TANSZÉKVEZETŐ

DIPLOMATERVEZÉSI FELADAT

Rohrsetzer Róbert Richard
Mechatronikai mérnök hallgató részére

Mélytanulás alkalmazása fizika informált környezetben

A tengelysúlymérés kulcsfontosságú szerepet játszik a közúti infrastruktúra védelmében és a közlekedésbiztonság növelésében. A közúti járművek tengelyterhelése közvetlen hatással van az útpálya, a hidak és más közlekedési létesítmények állapotára. A túlterhelt járművek gyorsabb útkopást, repedéseket és akár szerkezeti károsodásokat is okozhatnak, amelyek jelentős karbantartási és javítási költségeket eredményeznek. A tengelysúlymérés segít azonosítani azokat a járműveket, amelyek a megengedett terhelési határokat túllépik, így lehetőséget ad a hatóságoknak a megfelelő intézkedések megtételére.

A hallgató diplomatervezési feladata során egy Bridge Weigh-in-Motion rendszer fejlesztésébe kapcsolódik be. A hallgató feladata a hídra telepített nyúlásmérő szenzorok jeleiből a hídon áthaladó járművek paramétereinek (pl. tengelytávolságok, sebesség, tengelysúly) kinyerése. Korábban Wu és társai [1] kidolgoztak egy BwimNet nevű megoldást, ami lényegében a fizika informált neurális hálózat (Physics Informed Neural Network, PINN) megközelítés alkalmazása volt ezen a területen, bár erről a cikk szerzői nem tettek említést a cikkben.

A hallgató feladata a területen fellelhető szakirodalom megismerése. Ezek után feladata, hogy az említett cikket reprodukálja, esetleg továbbfejlessze, az így kapott eredményeket kiértékelje. Végül a hallgató feladata a kidolgozott megoldás integrációja egy, már meglévő Bridge Weigh-in-Motion algoritmusokat tartalmazó szoftvercsomagba.

A hallgató feladatának a következőkre kell kiterjednie:

- Elemezze a témában fellelhető szakirodalmat.
- A rendszer bővítése legalább egy mélytanulás alapú algoritmussal.
- A kidolgozott mélytanulási algoritmus eredményeinek kiértékelése mind robosztusság, mind skálázhatóság szempontjából.
- A kidolgozott megoldás beépítése a témavezető által meghatározott szoftvercsomagba.

[1] Wu, Y., Deng, L., & He, W. (2020). BwimNet: A novel method for identifying moving vehicles utilizing a modified encoder-decoder architecture. Sensors, 20(24), 7170.

Tanszéki konzulens: Szinyéri Bence, doktorandusz
Budapest, 2025. március 13.

Dr. Charaf Hassan
egyetemi tanár
tanszékvezető

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék
H-1117 Budapest, Magyar tudósok krt 2. Q.B.207
Telefon: 463-2870. Fax: 463-2871
www.aut.bme.hu

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Rohrsetzer Róbert

MÉLYTANULÁS ALKALMAZÁSA FIZIKA INFORMÁLT KÖRNYEZETBEN

KONZULENS

Szinyéri Bence

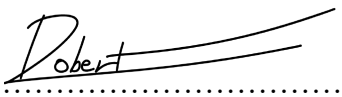
BUDAPEST, 2025

HALLGATÓI NYILATKOZAT

Alulírott **Rohrsetzer Róbert** szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzé tegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2025.12.11



.....

Rohrsetzer Róbert

Tartalomjegyzék

Összefoglaló

A hidak élettartamának és szerkezeti állapotának megbízható nyomon követéséhez elengedhetetlen az azokon áthaladó járművek dinamikus hatásainak ismerete. Minden jármű a hídon való áthaladása során terhelést és rezgéseket kelt, amelyek hosszú távon befolyásolják a szerkezet fáradását és élettartamát. A járművek által gerjesztett jeleket jellemzően a híd aljára szerelt nyúlásmérő bélyegekkel rögzítik, majd az így kapott adatokból meghatározhatók a járművek különböző paraméterei, például a tengelyterhelések és az össztömeg. Az ilyen rendszereket Bridge Weigh-In-Motion (BWIM) rendszereknek nevezik.

A diplomamunka célja a Wu és társai által felvázolt BwimNet, amely egy mélytanulás-alapú BWIM-algoritmus, reprodukálása és továbbfejlesztése, valamint a módszer tesztelése a komáromi híd szimulált és valós mérési adatai alapján. A kutatás során több neurális modult fejlesztettem, amelyeket a BME S1 szintetikus adathalmazzal tanítottam és a járművek különböző paramétereit becsülik: a keresztirányú pozícióbecslő a jármű keresztirányú pozícióját becsüli RMSE = 87,50mm hibával, a sebességbecslő a haladási sebességet becsüli 0,56 m/s átlagos hibával, míg a tengelyszámbecslő az áthaladó jármű tengelyeinek számát határozza meg 92,60% osztályozási pontossággal.

Ezen kívül a BwimNet reprodukálására érdekében elkészítettem egy jármű paraméter becslő modult, egy tengelysúly számító modult, egy jel rekonstruáló modult és egy olyan modult, amely az előző modulokat egybefűzi egy komplex rendszerré. A járműparaméter becslő modul a jármű sebességét, tengelytávolságát és azt az időpontot becsüli amikor az első tengely áthalad az első tengely fölött, viszont az eredmények átlagos hibájának nagysága miatt ez a modul még további fejlesztésre szorul. A tengelysúly számító és a jel rekonstruáló modul kielégítő eredményeket produkál viszont a modulok összekapcsolásakor a jármű paraméter becslő hibái tovább terjednek rajtuk ezzel növelve a rendszer hibáját.

Abstract

Reliable monitoring of the lifespan and structural condition of bridges requires a thorough understanding of the dynamic effects induced by passing vehicles. Each vehicle generates loads and vibrations as it travels across a bridge, which, over time, contribute to structural fatigue and influence the bridge's overall durability. The deformation signals produced by these vehicles are typically recorded using strain gauges mounted on the underside of the bridge. From these measurements, various vehicle parameters, such as axle loads and total weight, can be derived. Systems based on this principle are commonly referred to as Bridge Weigh-In-Motion (BWIM) systems.

The objective of this thesis is to reproduce and further develop the deep learning–based BWIM algorithm proposed by Wu et al. [1], and to evaluate its performance using both simulated and real measurement data collected from the Komárom bridge. During the research, I developed several neural modules trained on the synthetic BME S1 dataset, each designed to estimate different vehicle parameters. The lateral position estimator predicts the vehicle's transverse position with an RMSE of 87.50 mm; the speed estimator predicts the travel speed with an average error of 0.56 m/s; and the axle-count estimator determines the number of axles of the passing vehicle with an accuracy of 92,60%.

In addition, to enable the reproduction of BwimNet, I implemented a vehicle-parameter estimation module, an axle-weight computation module, a signalreconstruction module, and an integrative module that links the preceding components into a unified system. The vehicle-parameter estimation module predicts the vehicle's speed, axle spacing, and the moment at which the first axle passes over the first sensor; however, due to the relatively high average errors in its predictions, this module still requires further refinement. The axle-weight computation and signal-reconstruction modules yield satisfactory results, but when integrated, the errors originating from the parameter-estimation module propagate through the system, increasing overall error.

1 Bevezetés

1.1 Motiváció

A hidak állapotának folyamatos nyomon követése, valamint a szerkezeti fáradás előrejelzése napjainkban kiemelt jelentőséggel bír a közlekedési infrastruktúra biztonságának és fenntarthatóságának szempontjából. Ebben a folyamatban alapvető szerepet játszik a hídon áthaladó járműforgalom pontos mérése és elemzése, mivel a tengelyterhelés és az összsúly közvetlen hatással van a szerkezet élettartamára és terhelhetőségére. A megfelelő minőségű és részletességű terhelési adatok birtokában megbízható numerikus modellek hozhatók létre, amelyek képesek előre jelezni a híd állapotának változását, különös tekintettel a fáradásos károsodásokra. [1]

A Bridge Weigh-In-Motion (B-WIM) rendszerek éppen ezt a célt szolgálják: lehetővé teszik a járművek dinamikus paramétereinek, mint például a tengelysúly, a sebesség és a tengelytávolság, valós idejű mérését a forgalom akadályozása nélkül. E rendszerek egyik jelentős előnye, hogy nem invazívak, vagyis az érzékelők nem az úttestbe vannak beépítve, hanem a híd szerkezeti elemein, például bordákon vagy főtartókon, helyezkednek el. Ez a megközelítés nem igényli az útfelület megbontását vagy a közlekedés korlátozását, ami gyakorlati szempontból nagy rugalmasságot és költséghatékonyságot biztosít. [2]

1.2 Elért eredmények

A diplomamunkában a Komáromi Monostori hídra telepített, a híd aljára szerelt nyúlásmérő bélyegek által szolgáltatott szenzorjelekből álló adathalmazt használom fel. Ez az adathalmaz részletes információt nyújt a hídra ható terhelésekről, lehetővé téve a járművek áthaladásának és a híd válaszainak elemzését. A munka célja a Wu és munkatársai [3] által kifejlesztett BwimNet nevű módszer reprodukálása, valamint annak értékelése a Komáromi híd szimulált és valós mérési adatain.

A diplomamunka részeként három, konvolúciós neurális hálózaton alapuló modult készítettem, amelyek külön-külön a hídon áthaladó járművek keresztirányú pozíciójának, tengelyszámának és sebességének becslésére szolgálnak. Illetve megkíséreltem a BWIMnet algoritmus reprodukálását.

A keresztirányú pozícióbecslő modul a hídra telepített nyúlásmérő bélyegek jelének időbeli változásait használja bemenetként. A modul konvolúciós rétegei a szenzorjelek lokális mintázatait tanulják meg, amelyek a jármű áthaladása során keletkező nyúlásmintákhoz kapcsolódnak. A háló kimenete a jármű pozíciója a híd keresztirányában, a modell átlagos abszolút hibája 66,23 mm és teljesítménye RMSE alapján 87,50 mm. Ez a pontosság lehetővé teszi a járművek keresztirányú mozgásának pontos becslését.

A tengelyszám-becslő modul a szenzorok összegezett jeléből dolgozik, amely a jármű minden egyes tengelyének áthaladásakor keletkező deformációk összhatásait tükrözi. A modul célja, hogy a jármű tengelyeinek számát a lehető legnagyobb pontossággal megbecsülje. A modell a szintetikus adathalmazon 92,60% osztályozási pontosságot ért el, azonban a valós mérési adatokon ez az érték 43,96%-ra csökkent.

A sebességbecslő modul is a fő szenzorsor összegjelét használja bemenetként. A jármű sebessége a szenzorjelek időbeli változásának alapján becsülhető, mivel a gyorsabb áthaladás gyorsabb jelváltozást eredményez, míg a lassabb áthaladás lassúbb változást. A konvolúciós háló a jel dinamikáját tanulja meg, így képes pontos sebességbecslésre. A szintetikus adatokon a modell átlagos hibája 0,56 m/s, míg a valós mérési adatokon a hiba 1,57 m/s.

A BwimNet reprodukálására elkészítettem az egyes modulokat (tengelyszám becslő, jármű paraméter becslő, tengelysúly számító és jel rekonstruáló modul) amelyek külön-külön működnek. Létrehoztam a modulok integrált rendszerét, amelyben a komponensek közötti kommunikáció és adatáramlás sikeresen megvalósult. Ennek ellenére a kísérletek során a rendszer nem érte el a kívánt működési stabilitást.

1.3 A dolgozat felépítése

A 2. fejezet áttekintést nyújt a diplomamunka megértéséhez szükséges elméleti háttérről. A fejezet tárgyalja a mesterséges intelligencia szintjeit, a neurális hálók működési elveit, valamint a BWIM (Bridge Weigh-In-Motion) algoritmusok alapjait. Ezenkívül bemutatja a kutatás során felhasznált adathalmazt, valamint a szakirodalmi hivatkozásként szolgáló BwimNet algoritmust.

A 3. fejezet a saját fejlesztésű megoldások és a kísérleti eredmények részletes ismertetését tartalmazza. A fejezet a 3.1 alfejezettel indul, amely a járművek

keresztirányú pozíciójának becslésére szolgáló modult mutatja be. Ezt követi a 3.2 alfejezet, amely azokat az általános adatfeldolgozási és előkészítési elveket foglalja össze, amelyek a későbbi modulok többségére egységesen érvényesek. A 3.3 alfejezet a tengelyszámbecslő modult, míg a 3.4 alfejezet a sebességbecslő modult és azok eredményeit tárgyalja. Végezetül a 3.5 alfejezet a BwimNet rendszer rekonstrukciójára tett kísérletet, a teljes pipeline integrációját, valamint az ahhoz szükséges további komponenseket, a járműparaméter-becslő, a tengelysúlyszámító és a jelrekonstruáló modult, ismerteti.

A dolgozatot a 4. fejezet zárja, amely összegzi az elért eredményeket, értékeli a kifejlesztett modulok és az integrált rendszer teljesítményét, valamint kitér a munka során elért személyes szakmai fejlődésre.

2 Irodalomkutatás

Az elmúlt évtizedben a mesterséges intelligencia [4], a gépi tanulás [5] és a mélytanulás [6] kutatási területei rendkívüli figyelmet kaptak, köszönhetően a technológiai fejlődésnek és az óriási adatmennyiség robbanásszerű növekedésének, amelynek feldolgozásában ezek a módszerek új lehetőségeket kínálnak, és jelentősen hozzájárulnak a komplex problémák megértéséhez és megoldásához. A dolgozatban ismertetett BwimNet megközelítés szorosan kapcsolódik ezekhez a területekhez, különösen a mélytanulás keretében, ahol konvolúciós neurális hálózatokat (CNN) [7] és teljesen összekapcsolt rétegeket (FC) [8] alkalmazunk a komplex adatfeldolgozáshoz. Emellett részletesen bemutatom a BWIM-algoritmusokat is, mivel a diplomamunkám ezekre az algoritmusokra épül, így elengedhetetlen azok alapos megértése ahhoz, hogy a javasolt modell hatékonyságát értékelni tudjuk. A fejezet célja, hogy tömör, de átfogó áttekintést nyújtson az alapfogalmakról, beleértve a neurális hálózatok alapelveit, valamint a dolgozatban felhasznált módszerek tudományos hátteréről, ezzel megalapozva a későbbi fejezetek megértését. Ezáltal betekintést nyerve abba, hogy hogyan illeszkedik a BwimNet a modern AI-trendekbe, és milyen potenciális alkalmazási területei lehetnek a jövőben.

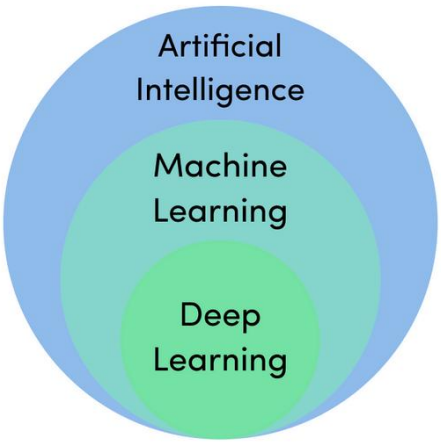
2.1 Mesterséges intelligencia, gépi tanulás és mélytanulás

Az elmúlt évtizedben a mesterséges intelligencia (AI), a gépi tanulás (ML) és a mélytanulás (DL) kutatási területei robbanásszerű fejlődésen mentek keresztül, elsősorban a számítási kapacitás növekedésének, a hatalmas adatmennyiségek elérhetőségének és az algoritmikus innovációknak köszönhetően. Az AI egy átfogó tudományág, amely arra törekszik, hogy gépeket ruházzon fel emberihez hasonló intelligenciával, lehetővé téve számukra a problémamegoldást, döntéshozatalt és tanulást. Alapvetően az AI rendszerek szabályalapú programozástól kezdve a komplex modellekig terjednek, ahol a gépek nem csupán utasításokat követnek, hanem adaptálódnak a változó környezetekhez.

A gépi tanulás (ML) az AI egyik kulcsfontosságú ága, amely arra összpontosít, hogy a rendszerek adatokból tanuljanak, anélkül, hogy explicit programozásra lenne szükség minden lépéshez [5]. Az ML működése alapvetően három fő megközelítésre épül. Felügyelt tanulásra, ahol a modell címkézett adatokkal tanítanak (pl. képek

osztályozása macskákra és kutyákra). Felügyelet nélküli tanulásra, amely mintákat fedez fel címkézetlen adatokban (pl. hasonló geometriai alakzatok, mint háromszögek és négyzetek csoportosítása). Valamint megerősítéses tanulásra, ahol a modell jutalmak és büntetések révén optimalizálja döntéseit. Ezek a módszerek matematikai alapokra épülnek, például lineáris regresszióra vagy döntési fára, és lehetővé teszik, hogy a modellek ismert adatokból mintázatokat sajátítsanak el, valamint önállóan döntsenek, anélkül, hogy külön programozni kellene őket. [9]

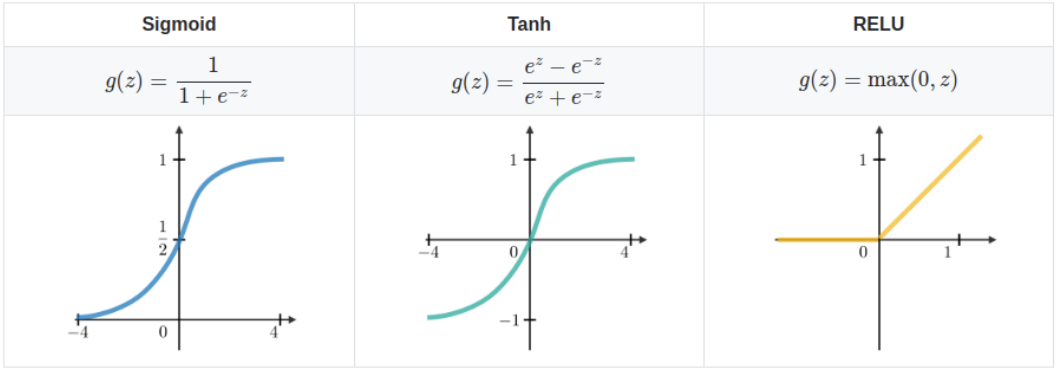
A mélytanulás (DL) pedig a gépi tanulás egy speciális formája, amely neurális hálózatokon alapul, modellezve az emberi agy struktúráját [6]. A DL modellek rétegekből állnak, ahol minden réteg egyre absztraktabb jellemzőket von ki az adatokból. Például egy konvolúciós neurális hálózat (CNN) képeket feldolgozva először éleket és formákat azonosít az alsó rétegekben, majd összetettebb objektumokat, mint arcokat vagy tárgyakat a felsőbb szinteken. A teljesen összekapcsolt rétegek (FC) pedig ezeket a kinyert jellemzőket integrálják a végső döntéshozatalhoz, például osztályozáshoz. A DL sikere nagyrészt a backpropagation algoritmusnak köszönhető, amely a hibákat visszaterjeszti a hálózaton keresztül, finomhangolva a súlyokat optimalizációs módszerekkel. Ez a megközelítés különösen hatékony nagy adatmennyiségeknél, és forradalmasította területeket, mint a képfelismerés, természetes nyelvfeldolgozás vagy akár orvosi diagnosztika. [9] [10]



1. ábra: Mesterséges intelligencia szintjei [11]

2.2 Neurális hálók működési alapjai

A neurális hálózatok működése az emberi agy idegsejtjeinek működésén alapul, ám leegyszerűsített matematikai modelleket alkalmaznak [12]. A mesterséges neuronok bemeneteket fogadnak, amelyeket súlyokkal szoroznak, majd egy aktivációs függvény segítségével határozzák meg, hogy milyen kimenetet adnak tovább a következő rétegnek. Ez az aktivációs függvény kulcsfontosságú, mivel lehetővé teszi, hogy a hálózat képes legyen összetett mintázatokat és összefüggéseket felismerni az adatokban, például képek, szövegek vagy hangok elemzése során. Ilyen aktivációs függvény lehet például a ReLU, Sigmoid és a Tanh, melyeket a 2. ábra szemléltet. [13]



2. ábra: ReLU, Sigmoid és Tanh függvények (nashtechglobal.com)

A neurális hálózatok több rétegből állnak, amelyek mindegyike különböző szerepet tölt be. A bemeneti réteg fogadja a nyers adatokat, például egy kép pixelértékeit vagy egy szöveg numerikus reprezentációját [10]. A rejtett rétegek, amelyek száma és mérete a hálózat komplexitásától függ, matematikai műveletekkel dolgozzák fel ezeket az adatokat, hogy egyre absztraktabb jellemzőket vonjanak ki belőlük. Végül a kimeneti réteg adja meg a végső eredményt, például egy osztályozási feladat esetén annak valószínűségét, hogy egy kép macskát vagy kutyát ábrázol. [12]

A hálózat tanulása során egy veszteségfüggvényt használnak, amely azt méri, hogy a kimenet mennyire tér el a helyes választól. A cél ennek a veszteségnek a minimalizálása, amit a gradienscsökkenés nevű optimalizálási módszerrel érnek el [13]. Ez a módszer lépésről lépésre módosítja a neuronok súlyait, hogy a hálózat egyre pontosabb előrejelzéseket adjon. A folyamat során a hálózat fokozatosan "tanul" az adatokból, és finomhangolja a belső paramétereit. [10]

A neurális hálózatok egyik legnagyobb előnye, hogy képesek önállóan azonosítani és kinyerni az adatok releváns jellemzőit, anélkül, hogy a fejlesztőnek manuálisan meg kellene határoznia, mire figyeljen a modell [9]. Például egy képen felismerhetik az éleket, formákat vagy színeket, egy szövegben pedig a kulcsszavakat vagy nyelvtani szerkezeteket. Ez a jellemzőkinyerési képesség különösen értékes összetett adatok, például képek, videók vagy hanganyagok elemzésekor, és lehetővé teszi a hálózatok alkalmazását számos területen, például spam e-mailek szűrésében, beszédfelismerésben, orvosi képalkotásban, szenzoradatok feldolgozásában (mint például valós idejű mérési adatok elemzése mérnöki rendszerekben), vagy akár önvezető autók környezetének valós idejű elemzésében [10].

A hatékony tanuláshoz azonban nagy mennyiségű adatra és jelentős számítási kapacitásra van szükség [10]. Emiatt gyakran speciális hardvereket, például grafikus processzorokat (GPU-kat) vagy dedikált AI-chipeket használnak a számítások felgyorsítására. Bár a neurális hálózatok rendkívül rugalmasak és alkalmazkodóképesek, egy gyakori kihívás a túltanulás, amikor a modell túlságosan az edzőadatokhoz igazodik, és új, ismeretlen adatokon rosszabbul teljesít. Ezt a problémát különböző technikákkal, például regularizációval, dropout módszerrel vagy adatbővítéssel lehet mérsékelni. [12]

A neurális hálózatok forradalmasították a gépi tanulást, és mára nélkülözhetetlen eszközzé váltak számos technológiai területen. Kiemelkedő eredményeket értek el olyan feladatokban, mint a képfelismerés (például arcfelismerő rendszerek), a természetes nyelvi feldolgozás (például fordítóprogramok vagy csevegőbotok), a beszédfelismerés (például virtuális asszisztensek), sőt még a stratégiai játékokban is, mint például a sakk vagy a Go [6]. A folyamatos kutatások és technológiai fejlesztések révén a neurális hálózatok egyre hatékonyabbá és sokoldalúbbá válnak, új lehetőségeket nyitva meg az AI alkalmazásaiban. [10]

2.2.1 Konvolúciós neurális hálók (CNN) és Teljesen összekapcsolt rétegek (FC)

A konvolúciós neurális hálózatok (CNN) a mélytanulás egyik kulcsfontosságú architektúrája, különösen hatékonyak képek, videók és idősoros szenzoradatok feldolgozásában [7]. Működésük a konvolúciós rétegekre épül, amelyek kis szűrőkkel azonosítják az adatok lokális mintázatait, például éleket, textúrákat képeknél, vagy

hullámformákat szenzoradatoknál, mint amilyenek a BWIM (Bridge Weigh-In-Motion) rendszerekben a hídterhelés elemzéséhez szükségesek. [10]

A CNN-ek egyik fő előnye a súlymegosztás, amely lehetővé teszi, hogy ugyanazokat a paramétereket alkalmazzuk az egész bemenetre [10]. Ennek köszönhetően jelentősen csökken a tanulandó súlyok száma, ami gyorsabb és hatékonyabb tanulást eredményez, valamint javítja a hálózat általánosító képességét. A pooling rétegek az adatok dimenzióját csökkentik, miközben megtartják a legfontosabb jellemzőket, így növelik a modell robusztusságát és mérséklik a túltanulás kockázatát. Ez a mechanizmus különösen előnyös zajos vagy nagy volumenű szenzoradatok feldolgozásánál.

A CNN-eket gyakran teljesen összekapcsolt (fully connected, FC) rétegek egészítik ki, amelyek a hálózat döntéshozatali szakaszát alkotják [13]. Az FC rétegekben minden bemeneti neuron összeköttetésben áll minden kimeneti neuronnal, így integrálják a konvolúciós rétegek által kinyert jellemzőket. Például egy BWIM rendszerben az FC rétegek a szenzoradatok mintázataiból becsülik a jármű tömegét vagy a híd terhelését. A konvolúciós rétegek az alacsony szintű mintázatokat azonosítják, míg az FC rétegek ezekből végső döntést hoznak, például osztályozást vagy regressziót végeznek [14].

A CNN-ek és FC rétegek kombinációja forradalmasította a gépi tanulást, és kiemelkedően teljesít képfelismerésben, beszédfelismerésben, valamint szenzoradatok elemzésében, például hidak terhelésmonitorozásában. A CNN-ek különösen alkalmasak nagy mennyiségű, zajos szenzoradat valós idejű feldolgozására, így a BWIM rendszerekben pontos és megbízható eredményeket nyújtanak.

2.3 BWIM algoritmusok

A BWIM (Bridge Weigh-in-Motion) rendszerek olyan mérési módszereket alkalmaznak, amelyek lehetővé teszik a hidakon áthaladó járművek súlyának, sebességének, tengelyeik számának, tengelyterhelésének és más paramétereinek meghatározását anélkül, hogy a forgalmat le kellene állítani [15]. A technológia lényege, hogy a híd alsó részén elhelyezett nyúlásmérő bélyegekkel regisztrálják a járművek által okozott szerkezeti változásokat. Ezekből az adatokból, megfelelő kalibrációval, ki lehet számítani a járművek jellemzőit. Az eredeti BWIM koncepciót egy 1979-es fejlesztés vezette be, ahol a tengelyeket útfelületre helyezett kapcsolószalagok vagy levegővel töltött csövek érzékelték, amelyek rögzítették az áthaladások időpontjait. Ez a korai megoldás azonban hajlamos volt a külső tényezők, például időjárás vagy mechanikai

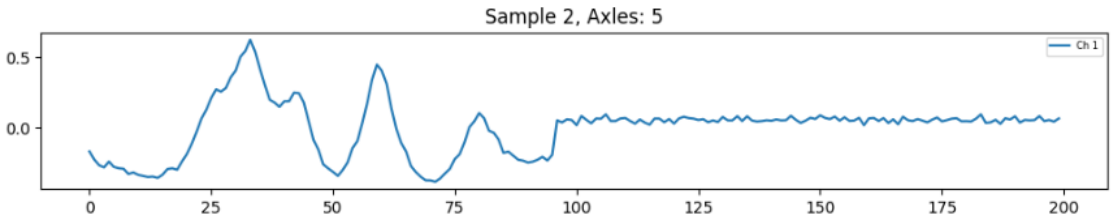
kopás miatti hibákra, és rendszeres ellenőrzést igényelt. Ezzel szemben, a Pavement WIM (Weigh-in-Motion) rendszerek esetében a szenzorok már az úttestbe ágyazva kerülnek telepítésre, minimalizálva az útfelületi interferenciát, azonban továbbra is beavatkozást igényelnek a burkolatba. A kortárs rendszerekben ezt felváltotta a híd alsó részén elhelyezett érzékelők alkalmazása, amit semmi az úton (Nothing On Road, NOR) BWIM néven ismernek. Ez a kialakítás nem igényel semmilyen elemet az úttesten, ami kedvezőbb a fenntartás, a rendszer védelme, a hosszú élettartam és a könnyű áthelyezhetőség szempontjából. A NOR BWIM gyorsan üzembe helyezhető, alkalmas rövid idejű vagy tartós megfigyelésre, és meglévő hidakra is utólag felszerelhető forgalmi zavarok nélkül. A BWIM rendszereket számos területen használják, például közúti járművek felügyeletére, túlterhelt szállítmányok azonosítására, hidak állapotának nyomon követésére, valamint forgalmi adatok gyűjtésére statisztikai célokra.

2.3.1 Ablakdetektálás

A BWIM (Bridge Weigh-in-Motion) rendszerekben a járművek detektálása és a nyúlásmérő szenzorok által szolgáltatott jelek feldolgozása alapvető lépés a pontos paraméterbecsléshez. Ilyen például a tengelysúlyhoz vagy sebesség [16]. Az ablakdetektálás célja a jármű áthaladásának pontos időbeli meghatározása a szenzorsor felett, amely lehetővé teszi, hogy a rendszer ne a teljes nyúlásmérő jeladatot, hanem csak a jármű tényleges áthaladásának idejére eső, releváns jeleket dolgozza fel. Ezáltal a jel eleje és vége, amely nem tartalmaz érdemi információt a jármű paramétereiről, hatékonyan kivágható (izolálható), optimalizálva a számítási időt és csökkentve a háttérzaj hatását. Ez az időbeli ablak azért szükséges, mivel a BWIM algoritmusok a szenzorok által mért deformációk időbeli mintázataiból becslik a jármű jellemzőit. A szenzorok jellemzően nyúlásmérő érzékelők formájában épülnek be a híd szerkezetébe, ahol a jármű súlya okozta hajlást mérik. A detektálás során küszöbérték-alapú módszereket alkalmaznak, amelyeknél, ha a jel meghalad egy előre meghatározott szintet, az jelzi a jármű belépését az ablakba, míg a jel visszaesése a kilépést mutatja. Gyakorlati alkalmazás során a rendszer pontosságát csökkenthetik a mérést terhelő zavarhatások, különösen a szerkezeti vibrációk, a háttérzaj, illetve a forgalmi interferenciák. Ilyenkor ideiglenesen kamerarendszereket integrálnak a BWIM-be, amelyek számítógépes látás (computer vision) technikákkal, például objektumkövetéssel, vizuálisan azonosítják a jármű pozícióját. Ezek a rendszerek főként a rendszer tanításához szolgálnak, ahol a kamerás adatokkal kalibrálják a deformációalapú modelleket, de hosszú távon

elhagyhatók a költségek miatt. A kombinált megközelítés javítja a jel-zaj arányt (signal-to-noise ratio) és csökkenti a hamis pozitívumokat. [17]

A nyúlásmérő szenzor által szolgáltatott jelek gyakran egymásra torlódó vagy átfedő csúcsot (peaks) tartalmaznak, amelyek pontatlan járműintervallumokat jeleznek, megnehezítve a pontos azonosítást [18]. Egy ilyen esetet mutat be a 3. ábra, ahol megfigyelhető, hogy az 5 tengelyes jármű első három tengelyhez tartozó csúcsok kevésbé elkülöníthetők, mint az utolsó két tengelyé.



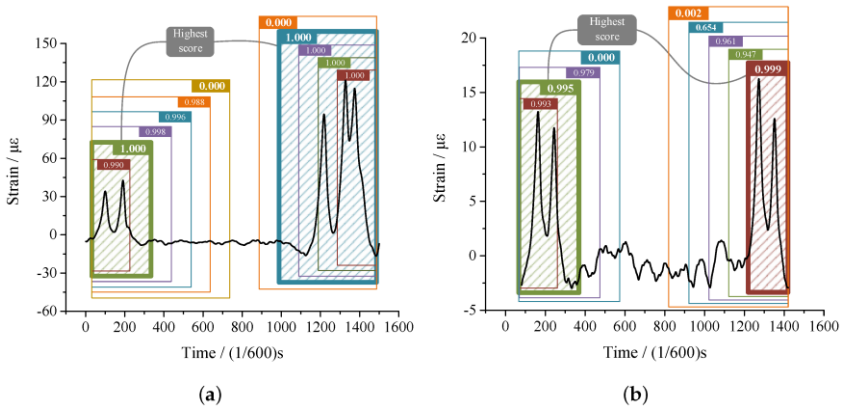
3. ábra: Nyúlásmérő szenzor által szolgáltatott jel egy 5 tengelyes jármű esetén

Erre szolgál az NMS (Non-Maximum Suppression, NMS), egy szűrőalgoritmus, amely a jelcsúcsok közül a legerősebbet tartja meg, miközben elnyomja a gyengébb, átfedő jelölteket. Az NMS működése a következő lépésekből áll: először a nyúlásmérő szenzor által szolgáltatott jelből potenciális intervallumokat azonosítanak csúcsdetektálással, ahol minden egyes csúcs egy lehetséges járműáthaladást jelöl [19]. Ezután minden intervallum-pár között kiszámítják az átfedést az IoU (Intersection over Union) metrika alapján, amely az átfedés területét osztja a két intervallum uniójának területével, ha ez az érték meghalad egy küszöböt, például ötven százalékot, akkor átfedésről beszélünk. Végül a legmagasabb konfidencia értékkel rendelkező detekciót megtartják, míg a vele átfedésben lévő, gyengébb jelölteket törlik. Ezt a folyamatot ismételgetik, amíg minden redundancia eltűnik. Ezt a folyamatot szemlélteti a 4. ábra. Ez a módszer a biztosítja, hogy egyetlen járműintervallumot ne osszanak fel több részre, így javítva a tengelysúlybecslés pontosságát. [20]

A hagyományos NMS korlátai, például a fix küszöbök zajra való érzékenysége okán, neurális hálózatokat integrálnak a rendszerbe, ami intelligensebbé teszi a folyamatot [16]. A működés elve az, hogy a neurális hálózat, például egy CNN, elemzi a nyúlásmérő szenzor jeleit, és több potenciális intervallumot javasol konfidencia értékekkel. Ezeket a javaslatokat aztán az NMS szűri, ahol a hálózat által adott értékek alapján választják ki a legjobbakat, elnyomva a redundánsakat [18]. Így a rendszer tanul a korábbi adatokból, javítva a pontosságot bonyolult helyzetekben, például több jármű

egyidejű áthaladásakor, és kevesebb manuális beállítást igényel. Például hibrid rendszerekben ez a kombináció a nyúlásmérő szenzor által szolgáltatott jeleket video adatokkal ötvözi, ahol a hálózat a híd befolyásfelületét (influence surface) is figyelembe veszi. [17] [18]

Összefoglalva, az ablakdetektálás az időbeli mintázatokra támaszkodik, az NMS szűri a redundanciát, míg a neurális integráció intelligens kiválasztást biztosít, együttesen növelve a BWIM rendszerek megbízhatóságát.



4. ábra: Járművek ablakainak kiválasztása [16]

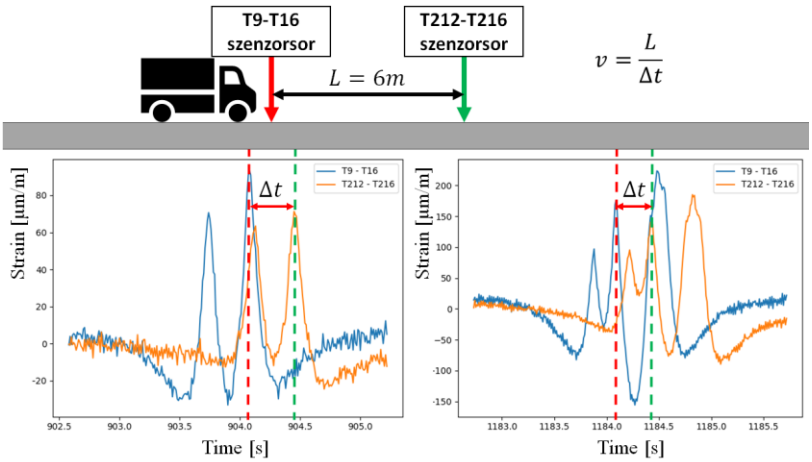
2.3.2 Sebességbecslés

A BWIM (Bridge Weigh-in-Motion) rendszerekben a sebességbecslés létfontosságú elem, mivel elengedhetetlen a tengelysúlyok pontos kiszámításához [21]. A sebesség ugyanis befolyásolja a nyúlásmérő szenzor által szolgáltatott jelek időbeli eloszlását. Ennek hiányában a jármű áthaladásának idejére vonatkozó pontatlanságok összeadódhatnak, jelentősen rontva a súlybecslés pontosságát. A becslést nyúlásmérő szenzorokkal felszerelt rendszerekben végzik, ahol a szenzorok a híd szerkezetében mérik a jármű áthaladásakor keletkező hajlást. A folyamat alapja két szenzorsor használata, amelyek általában néhány méter távolságra helyezkednek el egymástól a hídon, lehetővé téve a jármű áthaladásának időtartamának mérését ezen a szakaszon. [22]

A sebesség kiszámítása egyszerűen a távolság és az eltelt idő hányadosaként történik. a hat méteres szakaszt alapul véve a jármű áthaladásának idejét mérik, majd ebből kapják a sebességet méter per másodpercben kifejezve. [23]. Ez a módszer azért hatékony, mert a két szenzorsor jelei rendkívül hasonlók, hiszen ugyanaz a jármű okozta deformációt rögzítik, egy meghatározott hosszirányú távolságon belül. A hasonlóságot kihasználva a jelek összehasonlításával azonosítják a lokális szélsőértékek

időkoordinátáit, amelyek a jármű tengelyeinek áthaladását jelzik. Például a csúcsok vagy mélypontok pozícióit hasonlítják össze a két jel között, és ebből számolják ki az eltelt időt minden egyes tengelyre vonatkozóan. Ez a lépés látható az 5.ábrán, ahol a két, nyúlásmérő szenzor által szolgáltatott, jel görbéjét egymásra helyezik, kiemelve az időeltolódásokat. A gyakorlatban rögzítik az időeltolódásokat minden szenzoron, majd átlagot képeznek ezekből, hogy minimalizálják a zajból eredő hibákat és növeljék a pontosságot [24]

A valós körülmények között, például több jármű egyidejű áthaladásakor, a sebességbecslés kihívásai növekednek, de a korrelációs módszerek robosztusak maradnak [18]. Például a jelek keresztkorrelációjával automatikusan megkeresik a maximális hasonlóságot a két jel között, ami közvetlenül adja az időeltolódást, így a sebességet. Ez a technika különösen előnyös, mert nem igényel külső sebességmérő eszközöket, például radarokat, és teljes mértékben a meglévő nyúlásmérő szenzorokra támaszkodik. Modern kiegészítésekben neurális hálózatokat is bevonnak a jelek elemzésébe, amelyek tanulják a sebességmintázatokat egyetlen szenzor alapján is, javítva a becslés megbízhatóságát bonyolult forgalmi helyzetekben. Összességében a sebességbecslés biztosítja a BWIM rendszerek időbeli felbontását, alapot adva a súlyparaméterek pontos kalkulációjához, miközben minimalizálja a hardveres költségeket. [25]



5. ábra: Jármű áthaladásakor keletkezett összegjel az elsődleges (T9-T16) és másodlagos (T212-216) szenzorsoron és abból való sebesség számolás [19]

2.3.3 Tengelydetektálás

A BWIM (Bridge Weigh-in-Motion) rendszerekben a tengelydetektálás központi szerepet tölt be, hiszen ennek révén határozható meg az áthaladó jármű tengelyeinek száma, valamint rögzíthető pontosan az a pillanat, amikor ezek a tengelyek elhaladnak a fő szenzorsor fölött [26]. Ez az adat elengedhetetlen a tengelysúlyok kiszámításához, mivel a nyúlásmérő szenzor által szolgáltatott jelek elemzésekor kizárólag a tengelyek időbeli pozíciói alapján lehet a terhelést megfelelően kiosztani a rendszerben.

Diplomamunkám a komáromi hídra telepített szenzorokra támaszkodik, amely egy ortotróp acélpályás híd, így a detektálási technikák ehhez a szerkezethez vannak igazítva. A híd szerkezeti típusa alapvetően befolyásolja a tengelydetektálás sikerességét, hiszen a hidak eltérően reagálnak a járművek dinamikus terhelésére. Az ortotróp acélpályás hidaknál a pálya merevítése ortogonális irányokban történik, ami kiváló teherelosztást tesz lehetővé, ám a rendszer érzékenyebbé válik a helyi deformációkra. Itt élesebb, rövidebb ideig tartó jeleket mérünk. Ez megkönnyíti a gyors detektálást, de növelheti a zajhatást. Ezzel szemben a vasbeton hidak monolitikus pályaszerkezete a nagyobb tömeg és tehetetlenség miatt lassabban reagál a jármű terhelésére. Ezért a mért jelek le- és felfutása nem annyira gyors, az idő-megnyúlás görbe pedig egyenletesebb és időben elnyújtottabb [19].

A tengelydetektálás kiindulópontja a bemeneti jel, azaz a nyúlásmérő szenzor által szolgáltatott jel, elemzése, ahol gyakran a jel második deriváltját alkalmazzák [21]. A detektálás alapvető kihívása az egymáshoz közeli tengelyek (pl. tandem vagy tridem konfigurációk) által okozott jelalak, mivel áthaladásuk nem feltétlenül jár együtt lokális maximum megjelenésével a mért jelben. Ilyen esetekben a nyúlásjel maximumai összeolvadnak, megnehezítve az egyedi tengelyek elkülönítését. A második derivált vizsgálatának előnye, hogy általa a nyúlásjel inflexiós pontjai sokkal stabilabban azonosíthatók. Ezen inflexiós pontok pozíciója megbízhatóan meghatározható még a tandem-tridem által okozott nyúlások esetén is, lehetővé téve a tengelyidőpontok precíz rögzítését. Mindazonáltal, a második derivált módszer alkalmazása korlátozott lehet az alacsony jel-zaj viszony (Signal-to-Noise Ratio, SNR) mellett, mivel nagyfokú zajérzékenysége hibás detektálást okozhat. Előnye a számítási gyorsaság és egyszerűség, bár a zajra való érzékenység miatt jellemzően szűrőkkel, például simító algoritmusokkal párosítják. [27]

A Virtuális Tengely Módszer (Virtual Axle Method, VAM) a Moses-módszer alapelveire épül (a hatásvonal fogalmának használatára), de egy kritikus ponton túl is lép rajta. A Moses-módszer a bemeneti adatoknak (tengelypozícióknak) a pontos ismeretét feltételezi, míg a VAM a tengelykonfigurációt becsüli (optimalizálja) a mért jel alapján, így a detektálási funkciót integrálja a súlybecslésbe. [28]. A módszer lényege, hogy a nyúlásmérő szenzor által szolgáltatott jelet egy matematikai modellel közelíti meg. A modell illesztéséhez a rendszer előre meghatározott, valid tengelykombinációk halmazából próbál ki lehetséges járműkonfigurációkat a mért jelre illesztve [28]. A modell illesztése iteratív módon történik, optimalizálva a tengelyek számát és pozícióit, amíg a modell által generált jel a legkisebb maradék hibával illeszkedik a valós nyúlásjelhez, miközben figyelembe veszi a híd hatásvonalát (influence line). A hagyományos módszerekkel összehasonlítva magasabb pontosság érhető el és jobb teljesítményt nyújthat összetett forgalmi körülmények között, például több jármű együttes áthaladásakor. Ez a megközelítés viszonylag állandó sebességű áthaladásokra ideális.

Hatékony kiegészítő eszköz a folytonos wavelet transzformáció (CWT) [27]. Ez az eljárás a jelet hullámfüggvényekkel (waveletekkel) elemzi, amelyek a jellokalizációt és a frekvenciaelemzést egyidejűleg végzik. A CWT különböző skálákon, azaz eltérő időbeli felbontásokon, szűr mintázatokat. A kisebb skálák a részletes, rövid időtartamú jellemzőket, mint például az egyedi tengelyek áthaladását jelző csúcsokat, emelik ki, míg a nagyobb skálák az általános szerkezeti válasz trendjét tárják fel. Ez a folyamat biztosítja, hogy a tengelyek áthaladásához tartozó frekvenciák automatikusan detektálhatók legyenek, még széles frekvenciatartományú jelek esetén is. A CWT, a hagyományos frekvenciaelemzési eljárásokhoz (pl. Fourier-transzformáció) képest, relatíve jobb zajelnyomó képességgel bír a jeltisztítás során, azonban hátránya, hogy a megfelelő anyawavelet (mother wavelet) kiválasztása kritikus a pontosság szempontjából, és a folytonos transzformáció számításigényes lehet. [21]

Összefoglalva, a tengelydetektálás adja a BWIM rendszerek vázát, ahol a második derivált egyszerűséget és számítási gyorsaságot, a virtuális tengely módszer önállóságot (a külső detektoroktól való függetlenséget), a wavelet transzformáció pedig részletességet és zajszűrési képességet biztosít. Ezek a technikák együttesen teszik elérhetővé a megbízható súlybecslést, miközben csökkentik a külső berendezések függőségét. A jövőbeni továbbfejlesztésekben neurális hálózatokkal még inkább növelhető a valós idejű hatékonyság. [28] [21]

2.3.4 Tengelysúlybecslés

A BWIM (Bridge Weigh-in-Motion) rendszerekben a tengelysúlybecslés a folyamat utolsó eleme, ahol az előző lépések, az ablakdetektálásból származó időbeli ablakok, a sebességbecslésből nyert sebességértékek és a tengelydetektálásból meghatározott tengelypozíciók, együttesen szolgálnak alapul a jármű tengelyeinek súlyának kiszámításához. Ez a becslés nélkülözhetetlen a teljes járműsúly meghatározásához, valamint a túlterhelés detektálásához, mivel a nyúlásmérő szenzor által szolgáltatott jelek amplitúdója közvetlenül összefügg a tengelyek terhelésével. A hagyományos matematikai módszerek mellett ma már a gépi tanulás (machine learning) dominál, mivel jobban kezeli a hiányos adatokat és tanul a valós adatokból anélkül, hogy merev fizikai modelleket feltételezne. Ez a fejezet áttekinti a klasszikus Moses-módszert és annak továbbfejlesztéseit [29], valamint a modern neurális hálózatokon (neural networks) alapuló technikákat, kiemelve a NOR BWIM (Nothing-on-Road BWIM) előnyeit és a többsávos esetek komplexitását. [30]

A hídra szerelt súlymérés rendszerek egyik első módszere a Moses-módszer, amelyet az 1970-es évek végén fejlesztettek ki, és ma is a matematikai alapú becslések kiindulópontja [30]. Ez a technika a híd hatásvonalát (influence line) használja fel. A hatásvonal egy matematikai függvény, amely egy híd meghatározott keresztmetszetében (vagy egy rögzített pontjában) ébredő belső erő (pl. nyomaték, nyíróerő) vagy a támaszerő változását írja le egy egységnyi terhelés mentén, amikor az végighalad a híd teljes hosszán. A módszer lényege egy lineáris egyenletrendszer felállítása: a mért, nyúlásmérő szenzor által szolgáltatott jeleket egyenletbe foglalják az ismert tengelypozíciókkal és sebességgel, majd az ismeretlen tengelysúlyokra megoldják az egyenletrendszert. Például, ha három tengelyt detektáltunk, a rendszer három egyenletet generál, és pszeudoinverz segítségével keresi azokat a súlyokat, amelyek a legjobban illeszkednek a mért adatokra.

Előnye az egyszerűség és a fizikai alapokba gyökerező megbízhatóság. Minimális számítási kapacitást igényel, és jól működik ideális körülmények között, például egyenletes sebességnél és tiszta jeleknél [31]. Hátránya azonban a bemeneti paraméterektől való erős függés: a módszer feltételezi a tengelyek trajektóriájának teljes ismeretét (beleértve a tengelydetektálás pontosságát és az áthaladási sebesség precíz értékét). Az ebből származó hibák jelentős torzítást okozhatnak a súlybecslésben. Továbbá, a lineáris egyenletrendszer érzékeny a mérési zajokra és az útfelület

minőségéből adódó dinamikus hatásokra. Ez a módszer alapverziója jól kezeli az éles jeleket, de valós forgalomban finomhangolást igényel a pontosság növeléséhez.

A probabilisztikus megközelítések további mélységet adnak, ezek nem egyetlen determinisztikus súlyértéket adnak, hanem valószínűségi eloszlást, figyelembe véve az adatok bizonytalanságát (pl. szenzorhibák vagy sebességeltérések) [32]. Például egy Bayes-i keretrendszerben az előzetes információkat (pl. tipikus járműsúlyokat) kombinálják a mért adatokkal, így kapunk egy valószínűségi sűrűségfüggvényt minden tengelyre. Ez különösen hasznos változó forgalomban, mert nem csak a "legvalószínűbb" súlyt adja, hanem a bizonytalanság mértékét is, segítve a kockázatbecslést. Ezek a variációk, mint a Moses-módszer kiterjesztései több sávos hidakra, javítják a pontosságot akár 10-15 százalékkal, de még mindig érzékenyek a tengelydetektálás és sebességbecslés hibáira. [19]

A matematikai módszerek korlátai miatt a gépi tanulás, különösen a neurális hálózatok, egyre inkább kiszorítja őket a hídra szerelt súlymérés rendszerekből, mivel kiválóan kezeli a zajos, hiányos adatokat és tanul a valós forgatókönyvekből anélkül, hogy merev fizikai modelleket feltételezne [33]. Ezek a technikák a korábbi lépések kimeneteit (pl. sebesség, tengelyidők) bemenetként használják, és a teljes jelet elemezve becslik a súlyokat, gyakran felügyelt tanulással, ahol kalibrációs adatokkal tanítják a modellt. A neurális hálózatok alkalmazása itt különösen hatékony, mivel képesek a zajjal és dinamikus torzításokkal terhelt jelekben is feltárni nemlineáris összefüggéseket, ezáltal meghaladva a hagyományos lineáris egyenletrendszerek korlátait. A tanítási folyamatban tipikusan MSE (mean squared error) veszteségfüggvényt alkalmaznak, amely minimalizálja a becsült és valós súlyok közötti különbséget, miközben dropout rétegekkel előzik meg a túltanulást (overfitting), ami gyakori nagy dimenziójú jeladatoknál. [19]

A NOR BWIM (Nothing-on-Road BWIM) koncepció előnyei itt különösen hangsúlyosak, hiszen a rendszer nem igényel útfelületi szenzorokat, hanem kizárólag a híd szerkezeti válaszreakcióira támaszkodik, ami jelentősen csökkenti a telepítési és karbantartási költségeket [33]. Mivel ennél a kialakításnál nem állnak rendelkezésre útfelületi szenzorok a jármű pozíciójának meghatározására, a rendszernek a híd válaszából kell kikövetkeztetnie a terhelés eloszlását. A neurális hálózatok nagy előnye, hogy képesek megtanulni ezeket a komplex összefüggéseket, így hatékonyan kezelik azt a bizonytalanságot is, amit a járművek sáv közepétől való eltérése okoz [33].

Amennyiben azonban az adott részfeladat nem igényli a keresztirányú pozíció-információ explicit megőrzését, a rendszer robusztussága jelentősen növelhető a szenzorjelek megfelelő aggregálásával. Fizikai szempontból ugyanis, míg a hatásfelület (Influence Surface) értéke függ a terhelés keresztirányú pozíciójától, addig a teljes keresztmetszetet lefedő szenzorsor jeleinek összege kiegyenlíti ezeket az eltéréseket. Ennek eredményeként az aggregált válaszjel gyakorlatilag függetlenné válik a jármű sávon belüli pozíciójától, így a probléma visszavezethető az egyszerűbb, hatásvonal (Influence Line) alapú megközelítésre, tehermentesítve ezzel a neurális hálózatot a felesleges komplexitás alól.

A tengelysúlybecslés hatékonysága függ a híd sávszámától is. Egysávos hidaknál a jármű pozíciója fix, így a Moses-módszer egyszerűsített változata elegendő. Ezzel szemben, több sávos hidaknál a helyzetet a transzverzális eloszlás bonyolítja, mivel két jármű egyidejűlegvaló áthaladása és a járművek sávon belüli keresztirányú pozíciója közvetlenül megváltoztatja a terheléseloszlást a híd szerkezetén. [33]

Az integrált megközelítések, mint a hibrid modellek, ötvözik a Moses-módszert gépi tanulással. A neurális réteg finomhangolja az egyenletrendszerhez szükséges bemeneti paramétereket [19]. Ez különösen előnyös több jármű esetén, ahol a probabilisztikus elemekkel együtt kezelik az interferenciát. [32]

Összefoglalva, a tengelysúlybecslés a hídra szerelt súlymérés lényege, ahol a Moses-módszer történelmi alapot ad, de a probabilisztikus és neurális technikák biztosítják a modern hatékonyságot. Ezek a módszerek kihasználják az előző lépések eredményeit, növelve a pontosságot valós körülmények között, és nyitva hagyják az utat további hibrid fejlesztésekhez.

2.4 Fizikainformált neurális hálózatok

A fizikainformált neurális hálózatok (angolul Physics-Informed Neural Networks, röviden PINN-ek) olyan mesterséges neurális hálózatok, amelyeket kifejezetten fizikai problémák megoldására terveztek [34]. Ezek a problémák gyakran azért jelentik a legnagyobb kihívást, mert csak korlátozott mennyiségű, zajos mérési adat áll rendelkezésre, ezért a hagyományos gépi tanulási módszerek nem mindig adnak megbízható eredményt. A fizikainformált megközelítés lényege, hogy a modell nem kizárólag az adatokra támaszkodik, hanem beépíti a fizikai világot leíró törvényeket és

szabályokat is. Ilyen törvények lehetnek például a differenciálegyenletek, megmaradási törvények, peremfeltételek vagy más fizikai konzervációs elvek.

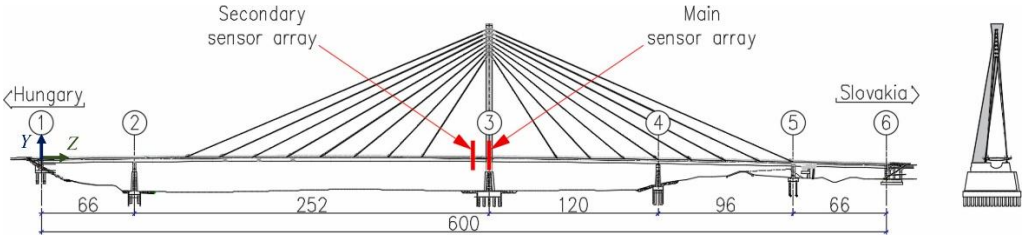
Ezek a hálózatok abban térnek el a hagyományos neurális hálózatoktól, hogy a tanulási folyamat során nemcsak az adatokhoz való illeszkedést próbálják optimalizálni, hanem azt is biztosítják, hogy a hálózat kimenete fizikailag értelmezhető és konzisztens legyen [35]. Ezt úgy érik el, hogy a veszteségfüggvénybe beépítenek egy úgynevezett fizikai konzisztencia tagot, amely például egy parciális differenciálegyenlet megsértésének mértékét is tartalmazza. Így a tanulás során a hálózat egyszerre tanul az adatokból és a fizika szabályaiból, ami különösen hatékonnyá teszi olyan rendszerek modellezésére, ahol kevés adat áll rendelkezésre, de a mögöttes fizikai törvényszerűségek ismertek.

A PINN-ek alkalmazása széles körű és rendkívül aktív kutatási téma. Használják őket többek között folyadékdinamikai problémák, hővezetési egyenletek, hullámegyenletek, strukturális/szerkezeti alkalmazások és egyéb komplex fizikai rendszerek modellezésére is [34]. Előnyük, hogy mivel a fizikai ismereteket közvetlenül beépítik a modellbe, az extrapolációs képességük is jobb lehet, mint a hagyományos, kizárólag adat vezérelt hálózatoké. Ez különösen fontos olyan szimulációkban, ahol az adatok előállítása és címkézése erőforrás igényes, időigényes vagy egyszerűen nem lehetséges.

2.5 Az adathalmaz bemutatása

Ebben a részben bemutatom a diplomaterv során felhasznált adathalmazt, amely a BME-Simulated-I nevű szimulált adathalmaz. Az adatgyűjtés alapját a Komáromi híd aljára felszerelt 26 darab nyúlásmérő bélyeg szolgáltatja, amelyek külön-külön csv fájlokban tárolják az egyes szenzorok által mért időfüggő jelváltozásokat.

A szenzorok elhelyezése két keresztmetszetben történt, és két csoportra bonthatók, egy primer szenzorsorra (16 db szenzor) és egy szekunder szenzorsorra (10 db szenzor). Ezek elhelyezkedését és pontos pozícióját az 6. Ábra szemlélteti. A nyúlásmérő bélyegek célja a híd szerkezetében keletkező feszültségváltozások regisztrálása a járművek áthaladása során, amely adatok alapján következtetni lehet többek között a járművek tengelyterhelésére, sebességére és keresztirányú pozíciójára.



6. Ábra: Szenzorsor pozíciója a hídon (Forrás: [19])

Az adathalmaz különféle forgalmi szituációkat is lefed, így alkalmas különböző tengelyszámú (2–9 tengelyes) járművek vizsgálatára. A járművek haladási módját illetően az adatok megkülönböztetik a Tandem és nem-Tandem tengelyelrendezéseket. A tandem tengely (vagy tengelycsoport) két vagy több, egymáshoz közel elhelyezkedő tengelyt jelöl (pl. <1,8m), amelyek a WIM rendszerekben összefolyó deformációs jelet okozhatnak, míg a nem-tandem tengely önálló, jól elkülönülő jelet eredményez. Emellett az adatok megkülönböztetik azt is, hogy a járművek konvojban vagy egymástól függetlenül közlekednek-e. Továbbá az adatok arra is kiterjednek, hogy a járművek egy vagy két forgalmi sávon közlekednek-e, így lehetőség nyílik a sávhasználat elemzésére is. [19]

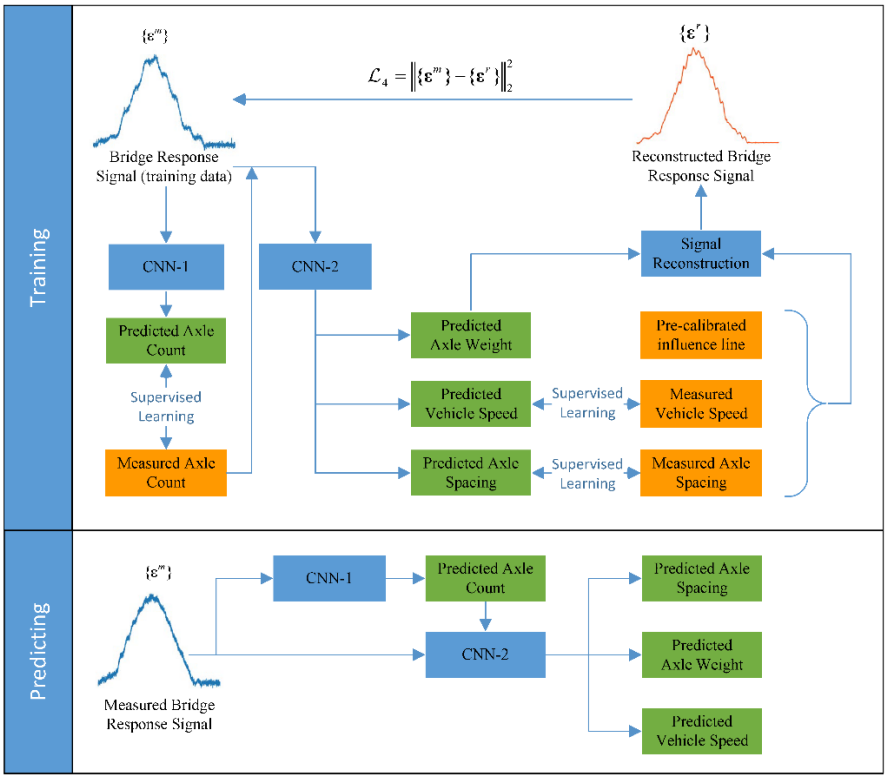
2.6 Cikk által kidolgozott BwimNet bemutatása

A Bridge Weigh-In-Motion (BWIM) egy innovatív módszer, amely a híd dinamikus alakváltozásából származó jelek elemzésével képes azonosítani a hídon áthaladó járművek által generált terheléseket. Ez a technológia lehetővé teszi a járműterhelések pontos becslését, ami kulcsfontosságú a hidak állapotának monitorozásában és élettartamuk meghosszabbításában. A pontos terhelésbecslés érdekében elengedhetetlen a járművek tengelykonfigurációjának és sebességének előzetes ismerete, amelyet hagyományosan kiegészítő szenzorok telepítésével (pl. kamerák) érnek el. Az ilyen szenzorok alkalmazása azonban számos gyakorlati kihívással jár, például a magas telepítési és karbantartási költségekkel, valamint a környezeti hatásokkal szembeni érzékenységgel.

Bár a forgalmi terhelések becslésére neurális hálózatok is alkalmazhatók, a hídon áthaladó járművek súlyadatainak megszerzése továbbra is jelentős nehézséget okoz, mivel ezek az adatok gyakran nem állnak rendelkezésre valós idejű mérések formájában. E korlátok áthidalására a cikk egy fejlett jelkódoló-dekódoló (encoder-decoder) architektúrát javasol, amely konvolúciós neurális hálózatok (CNN) integrálásával képes

a híd dinamikus válaszjelei alapján a járművek kulcsfontosságú paramétereinek, például sebesség, tengelytávolság és tengelyterhelés, pontos becslésére, külső súlymérési adatok nélkül.

A bemutatott módszer két konvolúciós neurális hálózatra (CNN) épül, amelyek egymással szorosan együttműködve biztosítják a kívánt eredményeket. Az első neurális háló (CNN1) feladata a hídon áthaladó jármű tengelyszámának meghatározása a híd dinamikus válaszjelei alapján. A második neurális háló (CNN2) az első háló által szolgáltatott eredményeket, valamint a szenzorok által gyűjtött adatokat felhasználva becsüli a további járműparamétereket, így a tengelyterhelést, a jármű sebességét és a tengelytávolságot. A két neurális háló által generált adatok segítségével a rendszer rekonstruálja a szenzorjeleket, amelyeket az eredeti, mért jelekkel összehasonlítva egy hibafüggvényt számol. A cél ennek a hibának a minimalizálása, amely tanulási folyamaton keresztül valósul meg. A módszer működését és a neurális hálózatok szerepét a 7. Ábra szemlélteti.



7. Ábra: A javasolt módszer keretrendszere. CNN-1: a tengelyszámot becslő neurális háló; CNN-2: a tengelysúlyt, tengelytávolságot és sebességet becslő neurális háló (Forrás: [3])

A bemeneti szenzoradatok előkészítése során a jeleket a min-max normalizációs módszerrel 0 és 1 közötti tartományra skálázzák, hogy azok egységes formátumban

kerüljenek a neurális hálózatba. A tanulási folyamathoz szükséges adatokat, például a tengelytávolságot és a jármű sebességét, kiegészítő eszközök, például kamerák vagy más szenzorok biztosítják. Ez lehetővé teszi, hogy a javasolt neurális hálózati architektúra a járművek súlyadatainak hiányában is hatékonyan tanítható legyen, így csökkentve a költséges mérőeszközök iránti igényt. A módszer alkalmazásával nemcsak a hídterhelések pontosabb becslése válik lehetővé, hanem a hidak hosszú távú fenntarthatóságának támogatása is, amely jelentős gazdasági és biztonsági előnyökkel jár. Mivel a referencia cikkben használt eredeti adathalmaz nem állt rendelkezésünkre így az elkészült modulokat csak a BME S1 adathalmazon tanítottam és teszteltem. [3]

3 Saját Megvalósítás

Ebben a fejezetben részletesen bemutatom a hivatkozott cikkben ismertetett BwimNet módszer saját implementációját, amelyet célzott módosításokkal továbbfejlesztettem a Komáromi Monostori híd nyúlásmérő szenzorjai által gyűjtött adatok felhasználásával. A korábbi fejezetekben tárgyalt BWIM (Bridge Weigh-In-Motion) rendszerek és a fizikainformált neurális hálózatok (PINN-ek) alapjaira építve a célom olyan modulok létrehozása, amik képesek a bemeneti szenzor jelekből megbecsülni a járművek sebességét, tengelyszámát és azok keresztirányú pozicióját. Az implementáció során a cikkben javasolt konvolúciós neurális hálózati (CNN) architektúrát vettem alapul, de kísérletet tettem a rendszer olyan irányú módosítására, amely mellőzi a kiegészítő szenzorokat. Ez a megközelítés sikeres megvalósítás esetén jelentősen egyszerűsítené a telepítést.

3.1 Keresztirányú pozíció becslő modul

A keresztirányú pozícióbecslő modul a kifejlesztett BWIM-rendszer kezdeti komponenseként szolgál. A modul feladata a járművek keresztirányú pozíciójának becslése a híd pályalemezén, amely információ elengedhetetlen a terheléseloszlás pontos modellezéséhez, hiszen a járművek oldalirányú elhelyezkedése közvetlenül befolyásolja a szenzorok által regisztrált deformációk mértékét és eloszlását. A saját megvalósításomban ez a modul a BME-Simulated-I szimulált adathalmazra támaszkodik, és a Komáromi Monostori híd nyúlásmérő szenzorai által generált adatok feldolgozását végzi.

3.1.1 Adatfeldolgozás és előkészítés

A járművek pozícióját jellemzően a forgalmi sávok geometriájához viszonyítva határozzuk meg. A vizsgált hídon a két sáv keresztmetszetének figyelembe vett szélessége (a szimulált sáv szélessége) külön-külön 3,75 méter. Ebből adódóan a járművek lehetséges keresztirányú pozíciója a sávközéptől számítva -1,875 m és +1,875 m közé esik. Mivel a neurális hálózatok tanítása hatékonyabb egységes skálán, a fizikai paramétereket célszerű normalizálni. A becslés nálam ezért a 0 és 1 közötti tartományon történik, ahol a 0 érték a sáv bal szélét (-1,875 m), az 1 pedig a jobb szélét (+1,875 m) képviseli.

Az idősoros adatok feldolgozása során gyakori eljárás a dimenziószám csökkentése a számítási igény minimalizálása érdekében, például a jelek jellegzetes statisztikai mutatóinak kinyerésével. Ennek megfelelően a modul nem a teljes időbeli lefutást, hanem a szenzorjelek terjedelmét használja bemenetként, amelyek a jelek maximális és minimális értékei közötti különbséget képviselik. Ezzel a módszerrel az idősoros adatok komplexitását sikerült egy kompaktabb vektorrá redukálni.

A BWIM rendszerekben a szenzorok elhelyezkedése alapján megkülönböztetünk elsődleges (primer) és másodlagos (szekunder) szenzorsorokat, ahol jellemzően az elsődleges sor szolgáltatja a legrelevánsabb információt a teherbecsléshez. Bár a rendelkezésre álló adathalmaz fájljai összesen 26 szenzor adatát tartalmazzák, a fejlesztés során kizárólag az elsődleges szenzorsorhoz tartozó érzékelőket vettem figyelembe. A Komáromi híd esetében ez az első 16 szenzort (main sensors) jelenti. A másodlagos (secondary) szenzorokat a jelenlegi modulban nem használtam fel, ezzel is a legfontosabb mérési pontokra fókuszálva.

Az adatok technikai előkészítése során betöltöttem a híd szenzorjeleit tartalmazó train és test adathalmazokat. Mivel a nyers mérések változó hosszúságú idősorok, az egységesítés érdekében a rövidebb mintákat nullákkal történő kiegészítéssel (zero-padding) a leghosszabb jelsorozat méretére bővítettem. A feldolgozás következő lépésében a dimenziócsökkentés részeként leválogattam a releváns, elsődleges szenzorsorhoz tartozó 16 csatornát, majd ezeken kiszámítottam a jelek terjedelmét. A kapott értékeket MinMaxScaler segítségével 0 és 1 közé skáláztam, biztosítva a modell stabilitását és a tanulás hatékonyságát.

A tanításhoz a train adathalmazból véletlenszerűen elkülönítettem 10%-ot validációs halmaznak, míg a fennmaradó 90% szolgált a tényleges tanításra. A teszteléshez különálló adathalmazt használtam, így megelőzve a túltanulás (overfitting) jelenségét és biztosítva a modell generalizálhatóságát. Ez a szétválasztás elengedhetetlen annak ellenőrzésére, hogy a betanított modell mennyire teljesít jól ismeretlen adatokon.

3.1.2 Modellarchitektúra

A kifejlesztett modell egy háromrétegű Fully Convolutional Neural Network (FCNN), amelynek felépítése a következő:

- Bemeneti réteg: 16 neuronnal rendelkezik, ami megfelel az elsődleges szenzorok számának.
- Első rejtett réteg: A bemenetet 64 kimeneti neuronra bővíti. Ezt ReLU aktivációs függvény és 20%-os dropout követ a túltanulás elkerülése érdekében.
- Második rejtett réteg: A 64 bemenetet 128 kimenetre transzformálja, szintén ReLU aktivációval és 30%-os dropouttal.
- Kimeneti réteg: Egyetlen értéket állít elő, amely a normalizált keresztirányú pozíciót képviseli.

A dropout rétegek véletlenszerűen nullázzák a neuronok egy részét a tanítás során, növelve a modell robusztusságát. A modell implementációja PyTorch Lightning keretrendszerben történt, ami leegyszerűsíti a tanítási ciklust és a veszteségkövetést.

3.1.3 Tanítási eljárás

A tanítás 100 epochon keresztül zajlott Adam optimalizálóval, 0,001 tanulási rátával. Az Adam algoritmus adaptív módon korrigálja a lépésméretet a gradiensek alapján, ezzel gyorsabb konvergenciát eredményezve a hagyományos SGD optimalizálóhoz képest. Veszteségfüggvényként az átlagos négyzetes hibát (Mean Squared Error, MSE) használtam, mivel regressziós feladatot oldunk meg, ahol a cél a predikció és a valós pozíció (label) közötti eltérés minimalizálása.

A folyamat során backpropagation algoritmust alkalmaztam, minden iterációban a modell becslését összehasonlítottuk a valós címkékkel, és a kiszámított hibát visszaterjesztettük a hálózaton keresztül, módosítva a súlyokat a pontosabb becslés érdekében. A validációs halmaz segítségével early stopping technikát alkalmaztam a túltanulás megelőzésére.

3.1.4 Eredmények és kiértékelés

A tesztelés során a modell teljesítményét az átlagos abszolút hiba (MAE) és a gyökér átlagos négyzetes hiba (RMSE) mérőszámokkal értékeltem. A kapott eredmények a következők:

- Átlagos abszolút hiba (MAE): 66,23 mm
- Négyzetes középhiba (RMSE): 87,50 mm

A 3750 mm-es sávszélességhez viszonyítva ez az eredmény jelzi, hogy a modell megbízhatóan becsüli a pozíciót zajos szenzoradatokból is.

3.2 Általános adatfeldolgozási és előkészítési eljárások

A dolgozatban bemutatott mélytanulási modulok többsége (tengelyszámbecslő, sebességbecslő, járműparaméter-becslő) idősoros jelek feldolgozásán alapul. A fejlesztés és a tanítás alapjául a BME-Simulated-I (BME S1) szintetikus adathalmaz szolgált. Mivel a neurális hálózatok hatékony tanítása megköveteli a bemeneti adatok egységes szerkezetét, a fejlesztés során a különböző moduloknál azonos vagy hasonló adat-előfeldolgozási lépéseket alkalmaztam. Ebben az alfejezetben összefoglalom ezt az egységes eljárást, amely minden idősoros bemenetet igénylő modulra érvényes.

3.2.1 Szenzorkiválasztás, aggregálás és időablak-kezelés

A BWIM rendszerekben a híd válaszát jellemzően több szenzorral mérik. A számítási igény optimalizálása és a bemeneti dimenziószám csökkentése érdekében a fejlesztés során a rendelkezésre álló 26 szenzor közül kizárólag a primer (elsődleges) szenzorsor 16 jelét használtam fel. A kiválasztott 16 szenzor egyedi időfüggő jeleit minden esetben összegeztem. Ez az eljárás fizikai szempontból előnyös, mivel az összegzett jel a híd globális válaszát reprezentálja, miközben az átlagolódás hatására a lokális szenzorzajok mértéke csökken, javítva a jel-zaj arányt (SNR).

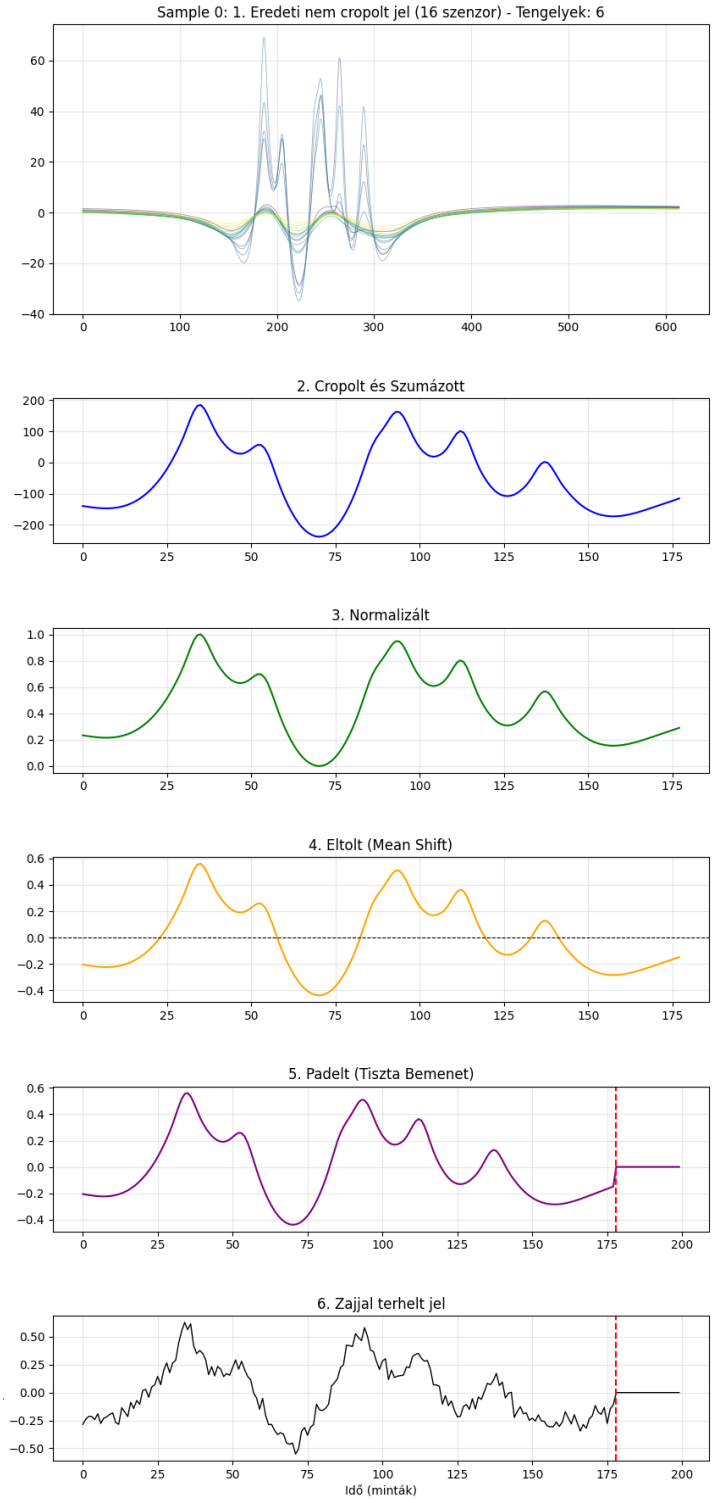
A közúti forgalomban részt vevő járművek sebessége és tengelytávolsága változó, ezért a szenzorok által rögzített jelek hossza eltérő. A tanítás során az időablak kivágásakor (cropping) a jármű pontos áthaladási idejéhez véletlenszerűen generált biztonsági tartományt (margót) adtam hozzá. Ez a véletlenszerű eltolás (jittering) növeli a modell robusztusságát, mivel megakadályozza, hogy a hálózat túlzottan a jel pozíciójára hagyatkozzon.

3.2.2 Jelformázás, normalizálás és adatbővítés

A kivágott jelek amplitúdóját első lépésben Min-Max normalizációval a [0, 1] tartományba skáláztam, biztosítva ezzel a különböző járművek jeleinek egységes nagyságrendjét a stabil konvergencia érdekében. Ezt követően a jeleken középérték-eltolást végeztem, azaz minden idősorból kivontam a saját átlagát. Ez a művelet eltávolítja a statikus eltolódásokat, és biztosítja, hogy a jelek a nulla érték körül ingadozzanak.

Mivel a konvolúciós neurális hálózatom fix méretű bemenetet igényel, a neurális háló bemeneti jelének hosszát, a szakirodalommal összhangban, egységesen 200 mintapontra fixáltam. Amennyiben a már normalizált és nullára centrírozott jel rövidebb volt ennél, a hiányzó értékeket a jel végén nullákkal pótoltam (zero-padding). Ezzel szemben, ha a jel hossza meghaladta a 200 mintapontot, az adott mintát a tanítás során figyelmen kívül hagytam. Mivel a jel átlaga az előző lépés miatt nulla, a nullákkal való kiegészítés törésmentesen illeszkedik a jelhez. Végezetül, a modell valós, zajos környezetben való robusztusságának növelése érdekében a tanítás során a bemeneti jelekhez véletlenszerű, 4-6% közötti Gauss-eloszlású zajt adtam hozzá.

A teljes előfeldolgozási láncolat lépéseit a 8. ábra szemlélteti egy konkrét járműáthaladás példáján. Az ábra fentről lefelé haladva mutatja be a jel átalakulását. A legfelső grafikon a 16 primer szenzor eredeti, nyers jelét ábrázolja a teljes rögzített időtartamban. A második sorban a járműablak szerinti vágás és a szenzorjelek összegzése után kapott jel látható. Ezt követi a Min-Max normalizált állapot (3. sor), majd a középérték-eltoláson átesett, nullára centrírozott változat (4. sor). Az ötödik grafikon a 200 mintapont hosszúságúra kiegészített (padelt), de még zajmentes jelet mutatja, amely az adatbázis tiszta alapját képezi. Végezetül, a legalsó, hatodik grafikon a neurális hálózat tényleges bemenetét reprezentálja, ahol a jelhez már mesterséges zajt adtunk, biztosítva a modell robusztusságát.



8. ábra: A teljes adat-előfeldolgozási láncolat lépéseinek vizualizációja egy kiválasztott mintán

3.3 Tengelyszámbecslő modul

A járművek tengelyeinek pontos meghatározása a BWIM (Bridge Weigh-In-Motion) rendszerek egyik kulcslépése, mivel ez az információ szolgáltatja az alapot a tengelysúlyok, tengelytávolságok és a teljes járműtömeg becsléséhez. A hibás tengelyszámbecslés dominóhatásként torzíthatja a teljes súlybecslési láncot, ezért a pontos és robusztus detektálás kiemelt fontosságú. A jelen munkában kifejlesztett tengelyszámbecslő modul célja, hogy a hídra telepített nyúlásmérő szenzor által szolgáltatott jelek alapján mély neurális háló segítségével automatikusan meghatározza az áthaladó jármű tengelyeinek számát. A modul tanítása és tesztelése során csak az egysávos adatokat használtam.

3.3.1 Adatfeldolgozás és előkészítés

A modul fejlesztéséhez a BME S1 szimulált adathalmaz szolgált alapul. Az adatok előkészítése során a 3.2 fejezetben részletezett általános adatfeldolgozási eljárást követtem. Ennek megfelelően a bemenetet a primer szenzorsor 16 érzékelőjének jeleiből képeztem: az időablak kivágása után a szenzorjeleket összegeztem, így egyetlen aggregált idősort kaptam, amely a híd globális válaszát reprezentálja.

A neurális hálózat bemenete tehát a 3.2.2 pontban leírtak szerint normalizált, középérték-eltoláson átesett, és egységesen 200 mintapont hosszúságúra kiegészített jel. Ez a formátum biztosítja, hogy a hálózat a különböző sebességű és tengelytávolságú járművek esetén is konzisztens bemenetet kapjon, míg a tanítás során alkalmazott mesterséges zajinjektálás a modell robusztusságát növeli.

3.3.2 Modellarchitektúra

A tengelyszámbecslő neurális hálózat a PyTorch keretrendszerben implementált, egyedi AxleCounter nevű modul formájában valósult meg. A hálózat fő komponensei egy-dimenziós konvolúciós és reziduális blokkok, amelyek időbeli mintázatok kinyerésére és nemlineáris jellemzők tanulására optimalizáltak.

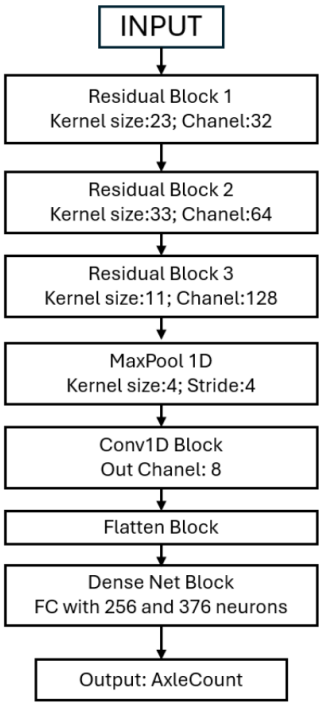
Az architektúra szerkezete a következőképpen épül fel és a 9. ábra szemlélteti:

1. **Három egymást követő ResidualBlock** (kernelméretek: 23, 33, 11), amelyek fokozatosan növelik a reprezentációs mélységet (32 → 64 → 128 csatorna). A reziduális kapcsolatok megakadályozzák a gradiens eltűnését, elősegítve a mélyebb hálózat hatékony tanulását. Ezek a blokkok képesek megőrizni a korábbi

jellemzőket, miközben új mintázatokat is tanulnak, így a jármű tengelyeinek időbeli eloszlása jól felismerhetővé válik.

2. **MaxPool1D réteg** (kernel_size=4, stride=4), amely a jelhossz negyedére csökkentésével csökkenti a számítási igényt, miközben megőrzi a domináns szerkezeti jellemzőket.
3. **Konvolúciós blokk** (ConvBlock1D) 8 kimeneti csatornával, amely magasabb szintű absztrakt mintázatokat nyer ki, például a tengelyek közötti periódicitást.
4. **Flatten réteg**, amely a konvolúciós kimenetet vektorrá alakítja.
5. **DenseNet modul**, amely két teljesen összekapcsolt (fully connected) réteget tartalmaz, 256 és 376 neuronnal. Az aktiváció ReLU, a dropout valószínűség 0,1. A kimeneti réteg dimenziója megegyezik a becsülendő tengelyszám kategóriáinak számával (2–6 tengely).

A hálózat tehát a nyúlásmérő szenzor által szolgáltatott jelből hierarchikusan építkező, absztrakt jellemzőket tanul. Az első konvolúciós rétegek lokális mintázatokat, míg a mélyebb rétegek már globális jelstruktúrákat azonosítanak. Ez lehetővé teszi, hogy a hálózat a jármű áthaladása során megjelenő ismétlődő csúcsokat (tengelyek) automatikusan felismerje, anélkül, hogy heurisztikus jel-feldolgozó algoritmusokat kellene alkalmazni.



9. ábra: A tengelyszám becslő neurális háló architektúrája

3.3.3 Tanítási eljárás

A tanításhoz felügyelt tanulási módszert alkalmaztam, ahol a háló célja a járművek tengelyszámának osztályozása volt. A tanulóhalmaz 10 000 szimulált járműadatból állt, amelyek Non-Tandem, Non-Convoy és Tandem, Non-Convoy kategóriákat tartalmaztak, 2–6 tengelyes járműverziókra bontva.

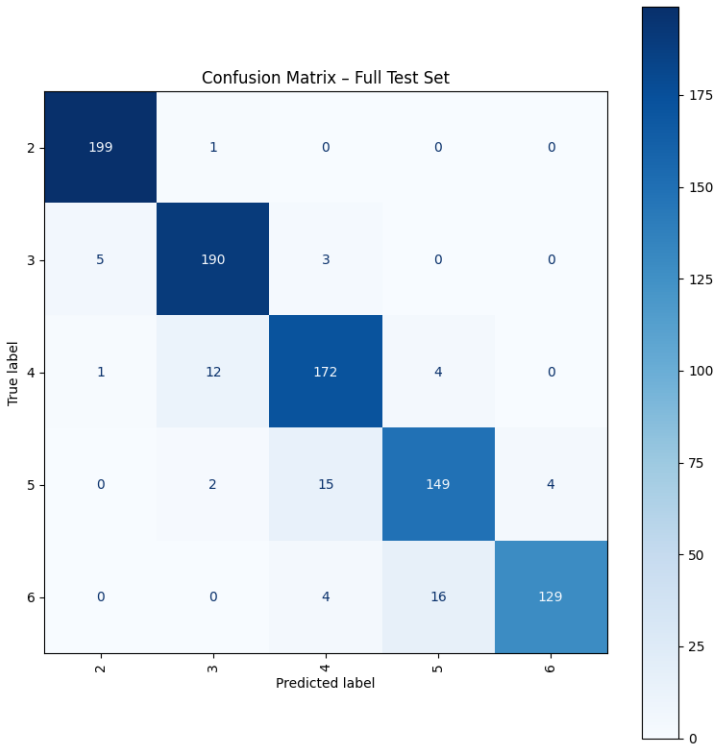
A tanítás során az Adam optimalizálót használtam $learning\ rate = 10^{-5}$ értékkel, a veszteségfüggvény pedig kategóriás keresztentrópia (CrossEntropyLoss) volt. A folyamat 200 epochon keresztül zajlott, early stopping technika alkalmazásával, amely a validációs veszteség javulásának hiánya esetén megszakította a tanulást a túltanulás elkerülése érdekében. Az optimális paraméterek elérése után a háló súlyait elmentettem, és külön értékelést végeztem a valós adatokon.

3.3.4 Eredmények és kiértékelés

A modell teljesítményét két különböző adatkategórián vizsgáltam:

Szimulált BME S1 adatokon a tengelyszámbecslő modul 92,60%-os osztályozási pontosságot ért el. Ez az eredmény igazolja, hogy a háló hatékonyan megtanulta a szimulált jelek és a tengelykonfigurációk közötti korrelációt. A tanulás során jól általánosított a különböző járműtípusokra és konfigurációkra, ami megerősíti a CNN-alapú megközelítés alkalmasságát idősoros szenzoradatokra.

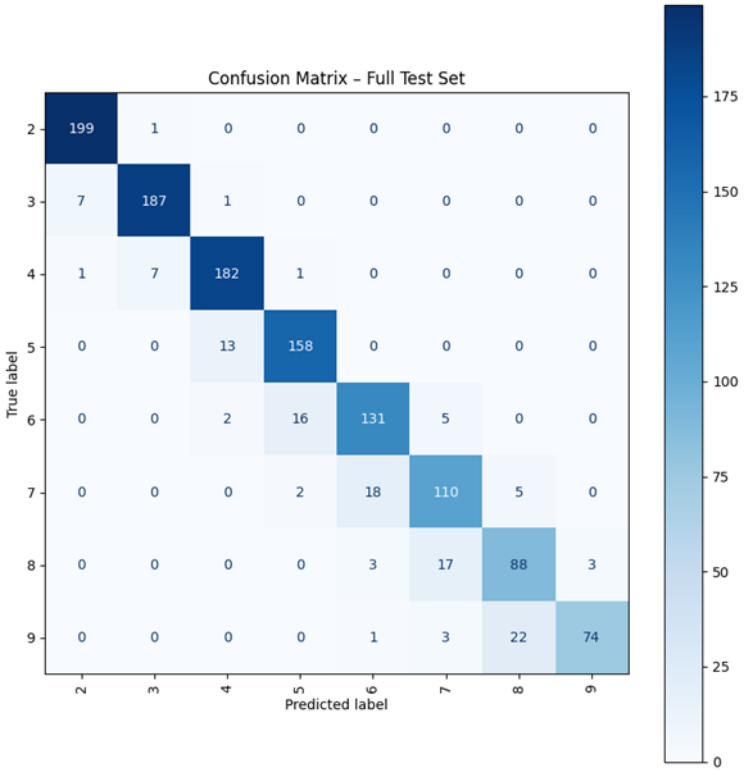
Az eredmény jobb kiértékelésére elkészítettem a 10.ábrán látható konfúziós mátrixot, amin megfigyelhetjük, hogy a tengelyszám becslő modul a különböző tengely konfigurációkat milyen pontossággal talált el és hogy a tévesztéseknek milyen az eloszlása és iránya.



10. ábra: Általam fejlesztett tengelyszámbecslő konfúziós mátrixa 2-6 tengelyes konfigurációkra

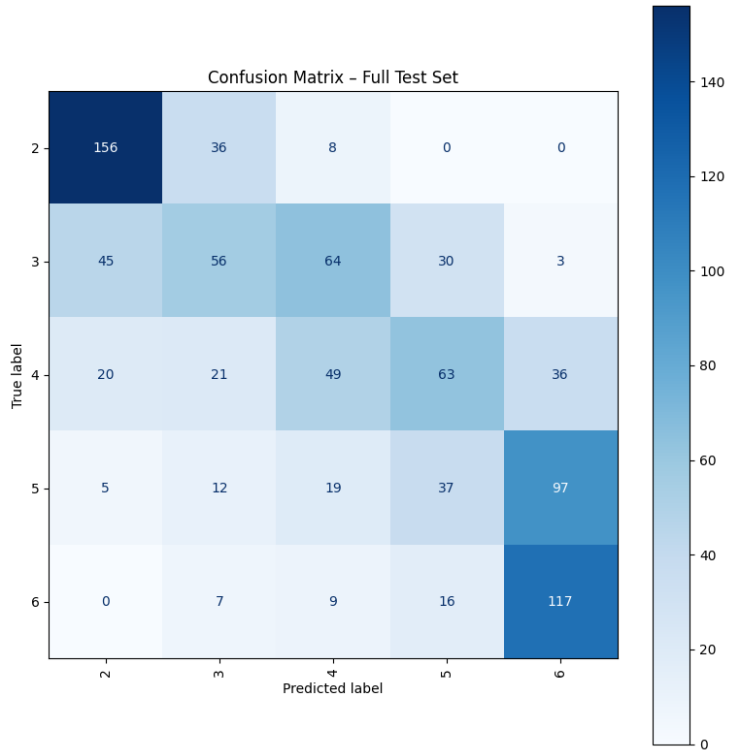
A valós komáromi hídadatokon a pontosság 43,96%-ra csökkent, ami jelentős eltérést mutat a szimulált környezethez képest. Ennek elsődleges oka vélhetően az, hogy a tanítás során alkalmazott mesterséges zajinjektálás tisztán Gauss-eloszlást követett, míg a valós mérési környezetben a jelek gyakran nem-Gauss típusú zajokat és torzításokat is tartalmaznak. Mivel a hálózat ezekkel a specifikus zajmintázatokkal nem találkozott a tanítás során, ez korlátozta a modell generalizációs képességét.

A vizsgálat kiterjesztéseként a hálózat tanítását elvégeztem a szélesebb, 2–9 tengelyes tartományra is. Bár a gyakorlatban a hatnál több tengellyel rendelkező járművek előfordulása ritka, a kísérlet célja a modell robusztusságának vizsgálata volt komplexebb konfigurációk esetén. Ebben a kiterjesztett tartományban a szintetikus adatokon elért pontosság 89,89%-ra adódott, míg a valós adatokon a modell teljesítménye 24,18%-ra esett vissza. A szintetikus adatokon tanított háló konfúziós mátrixát a 11. ábra szemlélteti.

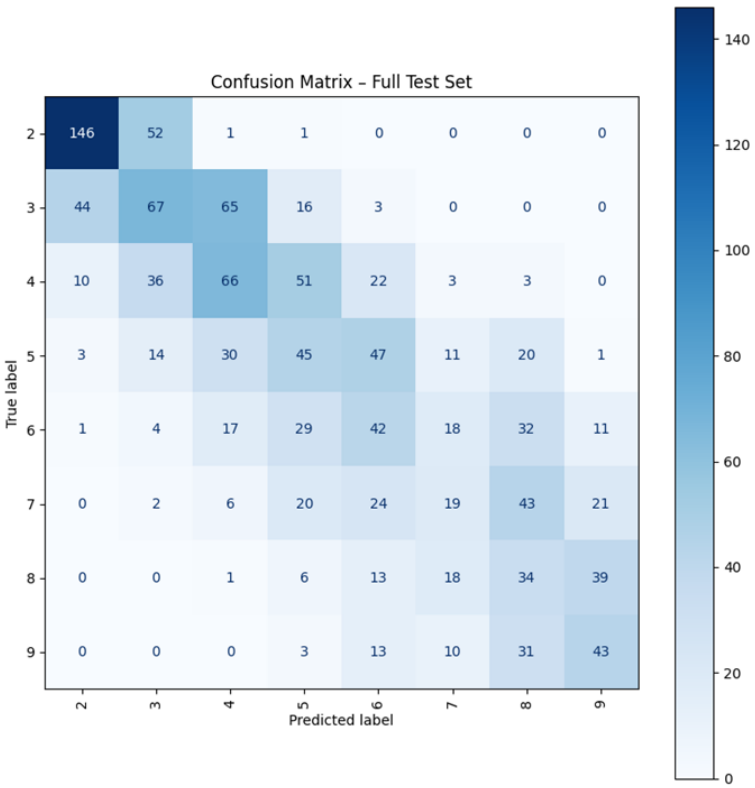


11. ábra: Általam fejlesztett tengelyszámbecslő konfúziós mátrixa 2-9 tengelyes konfigurációkra

Szakirodalmi referenciaként implementáltam és újra tanítottam a Wu és társai [3] által publikált eredeti tengelyszámbecslő architektúrát is, mind a 2–6, mind a 2–9 tengelyes konfigurációkra. A referencia-architektúra a 2–6 tengelyes tartományban a szintetikus adatokon 45,81%-os, míg a valós méréseken 31,87%-os pontosságot ért el. A kiterjesztett, 2–9 tengelyes vizsgálat esetén ezek az értékek rendre 36,75%-ra (szintetikus) és 25,27%-ra (valós) módosultak. A szintetikus adathalmazon elért eredményének eloszlását a 2-6 tengelyes esetben a 12. ábra, míg a 2-9 tengelyes esetekben a 13. ábra szemlélteti.



12. ábra: Wu és társai által fejlesztett tengelyszámbecslő konfúziós mátrixa 2-6 tengelyes konfigurációkra



13. ábra: Wu és társai által fejlesztett tengelyszámbecslő konfúziós mátrixa 2-9 tengelyes konfigurációkra

3.3.5 Pipeline Integráció

A tengelyszám-becslő modul működésének megértését követően tekintsük át, miként illeszthető ez a modul egy feldolgozási folyamatba (pipeline). Ebben az alfejezetben részletesen bemutatom a modul integrációjának módját, a használt bemeneti és kimeneti adatokat, valamint a kód által végrehajtott feldolgozási lépéseket. Az integráció során a Szinyéri Bence által megalapozott neuralbwim csomagot és abban szereplő modulokat használtam alapként.

Adatbetöltés a KomaromKozutLoader() segítségével:

A pipeline első lépése a valós járműjelek betöltése. Ehhez a neuwalbwim csomagból a KomaromKozutLoader() osztályt használjuk, amely a Komáromi híd forgalmából származó mérési adatokat gyűjti össze. A loader a mintákat fájlokból gyűjti, majd a kiválasztott mintát Signal objektummá alakítja, amely a szenzorok által mért időfüggő jeleket tartalmazza:

```
loader = KomaromKozutLoader()
path = loader.collect()[0]
signal, _ = loader.load_sample(path)
```

14. ábra: A pipelineban a fájlok betoltésére szolgáló kód

A signal objektum a pipeline bemeneti adatát képezi, és tartalmazza az összes szenzor méréseit a jármű áthaladása során.

Járműablak detektálás CNN alapú modullal:

A következő lépésben a járművek pontos időablakát detektáljuk. Ezt a feladatot a neuralbwim csomagból a CnnBasedWindowDetectorV2_create modul látja el, amely egy előre betanított konvolúciós neurális hálózaton alapul. A modul bemenete a teljes jel, a kimenete pedig az egyes járművekhez tartozó időablakok (time_window), amelyeket a pipeline a következő modulhoz továbbít. Az integráció során a konfigurációs fájlok és a súlyok megadása kritikus a helyes működéshez:

```
pipeline = PipelineBase()
pipeline.append(
    CnnBasedWindowDetectorV2_create(
        config_folder_path=ResHelper.fetch(f'Komarom_full/alg_configs/window_detectors/edge/v1'),
        config='CnnBasedWindowDetector_config.json',
        net_config='net_arch_config.json',
        pl_ckpt_path='weights.ckpt',
        net_name='net',
        denoise='denoise_pipeline.json'
    )
)
pipeline.append(
    AxleCountPredictor(
        in_parameters=AxleCountPredictor.InputParameters(
            signal=VehicleModel.Signal, time_window=VehicleModel.MainSensorTimeWindow),
        out_parameters=AxleCountPredictor.OutputParameters(
            axle_num='axle_count'),
        checkpoint_path = ResHelper.fetch('C:/Users/rober/Documents/GitHub/neuralbwim/src/rohrsetzer_sandbox/'
                                          'bwimnetpp/bwimnetpp_training/tmp_ckpt/axlecountercnn-epoch=82-val_loss=0.37.ckpt')
    )
)
```

15. ábra: Pipeline komponensek: Ablak detektáló és tengelyszám becslő

Ez a lépés biztosítja, hogy a tengelyszám-becslő modulnak rendelkezésre álljanak az időablakok, amik szükségesek a jel megfelelő feldolgozásához.

Tengelyszám-becslés az AxleCountPredictor modullal:

Az AxleCountPredictor modul a pipeline következő eleme, amely a CNN által meghatározott járműablak és a szenzorjelek alapján predikálja a jármű tengelyeinek számát. A modul bemenete a Signal objektum és az időablak (time_window), a kimenete pedig az axle_num, azaz a becsült tengelyszám.

A modul inicializálásakor betöltjük a betanított CNN súlyait, és a modellt eval() módba állítjuk, hogy csak becslést adjon, tanulás ne történjen futás közben:

```
def __init__(self, in_parameters: InputParameters, out_parameters: OutputParameters, checkpoint_path: str):
    super().__init__(in_parameters=in_parameters, out_parameters=out_parameters)
    self._device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    self._model = AxleCounter(signal_length=200, num_axles=8, in_channels=1)

    # PyTorch Lightning checkpoint helyes betöltése
    checkpoint = torch.load(checkpoint_path, map_location=self._device)
    state_dict = checkpoint['state_dict']
    # 'model.' prefix eltávolítása a kulcsokból, ha létezik
    stripped_state_dict = {k.replace('model.', ''): v for k, v in state_dict.items()}
    self._model.load_state_dict(stripped_state_dict)

    self._model.to(self._device)
    self._model.eval()
```

16. ábra: A pipeline inicializálása

Ezzel biztosítjuk, hogy a hálózat a tanult paraméterek alapján adjon kimenetet.

Ezen kívül szükséges a bemeneti szenzor adatok elő feldolgozása is mielőtt azt a tengelyszám becslő neurális hálónak tovább adnánk. A feldolgozás során a bemeneti jelekből az időablak segítségével kivágjuk a jármű ablakát majd azokat a jeleket, ahol ez az ablak 200-nál hosszabb, azokat kihagyjuk, ha pedig rövidebb mint 200 akkor padding segítségével feltöltjük 0-kal hogy pont 200 legyen a hossza. A szenzor adatok közül csak az elsődleges szenzor sor összegjelét használjuk fel (az első 16db), azt az átlaggal eltoljuk majd normalizáljuk mielőtt azt a tengelyszám becslőnek átadjuk:

```
# Crop signal based on time_window
cropped_signal = signal.crop_by_time(time_window[0], time_window[1])
if len(cropped_signal) > 200:
    raise ValueError(f"Cropped signal too long: {len(cropped_signal)} > {200}, skipping.")
sensor_names = cropped_signal.sensors[:16]
sum_signal = cropped_signal.sum(sensor_names)
sum_signal -= sum_signal.mean()
# Pad
actual_len = len(sum_signal)
pad_width = (0, 200 - actual_len)
sum_signal = np.pad(sum_signal, pad_width, mode='constant')
# Tensor, normalize
input_tensor = torch.from_numpy(sum_signal).unsqueeze(0).unsqueeze(0)  # [1, 1, signal_length]
input_tensor = input_tensor.float()
```

17. ábra:Fájlok előkészítése a tengelyszám becslő modulnak

Ez az előkészítés biztosítja, hogy minden jármű jelhossza egységes legyen a hálózat számára, és a hálózat képes legyen megbízható becslést adni.

A CNN a bemeneti jelből nyolc kimeneti értéket ad, amelyek a lehetséges tengelyek valószínűségeit reprezentálják. A legvalószínűbb osztály kerül kiválasztásra, majd a kimeneti axle_num változóban tárolódik:

```
# Predict with model
with torch.no_grad():
    output = self._model(input_tensor)
    _, predicted_class = torch.max(output, 1)
    axle_num = predicted_class.item() + 2  # +2 because class 0 is 2 axles

return [vm], self.OutputParameters(axle_num=axle_num)
```

18. ábra: A tengelyszám becslő modul kimenete

Ez a logika biztosítja, hogy a modul mindig a legvalószínűbb tengelyszámot adja vissza a járműre vonatkozóan.

A teljes pipeline végrehajtásakor a jelek végig mennek az ablakdetektoron, majd a tengelyszám becslő modullal becsüljük az egyes járművek tengelyszámát. A végeredmény a VehicleModel objektum kimeneti attribútumai között tárolódik.

A teljes kód az alábbi képeken látható:

```
class AxleCountPredictor(PipelineItem):  3 usages   Rotidronik +1 *

    Edit | Explain | Test | Document | Fix
    @dataclass   Szinyéri Bence
    class InputParameters:
        signal: Union[str, Signal]
        time_window: Union[str, Tuple[TimeStamp, TimeStamp]]

    @dataclass   Szinyéri Bence
    class OutputParameters:
        axle_num: Union[str, int]

    Edit | Explain | Test | Document | Fix
    def __init__(self, in_parameters: InputParameters, out_parameters: OutputParameters, checkpoint_path: str):
        super().__init__(in_parameters=in_parameters, out_parameters=out_parameters)
        self._device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        self._model = AxleCounter(signal_length=200, num_axles=8, in_channels=1)

        # PyTorch Lightning checkpoint helyes betöltése
        checkpoint = torch.load(checkpoint_path, map_location=self._device)
        state_dict = checkpoint['state_dict']
        # 'model.' prefix eltávolítása a kulcsokból, ha létezik
        stripped_state_dict = {k.replace('model.', ''): v for k, v in state_dict.items()}
        self._model.load_state_dict(stripped_state_dict)

        self._model.to(self._device)
        self._model.eval()
```

19. ábra: A pipelineba beépített tengelyszám becslő modul teljes kódja

```
    def process(self, vm: Union[Dict, VehicleModel], params: InputParameters) -> \  Rotidronik +1
            Tuple[List[Union[Dict, VehicleModel]], Union[OutputParameters, List[OutputParameters]]]:
        signal: Signal = params.signal # itt ez mar egy Signal objektum
        time_window: Tuple[TimeStamp, TimeStamp] = params.time_window

        # Crop signal based on time_window
        cropped_signal = signal.crop_by_time(time_window[0], time_window[1])
        if len(cropped_signal) > 200:
            raise ValueError(f"Cropped signal too long: {len(cropped_signal)} > {200}, skipping.")



        sensor_names = cropped_signal.sensors[:16]
        sum_signal = cropped_signal.sum(sensor_names)
        sum_signal -= sum_signal.mean()

        # Pad
        actual_len = len(sum_signal)
        pad_width = (0, 200 - actual_len)
        sum_signal = np.pad(sum_signal, pad_width, mode='constant')
        # Tensor, normalize
        input_tensor = torch.from_numpy(sum_signal).unsqueeze(0).unsqueeze(0)  # [1, 1, signal_length]
        input_tensor = input_tensor.float()

        # Predict with model
        with torch.no_grad():
            output = self._model(input_tensor)
            _, predicted_class = torch.max(output, 1)
            axle_num = predicted_class.item() + 2  # +2 because class 0 is 2 axles

        return [vm], self.OutputParameters(axle_num=axle_num)
```

```
if __name__ == '__main__':
    loader = KomaromKozutLoader()
    path = loader.collect()[0]
    signal, _ = loader.load_sample(path)
    pipeline = PipelineBase()
    pipeline.append(
        CnnBasedWindowDetectorV2_create(
            config_folder_path=ResHelper.fetch(f'Komarom_full/alg_configs/window_detectors/edge/v1'),
            config='CnnBasedWindowDetector_config.json',
            net_config='net_arch_config.json',
            pl_ckpt_path='weights.ckpt',
            net_name='net',
            denoise='denoise_pipeline.json'
        )
    )
    pipeline.append(
        AxleCountPredictor(
            in_parameters=AxleCountPredictor.InputParameters(
                signal=VehicleModel.Signal, time_window=VehicleModel.MainSensorTimeWindow),
            out_parameters=AxleCountPredictor.OutputParameters(
                axle_num='axle_count'),
            checkpoint_path = ResHelper.fetch('C:/Users/rober/Documents/GitHub/neuralbwim/src/rohrsetzer_sandbox/'
                                              'bwimnetpp/bwimnetpp_training/tmp_ckpt/axlecountercnn-epoch=82-val_loss=0.37.ckpt')
        )
    )
    vehicle_models = pipeline(signal)
    for vm in vehicle_models:
        print(f'Axle number: {vm["axle_count"]}')
```

20. ábra: A pipelineba beépített tengelyszám becslő modul teljes kódja (folytatás)

3.4 Sebességbecslő modul

A járművek sebességének pontos meghatározása kulcsfontosságú a BWIM (Bridge Weigh-In-Motion) rendszerekben, mivel a sebesség közvetlenül meghatározza a hídon elhelyezett nyúlásmérő szenzorok által rögzített jelek időbeli és amplitúdóbéli alakulását. A sebesség ismerete elengedhetetlen a jármű tengelyei közötti távolság, valamint az egyes tengelyterhelések meghatározásához, ezért a sebességbecslés pontossága meghatározó a teljes BWIM-rendszer megbízhatósága szempontjából. A jelen munkában fejlesztett sebességbecslő modul célja, hogy a híd szenzoraiból származó nyúlásjelek alapján, mély neurális hálózat segítségével, külső mérőeszközök nélkül becsülje meg a járművek áthaladási sebességét. A megoldás előnye, hogy kizárólag a hídon elhelyezett érzékelők adataira támaszkodik, így könnyen integrálható bármely BWIM rendszerbe, és valós időben is alkalmazható.

3.4.1 Adatfeldolgozás és előkészítés

A sebességbecslő modul bemeneti adatainak előkészítése során a 3.2 fejezetben ismertetett általános eljárást követtem. A hálózat bemenetét ebben az esetben is a primer szenzorsor 16 érzékelőjének összegzett jele alkotja, mivel ez a redukált, aggregált jel stabilan reprezentálja a híd globális válaszát, miközben csökkenti a lokális szenzorzajok hatását.

A neurális hálózat bemenete tehát a járműablak kivágását követően normalizált, középérték-eltoláson átesett, és egységesen 200 mintapont hosszúságúra kiegészített (padelt) idősor. Ez az egységesített formátum teszi lehetővé, hogy a konvolúciós hálózat a különböző sebességgel áthaladó járművek jeleit, a mintavételi hossz eltéréseitől függetlenül, hatékonyan feldolgozza. A modell robusztusságának növelése érdekében a tanító adathalmazt itt is mesterséges zajinjektálással bővítettem.

3.4.2 Modellarchitektúra

A sebességbecslő hálózat alapját egy egy-dimenziós konvolúciós neurális hálózat (1D CNN) képezi, amelyet kifejezetten idősoros szenzoradatok feldolgozására fejlesztettem. Az architektúrát a SpeedPredictor osztály implementálja, amely több konvolúciós és reziduális blokkot tartalmaz. A háló szerkezete a jel időbeli jellemzőinek hierarchikus kinyerésére épül, így képes a nyúlásmérő szenzor által szolgáltatott jelekben megbújó lokális és globális mintázatok felismerésére.

A konvolúciós rész három egymást követő reziduális blokkot tartalmaz, amelyek kernelmérete fokozatosan csökken (33, 23, 11). Ez a kialakítás lehetővé teszi, hogy a hálózat először szélesebb időtartamra kiterjedő jelstruktúrákat, majd finomabb, nagyobb felbontású mintázatokat tanuljon meg. A reziduális kapcsolatok segítik a stabil tanulást azáltal, hogy megőrzik a korábbi rétegek információit, és megakadályozzák a gradiens eltűnését a visszaterjesztés során. A konvolúciós rétegeket egy további ConvBlock1D réteg egészíti ki, amely tovább mélyíti a reprezentációt, és képes magasabb szintű jellemzők kinyerésére, például a jármű sebességére utaló frekvencia- vagy amplitúdómintázatok azonosítására.

A hálózat kimenete a konvolúciós blokkok után lapításra kerül, majd egy FC-be kerül. Ez két rejtett réteget tartalmaz, amelyek 128 és 64 neuront foglalnak magukban. A rejtett rétegek aktivációs függvénye LeakyReLU, amely hatékonyan kezeli a negatív bemeneteket, és megelőzi az egységek „elhalását”. A hálóban 0,1 értékű dropout rétegek kerültek elhelyezésre, melyek a túltanulás kockázatát csökkentik, miközben megőrzik a modell reprezentációs képességét. A kimeneti réteg egydimenziós, és a jármű sebességét adja meg méter per másodpercben. A háló kimenetére egy sigmoid aktivációs függvény került, amelyet lineáris skálázással 10 és 30 m/s közötti sebességtartományra vetítettem. Ez a tartomány lefedi a Komáromi hídon mért járművek jellemző sebességét, és biztosítja, hogy a becsült értékek fizikailag értelmezhető tartományban maradjanak.

Ez a konvolúciós architektúra nagy előnye, hogy képes közvetlenül az idősoros jelekből megtanulni a sebességre jellemző mintázatokat anélkül, hogy explicit jelfeldolgozási lépéseket (például csúcsdetektálást vagy Fourier-analízist) kellene alkalmazni. A háló a szenzorjel alakjából, amplitúdó-változásaiból és időbeli jellemzőiből implicit módon tanulja meg a sebességhez kötődő összefüggéseket.

3.4.3 Tanítási eljárás

A tanítás felügyelt módon történt, ahol a háló célja a jármű sebességének előrejelzése volt. A tanuláshoz ugyanazt a 10 000 szimulált mintát használtam, amelyek különböző tengelykonfigurációkat (2–6 tengely) és járműtípusokat tartalmaztak Non-Tandem Non-Convoy, valamint Tandem Non-Convoy kategóriákban. A tanulóhalmaz sokfélesége lehetővé tette, hogy a háló különböző mozgási mintázatokat tanuljon, ami javította az általánosító képességét.

A tanításhoz az Adam optimalizálót alkalmaztam, amely adaptív tanulási rátával dolgozik, és gyors, stabil konvergenciát biztosít. A tanulási ráta értékét 10^{-4}-re állítottam be, amely az empirikus tapasztalatok alapján jó kompromisszumot jelentett a tanulás sebessége és stabilitása között. A veszteségfüggvény Cross-Entropy Loss volt, amely a sebességtartományt diszkrét osztályokra bontotta, így a modell kategóriaalapú tanulást végzett. Bár a sebességbecslés természeténél fogva regressziós probléma, a kategorizált tanulás stabilabb gradiensáramlást és gyorsabb konvergenciát eredményezett. A tanítást 200 epochon keresztül végeztem, azonban early stopping technikát is bevezettem, amely a validációs veszteség növekedése esetén leállította a tanulást. Ezzel sikerült elkerülni a túltanulást, és a modell paramétereit az optimális tartományban rögzíteni.

A tanítás során a háló stabil konvergenciát mutatott, a veszteség monoton csökkenést követett, és a validációs teljesítmény gyorsan javult az első száz epoch alatt. A reziduális blokkok és a dropout együttesen biztosították a modell általánosíthatóságát, miközben megakadályozták a túlspecializálódást a tanítóadatokra.

3.4.4 Eredmények és értékelés

A tanított hálózat teljesítményét két adathalmazon vizsgáltam: egyrészt a szimulált BME S1 adatokon, másrészt a valós, Komáromi hídon mért szenzorjeleken. A szimulált környezetben a modell kiemelkedően jó eredményt ért el, az átlagos abszolút sebességhiba mindössze 0,56 m/s volt. Ez azt jelenti, hogy a háló gyakorlatilag képes pontosan reprodukálni a jármű sebességét, a tanulási folyamat során pedig sikeresen megtanulta a nyúlásmérő szenzor által szolgáltatott jelek és a sebesség közötti összefüggést.

Amikor a modellt a valós komáromi hídadatokra alkalmaztam, a sebességbecslés átlagos hibája 1,57 m/s-ra nőtt. Ez a hibanövekedés várható volt, hiszen a valós mérésekben a szenzorjelek jelentősen zajosabbak, a híd szerkezeti válasza eltér a szimulációban feltételezett modelltől, és a forgalmi körülmények, például a járművek tengelytávolsága, sebessége, illetve sávpozíciója, is nagyobb változatosságot mutatnak. Ennek ellenére a kapott hibaérték a BWIM rendszerek szempontjából még mindig teljes mértékben elfogadható, különösen figyelembe véve, hogy a sebességbecslés pusztán a híd deformációs válaszából történt, minden külső szenzor nélkül.

A vizsgálatok alapján a fejlesztett CNN-alapú sebességbecslő hálózat mind a szimulált, mind a valós környezetben képes értelmezhető és konzisztens eredményekre.

Míg a valós adatokon a pontosság elmarad a szimulációban tapasztalt értékektől, a modell még így is képes a sebesség nagyságrendi és relatív változásainak megbízható követésére. A hálózat tehát képes általánosítani, és megbízhatóan működik különböző környezeti és forgalmi feltételek mellett is, ami megerősíti a mély tanulási megközelítés létjogosultságát a BWIM rendszerekben történő sebességbecslésre.

3.4.5 Pipelineba Integrálás

Miután megértettük, hogyan működik a sebességbecslő modul, érdemes megvizsgálni, miként illeszthető be egy összetettebb feldolgozási folyamatba (pipeline). Egy ilyen integráció lehetővé teszi, hogy a modul ne önállóan, hanem egy nagyobb rendszer részeként működjön, ahol az általa előállított adatok további feldolgozási lépések bemenetéül szolgálnak. Ebben a fejezetben részletesen bemutatom a modul beépítésének módját, az integráció során figyelembe vett szempontokat, valamint a használt bemeneti és kimeneti adatok típusait. Emellett áttekintem a kód által végrehajtott feldolgozási lépéseket, és azt is, hogyan illeszkednek ezek a pipeline általános adatáramlásába és működésébe.

Adatbetöltés a KomaromKozutLoader() segítségével

A pipeline első lépése, a tengelyszám becslő modul pipelineositásához hasonlóan, a valós járműjelek betöltése. Ehhez a Szinyéri Bence által biztosított KomaromKozutLoader osztályt használjuk, amely a Komáromi híd forgalmából származó mérési adatokat gyűjti össze és tölti be. A loader a mintákat fájlokból gyűjti, majd a kiválasztott mintát Signal objektummá alakítja, amely a szenzorok által mért időfüggő jeleket tartalmazza:

```
loader = KomaromKozutLoader()
path = loader.collect()[0]
signal, _ = loader.load_sample(path)
```

21. ábra: A fájlok pipelineba történő betöltése

Ez a signal objektum tartalmazza az összes szenzor méréseit, ami a pipeline modulok első bemenetét képezi.

Járműablak detektálás CNN alapú modullal

A signalt először a Szinyéri Bence által biztosított jármű ablak detektáló modul (CnnBasedWindowDetectorV2_create) kapja meg, amely egy előre betanított konvolúciós neurális hálózaton alapul és a jelből kiszámolja a jármű ablakának idő

tartományát (time_window) és azok időpontját adja tovább a következő pipeline modulnak, azaz a sebesség becslő modulnak. Az integráció során a konfigurációs fájlok és a súlyok megadása kritikus a helyes működéshez:

```
pipeline = PipelineBase()
pipeline.append(
    CnnBasedWindowDetectorV2_create(
        config_folder_path=ResHelper.fetch('Komarom_full/alg_configs/window_detectors/edge/v1'),
        config='CnnBasedWindowDetector_config.json',
        net_config='net_arch_config.json',
        pl_ckpt_path='weights.ckpt',
        net_name='net',
        denoise='denoise_pipeline.json'
    )
)
pipeline.append(
    SpeedPred(
        in_parameters=SpeedPred.InputParameters(
            signal=VehicleModel.Signal,
            time_window=VehicleModel.MainSensorTimeWindow
        ),
        out_parameters=SpeedPred.OutputParameters(abs_speed='abs_speed'),
        checkpoint_path=ResHelper.fetch(
            'C:/Users/rober/Documents/GitHub/neuralbwim/src/rohrsetzer_sandbox/bwimnetpp/bwimnetpp_training/'
            'tmp_ckpt/speedpredcnn-epoch=47-val_loss=0.65.ckpt'
        )
    )
)
```

22. ábra: Pipeline komponensek: Ablak detektáló és sebesség becslő

Sebesség becslés a SpeedPred modullal

A sebesség becslő modul a pipeline következő eleme amely a jármű ablak és a szenzor jelekből a hídon áthaladó jármű sebességét fogja becsülni. A bemenet az előző modulból kijövő idő ablak (time_window) és a Signal Objektum, a kimenet pedig a jármű abszolút sebessége m/s-ban.

A modell inicializálásánál betöltjük a megfelelő súlyokat és a modelt eval() modba álltijuk, hogy csak becslést adjon, tanulás ne történjen futás közben:

```python
def __init__(self, in_parameters: InputParameters, out_parameters: OutputParameters,checkpoint_path: str):
    super().__init__(in_parameters=in_parameters, out_parameters=out_parameters)
    self._device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    self._model = SpeedPredictor(signal_length=200, in_channels=1)

    # PyTorch Lightning checkpoint helyes betöltése
    checkpoint = torch.load(checkpoint_path, map_location=self._device)
    state_dict = checkpoint['state_dict']
    # 'model.' prefix eltávolítása a kulcsokból, ha létezik
    stripped_state_dict = {k.replace('model.', ''): v for k, v in state_dict.items()}
    self._model.load_state_dict(stripped_state_dict)

    self._model.to(self._device)
    self._model.eval()
```

23. ábra: A sebesség becslő pipeline inicializálása

Mielőtt megkapná a sebesség becslő a szenzor adatokat, azelőtt még különböző átalakításokat kell rajtuk elvégezni, hogy illeszkedjenek a CNN által elvárt bemenethez.

A bemenetként kapott időablak segítségével a szenzor adatokból kivágjuk a jármű ablakot, és csak akkor dolgozunk vele, ha a hossza 200 alatt van, amennyiben ez nem teljesül, az adott fájlt eldobjuk és a következő fájlra térünk. Ha a 200 alatti az ablak hossza akkor viszont 200 hosszúságúra ki paddeljük nullákkal. Ezek után a szenzor adatokból kiválasztjuk az első 16 darabot, tehát az elsődleges szenzor sort, eltoljuk őket az átlagukkal, az első 16 darab szenzor összegjelét vesszük majd normalizáljuk a jelet és ezt fogjuk továbbadni a CNN-nek, amely megbecsüli a sebességet m/s-ban.

```python
# Crop signal
cropped_signal = signal.crop_by_time(time_window[0], time_window[1])
if len(cropped_signal) > 200:
    raise ValueError(f"Cropped signal too long: {len(cropped_signal)} > 200, skipping.")

# 16 szenzor összege + középérték-levonás
sensor_names = cropped_signal.sensors[:16]
sum_signal = cropped_signal.sum(sensor_names)
sum_signal -= sum_signal.mean()

# Pad 200 hosszra
pad_width = (0, 200 - len(sum_signal))
sum_signal = np.pad(sum_signal, pad_width, mode='constant')

# Tensor előkészítés és normalizálás
input_tensor = torch.from_numpy(sum_signal).unsqueeze(0).unsqueeze(0).float()
input_tensor = (input_tensor - input_tensor.mean(dim=-1, keepdim=True)) / (
        input_tensor.std(dim=-1, keepdim=True) + 1e-8
)
```

24. ábra: Az adatok előkészítése

```
# Predikció
with torch.no_grad():
    pred = self._model(input_tensor.to(self._device))
    abs_speed = float(pred.item())  # m/s
```

25. ábra: Az adatok átadása a sebességbecslőnek

Ez az előkészítés biztosítja, hogy minden jármű jelhossza egységes legyen a hálózat számára, és a hálózat képes legyen megbízható becslést adni.

A pipeline teljes implementálásának kódja az alábbi képeken látható:

```
class SpeedPred(PipelineItem):  3 usages  Rotidronik *

    Edit | Explain | Test | Document | Fix
    @dataclass  Rotidronik
    class InputParameters:
        signal: Union[str, Signal]
        time_window: Union[str, Tuple[TimeStamp, TimeStamp]]

    @dataclass  Rotidronik *
    class OutputParameters:
        abs_speed: Union[str, float]

    Edit | Explain | Test | Document | Fix
    def __init__(self, in_parameters: InputParameters, out_parameters: OutputParameters,checkpoint_path: str):
        super().__init__(in_parameters=in_parameters, out_parameters=out_parameters)
        self._device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        self._model = SpeedPredictor(signal_length=200, in_channels=1)

        # PyTorch Lightning checkpoint helyes betöltése
        checkpoint = torch.load(checkpoint_path, map_location=self._device)
        state_dict = checkpoint['state_dict']
        # 'model.' prefix eltávolítása a kulcsokból, ha létezik
        stripped_state_dict = {k.replace('model.', ''): v for k, v in state_dict.items()}
        self._model.load_state_dict(stripped_state_dict)

        self._model.to(self._device)
        self._model.eval()

    def process(self, vm: Union[Dict, VehicleModel], params: InputParameters) -> \  Rotidronik
            Tuple[List[Union[Dict, VehicleModel]], Union[OutputParameters, List[OutputParameters]]]:

        signal: Signal = params.signal  # itt ez mar egy Signal objektum
        time_window: Tuple[TimeStamp, TimeStamp] = params.time_window
        # Crop signal
        cropped_signal = signal.crop_by_time(time_window[0], time_window[1])
        if len(cropped_signal) > 200:
            raise ValueError(f"Cropped signal too long: {len(cropped_signal)} > 200, skipping.")

        # 16 szenzor összege + középérték-levonás
        sensor_names = cropped_signal.sensors[:16]
        sum_signal = cropped_signal.sum(sensor_names)
        sum_signal -= sum_signal.mean()

        # Pad 200 hosszra
        pad_width = (0, 200 - len(sum_signal))
        sum_signal = np.pad(sum_signal, pad_width, mode='constant')
```

```
        # Tensor előkészítés és normalizálás
        input_tensor = torch.from_numpy(sum_signal).unsqueeze(0).unsqueeze(0).float()
        input_tensor = (input_tensor - input_tensor.mean(dim=-1, keepdim=True)) / (
                input_tensor.std(dim=-1, keepdim=True) + 1e-8
        )

        # Predikció
        with torch.no_grad():
            pred = self._model(input_tensor.to(self._device))
            abs_speed = float(pred.item())  # m/s

        return [vm], self.OutputParameters(abs_speed=abs_speed)
```

```
if __name__ == '__main__':
    loader = KomaromKozutLoader()
    path = loader.collect()[0]
    signal, _ = loader.load_sample(path)

    pipeline = PipelineBase()
    pipeline.append(
        CnnBasedWindowDetectorV2_create(
            config_folder_path=ResHelper.fetch('Komarom_full/alg_configs/window_detectors/edge/v1'),
            config='CnnBasedWindowDetector_config.json',
            net_config='net_arch_config.json',
            pl_ckpt_path='weights.ckpt',
            net_name='net',
            denoise='denoise_pipeline.json'
        )
    )
    pipeline.append(
        SpeedPred(
            in_parameters=SpeedPred.InputParameters(
                signal=VehicleModel.Signal,
                time_window=VehicleModel.MainSensorTimeWindow
            ),
            out_parameters=SpeedPred.OutputParameters(abs_speed='abs_speed'),
            checkpoint_path=ResHelper.fetch(
                'C:/Users/rober/Documents/GitHub/neuralbwim/src/rohrsetzer_sandbox/bwimnetpp/bwimnetpp_training/'
                'tmp_ckpt/speedpredcnn-epoch=47-val_loss=0.65.ckpt'
            )
        )
    )

    vehicle_models = pipeline(signal)
    for vm in vehicle_models:
        print(f"Predicted speed: {vm['abs_speed']:.2f} m/s")
```

26. ábra: Sebességbecslő pipeline teljes kódja

3.5 Egyéb modulok

A korábban bemutatott tengelyszámbecslő és sebességbecslő modulok sikeres működése után a fejlesztés következő lépése az volt, hogy a rendszer képes legyen önállóan, külső tanítócímkék nélkül tanulni a szenzorjelekből. A cél egy olyan komplex, end-to-end BWIM-rendszer létrehozása volt, amely a hídon mért nyúlásjelek alapján önmagát optimalizálva képes meghatározni a járművek legfontosabb paramétereit, például a sebességet, tengelytávolságokat és tengelysúlyokat, anélkül, hogy ezekhez előzetesen megadott, manuálisan címkézett tanítóadatokra lenne szükség.

A koncepció lényege egy rekonstrukciós alapú tanulási folyamat volt. Ennek során a rendszer a szenzorok által mért eredeti jelfolyamot kapja bemenetként, majd több egymásra épülő almodul segítségével először megbecsüli a jármű főbb fizikai paramétereit, ezekből pedig újra alkotja (rekonstruálja) a várható szenzor jelet. Az így előállított rekonstruált jel és az eredeti mérési adat közötti eltérés szolgál tanulási visszacsatolásként. A háló paraméterei ezen a hibán keresztül frissülnek, tehát a rendszer önállóan, felügyelet nélküli (unsupervised) módon tanulja meg a jel és a járműparaméterek közötti összefüggéseket.

A koncepció megvalósításához három új modult fejlesztettem:

- a Járműparaméter-becslő modult,
- a Tengelysúlyszámító modult, valamint
- a Jel-rekonstruáló modult.

A járműparaméter-becslő modul célja az volt, hogy a tengelyszám-becslő eredménye és a szenzorjel alapján a háló képes legyen előállítani a jármű sebességét, tengelytávolságait és az időpontot, amikor az első tengely áthalad a szenzorsor felett. Ezen paraméterek segítségével a tengelysúlyszámító modul kiszámítja az egyes tengelyek terhelését, majd a rekonstruáló modul ezekből újra alkotja a teljes szenzor jelet. A tanulás során a rekonstruált és az eredeti jel közötti különbség adja a veszteségfüggvényt, amely alapján a rendszer képes lenne önmagát finom hangolni.

A fejlesztett modulok közül a Járműparaméter-becslő modul működőképes formában elkészült, és képes a sebesség, valamint a tengelytávolságok és áthaladási időpontok becslésére, ugyanakkor a pontossága még nem kielégítő, különösen a

tengelytávolságok és az első tengely áthaladási idejének meghatározásában, még a szimulált adatok esetében sem.

A tengelysúlyszámító modul implementációja is elkészült és képes a jármű tengelyeinek súly becslésére 2.73kN átlagos hibával. A számított tengelyterhelések megfelelnek a várakozásoknak viszont mivel a jármű paraméter becslő modulon alapul igy annak hibái itt is megjelennek és tovább rontják a rendszer hibáját.

A Jel-rekonstruáló modul ezzel szemben önmagában jól működik, képes a szenzorjelek rekonstruálására a megadott paraméterek alapján, ugyanakkor mivel a korábbi két modul még nem produkál kellően pontos bemenetet, önmagában a rekonstruált jel jelenleg nem hordoz érdemi információt a rendszer tanulása szempontjából.

Összességében ezek a kísérleti modulok az önálló tanulásra képes BWIM-rendszer irányába tett első, alapvető lépéseket képviselik. A fejlesztés végső célja egy olyan integrált architektúra létrehozása volt, amelyben a különálló komponensek, a tengelyszám-becslő, a járműparaméter-becslő, a tengelysúlyszámító és a jelrekonstrukciós modul, egymással összekapcsolva, a rekonstruált és a mért jel közötti hibát felhasználva képesek lennének önállóan, külső címkézés nélkül tanulni.

A diplomamunka során elkészült ennek az integrált rendszernek a működő prototípusa, amelyben a modulok közötti adatáramlás és a tanulási folyamat implementálva lett. Ugyanakkor a tanulás stabilitása és a konvergencia minősége még nem bizonyult kielégítőnek, a rendszer nem tudott megbízható módon javítani a saját becslésein, és az elért eredmények messze elmaradtak a várttól.

Ennek fő okai egyrészt az egyes almodulok korlátozott pontosságában, másrészt a tanulási folyamat komplexitásában keresendők, hiszen a rekonstruált jel hibáját visszavezetni a járműdinamikai paraméterekre igen összetett, nemlineáris problémát jelent. Mindazonáltal az elkészült integrációs kód és a kísérleti eredmények azt mutatják, hogy a koncepció megvalósítható, és további fejlesztéssel, reális lehetőség nyílik egy valóban önkalibráló BWIM-megoldás kialakítására, amely emberi beavatkozás nélkül képes tanulni a valós szenzoradatokból.

3.5.1 Járműparaméterbecslő modul

A járműparaméter-becslő modul a BWIM-rendszer egyik legösszetettebb és legfontosabb komponense, amely a járművek áthaladása során mért szenzorjelekből próbálja meg visszakövetkeztetni a jármű alapvető fizikai és mozgási paramétereit. A modul célja, hogy a tengelyszám-becslő modul kimenetét és a híd szenzorainak idősoros jelét felhasználva meghatározza a jármű sebességét, tengelytávolságait, valamint az időpillanatot, amikor az első tengely áthalad az elsődleges szenzorsor felett.

A járműparaméterek pontos ismerete alapvető feltétele a tengelysúlyok és a teljes járműterhelés meghatározásának, hiszen ezek az adatok szolgálnak alapul a híd válaszának modellezéséhez és a jelrekonstrukcióhoz. A modul tehát nem önállóan, hanem a teljes BWIM-rendszer integrált részeként működik, a cél pedig egy olyan architektúra kialakítása volt, amely a jármű mozgását leíró paramétereket kizárólag a híd nyúlásmérő szenzor által szolgáltatott jelei alapján képes meghatározni.

A modul fejlesztése során kiemelt szempont volt, hogy a modell képes legyen általánosítani különböző tengelyszámú járművekre. A hálózat kialakításakor ezért a korábban sikeresnek bizonyult CNN-alapú architektúrákat vettem alapul, de azok működését továbbfejlesztettem, hogy képes legyen komplex, többkimenetű paraméterbecslésre, és bizonytalanságkezelésre is. A modul tehát nem pusztán egy újabb becslőhálózat, hanem egyfajta központi összekötő elem, amely a jármű mozgását leíró paraméterek révén hidat képez a nyers szenzoradatok és a későbbi fizikai modellezés között.

3.5.1.1 Adatfeldolgozás és előkészítés

A járműparaméter-becslő modul bemeneti adatainak előállításakor a 3.2 fejezetben definiált általános előfeldolgozási láncolatot alkalmaztam. A bemeneti jelet ebben az esetben is a primer szenzorsor 16 érzékelőjének összegzésével állítottam elő, amely hatékonyan reprezentálja a híd globális válaszát a jármű áthaladásakor.

A neurális hálózat bemenetét így a járműablak kivágása, a Min-Max normalizálás, a középérték-eltolás, valamint a 200 mintapontos hosszra történő nullás kiegészítés (padding) képezi. Ezzel biztosítható, hogy a hálózat egységes formátumú és skálájú adatokat dolgozzon fel. A tanítási fázisban a modell általánosító képességének javítása és a valós mérések bizonytalanságának szimulálása érdekében a bemeneti jeleket 2%-os véletlenszerű zajjal terheltem.

3.5.1.2 Modellarchitektúra

A modul megvalósítása egy egy-dimenziós konvolúciós neurális hálózaton (1D CNN) alapul, amelyet kifejezetten az idősoros jelek feldolgozására és a járműdinamikai paraméterek regressziós becslésére terveztem. Az architektúra implementációját a VehParamCnn osztály tartalmazza, amely több egymásra épülő konvolúciós és FC rétegből áll.

A konvolúciós rész három reziduális blokkot tartalmaz (kernelméretek: 33, 23,11), melyek a bemeneti jel különböző időbeli léptékű jellemzőit képesek kinyerni. A reziduális kapcsolatok lehetővé teszik az információ hatékony átáramlását a háló mélyebb rétegeibe, csökkentve ezzel a gradiensvesztés problémáját. A konvolúciós blokkokat egy további ConvBlock1D réteg egészíti ki, amely a jellemzők finomítását és a reprezentációs szint emelését szolgálja.

A háló konvolúciós kimenetét lapítás után egy DenseNet típusú sűrű hálózatrész dolgozza fel, amely két rejtett rétegből áll (512 és 256 neuronnal). Az aktivációs függvény LeakyReLU, míg a dropout értéke 0.1, amely a túltanulás elkerülését szolgálja. A kimeneti réteg a jármű paramétereit, sebességet, tengelytávolságokat és első tengelyidőt, adja meg, továbbá minden becsléshez tartozik egy valószínűségi érték (q), amely azt mutatja, mennyire valószínű az adott becslés a háló belső reprezentációja szerint.

A modell kimenete négy komponensből áll:

- sebesség (speed) – a jármű átlagos áthaladási sebessége,
- tengelytávolságok (axle spacings) – az egymást követő tengelyek közötti távolság,
- első tengely áthaladási ideje (first axle time) – az az időpont, amikor az első tengely eléri az elsődleges szenzorsort,
- valószínűségi eloszlás (q) – az egyes becslések relatív megbízhatósága, amelyet softmax függvény segítségével számít a hálózat.

A kimeneti aktivációk skálázása biztosítja, hogy a becsült értékek fizikailag értelmezhető tartományban maradjanak:

- sebesség: 10–30 m/s,
- tengelytávolságok: 0,5–6,5 m,
- első tengelyidő: 0–3 s.

Ez a struktúra lehetővé teszi, hogy a háló ne csak numerikus becslést adjon, hanem a valószínűségi kimenet segítségével azt is jelezze, mely becslések a leginkább megbízhatók. Ez a tulajdonság különösen fontos, amikor a háló később önálló tanulási folyamatban vesz részt, például a jelrekonstrukciós modul visszacsatolásával.

3.5.1.3 Tengelyszám-függő működés

A járműparaméter-becslő modul egyik leglényegesebb eltérése a korábbi hálózatokhoz képest, hogy a működése közvetlenül függ a jármű tengelyszámától. Míg a tengelyszámbecslő és a sebességbecslő modulok egyetlen, közös hálózattal képesek voltak kezelni az összes járműtípust, itt a tengelytávolság-becslés miatt ez nem kivitelezhető. Egy háromtengelyes jármű esetében például két tengelytávolságot kell meghatározni, míg egy héttengelyes járműnél már hatot. A kimeneti dimenzió tehát dinamikusan változik, ami kizárja a közös háló használatát.

Ezt a problémát úgy oldottam meg, hogy minden tengelyszámhoz külön hálózati példányt hoztam létre. Így a 2–6 tengelyes járművekhez összesen nyolc külön VehParamCnn modell tartozik, és minden jármű esetében a rendszer automatikusan a megfelelő hálózatot választja ki a becsléshez.

További sajátosság, hogy a háló nem egyetlen paraméterkombinációt becsül, hanem nyolc párhuzamos becslést (num_pred = 8) generál, amelyekhez egy-egy valószínűségi értéket rendel. Ez lehetővé teszi a becslések közötti bizonytalanság modellezését, hiszen a háló egyszerre több lehetséges megoldást vizsgál, és a softmax függvény segítségével megállapítja, melyik a legvalószínűbb. Ennek köszönhetően a modell nem determinisztikus, hanem statisztikai értelemben is értékeli a járműparaméterek becslésének megbízhatóságát. Ez a megoldás különösen előnyös a későbbi, valós adatokon történő alkalmazásokban, ahol a szenzorjelek torzítása vagy zajossága miatt a determinisztikus becslés nem lenne elégséges.

3.5.1.4 Tanítási eljárás

A modell tanítását tengelyszámonként külön végeztem. Minden tengelyszámhoz 1000 darab szimulált járműmintát használtam fel, amelyek az adott tengelykonfigurációra jellemző jelalakokat tartalmazták. A tanítást 200 epochon keresztül végeztem, early stopping alkalmazásával, amely megakadályozta a túltanulást. Az optimalizáláshoz Adam algoritmust használtam, $lr = 10^{-3}$ tanulási ráta értékkel.

A tanulás során a háló stabil konvergenciát mutatott, a veszteség folyamatosan csökkent, a validációs hibák alakulása pedig azt jelezte, hogy a háló képes általános mintázatokat megtanulni, nem csupán a tanító adatokat reprodukálni. A hozzáadott zaj segítette a hálót abban, hogy robusztusabbá váljon a kisebb jelszint-ingadozásokkal és szenzorhibákkal szemben.

3.5.1.5 Eredmények és értékelés

A járműparaméter-becslő modul teljesítményét tengelyszámonként értékeltem, mivel a pontosság jelentősen függ a jármű konfigurációjától. A négytengelyes járművek esetében az átlagos hibák a következők voltak:

- sebességhiba: 0,77 m/s,
- tengelytávolság-hiba: 0,47 m,
- első tengely időzítési hiba: 0,06 s.

A hattengelyes járműveknél a hibák növekedtek:

- sebességhiba: 0,85 m/s
- tengelytávolság-hiba: 0,56 m
- első tengely időzítési hiba: 0,08 s

Ez a tendencia várható volt, hiszen a nagyobb tengelyszámú járművek esetében a nyúlásmérő szenzor által szolgáltatott jel hosszabb és komplexebb, ami több bizonytalanságot hordoz. Ennek ellenére a sebességbecslés stabil maradt, ami arra utal, hogy a háló jól tanulta meg a jármű mozgásának globális jellemzőit.

Az architektúra tesztelése során a Wu és társai által bemutatott architektúrát is vizsgáltam a 4 tengelyes és a 6 tengelyes jármű adatokon. Az négytengelyes járművek esetében az átlagos hibák:

- sebességhiba: 0,84m/s
- tengelytávolság-hiba: 0,51 m
- első tengely időzítési hiba: 0,025 s

A hattengelyes járműveknél a hibák:

- sebességhiba: 1,04 m/s

- tengelytávolság-hiba: 0,86 m
- első tengely időzítési hiba: 0,03 s

Annak ellenére, hogy az első tengely időzítési hibája kisseb a Wu és társai által bemutatott architektúrában, inkább a saját fejlesztett architektúrámat használtam az integráció során mivel a sebesség és a tengelytávolság hibája ott alacsonyabbak voltak.

Összességében a járműparaméter-becslő modul sikeresen demonstrálja, hogy a híd szenzorjeleiből és a tengelyszám-információból neurális hálózati úton előállíthatók a jármű dinamikai paraméterei. Bár a modell pontossága jelenleg még nem elegendő a teljesen önálló, felügyelet nélküli tanulási rendszerhez, az eredmények egyértelműen alátámasztják a CNN-alapú megközelítés hatékonyságát és tovább fejleszthetőségét.

3.5.2 Tengelysúly számító modul

A tengelysúlyszámító modul a BWIM-rendszer azon komponense, amely a jármű mozgási paraméterei alapján, a sebesség, a tengelytávolságok, az első tengely áthaladási ideje és az keresztirányú pozíció ismeretében, kiszámítja az egyes tengelyekhez tartozó függőleges terheléseket (tengelysúlyokat). A modul nem neurális hálózaton alapul, hanem fizikai modellezésen és lineáris algebrai inverzión, így a korábbi hálózati modulok által becsült paraméterekre épül, de maga determinisztikus módon működik.

A cél az volt, hogy a jármű mozgását leíró becsült paraméterekből rekonstruálható legyen a jármű által a hídra gyakorolt terhelés eloszlása, és ezáltal meghatározhatóvá váljon az egyes tengelyek súlya kN-ban. A modul a BWIM-rendszer kulcseleme, hiszen ez biztosítja a kapcsolatot a neurális modellek által becsült járműparaméterek és a tényleges, mérhető fizikai mennyiségek között.

3.5.2.1 Működési elv és cél

A tengelysúlyok számítása azon az elven alapul, hogy a hídon elhelyezett szenzorok által mért jel a jármű tengelyeinek terheléséből származik. A híd válaszát egy befolyásfelület (influence surface) írja le, amely megmutatja, hogy a híd egy adott pontján mért deformáció hogyan függ egy adott pozícióban elhelyezkedő terheléstől. A híd teljes válasza tehát az összes tengely által keltett hatás szuperpozíciójaként írható fel:

$$s(t) = \sum_{i=1}^{N_{axle}} I\big(x_i, y_i(t)\big) * P_i$$

ahol:

- $s(t)$ a mért jel,
- $I\big(x_i, y_i(t)\big)$ az i-edik tengelyhez tartozó befolyásfelület értéke a $\big(x_i, y_i(t)\big)$ pozícióban,
- P_i az i-edik tengely által kifejtett tengelysúly (kN),
- $y_i(t)$ a tengely pillanatnyi hosszirányú pozíciója a hídon az idő függvényében.

A modul feladata ezen egyenlet „megfordítása", azaz a mért $s(t)$ jel és az ismert mozgási paraméterek (sebesség, tengelytávok, első tengelyidő, x-pozíció) alapján visszaszámítja az ismeretlen tengelysúlyokat (P_i)

3.5.2.2 Matematikai háttér és számítási folyamat

A számítás alapja az időbeli és térbeli kapcsolat meghatározása a tengelyek és a mérési jel között. A jármű sebessége (v) és a tengelyek közötti távolságok (d_i) segítségével kiszámíthatók a tengelyek áthaladási időpontjai a szenzor fölött:

$$\Delta t_i = \frac{d_i}{v}$$

$$t_i = t_{first} + \sum_{j=1}^{i-1} \Delta t_j$$

ahol:

- t_{first} az első tengely szenzor feletti áthaladási ideje,
- t_i az i-edik tengely időpillanata, amikor az a szenzorsor fölött van.

Ezután a jármű mozgásának időbeli leképezése segítségével meghatározható a tengelyek hossz-irányú pozíciója a hídon:

$$y_i(t) = v * (t - t_i)$$

A fenti képlet megmutatja, hogy az idő minden pillanatában mekkora az

i-edik tengely és az adott szenzor közötti távolság. Mivel egysávos forgalmi viszonyokat feltételezünk, az x irányú eltolás fixen 1,5 méter (a sáv középvonala), így $x_i = 1.5m$ állandónak vehető.

Ezek után a modul kiszámítja a befolyásmátrixot I, amely tartalmazza a híd válaszát minden tengely és időpillanat kombinációjára. Az I mátrix dimenziója $[N_{signal}, N_{axle}]$, ahol minden oszlop egy-egy tengely hatását mutatja a teljes jelfolyamra.

A mérési egyenlet mátrixalakban így írható fel:

$$s = I * P$$

ahol:

- s a mért jel vektora ($[N_{axle} \times 1]$),
- I a befolyásmátrix ($[N_{signal} \times N_{axle}]$),
- P pedig az ismeretlen tengelysúly-vektor ($[N_{axle} \times 1]$),

A cél a P meghatározása. Mivel az I mátrix általában nem négyzetes, hanem túldefiniált, a rendszer legkisebb négyzetes értelemben vett inverzét kell alkalmazni. Ezt a pszeudoinverz (I^+) segítségével számítjuk ki:

$$P = ReLU(I^+ * s)$$

A ReLU függvény biztosítja, hogy a negatív értékek (ami fizikailag értelmezhetetlen, mivel a tengelysúly nem lehet negatív) lenullázódjanak. Az eredményül kapott **P** vektor a tengelyekhez tartozó becsült súlyokat tartalmazza kN-ban.

A számítás egy olyan algoritmussal történik, amely a híd befolyásfelületeinek adatait használja fel. Ezek a befolyásfelületek írják le, hogy a híd adott pontján mért deformáció hogyan függ egy meghatározott helyen ható terheléstől. A szenzorsor minden érzékelőjéhez tartozik egy ilyen befolyásfelület, azonban a számítás során ezek nem külön-külön, hanem összegezve, egy egységesített befolyásmátrix formájában kerülnek felhasználásra. Ez a megközelítés lehetővé teszi, hogy a teljes szenzorsor válaszát egyetlen szuperponált hatásként kezeljük, ami jelentősen leegyszerűsíti a számítási folyamatot és csökkenti a hibák felhalmozódásának lehetőségét.

A számítás lépései a következők:

1. A sebesség és tengelytávok alapján meghatározódnak a tengelyek időbeli eltolásai.
2. Az időbeli eltolásokból kiszámítódnak a tengelyek $y_i(t)$ pozíciói.
3. A fix $x = 1.5m$ pozícióval együtt az (x, y) koordináták alapján lekérdezzük a híd befolyásfelületét, és felépítjük az I befolyásmátrixot.
4. Az I mátrix és a mért jel $s(t)$ felhasználásával kiszámítjuk a tengelysúly-vektort a pszeudoinverz segítségével: $P = I^+ * s$
5. A negatív értékek kizárására ReLU aktivációt alkalmazunk, így a végső tengelysúly-vektor $P \geq 0$

A modul eredményeként minden járműhöz, illetve, ha több becslés (num_pred) érkezik a járműparaméter-becslő modultól, akkor minden predikcióhoz külön, egy tengelysúly-vektor adódik, amely kN-ban tartalmazza az egyes tengelyek terhelését.

3.5.2.3 Értékelés és jelenlegi állapot

A tengelysúlyszámító modul működőképes formában elkészült, és képes a járműdinamikai paraméterekből, valamint a híd aljára szerelt nyúlásmérő szenzorok által biztosított jelekből fizikailag értelmezhető tengelysúlyokat előállítani. A modul tesztelését úgy végeztem el, hogy minden járműnek a tényleges járműparamétereit használtam bemenetként, biztosítva ezzel, hogy az eredmények összehasonlíthatóak legyenek a tényleges tengelysúlyokkal. A használt járműparaméterek a következők voltak: a jármű sebessége, a tengelytáv, az első tengely áthaladási ideje az elsődleges szenzorsor felett, valamint a jármű keresztirányú pozíciója.

A modul teljesítményét 8000 szimulált járművön teszteltem, amelyek egysávos, non-convoy és non-tandem konfigurációban szerepeltek. Az átlagos számítási hiba 2,73 kN volt, ami a járművek teljes tömegéhez és a tengelyek terheléséhez viszonyítva elfogadható pontosságnak tekinthető. A tesztek során egyértelműen megfigyelhető volt, hogy a modul nagyobb tengelyszámú járművek esetén pontosabban számolt, míg az alacsonyabb tengelyszámú járműveknél az eltérés valamivel nagyobb volt.

A bemeneti szenzor jelek feldolgozása során csak azokat a jeleket vettem figyelembe, amelyekhez a jármű ablaka legfeljebb 200 hosszú volt. Emellett a számításokhoz elég volt az elsődleges szenzor sor összegzett értékeit használni.

Jelenlegi állapotában a modul megbízhatóan reprodukálja a számítási folyamatot, és biztos alapot nyújt a későbbi önkalibráló BWIM-rendszerbe történő integrációhoz.

3.5.3 Jel Rekonstruáló modul

A jelrekonstrukciós modul a BWIM-rendszer végső lépése, amely a korábbi modulok által becsült járműparaméterek (sebesség, tengelytávolság, első tengelyidő, pozíció) és a tengelysúlyok alapján újra előállítja a szenzorok által mért jelet. A modul célja, hogy a becsült fizikai paraméterekből reprodukálja a mért jelalakot, és ezáltal lehetőséget teremtsen az egész rendszer önellenőrzésére és önkalibrálására. A rekonstrukció és a valós mérési jel közötti különbség a későbbiekben a tanulási folyamat hibafüggvényeként szolgálhat, lehetővé téve, hogy a teljes BWIM-rendszer a tényleges fizikai jelhez igazodva finomítsa saját becsléseit.

3.5.3.1 Működési elv és cél

A modul működésének alapját az a fizikai elv képezi, hogy a híd deformációs válasza az egyes tengelyek által kifejtett terhelések szuperpozíciójaként írható le. Amennyiben a tengelyek helyzete és a hozzájuk tartozó terhelés ismert, a hídon kialakuló jelalak egyértelműen meghatározható. Ezt a folyamatot a következő összefüggés írja le:

$$\hat{s}(t) = \sum_{i=1}^{N_{axle}} I\big(x_i, y_i(t)\big) * \widehat{P_i}$$

ahol:

- $\hat{s}(t)$ a rekonstruált jel,
- $I\big(x_i, y_i(t)\big)$ az i-edik tengelyhez tartozó befolyásfüggvény a híd geometriájában,
- $\widehat{P_i}$ a tengelysúlybecslésből származó terhelés,
- $\big(x_i, y_i(t)\big)$ a tengelyek pozíciói az idő függvényében.

A rekonstruált jel tehát azt mutatja meg, hogy a híd milyen választ adna a becsült paraméterekkel jellemzett jármű áthaladásakor. A rekonstruált és a mért jel közötti eltérés kvantitatívan kifejezi, hogy mennyire pontosak a korábbi modulok (tengelyszám-, sebesség- és tengelysúlybecslés) eredményei.

3.5.3.2 Rekonstrukciós folyamat matematikai leírása

A jelrekonstrukció lényegében a tengelysúlyszámítás fordított művelete.

A híd deformációját leíró egyenlet mátrixalakban a következő formát ölti:

$$s = I * P$$

ahol:

- s a mért jel,
- I a befolyásmátrix, amely megmutatja, hogy az egyes tengelyek mekkora hatást fejtenek ki a híd különböző pontjain,
- P a tengelysúly-vektor.

A jelrekonstrukció során az előző egyenletet használjuk. A becsült tengelysúly-vektorból ($\widehat{P_t}$) és a hozzá tartozó befolyásmátrixból (I) kiszámítjuk a rekonstruált jelet:

$$\hat{s} = I * \hat{P}$$

Ez a szorzás adja meg a rekonstruált jelfolyamot minden szenzor és minden időpillanat esetében. A gyakorlatban a rekonstrukció szenzoronként párhuzamosan történik, mivel a szenzorjelek azonos szerkezetű, de különböző intenzitású válaszokat képviselnek. Az így előállított $\hat{s}$ jel közvetlenül összevethető a mért s jellel, és a kettő különbsége szolgál a modell tanításának hibajelként:

$$\mathcal{L} = \|s - \hat{s}\|_2^2$$

A rekonstrukció tehát nemcsak validációs eszköz, hanem a teljes rendszer tanulási mechanizmusának is alapját képezi.

3.5.3.3 Működési módok

A modul kétféle üzemmódban képes működni, a felhasználás céljától függően:

1. Tanítási mód:

   Ebben az üzemmódban a modul minden rendelkezésre álló járműparaméter- és tengelysúlybecslés kombinációra elvégzi a jelrekonstrukciót. Mivel a járműparaméter-becslő és a tengelysúlybecslő modulok egyaránt nyolc különböző

becslést adnak (mindegyikhez tartozik egy valószínűségi érték is), a jelrekonstrukció ezeket mind figyelembe veszi, és nyolc különböző rekonstruált jelfolyamot hoz létre. Ez lehetővé teszi, hogy a tanítás során a hálózat a teljes becslési térből tanuljon, ne csak a legjobb eredményt vegye figyelembe. Ennek köszönhetően robusztusabb, általánosabban alkalmazható rendszer hozható létre.

2. Értékelési (inference) mód:

   Ebben az üzemmódban már nem a tanulás, hanem a validálás a cél, így a modul a rendelkezésre álló becslések közül kiválasztja a legvalószínűbbet (azaz a legjobb modellt), és ezzel hajtja végre a jelrekonstrukciót. Ennek eredményeként egyetlen, a legjobb paraméterkombinációhoz tartozó jelalak keletkezik, amely közvetlenül összevethető a mért jellel. Ez a mód a teljes rendszer gyakorlati alkalmazásakor, például valós hídmérések kiértékelésekor használható.

3.5.3.4 Implementációs megvalósítás és jövőbeli szerep

A modul a híd válaszának szimulációjára épül, és a korábbi lépésekben becsült fizikai paramétereket használja fel a jel előállításához. A befolyásmátrix (I) minden tengelyhez és időponthoz kiszámítja, hogy az adott terhelés hogyan befolyásolja a híd alakváltozását, majd a tengelysúly-vektorral (P) történő szorzással előállítja a rekonstruált jelet.

A két üzemmód különbsége abban rejlik, hogy tanítás közben a teljes becsléskészletet, míg értékelés során kizárólag a legvalószínűbb becsléseket használja fel.

A modul jelenleg stabilan működik, és pontosan végrehajtja a rekonstrukciós műveletet. Ugyanakkor a gyakorlati haszna önmagában korlátozott, mivel a működése erősen függ a korábbi modulok, különösen a járműparaméter- és tengelysúlybecslő, pontosságától. A jelrekonstrukciós modul igazi értéke abban rejlik, hogy a teljes BWIM-rendszer záróelemeként lehetővé teszi egy olyan önfelügyelt tanulási folyamat kialakítását, amely a rekonstruált és a mért jel közötti eltérés alapján képes finomítani saját paramétereit.

A későbbi fejlesztések célja a teljes pipeline integrálása, ahol a jelrekonstrukció hibája visszacsatolásként szolgálhat a járműparaméterek és tengelysúlyok becslésének javításához, így megteremtve az alapot egy valóban önkalibráló neurális BWIM-rendszer megvalósításához.

4 Összegzés

A Diplomamunkám célja a BWIM rendszerek működésének megismertetése és olyan mélytanuláson alapuló algoritmusok fejlesztése volt, amelyek képesek a hídra szerelt nyúlásmérő szenzorok által szolgáltatott jelekből a hídon áthaladó járművek paramétereinek becslésére, illetve a Wu és társai által fejlesztett BwimNet architektúra reprodukálása és kiértékelése.

A diplomamunkám során elkészítettem három önálló, konvolúciós neurális hálózaton (CNN) alapuló modult, amelyek a nyúlásmérő szenzorok által biztosított jelből a jármű különböző paramétereit becslik:

1. Keresztirányú pozícióbecslő: A 3.1-es fejezet ismerteti bővebben. A becslés átlagos abszolút hibája 66,23 mm volt.
2. Tengelyszámbecslő: A 3.3-es fejezet ismerteti. A becslés pontossága szintetikus adatokon 92,60%, míg valós adatokon csak 43,96% volt.
3. Sebességbecslő: A 3.4-as fejezet tárgyalja. A becslés átlagos abszolút hibája szintetikus adatokon 0,56 m/s, míg valós adatokon 1,57 m/s volt.

Ezen modulok felépítését és eredményeinek kiértékelését a releváns fejezetek bemutatják.

Az önálló modulok sikeres implementálása után a Wu és társai által fejlesztett BwimNet architektúra reprodukálásának érdekében további három modult hoztam létre, amelyek rendszer integrált működéséhez szükségesek:

1. Járműparaméter becslő: (3.5.1 fejezet) Ez a modul a tengelyszám és a nyúlásmérő szenzorok által szolgáltatott jelből becsli a sebességet, tengelytávolságot és az időpontot amikor az első tengely áthaladt a szenzorsor felett. A becslések pontossága tengelyszám függő.
2. Tengelysúly számító: (3.5.2 fejezet) A modul a járműparaméter-becslő eredményeire építve határozza meg a tengelyterheléseket, így a kimeneti hiba közvetlenül függ az előző lépés pontosságától.
3. Jel Rekonstruáló: (3.5.3 fejezet) A modul feladata a szenzorjel visszaállítása a becsült paraméterek alapján és a Tengelysúly számító modulhoz hasonlóan az előző modulok hibájától függ a pontossága.

A modulok elkészítése után reprodukáltam a teljes BwimNet rendszert és teszteltem azt szintetikus adatokon. A koncepció célja egy olyan önfelügyelt tanulási folyamat létrehozása volt, ahol a tanításhoz elegendő a nyers szenzorjel. Bár a modulok integrációja és az adatáramlás technikailag helyes, a tesztek során azt tapasztaltam, hogy a rendszer nem képes a stabil konvergenciára. Ennek oka vélhetően a részmodulok (különösen a paraméterbecslő) bizonytalanságaiból adódó hibaterjedés, amely megakadályozza a hatékony tanulást.

A konkrét fejlesztési eredményeken és a tapasztalt korlátok feltárásán túl a diplomamunka elkészítése jelentős személyes szakmai fejlődést is hozott. A fejlesztés kezdetén programozói ismereteim főként kisebb léptékű, izolált feladatokra (például rövid szkriptek írására) korlátozódtak. A kitűzött feladat azonban egy komplex, több modulból álló szoftverrendszer felépítését igényelte, amihez elengedhetetlen volt a Python nyelv és a modern fejlesztési elvek mélyebb elsajátítása. A munka során így nemcsak a mélytanulási algoritmusokat ismertem meg, hanem gyakorlati rutint szereztem a moduláris rendszertervezésben és a PyTorch keretrendszer alkalmazásában is. A projekt sikeres megvalósítása számomra a mérnöki szemléletmód és a kódolási kompetenciák terén is jelentős előrelépést jelentett.

Köszönet nyilvánítás

Ezúton szeretnék köszönetet mondani a Komáromi Monostori híd valós mérési adatainak felvételében nyújtott segítségéért a Magyar Közút Nonprofit Zrt.-nek.

A Doktoranduszi Kiválósági Ösztöndíj Program (DKÖP) által támogatott projekt a Kulturális és Innovációs Minisztérium Nemzeti Kutatási Fejlesztési és Innovációs Alapból nyújtott, valamint a Budapesti Műszaki és Gazdaságtudományi Egyetem támogatása alapján valósult meg.

Végezetül hálás szívvel mondok köszönetet szüleimnek és családomnak, akik mindvégig támogattak tanulmányaimban, és biztosították számomra a stabil hátteret a diplomamunka elkészítéséhez.

5 Irodalomjegyzék

[1] M. Zhang, X. Wang és Y. Li, „Fatigue Reliability Assessment of Bridges Under Heavy Traffic Loading Scenario,” *Infrastructures,* %1. kötet9, %1. szám12, p. 238, 2024.

[2] A. Žnidarič és J. Kalin, „Using bridge weigh-in-motion systems to monitor single-span bridge influence lines,” *International Journal of Advanced Structural Engineering,* %1. kötet12, p. 341–354, 2020.

[3] Yuhan Wu, Lu Deng és Wei He, „BwimNet: A Novel Method for Identifying Moving Vehicles Utilizing a Modified Encoder-Decoder Architecture,” *Sensors,* 14 December 2020.

[4] N. B. Dohn, Y. Kafai, A. Mørch és M. Ragni, „Survey: Artificial Intelligence, Computational Thinking and Learning,” *KI - Künstliche Intelligenz,* %1. kötet36, p. 5–16, 2022.

[5] S. Achuta Rao, K. Kondaiah, G. Rajesh Chandra és K. Kiran Kumar, „A Survey on Machine Learning: Concept, Algorithms and Applications,” *International Conference on Innovative Research in Computer and Communication Engineering,* p. 1301, február 2017.

[6] F. Emmert-Streib, Z. Yang, H. Feng, S. Tripathi és M. Dehmer, „An Introductory Review of Deep Learning for Prediction Models With Big Data,” *Frontiers in Artificial Intelligence,* %1. kötet3, %1. szám4, 2020.

[7] Z. Li, F. Liu, W. Yang, S. Peng és J. Zhou, „A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects,” *IEEE,* %1. kötet33, %1. szám12, 2021.

[8] B. Liquet, S. Moka és Y. Nazarathy, Mathematical Engineering of Deep Learning, New York: Chapman and Hall/CRC, 2024.

[9] C. Janiesch, P. Zschech és K. Heinrich, „Machine learning and deep learning,” *Electronic Markets,* %1. kötet31, p. 685–695, 2021.

[10] I. G. a. Y. B. a. A. Courville, Deep Learning, MIT Press, 2016.

[11] E. Csató, *Tengelysúlybecslés gépi tanulással,* BME, 2022.

[12] M. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

[13] C. C. Aggarwal, Neural Networks and Deep Learning, Springer, 2018.

[14] S. S. R. D. V. P. S. M. Shabbeer Basha, „Impact of fully connected layers on performance of convolutional neural networks for image classification," *Neurocomputing,* %1. kötet378, pp. 112-119, 2020.

[15] M. Lyndon, S. E. Taylor, D. Robinson, A. Mufti és E. J. O'Brien, „Recent Developments in Bridge Weigh-In-Motion (B-WIM)," in *Springer*, 2016.

[16] R. Ma, Z. Zhang, Y. Dong és Y. Pan, „Deep Learning Based Vehicle Detection and Classification Methodology Using Strain Sensors under Bridge Deck," *sensors,* %1. kötet20, %1. szám18, p. 5051, 2020.

[17] X. Jian, Y. Xia, S. Sun és L. Sun, „Integrating bridge influence surface and computer vision for bridge weigh-in-motion in complicated traffic scenarios," *Structural Control and Health Monitoring,* %1. kötet29, %1. szám11, p. e3066, 2022.

[18] T. Kawakatsu, K. Aihara, A. Takasu és J. Adachi, „Fully-neural approach to heavy vehicle detection on bridges using a single strain sensor," in *IEEE*, 2020.

[19] Bence Szinyéri, Bence Kővári, István Völgyi, Dénes Kollár és Attila László Joó, *A strain gauge-based Bridge Weigh-In-Motion system using deep learning,* BME, 2023.

[20] S. Ren, K. He, R. Girshick és J. Sun, „Faster R-CNN: Towards real-time object detection with region proposal networks," in *Curran Associates, Inc.*, 2015.

[21] A. Lansdell, W. Song és B. Dixon, „Development and testing of a bridge weigh-in-motion method considering nonconstant vehicle speed," *Engineering Structures,* %1. kötet152, p. 709–726, 2017.

[22] L. Deng, W. He, Y. Yu és C. Cai, „Equivalent shear force method for detecting the speed and axles of moving vehicles on bridges," *Journal of Bridge Engineering,* %1. kötet23, %1. szám8, p. 04018057, 2018.

[23] W. He, L. Deng, H. Shi, C. Cai és Y. Yu, „Novel virtual simply supported beam method for detecting the speed and axles of moving vehicles on bridges," *Journal of Bridge Engineering,* %1. kötet22, %1. szám4, p. 04016141, 2017.

[24] H. Kalhori, M. M. Alamdari, X. Zhu, B. Samali és S. Mustapha, „Non-intrusive schemes for speed and axle identification in bridge-weigh-in-motion systems," *Measurement Science and Technology,* %1. kötet28, %1. szám2, p. 025102, 2017.

[25] S.-Z. Chen, G. Wu és D.-C. Feng, „Development of a bridge weigh-in-motion method considering the presence of multiple vehicles," in *Elsevier*, 2019.

[26] Eugene J. OBrien, Donya Hajializade, Nasim Uddin, D. Robinson és R. Opitz, „STRATEGIES FOR AXLE DETECTION IN BRIDGE WEIGH-IN-MOTION SYSTEMS.," 2012.

[27] S. R. Lorenzen, H. Riedel, M. M. Rupp, L. Schmeiser, H. Berthold, A. Firus és J. Schneider, „Virtual axle detector based on analysis of bridge acceleration measurements by fully convolutional network," *Sensors,* %1. kötet22, %1. szám22, p. 8963, 2022.

[28] W. He, T. Ling, E. J. O'Brien és L. Deng, „Virtual axle method for bridge weigh-in-motion systems requiring no axle detector," *Journal of Bridge Engineering,* %1. kötet24, %1. szám9, p. 04019086, 2019.

[29] E. J. O'Brien, C. W. Rowley, A. Gonzalez és M. F. Green, „A regularised solution to the bridge weigh-in-motion equations," *International Journal of Heavy Vehicle Systems,* %1. kötet16, %1. szám3, p. 310–327, 2009.

[30] F. Moses, „Weigh-in-motion system using instrumented bridges," *Transportation Engineering Journal of ASCE,* %1. kötet105, %1. szám3, p. 233–249, 1979.

[31] H. Zhao, N. Uddin, E. J. O'Brien, X. Shao és P. Zhu, „Identification of vehicular axle weights with a bridge weigh-in-motion system considering transverse

distribution of wheel loads," *Journal of Bridge Engineering,* %1. kötet19, %1. szám3, p. 04013008, 2014.

[32] E. J. O'Brien, L. Zhang, H. Zhao és D. Hajializadeh, „Probabilistic bridge weigh-in-motion," *Canadian Journal of Civil Engineering,* %1. kötet45, %1. szám8, p. 667–675, 2018.

[33] Y. Yu, C. Cai és L. Deng, „Nothing-on-road bridge weigh-in-motion considering the transverse position of the vehicle," *Structure and Infrastructure Engineering,* %1. kötet14, %1. szám8, p. 1108–1122, 2018.

[34] Son, Seho & Jeong, Jinho & Jeong, Dayeon & Sun, Kyung & Oh, Ki-Yong, Physics-informed neural network: principles and applications, IntechOpen, 2024.

[35] D. K. &. J. Lee, „A Review of Physics Informed Neural Networks for Multiscale Analysis and Inverse Problems," *Multiscale Science and Engineering ,* %1. kötet6, pp. 1-11, 2024.

6 Ábrajegyzék

Hallgatói nyilatkozat
generatív mesterséges intelligencia alkalmazásáról

☐ **Nem használtam** semmilyen generatív MI segédeszközt.

☒ **Használtam** generatív MI segédeszközt. Az MI-vel generált tartalmakat ellenőriztem, a generált kimenetek valóságtartalmáról meggyőződtem, az alábbi táblázatban megfelelően jelöltem minden használatot.

Felhasználási módok	**Generatív MI eszköz(ök) neve**	**Érintett részek** (fejezet, oldalszám, hivatkozás)	**Használat becsült aránya** (felhasználási módonként)
Irodalomkutatás	ChatGPT, Gemini, Grok	Szakirodalom kutatás 2. fejezet (10.-25. oldal)	10%
Prompt lényegi része	Általam generált szöveg stilisztikai átdolgozása: „Megírtam az alábbi szöveget, ami tartalmazza az infókat a pl. BWIM el kapcsolatban, fogalmazd át nekem egy tudományosabb szöveggé anélkül, hogy a tartalmi lényegen változtatnál vagy újat adj hozzá.”		
Programkód generálása	GitHub Copilot, Gemini, Grok, ChatGPT	A 3. fejezetben (28.-63. oldal) bemutatott modulok vázának elkészítéséhez segítség	15%
Prompt lényegi része	Az adott általam kigondolt feladat leírása és ahhoz tartozó kód vázának elkészítése_ pl: „Segíts egy olyan python fügvény elkészítésében, amely a bementi [..] dimenziójú adatot, padingel, cropol.. stb” Egy már megirt CNN architekturát mintának használva egy új architektúra vázának gyors elkészítése pl: „Az kód alapján, amit megadtam készíts nekem egy Python nn.Module architektúrát, aminek x db Conv1D stb. layere van és ahol a kernel size, dropout, ..stb. egy adott értéken van.”		
Új ötletek, megoldási javaslatok generálása	GitHub Copilot, Gemini, Grok, ChatGPT	Neurális háló architektúrájához javaslatok	0%
Prompt lényegi része	Különböző gyengén teljesítő CNN-ek architektúrájának javítására ötletek: „Itt egy kód az általam elkészített CNN architektúrájához, de az eredmények még nem elég pontosak, tudnál javasolni változtatásokat/technikákat, amik növelnék a pontosságot?” (Használható ötletet nem kaptam, én kódomat próbálta módosítani és akkor is csak rontott rajta)		
Vázlat létrehozása (szövegstruktúra, vázlatpontok)	-	-	-
Prompt lényegi része			
Szövegblokkok létrehozása	Gemini, Grok, ChatGPT	Bevezetés (1.fejezet) Saját megvalósítás (3.fejezet) Összegzés (4.fejezet)	10%
Prompt lényegi része	Az általam megint szöveg és modulok kódja alapján segítsen átdolgozni a szövegemet, hogy tudományosabb legyen: „Megírtam mindent, amit tudni kell a pl. Sebesség becslő fejezetről, át tudod ezt nekem fogalmazni tudományosra és jól olvashatóra anélkül, hogy infót elvennél vagy hozzáadnál?”		
Képek generálása illusztrációs célból	-	-	-
Prompt lényegi része			

Adatvizualizáció, grafikonok generálása adatpontok alapján	-	-	-
Prompt lényegi része			
Prezentáció készítése	-	-	-
Prompt lényegi része			
Egyéb (nevezze meg)	-	-	-
Prompt lényegi része			
Összesített százalékos érték (a feladat érdemi részére nézve)			20%

Összesített érték rövid, szöveges indoklása:
A diplomamunka tartalmi részét, szakmai koncepcióját és az abban szereplő információkat önállóan alkottam meg. A generatív MI eszközöket (ChatGPT, Gemini) kizárólag a már általam megírt szövegrészek stilisztikai javítására, valamint tudományos nyelvezetű átfogalmazására használtam, új gondolatok vagy tényanyag generálására nem. Minden MI által generált szöveget átolvastam és szükség szerint javítottam/módosítottam.

A szoftverfejlesztés során több eszközt (GitHub Copilot, Gemini, Grok, ChatGPT) is igénybe vettem részfeladatok kódvázának generálására. Mivel az így kapott kódok önmagukban jellemzően nem voltak működőképesek, azok hibajavítása, átdolgozása és a rendszerbe való integrálása jelentős mértékű önálló mérnöki munkát igényelt. Az MI így csupán a fejlesztés technikai alapjait gyorsította, de a végleges, működő algoritmusok és a rendszerarchitektúra a saját munkám eredménye.

Figyelem: a fenti százalékértékektől függetlenül az MI-vel generált minden tartalomért a szerző a felelős!