BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
IRÁNYÍTÁSTECHNIKA ÉS INFORMATIKA TANSZÉK

SEBESSÉGMÉRŐ RENDSZER FEJLESZTÉSE TÉRBELI TRANSZFORMÁCIÓS HÁLÓ ÉS MONOKULÁRIS MÉLYSÉGBECSLŐ ALGORITMUS FELHASZNÁLÁSÁVAL

Tanulmányi Diákköri Konferencia

Készítette:

Kürti Ádám

Konzulens:

Dr. Szemenyei Márton

2023

Tartalomjegyzék

Kivonat

A forgalom figyelése és irányítása a modern várostervezés és a rendvédelem kritikus eleme, a sebességmérő kamerák pedig kulcsfontosságú szerepet játszanak a közúti biztonság fenntartásában. A terület jelenlegi állapotát tekintve azonban számos kihívással néz szembe, beleértve a rendszámok nagy pontossággal való leolvasását és a jármű sebességének precíz mérését.

A rendszám felismerő rendszerek tipikusan konvolúciós neurális hálókat (CNN) alkalmaznak a képjellemzők kinyerésére, karakterek szegmentálására és azok leolvasására. A rendszámok pontos felismerését azonban megnehezíti azok számos változata, melyek különböző formákkal és kialakítással bírnak. További jelentős hátráltató tényező a változó szemszögből való észlelésük, ami jelentősen torzíthatja megjelenésüket és így a tartalmuk felismerését nehezíti, hiszen a konvolúciós hálók nem invariánsok térbeli változásokra. A szakirodalomban továbbá hosszú-rövidtávú memória (LSTM) háló alapú megoldásokkal is találkozhatunk, melyekkel egy karakter szegmentáció nélküli leolvasás valósítható meg, ugyanakkor ezek komplexebb felépítéssel és taníthatósággal bírnak.

Sebességmérést klasszikusan dedikált hardver szenzorokkal végzik. Ilyen szenzorok például az útba épített induktív hurokérzékelők, különböző lézerek és Doppler radarok, melyekkel nagy pontosságú eredmények érhetőek el, viszont beszerzésük és karbantartásuk jelentős költséggel jár. A járművek sebességének egyetlen kamerával történő mérését a szakirodalomban eddig többféle módszerrel is megközelítették, ilyen például az út geometriai információit, a mozgó objektumok képpontjai számának különbségét és a mozgásvektor homográfiai leképezését a képsíkról a globális koordinátasíkra felhasználó megoldások. Ezek a módszerek azonban körülményesek és nem mindig adnak pontos eredményeket.

Munkámban egyrészről egy olyan, teljesen konvolúciós mély neurális háló alapú megoldást fejlesztek ki, amely szegmentáció mentesen teszi lehetővé a rendszámok felismerését. Ezt a hálót egy úgynevezett térbeli transzformátor hálóval kombinálom, mellyel a konvolúciós hálóknak a térbeli gyengeségeit célzom kiküszöbölni. A hálót először mesterségesen generált rendszámok segítségével tanítom be, majd valós képeken finomhangolom és tesztelem.

Másrészről a sebességmérő kamera egységet monokuláris, azaz egykamerás, abszolút mélység becslő algoritmusokon alapuló sebességmérő rendszerrel próbálom megvalósítani, amikhez state-of-the-art mélységbecslő módszereket használok fel. A kapott mélység értékeket 3D rekonstrukció segítségével a járművek sebességének kiszámítására használom fel. A fejlesztett rendszert és a használt mélységbecslő algoritmusokat hardveres sebességméréssel validált adathalmazon értékelem ki.

A fejlesztett megoldásokat egy működő sebességfigyelő kamera rendszerként egyesítem, amit az olvasóközönség számára elérhetővé teszek.

Abstract

Traffic monitoring and control is a critical element of modern urban planning and law enforcement, and speed cameras play a key role in maintaining road safety. However, given the current state of the field, it faces several challenges, including reading license plates with high accuracy and measuring vehicle speed precisely.

License plate recognition systems typically use convolutional neural networks (CNN) to extract image features, segment characters, and read them. However, the exact recognition of license plates is made difficult by their numerous versions, which have different shapes and designs. Another significant hindering factor is their perception from a changing perspective, which can significantly distort their appearance and thus make it difficult to recognize their content, since convolutional nets are not invariant to spatial changes. In the literature, we can also find long-short-term memory (LSTM) network-based solutions, which can be used to read a character segmentation-free, but at the same time, they have a more complex structure and are harder to train.

Speed measurement is classically performed with dedicated hardware sensors. Such sensors are, for example, inductive loop sensors built into the road, various lasers and Doppler radars, which can be used to achieve high-precision results, but their acquisition and maintenance involve significant costs. The measurement of the speed of vehicles with a single camera has been approached in the literature using several methods, such as solutions that use the geometric information of the road, the difference in the number of pixels of moving objects, or the homographic mapping of the motion vector from the image plane to the global coordinate plane. However, these methods are cumbersome and do not always give accurate results.

In my work, on the one hand, I develop a solution based on a fully convolutional deep neural network, which enables the recognition of license plates without segmentation. I combine this with a so-called spatial transformer network, with which I aim to eliminate the spatial weaknesses of convolutional networks. I first train the network using artificially generated license plates, then fine-tune and test it on real images.

On the other hand, I am trying to implement the speed measurement camera unit with a monocular, i.e., single-camera, speed measurement system based on absolute depth estimation algorithms, for which I use state-of-the-art depth estimation methods. I use the obtained depth values to calculate the speed of the vehicles with the help of 3D reconstruction. I evaluate the developed system and the used depth estimation algorithms on a data set validated by hardware speed measurement.

I combine the developed solutions as a working speed monitoring camera system, which I make available to the reading public.

1 Bevezetés

A sebességmérő kamerák fontos szerepet játszanak a közúti biztonság és a közlekedés hatékonyságának javításában. Ezek a kamerák a forgalmi sebességkorlátozások betartatására és a gyorshajtás megakadályozására szolgálnak. Ez csökkenti a balesetek valószínűségét a veszélyes területeken, például iskolák vagy építkezések környékén. A gyorshajtás világszerte a közúti balesetek egyik kiemelkedő oka: a jármű sebességének növekedése közvetlen összefüggésben van mind a baleset bekövetkezésének valószínűségével, mind a baleset következményeinek súlyosságával, ezért a sebességmérő kamerák jelentős szerepet játszanak a közlekedési balesetek és halálesetek számának csökkentésében.

Ezen kamerák kihelyezése és karbantartása azonban költséges. A vételáruk jelentős részét a bennük található mérőhardver teszi ki. Ez Magyarországon fele-fele arányban lézeres, illetve radar alapú szenzorokat jelent. Elterjedésük azonban nem véletlen: ezen megoldások precíz sebességadatokkal tudnak szolgálni, melyeket más szenzorokkal eddig nem sikerült kiváltani. A szakirodalomban különböző megoldások találhatóak, amelyek csak egy RGB kamera segítségével próbálnak sebességet mérni. Ezek jelentősen költséghatékonyabbak, viszont nehézkes kalibrációs folyamatokat igényelnek.

A mért jármű sebességét megmérve, azt a járműhöz, pontosabban a rendszámtáblájához kell társítani. Ehhez az azonosításhoz pontos rendszámtábla felismerési eljárásra van szükség. Ez célszerűen szintén RGB kamerákkal végzendő és számos jelenlegi megoldás található rá. Legtöbbjük mély neurális hálókat használ erre a célra, ezen belül a különböző konvolúciós hálók, mint például a YOLO a rendszámtábla detektálásra és az azon lévő karakterek szegmentálására, más architektúrák, mint például a rekurrens neurális hálók és a hosszú-rövidtávú memóriája hálók a karakterek felismerésére kerülnek széleskörben felhasználásra. A szegmentációs technikáknál azonban a korai képjellemzők kinyerésénél (például karakterek szegmentálásnál) fellépő hibák átterjedhetnek a későbbi szakaszokra, amely hibás eredményhez vezethet. Ezen kívül ezek a módszerek összetettebbek lehetnek, mivel mindegy szakaszhoz külön modelleket és algoritmusokat igényelhetnek, melyeknek futattása, illetve tanítása is idő és számítási kapacitás igényesebb.

A fent említett problémákra keresek megoldást a dolgozatomban. A sebességmérést egy egykamerás megoldással közelítem meg, amihez mélységbecslő algoritmusokat használok. Ezek a megoldások manapság egy szélesebb körben terjednek el: mindenhol, ahol 2D képekből 3D tér rekonstruálására van szükség elterjedtek, legyenek azok kiterjesztett valóság vagy robot navigációs applikációk. Sebességmérésnél is ez az alapvető feladat: 3D-s adatok kinyerése 2D-s képekből, majd ezen információk alapján távolság különbségeket, az idő függvényében pedig sebességet meghatározni. Munkámban két modern monokuláris mélységbecslő algoritmust használok fel a 3D terek rekonstrukciójához: az egyik a széleskörben használt és bevált Adabins algoritmus, mely az adaptív tárolóival, melyekről a nevét kapta reformálta a területet. A másik megoldás, amit felhasználok az a ZoeDepth, amelynek a ZoeD-M12-N nevű variánsa a dolgozat írásakor state-of-the-art mélységbecslő algoritmus a NYU-Depth V2 adathalmaz benchmarkon. Az ezen algoritmusok kimenetéből kapott abszolút mélységét az egyes rendszámtábláknak a használt kamera belső paramétereinek segítségével a 3D térbe transzformálom, amelyből képkockáról képkockára távolságokat számolok, ebből pedig sebességet határozok meg. A kapott sebességeket egy induktív hurokérzékelővel validált adathalmazon értékelem ki.

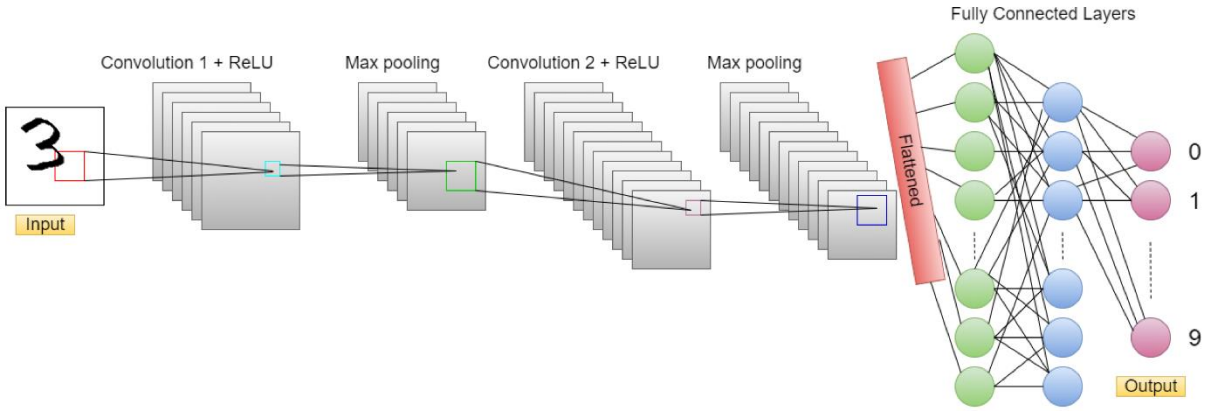
A rendszámtáblák detektálására és leolvasására pedig egy szegmentáció mentes megoldást fejlesztek ki, amely kevés paraméterrel, tehát gyors taníthatósággal és futási idővel is pontos megoldást ad. Ehhez egy teljesen konvolúciós hálót használok fel, amelynek nincsenek teljesen kapcsolt rétegei. Ezt a hálót egy térbeli transzformációs hálóval kombinálom, amellyel az eredeti háló és összességében a konvolúciós hálók térbeli gyengeségeit célzom kijavítani. A tanításhoz és értékeléshez egy 100.000-100.000 valós és generált rendszámtábla adatbázist használok fel.

A fent említett megoldásaimmal a rendszámtábla felismerés terén széleskörűen elérhető hardveren tanítható és futtatható, a sebességmérés témájában pedig a lehető legkevesebb előzetes kalibrációt igénylő megoldást célzok kifejleszteni. A dolgozatomban először a megoldások elméleti hátterét mutatom be, majd a felhasznált irodalomon keresztül a hasonló modern megoldásokat ismertetem. Ezután a javasolt módszereim implementálását végül az azzal elért eredményeket mutatom be.

2 Elméleti háttér

2.1 Konvolúciós hálók

A mélytanulási algoritmusok, amelyeket az emberi agy szerkezete ihletett, mesterséges neuronok összekapcsolt rétegeiből állnak. Ezek a hálózatok képesek automatikusan megtanulni, hogy hatalmas mennyiségű adatból felismerjenek mintákat és jellemzőket, ami különösen hatékonnyá teszi őket olyan feladatokban, mint a kép- és beszédfelismerés. A konvolúciós neurális hálók (továbbiakban CNN) a mély tanulási architektúrák egy alcsoportja, kifejezetten a rácsszerű adatok, például képek feldolgozására készültek.



1. ábra: A konvolúciós hálók általános felépítése (forrás: [1])

A konvolúciós neurális hálózatok jellemzően konvolúciós rétegekből állnak a képjellemzők kinyeréséhez, pooling rétegekből a mintavételezéshez, teljesen összekapcsolt rétegekből az osztályozáshoz, aktivációs függvényekből (például ReLU) a nemlinearitáshoz, ami komplex mintázat megtanulásának elérését teszi lehetővé és normalizáló rétegekből a képzés stabilitásának és pontosságának növelése érdekében. A teljesen összekapcsolt neurális hálózatokkal ellentétben, amelyek minden egyes bemeneti jellemzőt függetlenül kezelnek, a CNN-ek konvolúciós rétegeket használnak a helyi minták felismerésére, ami lehetővé teszi számukra a képek bonyolult részleteinek megragadását.

A mély tanulási modellek, köztük a CNN-ek tanítása a backpropagation nevű folyamatot foglalja magában, amelynek során a hálózat címkézett adatokból tanul. Több iteráción keresztül a modell úgy állítja be belső paramétereit, hogy minimalizálja a jóslatai és a tényleges címkék közötti különbséget. Ez az iteratív tanulási folyamat, amelyet gyakran olyan algoritmusok hajtanak végre, mint a sztochasztikus gradiens ereszkedés (SGD), fejleszti a hálózat azon képességét, hogy pontos előrejelzéseket tudjon készíteni.

A CNN-eknek a minták felismerésében való kiválóságuk ellenére teljesítményük változik a kép méretezésének és forgatásának függvényében. Ha a CNN-eket meghatározott méretarányokra vagy orientációkra tanították meg, akkor nehézségekbe ütközhetnek az általánosítással, amikor e szempontok változatosságaival találkoznak. Például, ha egy CNN-t elsősorban macskaképekre képeztek ki egy adott orientációban, akkor elbizonytalanodhat, amikor megpróbál egy más pózban vagy méretben lévő macskát azonosítani. Ez az invariancia hiánya azért merül fel, mert a hálózatban megtanult jellemzők nem igazodnak megfelelően az átalakított bemenethez, ami téves osztályozáshoz vezet.

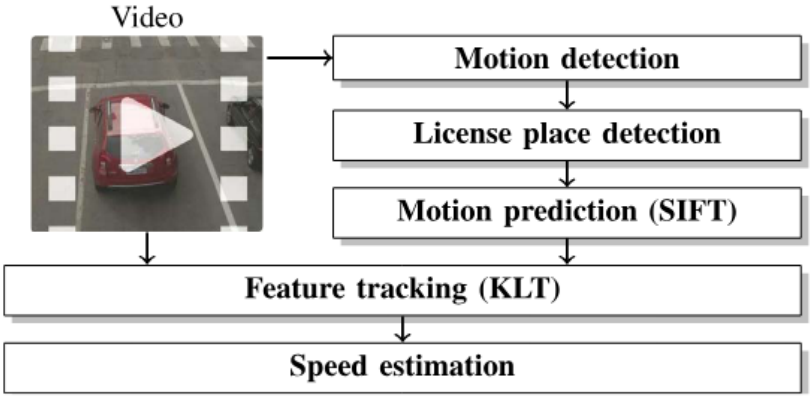
3 Feldolgozott irodalom

3.1 Videó alapú sebességmérés

A fejlesztett sebességmérő kamerán videó alapú sebességmérést valósítok meg. Ehhez a szakirodalomban már meglévő megoldásokat tanulmányoztam, amiket a következőkben mutatok be.

A videóalapú sebességmérő megoldások egyik és legnépszerűbb megoldása a homográfia kiszámításán alapuló megközelítések, amelyek egy sík (az út) lineáris transzformációjaként a 3D-s vetületi térből egy 2D-s vetületi térbe (a kamera képsíkjába) való transzformáción alapulnak. Ily módon a képek madártávlatba transzformálhatók, amelyben a pixeleltolódások közvetlenül valós távolságokká alakíthatók át. Ez a megközelítés a leggyakrabban alkalmazott a rögzített rendszerek esetében [2].

A sebességmérő rendszereket jellemzően intruzív és nem intruzív kategóriákra osztják, ahol az intruzív érzékelők fizikai telepítést igényelnek az úttestre, a nem intruzív érzékelők pedig alternatív módszereket használnak, például lézeres mérőket és Doppler-radarokat. A videó alapú sebességmérés ötlete nem újkeletű [3]. A szerzők a járművek sebességének mérésére egy nem intrúzív, videóalapú rendszert javasolnak, amely digitális kamerákat használ a járművek képének rögzítésére és mozgásának elemzésére. Megjegyzik, hogy a meglévő videóalapú rendszerek drágák lehetnek és gyakori karbantartást igényelnek, de azt javasolják, hogy a digitális kameratechnológia fejlődése a videóalapú rendszereket más, nem intruzív érzékelőkkel szemben olcsóbb alternatívává teheti.

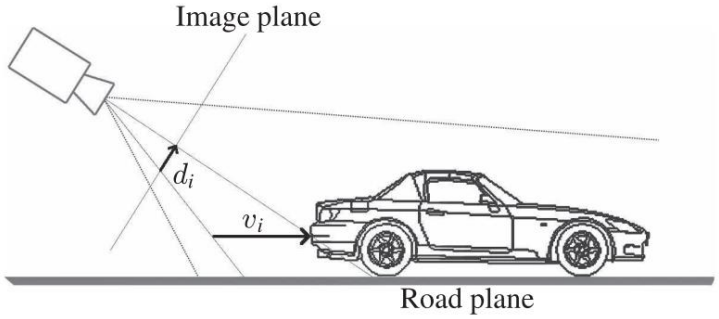


2. ábra: A szerzők által javasolt megoldás blokkdiagramja (forrás: [3])

A javasolt rendszerben alkalmazott mozgásérzékelés a Motion History Image (MHI) koncepción alapul. Az MHI egy olyan bináris kép, amely a videósorozat mozgásának előzményeit mutatja be, a világosabb pixelek a közelmúltbeli mozgást jelzik. Az MHI kiszámítása a képkocka-különbségek küszöbértékének meghatározásával, majd az MHI időbeli frissítésével történik egy decay függvény segítségével. Az így kapott bináris szegmentációs maszkot a mozgó járműveket tartalmazó releváns régiók azonosítására használják.

A rendszámtábla régiójának felismerése után a rendszer kiválasztja a megkülönböztető jellemzők egy csoportját, és több képkockán keresztül követi azt. A cél egy olyan lista előállítása, amely a követett jellemzők pályáját tartalmazza. A jellemzők kiválasztása minden jármű esetében csak egyszer történik, közvetlenül a rendszámtábla észlelése után. Shi és Tomasi

megközelítését követve, a "jó jellemző" olyan régió, amely egynél több irányban nagy intenzitásváltozást mutat, például texturált régiók vagy sarkok. A Kanade-Lucas-Tomasi (KLT) algoritmust használják a jellemzők képkockákon keresztüli követésére. A nagy sebességgel mozgó járművek nagy elmozdulásaival való megbirkózás érdekében egy kezdeti mozgásbecslést végeznek a Scale-Invariant Feature Transform által kivont jellemzők összevetésével.



3. ábra: A kép és az út síkjának illusztrációja (forrás: [3])

A rendszer a jármű sebességeket az elmozdulás vektorokból számolja ki. Minden elmozdulás vektor a jármű pillanatnyi sebességével asszociálható, melynek mértékegysége pixel/képkocka a kép síkjában. Ezt a mennyiséget km/h-ba átváltva valós sebességet kapunk. Ehhez egy fontos feltételezést használnak, mely szerint mindegyik sáv az úton egy-egy képsíkon fekszik. Ez a feltételezés teszi lehetővé a pixelben lévő d_i mozgásvektorok leképezését a kép síkjáról a talaj síkjára méterben lévő v_i elmozdulásokká. A pinhole kamera modellt feltételezve ez a leképezés a jelenet egyetlen nézete alapján, homográfiával - síkról síkba vetített projektív transzformációval - végezhető el, ezt a folyamatot inverz perspektívának is nevezik. Ehhez szükség van egy 3x3 méretű H homográfiai mátrixra, amellyel a $p_i = (x_i, y_i)$ képpontot a $p_w = (x_w, y_w)$ világ síkjába transzformálhatjuk a következőképpen:

$$\begin{bmatrix} x_w \\ y_w \\ 1 \end{bmatrix} = \begin{bmatrix} zx_w \\ zy_w \\ z \end{bmatrix} = H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \quad (1)$$

A homográfia mátrixot négy képbéli pixelpontnak négy, a valóságban ismert ponthoz asszociálásával kapjuk meg. Ehhez a szerzők az induktív hurokérzékelők jelöléseit használták az aszfalton, melyek 4.8 méter x 2.0 méteres téglalapokat formálnak. A szerzők továbbá feltételezték, hogy az út sávjai különböző síkokon fekszenek, ezért sávonként egy-egy homográfia mátrixot mértek ki.

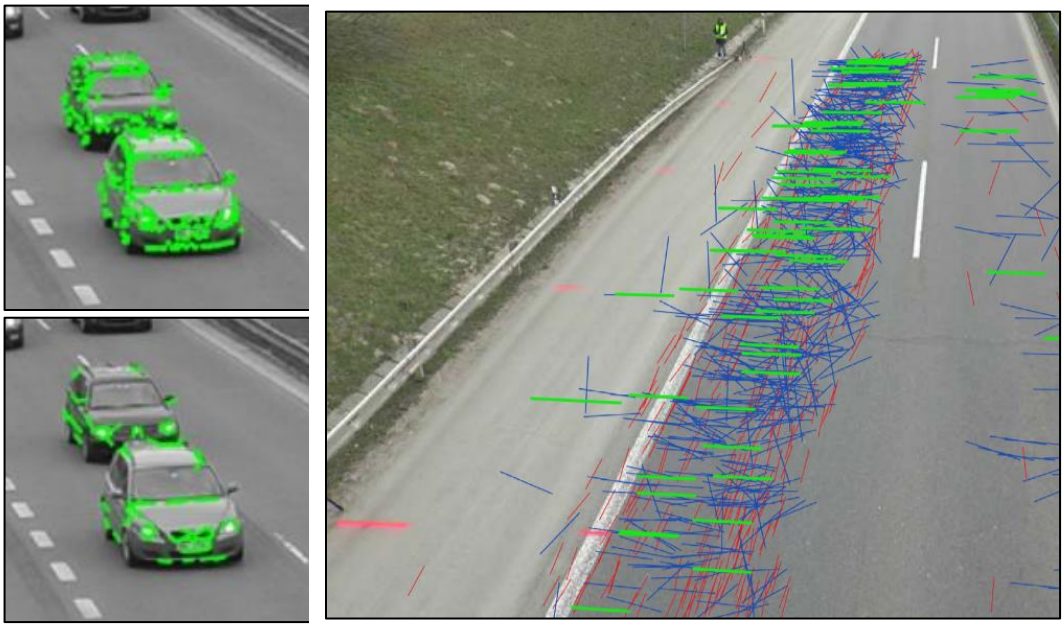
A javasolt sebességbecslési módszerben az egyes járművek mért sebességét megszorozzák egy S konstans tényezővel, amely a kísérletekben 0.9-re van beállítva. Ez a hibás mérések hatásának mérséklésére szolgál, amelyet az a tény okoz, hogy a tényleges rendszámtáblák mindig az út szintje felett vannak, ami a tényleges sebességnél nagyobb számított sebességet eredményez. A jármű végleges sebességét a pillanatnyi sebesség több képkockán keresztüli átlagolása eredményezi, miközben a jármű egy adott képterületnél tartózkodik. A kísérletek során a sebességmérési régiót a ground truth értékeket szolgáltató hurokdetektorokhoz közel használták, hogy lehetővé tegyék a mért sebességek és a ground truth sebességek közvetlen összehasonlítását. A szerzők megjegyzik, hogy az S-tényező használata egyszerű, de hatékony,

amennyiben a követett jellemzők megközelítőleg azonos távolságra vannak a talajtól. Ettől azonban nagyobb járművek könnyedén eltérhetnek.

A javasolt rendszer jól teljesít városi környezetben: a több mint 8000 járművel készült, közel öt órányi videón tesztelve az átlagos hibája -0.5 km/h volt, és az esetek több mint 96%-ában sikerült a szabályozási határértékeken (+3/-2 km/h) belül maradnia. A módszer azonban a homográfiai mátrix kimérését és a rendszer nagyon precíz kalibrációját igényli, ami időigényes és nehézkes feladat.

Videóalapú sebességméréshez a videókon található objektumok valós mérete is felhasználható, például a rendszámtábláé [4] vagy járműé [5]. A J. Sochor és társai által javasolt megoldás [5] automatikus módszert kínál a térfigyelő sebességmérő kamerák kalibrálására, beleértve a jelenet méretskáláját is, melynek meglétével a sebesség könnyedén kiszámolható.

Az automatikus kalibrálás a következőképpen működik: Az első két eltűnési pont (vanishing point) és a principális pont felhasználásával kiszámítható a fókusztávolság, a harmadik eltűnési pont, az útsík normálvektora és az útvonal síkja. Az útsíkot azonban csak a méretarányig lehet kiszámítani (mivel az útsík távolságát nem lehet csak az eltűnési pontokból meghatározni). A kapott útsíkból kiszámolható 3D koordináta a képsík egy tetszőleges pontjából. Az útsíkon a távolságokat közvetlenül 3D koordinátákkal lehetséges mérni; mivel azonban az út síkja egy konstans értékkel előre meghatározott távolságra el van tolva, a P1 és P2 pontok közötti távolságot nem lehet közvetlenül méterben (vagy más valós távolságegységben) kifejezni. Ezért egy másik kalibrációs paramétert kell bevezetni, amelyet skálázó faktornak nevezünk, és amely az út síkjának pszeudo-egységeiből méterekre alakítja át a távolság skálázásával.

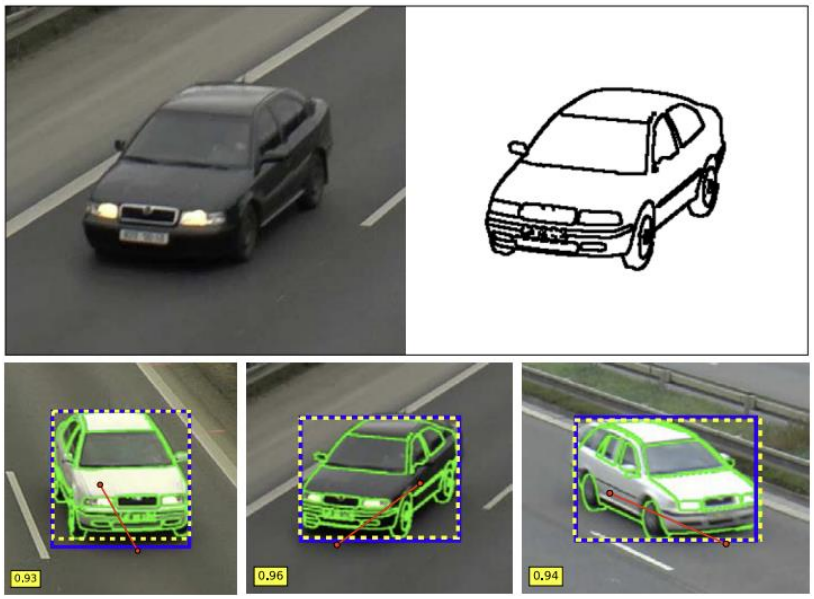


4. ábra: A járműveken edgelet-ek detektálása (forrás: [5])

A megoldás a két eltűnési pontot automatikusan határozza meg. Az első eltűnési pontot a képen lévő sávok metszéspontjából becsülik meg. A tanulmány a mozgó járműveken lévő releváns pontok követésével kialakított vonalak kaszkádos Hough-transzformációját használja az első eltűnési pont becsléséhez. A második eltűnési pont észleléséhez az elhaladó járművek éleit, úgynevezett edgelet-eket használják, mivel az élek által alkotott vonalak egybeesnek az első eltűnési ponttal. Az élek detektálása úgy történik, hogy a kép gradiens nagyságának lokális

maximumaiból kiindulási pontokat keresnek, és minden ilyen pont körül egy mátrixot alkotnak. Az élek orientációját ezután ezek a pontok körül összegyűjtött foltokból számítják ki. Ennek a megközelítésnek az eredményét ábrázolja a 4. ábra, ahol bal oldalt felül a kiindulási pontok, alul a kiszámolt edgelet-ek, jobb oldalt pedig a piros vonalak jelzik az első eltűnési pontot közelítő egyeneseket, a kék, illetve zöld vonalak pedig második eltűnési pontét.

A méretarány meghatározásához a járművek renderelt 3D modelljeinek és a képen észlelt befoglaló dobozoknak (bounding boxoknak) való összehasonlítását használja a jelenet méretarányának pontos megállapításához. Ehhez a járművek pontos osztályozását kísérlik meg, miszerint a gyártót, a modellt, a variánst és az évjáratot is meghatározzák a detektálás során. A detektált 3D befoglaló dobozokat ezután a megfelelő 3D-s CAD modellhez illesztik ahogy azt a 5. ábra illusztrálja, ahol felül a detektált jármű és hozzá tartozó 3D modell, alul pedig járműre illesztett befoglaló doboz és a modell váza látható. A befoglaló doboz pixelben mért méretét a valós 3D modell méterben adott nagyságával összevetve meghatározható a méret skálázási arány.



5. ábra: A méret skála arány meghatározásához használt 3D modell illesztés a detektált járművekre (forrás: [5])

A járművek sebessége a követésük során a képkockák közötti idő segítségével kerülnek kiszámításra. A javasolt módszer a 1.10 km/h-s átlagos sebességmérési hibát ért, mellyel az addigi state-of-the-art automatikus kalibrációs technikát 86%-kal, a manuális kalibrációt pedig 19%-kal teljesítette túl. A javasolt módszernek nincsenek megkötések a kamera elhelyezésére vonatkozóan, és tetszőleges nézőpontból használható. A járművek járműméretein alapuló megközelítéseknek hátrányaként azonban egyértelműen felhozható a finom szemcseméretű autómodell-osztályozó rendszer alkalmazásának szükségessége, a pontos járműtípus detektálásától való függés, illetve a valós, 3D-s modellek beszerzésének nehézsége.

Az eddig említett módszereken kívül léteznek olyan módszerek is, amelyek a képkockákon vagy az úttest megjelölt vonalakat vagy régiókat használnak a sebesség becsléséhez [6] [7]. Ezek a módszerek nem igénylik a kamerarendszer kalibrálását. A járművek mindig azonos távolságban kerülnek észlelésre, sebességüket az előre meghatározott virtuális vonalaknak vagy régióknak keresztezésekor, az időbeli mintavételi ráta segítségével (például képkocka sebesség)

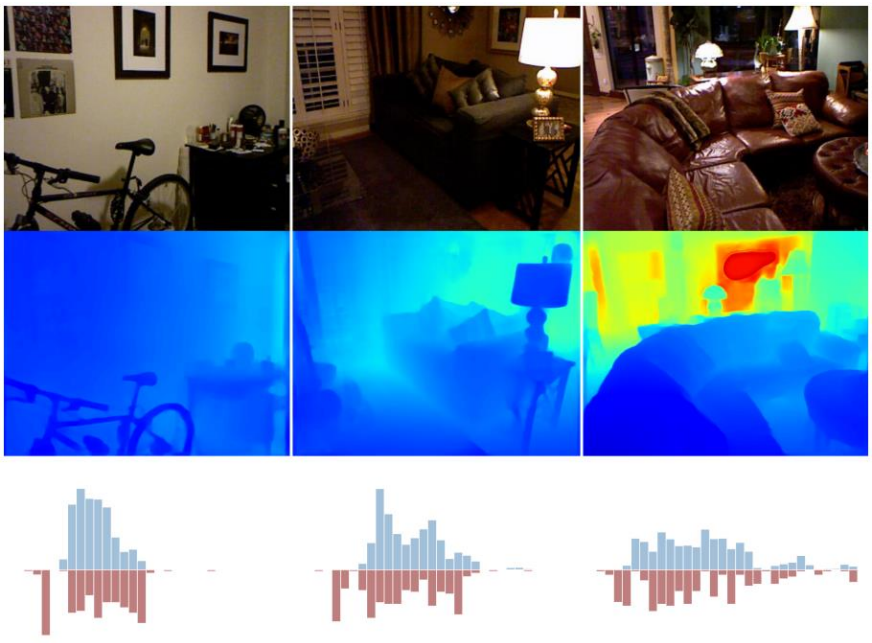
határozzák meg. A pontos távolságbecsléshez a jármű valamely részének talajra levetített pontját kell pontosan meghatározni. Ez a feladat összetett a perspektivikus korlátok, a térbeli és időbeli diszkretizációs problémák, az árnyékok stb. miatt. Ezeket a problémákat több behatolási vonal használatával lehet enyhíteni.

3.1.1 Metrikus mélységbecslés

Munkámban az előző fejezetben is említett pinhole kamera modell segítségével való távolságmérés alapú sebességbecslést alkalmazok, amikhez különböző metrikus mélységbecslő algoritmusokat használok fel. Ezeknek a bemutatása következik most.

AdaBins egy 2020-ban publikált mélységbecslő algoritmus, amely jó minőségű mélység térképek generálását kísérli meg RGB bemeneti képekből. A megoldást kifejlesztő szerzők munkájának motivációját az adta, hogy a meglévő konvolúciós rétegeken alapú architektúrák csak akkor dolgozzák fel a globális információkat egy képen, ha a tenzorok egy nagyon alacsony térbeli felbontást érnek el. Ezzel szemben ők nagy felbontással végzik a globális információk feldolgozását. Ennek megvalósításához a mélységi értékek eloszlásának elemzését és módosítását eszközölik.

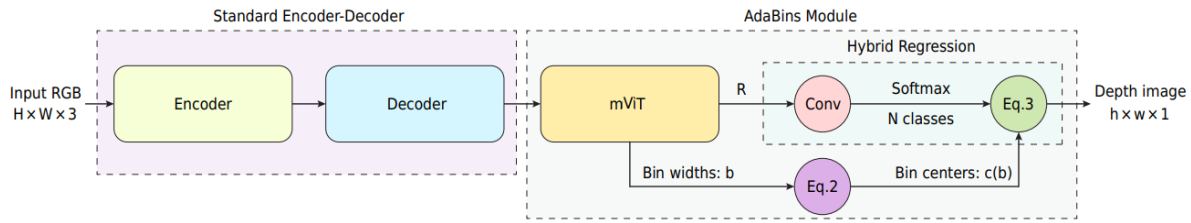
A szerzők rávilágítanak arra a problémára, hogy a bemeneti RGB képek mélységi eloszlása nagy mértékben változik, ahogy a 7. ábra is mutatja. Fent láthatóak az RGB bemeneti képek, középen az AdaBins által megjósolt mélység, majd alul az ground truth mélységértékeinek hisztogramja (kék) és a megbecsült adaptív mélység-pontok hisztogramja (piros), balról jobbra növekvő mélységértékekkel. Látható, hogy közeli képeknél a tárolók kisebb mélységértékeknél összpontosulnak, azonban távoli képeknél ez az eloszlás elterül az egész intervallumon.



6. ábra: Az AdaBins illusztrációja (forrás: [8])

Meglévő megoldások a bin szélességekre vagy előre meghatározott uniform, illetve log-uniform szélességeket használtak, vagy tanult szélességeket, amik viszont egyes adathalmazokon eltérőek. Az AdaBins olyan megközelítést alkalmaz, ahol a háló megtanul

adaptívan fókuszálni a mélységi tartomány azon régióira, amelyek nagyobb valószínűséggel fordulnak elő a bemenetei képen.



7. ábra: Az Adabins háló architektúrája (forrás: [8])

Az 7. ábra mutatja az AdaBins felépítését. Az architektúra két fő elemből áll: 1) egy enkóder-dekóder egységből, amely egy előre tanított EfficientNet B5 enkóderre lett építve és egy általános felmintavételező dekóderből, valamint 2) az adaptív bin szélesség becslő modulból. Az AdaBins modulban a Mini-ViT transzformer blokk kimenetéből származnak a kiszámított bin szélességek. A mélységtartomány bin-ekre van felosztva, és az egyes bin-ek középértékét képenként adaptívan becsüli meg a modul. A végső mélységértékek ezután ezen bin-ek középpontjainak a lineáris kombinációjaként kerül meghatározásra. Ez a megközelítés lehetővé teszi a modell számára, hogy alkalmazkodjon az egyes bemeneti képek sajátos mélységi jellemzőihez, javítva a mélységbecslés pontosságát.

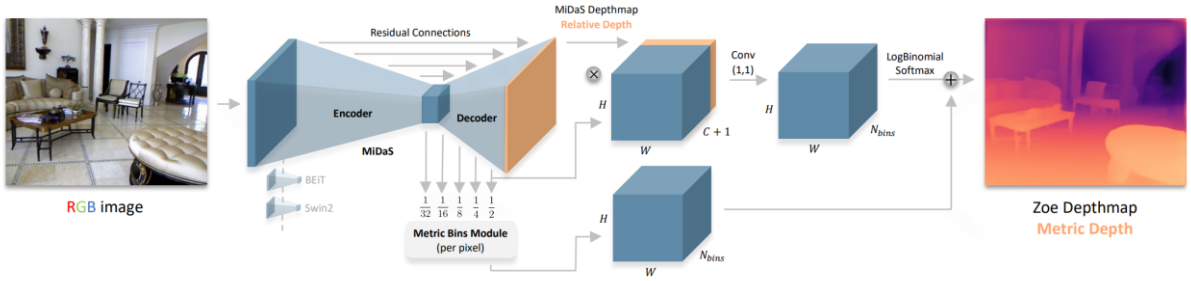
Az így kapott modellt pixel-szerinti mélységi hibával és a tároló középértékeinek sűrűségi hibafüggvényeinek összegével tanították, ahol az utóbbi 0.1-es súlyozással került beszámításra. A kész modell 78 millió paraméterből áll. A modell az NYU Depth v2 és KITTI adatbázis lett tanítva és kiértékelve: Az NYU Depth v2 [9] egy 120.000 beltéri jelenet 640x480 pixel felbontású mélység térképéből álló adathalmaz, amelyben a mélységek 10 méteres felső határon belül vannak. A KITTI adathalmaz [10] egy kültéri, mozgó járműről felvett sztereo képekből és 3D lézer felvételekből áll, körülbelül 1240x370 pixel nagyságúak és 80 méteres maximum mélységűek, viszont a mélységi térképei nagyon alacsony sűrűségűek és sok hiányzó adatot tartalmaznak.

Ezzel a megoldással az AdaBins state of the art megoldás lett 2020-ban: az NYU-Depth-v2 teszt adathalmazon elért 0.364-es és a KITTI adathalmazon elért 2.36-os négyzetes középértékű hibával megelőzte a BTS és DAV vetélytársait.

Az AdaBins bin alapú mélység érték diszkretizációs technikáját több arra építő módszer is használja: A LocalBins [11] a mélység eloszlásokat egy lokális szomszédságban vizsgálja, ahelyett, hogy az egész képen globálisan tenné azt. A BinsFormer [12] nevű megoldás pedig egy kiegészítő jelenetosztályozási lekérdezést tartalmaz a bin-ek generálásához, és egy több skálán alapuló stratégiát használ az adaptívan generált bin-ek finomítására. A PixelFormer [13] a mélységbecslést képpont-lekérdezésekként kezeli, amelyeket a kihagyásos figyelem segítségével finomítanak, és amelyeket a bin-központok előrejelzésére használnak a dekódolt jellemzők kihasználása nélkül.

Munkámban az AdaBins algoritmust is felhasználom az úttest feletti kamera képéből való mélységbecslésre. Azonban máris meglátjuk, hogy ennél modernebb és jobb teljesítményű megoldások is publikálva lettek azóta, például a ZoeDepth.

A ZoeDepth, mely a zero-shot transfer betűiből kapta nevének első felét az AdaBins-hoz hasonlóan az egy képből való mélységbecslés problémáját kísérli megoldani. A szerzők fő motivációja a relatív mélységbecslés és a metrikus mélységbecslés közötti szakadék áthidalása. A javasolt módszer célja a relatív mélységbecslés általánosítási teljesítményének kombinálása a mélységbecslést mérő metrikus skálával. A szerzők azzal érvelnek, hogy bár a relatív mélységbecslés lenyűgöző általánosítási teljesítményt mutat, hiányzik belőle a sok alkalmazáshoz szükséges metrikus mérőszám. A metrikus mélységbecslés ezzel szemben metrikus skálát biztosít, de általában gyenge az általánosítási teljesítménye. A ZoeDepth úgy kívánja leküzdeni ezeket a korlátokat, hogy előre betanítja a kódoló-dekódoló architektúrát egy adatkészlet relatív mélységének felhasználásával, majd az új metrikus tároló modul segítségével finomhangolja a modellt egy vagy több adatkészleten a metrikus mélység pontos előrejelzési képességéhez. A ZoeDepth-ben a zero-shot transfer arra utal, hogy a javasolt modell képes új tartományokra általánosítani anélkül, hogy további tanításra vagy finomhangolásra lenne szüksége. Konkrétan a szerzők megmutatják, hogy egyetlen, több tartományra kiterjedő metrikus mélységbecslési modelljük minden tartományban alkalmazható, beltéri vagy kültéri, szimulált vagy valós területeken, tartományspecifikus képzés vagy finomhangolás nélkül.

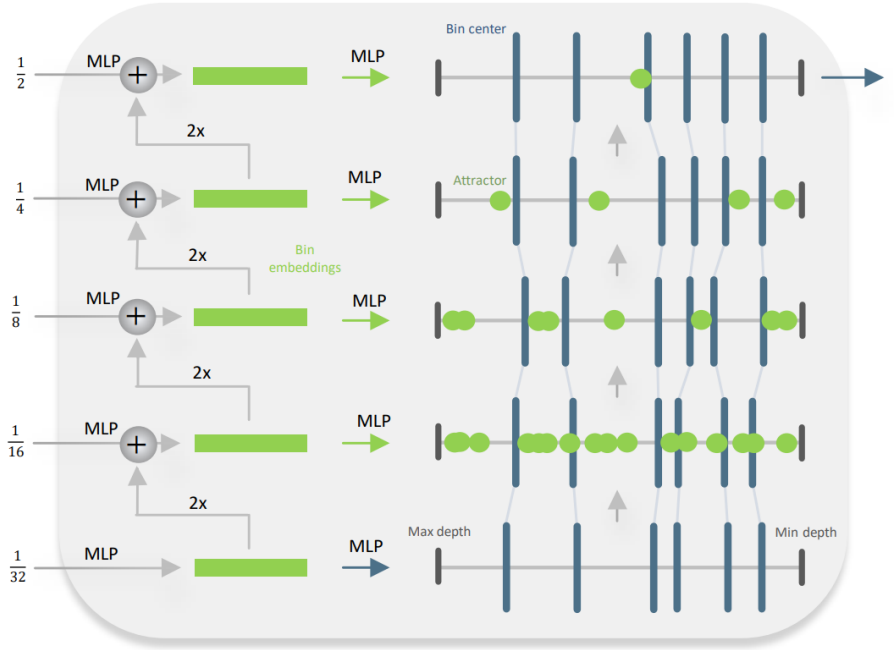


8. ábra: ZoeDepth architektúrája (forrás: [14])

A ZoeDepth az előző fejezetben bemutatott AdaBins architektúrára épít, és továbbfejleszti azt (8. ábra): a DPT architektúra kódolóját egy frissített transzformátor alapú gerinchálózatra cseréli és kiegészíti a dekódert egy metrikus tároló modullal a metrikus mélységbecsléshez. A metrikus tároló modul ugyanazon az adaptív tárolók elvét követi, amelyet eredetileg az AdaBins vezetett be.

Az attraktor rétegek a ZoeDepth újonnan javasolt építőelemei, amelyek a mélységintervallumon balra vagy jobbra mozgatva a tárolók több skálán történő finomítását valósítják meg. A modell a többskálás jellemzők segítségével megjósolja a mélységintervallumon azon pontok halmazát, amelyek felé a tárolóhelyek középpontjai vonzódnak. Minden egyes dekódoló rétegben egy MLP bemenetként veszi egy képpontnál a jellemzőket, és megjósolja az adott képpont pozíciójához tartozó n darab attrakciós pontok halmazát. A beállított tároló-középpontot ezután az eredeti bin-központ és az inverz attraktor-képlet által adott kiigazítás összegeként számítják ki. A szerzők azzal érvelnek, hogy az attraktoros stratégia azért előnyösebb a felosztással szemben, mert ez egy összehúzódó folyamat, míg a felosztás eredendően táguló. Az előrejelzésnek a dekódoló rétegekkel finomabbá és fókuszáltabbá kell válnia, amit az attraktorok anélkül érnek el, hogy helyi korlátokkal foglalkoznának.

A metrikus bins modul a ZoeDepth architektúra egyik kulcsfontosságú eleme, amely az adaptív binning elvét követi. A modul a DPT-architektúra dekóderéből származó többskálájú jellemzőket veszi bemenetként, és minden egyes képpontnál megjósolja a bin-középpontokat. A modul megtanulható paraméterek segítségével adaptívan igazítja a bin-ek méretét és pozícióját a jelenet geometriájához. Az adaptív binning elvének javasolt módosításai közé tartozik az attraktor rétegek, egy bin aggregációs stratégia és egy veszteségfüggvény használata. Az attraktor rétegek a mélységintervallumon lévő pontok egy olyan halmazának előrejelzésével finomítják a bin-középpontokat, amelyek felé a bin-középpontok vonzódnak. A bin-aggregációs stratégia a teljesítmény javítása érdekében több fejből származó előrejelzéseket aggregál. A veszteségfüggvény egy sima L1 veszteség és egy skálaváltozatlan mélységveszteség kombinációja, amely arra ösztönzi a modellt, hogy pontos és konzisztens mélységértékeket jósoljon különböző skálákon keresztül.



9. ábra: A metrikus bin modul felépítése (forrás: [14])

A ZoeDepth architektúrában a relatív mélység előrejelzésére a MiDaS előre betanított gerinchálózatát használják. A háló egy transzformátor-architektúrán alapul, amely különböző számítógépes látási feladatoknál bizonyítottan hatékony. Konkrétan a MiDaS gerinchálózat a BEiT-384-L transzformátor gerinchálózatot használja, amely 384 réteggel rendelkezik, és egy nagyméretű, változatos jelenetekből álló adathalmazon van előre betanítva. A BEiT-384-L gerinchálózat a MiDaS modell 344M paraméteréből 305M-et foglal el. A MiDaS-modellt a relatív mélységjósláshoz előzetesen betanítják egy adathalmazon, majd a metrikus mélységjósláshoz egy vagy több adathalmazon finomhangolják a javasolt metrikus bins modul segítségével. A szerzők megmutatják, hogy a MiDaS kódoló gerincének megválasztása jelentős hatással van a ZoeDepth modell paramétereinek számára, és hogy a MiDaS kódoló gerincének kicserélése csökkentheti a paraméterek számát és javíthatja a teljesítményt.

A szerzők a ZoeDepth több változatát tanítják és értékelik, amelyeket a következő konvenció szerint neveznek el: ZoeD-{RDPT}-{MFT}, ahol az RDPT a relatív mélység előképzéséhez használt adathalmazokat, az MFT pedig a metrikus mélység finomhangolásához használt

adathalmazokat jelöli. A szerzők a következő modelleket tanítják és értékelik: ZoeD-X-N, ZoeD-X-K, ZoeD-M12-N, ZoeD-M12-K és ZoeD-M12-NK.

Minden modell a timm által gyártott BEiT 384-L gerincet használja, amelyet előzetesen az ImageNet-en lett tanítva. A ZoeD-X-N és a ZoeD-X-K modelleket közvetlenül a metrikus mélységre finomhangolták a NYU Depth v2 és a KITTI modelleken, a relatív mélységbecsléshez szükséges előképzés nélkül. A ZoeD-M12-N és ZoeD-M12-K modellek a metrikus mélységre való finomhangolási szakasz előtt a relatív mélységbecslésre is tartalmaznak előképzést az M12 adathalmaz keverékén.

A NYU Depth V2 adathalmazon és a Virtual KITTI 2 adatkészleten is a ZoeDepth minden mérőszámon felülmúlja az AdaBins-t. A két módszer eredményeit a mutatja.

Algoritmus	**Virtual KITTI 2**	**NYU Depth v2**
AdaBins	5.420	0.364
ZoeDepth	**5.095**	**0.277**

1. táblázat: Az AdaBins és ZoeDepth által elért mélység becslési eredmények különböző benchmarkokon

Összességében a ZoeDepth mindkét adathalmazon jobb teljesítményt ér el, mint az AdaBins (1. táblázat), ami bizonyítja a javasolt metrikus bins modul és az attraktor rétegek hatékonyságát a mélységbecslési pontosság javításában. A szerzők azt is bemutatják, hogy a javasolt modell felhasználható új adatkészletekre, lenyűgöző eredményeket érve el előzetes tanítás vagy finomhangolás nélkül. Azt viszont meg kell jegyezni, hogy a ZoeDepth közel ötször annyi paramétert tartalmaz, mint az AdaBins, ez pedig a tanítási és futási teljesítményre is jelentős hatással van.

3.2 Rendszámtábla felismerés és leolvasás

Ha egy járműnek meghatároztuk a haladási sebességét, következő lépés az, hogy azt egy leolvasott rendszámtáblához rendeljük, a jármű azonosításának érdekében. A következőkben ilyen rendszámfelismerő megoldásokat ismertetek.

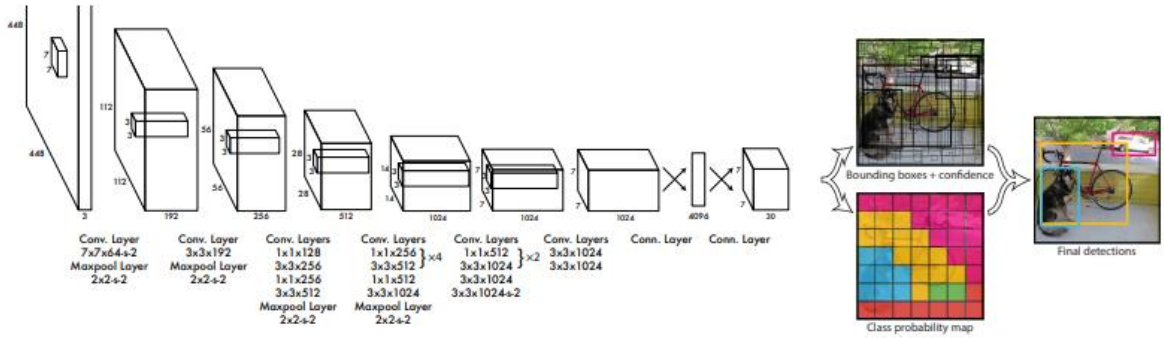
A rendszámtáblák felismeréséhez tradicionális és tanuló eljárások is találhatóak a szakirodalomban. A Luvizon és társainak korábban már ismertetett munkája [3] egy textúraosztályozó megoldásra épülő rendszámtábla detektálást valósít meg. A rendszer a rendszámtábla betűit alkotó karaktervonások gradiens eloszlási jellemzőinek detektálására specializálódott. A rendszer először azonosítja a nagyszámú, szövegnek minősített ablakot tartalmazó régiókat, majd kiválasztja a kép aljához legközelebb eső régiót, amely a legtöbb esetben megfelel a rendszámtáblának. A rendszámtáblát egyetlen tengelyhez igazított téglalap közelíti, amely a kiválasztott régió összes szövegablakát magába foglalja. Megjegyzik azonban, hogy detektált a téglalap nem mindig igazodik pontosan a rendszámtáblához, hiszen nem leolvasási, hanem követési feladatra van felhasználva. A szerzők megjegyzik továbbá, hogy rendszámtábla-felismerő módszerük képes kezelni a motorkerékpárok rendszámtábláit, amelyek eltérő méretekkel rendelkeznek, és egy helyett két szövegsort tartalmaznak, valamint kisebb betűket és számjegyeket, amelyek gyakrabban összeolvadnak egymással. [15]-ben a szerzők a kép HSV színtérbe transzformált változatán végeznek el morfológiai műveleteket, mint a dilatáció és zárás. A kapott képet ezután annak területével és képarányával analizálva

találják meg a lehetséges rendszámtáblákat. Megjegyzik, hogy különböző megvilágítási viszonyokban más-más küszöbértékekre lehet szükség a HSV kép csatornáinak szűrésére, amire saját algoritmust fejlesztettek ki. A megoldással state-of-the-art közeli eredményeket tudtak elérni.

Manapság az egyre több rendelkezésre álló adat és a gyorsabb grafikus kártyák térnyerésével a tanuló eljárások óriási figyelmet kapnak. Bármilyen objektum felismeréséhez manapság széleskörűen használt a YOLO (You Only Look Once) architektúra [16]. Éppúgy használható járművek felismerésére, mint ahogy rendszámtáblákra vagy éppen az azon lévő karakterekre.

Az első YOLO architektúra egy egységes, valós idejű tárgyfelismerő rendszer, amely abban különbözött a többi objektumdetektálási módszertől, hogy azok a detektáláshoz osztályozókat hasznosítottak újra, a YOLO viszont az objektumdetektálást regressziós problémaként fogalmazza meg a térben elkülönített befoglaló dobozok és a hozzájuk tartozó osztályvalószínűségek tekintetében. A nevét arról kapta, hogy a bemeneti képre csak egyszer néz rá az azon található objektumok detektálásához és lokalizálásához.

A rendszer a bemeneti képet egy S × S méretű rácsra osztja, B befoglaló dobozt predikál azok konfidencia értékeivel C különböző osztály számára rács elemenként. Minden befoglaló doboz 5 értékből áll: x, y, w, h, és konfidencia érték. Az (x, y) koordináták a doboz középpontjának koordinátáit jelentik a rácsháló cellájának határaihoz képest. A w szélesség és h magasság a teljes képhez viszonyítva van megbecsülve. A konfidencia érték képviseli az IOU-t (Intersection Over Union), tehát azt, hogy a predikált befoglaló doboz és a ground truth doboz mekkora átfedésben van egymással. Minden rács cella C darab feltételes osztály valószínűséget is predikál, ezeket a tesztelés során az individuális doboz konfidenciákkal összeszorozva, osztály-specifikus konfidencia értékek állnak elő minden dobozra. A modell architektúráját és működését illusztrálja a 10. ábra.



10. ábra: A YOLO modell működése (forrás: rubikscode.net)

A tanulás során a háló minimalizál egy veszteségfüggvényt, amely közvetlenül megfelel a felismerési teljesítménynek. A veszteségfüggvény figyelembe veszi mind a lokalizációs hibát (mennyire tér el a megjósolt határoló doboz a ground truth-tól), mind az osztályozási hibát (mennyire pontosak a megjósolt osztályvalószínűségek).

Inferencia során a hálózat egy bemeneti képből az előrejelzések tenzorát állítja elő. A jóslatokat ezután utólagosan feldolgozzák az egymást átfedő határoló dobozok és az alacsony megbízhatóságú észlelések eltávolítása érdekében, ezt non-maxima elnyomásnak (Non

Maximal Suppression) hívják. A megmaradt észlelések ezután megjelennek a bemeneti képen, a hozzájuk tartozó osztályvalószínűségekkel együtt.

A csúszó ablak- és régiójavaslat-alapú technikákkal ellentétben a YOLO a teljes képet látja a tanítás és a tesztelés során, így implicit módon kódolja az osztályokra vonatkozó kontextuális információkat, valamint az osztályok megjelenését. Ennek köszönhetően a YOLO sokkal gyorsabb megoldásnak bizonyult, mint meglévő objektum detektor társai. Azonban a lokalizációs hibája nagyobb volt, mint az akkori state-of-the-art Fast R-CNN megoldásnak.

A YOLOv2 [17] ezt a hiányosságot volt hivatott javítani. Az új változat a Darknet-19 gerinchálót használta és számos újdonságot eszközölt, többek között batch normalizációt, teljesen konvolúciós architektúrát és anchor dobozokat az átlagos precízió növeléséhez. Az anchor doboz célja, hogy felismerje a több különböző méretű objektumot, amelyek középpontja ugyanabban a cellában található. Ezekkel a módosításokkal a YOLOv1 63.4%-os eredményét a PASCAL VOC2007 adathalmazon 78.6%-ra javította.

[18] A YOLOv3 2018-ban jelentős változásokat és egy nagyobb architektúrát vezetett be, annak érdekében, hogy a state-of-the-art teljesítményét elérje, viszont gyorsaságát megtartsa. Az új modell az új Darknet-53 gerinchálózatot használja, aminek reziduális összeköttetései is vannak. A YOLOv3 több skálás predikcióra is képes, amivel különböző méretskálájú objektumokat jobban tudott detektálni. A modell az MS COCO adathalmazon state-of-the-art eredményt ért el, mindezt kétszer gyorsabb futási idővel társainál.

[18] A YOLOv4 számos új funkciót és fejlesztést vezet be a YOLOv3-hoz képest, beleértve a súlyozott-részleges-összeköttetéseket (Weighted-Residual-Connections), a szakaszokon átívelő részleges összeköttetéseket (Cross-Stage-Partial-connections) és a kereszt-batch normalizálást. Ennek eredményeképpen a YOLOv4 átlagos pontossága és másodpercenkénti képkockái 10%-kal, illetve 12%-kal nőttek a YOLOv3-hoz képest.

A 2020-ban kiadott YOLOv5 [19] az Ultralytics által lett fejlesztve PyTorchban. Az azt megelőző verzió fejlesztésein túl az Ultralytics AutoAnchor algoritmusát is felhasználja. A készítők 5 különbözően skálázott megoldást kínálnak: YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large) és YOLOv5x (extra large), amelyekben a konvolúciós moduloknak a szélessége és mélysége változó a specifikus applikációk és hardver igények kielégítésének céljából.

[18] A YOLOv6 és YOLOv7 verziók változó architektúrájukkal és új funkciójukkal tovább fejlesztik az előző rendszerek pontosságát. A YOLOv6 bevezet egy BiC modult az érzékelő nyakába, amely javítja a lokalizációs jeleket, és elhanyagolható sebességcsökkenés mellett teljesítménynövekedést eredményez. A YOLOv7 mind sebességben, mind pontosságban felülmúlja az összes ismert objektumdetektort az 5 FPS és 160 FPS közötti tartományban, és az összes ismert valós idejű objektumdetektor közül a legnagyobb pontossággal rendelkezik 56.8% AP 30 FPS vagy magasabb sebességgel.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

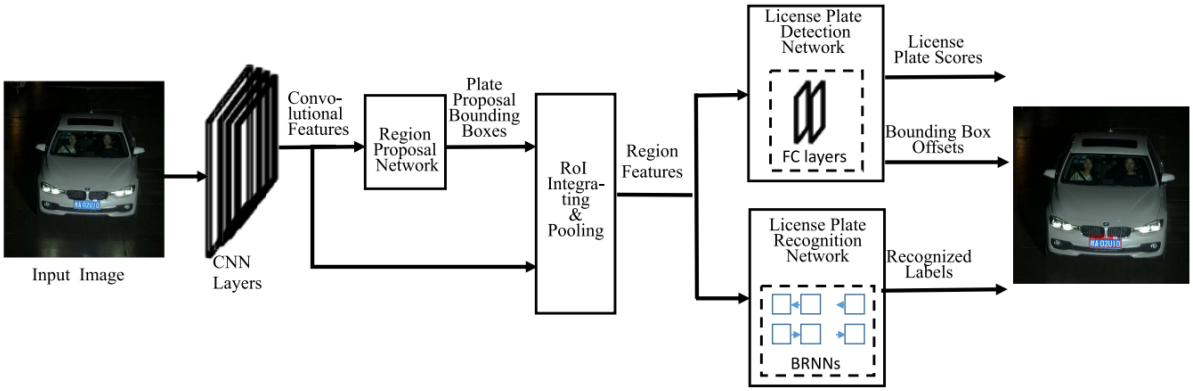
11. ábra: A YOLOv8 különböző modelljeinek tulajdonságai (forrás: [20])

A legfrissebb, YOLOv8 [20] verzió 2023 januárjában került kiadásra, ugyanattól a cégtől, amelyik a YOLOv5-t is kifejlesztette. A legújabb verzió az ötös verzióval megegyezően öt darab előre skálázott és tanított verzióval érhető el, a 11. ábra szemlélteti a különböző modellek tulajdonságait. A YOLOv8 a korábbi verziók sikerére épít, új funkciókat és fejlesztéseket vezet be a nagyobb teljesítmény, rugalmasság és hatékonyság érdekében. Támogatja a látásalapú mesterséges intelligencia feladatok teljes skáláját, beleértve a felismerést, szegmentálást, pózbecslést, követést és osztályozást. A YOLOv8 futtatható a parancssori felületről (CLI), vagy telepíthető PIP csomagként is. Ezenkívül számos integrációval rendelkezik a címkézéshez, a képzéshez és a telepítéshez.

A YOLO architektúráját a rendszámtábla leolvasásban is széleskörűen használja a szakirodalom [21]. A [22] CR-Net nevű karakter szegmentáló és felismerő konvolúciós háló is YOLO alapokon fekszik. A CR-Net a YOLO és FAST-YOLO architektúrákat használja fel, pár módosítással. A szerzők megjegyzik, hogy az eredeti YOLO modellnek fix bemeneti és kimenete képaránya van, ami 1:1. A brazil rendszámok, amiknek a leolvasására a módszer ki lett fejlesztve viszont 3:1 arányúak, ezért az eredeti modell paramétereit meg kellett változtatni a helyes működéshez. A szerzők a háló bemeneti és kimeneti felbontásával is problémába ütköztek. A YOLO 13x13 pixeles kimenete nem volt elég részletes ahhoz, hogy 7 horizontálisan egymás melletti objektumot (karaktert) szegmentálni tudjon. Ennek orvoslására a szerzők a horizontális felbontást majdnem megháromszorozták, amellyel a háló 30x10 pixel méretű képeket adott kimeneteként. Bemeneti felbontásként a szerzők 240x80-as nagyságot alkalmaztak az eredeti 416x416-os nagyság helyett, ezért architektúrális változtatásokat kellett eszközölniük. A max pooling réteg számát lecsökkentették, illetve az eredeti háló csak első 11 rétegét használták fel a dimenzió redukálás csökkentése érdekében. Ezen kívül a háló végére még 4 réteget csatoltak a több nem-linearitás elérése érdekében.

A CR-Net-et több publikált munka is felhasználta. Laroca és társai [23] több országhoz tartozó rendszámtáblákat tartalmazó adathalmazokon is kiértékelték, melyeken legtöbb esetben 95% feletti pontosságot értek el. A modell különböző YOLO architektúrákkal saját készítésű indiai rendszámtábla adathalmazon is kiértékelésre került [24], a legjobb architektúrának a YOLOv4 bizonyult 98,3%-os pontossággal. Ezek a YOLO alapú módszerek gyors futási idővel bírnak, jellemzően 5-20 ms-os inferencia időkről beszélhetünk. Ugyanakkor a szegmentációt alkalmazó megoldások hajlamosak eltorzult karaktereknél a karakterhatárok téves azonosítására, illetve érzékenyebbek különböző megvilágítási és árnyékolási körülményekre, mint más szegmentáció mentes megoldások.

A rendszámtábla leolvasásban gyakran találunk különböző RNN (rekurrens neurális háló) alapú megoldásokat is [21], amelyekkel szegmentáció nélküli felismerés is megvalósítható. Li és társai olyan megoldást [25] javasoltak, amely egy konvolúciós neurális hálózatot használ a rendszámtáblák felismeréséhez és egy rekurrens neurális hálózatot a rendszámtáblák leolvasásához. A karakterek szegmentálásának elkerülése érdekében a szerzők a karakterfelismerési problémát szekvenciacímkézési problémaként kezelték. Kétirányú rekurzív neurális hálózatokat (BRNN) és hosszú rövidtávú memóriát (LSTM) használtak ehhez.



12. ábra: RNN-t és LSTM-et felhasználó rendszámfelismerő modell architektúrája (forrás: [25])

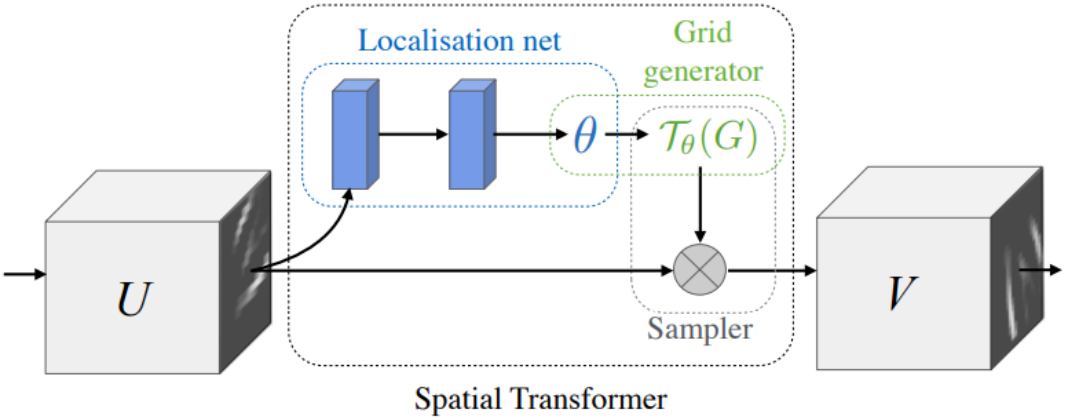
Az LSTM egy új cellaszerkezetet, az úgynevezett memóriacellát, és három multiplikatív kaput (azaz bemeneti kaput, felejtési kaput és kimeneti kaput) használ, amelyek szelektíven képesek hosszú ideig tárolni az információt. A rendszámtábla szövege egy karaktersorozatnak tekinthető, amelynek a feldolgozására az LSTM kiváló módszer. A javasolt megoldás többféle rendszámtábla adathalmazon kiértékelésre került, legtöbb helyen 95%-nál jobb pontossággal, viszont lassú, 300-400 milliszekundumos futási idővel. Ezen modellek komplexitása mind a tanítás, mind pedig az inferencia közbeni gyorsaság vesztésben érvényesül.

3.2.1 Térbeli transzformációs háló

Mint az az elméleti bevezető fejezetben is említésre került, a konvolúciós hálóknak vannak térbeli gyengeségeik, melyeket csak megfelelő augmentáció és modell komplexitás mellett lehet kiküszöbölni. Ha az a célunk, hogy a rendszer bemeneti képekről tudjon információt kinyerni, akkor kívánatos, hogy szét tudja választani a tárgy pozícióját és részeinek deformációját a textúrájától és az alakjától. A CNN-ekben a helyi max-pooling rétegek bevezetése segített ennek a tulajdonságnak a kielégítésében, mivel lehetővé tette, hogy a hálózat térben némileg invariáns legyen a jellemzők helyzetét illetően. A max-pooling jellemzően kis térbeli támogatottsága (pl. 2 × 2 pixel) miatt azonban ez a térbeli invariancia csak a max-pooling és a konvolúciók mély hierarchiájában valósul meg, és a CNN-ben konvolúciós rétegek aktivációi valójában nem invariánsak a bemeneti adatok nagy transzformációira. A CNN-ek e gyengesége annak köszönhető, hogy csak korlátozott, előre meghatározott pooling-mechanizmussal rendelkeznek az adatok térbeli elrendezésében bekövetkező változások kezelésére.

Ennek a gyengeségnek az orvoslására tesz kísérletet a térbeli transzformációs háló vagy modul (Spatial Transformer Network), mely egy általános neurális háló architektúrájába beépítve térbeli transzformációs készségeket biztosít annak. A térbeli transzformer modul lehetővé teszi az adatok térbeli manipulálását egy neurális hálózaton belül a jellemzőtérképek aktív térbeli

átalakításával, magától a jellemzőtérképtől függően, külön tanítási felügyelet vagy az optimalizálási folyamat módosítása nélkül. Az átalakítás ezután a teljes jellemzőtérképen (nem lokálisan) történik, és magában foglalhatja a méretezést, a vágást, az elforgatást, valamint a nem merev deformációkat. Ez lehetővé teszi a térbeli transzformátorokat tartalmazó hálózatok számára, hogy ne csak a kép legrelevánsabb régióit válasszák ki, hanem ezeket a régiókat egy kanonikus, elvárt pozícióba is transzformálják, hogy a következő rétegekben egyszerűsítsék a felismerést.



13. ábra: A térbeli transzformációs háló architektúrája (forrás: [26])

A Spatial Transformer architektúra három komponensből áll, amiket a 13. ábra illusztrál: egy lokalizációs hálózatból, egy rácsgenerátorból és egy mintavevőből. A lokalizációs hálózat fogadja az U bemeneti jellemzőtérképet, és meghatározza a jellemzőtérképre alkalmazandó térbeli transzformáció θ paramétereit. A megbecsült transzformációs paramétereket ezután a τ_θ transzformációval egy G mintavételi rács létrehozására használja, amely azon pontok halmaza, amelyekből a bemeneti térképet a transzformált kimenet előállításához mintavételezni kell. Végül a jellemzőtérképet és a mintavételi rácsot a mintavevő bemenetével a rácspontokban mintavételezett V kimeneti térképet állítja elő. E három komponens kombinációja alkotja a térbeli transzformátort.

A lokalizációs háló által meghatározott transzformációs paramétereknek száma változhat a paraméterezett transzformáció típusának függvényében, egy 2D-s affin transzformációnál a θ paraméter mátrix 6 dimenziós. A lokalizációs háló bármilyen formát ölthet, legyen az teljesen kapcsolt vagy konvolúciós háló, viszont tartalmaznia kell egy végső regressziós réteget a transzformációs paraméterek produkálásához. A transzformációt írja le a következő egyenlet:

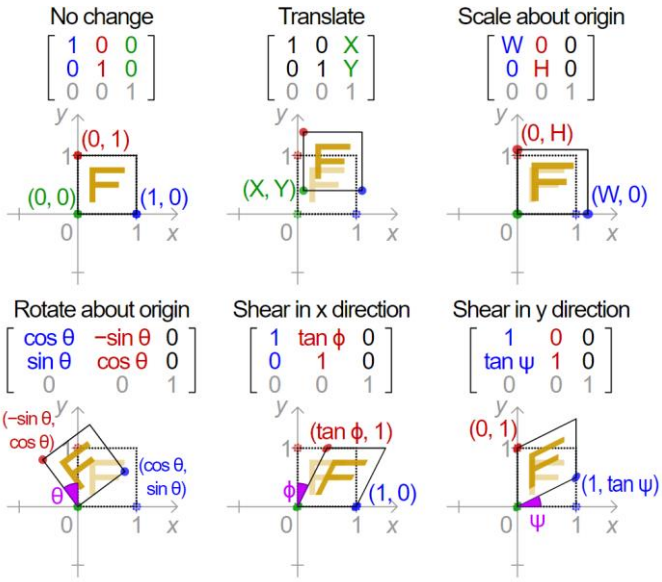
$$\theta = f_{loc}(U), \tag{2}$$

ahol, a θ a transzformációs paraméter mátrix, f_{loc} a lokalizációs háló függvény, U pedig a bemeneti jellemzőtérkép. 2D-s affin transzformációt feltételezve, θ egy 2x3 méretű, tanulható affin transzformációs mátrix lesz, jelöljük ezt A_θ-val:

$$A_\theta = \begin{bmatrix} s_x & r_y & t_x \\ r_x & s_y & t_y \end{bmatrix}, \tag{3}$$

ahol az s_x és s_y paraméterek a méret skálázó, r_x és r_y a nyírási, t_x és t_y pedig a transzlációs értékek x és y irányba.

A 14. ábra szemléletesen illusztrálja az affin transzformációk működését példákon keresztül.



14. ábra: Affin transzformáció példák (forrás: wikipédia.org)

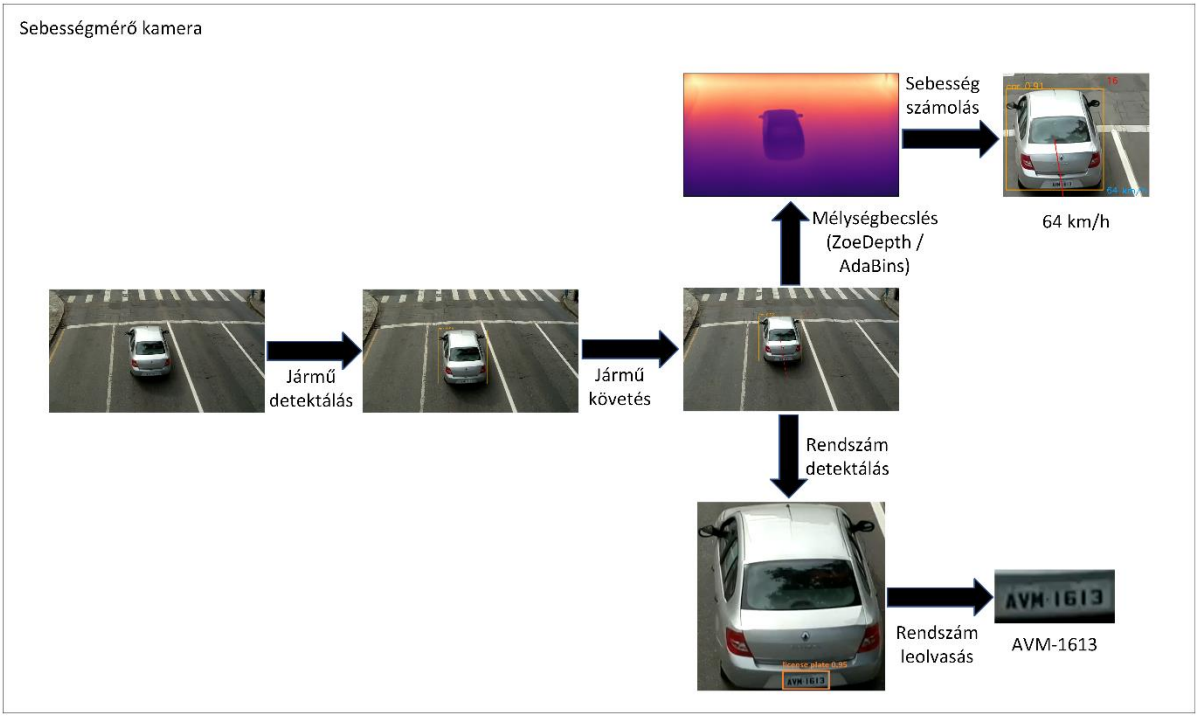
Az STN egy önálló modul, amely tetszőleges ponton és tetszőleges számban beilleszthető egy CNN-architektúrába, így térbeli transzformátorhálózatok jöhetnek létre. A modul számítási szempontból nagyon gyors, és nem rontja a tanítási sebességet, naiv használat esetén nagyon kis időbeli többletköltséget okoz. A térbeli transzformátorok elhelyezése a CNN-ben lehetővé teszi a hálózat számára, hogy megtanulja, hogyan kell aktívan átalakítani a jellemzőtérképeket, hogy segítsen minimalizálni a hálózat általános költségfüggvényét a tanítás során. Az egyes tanítási minták átalakításának módjára vonatkozó tudás a tanítás során a lokalizációs hálózat súlyaiba (és a térbeli transzformátor előtti rétegek súlyaiba is) kerül tömörítve.

A transzformációs hálók hasznosságára a szerzők több valós példát is felhoznak: Az ilyen hálókkal tanított konvolúciós hálók a torzított kézzel írott karakterek, házszámok felismerését is jelentősen javította [26]. A gyakorlatban ezek a hálók mély neurális hálókkal foglalkozó Python könyvtárakkal is implementálhatóak [27]. Ilyen hálót használtam a munkámban a rendszámok helyes orientációjának meghatározásában és azokba való transzformációhoz, amellyel jelentősen növelni tudtam annak pontosságát.

4 Javasolt megoldások

4.1 Módszertan

Az előző fejezetben egyes bemutatott megoldások problémáit feltérképezve, illetve egyes ismertetett megoldások felhasználásával a munkámban egy olyan sebességmérő kamera rendszer kifejlesztését tűzöm ki célul, amelynek sebességbecslő része a körülményes és időigényes kalibrációs folyamatokat, illetve költséges hardverkövetelményeket kiváltja, rendszámfelismerő része pedig jó pontosságú megoldást ad a különböző torzításokkal korruptált rendszámtáblák leolvasására, mindezt egy könnyű felépítésű, taníthatóságú és futtathatóságú neurális hálóval.



15. ábra: A tervezett rendszer folyamatábrája

A tervezett rendszer folyamatábráját a 15. ábra szemlélteti. A rendszer a bemeneti képen járműveket keres a YOLOv8 objektumdetektor segítségével. A megtalált járművekre befoglaló dobozt illeszt, majd ezt továbbítja a követő algoritmusnak. Az objektumkövető algoritmus egy előre meghatározott képkocka után, melyeken detektálva volt az objektum egy példányként elmenti azt. A rendszer ilyen példányokhoz (tehát járművekhez) társít minden hozzá tartozó és megszerzett információt. A detektált jármű befoglaló doboza a rendszámtábla detektorhoz kerül, ami egy másik, rendszámtáblákra finomhangolt YOLOv8 objektumdetektor. A rendszámtáblára illesztett befoglaló doboz ezután a karakter-szegmentáció nélküli karakter felismerő konvolúciós hálóhoz kerül, ami kimeneteként a leolvasott rendszámtáblát adja string formátumban. A detektált járművet tartalmazó képkockán a mélységbecslő algoritmus mélységtérképet ad vissza kimeneteként, amin az egyes járművek rendszámtábláját lefedő képpontok átlagolásából a jármű kamerától való távolságát határozza meg a rendszer. A pinhole kamera modell segítségével ezt a kamera koordinátarendszeréből a világ koordinátarendszerébe transzformálva elmozdulást határozok meg, ebből pedig a kamera másodpercenkénti képkocka specifikációjával sebességet számolok. Az egyes járműhöz tartozó releváns információkat, mint

a sebessége, leolvasott rendszámkódja vagy a megtett útvonalát a követett példányokhoz társítom és naplózom azokat.

A rendszer gyakorlati megvalósításához a következő eszközöket használom: A programkód Python 3.10 verzióban íródik és a 2. táblázatban felsorolt főbb könyvtárakat alkalmazza. A neurális háló tanításhoz egy Dual Nvidia RTX 2080 Super és egy RTX 3070 GPU-val ellátott rendszert veszek vegyesen igénybe, tesztelésükhöz a 3000-es szériát használom.

Alkalmazott könyvtár	Felhasználás célja
torch, torchvision	Neurális háló implementáció
opencv, PIL	Videó és kép manipuláció
matplotlib	Ábrázolás és illusztráció
wandb	Neurális háló tanításának és tesztelésnek naplózása
pandas	Adat manipuláció
sys, os, shutil, time	Rendszer specifikus kisegítő függvények

2. táblázat: A felhasznált Python könyvtárok

4.2 Rendszámtábla felismerés

4.2.1 A javasolt rendszámfelismerő modell

A bemutatott publikációktól eltérően nem szegmentációs, sem RNN alapú megoldással közelítem a rendszám felismerés problémáját. Ahogy azt a 15. ábra is mutatja, a modellem YOLO hálózatokra támaszkodik a járművek és a rendszámtáblájuk detektálásához. Az eredményezett kivágott rendszámtáblákat egy teljesen konvolúciós hálóval kísérlem meg leolvasni. A következőkben az egyes komponenseket részletesen mutatom be.

4.2.2 Jármű detektálás és követés

A videón található járművek detektálásához a YOLOv8-at használom fel. Az Ultralytics által fejlesztett háló előtanítva rendelkezésre áll különböző skálázott verziókban a készítők GitHub oldalán [20]. A háló az MS COCO [28] nevű adathalmazon került előtanításra, amely egy objektumfelismerésre, szegmentálásra, kulcspontok felismerésére használt 328.000 képből álló adathalmaz. Az adathalmazban 91 féle osztály található, köztük járművek, kültéri területek, állatok, ételek vagy bútor objektumok. A jármű csoporton belül találhatóak kerékpár, autó, motorkerékpár, repülőgép, busz, vonat, teherautó és hajó osztályok. A közúton megtalálható járműtípusokkal, tehát autókkal, motorkerékpárokkal, buszokkal és teherautókkal előre tanítottan érkezik a YOLOv8 modell. Az elérhető öt skálázott modell közül kísérleteim során az m méretű háló kielégítő pontosságot nyújtott elfogadható gyorsaság (~20 ms RTX 3070-es GPU-n) mellett, ezért nem finomhangoltam tovább külön csak járműveket tartalmazó adathalmazon. A háló működését a 3.1 fejezetben ismertetett videókból álló adathalmazon verifikáltam. Az s méretű háló pár milliszekundummal gyorsabban futott, viszont a videóképeken a járműveket alkalmanként nem detektálta vagy helytelen kategóriába osztályozta. A predikcióhoz 0.4-es minimum konfidencia értékküszöböt használtam, 640x640 pixel méretű bemeneti képet, az NMS számára pedig 0.2-es IoU értéket. Előbbi értéket a fals pozitív eredmények kiszűrésére használtam, utóbbit pedig egyes duplikált befoglaló dobozt

eredményező predikció kiszűréséhez, például a motoros nélküli és az azzal együtt predikált eredmények egyesítésére.

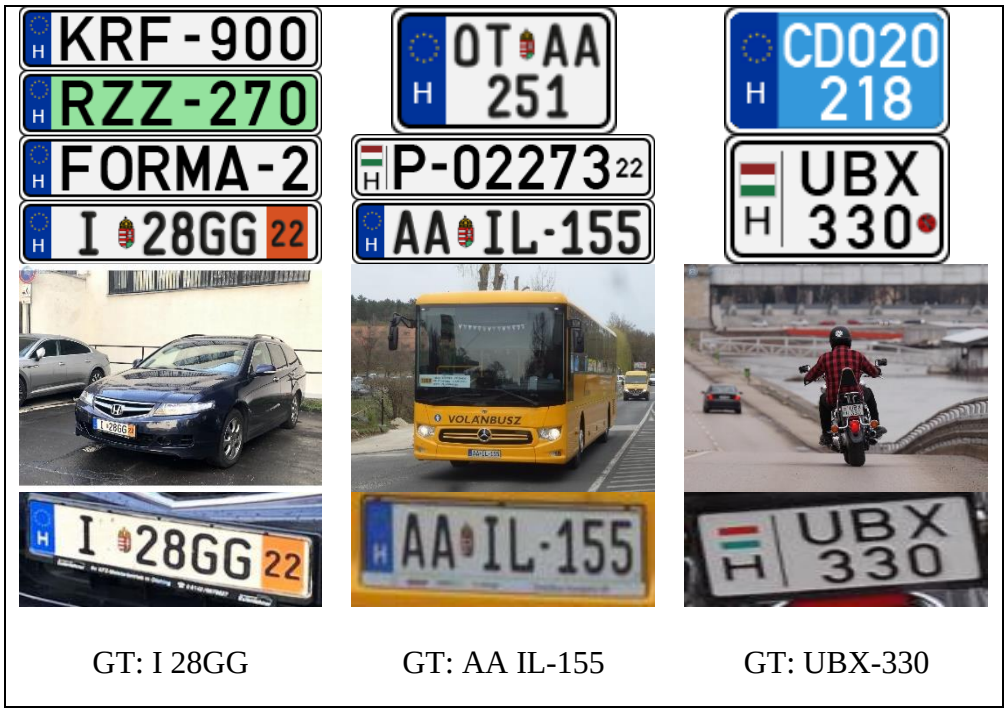
A detektált járműveket a későbbi sebességméréshez és egyéb információk hozzárendeléséhez követni kell, azaz példányosítani. Ehhez a SORT [29] (Simple Online and Realtime Tracking) követő algoritmust használom. Az algoritmus a jármű detekciók mátrixát várja bemenetként és három állítható paraméter alapján példányosítja azokat a követés érdekében: A max_age paraméter meghatározza azon maximum képkockák számát ameddig az algoritmus vár egy követett objektum jelenlétére a képen, mielőtt inaktívvá tenné azt a példányt. A min_hits küszöbérték paraméter azt határozza meg, hogy hány darab egymás utáni detekció szükséges egy követett példány előállításához. Az iuo_treshold küszöbérték paraméter pedig két befoglaló doboz közötti átfedés minimum értékét határozza meg ahhoz, hogy egy detekció egy adott példányhoz legyen társítva. A paramétereket max_age=2, min_hits=5, iou_threshold=0.3 értékekkel vettem fel. A rendszer a követett példányokhoz rendeli az adott járműről begyűjtött információkat, mint például annak sebessége, megtett trajektóriája vagy leolvasott rendszámtábla kódja.

4.2.3 Felhasznált adathalmaz

A járműdetektor kimeneteként kapott befoglaló dobozok kivágásával a járműre közelített képet kaptam. Ez volt a bemenete a rendszámtábla detektornak, amihez szintén a YOLOv8 objektumdetektort használtam fel. Mivel azonban a YOLO nincs előtanítva rendszámtáblák detektálására, illetve, ha előre is volna, az egyes országokban használt táblák jelentősen eltérő kinézettel rendelkeznek, ezért nem magyarországi rendszámtáblákon tanított detektorok jelentősen alacsonyabb pontossággal működnének, ezért saját adathalmazon tanítottam be azt. A következőkben a rendszámdetektáláshoz, illetve később a leolvasó neurális háló tanításához és kiértékeléséhez is használt adathalmazt mutatom be.

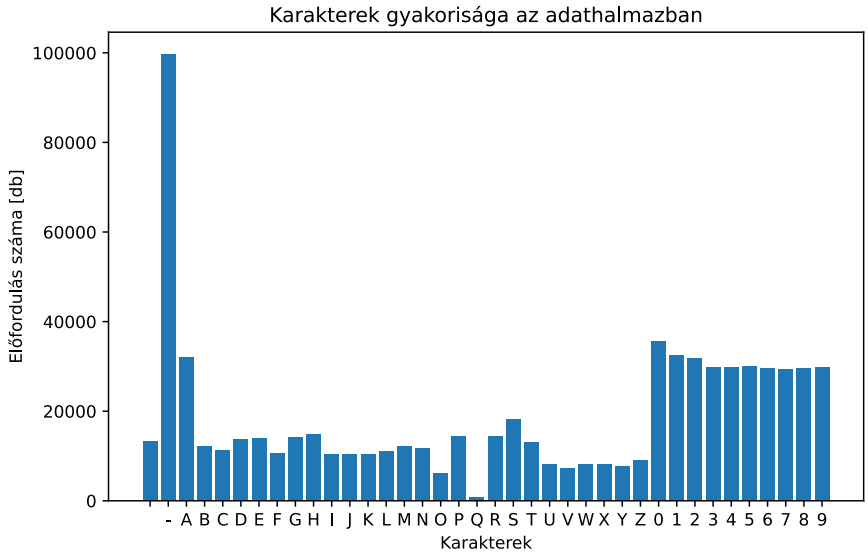
Az adatbázis a platesmania.com [30] webhelyről került beszerzésre, amely a legnagyobb járműrendszámokkal és minden velük kapcsolatos témával foglalkozó internetes projekt, fotógalériájukban összesen több, mint 21 millió fotóval, köztük közel 400.000 magyar rendszámtáblával. A beszerzett adathalmaz 2 fő részből áll. 1) Az egyik 100.000 darab különböző felbontású, kamerával készült képből áll, melynek mindegyike minimum egy járművet és annak rendszámtábláját ábrázolja valós viszonyok között 2) A másik része az ezen járművekhez tartozó rendszámtáblák 100.000 darab számítógéppel generált és renderelt képéből áll. Az adathalmaz tartalmaz régi és új formátumú, zöld (elektromos és hibrid), sárga (bérfuvarozás), kék (diplomáciai járművek), old-timer (OT kezdetű), ideiglenes (I kezdetű), illetve egyedileg igényelt rendszámtáblákat is. A rendszámok 96%-a egysoros, 4%-a pedig többsoros elrendezésű. Az összes rendszám magyarországi rendszám. A valós járműveket ábrázoló képek 75%-ának 1440x1080 pixeles felbontása volt, ezért a maradék 25%-ot is átméreteztem erre a felbontásra. A renderelt egysoros képek felbontása 235x50 pixel, a többsorosoké pedig (~100-130)x(~60-90) pixel.

Az adathalmazból mintaképeket mutat a 16. ábra, felülről lefelé sorban először renderelt rendszámtáblák, alatta a járművekről készített képek, alatta a járművek rendszámtáblái kivágva, majd legalul a hozzá tartozó címke kód.



16. ábra: Példaképek a felhasznált adathalmazból

Az adathalmazban minden képnek a ground truthja a hivatalosan nyilvántartott rendszám kód, tehát a rendszámokon esetlegesen nem látható kötőjellel és szóközzel ellátott formátum. A 17. ábra szemlélteti a karakterek előfordulási gyakoriságát a rendszámokon, ahol az első oszlop a szóköz karaktert jelöli.



17. ábra: A rendszámtáblák ground truth-jaiban előforduló karakterek gyakoriságai

A képeken látható járművek rendszámtábláinak azonban nem voltak jelölve a befoglaló doboz ground truth-jai. Ehhez egy korábbi, 1000 képből álló, befoglaló dobozokkal ellátott adathalmazt használtam fel a rendszámtábla detektor YOLO modell iniciális tanítására. Ezzel adekvát teljesítményű hálóhoz jutottam, amit az eredeti adathalmazon inkrementálisan és iteratívan tanítva, címke adatokat tudtam generálni. Az eljáráshoz magas 0.9-es konfidencia

küszöböt használtam a fals pozitív eredmények kiszűrésére, így az eljárás végén csak 700 darab címkézetlen mintám maradt, amelyeket manuálisan címkéztem fel. Egynél több járművet ábrázoló képnél csak az a befoglaló doboz került felcímkézésre, amelyiknek a rendszámtáblájához renderelt kép is tartozott. Ennél a pontnál az egyes járműveket illusztráló képeknek nem csak a rendszámtábláknak a kódja volt elérhető, mint ground truth, hanem az azt befoglaló dobozok koordinátái is.

4.2.4 Rendszámtábla detektor modell

A rendszámtábla detektáláshoz, mint fentebb is említettem a YOLOv8 objektumdetektort használtam. Az adathalmazt a renderelt képek felbontásának segítségével szétosztottam egy- és többsoros részekre. Mint ahogy arról később írni fogok, ez azért volt fontos, mert a karakter leolvasó modellt teljesítményét jelentősen javította. A rendszámtábla detektor modellt tehát két osztályú címkével tanítottam be, így az megtanulta megkülönböztetni az egysoros és többsoros táblákat. Az adathalmazt 80-10-10 arányban tanító-validációs-teszt alhalmazokra osztottam. A tanítást 30 epochon keresztül végeztem, 640x640-es bemeneti képfelbontással és 16-os batch mérettel és a YOLO-ba beépített automatikus augmentációs technikákkal.

4.2.5 Rendszám leolvasó modell

A rendszámokra illesztett befoglaló dobozokat kivágva, a rendszámtáblákra közelített képeket kapunk a meglévő renderelt táblák mellé. Mindkét részhalmaz fontos szerepet játszik modell tanításában.

Layer (type)	Input Shape	Param #	Tr. Param #
Conv2d-1	[1, 3, 32, 80]	4,736	4,736
MaxPool2d-2	[1, 32, 26, 74]	0	0
ReLU-3	[1, 32, 13, 37]	0	0
Conv2d-4	[1, 32, 13, 37]	25,632	25,632
MaxPool2d-5	[1, 32, 9, 33]	0	0
ReLU-6	[1, 32, 4, 16]	0	0
Linear-7	[1, 2048]	65,568	65,568
ReLU-8	[1, 32]	0	0
Linear-9	[1, 32]	198	198
Conv2d-10	[1, 3, 32, 80]	864	864
Conv2d-11	[1, 32, 32, 80]	9,216	9,216
Conv2d-12	[1, 32, 32, 80]	18,496	18,496
ReLU-13	[1, 64, 16, 40]	0	0
BatchNorm2d-14	[1, 64, 16, 40]	128	128
Conv2d-15	[1, 64, 16, 40]	36,864	36,864
Conv2d-16	[1, 64, 16, 40]	36,864	36,864
Conv2d-17	[1, 64, 16, 40]	73,856	73,856
ReLU-18	[1, 128, 8, 20]	0	0
BatchNorm2d-19	[1, 128, 8, 20]	256	256
Conv2d-20	[1, 128, 8, 20]	147,456	147,456
Conv2d-21	[1, 128, 8, 20]	147,456	147,456
Conv2d-22	[1, 128, 8, 20]	295,168	295,168
ReLU-23	[1, 256, 4, 10]	0	0
BatchNorm2d-24	[1, 256, 4, 10]	512	512
Conv2d-25	[1, 256, 4, 10]	589,824	589,824
Conv2d-26	[1, 256, 4, 10]	589,824	589,824
Conv2d-27	[1, 256, 4, 10]	1,180,160	1,180,160
ReLU-28	[1, 512, 2, 5]	0	0
BatchNorm2d-29	[1, 512, 2, 5]	1,024	1,024
Conv2d-30	[1, 512, 1, 10]	19,494	19,494

Total params: 3,243,596

Layer (type)	Input Shape	Param #	Tr. Param #
Conv2d-1	[1, 3, 32, 320]	4,736	4,736
MaxPool2d-2	[1, 32, 26, 314]	0	0
ReLU-3	[1, 32, 13, 157]	0	0
Conv2d-4	[1, 32, 13, 157]	25,632	25,632
MaxPool2d-5	[1, 32, 9, 153]	0	0
ReLU-6	[1, 32, 4, 76]	0	0
Linear-7	[1, 9728]	311,328	311,328
ReLU-8	[1, 32]	0	0
Linear-9	[1, 32]	198	198
Conv2d-10	[1, 3, 32, 320]	432	432
Conv2d-11	[1, 16, 32, 320]	2,304	2,304
Conv2d-12	[1, 16, 32, 320]	4,640	4,640
ReLU-13	[1, 32, 16, 160]	0	0
BatchNorm2d-14	[1, 32, 16, 160]	64	64
Conv2d-15	[1, 32, 16, 160]	9,216	9,216
Conv2d-16	[1, 32, 16, 160]	9,216	9,216
Conv2d-17	[1, 32, 16, 160]	18,496	18,496
ReLU-18	[1, 64, 8, 80]	0	0
BatchNorm2d-19	[1, 64, 8, 80]	128	128
Conv2d-20	[1, 64, 8, 80]	36,864	36,864
Conv2d-21	[1, 64, 8, 80]	36,864	36,864
Conv2d-22	[1, 64, 8, 80]	73,856	73,856
ReLU-23	[1, 128, 4, 40]	0	0
BatchNorm2d-24	[1, 128, 4, 40]	256	256
Conv2d-25	[1, 128, 4, 40]	147,456	147,456
Conv2d-26	[1, 128, 4, 40]	147,456	147,456
Conv2d-27	[1, 128, 4, 40]	295,168	295,168
ReLU-28	[1, 256, 2, 20]	0	0
BatchNorm2d-29	[1, 256, 2, 20]	512	512
Conv2d-30	[1, 256, 2, 20]	589,824	589,824
Conv2d-31	[1, 256, 2, 20]	589,824	589,824
Conv2d-32	[1, 256, 2, 20]	1,180,160	1,180,160
ReLU-33	[1, 512, 1, 10]	0	0
BatchNorm2d-34	[1, 512, 1, 10]	1,024	1,024
Conv2d-35	[1, 512, 1, 10]	19,494	19,494

Total params: 3,505,148

18. ábra: A karakter leolvasó modellek architektúrái. Kék színnel látható a térbeli transzformációs háló modul, sötétzölddel egy konvolúciós leskálázó blokk, világoszölddel az ezekből a blokkokból épített leskálázó modul, pirossal pedig az osztályozó konvolúciós réteg.

A rendszámtáblák karaktereinek leolvasásának problémáját szegmentáció mentesen egy teljesen konvolúciós háló alapú megoldással közelítem meg. Az egysoros és többsoros táblákra is külön-külön modellt alkalmazok. A modellek architektúráit a 18. ábra szemlélteti, ahol a bal

oldalt a többsoros modell architektúrája látható, jobb oldalt pedig az egysorosé. Kék színnel látható a térbeli transzformációs háló modul, sötétzölddel egy konvolúciós leskálázó blokk, világoszölddel az ezekből a blokkokból épített leskálázó modul, pirossal pedig az osztályozó konvolúciós réteg.

A modellek elején található térbeli transzformációs hálók, ahogy korábban ismertetve lett, a neurális hálónak segít a különböző térbeli transzformációk megtanulásához, amikkel a geometriai invarianciáját növelhetjük. A lokalizációs moduljához, ami az első 6 réteget jelenti, általános konvolúciós hálóknál használt jellemzőkinyerő, illetve dimenziócsökkentő rétegeket használok. Az affin paraméter predikcióhoz a 7-9-es rétegek kerülnek felhasználásra, amikben teljesen kapcsolt rétegekkel határozza meg a modell a szükséges térbeli transzformációs paramétereket. Ezeket az architektúrában nem jelölt affin rács és rács mintavételező funkciókkal applikálom a bemenetre. Ennek kimeneti felbontása az eredeti bemenetével megegyezik, tehát az azt követő felépítés és logika változatlan maradhat.

A konvolúciós blokkok a modellek fő építőelemei, négy darab ilyen található a többsoros modellben, az egysorosban pedig öt darab. A blokkban három darab konvolúciós réteg követi először egymást, ahol az első kettő 1-es padding-el és 1-es stride-al a bemenettel megegyező szélességű és magasságú kimenetet generál a csatornaszám szimultán növelésével. A harmadik konvolúció felelős a kép leskálázásáért, ezt 2-es stride-al teszi, miközben megduplázza a kimeneti csatornák számát. Egy-egy blokk a kép magasságát és szélességét a felére csökkenti a bemenethez képest, miközben képjellemzőket tanul belőle. A blokk végén a nemlinearitásért felelős ReLU réteg és a gyorsabb és stabilabb tanulás érdekében batch normalizációs réteg helyezkedik el.

A modellek fix méretű bemenetekkel dolgoznak: az egysoros modell 32x320-as felbontású bemenetet vár, a többsoros pedig 32x80-asat. Ezeket a konvolúciós blokkok egyenként elfelezik, amíg az egysoros modell esetében 1x10 pixel méretet, többsoros modell esetében pedig 2x5 pixel méretet nem eredményeznek. A modellek végén található osztályozó réteg szintén egy konvolúciós réteg 1x1-es méretű kernel mérettel és a maximum felismerni kívánt karakterek számával megegyező kimeneti mérettel. Ezt a számot magyar rendszámok esetében 10-re állítottam, amibe a 100.000 minta mindegyike belefért. Ez a dimenzió a többsoros modell esetében 2x5-re formálódik át, hiszen két sorban helyezkednek el a karakterek ezeken a rendszámtáblákon. Ez tehát azt jelenti, hogy a rendszer előre megadott és fix maximális karakterszámot predikál kimenetként és a bemeneti kép felbontását is ilyen arányban várja (32 x 10 = 320, 32 x 5 = 80). Ezek a fix méretek a tesztképek méretéből és a modell megfelelő mélységének elérése érdekében lettek kiválasztva. A fix 10 predikált karakter 38 féle értéket vehet fel: 26 betűt az angol ABC-ből, 10 féle számjegyet, kötőjelet és szóközt. A szóköz célja az üres vagy nem kódot tartalmazó területek helyettesítése. Ezekből előáll egy megengedett karakter string, amit a 3. táblázat felső sora mutat. A rendszám karaktereket balról kezdve kódolom, 10-nél rövidebb rendszám esetén szóközökkel töltöm ki az üres helyeket. A 3. táblázatban található pár példa a kódolásra.

„ ABCDEFGHIJKLMNOPQRSTUXYVWZ0123456789-”.	
AA AA-001	tensor([1, 1, 0, 1, 1, 37, 27, 27, 28, 0])
KMX-067	tensor([11, 13, 22, 37, 27, 33, 34, 0, 0, 0])

3. táblázat: A rendszámok karakter kódolása

A modellek tanításához kereszt-entrópia veszteséget és az Adam sztochasztikus optimalizálót használtam. A tanulási rátához lépésfüggvény alapú ütemezőt használtam, 0.001-es kezdeti, 0.0005-ös decay-el, 150-es lépésközzel és 0.99-es szorzóval. Kísérletezéseim szerint az STN háló alacsonyabb tanulási rátával sokkal gyorsabban konvergált, mint például 0.01-el. A modellek 50 epochon át lettek tanítva. A tanító adatokkal szembeni túlilleszkedés (overfitting) ellen augmentációt használtam. A használt tanítási stratégia a következő volt:

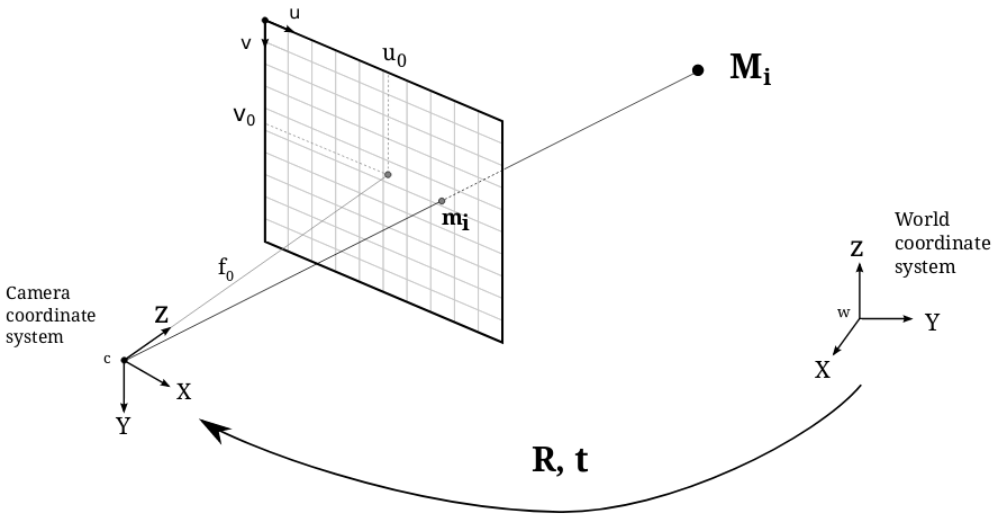
1. Tanítás a renderelt képeken augmentációval
2. Transzfertanítás a renderelt képeken augmentációval és random transzformációval
3. Transzfertanítás valós kivágott képeken augmentációval

Az első tanítás alatt a modellek megtanulják ideális környezetben a karakterek mintáit, körülbelüli elhelyezkedésüket. A második fázisban az STN modult tanítom főként, itt a modell megtanulja, hogy a perspektivikusan torzított rendszámokat és karakterek hogyan olvassa le. A harmadik fázisban létező és alapból torzított rendszámokon tanul meg a modell valós helyzetekre általánosítani. Mind a három fázisban 0.3-as világosság, 0.3-as kontraszt és 0.05-ös szaturációs augmentációt és normalizációt használtam. A második fázisban ezek mellé 0.5 gyakoriságú és 0.5 erősségű véletlen perspektivikus transzformációt is eszközöltem. Az STN háló súlyainak inicializálásánál az affin mátrixban az eltolásért és a nyírásért felelős paramétereket nulláztam, skálázó paramétereit pedig 1-re állítottam. Ez azért kellett, mert ezen paraméterek randomizálásával a háló túlságosan nagy transzformációt alkalmaz a bemeneten, ez pedig a tanulást hátráltatja, vagy egyenesen lehetetlenné teszi.

4.3 Sebességmérés

4.3.1 A javasolt sebességmérő modell

A sebességmérést valós koordinátarendszerben mért távolságmérésen keresztül közelítem meg. Ehhez videokamera belső paramétereit és a pinhole kamera modellt használom fel.



19. ábra: A pinhole kamera modell (forrás: google.com)

A pinhole kamera modellt a 19. ábra szemlélteti és a következő egyenletekkel írható le:

$$u = f_x \frac{x}{z} + p_x; \qquad v = f_y \frac{y}{z} + p_y; \tag{4}$$

Az egyenletben u és v pixel koordináták, x, y és z térbeli koordinátái az objektumnak méterben, f_x és f_y a kamera fókusztávolsága x és y irányba pixelben, p_x és p_y pedig a principális pont koordinátája x és y irányban pixelben. A videóképen a járművek sebességméréséhez azok rendszámtábláit követem. Ahhoz, hogy a járműnek a pozícióját (pontosabban a rendszámtábla pozícióját) világ koordinátarendszerében meghatározzuk, az egyenlet átrendezése szükséges:

$$x = \frac{u - p_x}{f_x}\, z; \qquad y = \frac{v - p_y}{f_y}\, z; \tag{5}$$

A detektált rendszámtábla közepét véve ismertek az u és v koordináták a kamera síkjában. A p_x és p_y principális pontokat feltételezem, hogy a kamera síkjának közepén helyezkednek el, ezek a paraméterek tehát a videó horizontális és vertikális felbontásának a felét veszik fel. A fókusztávolság a (6)-os egyenlet alapján meghatározható az adott kamera típusát ismerve:

$$f = \frac{\left(\frac{f_0}{s} * i\right)}{\alpha};\ \alpha = \frac{r}{r_0}\ ; \tag{6}$$

Az egyenletben f a fókusztávolság pixelben, f_0 a fókusztávolság milliméterben, s a kameraszenzor mérete, i a kép felbontása, α az arányossági tényező, ami a kamera valós felbontása r és a felvett média felbontása r_0 arányából adódik ki. Az α tényezőre azért volt szükség, mert a használt kamerával felvett videó széle le lett vágva a szükséges felbontás eléréséhez, tehát az kamera szenzor felbontása nem egyezett meg a felvett videó felbontásával.

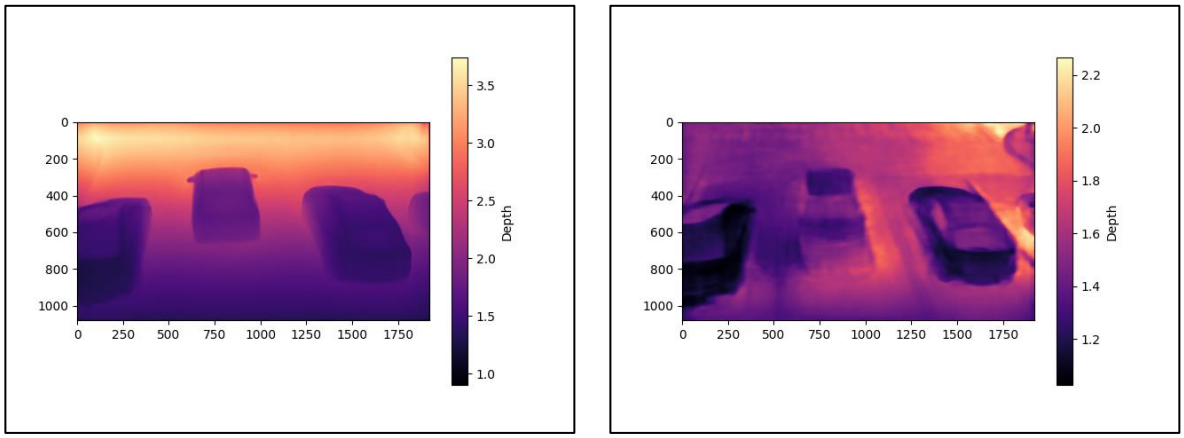
Az (5)-ös egyenletből csak a z paraméter, az objektum kamerától való távolsága szükséges még az objektum x és y koordinátájának meghatározásához, ez viszont ismeretlen.

Ennek a távolságnak a becsléséhez kísérlek meg neurális háló alapú mélységbecslő algoritmusokat felhasználni. Ehhez a 3.1.1 fejezetben bemutatott AdaBins és ZoeDepth metrikus távolság megbecslésére képes algoritmusokat használnom és hasonlítom össze. Ha egy adott képkockán minimum egy jármű és annak rendszámtáblája detektálásra került, akkor lefut a mélységbecslő algoritmus.

A ZoeDepth-et torch hub-on keresztül könnyedén be lehet tölteni. A fejlesztők 3 modellt kínálnak a GitHub oldalukon, ezek a ZoeD_N, ZoeD_K, illetve ZoeD_NK. Az összes modell 12 mélységtérképpel ellátott adathalmaz keverékén lett relatív mélységek predikálására előtanítva, majd az N végződésű modell a beltéri NYU Depth v2 adathalmazon, a K végződésű a KITTI kültéri adathalmazon, az NK végződésű pedig mindkettő említett adathalmazon lett tanítva metrikus mélységek becslésére. A modellek működésének verifikálása közben az NK variáns bizonyult a legpontosabb és legkonzisztensebb relatív mélységbecslőnek, ezért a továbbiakban ezzel a modellel tesztelem a rendszert.

Az AdaBins hálónál a fejlesztők által javasolt módszert használtam az inferenciához, amivel egy inferencia segítő függvény a bemenet alapján meghatározza a bin-középpontokat. Az előtanított hálók közül az NYU Depth v2 adathalmazon tanítottat használtam fel a KITTI-s

helyett, mert ez sokkal konzisztensebb eredményt nyújtott az egyes video képkockák változása mellett.



20. ábra: A mélységbecslő algoritmusok által predikált mélységtérképek (Bal oldalt a ZoeDepth, jobb oldalt az AdaBins)

Az 20. ábra szemlélteti a mélységbecslő algoritmusok predikált mélységtérképét, bal oldalt a ZoeDepth, jobb oldalt pedig az AdaBins eredménye található, mellettük lévő színskálán pedig a méterben értendő kamerától vett távolságok. Látható, hogy a ZoeDepth sokkal jobban el tudja különíteni az egyes járműveket az aszfalttól, mint az AdaBins, ami még a felfestett sávokra is más mélységet predikált, mint a közvetlenül velük határos aszfaltra. Észrevehető továbbá, hogy mindkét megoldás jelentősen torzított skálával dolgozik, a valóságban a minimum és maximum távolságok körülbelül 4 és 13.5 méter között lehetnek. A hálókat nem finomhangoltam saját adathalmazon. Ennek oka egy részről az, hogy célom volt az algoritmusok out-of-the-box teljesítményét megvizsgálni. A példaképeken látható, hogy a ZoeDepth, ami a publikációjában hivatkozik az adott helyzethez tanítatlan állapotban is egyedülálló általánosítási képességére, valóban remek relatív mélységképeket generál a régebbi AdaBins-hez képest. A másik ok, hogy a szakirodalomban nem találtam olyan RGB-D (mélységtérképes) adathalmazt, ami az úton haladó járműveket felülről felvéve tartalmaz mintákat. A KITTI adathalmaz nagyobb távolságai miatt bár előnyösnek bizonyulhatna, mégsem készíti fel az algoritmust felülről való becslésre, illetve az adathalmazban található címkék sem minden esetben pontosak az akvizíciós technikák miatt.

A helytelen skálázásnak a problémáját manuális korrekcióval közelítettem meg. Ehhez egy skálázási tényezőt határozok meg, amihez a valóságban méterben ismert pontok távolságát és annak a képen pixelben mért távolságából aránytényezőt számítok ki. A valós pontokat a sebességmérés teszteléséhez használt, sebesség ground truthokat biztosító aszfaltba épített induktív hurokérzékelők aszfalton lévő jelöléseit használom. Ennek a sávval párhuzamosan vett hossza 4.8 méter. A két széléhez tartozó pontnak a világ koordinátarendszerébe transzformált pozíciójához a mélységtérképeknek az adott ponthoz tartozó értékét veszem a z paraméter értékének, amiből a (5)-es egyenlettel a pontok x és y koordinátáját kapom meg. Ezeknek a pontoknak a valóságban mért és ismert távolságát az imént rekonstruált távolsággal való leosztásával megkapjuk az arányossági tényezőt, ami a mélységbecslés helytelen skáláját tükrözi. Ezzel az értékkel felszorozva az eredeti mélységtérkép pontjait a valósághoz sokkal közelebbi eredményt kapunk. Az arányossági tényezőt a videókép alatt folyamatosan naplózom és bizonyos időintervallumonként átlagolva frissítem.

A mélységtérképek az esetek 10-20%-ban más és más skálázást vesznek fel. Ezt szimplán a kép kompozíció módosítása okozza. Ezt a kis számú esetet a naplózott arányossági tényező segítségével ki lehet szűrni. A tényezőnek egy bizonyos küszöbértékén kívüli eseteit, így az ahhoz tartozó mélységtérképet is a rendszer automatikus elutasítja és a következő képkockára ugrik. Ha az arányossági tényező a küszöbértékeken belül van, akkor az eredeti mélységtérkép azzal az értékkel skálázódik a helyes mélységtérkép elérése érdekében. A konzisztencia érdekében a keletkező mélységtérképeket egy bizonyos intervallumba normalizálom, amit [4, 13.5] -nek vettem fel a videón haladó járművek valós tengelytávjaiból megközelítve azt.

A korrigált mélységtérképet felhasználva a járművek rendszámtábláihoz tartozó valós koordinátákat meg lehet határozni. Ezeknek a koordinátáknak az egymást követő képkockák közötti Euklideszi távolságát a (7)-os egyenlet alapján lehet kiszámítani:

$$d(p,q) = \sqrt{\left(\begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} - \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix}\right)^2} \tag{7}$$

Az egyenletben a p és q a két pont, melyek közötti távolságra vagyunk kíváncsiak. A távolságot meghatározva és a kamera másodpercenkénti képkocka sebességét felhasználva pillanatnyi sebességét tudunk számolni.

A meghatározott sebességeket naplózva, 3 vagy több elemnél azok átlagolásra kerülnek, a jármű sebességének pontosabb meghatározásához. Az egyes képkockák és az átlagos sebesség közötti különbségeket csak egy bizonyos küszöbértéken belül fogadja el a rendszer, ezt $\pm 5\%$-ra állítottam be, az esetleges hamis predikciók kiszűrésére.

5 Eredmények

5.1 Rendszámtábla felismerés

A rendszámtáblák detektálásához és azok leolvasásához is ugyanazt a saját adathalmazt használtam, amit a 4.2.3-as fejezetben ismertettem. Az adathalmazt 80-10-10 arányban tanító-validációs-teszt alhalmazokra osztottam, amelynek 10.000 képből álló teszt halmazán végeztem az összes tesztelést.

5.1.1 Rendszámtábla detektálás

A 4. táblázat mutatja a különböző méretű modellek teszt eredményeit. Látható, hogy mind a három modell pontosságban nagyon közel van egymáshoz, a legjobb pontosságúnak az medium méretű háló bizonyult, leggyorsabbnak pedig a nano. A legjobb választásnak viszont az small méretű hálót tekintettem, minimális gyorsaság vesztés mellett viszonylag jó pontosságot ért el. A metrikákat vizsgálva megjegyzendő, hogy a használt adathalmazban vannak olyan minták, ahol több jármű van a képen, viszont mindenhol csak egy jármű rendszámtáblájához tartozik befoglaló doboz címke. Ennek eredményeképpen a precision és recall értékek torzítottak, a valóságban magasabb értékeket vesznek fel.

Teszt metrika	YOLOv8n	YOLOv8s	YOLOv8m
Precision	0.98	<u>0.981</u>	**0.982**
Recall	0.971	<u>0.974</u>	**0.975**
mAP50	0.985	<u>0.986</u>	**0.988**
mAP50-95	0.956	<u>0.961</u>	**0.966**
Inferencia sebesség (RTX 3070) [ms]	**9.2**	<u>12.5</u>	21.8

4. táblázat: Különböző rendszámtábla detektor modellek teszt metrikái (legjobb **félkövér**, második legjobb <u>aláhúzott</u> karakterekkel)

5.1.2 Rendszámtábla karakter leolvasás

A rendszámtábla leolvasó modellekkel külön-külön elért eredményeket mutatja az 5. táblázat.

Teszt metrika	**Egysoros modell**	**Többsoros modell**
Renderelt képeken teljes rendszám teszt pontosság	99.94%	97.53%
Renderelt képeken karakter leolvasási teszt pontosság	99.99%	99.62%
Renderelt képeken teljes rendszám teszt pontosság persp. transzformációval	99.79%	96.98%
Renderelt képeken karakter leolvasási teszt pontosság persp. transzformációval	99.97%	99.56%
Valós képeken teljes rendszám teszt pontosság	97.82%	82.69%
Valós képeken karakter leolvasási teszt pontosság	99.52%	96.35%
Kombinált valós képeken teljes rendszám teszt pontosság	**97.27%**	
Kombinált valós képeken karakter leolvasási teszt pontosság	99.40%	

5. táblázat: A karakter leolvasó modellek eredményei

Látható, hogy az egysoros modell a renderelt képeken, mind perspektivikus transzformációval, mind anélkül közel hibátlan teljesítményt ért el karakterfelismerésben, kicsit alacsonyabb teljesítményt az egész rendszámtábla pontos felismerési képességében. Valós képeken nagyon jó, 97.8%-os pontosságot ért el, amivel a 3.2 fejezetben bemutatott módszerekkel hasonló teljesítményt nyújt.

A többsoros modell ennél kicsit rosszabbul teljesít, ennek az oka lehet a táblákon nem látszódó kötőjel kihagyása vagy a többsoros táblák általános komplexebb felépítése egysoros társaiknál.

Mindkét modell kiemelkedően gyors futási idővel bír a relatíve kicsi, 3-3.5 milliós paraméterszám miatt. A hálók 8-10 milliszekundumos) inferencia idővel dolgoznak (RTX 3070-es GPU-val).

A rendszámfelismerő rendszer azonban nem tudja előre megmondani melyik modellt kellene használni egy bemeneti rendszámképen. Ehhez a YOLO rendszámdetektort, ami különbséget tud tenni az egysoros és többsoros táblák között, kombinálni kell ezekkel a modellekkel. Az így kapott rendszer az tábladetektor és osztályozó kimenetének függvényében az egyik vagy másik modellnek adja tovább a kivágott rendszámtáblát. A karakter felismerő modellen kimeneti konfidencia értékeit megvizsgálva, egy bizonyos küszöbérték alatt mindkét modellel elvégezhető a leolvasás, ezzel minimális plusz futási idő árán, kiküszöbölhetjük a táblaosztályozó rész hibáit. Ezzel a módszerrel az egész rendszámfelismerő rendszert end-to-

end tekintve 97.20%-os teszt pontosságot kaptam, ami a szakirodalomban használt megoldásokhoz képest is kiváló eredmény.

A 6. táblázat mutatja a rendszerrel készített ablációs vizsgálatot. A módszer segítségével megtudhatjuk az egyes technikák befolyását a végső rendszer eredményébe.

Variáns	**Rendszámfelismés end-to-end pontossága (teljes rendszám)**
Csak egysoros modell	84.72%
Csak egysoros modell + STN	94.42%
Egysoros/többsoros modell + STN	**97.20%**

6. táblázat: Az egyes variánsokról készített ablációs vizsgálat

Az eredmények alapján látható, hogy az egy modellel, tehát csak az egysoros modellel való rendszám felismerés 85%-os pontossága viszonylag alacsony. Ezt a térbeli transzformációs háló a konvolúciós háló gyengeségeinek feljavításával jelentősen feljebb tornázza. A 21. ábra szemlélteti az STN működését, kiválóan látható, hogy a modul a szinte olvashatatlanságig eltorzult képekből is teljesen felismerhető eredményt készít. Két modellel, az egysoros és többsoros rendszámtáblák külön-külön detektálásával és leolvasásával a rendszer end-to-end pontossága 97.2%, ez pedig state-of-the-art módszerekkel is versenyképes eredmény.

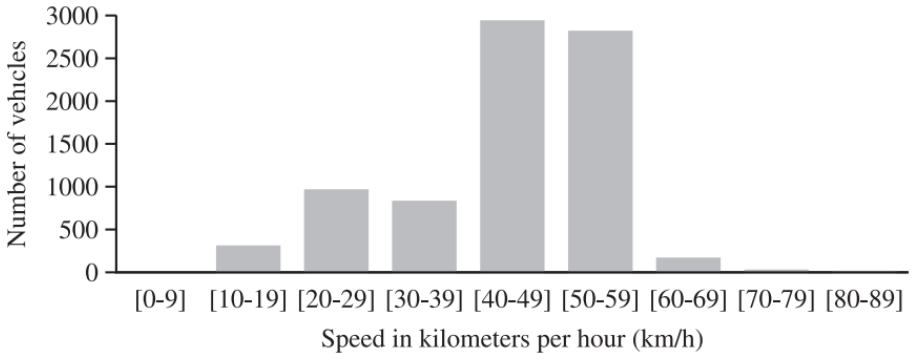
21. ábra: A STN működésének illusztrációja. Felül az eredeti, alul pedig az STN által transzformált verziójuk.

5.2 Sebességmérés

A kifejlesztett sebességmérő rendszer pontosságát a 3.1-es fejezetben ismertetett homográfiai mátrix alapú megoldáshoz [3] tartozó adathalmazon értékelem ki.

22. ábra: Egy példa képkocka az adathalmazból, ahol piros téglalapok jelölik az induktív hurokdetektorok elhelyezkedését, a sárga régió pedig az én rendszerem sebességmérési területét (forrás: [3])

A szerzők 20 videóból álló adathalmazt gyűjtöttek össze, amelyet a Raspberry Camera Module v2 nevű, 5 megapixeles CMOS-képérzékelővel rögzítettek, 1920 × 1080 pixeles képfelbontással, másodpercenként 30.15 képkocka sebességgel. A videókat az időjárási és felvételi körülményeknek megfelelően 5 sorozatra osztották. Minden videóhoz tartozik egy egyszerű XML-formátumú ground truth fájl, amely tartalmazza az egyes járművek első rendszámtáblájának határoló dobozait, valamint az egyes járművek tényleges sebességét. A ground truth sebességeket egy induktív hurokdetektoron alapuló, nagy pontosságú sebességmérővel nyerték, amelyet a brazil nemzeti mérésügyi hivatal (Inmetro) megfelelően kalibrált és jóváhagyott. Megjegyzendő, hogy a videókban vannak olyan járművek, amelyeknek nem látható a rendszámtáblája, és hogy a sebességmérő néha nem tudja megfelelően hozzárendelni a sebességet egy járműhöz.



23. ábra: Az adathalmazban lévő járművek sebességeinek eloszlása (forrás: [3])

Az 23. ábra a járművek sebességének eloszlását mutatja az adathalmazban (az adott útvonalon 60 km/h a sebességkorlátozás). A szerzők megállapítják, hogy a motorkerékpárok kihívást jelentenek a ground truth sebességmérő számára, amely csak az esetek 43%-ában tudta megmérni a sebességüket (szemben a közönséges járművek 92%-ával).

A jármű sebességét relatíve távol mértem a ground truth értékeket származtató induktív hurokdetektor helyzetétől, a mérés területét a 22. ábra sárga régiója mutatja. Ennek oka az, hogy a mélységbecslő algoritmusok ezen a területen konzisztensebb és pontosabb eredményt adtak, mint a képen felfelé. Ez tehát azt jelenti, hogy a kiértékelés során csak azok a járművek veendők figyelembe, amelyek közel konstans sebességgel, kis lassulással vagy gyorsulással változtatták sebességüket. A videókat készítő kamera egy lámpás kereszteződésnél helyezkedik el, ez azt jelenti, hogy egyes járművek a videókban megállnak a kép közepén. Ezeket a járműveket kivettem az értékelésből, hiszen gyorsításuk során az általam mért terület és a hurokdetektorok által mért terület megtétele alatt jelentősen felgyorsul a jármű, ami fals eredményekhez vezet.

A teszteléshez az adathalmaz subset01 alhalmaz első videóját, illetve a subset02a alhalmaz első és második videóját használtam.

Teszt metrika	**Egység**	**Százalék**
Összes nem megálló jármű	262 db	100%
Nincsen ground truth	20 db	7,6%
Nem sikerült a mérés	42 db	16%
Átlag sebességkülönbség	**-0.36 km/h**	**-0.06%**
± 5%-on belül	**94 db**	**44%**
± 10%-on belül	**166 db**	**78%**
± 10%-on kívül	**47 db**	**22%**
-3 és 2 km/h közötti sebességkülönbség	**102 db**	**47,7%**
nagyobb, mint 2 km/h sebességkülönbség	**76 db**	**35,2%**
kisebb, mint -3 km/h sebességkülönbség	**37 db**	**17,1%**
± 10%-on belül és 40 km/h feletti	**143 db**	**73,7%**
±5%-on belül és 40 km/h feletti	**83 db**	**42,8%**
Átlag sebességkülönbség	1.33 km/h	4.01%
± 5%-on belül	23 db	11,5%
± 10%-on belül	51 db	25,2%
± 10%-on kívül	162 db	74,5%
-3 és 2 km/h közötti sebességkülönbség	29 db	13,7%
nagyobb, mint 2 km/h sebességkülönbség	90 db	42,1%
kisebb, mint -3 km/h sebességkülönbség	94 db	44,2%
± 10%-on belül és 40 km/h feletti	56 db	21%
±5%-on belül és 40 km/h feletti	21 db	10.3%

7. táblázat: Az elért eredmények. Kék szín jelzi a követő és detektáló algoritmusok teljesítményét, a zöld résszel jelzett sorok a ZoeDepth, narancsszínnel pedig az AdaBins mélységbecslő algoritmussal elért teljesítményt.

Az 7. táblázat tartalmazza az elért eredményeket. Az első három kékkel jelzett sor jelzi a követő és detektáló algoritmusok teljesítményét, a zöld résszel jelzett sorok a ZoeDepth, narancsszínnel pedig az AdaBins mélységbecslő algoritmussal elért teljesítményt.

Látható, hogy a fejlesztett rendszer az esetek 16%-ban nem tudta megmérni az elhaladó járműnek a sebességét. Ennek oka egyrészt az olyan járművek detektálása, amelyekről a detektor nem tudta eldönteni, hogy melyik osztályba tartozzon és képkockáról képkockára váltogatta az osztályát, ezzel pedig a követő algoritmust zavarta össze. Másik oka természetesen az, hogy vannak olyan járművek, amiket nem tud észlelni kellő pontossággal a detektor, ilyenek a buszok, teherautók, motorok és különleges járművek. Továbbá problémát okozott a rendszámok detektálása, aminek fő oka az, hogy a fejlesztett rendszer magyar rendszámokon tanult, a sebességmérő teszt adathalmazban pedig brazil rendszámok láthatóak, amik jelentősen más kinézettel bírnak. Ezen kívül az azonos szín miatt a jármű fényezésébe beleolvadó, árnyékolt, koszos rendszámtáblák vagy azoknak teljes hiánya is okozhatta a mérés elmulasztását. Megjegyzendő, hogy az esetek körülbelül 8%-ban az induktív hurokdetektorral sem sikerült a jármű sebességét megmérni, tehát az sem tekinthető hibátlannak.

Az is látható, hogy a ZoeDepth sokkal jobbnak bizonyult a feladatra, mint az AdaBins igaz közel ötször annyi paraméterrel rendelkezik (345M vs 78M). Futási sebessége mindkét algoritmusnak viszonylag lassú volt, a ZoeDepth kb. 350 ms-ot, az AdaBins kb. 150 ms-ot igényelt az inferenciához (RTX 3070-es GPU-val). A ZoeDepth-el az esetek 48%-ban sikerült az adathalmazhoz tartozó publikációban említett [-3;2] km/h-ás intervallumban maradni, ez a publikáció 96%-os pontosságához viszonyítva azonban jócskán alacsonyabb. Az esetek 78%-ban sikerült a mérőrendszernek ± 10 %-on belül maradnia. A fejlesztett rendszer azonban az esetek közel ötödében ± 10 %-os intervallumon kívül esik, tehát aránylag sok az egyáltalán nem megbízható mérés.

Ennek az egyik oka természetesen a mélységbecslő algoritmus nem kellő pontosságú predikciója, másik pedig a tény, hogy az én mérésem nem ugyanott történik, ahol a ground truth mérése. Ha a járművek közül kivesszük azokat, amik lassú sebességgel, nagy valószínűséggel gyorsulva haladtak, tehát amelyeknél a mérésem és a ground truth között eredendően nagy hiba van és csak azokat nézzük, ahol azok 40 km/h felett mozogtak, akkor javul a rendszerem pontossága. Ebben az esetben a mért sebességek 74%-a esik a ± 10 %-os intervallumba. Ezeknél az eseteknél feltételezhető a járművek konstans sebessége, tehát a rendszerem nagy valószínűséggel a jármű valós sebességét méri, nem pedig a múltban mért sebesség kerül összehasonlításra a ground truth-al.

A módszer nem hibátlan. Bizonyára segítene, ha a mélységbecslő algoritmusokat finom hangolnánk egy utat felülről figyelő RGB-D kamera kimenetével, ehhez azonban nem találtam forrást a szakirodalomban. A megoldás azonban csak két pont valóságban mért távolságát igényli a működéshez, ez nagy könnyítés a kalibrációt igénylő megoldásokkal szemben.

6 Összegzés

Munkám során az előre eltervezett sebességmérő kamera rendszert fejlesztettem ki. A megoldásom egy monokuláris mélységbecslés alapú sebességmérést és egy térbeli transzformációs hálóval megerősített teljesen konvolúciós rendszámtábla felismerést alkalmaz. Az irodalomkutatás során megtudtam, hogy a videóalapú sebességmérésnek léteznek modern megoldások, ezek viszont vagy kalibrációs vagy adat akvizíciós szempontból nehézkesen implementálhatóak. Erre a problémára egy olyan rendszert fejlesztettem ki, amely csak egy RGB kamerát és a kamera képén két pontnak a való életben precízen megmért távolságát igényli. A rendszert tehát elég egyszer felszerelni és egy lézeres távolságmérővel pontosan kimérni hozzá a referenciapontokat. A rendszámtábla leolvasás modern technikáiról továbbá megtudtam, hogy az ilyen modellek vagy szegmentációs hibákkal küzdenek, vagy jelentős komplexitással és mérettel bírnak. Ezt orvosolandó egy szegmentáció mentes és könnyű, pár millió paraméterből álló modellt fejlesztettem ki, amely könnyen tanítható és futtatható olcsóbb hardvereken is.

Az elkészült rendszámfelismerő modell kiváló eredményeket ért el. 10.000 darab valós, magyar rendszámtábla képből álló adathalmazon tesztelve egy- és többsoros táblákat is figyelembe véve 97.2%-os end-to-end pontosságot ért el. Ez az eredmény a szakirodalommal összevetve is kiemelkedő teljesítmény. A modell 20-30 milliszekundumos futási sebessége (RTX 3070 GPU-val) is figyelemreméltó, amely valós idejű alkalmazást is lehetővé tesz.

A fejlesztett sebességmérő modell a kiértékelt adathalmazon 40 km/h felett haladó járművek 74%-ának $\pm$10 %-os intervallumon belül meg tudta becsülni a sebességét. Azonban azt is figyelembe kell venni, hogy a rendszer a generált mélységtérkép konzisztenciája és pontossága miatt máshol méri a sebességet, mint az adathalmazban ground truth-ot adó érzékelő. Ez tehát azt jelenti, hogy nem kizárható, hogy a két mérés között a járművek sebességet változtattak, a módszerek tehát nem teljesen összehasonlíthatóak. A felhasznált modellek közül a ZoeDepth jelentősen felülmúlta a régebbi AdaBins megoldást, ezzel ígéretes megoldásnak tűnik bármely mélységalapú problémára. Azonban a sebességmérés területén, ahol precíz mérésekre van szükség, felhasználás specifikus adathalmazon való tanítás lenne szükséges a pontosabb működéshez. Sajnos ehhez a munkám során nem találtam elérhető forrást. A mélységbecslés továbbá jelentős időigényű, amihez drága hardverre van szükség, ha valós idejű futást szeretnénk elérni.

A jövőben tervezem a megvalósított rendszert továbbfejleszteni. Ez magában foglalja a sebességmérés terén a saját mélységtérképes, térfigyelő kamera pozícióból készített adathalmaz készítését, amivel a sebességmérés pontosságát reményeim szerint jelentősen javítani tudom. Érdekes fejlesztés lenne továbbá a modellek ritkítása, azaz a paraméterszámának csökkentése a teljesítmény csökkenésének egyidejű elkerülésével. A rendszámtábla felismerő modellen is tervezek még bővíteni, mégpedig más országok rendszámtábláinak felismeri lehetőségével. Ezt terveim szerint először egy régióosztályozó modellel közelíteném meg, amely kimenetének alapján később más országok tábláin tanított modelleket is használnék.

Irodalomjegyzék

[1] A. A. Bataneih, D. Kaur, M. Al-khassaweneh és E. Al-sharoa, „Automated CNN Architectural Design: A Simple and Efficient Methodology for Computer Vision Tasks," 2023. [Online]. Available: https://www.mdpi.com/2227-7390/11/5/1141. [Hozzáférés dátuma: 22 10 2023].

[2] D. F. Llorca, A. H. Martínez és I. G. Daza, „Vision-based vehicle speed estimation: A survey," *IET Intelligent Transport Systems,* %1. kötet15, %1. szám8, pp. 973-1091, 2021.

[3] D. C. Luvizon, B. T. Nassu és R. Minetto, „A Video-Based System for Vehicle Speed Measurement in Urban Roadways," *IEEE Transactions on Intelligent Transportation Systems,* %1. kötet18, %1. szám6, pp. 1393-1404, 2017 June.

[4] C. Ginzburg, A. Raphael és D. Weinshall, „A Cheap System for Vehicle Speed Detection," 2015. [Online]. Available: https://arxiv.org/pdf/1501.06751.pdf. [Hozzáférés dátuma: 26 10 2023].

[5] J. Sochor, R. Juránek és A. Herout, „Traffic surveillance camera calibration by 3D model bounding box alignment for accurate vehicle speed measurement," *Computer Vision and Image Understanding,* %1. kötet161, pp. 87-89, 2017.

[6] Z. Czapla, „Vehicle speed estimation with the use of gradient-based image conversion into binary form," *Signal Processing: Algorithms, Architectures, Arrangements, and Applications,* pp. 213-216, 2017.

[7] M. Dahl és S. Javadi, „Analytical Modeling for a Video-Based Vehicle Speed," *Computer Vision and Graphics - International Conference,* 2010.

[8] S. F. Bhat, I. Alhashim és P. Wonka, „AdaBins: Depth Estimation using Adaptive Bins," 2020. [Online]. Available: https://arxiv.org/pdf/2011.14141.pdf. [Hozzáférés dátuma: 24 10 2023].

[9] N. Silberman, D. Hoiem, P. Kohli és R. Fergus, „Indoor Segmentation and Support Inference from RGBD Images," [Online]. Available: https://cs.nyu.edu/~silberman/papers/indoor_seg_support.pdf. [Hozzáférés dátuma: 20 10 2023].

[10] Karsruhe Institute of Technology, „KITTI Vision Benchmark," 2012. [Online]. Available: https://www.cvlibs.net/datasets/kitti/. [Hozzáférés dátuma: 22 10 2023].

[11] S. F. Bhat, I. Alhashim és P. Wonka, „Localbins: Improving depth estimation by learning local distributions," *European Conference on Computer Vision,* pp. 480-496, 2022.

[12] Z. Li, X. Wang, X. Liu és J. Jiang, „Binsformer: Revisiting adaptive bins for monocular depth," 2022. [Online]. Available: https://arxiv.org/pdf/2204.00987.pdf. [Hozzáférés dátuma: 26 10 2023].

[13] A. Agarwal és C. Arora, „Attention Attention Everywhere: Monocular Depth Prediction with Skip Attention,” 2022. [Online]. Available: https://arxiv.org/pdf/2210.09071.pdf. [Hozzáférés dátuma: 25 10 2023].

[14] S. F. Bhat, R. Birkl, D. Wofk, P. Wonka és M. Müller, „ZoeDepth: Zero-shot Transfer by Combining Relative and Metric Depth,” 2023. [Online]. Available: https://arxiv.org/pdf/2302.12288.pdf. [Hozzáférés dátuma: 27 9 2023].

[15] F. Sultan, K. Khan, Y. A. Shah, M. Shahzad, U. Khan és Z. Mahmood, „Towards Automatic License Plate Recognition in Challenging Conditions,” *Applied Sciences,* %1. kötet6, 2023.

[16] J. Redmon, S. Divvala, R. Girshick és A. Farhadi, „You Only Look Once: Unified, Real-Time Object Detection,” 2015. [Online]. Available: https://arxiv.org/pdf/1506.02640.pdf. [Hozzáférés dátuma: 25 10 2023].

[17] J. Redmon és A. Farhadi, „YOLO9000: Better, Faster, Stronger,” [Online]. Available: https://arxiv.org/pdf/1612.08242.pdf. [Hozzáférés dátuma: 25 10 2023].

[18] J. Terven és D. Cordova-Esparza, „A Comprehensive Review of YOLO: From YOLOv1 and Beyond,” 2023. [Online]. Available: https://arxiv.org/pdf/2304.00501.pdf. [Hozzáférés dátuma: 28 10 2023].

[19] Ultralytics, „Ultralytics YOLOv5 GitHub page,” [Online]. Available: https://github.com/ultralytics/yolov5. [Hozzáférés dátuma: 22 09 2023].

[20] Ultralytics, „Ultralytcs YOLOv8 GitHub page,” 2023. [Online]. Available: https://github.com/ultralytics/ultralytics. [Hozzáférés dátuma: 27 9 2023].

[21] M. M. Khan, M. U. Ilyas, I. R. Khan, S. M. Alshomrani és S. Rahardja, „License Plate Recognition Methods Employing Neural Networks,” *IEEE Access,* %1. kötet11, pp. 73613-73646, 2023.

[22] S. Montazzolli és C. R. Jung, „Real-Time Brazilian License Plate Detection and Recognition Using Deep Convolutional Neural Networks,” in *SIBGRAPI,* Brazil, 2017.

[23] R. Laroca, L. A. Zanlorensi, G. R. Goncalves, E. Todt, W. R. Schwartz és D. Menotti, „An Efficient and Layout-Independent Automatic License Plate Recognition System Based on the YOLO detector,” [Online]. Available: https://arxiv.org/pdf/1909.01754.pdf. [Hozzáférés dátuma: 26 10 2023].

[24] M. Usama, H. Anwar, A. Anwar és S. Anwar, „Vehicle and License Plate Recognition with Novel Dataset for Toll Collection,” 2022. [Online]. Available: https://arxiv.org/pdf/2202.05631v2.pdf. [Hozzáférés dátuma: 27 10 2023].

[25] H. Li, P. Wang és C. Shen, „Toward End-to-End Car License Plate Detection and Recognition With Deep Neural Networks,” *IEEE Transactions on Intelligent Transportation Systems,* %1. kötet20, %1. szám3, pp. 1126-1136, 2019.

[26] M. Jaderberg, K. Simonyan, A. Zisserman és K. Kavukcuoglu, „Spatial Transformer Networks,” 2016. [Online]. Available: https://arxiv.org/pdf/1506.02025.pdf. [Hozzáférés dátuma: 22 08 2023].

[27] G. Hamrouni, „PyTorch Spatial Transformer Networks Tutorial,” [Online]. Available: https://pytorch.org/tutorials/intermediate/spatial_transformer_tutorial.html. [Hozzáférés dátuma: 24 09 2023].

[28] T. Lin et al., „Microsoft COCO: Common Objects in Context,” 2014. [Online]. Available: https://arxiv.org/pdf/1405.0312.pdf. [Hozzáférés dátuma: 26 10 2023].

[29] A. Bewley, Z. Ge, L. Ott, F. Ramos és B. Upcroft, „Simple Online and Realtime Tracking,” *IEEE International Conference on Image Processing (ICIP),* pp. 3464-3468, 2016.

[30] „platesmania.com,” 2023. [Online]. Available: https://platesmania.com/. [Hozzáférés dátuma: 20 10 2023].

[31] R. Laroca et al., „A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector,” 2018. [Online]. Available: https://arxiv.org/pdf/1802.09567.pdf. [Hozzáférés dátuma: 28 10 2023].