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

Klaffl Attila András

MÉLYTANULÁSI ALGORITMUSOK (DEEP LEARNING) ALKALMAZÁSA A GYAKORLATBAN

KONZULENS

Szinyéri Bence

BUDAPEST, 2025

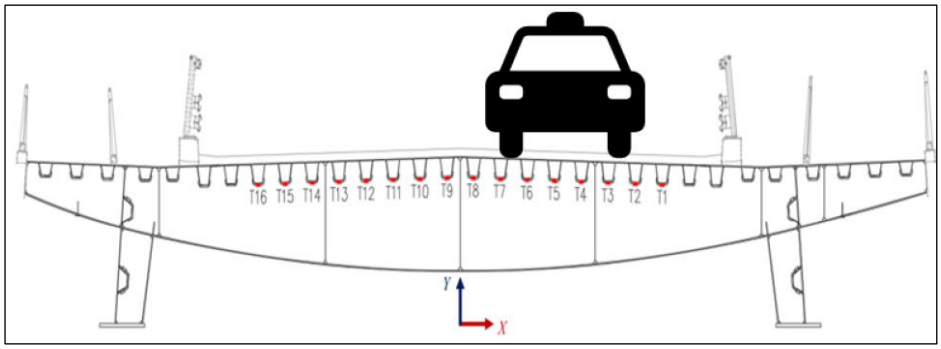
Tartalomjegyzék

Bevezetés

A közúti járművek tengelyszámának és tengelyterhelésének, valamint a hidak terhelésének meghatározásához jellemzően különféle szenzorok állnak rendelkezésre. Egyik elterjedt módszer a hidakra épített nyúlásmérőbélyegek alkalmazása. A mérési elrendezés központi elemei a híd alsó szerkezetére elhelyezett nyúlásmérő szenzorok, amelyeket az *1. ábra* is szemléltet. A híd kettő keresztmetszetében összesen 26 szenzor található egyenletes eloszlásban. A járművek súlya által okozott nyúlásból előállítható egy idő-deformáció görbe. A valós körülmények között azonban gyakran előfordul, hogy több jármű halad át egyszerre a hídon, egymással szemben, a különböző sávokban, ami jelentősen megnehezíti az egyes járművekhez tartozó szenzorjelek elkülönítését és értelmezését.

A feladat célja egy olyan mélytanulási módszer bemutatása, amely képes a szenzoradatokból kinyerni az egyes járművekhez tartozó jelkomponenseket, ha a mért jelek összeolvadnak. A megközelítés egydimenziós konvolúciós neurális hálózatot alkalmaz, amely képes az időbeli jellemzők alapján azonosítani az egyik jelkomponenst.

A rendszer része egy adathalmaz készítő és feldolgozó szkript és egy egyedi veszteségfüggvény, amely egyszerre igyekszik biztosítani a jelek pontosságát és az időbeli trendek megőrzését. A dokumentáció részletesen bemutatja az adathalmaz létrehozásának módszerét, a hálózat architektúráját, az alkalmazott eljárásokat, valamint az elért eredményeket is.

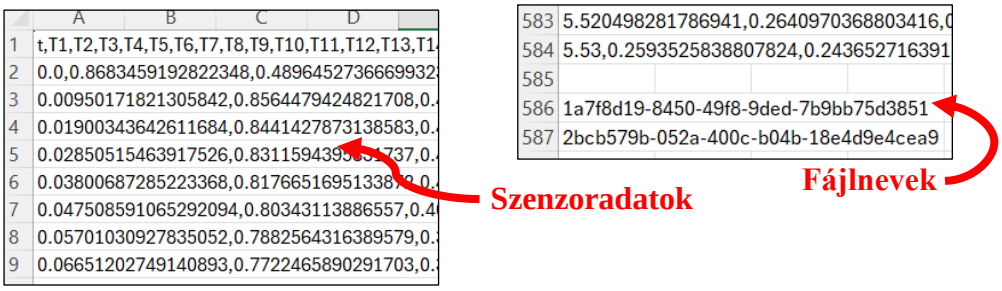


1. ábra Nyúlásmérő bélyegek elhelyezkedése. *Forrás: Szinyéri B., Csató E. „Mélytanuló algoritmusok alkalmazása Bridge Weigh-in-Motion rendszerekben", Budapesti Műszaki és Gazdaságtudományi Egyetem,* 2023.

Adathalmaz

A tanításhoz felhasznált adathalmaz létrehozásához, valamint az előfeldolgozáshoz külön Python szkriptet készítettem, amely automatizált módon végzi a szenzorjelek összevonását és mentését. A szkript célja, hogy a különböző sávokban elhaladó járművek által generált jelek egyesítésével olyan adathalmazt hozzon létre, mely megfelelően képes szimulálni azt az esetet, amikor a két jármű egyidőben halad el egymás mellett. Ez már képes egy jelszétválasztó neurális hálózat tanítására.

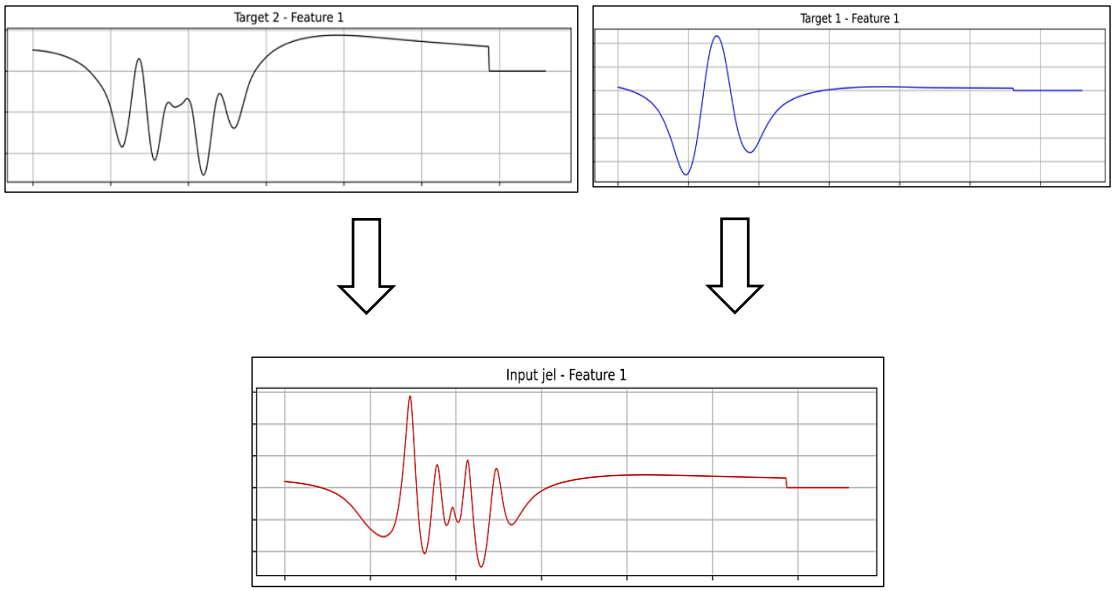
A már rendelkezésre álló adathalmazban található fájlok, az egyes sávokban elhaladó járművek által generált szenzorjeleket tartalmazzák. A metódus a két sávból (HU – SK, SK-HU) véletlenszerűen kiválasztott jeleket dolgozza fel párosával, és kombinálja azokat egyetlen jellé. A fájlok beolvasását követően először közös időskálára kell hozni a jeleket, mivel azok jellemzően különböző időpontokban kerültek rögzítésre. Természetesen nem garantálható, hogy az adatsorok azonos hosszúságúak. Ezeket a problémákat a program lineáris interpoláció segítségével oldja meg. Először a két jelet relatív időskálára hozza, azaz mintha mindkettő mérést azonos időpontban indítottunk volna, majd egy sűrűn mintavételezett időtengelyre interpolálja őket. Immáron miután mindegyik adatsor nulla idővel kezdődik és azonos hosszúságúak, az egyes szenzorok jelei egyszerűen összeadásra kerülnek. Az adatok szintén egy CSV-fájlban kerülnek mentésre, melynek egyes oszlopai a 26 db szenzorhoz tartozó értékeket *(T1-T16, T201-205 és T212-T216)* tartalmazza. A fájl utolsó két sorában szöveges formában elmentésre kerülnek továbbá a jelkomponensekhez tartozó fájlok nevei. Ezt mutatja a *2. ábra* is.



2. ábra Adathalmaz CSV-fájl

Ezutóbbi különösen fontos, hiszen a háló tanítása során szükséges van az eredeti jelkomponensek ismeretére, hogy a modell által kiadott jelet össze lehessen hasonlítani az eredetivel.

Az adathalmazok létrehozásával és azok kezelésével a *merge.py*, illetve a *dataLoader.py* szkriptek foglalkoznak. A szenzoradatok létrehozása során a program véletlenszerűen kiválaszt egy-egy adatsort a bal és jobboldali sáv adatai közül, és összegzi. Az összegzés a fent ismertetett módon, közös időtengely kijelölésével történik. (*3. ábra*) A szkript tartalmaz egy opcionálisan használható vizualizációs modult is, amely segíthet az összegezett jelalak ellenőrzésében.



3. ábra Bemeneti jel előállítsa kettő jelkomponensből

Az adatok feldolgozását *a SensorDataProcessor* osztály végzi, amelyben három függvény került definiálásra. A *collect_csv_files* az adatok beolvasását végzi, itt paraméterént megadható a fájlok száma is. A process_csv_files metódus végzi el az adatok feldolgozását és előkészítését a neurális hálózat számára. Minden egyes blokk esetén eltávolítja az utolsó két sort, amelyek szöveges formátumban tartalmazzák a szülő fájlok neveit. Az adatokat ezután a maximális méretre, 660 pont hosszúságúra egészíti ki, amennyiben az ennél rövidebb, ezzel biztosítva, hogy egyforma hosszúságúak legyenek. Végül egy háromdimenziós tenzor formátumban kerülnek tárolásra, ahol a dimenziók a mintaszám, időlépés, szenzorok száma. A harmadik függvény a *load_original_signals*, amely az eredeti jelek értékeit tölti be. A metódus emellett egy úgynevezett padding maszkot is készít, amelynek bináris értékei megadják, hogy az adott helyen volt e valós szenzorérték vagy sem. Ezutóbbi a veszteségfüggvény számítása során játszik fontos szerepet.

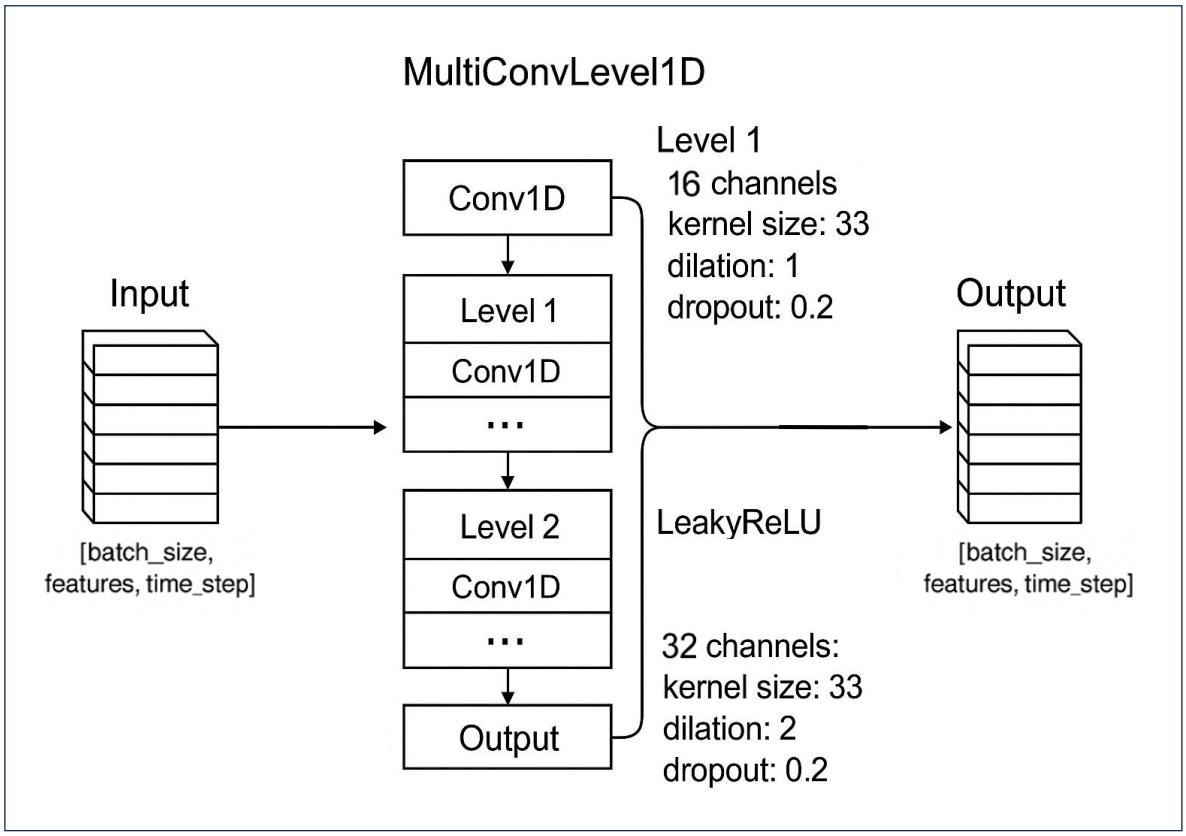
Neurális hálózat alapú jelszétválasztás

A modell egy hierarchikus egydimenziós CNN architektúrát alkalmaz, amely időbeli adatokra van optimalizálva egy olyan veszteségfüggvénnyel, amely kiegyensúlyozza a rekonstrukciós pontosságot és az időbeli trendek megőrzését. A hálózat célja, hogy az egyik komponenst, mint zajt, eltávolítsa a bementi jelből, így előállítva a másik komponenst. Ezután egyszerű kivonással visszakapjuk az eltávolított komponenst is.

1.1 A CNN architektúrája és működése

A modell alapvető komponense a *MultiConvLevel1D* modul, amely egy hierarchikus 1D CNN struktúrát valósít meg, több szinttel, ahol minden szint testreszabható rétegekkel, csatornákkal, kernelméretekkel és dilatációkkal rendelkezik. A bemeneti jel 26 csatornát tartalmaz, amelyek a különböző szenzoradatok jellemzőit tartalmazzák, és a kimenet is 26 csatornás, amely megfelel az egyik jelkomponens méretének. A hálózat két szintet tartalmaz, mindegyik három konvolúciós réteggel. Az első szint 16 csatornát használ 1-es dilatációval, míg a második szint 32 csatornát 2-es dilatációval, így a modell képes mind a helyi, mind a hosszabb távú időbeli mintázatokat megragadni. A kernelméret mindkét szinten 33, amely elég nagy ahhoz, hogy jelentős időbeli függőségeket is feldolgozzon. Az aktivációs függvényként *LeakyReLU*-t alkalmaztam 0.01-es negatív lejtéssel, ami javítja a gradiensáramlást a mélyebb hálózatokban, különösen a több szint miatt. A túltanulás elkerülése érdekében 0.2 valószínűségű dropout réteget használtam a tanítási fázisban. A reziduális kapcsolatok engedélyezése lehetővé teszi, hogy a bemenet közvetlenül hozzáadódjon a kimenethez, ami javítja a gradiens stabilitását és a tanítási hatékonyságot.

A bemeneti jel alakja *[batch_size, time_steps, features]*. Ahhoz, hogy megfeleljen a modell bemeneti követelményeinek, transzponálnunk kell *[batch_size, features, time_steps]* formára. A *MultiConvLevel1*D hálózaton való áthaladás után a kimenet azonos alakú, amelyet vissza transzponálunk az eredeti formára a veszteségszámításhoz. A súlyok Kaiming normál inicializációval kerülnek beállításra, amely jól igazodik a *LeakyReLU* aktivációhoz, hogy optimalizálja a konvergenciát és csökkentse a korai tanítási instabilitásokat.



4. ábra CNN felépítése

1.2 Veszteségfüggvény

A modell egy egyedi veszteségfüggvényt használ, amely az átlagos négyzetes hibát (MSE) kombinálja egy trendalapú büntetéssel, hogy egyszerre biztosítsa a pontszerű pontosságot és az időbeli dinamikák megőrzését. A teljes veszteség a következőképpen számítható: a teljes veszteség egyenlő az MSE veszteség és a trend veszteség súlyozott összegével, ahol alfa és béta a súlyok.

$$L_{\text{total}} = \alpha \cdot L_{\text{MSE}} + \beta \cdot L_{\text{trend}} \tag{1.1}$$

Az MSE veszteség a becsült komponens és a céljel közötti pontszerű különbséget méri, maszkkal súlyozva, hogy csak a releváns jelértékekre fókuszáljon, így kizárva a zajos vagy irreleváns adatpontokat. Az *(1.2) egyenlet* alapján az MSE veszteség a prediktált komponens és a céljel maszkkal szorzott különbségének négyzetes átlaga.

$$L_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}\left((\widehat{y_i}\cdot m_i) - (y_i \cdot m_i)\right)^2 \tag{1.2}$$

A trendveszteség az időbeli különbségek eltéréseit bünteti, három különböző időbeli eltolásra (k=1, 2, 3) számolva, ahol K=3 a maximális eltolás. Minden k értékre a veszteség két komponensből áll: egy súlyozott különbség veszteségből és egy laposssági büntetésből. *((1.3) egyenlet)*

$$L_{\text{trend}} = \sum_{k=1}^{K}\frac{1}{T-k}\left(L_{\text{diff},k} + L_{\text{flat},k}\right) \tag{1.3}$$

A súlyozott különbség veszteség a prediktált és a céljelek időbeli különbségei közötti MSE-t méri, amelyet az amplitúdóval súlyozunk, hogy a nagyobb változások nagyobb hatást gyakoroljanak. Az amplitúdósúly az abszolút különbségek összege alapján számolódik, majd az oszlopok mentén átlagolja, és egy minimális 0.1-es értékkel korlátozza, hogy elkerüljük a túl kicsi súlyokat. Ezt írják le az *(1.4) – (1.5) egyenletek.*

$$L_{\text{diff},k} = \frac{1}{N}\sum_{i=1}^{N} w_{\text{amp},i}\cdot\left(\left(\widehat{y_{\iota,t+k}} - \widehat{y_{\iota,t}}\right)\cdot m_{i,t} - \left(y_{i,t+k} - y_{i,t}\right)\cdot m_{i,t}\right)^2 \tag{1.4}$$

$$w_{\text{amp},i} = \max\left(\frac{1}{F}\sum_{f=1}^{F}\left(\left|\widehat{y_{\iota,t+k}} - \widehat{y_{\iota,t}}\right| + \left|y_{i,t+k} - y_{i,t}\right|\right), 0.1\right) \tag{1.5}$$

A laposság büntetés a céljel minimális változású régióiban alkalmaz exponenciális súlyozást a célkülönbségek alapján, így, ha a céljel különbsége közel nulla, a büntetés nagyobb súlyt kap. A trendveszteség minden k-ra az időbeli lépések számával (T-k) normalizálódik, hogy kiegyensúlyozza a különböző eltolások hatását. Ez a kombinált veszteség biztosítja, hogy a modell nemcsak a helyi pontosságra, hanem a globális időbeli trendekre is figyel. *((1.6) – (1.7) egyenletek* alapján)

$$L_{\text{flat},k} = \frac{1}{N}\sum_{i=1}^{N} w_{\text{flat},i}\cdot\left(\left(\widehat{y_{\iota,t+k}} - \widehat{y_{\iota,t}}\right)\cdot m_{i,t}\right)^2 \tag{1.6}$$

$$w_{\text{flat},i} = e^{-10\cdot\left|\left(y_{i,t+k} - y_{i,t}\right)\right|} \tag{1.7}$$

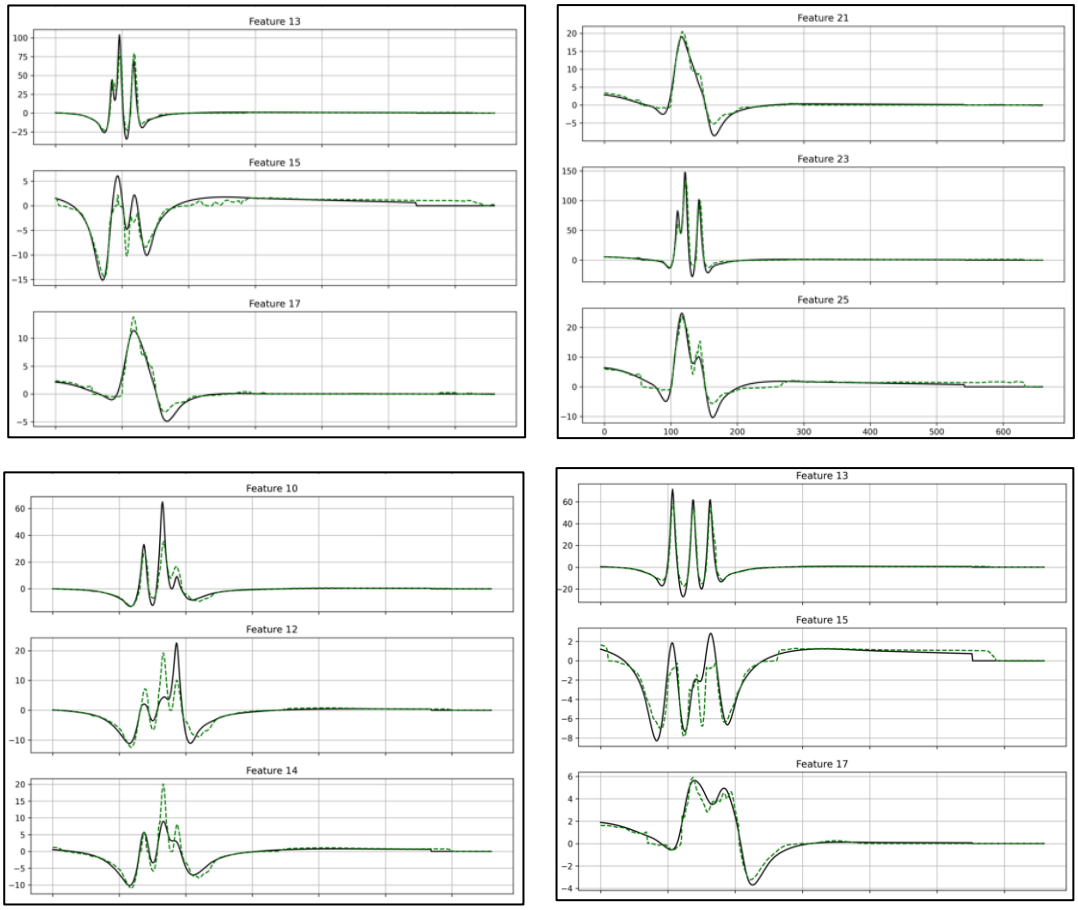
1.3 Tanítás

A modell a PyTorch Lightning *Trainer* osztályával került tanításra, amely rugalmas és hatékony tanítási környezetet biztosít. Optimalizálóként *Adam* algoritmust használtam 0.001-es tanulási rátával, amely jól működik mély neurális hálózatok esetében. A tanulási rátát egy *ReduceLROnPlateau* ütemező szabályozza, amely 0.5-ös faktorral csökkenti a rátát, ha a validációs veszteség 5 epochon keresztül nem javul, így adaptívan igazítva a tanítást a modell konvergenciájához. A *ModelCheckpoint* hívás a legjobb modellt menti a validációs veszteség alapján. A mentett fájlnevek tartalmazzák az epoch számát és a veszteség értékét a könnyű azonosítás és a tanulási folyamat nyomon követhetősége érdekében.

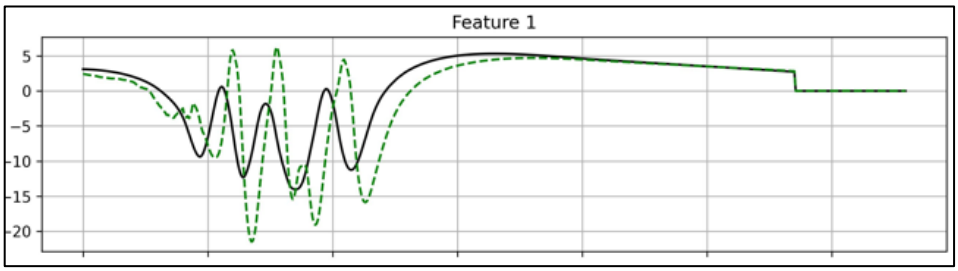
Eredmények

A tanított neurális hálózat értékelését az *5-6. ábrákon* látható szenzorértékek összehasonlításával végeztem. A diagrammokon fekete színnel az eredeti jel, míg zöld vonallal a modell által prediktált jel olvasható le. A vizsgálat célja, hogy a modell mennyire képes zajként eltávolítani a másik komponenst a bemeneti jelből és ezzel rekonstruálni az egyik sávhoz tartozó mérést.

A legtöbb jellemző esetén a becsült értékek jól közelítik a célgörbéket, alakhűen adja vissza különösen a csúcsokat és lengésszerű mintázatokat. Ezek mellett a modell implicit módon zajszűrést is végez, leginkább a domináns mintázatokra fókuszál. Ugyanakkor esetenként megfigyelhetőek időbeli eltolódások, illetve amplitúdóbeli torzulások is. Ennek ellenére a rekonstruált jelalakok továbbra is megfelelnek az elvártaknak. Fontos megemlíteni, hogy a hálózat képes egyszerre több időbeli jellemző párhuzamos feldolgozására, azaz nem csak megtanulja külön-külön a komponenseket, hanem általános mintázatokat képes azonosítani.



5. ábra Véletlenszerűen kiválasztott fájlok eredményei



6. ábra Amplitúdótorzulás. *A fejezetben ismertetett módon esetenként megfigyelhető az amplitúdótorzulás jelensége, ahogy a fenti szenzorjel esetén is. Azonban a becsült jelalak továbbra is mutatja, hogy a modell jól tanult, az alakhűség fennáll.*