta a ground truth adatokat, a hibafüggvény pedig a TensorFlowban található SparseCategoricalCrossentropy volt. Fontos megemlíteni, hogy a tanítás a datasetnek csak egy kis részhalmazán történt, erőforrás, illetve időgazdálkodási szempontokból.



4.1. ábra. Az Attention Segmentation háló bemenete (fent) és kimenete (lent) rövid KITTI tanítás után

A rövid tanítás eredményéből is már látható, hogy a háló struktúrája valóban alkalmas lehet a szegmentáció feladatának megoldására, mivel a bemeneti képen különálló területeket képes volt a tanítási körülményekhez képest megfelelően különálló osztályokba kategorizálni.

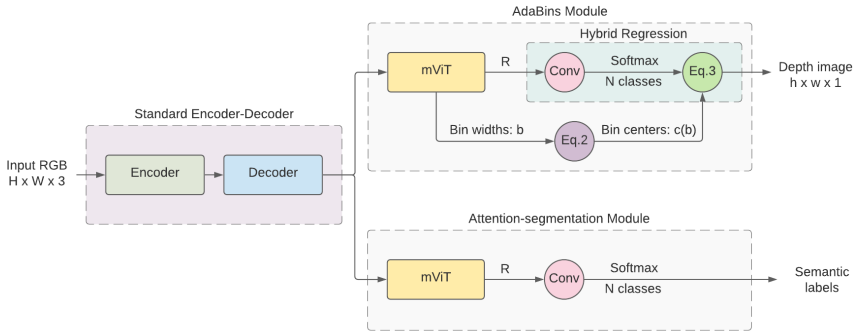
4.2. AdaBins segmentetion hibrid architektúra

A szegmentáló háló tesztelését követően egy következő lépésként kialakítottam egy olyan, az AdaBins és a belőle tervezett szegmentáló hálót egy új architektúrává alakítani, mellyel egyszerre kíséreltem meg megoldani a mélységbecslés és a szemantikus szegmentáció problémáját.

4.2.1. Kialakítás

Annak érdekében, hogy minél jobb generalizációt érjen el a háló, úgy alakítottam ki a struktúrát, hogy legyen az elején egy általános modul a két feladatra közös információk kinyerésére. A háló ezt a közös modult követően két külön ágra elágazva próbálja megoldani a két specifikus részfeladatot.

Az általános, bemenethez közeli közös modul az eredeti AdaBins modularizációt követve a U-net modul lett. Ebből ágazik szét egyrészt az AdaBins modul a mélységbecslés megoldásához, illetve a 4.1.1-ben prezentált figyelmi alapú szegmentáló modul a pixelek szemantikus osztályozásához.



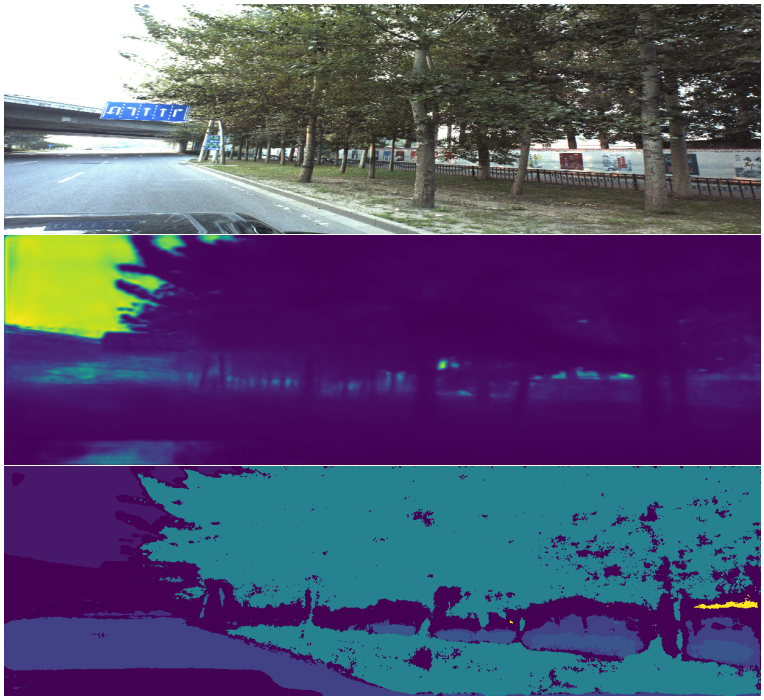
4.2. ábra. Az AdaBins-Segmentation háló

4.2.2. Tanítási eredmények Apolloscape Scene Parsing adathalmazon

Az AdaBins-Segmentation háló tanításához az Apolloscape adathalmaz [19] egy kisebb részét használtam. Az Apolloscape Scene Parsing dataset a szemantikus osztályok mellett tartalmaz mélységképeket is a bemeneti RGB képekhez. Utóbbiakat összehasonlítva a KITTI mélységkép adathalmazával, ennél sokkal sűrűbb mélységképekről beszélhetünk, mint a KITTI depth dataset esetében, ami segíthet a hálónak jobban konvergálni, hiszen több helyen hasonlíthatóak össze a kimenetek az elvárt eredményekkel. Fontos megjegyezni, hogy az Apolloscape ezen mélységképei szűrt képek, mivel a dinamikusnak tekinthető (például járművek, gyalogosok) objektumok nem szerepelnek rajtuk, azokat kiszűrték és csak a statikus környezet mélységét láthatjuk. Azért, hogy a hálónak ne legyen szükséges ezt a dinamikus szűrést is megtanulnia, a szemantikus ground truth adatoknál a dinamikus osztályba tartozó pixelek kimaszkolásra kerültek a mélységképek hibafüggvényének számításához.

A másik fontos különbség az adatok sűrűsége mellett, hogy az Aplloscape mélységképei az ég és nagyon távoli, már nem jól látható területekhez nem érvénytelen adatokat rendel, hanem a legnagyobb felvehető értéket. Ez azért lehet előnyősebb, mert például amikor pontfelhők előállítására akarjuk felhasználni a becsült mélységképeinket, akkor látható, hogy a KITTI típusú tanító adatok azt okozzák, hogy az ég, illetve a képen egyre feljebb található objektumokhoz tartozó mélységek álatlánosan alacsonyabb mélységértékeket fognak felvenni, ezáltal a pontfelhő a teteje felé haladva egy visszanyúló alakot vesz fel. Amennyiben a háló olyan adatokat kap, ahol az éghez inkább az értelmezési tartományának maximumát társítja, ez a probléma esetlegesen elkerülhető.

Ennek a hálónak a tanításánál a szegmentáció hibájának számításához a SparseCategoricalCrossentropy helyett CategoricalCrossentropy-t használtam. Ebben a kialakításban, illetve tanítási környezetben ennek használata egyszerűbb megoldásnak tűnt.



4.3. ábra. Az AdaBins-Segmentation háló eredményei Apolloscape tanítás után. Felül a bemeneti RGB kép, középen a mélységbecslő fej kimenete, alul pedig a szegmentáló fej kimenete

A tanítás eredményeit a 4.3 ábra prezentálja. A mélységbecslő modul valóban képes volt az éghez nagy értékeket rendelni, a bemeneten látható objektumok alakja felismerhető a becsült mélységképen is.

A szegmentáló modul eseén a növényzetet tartalmazó jobb oldali területek osztályozása némileg zajosnak tekinthető, azonban a fák esetén ez betudható akár az átszűrődő háttérnek is. Ezen kívül a fák lombja és az azzal szomszédos szemantikus területek között egyfajta határréteg jelenik meg, amely szintén korrigálandó.

Általánosságban elmondható, hogy az AdaBins-Segmentation hibrid háló architektúra képes volt a mélységbecslés és a szemantikus szegmentáció megtanulására egy kisebb, kb. 3500 adatból álló részhalmazon, amely okot adhat a későbbi, nagyobb tanításokkal való kísérletekre.

5. fejezet

Összefoglalás és konklúzió

5.1. Összefoglalás

Munkám során tanulmányoztam egy napjainkban újnak tekinthető mélységbecslő architektúrát, amelyet iteratív módon implementáltam és tanítottam különböző adathalmazokon. A tanítások eredményeit vizsgálva látható volt, hogy sikerült kialakítani a feladat megoldására alkalmas struktúrát. Egy másik konklúzió a hibafüggvény alakulása alapján, hogy az általános (tanításnál nem használt bemenetekre jellemző) eredmények tovább javíthatóak, hiszen a validációs hiba általában észrevehetően magasabb maradt, mint a tanítási hiba. Ennek lehetséges megoldásai nagyobb adathalmaz alkalmazása, bonyolultabb adatkiterjesztés használata, illetve az egyes regularizációk további hangolása.

Ezt követően a mélységbecslőből kiindulva kialakítottam egy szemantikus szegmentációra alkalmas struktúrát, melyet egy kisebb adathalmazon tanítottam is. Ennek eredményéből megállapítottam, hogy a módosított architektúra el tudja végezni a szegmentáció feladatát, és nagyobb adathalmazon való tanítások esetén az eredmények további javulása is lehetséges lenne.

A mélységbecslő és a szegmentáló hálókból egy új, hibrid architektúrát kialakítva megkíséreltem a mélységbecslés és a szemantikus szegmentáció együttes megoldását. A tervezett háló tanítását követően annak eredményeiből látható, hogy a hibrid megoldás működőképesnek tekinthető, és a korábbiakhoz hasonlóan elmondható, hogy a tanítóadatok számának növelése, illetve a bonyolultabb adatkiterjesztés várhatóan növelné a pontosságot.

5.2. Adathalmazok összehasonlítása tanítás szempontjából

A neurális hálók működésénél nagy jelentőséggel bír, hogy milyen adatokat használunk fel a tanításhoz. A különböző háló megoldásokat több különböző tanító adathalmazon is tanítottam, ezáltal lehetőségem nyílt a különböző adathalmazok hasznosságának és kimenetre gyakorolt hatásának összehasonlítására.

Az NYU Depth Dataset V2 [10] beltéri képeket tartalmazó, érvényes mélységértékeket sűrűn tartalmazó adathalmaz, mely alkalmasnak bizonyult az AdaBins tanítására, az eredményül kapott kimeneti mélységbecslés közelített a ground truth adatokon láthatóhoz. Megjegyzendő, hogy míg a tanítóadatoknál megfigyelhető volt egyfajta bizonytalanság az objektumok széleinél, ezt azonban a háló tanulása után némileg kiszűrte, így a bemeneti RGB kép objektumainak körvonalát jobban képes volt lekövetni. A tanítóadatok ezen bizonytalanságából kifolyólag nem feltétlenül probléma ennél az adathalmaznál egy kisebb értékű hiba a tanítás végén, ugyanis az adódhat ezen szűrt kimenet és a tanítóadatok bizonytalan élei közti különbségből is.

A KITTI depth dataset [18] előnye, hogy azokon a helyeken, ahol rendelkezésre állnak érvényes adatok, ott relative nagy pontosságú kültéri mélységértékek állnak rendelkezésre. Ez segíti a hálót, hogy valós kültéri alkalmazásokban jobban használható legyen. Hátrányos tulajdonság azonban, hogy vizsonylag sok helyen tartalmaznak érvénytelen mélységértékeket a tanítóadatok, aminek következtében a hibafüggvény kevesebb helyen tudja meghatározni, hogy helyes kimenetet szolgáltatott-e a háló. Ez leginkább az ég esetén figyelhető meg, ugyanis a kép ilyen területeihez alacsony mélységet társít a háló, és az is megfigyelhető volt, hogy általánosságban az objektumokhoz tartozó mélység lokálisan is csökken a képen felfelé haladva. Ez a tulajdonság például egy pontfelhő alapú 3D rekonstrukció esetén kevésbé tekinthető előnyösnek. [13]

A KITTI szegmentációs adathalmaz [1] az AdaBins-ből kiindulva tervezett szegmentáló háló architektúránál került felhasználásra, és a tanítás szempontjából elmodható, hogy megfelelő tanítóadatokat tud szolgáltatni a pixelek szemantikus osztályozásához.

Az Apolloscape Scene Parsing [19] adathalmaz egy szegmentációs adathalmaz, amely mélységértékeket is tartalmaz. Előnyös tulajdonsága, hogy sűrű ground truth mélységképeket szolgáltat a KITTI mélységekkel ellentétben. Nehézséget okoz hasz-

nálatában, hogy nem egyszerű mélységadatokat tartalmaz, hanem annak egy olyan módosítását, amely a statikus objektumokat kiszűri a látott környezetből. Egy másik megfigyelhető eredménybeli eltérés a KITTI tanítás eredményeihez képest, hogy itt a mélység értéke az úton kevésbé tűnük átmenetesnek, mint a másik esetben. Ez okozhat esetleges problémát a 3D rekonstrukció során, amennyiben a mélységek alapján nem megfelelően sikerül az út pontjainak elhelezése a pontfelhőben.

Irodalomjegyzék

[1] Hassan Alhaija – Siva Mustikovela – Lars Mescheder – Andreas Geiger – Carsten Rother: Augmented reality meets computer vision: Efficient data generation for urban driving scenes. *International Journal of Computer Vision (IJCV)*, 2018.

[2] Shariq Farooq Bhat – Ibraheem Alhashim – Peter Wonka: Adabins: Depth estimation using adaptive bins. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (konferenciaanyag). 2021, 4009–4018. p.

[3] Michael Bloesch – Jan Czarnowski – Ronald Clark – Stefan Leutenegger – Andrew J Davison: Codeslam—learning a compact, optimisable representation for dense visual slam. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (konferenciaanyag). 2018, 2560–2568. p.

[4] Li Deng: The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29. évf. (2012) 6. sz., 141–142. p.

[5] Alexey Dosovitskiy – Lucas Beyer – Alexander Kolesnikov – Dirk Weissenborn – Xiaohua Zhai – Thomas Unterthiner – Mostafa Dehghani – Matthias Minderer – Georg Heigold – Sylvain Gelly és mások: An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[6] David Eigen – Christian Puhrsch – Rob Fergus: Depth map prediction from a single image using a multi-scale deep network. *arXiv preprint arXiv:1406.2283*, 2014.

[7] Sowmya Kamath S és mások: A bag of visual words model for medical image retrieval. *arXiv e-prints*, 2020., arXiv–2007. p.

[8] Fayao Liu – Chunhua Shen – Guosheng Lin – Ian Reid: Learning depth from single monocular images using deep convolutional neural fields. *IEEE transactions on pattern analysis and machine intelligence*, 38. évf. (2015) 10. sz., 2024–2039. p.

[9] Raul Mur-Artal – Jose Maria Martinez Montiel – Juan D Tardos: Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31. évf. (2015) 5. sz., 1147–1163. p.

[10] Pushmeet Kohli Nathan Silberman, Derek Hoiem – Rob Fergus: Indoor segmentation and support inference from rgbd images. In *ECCV* (konferenciaanyag). 2012.

[11] Bogár György Richárd: Önálló laboratórium 1 - deep learning integrációs lehetőségeinek vizsgálata slam eljárásokban, 2020.

[12] Bogár György Richárd: Deep learning integrációs lehetőségeinek vizsgálata slam eljárásokban, msc Önálló laboratórium 2, 2021.

[13] Bogár György Richárd: Szakmai gyakorlat beszámoló, adabins mélységbecslő fejlesztése, tanítása és használata felhő alapú térképi adatbázis-építésnél, 2021.

[14] Ethan Rublee – Vincent Rabaud – Kurt Konolige – Gary Bradski: Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision* (konferenciaanyag). 2011, 2564–2571. p.

[15] Muhamad Risqi U Saputra – Pedro PB de Gusmao – Sen Wang – Andrew Markham – Niki Trigoni: Learning monocular visual odometry through geometry-aware curriculum learning. In *2019 International Conference on Robotics and Automation (ICRA)* (konferenciaanyag). 2019, IEEE, 3549–3555. p.

[16] Mingxing Tan – Quoc V. Le: Efficientnet: Rethinking model scaling for convolutional neural networks. *ArXiv*, abs/1905.11946. évf. (2019).

[17] Keisuke Tateno – Federico Tombari – Iro Laina – Nassir Navab: Cnn-slam: Real-time dense monocular slam with learned depth prediction. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (konferenciaanyag). 2017, 6243–6252. p.

[18] Jonas Uhrig – Nick Schneider – Lukas Schneider – Uwe Franke – Thomas Brox – Andreas Geiger: Sparsity invariant cnns. In *International Conference on 3D Vision (3DV)* (konferenciaanyag). 2017.

[19] Peng Wang – Xinyu Huang – Xinjing Cheng – Dingfu Zhou – Qichuan Geng – Ruigang Yang: The apolloscape open dataset for autonomous driving and its application. *IEEE transactions on pattern analysis and machine intelligence*, 2019.