Általános információk, a diplomaterv szerkezete

A diplomaterv szerkezete a BME Villamosmérnöki és Informatikai Karán:

1. Diplomaterv feladatkiírás
2. Címoldal
3. Tartalomjegyzék
4. A diplomatervező nyilatkozata az önálló munkáról és az elektronikus adatok kezeléséről
5. Tartalmi összefoglaló magyarul és angolul
6. Bevezetés: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása
7. A feladatkiírás pontosítása és részletes elemzése
8. Előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések
9. A tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása
10. A megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek
11. Esetleges köszönetnyilvánítások
12. Részletes és pontos irodalomjegyzék
13. Függelék(ek)

Felhasználható a következő oldaltól kezdődő LaTeXdiplomatervsablon dokumentum tartalma.

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmargóval készüljenek (mindenhol 2,5 cm, baloldalon 1 cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel, de ettől kismértékben el lehet térni, ill. más betűtípus használata is megengedett.

Minden oldalon – az első négy szerkezeti elem kivételével – szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellátni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellátni. A fejezeteket decimális aláosztással számozzuk, maximálisan 3 aláosztás mélységben (pl. 2.3.4.1.). Az ábrákat, táblázatokat és képleteket célszerű fejezetenként külön számozni (pl. 2.4. ábra, 4.2. táblázat vagy képletnél (3.2)). A fejezetcímeket igazítsuk balra, a normál szövegnél viszont használjunk sorkiegyenlítést. Az ábrákat, táblázatokat és a hozzájuk tartozó címet igazítsuk középre. A cím a jelölt rész alatt helyezkedjen el.

A képeket lehetőleg rajzoló programmal készítsék el, az egyenleteket egyenlet-szerkesztő segítségével írják le (A LaTeX ehhez kézenfekvő megoldásokat nyújt).

Az irodalomjegyzék szövegközi hivatkozása történhet sorszámozva (ez a preferált megoldás) vagy a Harvard-rendszerben (a szerző és az évszám megadásával). A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámozott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban mindig az eredeti nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben (a LaTeX-sablon a BibTeX segítségével mindezt automatikusan kezeli). Minden publikáció a szerzők után a következő adatok szerepelnek: folyóirat cikkeknél a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóiratok címét csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internetes hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal tulajdonosát és tartalmát (mivel a link egy idő után akár elérhetetlenné is válhat), valamint az elérés időpontját.

Fontos:

- A szakdolgozatkészítő / diplomatervező nyilatkozata (a jelen sablonban szereplő szövegtartalommal) kötelező előírás, Karunkon ennek hiányában a szakdolgozat/diplomaterv nem bírálható és nem védhető!
- Mind a dolgozat, mind a melléklet maximálisan 15 MB méretű lehet!

Jó munkát, sikeres szakdolgozatkészítést, ill. diplomatervezést kívánunk!

FELADATKIÍRÁS

A feladatkiírást a tanszéki adminisztrációban lehet átvenni, és a leadott munkába eredeti, tanszéki pecséttel ellátott és a tanszékvezető által aláírt lapot kell belefűzni (ezen oldal *helyett*, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell beleszerkeszteni ezt a feladatkiírást.

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Távközlési és Médiainformatikai Tanszék

Mély neuronháló alapú orosz nyelvű beszédfelismerő fejlesztése

Szakdolgozat

Készítette

Dobsinszki Gergely

Konzulens

dr. Mihajlik Péter

2020. november 30.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Dobsinszki Gergely*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot 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(k), 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 autentikált felhasználók számára) közzétegye. 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.

Budapest, 2020. november 30.

Dobsinszki Gergely
hallgató

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomatervnek. A sablon használata opcionális. Ez a sablon LaTeX alapú, a *TeXLive* TeX-implementációval és a PDF-LaTeX fordítóval működőképes.

Abstract

This document is a LaTeX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* TeX implementation, and it requires the PDF-LaTeX compiler.

1. fejezet

Bevezetés

1.1. Gépi beszédfelismerés

Az automatikus gépi beszédfelismerés (ASR) régóta foglalkoztatja az emberek fantáziáját. Személyi asszisztens formájában már évtizedekkel ezelőtt találkozhatott vele az átlagos sci-fi kedvelő. De már napjainkban is ott lapul a zsebünkben a technológia, az okos-telefonjaink, melyek a legtöbb ember számára a mindennapok részét képzik, rendelkeznek beszédfelismerő funkcióval. Legyen az egy egyszerűbb személyi asszisztens, amely segítségével például könnyedén beállíthatóak emlékeztetők, vagy egy egyszerű Google Search-ön alapuló keresés, melyhez a begépelést hang segítségével is el lehet végezni. Nappalinkban is egyre inkább hódít, jó példa rá az Amazon Alexa-ja, aminek a segítségével több okos eszköz is kezelhető a lakóterünkben.

Ezek a technológiák mind automatikus beszédfelismerésen alapszanak, amelyet alkalmazva az ember kizárólag hang interfész segítségével tud kommunikálni a géppel, mintha csak egy másik emberrel csevegne. Ez a megközelítés nagy szabadságot nyújt a kommunikáló számára, nem köti többé billentyűzet, egér, érintőképernyő de még monitor, kijelző sem. Az egyre inkább pontosodó, esetenként még az emberi precizitáson is túlmutató, felismerés nyújtotta lehetőségek felhasználási területe egyre bővül, a technológia egyre inkább bekerül a köztudatba és mindennapjainkba.

1.2. Út az end to end alapú gépi beszédfelismerésig

A fonográf, egy Thomas Eddison által jegyzett 1877-es találmány, volt az első eszköz mely hang rögzítésére alkalmas volt. A módszer finomításával és térhódításával, illetve a számítógépek megjelenésével, és a digitalizációval egyre inkább elterjedt és fejlődött a hangrögzítés. Az első komoly eredményeket felmutató kutatási projektet a beszédfelismerés területén az Egyesült Államokbeli Fejlett Védelmi Kutatási Projektek Ügynöksége (DARPA) finanszírozta. A kutatás célja egy 1000 szavat felismerni képes program megalkotása volt, melyet a kitűzött 5 éven belül sikerült is megalkotni [**?**].

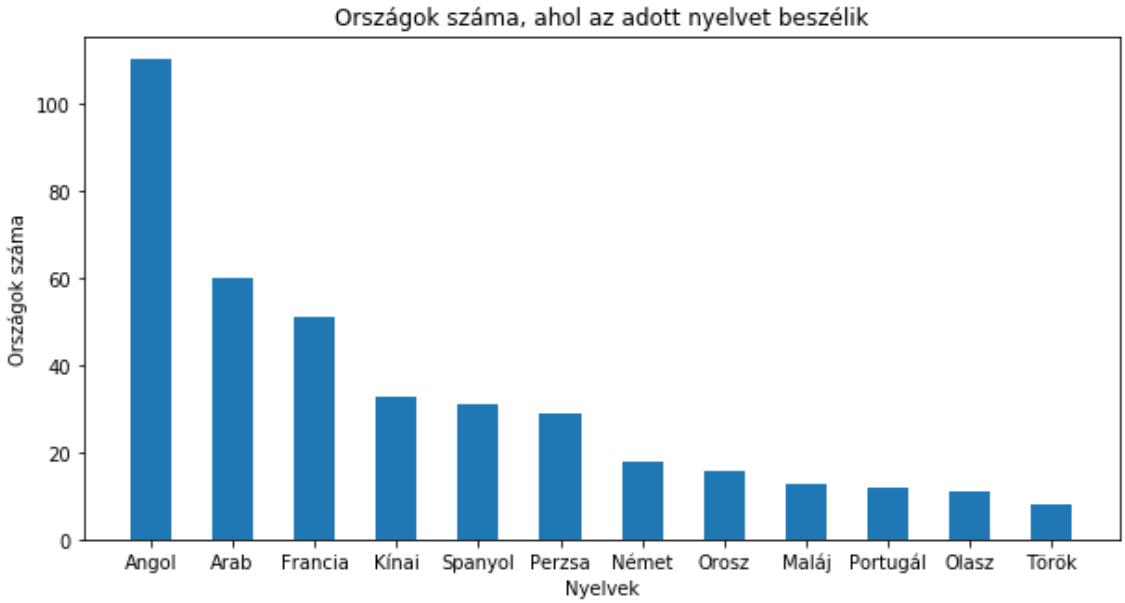
Az 1980-as években egy újabb technológia a rejtett Markov-modell (RMM), egy időbeli valószínűségi modell, forradalmasította a gépi beszédfelismerést. A modell segítségével lényegesen javítható volt a beszédfelismerés pontossága. Az RMM napjainkban is nagy sikerességnek örvend, rendkívül elterjedten használják különböző elemekkel, akusztikus- és nyelv modellekkel párosítva a minél nagyobb pontosságú beszédfelismerés végett.

A neurális hálók térhódításával 2010-es években megjelent egy alternatív, folyamatosan terjedő megközelítés, az end-to-end alapú beszédfelismerés. Segítségével kiküszöbölhető a komplex, több elemből álló módszerek alkalmazása, és egy egységből felépülő modell állítható elő.

1.3. Az orosz nyelv

Ahogy az a 1.1-es ábrán látható világszerte az egyik legelterjedtebb nyelv az angol, mely nem csak a közösségi, társadalmi és politikai, hanem a tudományos életet is uralja. Nem meglepő, hogy ezt a nyelvet tekintjük a beszédfelismerő rendszerek pontosságának mérésekor alapnak, ezen a nyelven vannak a legkiterjedtebb szabadon elérhető beszéd adatbázisok is.

Számos nyelvet beszélnek világszerte az angol mellett, sokak számára viszont az angol nem az anyanyelvük, estleg egyáltalán nem beszélik azt. Beszédfelismerés során tehát szükséges egyéb hivatalos nyelveket is vizsgálni és kutatni.



1.1. ábra. Országok száma, melyekben az adott nyelvet beszélik. [**?**]

Oroszul több mint 140 millió ember anyanelve és további 110 millióan használják az oroszt második nyelvként[1], így az orosz nyelv igen fontos szerepet tölt be a nemzetközi porondon. Az 1.1-es ábrán látható, hogy elterejedt nyelvnek számít több országban is. Az end-to-end technológia és a neurális hálók által nyújtott lehetőségek megkönnyítik a nyelvek közti átállást, így egy angoltól eltérő nyelv vizsgálata és kutatása gördülékenyen megvalósítható egy angol nyelvű példa alapján.

1.4. Diplomaterv felépítése

A követelmények elemzése fejezetben a sarkalatos pontok kerülnek részletezésre, kifejtésre. Ezen pontok kidolgozásának lehetőségei, esetleges nehézségei kerülnek említésre a felsorakoztatva a tervezés előtt felmerült, megvalósított és elvetett ötleteket.

Az end-to-end beszédfelismerés elmélete rész a beszédfelismerés teóriájával foglalkozik, különösképp kitérve a neurális hálókat alkalmazó implementációra és azok alapjaira. Továbbá tartalmazza az eddigi legjobb megoldásokat, melyeket össze is hasonlít, röviden értékelve azok sikerességét. A levonható következtetések fényében

[1]Az orosz nyelv, Wikipedia `https://hu.wikipedia.org/wiki/Orosz_nyelv`

felvázolja a fejlesztés kiindulási alapjának tekinthető állapotot és tervezésbeli döntéseket.

A tervezés és implementáció fejeztet magába foglalja az előzetes döntéseket, a lehetséges kísérletek várható eredményét. A döntési lehetőségeket, értékelve és az azok mögötti ötleteket, esetleges alternatívákkal kiegészítve. A végső soron választott megoldások alátámasztása és indoklása is megjelenik a részben.

Az összefoglaló tartalmazza az elért eredmények bemutatását, azok értékelését és a végső következtetések levonását.

2. fejezet

A követelmények elemzése

2.1. Beszédfelismerés bemutatása

Az irodalomkutatás során az end-to-end alapú beszédfelismerést különböző források, cikkek, internetes blogok és papírok alapján célszerű végezni. Mivel egy egészen új és feltörekvő megközelítésről van szó, célszerű a forrásokat megvizsgálni, összevetni az azonos témáról írtakat az alátámasztás és a könnyebb értelmezhetőség végett.

A neurális hálók működésének bemutatása elengedhetetlen a szerepük nyomán a beszédfelismerésben. Fontos kitérni az end-to-end alap mivoltára, ennek előnyeinek és hátrányainak bemutatása az eddig bevált, hagyományos módszerekkel szemben.

Érdemes az elméleten túl a potenciálisan felhasználható megoldásokat részletesen megismerni, melyek implementálhatók a későbbi tervezés során a jobb eredmények végett.

2.2. Korszerű architektúrák és tervezés

Többféle neuronhálós architektúrák léteznek különböző feladatokra, így a beszédfelismerésre is. Felderítésük, előnyeik és hátrányaik megismerése és összevetésük az irodalomkutatás során végezhető el.

A neurális hálók természetüknél fogva igen összetettek, ezért gyakori az egyes rendszerek magas erőforrásigénye, melyeket a erőforrásszűkében akár kivitelezhetetlen is lenne használni. Ezért célszerű, hogy olyan architektúrát válasszunk, ami az előzőek alapján optimális teljesítményt nyújt, azaz látványos

és felhasználható eredmények elérésére képes a lehető legrövidebb idő alatt a legkevesebb erőforrást felhasználva. A rövid idő alatt lefutó modellek a tesztelési fázist is nagyban felgyorsíthatják, így az idő szerepe kulcsfontosságú, míg természetesen fontos és nem elhanyagolható a pontosság is.

Kezdetben angol nyelven célszerű tesztelni az megtervezett architektúrát és az egyes paramétereinek pontosítását, mivel angol nyelven rengeteg eredmény elérhető, ezáltal összevethető. Ha egy megtervezett rendszer működésének megítélése angol nyelvű adatok alapján megfelelő akkor már célszerű áttérni az orosz nyelvre.

2.3. Orosz nyelvű implementáció

Egy angoltól eltérő nyelvre való áttéréskor el kell végezni annak a vizsgálatát, hogy az újabb nyelv milyen különbségeket rejt. Egy gyakori eltérés az ABC, de előfordulhatnák más előfeldolgozásbeli különbségek is. Ha egy keretrendszer angol nyelvre van optimalizálva, előfordulhat, hogy ezeket a finomításokat el kell vetni és egy általános megközelítést kell alkalmazni az új nyelv tanítása során.

A célnyelvi adatokat különböző forrásokból lehet begyűjteni. Mivel az adatbázisok csak nyíltak lehetnek, ezért ezek megbízhatóságát meg kell vizsgálni akár szúrópróbaszerűen akár más felhasználók véleménye alapján. Az adatok mérete mind a neurális modellek tanítási ideje mind tárhelykapacitás miatt nem szabad, hogy túl nagy legyen, meg kell találni az optimális mennyiséget.

Tanítás során a jobb eredmény elérése érdekében alkalmazható a transfer-learning, ami egy elterjedt tanítási módszer neurális hálók használata során. Ez a folyamat egy adott architektúrájú, előre tanított modell továbbtanítását foglalja magába. Beszédfelismerés esetén az eredeti modelltől eltérő nyelv alkalmazásánál kihívást jelent a nyelvre való áttérés. Szerencsére lehetőség van az előre tanított modell tetszőleges részeit átvenni, nem szükséges a nyelvspecifikus elemeket is importálni, így azok tanításhoz kezdődhet szeparálva, nulláról.

Célszerű az architektúra tervezésénél gondolni a szabadon elérhető előre tanított modellekre és ilyen felépítéseket is fontolóra venni és megvizsgálni, hiszen csak ezekkel lehetséges a transfer-learning alkalmazása.

2.4. Optimalizálás és kiértékelés

Az elkészült modellt ki kell értékelni, összehasonlítani más eredményekkel és véleményezni az így elért eredményt. A kiéréskeléshez egy elterjedt módszert kell ezért használni, hogy összemérhetőek legyenek az eredmények.

Lehetőség van az eredmény javítására különböző módszerekkel, melyek már ismertek a neurális hálók körében, vagy főként beszédfelismerésnél jelennek meg. Az előbbi csoportba tartozik például az augmentálás, mely folyamat során a meglévő tanítóhalmazt dúsítják különböző módszerekkel. Utóbbi csoportba tartoznak a különböző dekóderek, annak a módja hogy hogyan értelmezzük a valószínűségként kapott kimeneteket az egyes időpillanatokban, illetve a nyelv modellek.

2.5. Munka összegzése és kitekintés

Befejezettképp kerül sor a kitűzött feladatok sikerességének bemutatására, az elért eredmények részletezéséve és az esetleges kudarcok a zsákutcák leírására.

További, ki nem próbált lehetőséget és fejlesztési ötletek javaslását is elvégezzük, indokolva, hogy miért is jelenthetnek fejlődést az eredményekben és hogy ezek miért nem lettek megvalósítva, kipróbálva.

3. fejezet

Az end-to-end beszédfelismerés elmélete

3.1. Hagyományostól az end-to-end-ig

A hagyományos ASR-ok (automatikus beszédfelismerők) több elemből álló, összetett rendszerek, melyek manapság igen pontosan meg tudják állapítani mi volt az eredetileg elhangzott szöveg, és leképezik azt írásos formába. Az egyik legelterjedtebb a HMM (Rejtett Markov Modell) alapú beszédfelismerő rendszer [1]. A rendszernek több eleme is van: egy dekóder mely tartalmaz egy nyelv modellt, akusztikus modelleket és egy kiejtési szótárat is, illetve tartalmaz egy elemet, mely a bemeneti hangból a kiemeli jellemzőket, átalakítja őket a rendszer által kezelhető formájúvá. A HMM-en alapuló rendszerek fő eleme a Viterbi algoritmus [**?**], mely dinamikus programozást alkalmazva találja meg a legjobb illeszkedést a bemeneti szöveg és az adott beszéd modell közt.

A rendszer különböző elemeo erősen függenek egymástól. Ha az egyik nem rendeltetés szerűén működik és rossz kimenetet ad, akár saját vagy egy korábbi elem hibájából kifolyólag az befolyásolja a többi elemet is, így elrontva, torzítva a kapott végeredményt. Az egyes elemeket külön-külön kell kalibrálni, tanítani és tökéletesíteni.

Ezzel szemben, a napjainkban egyre inkább népszerű, E2E (end-to-end) alapú beszédfelismerő rendszerek [2] sokkal egyszerűbb, kevesebb köztes elemet

igénylő megközelítést nyújtanak. Bizonyos szituációkban, ahol egy specifikusabb szövegkörnyezetben, pl. pénzügyi vagy jogi, elhangzó szavak azonosítása a cél, már a hagyományos ASR rendszereknél is pontosabb eredményeket képesek produkálni.

Az E2E rendszerek mély neurális hálókon alapszanak, közvetlen bemenetük a nyers hang, kimenetük pedig a becsült szöveg. Természetesen a neurális háló bemenetéhez a hangot még előre fel kell dolgozni, amit pre-processing-nek neveznek. A háló kimenete pedig az egyes bemeneti időegységekre becsült karakterek valószínűségei melyekből ki kell nyerni a végső, feltételezett szöveget.

3.2. A beszédfelismerés lépései

3.2.1. Előfeldolgozás

A beszéd előfeldolgozása az első fontos lépése a beszédfelismerő rendszereknek. Vannak törekvések, melyekben a E2E neurális hálónak közvetlenül a nyers, feldolgozatlan hanganyagot adják meg és ez alapján tanítják [4], ezáltal még inkább tisztán a neurális háló működésére támaszkodva. Egyelőre ennek a megközelítésnek az eredményei messze elmaradnak a hang előfeldolgozásával elérhető SOTA modellek pontosságától.

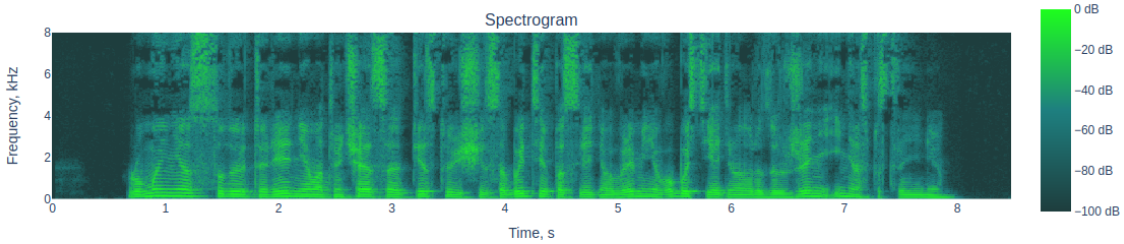
A két fő megközelítés többnyire hasonló, apró eltérésekkel. Az egyik legelterjedtebb és bevált átalakítás a spektogram.

Ahogy az 1. ábrán is látható, a spektogramm három tengely mentén reprezentálja az adatot. A vízszintes tengelyen az idő, míg a függéleges tengelyen a frekvencia van megadva. Van továbbá egy harmadik tengely is, ami színnel van jelölve és mértékegysége a decibel. Ez a jel amplitúdója vagy energiája. Világosabb szín erősebb, hangosabb hangot, illetve sötétebb szín gyengébb, alacsonyabb hangot jelképez.

A szokványos spektrogrammon kívül használatos még a MFCCs (Mel-Frequency Cepstral Coefficients) is. Felépítése hasonló a spektogramméval, viszont a frekvencia helyett MFC koefficiens [**?**] értéke van az y tengelyen. Ez a Mel skála, amely úgy van megválasztva, hogy a skálán egyelő távolságra lévő hangok az emberi fül számára is hasonló távolságúnak tűnjenek. A frekvenciához képest egy lényeges

változást jelent, hiszen az ember számára könnyen hallható kisebb frekvenciákon, például az 500 Hz és 1000 Hz közötti különbség, míg 6500 Hz és 7000 Hz között már kevésbé számottevő az eltérés. Ez utóbbi érték nagyságrendekkel kisebb a frekvencia értéknél így lényegesen befolyásolhatja a neurális háló tanulási folyamatát.

Az előfeldolgozott jelet időegységekre bontjuk, melyek már a neurális háló bemenetét képzik. A kimeneten az egyes időegységekre kapott predikciókból pedig összeállítható a feltételezett szöveg.



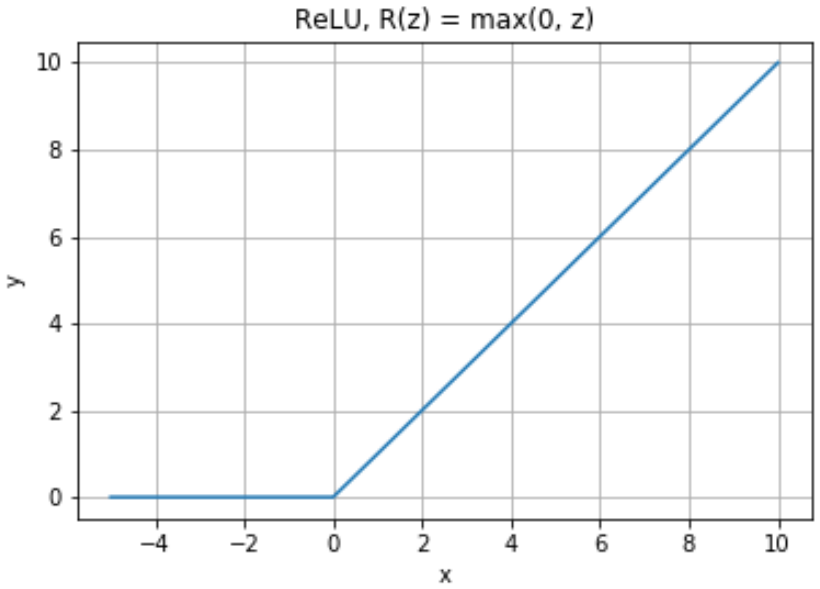
3.1. ábra. Spectogram.

3.2.2. Neurális hálók

A neurális hálók az emberi idegek működésének mintája alapján modellezett láncolatok. Számos különböző, komplex feladat megoldására előszeretettel alkalmazzák őket. Röviden összefoglalva három fő alkotó részt kell elképzelni. Egyik a bemenet, például egy kép esetén a pixelek, beszédfelismerés esetén az egyes időpillanatokhoz tartozó értékek vektor formában. Spektogram esetén az időpillanathoz tartozó vektor egyes elemei a frekvenciát jelképezik és az elemek értéke pedig a hangerősséget.

A neurális háló közbülső, rejtettnek, nevezett része tetszőleges méreteket ölthet. Az egyes rétegek pontjai mindig az előző réteg pontjaival vannak összekötve úgy nevezett súlyok segítségével. Az újabb rétegek pontjainak értékét az abba befutó súlyok, illetve a súlyok és kiinduló pontjuk szorzatainak az összegével számítjuk. A pontok végső értéke még egy aktivációs függvénynek nevezett kiértékelésen is keresztül megy, mielőtt a következő réteg értékeit kiszámítatnánk vele. Az aktivációs függvények igen eltérőek lehetnek, egy népszerű függvény a ReLU, ami a 3.2-es ábrán látható. Ezeket a pontokat neuronoknak szokás nevezni, innen ered a neurálsi

háló kifejezés. A háló tanítása során ezen súlyok értékét módosítjuk úgy, hogy a legoptimálisabb, számunkra kedvezőnek vélt eredményt kapjuk.



3.2. ábra. A ReLU aktivációs függvény.

Az utolsó réteg a kimeneti réteg, ahol az eredményt kapjuk meg. A kimeneti réteg lehet regresszív, egy előre meg nem határozott, tetszőleges érték, vagy a mi esetünkben klasszifikáció, amikor előre meghatározott címkék (label-ek) valószínűségére vagyunk kíváncsiak. Legegyszerűbb estben a legvalószínűbb címke kerül kiválasztásra válaszként.

A valószínűség kiszámításához a softmax [**?**] függvényt szokás használni, ami szintén egy aktivációs függvény. A függvény egy vektor bemenetet megkapva normalizálja a vektor egyes értékeit úgy, hogy azok új értékei 0 és 1 közé essenek, miközben az összegük pontosan 1-et adjon. A softmax függvény, ahol x a bemeneti vektort jelöli, míg i és j a vektor egyes elemeinek a sorszámát:

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))}. \tag{3.1}$$

Neurális hálók egyik leggyakoribb tanítási módszere a felügyelt tanítás. Ez azt jelenti, hogy először például egy tanítatlan, véletlenszerűen inicializált, hálóból kiindulva kiértékeljük egy tetszőleges bemenetre a végeredményt. Majd

a végeredmény helyességétől függően változtatjuk a súlyok értékét a hálóban: azon súlyokat melyek egy rossz eredmény nagyobb valószínűségéhez járultak hozzá büntetjük, csökkentjük az értéküket, míg azokat melyek egy jó eredmény pozitív értékéhez járultnak hozzá növeljük. Ezt a folyamatot szokás backpropagation-nak nevezni. A büntetés mértéke is létfontosságú, hiszen fontos, hogy ne csak az adott bemenetre tudjon pontos végeredményt adni a háló, hanem általánosan is pontos legyen.

Az egyik legegyszerűbb réteg típus a Dense (sűrű) vagy más néven Fully Connected, ahogy a neve is utal rá az egyes neuronok a súlyokkal sűrűn vannak összekötve. Ez azt jelenti, hogy minden egyes réteg mindegyik neuronja össze van kötve az előző réteg összes neuronjával súlyokon keresztül. Több Dense réteg is követheti egymást, mely esetben az adott réteg összes neuronja össze van kötve a következő réteg összes neuronjával is.

Az E2E-hez használt leggyakoribb neurális háló réteg fajták az rekurrens (RNN) és konvolúciós (CNN) rétegek. Ez utóbbit elterjedten használják a különböző képfelismerő feladatok terültén is.

Lényegesen összetettebb a Dense rétegnél a konvolúciós réteg [5], melynél egy, a teljes bemenetnél kisebb, kernel-t (egyes forrásokban filter) mozgatunk végig a bemenet elemein és kiértékeljük az általa lefedett értékeket. A lefedett számokat a kernel velük fedésben lévő értékével összeszorozzuk, ezek az értékek főként a -1, 0 és 1, majd a kapott eredményeket összeadjuk. Ezt követően a kernel-t egy megadott értékkel, a stride-al mozgatva kiértékeljük a következő, kernel által lefedett számokat. A konvolúció végezhető egy vagy több dimenziós bemeneten, különböző kernel-ekkel párhuzamosan is, kép esetén például három csatornás képen, ahol a piros kék és zöld értékek külön csatornákon vannak reprezentálva. Egy opcionális lépés, hogy a bement széleit feltöltjük-e nullásokkal, ez az úgy nevezett zero padding. Ennek célja, hogy ugyan akkora legyen a kimenet, mint a padding nélküli bemenet volt. Ezen kívül is szükség lehet padding-re például egy 5x5-ös kép esetén, ha a kernel mérete 3x3 és a stride 3, akkor ki kell tölteni a kép széleit, hogy minden bemeneti értéken legalább egyszer végig menjen a kernel.

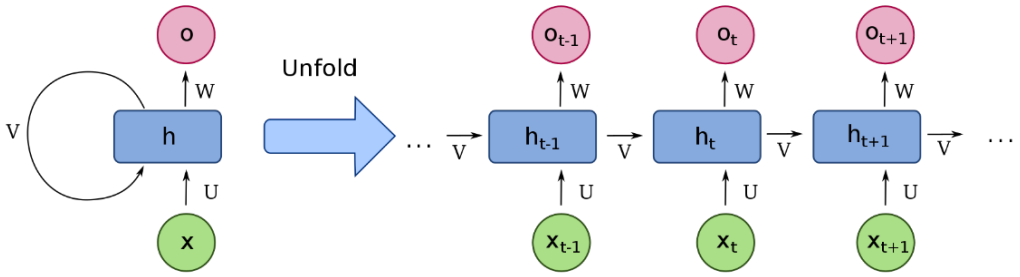
Egy CNN-hez kapcsolódó, kernellel rendelkező rétegtípus a pooling réteg, mely kifejezetten az adat méretének csökkenését szolgálja. Ezzel a célja a számítási

kapacitás csökkentése, miközben minél inkább megőrzi az adatban rejlő információt. Két használt pooling típus a max vagy average pooling. Előbbinél a kernel által befoglalt értékek maximumát választja, míg utóbbinál az átlagukat.

A konvolúciós vagy pooling rétegek működésükből adódóan csökkenthetik a kimenetük méretét, ezáltal ismerni kell annak pontos dimenzióját, a következő réteghez kapcsolódó számítások végett. A kimenet méretét a következő képlettel lehet kiszámolni, ahol 'O' a kimenet mérete, 'W' a bemenet mérete, 'P' a padding mérete, 'K' a kernel mérete és 'S' a stride:

$$\mathbf{O} = \frac{\mathbf{W} - \mathbf{K} + \mathbf{2P}}{\mathbf{S} + (1)}. \tag{3.2}$$

A rekurrens rétegek [6] egyik legérdekesebb tulajdonsága, hogy nem csupán a bemeneti súlyaikat használják a neuronok kiértékeléséhez, így felhasználva a korábban tanultakat, hanem emlékeznek a korábbi neuron értékeikre is, melyek szintén befolyásolják az aktuális bemenetre kapott eredményt. Beszédfelismerés esetén ezt úgy is el lehet képzelni, hogy ha például elhangzik egy magánhangzó, akkor utána nagy valószínűséggel egy mássalhangzót várunk, ennek fényében, sikeres tanítás esetén, a mássalhangzókat részesíti előnyben egy azt megelőző magánhangzó után. Korábbi időpillanatok, állapotok tehát hozzájárulnak egy újabb állapot kiértékeléséhez a rekurrens rétegben, ahogy ez a 3.3-as ábrán is látható.



3.3. ábra. A rekurrens réteg "kigörgetve" [**?**].

Két altípusa a rekurrens hálóknak az LSTM és GRU (Long Short Term Memory, Gated Recurrent Unit). Ezek a típusok hasonló elveken működnek, a különbség a neuronok korábbi értékeinek újabb értékekre történő ráhatásának

számítási módjában rejlik. Az LSTM [7] fő célja, hogy kijátssza a hagyományos rekurrens hálók egyik legnagyobb problémáját, azt, hogy a régebbi bemenet súlya nincs hasonló fontossággal kezelve, mint a jelenlegi bemenethez közelebbi érték. Könnyen belátható, hogy fontos lehet egy mondat végi szót vizsgálva, hogy mi volt a szó a mondat elején, akár a vizsgálandó mondat elemet megelőző néhány szónál is. Az LSTM kapukat használ annak megelőzése céljából, hogy egy korábbi értékelés elvesszen, ezáltal kezelhetetlenné téve annak beleszámítását az újabb értékelésnél.

A GRU egy LSTM-hez hasonló újabb fejlesztésű réteg. Egyik fő tulajdonsága, hogy kevesebb elemet tartalmaz, így kevesebb paramétert használ, csökkentve a neurális háló komplexitását, ezáltal a szükséges számítási időt is.

Megemlítendő még a Bi-directional (két irányú) réteg, amely mögött az a gondolat húzódik meg, hogy ne csak a múltbéli bemenetek befolyásolják az adott bemenetünket, hanem a jövőbeliek is. Két rekurrens réteget használ, melyek ellentétes irányban haladnak egymással, így tetszőleges kiértékelésnél használhatóak a korábbi, illetve elkövetkezendő neuronok eredményei is.

3.2.3. CTC dekódolás és loss számítás

A CTC (Connectionist Temporal Classification) [**?**] funkciója a kimeneti dekódolása, a válasz kiértékelése, mely folyamán átalakítja a kimenetet a megadott címkék valószínűségi szekvenciájára. A címkék lehetnek az egyes karakterek, amik egy adott nyelvben előfordulhatnak. Ez esetben a CTC kiértékeli, hogy melyik betű vagy egyéb karakter hangzott el adott időpillanatban, adott bemenetre [8]. Ereje abban rejlik, hogy nincs szükség a kiértékelésnél a kimeneteket egyenként, karakter szinten összekötni az elvárt szöveggel, elegendő a kimeneti szöveget megadni, azt, amit kapni szeretnék az adott bemenetre. Másik fontos előnye, hogy a felismert szöveget nem szükséges külön feldolgozni, mivel átalakítja azt a végleges, feltételezett formára. Feldolgozza, hogy egy hosszú hang esetén, mint az 'ú' több időpillanaton keresztül is hallatszik az 'ú' betű, de nekünk a végeredményben csak egy darab 'ú' szükséges.

Felmerülhet egy probléma az összevonással, amikor olyan egymást követő betűket vonunk össze, melyek a szóban külön szerepelnek, mint például az 'ellentét' szóban. Ennek elkerülése végett bevezet a CTC egy üres karaktert, ami nem egyenlő

a szóközzel, és ezt olyan időegységekhez helyezi, ahol nem ismerhető fel betű a megadott szótárból, karaktergyűjteményből. Az ilyen üres karakterrel elválasztott betűket nem fogja összevonni kiértékeléskor.

A neurális háló által végzett kiértékelés után az ember által feldolgozható kimenet megszerzéséig szükség van még a karakterek összeillesztésére. Ehhez különböző elven működő dekódolókat használunk. Az egyik legalapvetőbb dekódolási algoritmus a best path eljárás. Két dolgot végez: kiszámolja a legvalószínűbb útvonalat a kapott kimeneti időegységeken át, majd eltávolítja az egymást követő azonos karaktereket, melyek közt nem található az üres karakter. Végül az üres karaktereket is eltávolítja, így eredményezve a végső predikciót.

További feladata a neurális háló tanításához használt loss kiszámítása [**?**]. A hiba számítása során a szempontunkból lényegtelen milyen karaktersorozattal jut el a végeredményhez, csupán az a cél, hogy pontosan az adott kimenetet eredményezze. A loss-t tehát úgy számítja, hogy megnézi az összes olyan karaktersorozat valószínűségét, melyből megkaphatja a kívánt eredményt és összegzi őket. Ezáltal az eredeti GT (Ground Truth) szöveg valószínűségi értéke magas, hogyha a megszerzéséhez szükséges kombinációk értékének az összege magas.

A CTC loss számítása a 3.3-as, 3.4-es és 3.5-ös képeltek alapján történik, ahol GT az elvárt szöveg, X az egyes akusztikus keretek, C egy adott karaktersorozat és c_t egy adott karakter adott időpillanatban.

$$CTCloss = -\log P(GT|X). \tag{3.3}$$

Összes lehetséges helyes karaktersorozat valószínűségének az összege:

$$P(GT|X) = \sum_{C=A(GT)} P(C|X). \tag{3.4}$$

Egy adott, helyes karaktersorozat valószínűsége:

$$P(C|X) = \prod_{t=1}^{T} y(c_t, t)). \tag{3.5}$$

3.3. Létező megoldások

A beszédfelismerő rendszereket pontosságuk alapján tudjuk rangsorolni. Egy nemzetközileg elismert pontossági mérőszám a WER (Word Error Rate). Ezt a pontosságot szokás szerint egy publikusan elérhető, angol nyelvű hangoskönyv felolvasásokat tartalmazó adatbázison, a LibriSpeech-en[1] mérik.

Különböző, alacsony hibaarányú modellek léteznek, de ez egyik lekiemelkedőbb közülök az Nvidia QuartzNet[2] modellje, mely alacsony számú paraméterek mellett képes igen nagy precíziót fenntartani. Míg a QuartzNet bonyolultsága, paramétereinek a száma csupán töredéke a legtöbb SOTA modellének, a WER száma megközelíti azokét (3.1-es táblázat).

A modellekben elterjedt a konvolúciós- (CNN) és rekurrens neurális rétegek (RNN) használata.

Model	**Aug**	**LM**	**clean WER(%)**	**other WER(%)**	**Million Parameters**
wav2letter++	Speed Perturb	ConvLM	3.26	10.47	208
LAS	SpecAugment	RNN	2.5	5.8	360
Time-Depth Separable Convolutions	Dropout, Label Smoothing	N/A (greedy)	5.36	15.64	37
		4-gram	4.21	11.87	
		ConvLM	3.28	9.84	
Multi-Stream Self-Attention	Speed Perturb	4-gram	2.93	8.32	23
		4-LSTM	2.20	5.82	
QuartzNet-15x5	Spec Cutout, Speed Perturb	N/A (greedy)	3.90	11.28	19
		6-gram	2.96	8.07	
		Transformer-XL	2.69	7.25	

3.1. táblázat. Egyes angol nyelven tanult SOTA beszédfelismerő modellek és paramétereik.

[1]LibriSpeech oldala: `http://www.openslr.org/12/`

[2]Az Nvidia ismertető oldala: `https://developer.nvidia.com/blog/develop-smaller-speech-recognition-models-with-nvidias-nemo-framework/`

3.4. Következtetések

Megoldásomban az Nvidia QuartzNet alapján indultam el, annak alacsony paraméterszámát figyelembe véve. Egy Pytorch alapú toolkit-et, az Nvidia NeMo-t (Neural Modules) használva kísérleteztem QuartzNet kombinációkkal.

Főként az ingyenesen hozzáférhető Mozilla Common Voice adatbázist használva tanítottam orosz nyelvre a legsikeresebbnek ítélt, 15x5 architektúrájú modellt. A modellt nem véletlenszerűen beállított súlyokkal, hanem transfer learning-et alkalmazva, egy előre tanított angol modellbeli súlyokkal inicializáltam.

Az eredmény finomítása végett próbálkoztam több tanítóadat beiktatásával, mellyel a validációs adathalmaz WER értéke csökkent, így javult. Sajnos a több tanítóadat lényeges megnövelte a tanítás idejét így erőforrások szűkében csökkentette a kísérletezésre jutó időt, ezért ezt a módszert csak kis mértékben alkalmaztam.

4. fejezet

Tervezés

4.1. Tervezés, döntési lehetőségek értékelése

4.1.1. Python

A Python napjaink egyik legelterjedtebb programozási nyelve. Egy könnyen programozható és átlátható, interpretált nyelv, melyben a megírt program kódot a Python értelmező sorról-sorra értelmezi és futtatja, nincs különválasztva a forrás és a fordított kód. Deep learning körökben is a Python nyelv dominál, rengeteg deep learning toolkit erre a nyelvre épít, így választásom a fejlesztéshez természetesen a Python nyelvre esett.

4.1.2. NeMo

A neurális hálók komplikáltságából adódik, hogy pontos és precíz implementációjuk igen bonyolult és időigényes. Emiatt célszerű a már meglévő, bejáratott és bizonyított lehetőségeket használni, így felgyorsítva a fejlesztési folyamatot. Több deep learing platform is található az internetet, mint például a Tensorflow vagy PyTorch. Az Nvidia által fejlesztett NeMo egy olyan összetett és folyamatosan karbantartott toolkit, melynek segítségével különböző mesterséges intelligencia alapú társalgási, beszéddel kapcsolatos applikációkat, alkalmazásokat készíthetünk.

Az alap koncepcióját a toolkit-nek az úgynevezett Neurális Modulok alkotják. Ezek a modulok lehetnek például adat rétegek, kódolók, dekóderek, nyelv modellek vagy loss függvények. A modulok ereje abban rejlik hogy egymáshoz viszonyítva

tetszőlegesen építhetők, cserélhetők, törölhetők. Egy egyszerű módosítással kicserélhető például a loss számításához használt függvény anélkül, hogy a kódot a többi modulban módosítani kéne.

Az 3.1-es táblán látható modellek közül az Nvidia fejlesztette a legkevésbé számításigényeset, mely figyelemreméltó eredményeket produkál. Ez a QuartzNet modell, ami az alacsony paraméterszáma ellenére SOTA eredményeket ér el. A felsorolt okokból kifolyólag a fejlesztési folyamat elvégzéséhez választásom a NeMo toolkit-re esett.

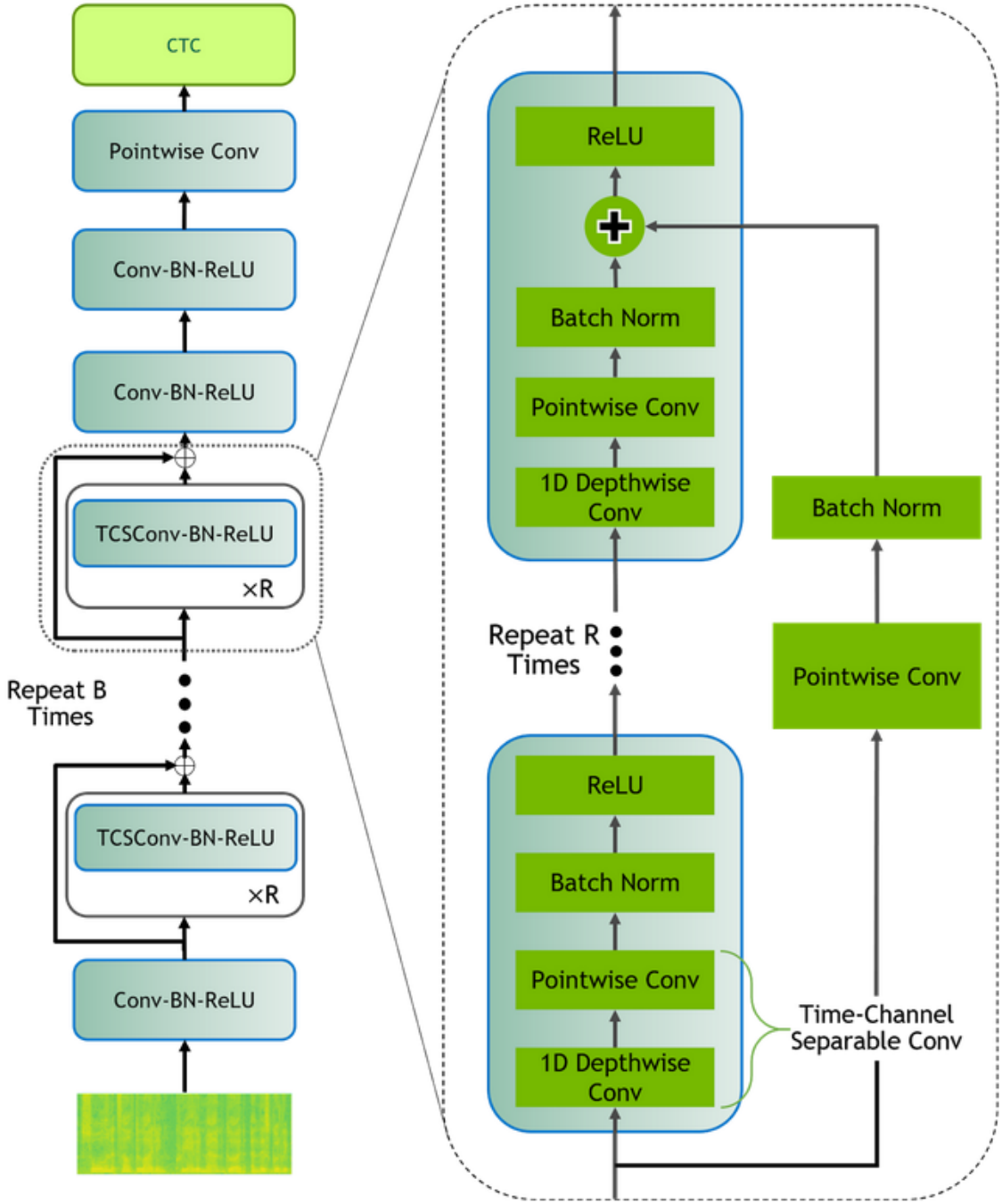
4.1.2.1. PyTorch

A NeMo a PyTorch nevű, Facebook-os kutatók által fejlesztett, mesterséges intelligencia fejlesztésére szakosodott nyílt forráskódú keretrendszerre épít. A PyTorch a TensorFlow-nál magasabb szintű, könnyebben kezelhető keretrendszer, mely egyre inkább elterjedtté válik a kutatók és fejlesztők körében.

4.1.2.2. QuartzNet

A kísérleteimet QuartzNet architektúrájú modellekkel tervezem elvégezni a korábban említett alacsony paraméterszáma és magas precizitása végett. Az architektúra (4.1-es ábra) nagyban hasonlít egy másik Nvidia által fejlesztett architektúrához, a Jasper-hez. A modell B darab blokkból áll. Egy opcionálisan beállítható dropout modul jelenhet meg a modellben, illetve CTC loss függvény számítás található benne. A már említett blokkok további, R darab blokkból állnak, melyek konvolúciós rétegekből, batch normalizálókból és ReLU aktivációs függvényből épülnek fel. Ezáltal a QuartzNet pontos felépítésére QuartzNet BxR alakban szokás hivatkozni. A legnagyobb Nvidia által készített a Quartznet 15x5, mely 18.9 millió paraméterből áll.

A QuartzNet a Jasper-től a konvolúciós rétegében tér el. A sima egy dimenziós konvolúciós réteg helyett, egy dimenziós time-channel separable konvolúciós rétegeket használ [**?**]. Ennek a fő tulajdonsága, hogy jóval kevesebb paramétert használ az eredeti koncepciónál. Ennek köszönhetően mélyebb hálókat lehet készíteni nagyobb konvolúciós filter-ekkel anélkül, hogy a a paraméterek száma, ezáltal a szükséges számítása kapacitás túlzottan megnövekedne.

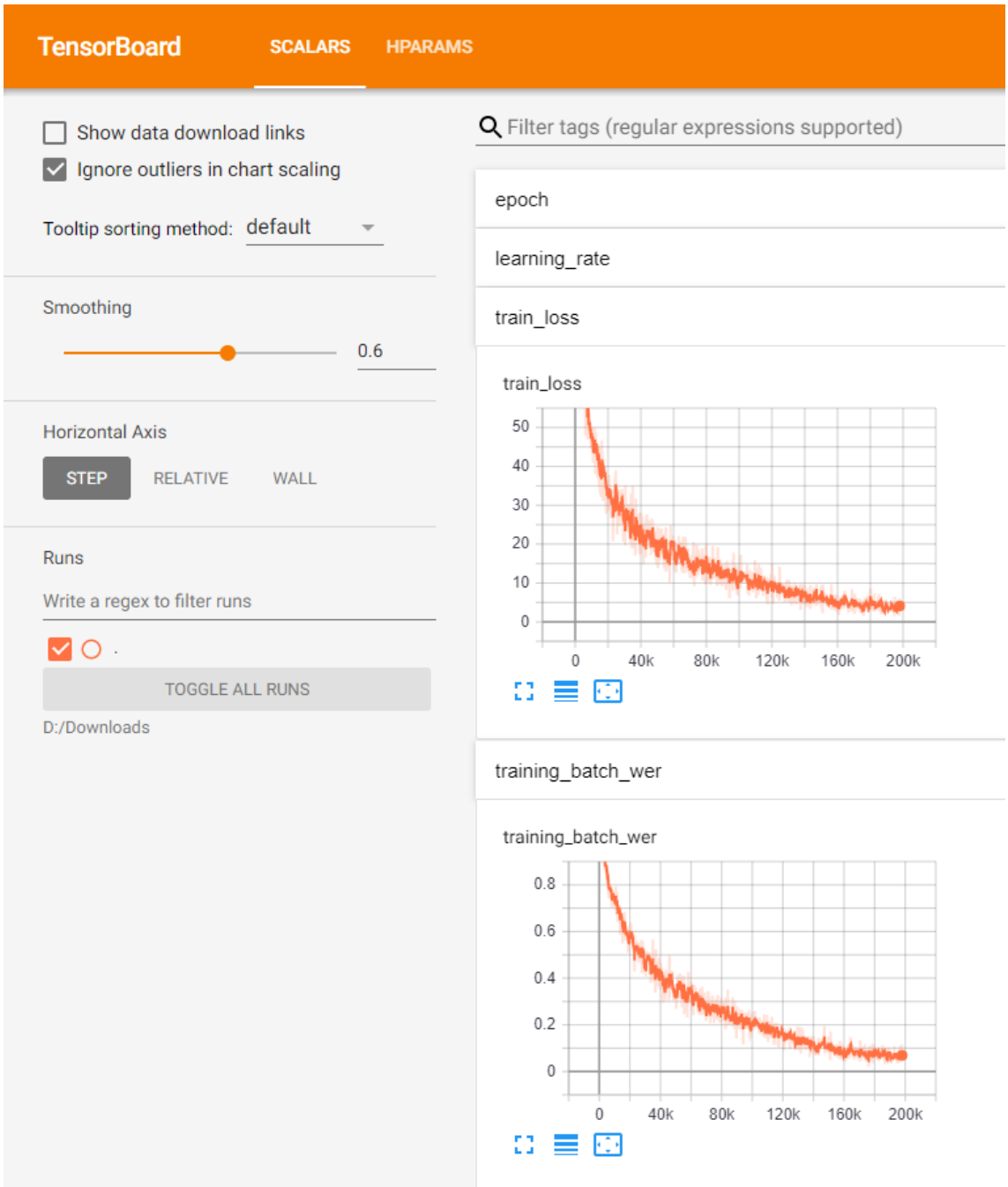


4.1. ábra. QuartzNet architektúra. [**?**]

4.1.3. TensorBoard

Az eredmények kiértékeléséhez és nyomonkövetéséhez a TensorBoard-ot, a TensorFlow egy vizualizációs toolkit-jét használom. Több funkcióval is rendelkezik, mint például a súlyok időbeni változásának vizualizációja, illetve a tanítóadatok, képek, szöveg vagy hang anyag kijelzése. Számomra a legfontosabb tulajdonsága az egyes metrikák, a loss és a pontosság, WER, értékének változása.

A TensorBoard könnyedén indítható a következő parancs kiadásával: *tensorboard – –logdir /lightning – logs* , ahol a logdir kapcsoló adja meg melyik könyvtárban keressen TensorBoard kompatibilis log-okat, event-eket. A szükséges event-ek előállítása könnyedén végezhető a NeMo toolkit segítségével, mely támogatja a megfelelő callback függvények használatát, amik az event-ek batch-enkénti frissítsét tesznek lehetővé.



4.2. ábra. Lokálisan futtatott TensorBoard.

4.2. Angol, majd orosz

Kezdetben a tesztelést az egyik leggyakoribb nyelven, az angolon, végeztem. Ennek oka főként az, hogy megbizonyosodjak a modellem és a NeMo beállításainak sikerességéről, hogy megfelelő eredményeket tudok elérni elterjedt, kisebb adatbázisokon. Másfelől a modellem hiperparamétereit, mint például a learning rate, optimalizáltam, és az orosz nyelven történő kísérletezésnek a legbiztatóbb beállításokkal tudtam nekiállni.

Az angol nyelvű eredményeket viszonyításként is fel tudtam használni az végső orosz eredményekkel való összehasonlításánál, ahol a nagyságrendileg azonos mennyiségű orosz adatoknak az angoléhoz hasonló eredményeket produkálhattak.

4.2.1. Google Colab

A NeMo-val való ismerkedéshez felhasználtam az Nvidia által írt jupyter notebook-okat[1], melyeket könnyedén lehet az ingyenes használható Google Colab-ból[2] futtatni. A Colab ingyenes hozzáférést nyújt GPU erőforrásokhoz, ahol egyenesen a böngészőből futtatható a Python kód és az eredményeket akár le is lehet tölteni.

Sajnos a Colab bizonyos órán belül felfüggeszti a munkamenetet, illetve a nagy mennyiségű adatok munkamenetenkénti le- és feltöltése időigényes. Ezért a rövidebb adatbázisokon, mint például az AN4-en, túl nem jelent a tesztelésen és toolkit-tel való ismerkedésen túl megoldást.

4.3. Adatbázisok

A megfelelő modell megtalálásához két darab angol nyelvű adatbázist is használtam. Mikor elégedett voltam az eredményekkel és megbizonyosodtam, hogy sikeresen beállítottam a NeMo toolkit-et is, áttértem az orosz nyelvű modell fejlesztésére.

Az adatbázisokon belül szükséges három különböző típusú egymástól független adathalmazt megállapítani. Az egyik a tanító adathalmaz, ezeket a hanganyagokat közvetlen a háló tanítására használjuk. Fontos egy validációs vagy teszt adathalmaz

[1] Tutorial notebook, `https://github.com/NVIDIA/NeMo/blob/main/tutorials/asr/01_ASR_with_NeMo.ipynb`

[2] Google Colab, `https://colab.research.google.com/notebooks/intro.ipynb`

is, amit tanítás közben használunk kiértékelésre, a tanítás általános pontosságának értékelésére. A harmadik dev adathalmaz pedig azért szükséges, mert a validációs adathalmaz pontosságát igyekszünk javítani, majd amikor ezzel elégedettek vagyunk egy tőle független adathalmazzal vizsgálhatjuk meg a végső pontosságot.

4.3.1. Adatok előfeldolgozása

Magukat a hangfájlokat is meg kell vizsgálni, egységesíteni kell a tanítás előtt. A leggyakrabban használt kiterjesztés a .wav, így én is ennek a használata mellett döntöttem. Szükség esetén az egyes hangfájlokat át kell konvertálni ebbe a formátumba a tanítás, kiértékelés előtt.

A hanganyagok mintavételezése sem elhanyagolható. Míg a túl alacsony mintavételezés minőségi romlást, ezáltal rosszabb eredményeket okozhat, a túl magas mintavételezés lassítja a tanítási folyamatot és akár szintén rosszabb eredményeket produkálhat.

Az adatbázisok esetén megvizsgálandó, hogy hány beszélő hangja található benne. Minél több van benne annál általánosabban alkalmazható jobb eredményekkel.

4.3.2. AN4

Az AN4[**?**] vagy ”census” angol nyelvű beszéd adatbázist 1991-ben vették fel a Carnegie Mellon Egyetemen. A különböző beszélők betűzve mondanak olyan adatokat mint születési dátum, telefonszám, név vagy egyéb véletlenszerűen generált, előre definiált kontroll szavakat.

Az adatbázis két különböző részre van osztva, az első a tanítást szolgálja, míg a második része a tesztelést. Ez előbbi 50 percnyi beszédet tartalmaz, míg a tesztelésre szolgáló rész 6 percet. Rövidségéből adódóan gyorsan tanítható és értékelhető, mely ideálissá teszi gyors kalibrálási, beállítási feladatok megoldására.

4.3.3. LibriSpeech

A LibriSpeech egy több száz órát tartalmazó, szintén angol nyelű, hangos könyv gyűjtemény. Ez az egyik leginkább használt adatbázis, melyet előszeretettel alkalmaznak tudományos, kutatási anyagok eredményeinek ismertetésekor is.

Az elérhető hanganyag fele tiszta, könnyen érthető, míg másik fele zajosabb körülmények közt lett felvéve vagy kevésbé kivehető. Külön tartalmaz tanító, teszt és dev adathalmazokat.

4.3.4. Mozilla Common Voice

A Mozilla Common Voice[3] egy Mozilla által indított projekt. Célja, hogy különböző nyelveken, így oroszul is, a közösség erejét felhasználva gyűjtsön anonim hanganyagokat. Bárki felveheti a saját hangját, előre meghatározott szövegeket felolvasva, illetve érvényesíthet mások által felolvasott szövegeket. A biztonság kedvéért egy szöveget két személynek is jóvá kell hagynia.

Orosz hangból durván 100 órányi áll rendelkezésre, de tüzetesebb vizsgálat után észrevettem, hogy ebből 10-10 óra a teszt és dev adathalmaz mérete, míg 20 óra a tanító adathalmazé. A maradék 60 órából több azért nincs használva, mert megismételt szöveget olvasnak fel benne, vagy rövid hanganyagokat tartalmaz. A tanító adathalmazt sikerült feldúsítsam 45 óra környékére azáltal, hogy belevettem a 100 órányi hangból az összes olyan hangot, melyek nem voltak benne sem a teszt sem a dev adathalmazokban.

4.3.5. További orosz adatbázisok

Mivel a Mozilla Common Voice orosz nyelvű adatbázisa legfeljebb 45 órányi tanítóadattal rendelkezik, míg a LibriSpeech-é mely eredményeihez viszonyítani szeretnénk 100 órán lett tanítva, ezért több tanítóadatot is be kellett vonni. Egy ingyenesen hozzáférhető github-on található projektben[4] több adatbázis is található, többnyire .opus formátumban. Elsősorban a radio2 adatbázissal dolgoztam.

[3]Mozilla Common Voice weboldala, `https://voice.mozilla.org/`

[4]Ingyenes orosz adatbázisok, `https://github.com/snakers4/open_stt/blob/master/README.md`

Fontos megemlíteni a Mozzila Common Voice és egyéb rádión található hanganyagok közötti lehetséges különbségeket. Míg előbbi többnyire bediktált, felolvasott szöveget tartalmaz különböző minőségben és tempóban, utóbbi kötetlenebb, tisztább minőségű beszélgetéseket is tartalmazhat.

5. fejezet

Implementáció

5.1. Tanítóadatok előkészítése

A letöltött tanítóadatokat NeMo által értelemzhető formátumúra kellett hozni. Ez magába foglalta a fájlok konvártálását .wav formátumba. Az egyes tanítóadatok más-más formátumban (.opus, .flac, .mp3) voltak eredetileg, illetve eltérő fájstruktúrákba voltak szervezve.

A konvertáláshoz a fentiek miatt a python os könyvtárának walk metódusát használtam mellyel egy mappát rekúrzívan be lehet járni. A konvertáláshoz az ffmpeg szoftvert használtam, mivel segítségével több formátum is könnyedén konvertálható .wav formátumba.

A fájlok átalakításán kívül szükséges volt az adatok .json fájljainak elkészítése, mely megmondja, hogy melyik hangfájl hol található, milyen hosszú és mi hangzik el rajta. A NeMo-nak ezután csupán az egyes manifest-eket, .json fájlokat, kellett megadni, azzal a kiegészítéssel, hogy melyiket kívánjuk tanító vagy teszt adathalamznak használni, és a további előfeldolgozási lépéseket már autómatikusan elvégezte. Természetesen több információ tárolását is támogatja a NeMo, de ezek az adatok elengedhetetlenek.

5.2. Modellek kimentése és betöltése

A tanított modelleket el kell menteni későbbi használat végett. Két módszert is alkalmaztam. Az egyik a logoláshoz hasonlóan callback-et használ, mindig a

legoptimálisabb eredményt elért epochot menti ki .ckpt fomrátumban, míg az utóbbi módszer a tanítás végén menti ki a modellt .nemo formátumban.

A mentésre a későbbi felhasználás meleltt azért is van szükség, mert a hosszú tanításokat, melyek akár 48 órán át is tarthatnak, szükséges lehet megszakítani. A megszakítás után fontos úgy betölteni a modellt, hogy az továbbtanítható legyen. A NeMo külön kezeli a modellt és a tanítást végző trainert, melykbe külön külön szükséges betölteni a folytatni kívánt checkpoint-ot.

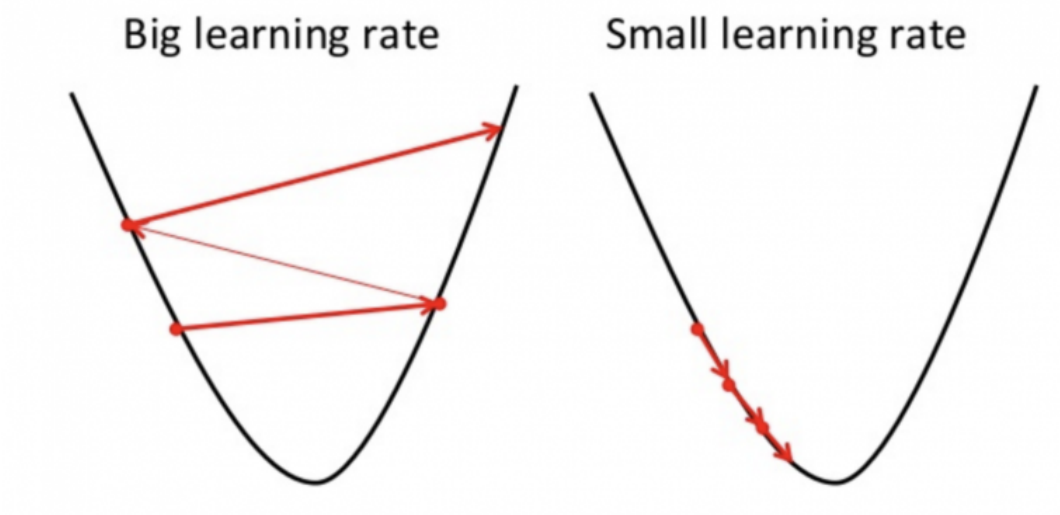
5.3. Kalibráció angol nyelv segítségével

Első lépésként egy kisebb adatbázison, az AN4-en teszteltem a NeMo keretrendszert, a Google Colab segítségével. Az alap paraméterekkel és konfigurációval az elért eredmények elmaradtak a referencia eredményekéhez képest. Az AN4-es tanítás során QuartzNet 5x1-es architektúrájával dolgoztam a kevés mennyiségű adat végett.

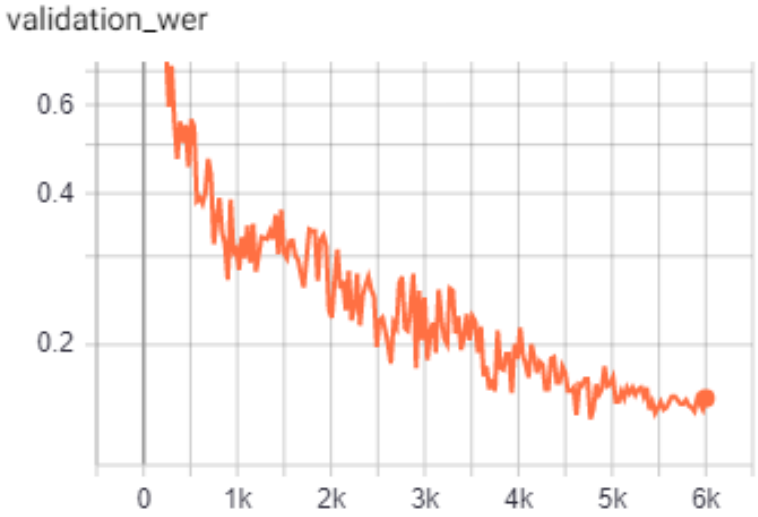
A pontosság növelése érdekében az egy gyakran használt módszer az epoch szám növelése. Egy epoch reprezentálja az összes tanítóadaton való kiértékelést és a súlyok javítását a hiba tükrében. Magas epoch szám jobb eredményeket produkál, de vigyázni kell, mert túlzott epoch szám esetében megjelenhet az úgy nevezett túltanítás jelenség. Ez annyit tesz, hogy a túlzottan is megtanulja a modell a tanítóadatokat, azokra nagyon jó eredményt ad, míg általánosan egyre rosszabbul teljesít.

Egy másik fontos paraméter aminek a módosításával növelni lehet az elért pontosságot a learning rate. A learning rate felel a hiba nyomán felmerült tanítás, azaz súlyok korrekciójának mértékéért. Túl nagy lépések, javítások, esetén a loss értéke, amit csökkenteni szeretnénk, túl nagy ugrásokat végez, akár a rossz irányba. Túl alacsony érték esetén lassan éri el a loss a minimum pontját, legyen az globális vagy lokális, ahogy az az 51-es ábrán látható. Szintén nem kívánt eredmény az alacsony learning rate esetén, hogy a loss értéke ideje korán beragad egy lokális minumum értékben, ami egy nagyobb lépéssel áthidalható lenne.

A fentiek tükrében növeltem az epoch számot és a learning rate-et, 200-ra és 0.02-re. Sikerült egy optimálisabb eredményt elérjek, így a felhasznált paraméterekkel tovább tudtam indulni egy nagyobb adatbázison való tesztelésre.



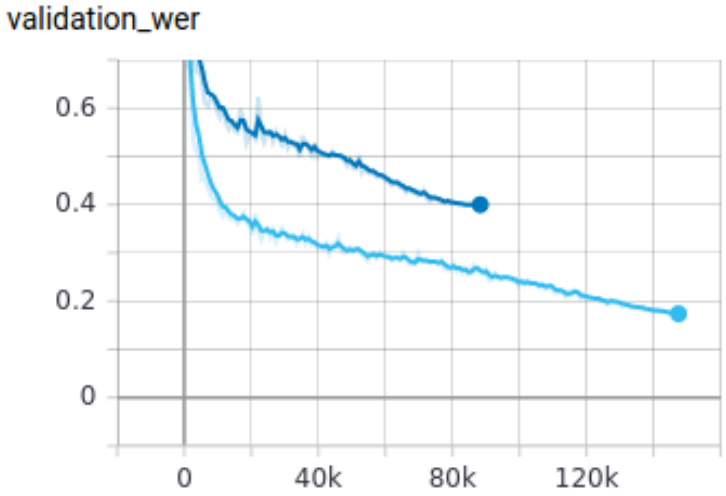
5.1. ábra. Learning Rate fontossága, `https://builtin.com/data-science/gradient-descent`.



5.2. ábra. AN4 adatbázis validációs adathalmazán elért eredmények.

Arról, hogy a beállításaim tényleg megfelelőek a LibriSpeech 100 órás adathalmazán bizonyosodtam meg. Azért nem ezzel kezdtem, mert 100 órányi hanganyag tanítása jóval tovább tart, így minél kevesebbszer akartam végigfuttatni a tanítást az optimalizálás érdekében.

A LibriSpeech-hez egy nagyobb méretű architektúrát használtam. A mélyebb hálók jellemzően pontosabb eredményeket képesek elérni a paraméterszám növelése mellett. Összevetettem az eredeti, 5x1-es modellt a nagyobb, 12x1-es QuartzNet modellel és a korábban optimálisnak vélt epoch számmal és egy magasabb learning rate-el. A különbség az 5.2-es ábrán látható.



5.3. ábra. LibriSpeech eredmények QuartzNet 5x1 (sötétkék) és 12x1 (világoskék) architektúrákon.

5.4. Orosz nyelvű hálók

Az orosz nyelvű implementációhoz már jó kiindulópontot jelentett a LibriSpeech-nél is alkalmazott architektúra. Az Mozilla Common Voice orosz adatbázisa átesett a szükséges formázásokon, és legeneráltam belőle a megfelelő .json fájlt. A legenerált fájlban odafigyeltem, hogy csak cirill betűk szerepeljenek, illetve szükség esetén kisbetűsítettem a szöveget.

5.4.1. NeMo config fájl

A NeMo a modellek leírásához config fájlokat használ. A fájlokban minden modellt és tanítást leíró paramétert meg lehet adni. Ezek közé tartozik a modell architektúrája, a használandó aktivációs rétegek, a learning rate és az epoch szám vagy az optimalizáló típusa.

Előnye, hogy könnyen módosítható és a programkódtól független. Megadhatók üresen hagyott paraméterek is, melyeket ‘??¿el lehet jelölni. Ezeket később programkódból ki lehet tölteni, mint például a használandó manifest fájl elérési útját.

5.4.2. Szükséges módosítások

A config fájlban két dolgot kellett átállítani az orosz nyelv használata előtt. Az egyik maguk a betűk, label-ek voltak, mivel az orosz ábécé merőben eltér az angolétól. A cirill karakterek könnyen kinyerhető a google translate orosz nyelvű virtuális billentyűzete segítségével.

A másik átállítandó paraméter a config fájl AudioToTextDataLayer részében a leiratok normalizálására szolgáló rész[1]. Erre azért van szükség, mert alapból elvégezné a modell, de szeretnénk elkerülni, mivel angol nyelvre lett tervezve így egyéb nyelveken nem működne megfelelően.

5.4.3. Architektúrák

A kísérletezéshez 2 különböző architektúrát, 12x1 és 15x5-et használtam. A választáson fő szempontja az volt, hogy ilyen architektúrákhoz rendelkeztem előre tanított angol nyelvű modellekkel. Ezeket később a transfer learning-nél használtam fel, és összehasonlítottam a véletlenszerűen inicializált súlyú modellek eredményeivel.

A tanításokat a Mozilla és Radio2 adatbázisán is futtattam. A Radio2-es adatbázis túl nagynak bizonyult, a hanganyagok méretét 8.7 másodpercben maximalizáltam, szemben a megszokott 16.7 másodperccel, így az összes tanítóadat 193.75 órát tett ki.

Fontos kiemelni, hogy a tanítás közbeni pontosság teszteléséhez, azaz a validációhoz mindegyik esetben a Mozilla adatbázishoz tartozó tesztadatokat használtam. A pontosságot külön-külön megvizsgáltam a Radio2-es adatbázisból leválasztott, tanításhoz fel nem használt hanganyagokon is.

5.4.4. Transfer learning

A Transfer learning-hez előre betanított modelleket kellett szerezzek. A QuartzNet 15x5 architektúrájú modell könnyen elérhető volt, az Nvidia által szabad rendelkezésre volt bocsájtva. A 12x1-es architektúrát konzulensem Dr. Mihajlik Péter bocsájtott rendelkezésemre. Ez a modell kevesebb adaton és rövidebb ideig

[1] `https://github.com/NVIDIA/NeMo/issues/234`

tanult mint a 15x5-ös, de jobb kiindulópont volt, aminek előállítására lehetőségem lett volna.

A modell betöltése hasonlóan történt a checkpoint-ból való folytatás betöltéséhez, azzal a különbséggel, hogy nem a teljese modellt használtam, csupán az encoder részét. Ez fontos különbség, hiszen a decoder használhatatlan lett volna, mivel az angol nyelvre lett tanítva. Így a decoder-hez véletlenszerűen inicializált súlyokat használtam, és a modell encoder részének súlyai kerültek átemelésre.

A tanítás ezen felül a megszokott szerint zajlott a megfelelő config fájl betöltésével, tanítóadatok és trainer beállításával, a súlyok kimentésével.

6. fejezet

Eredmények ismertetése és összefoglalás

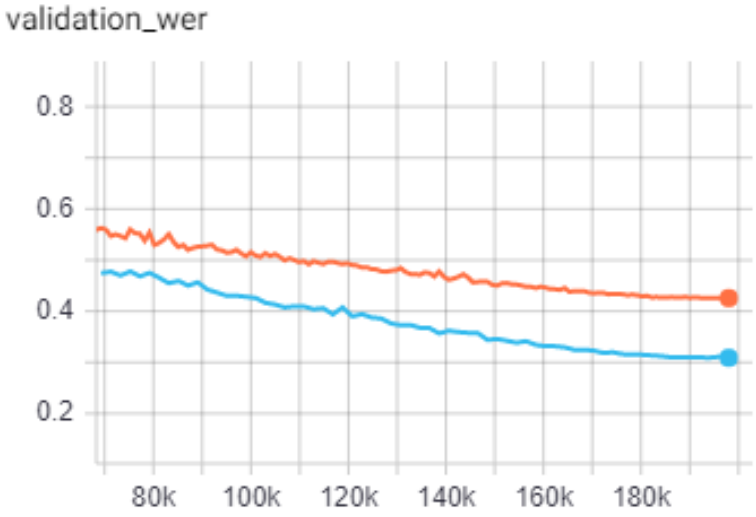
6.1. Különböző architektúrák összehasonlítása

A nagyobb modellek pontosabb eredményeket és alacsonyabb WER-t értek el adatbázistól függetlenül. Viszont ez az idő drasztikus növekedésével járt együtt. 12x1-es architektúra nagyjából 5 millió paramétert, míg a 15x5-ös architektúra 18.9 millió paramétert használt. Futtatási idő esetén ez azt eredményezte, hogy fele annyi epoch-hoz másfélszer annyi időre volt szüksége, ami 3-szoros sebesség csökkenést eredményez. Következtetésképp lehetőség szerint érdemes minél több erőforrással dolgozni, az idő csökkentése végett és a mélyebb háló útján elindulni a precízebb végeredmény céljából.

A 6.1-es ábrán látható viszont a mélyebb, összetettebb háló előnye, hiszen ekvseebb epoch alatt pontosabb eredményeket, alacsonyabb validation WER értéket sikerült elérni vele, mint a 12x1 architektúrával.

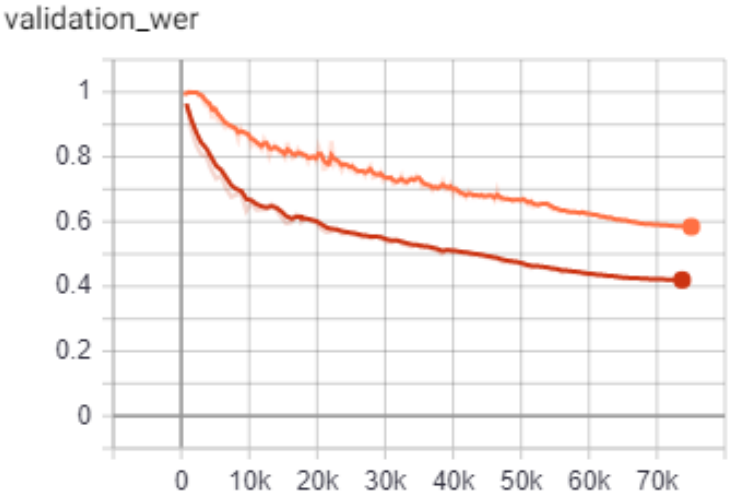
6.2. Véletlenszerűen inicializált vs transfer learning

Minden esetben megállapítható, hogy a transfer learning nagyban javította az elért eredményeket, annak ellenére, hogy az átvett modell súlyai egy másik ábécével,



6.1. ábra. 15x5-ös - narancs és 12x1-es - kék architektúrák.

másik nyelven lettek tanítva. Látható, hogy az emberek által generált hangok, a nyelvek struktúrája és logikája erősen összefügg.



6.2. ábra. 12x1-es architektúrák validation WER étékei pretrained - vörös és random inicializált - narancs súlyokkal.

6.3. Mozilla Common Voice vs Radio2

A két adatbázis fő különbsége az adatok mennyisége volt. Látható, hogy ...

Megjegyzendő, hogy a hanganyagok minősége is eltérő, ez tetten érhető...

6.4. Javítási lehetőségek

A jobb WER eredmények végett alkalmazhatóak a neurális hálók esetén a gyakran használt augmentációs módszerek. Ezek az elérhető hanganyag mennyiséget dúsítják fel különböző módszerekkel. Ez lehet az eredeti hanganyag torzítása, levágása, zaj hozzáadása vagy sebességének módosítása.

Nagyobb adatbázisok alkalmazása jobb általánosítást eredményez, ami a validációs WER érték csökkenésével jár együtt. Érdemes a különböző adatbázisokat összefogni és akár több ezer órányi adaton tanítani.

A sebesség növeléséhez a NeMo toolkit több lehetőséget is kínál. Az egyik a Mixed Precision[1], a másik módszer több GPU bevonása a tanítási folyamatba. Utóbbi esetben több, előre meghatározott GPU közt osztódik el az adat és kerül gyorsabb kiértékelésre.

Lehetséges még a tanítóadat pontos behatárolása. Mivel tanítás közben, a tanítási batch-ekbe különböző hosszúságú hangfájlok kerülnek betöltésre, így például egy 16-os batch méret esetén a GPU hamar kiértékeli az 5 másodperc hosszú hangfájlokat, de nem tud tovább haladni, míg a leghosszabbat, például egy 14 másodperceset nem értékelt ki. Ezen probléma elkerülése, és a GPU jobb kihasználása végett célszerű nagyjából egyforma hosszúságú adatokat felhasználni a tanításhoz.

Természetesen javulást eredményezhet a hiperparaméterekkel való kísérletezés, azok finomítása. Különböző adatok, nyelvek vagy architektúrák más-más paraméterekkel működnek jobban, nincsenek bevett számok, legfeljebb megközelítőleg.

6.5. Végső gondolatok

A beszédfelismerés már régóta velünk van, de egyre inkább elterjed és bekerül a köztudatba, mindennapjainkba a pontosság növekedésének következtében. Ehhez nagyban hozzájárul a klasszikus HM-től eltérő, napjainkban egyre inkább felkapott megközelítés a mély neuron háló alapú, end-to-end beszédfelismerés. Egyre több

[1]Mixed Precision lehetőség magyarázat az NVIDIA oldalán: `https://docs.nvidia.com/deeplearning/nemo/neural_mod_bp_guide/index.html`

toolkit, és platform jelenik meg, amelyek könnyítik a neurális hálókkal való munkát, azok fejlesztését.

Az angoltól eltérő nyelvű automatikus beszédfelismerő rendszerek fejlesztése és kutatása fontos terület, hiszen a különböző nemzetek polgárai a saját nyelvükön akarnak beszélni. Segítséget nyújthatnak ebben a folyamatban a precíz, angol nyelvű modellek a transfer learning által, aminek következtében drasztikus javulás érhető el az új modell WER pontosságánál.

A nap mint nap megjelenő újabb architektúrák implementálásával a pontosság tovább növelhető, így korábban elképzelhetetlen, az embernél is nagyobb pontosságú beszédfelismerés valósítható meg, mely az élet számos területén használható fel.

7. fejezet

LaTeX-eszközök

7.1. A szerkesztéshez használatos eszközök

Ez a sablon TeXstudio 2.8.8 szerkesztővel készült. A TeXstudio egy platformfüggetlen, Windows, Linux és Mac OS alatt is elérhető LaTeX-szerkesztőprogram számtalan hasznos szolgáltatással (7.1. ábra). A szoftver ingyenesen letölthető[1].

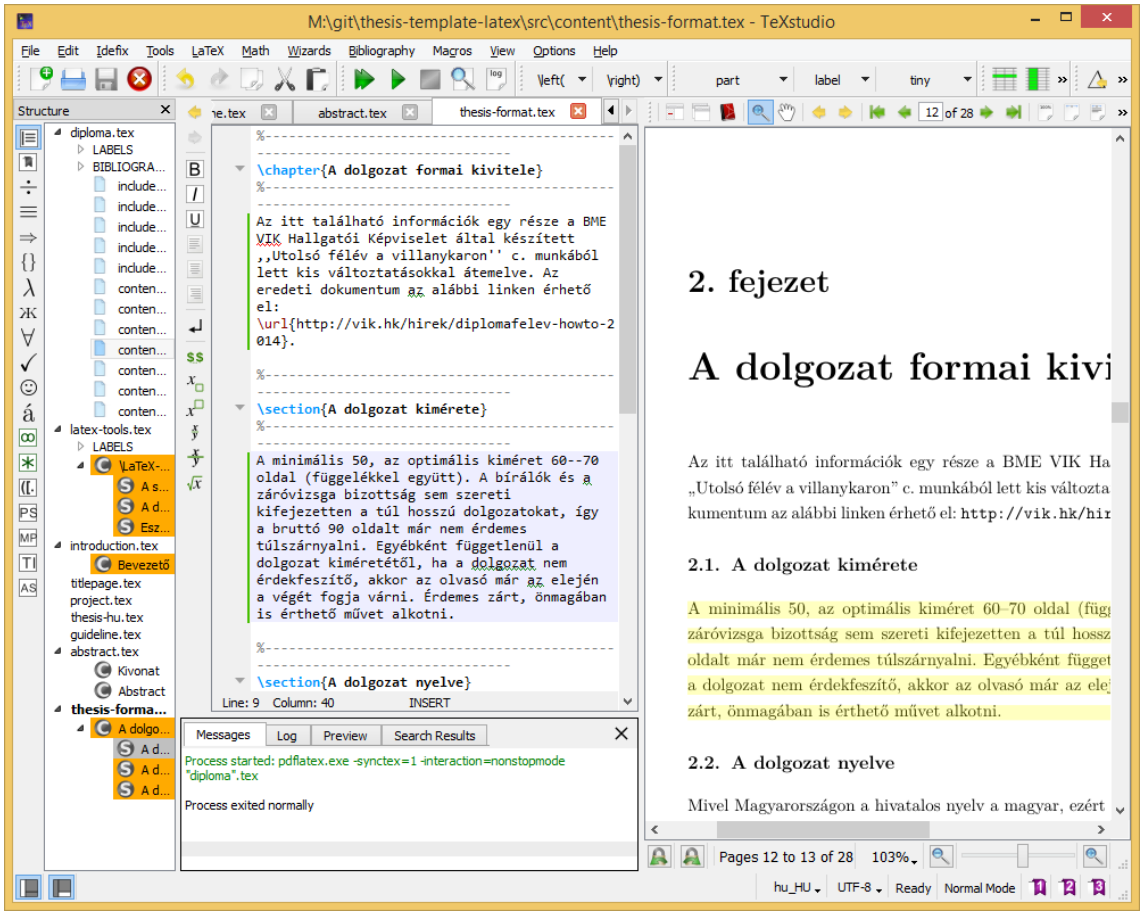
A TeXstudio telepítése után érdemes még letölteni a magyar nyelvű helyesírásellenőrző-szótárakat hozzá. A TeXstudio az OpenOffice-hoz használatos formátumot tudja kezelni. A TeXstudio beállításainál a `General` fülön a `Dictionaries` résznél tudjuk megadni, hogy melyik szótárat használja.

Egy másik használható Windows alapú szerkesztőprogram a LEd[2] (LaTeX Editor), a TeXstudio azonban stabilabb, gyorsabb, és jobban használható.

7.2. A dokumentum lefordítása Windows alatt

A TeXstudio és a LEd kizárólag szerkesztőprogram (bár az utóbbiban DVI-nézegető is van), így a dokumentum fordításához szükséges eszközöket nem tartalmazza. Windows alatt alapvetően két lehetőség közül érdemes választani: MiKTeX (`http://miktex.org/`) és TeX Live (`http://www.tug.org/texlive/`) programcsomag. Az utóbbi működik Mac OS X, GNU/Linux alatt és Unix-

[1] A TeXstudio hivatalos oldala: `http://texstudio.sourceforge.net/`
[2] A LEd hivatalos oldala: `http://www.latexeditor.org/`



7.1. ábra. A TeXstudio LaTeX-szerkesztő.

származékokon is. A MiKTeX egy alapcsomag telepítése után mindig letölti a használt funkciókhoz szükséges, de lokálisan hiányzó TeX-csomagokat, míg a TeX Live DVD ISO verzóban férhető hozzá. Ez a dokumentum TeX Live 2008 programcsomag segítségével fordult, amelynek DVD ISO verziója a megadott oldalról letölthető. A sablon lefordításához a disztribúcióban szereplő `magyar.ldf` fájlt a `http://www.math.bme.hu/latex/` változatra kell cserélni, vagy az utóbbi változatot be kell másolni a projekt-könyvtárba (ahogy ezt meg is tettük a sablonban) különben anomáliák tapasztalhatók a dokumentumban (pl. az ábra- és táblázat-aláírások formátuma nem a beállított lesz, vagy bizonyos oldalakon megjelenik alapértelmezésben egy fejléc). A TeX Live 2008-at még nem kell külön telepíteni a gépre, elegendő DVD-ről (vagy az ISO fájlból közvetlenül, pl. DaemonTools-szal) használni.

Ha a MiKTeX csomagot használjuk, akkor parancssorból a következő módon tudjuk újrafordítani a teljes dokumentumot:

```
$ texify -p thesis.tex
```

A `texify` parancs a MiKTex programcsomag `miktex/bin` alkönyvtárában található. A parancs gondoskodik arról, hogy a szükséges lépéseket (fordítás, hivatkozások generálása stb.) a megfelelő sorrendben elvégezze. A `-p` kapcsoló hatására PDF-et generál. A fordítást és az ideiglenes fájlok törlését elvégezhetjük a sablonhoz mellékelt `manual_build.bat` szkript segítségével is.

A TEX-eszközöket tartalmazó programcsomag binárisainak elérési útját gyakran be kell állítani a szerkesztőprogramban, például TeXstudio esetén legegyszerűbben az `Options / Configure TeXstudio... / Commands` menüponttal előhívott dialógusablakban tehetjük ezt meg.

A PDF-LATEX használata esetén a generált dokumentum közvetlenül PDF-formátumban áll rendelkezésre. Amennyiben a PDF-fájl egy PDF-nézőben (pl. Adobe Acrobat Reader vagy Foxit PDF Reader) meg van nyitva, akkor a fájlleírót a PDF-néző program tipikusan lefoglalja. Ilyen esetben a dokumentum újrafordítása hibaüzenettel kilép. Ha bezárjuk és újra megnyitjuk a PDF dokumentumot, akkor pedig a PDF-nézők többsége az első oldalon nyitja meg a dokumentumot, nem a legutóbb olvasott oldalon. Ezzel szemben például az egyszerű és ingyenes Sumatra PDF nevű program képes arra, hogy a megnyitott dokumentum megváltozását detektálja, és frissítse a nézetet az aktuális oldal megtartásával.

7.3. Eszközök Linuxhoz

Linux operációs rendszer alatt is rengeteg szerkesztőprogram van, pl. a KDE alapú Kile jól használható. Ez ingyenesen letölthető, vagy éppenséggel az adott Linux-disztribúció eleve tartalmazza, ahogyan a dokumentum fordításához szükséges csomagokat is. Az Ubuntu Linux disztribúciók alatt például legtöbbször a `texlive-*` csomagok telepítésével használhatók a LATEX-eszközök. A jelen sablon fordításához szükséges csomagok (kb. 0,5 GB) az alábbi paranccsal telepíthetők:

```
$ sudo apt-get install texlive-latex-extra texlive-fonts-extra texlive-fonts-recommended
    texlive-xetex texlive-science
```

Amennyiben egy újabb csomag hozzáadása után hiányzó fájlra utaló hibát kapunk a fordítótól, telepítenünk kell az azt tartalmazó TeX Live csomagot. Ha pl. a `bibentry` csomagot szeretnénk használni, futtassuk az alábbi parancsot:

```
$ apt-cache search bibentry
texlive-luatex - TeX Live: LuaTeX packages
```

Majd telepítsük fel a megfelelő TeX Live csomagot, jelen esetben a texlive-lualatex-et. (Egy LaTeX csomag több TeX Live csomagban is szerepelhet.)

Ha gyakran szerkesztünk más LaTeXdokumentumokat is, kényelmes és biztos megoldás a teljes TeX Live disztribúció telepítése, ez azonban kb. 4 GB helyet igényel.

```
sudo apt-get install texlive-full
```

8. fejezet

A dolgozat formai kivitele

Az itt található információk egy része a BME VIK Hallgatói Képviselet által készített „Utolsó félév a villanykaron” c. munkából lett kis változtatásokkal átemelve. Az eredeti dokumentum az alábbi linken érhető el: `http://vik.hk/hirek/diplomafelev-howto-2015`.

8.1. A dolgozat kimérete

Szakdolgozat esetében minimum 30, 45 körüli ajánlott oldalszám lehet az iránymutató. De mindenképp érdemes rákérdezni a konzulensnél is az elvárásokra, mert tanszékenként változóak lehetnek az elvárások.

Mesterképzésen a Diplomatervezés 1 esetében a beszámoló még inkább az Önálló laboratóriumi beszámolókhoz hasonlít, tanszékenként eltérő formai követelményekkel, – egy legalább 30 oldal körüli dolgozat az elvárt – és az elmúlt fél éves munkáról szól. De egyben célszerű, ha ez a végleges diplomaterv alapja is. (A végleges 60-90 oldal körülbelül a hasznos részre nézve)

8.2. A dolgozat nyelve

Mivel Magyarországon a hivatalos nyelv a magyar, ezért alapértelmezésben magyarul kell megírni a dolgozatot. Aki külföldi posztgraduális képzésben akar részt venni, nemzetközi szintű tudományos kutatást szeretne végezni, vagy multinacionális cégnél akar elhelyezkedni, annak célszerű angolul megírnia diplomadolgozatát. Mielőtt a

hallgató az angol nyelvű verzió mellett dönt, erősen ajánlott mérlegelni, hogy ez mennyi többletmunkát fog a hallgatónak jelenteni fogalmazás és nyelvhelyesség terén, valamint – nem utolsó sorban – hogy ez mennyi többletmunkát fog jelenteni a konzulens illetve bíráló számára. Egy nehezen olvasható, netalán érthetetlen szöveg teher minden játékos számára.

8.3. A dokumentum nyomdatechnikai kivitele

A dolgozatot A4-es fehér lapra nyomtatva, 2,5 centiméteres margóval (+1 cm kötésbeni), 11–12 pontos betűmérettel, talpas betűtípussal és másfeles sorközzel célszerű elkészíteni.

Annak érdekében, hogy a dolgozat külsőleg is igényes munka benyomását keltse, érdemes figyelni az alapvető tipográfiai szabályok betartására [**?**].

9. fejezet

A LaTeX-sablon használata

Ebben a fejezetben röviden, implicit módon bemutatjuk a sablon használatának módját, ami azt jelenti, hogy sablon használata ennek a dokumentumnak a forráskódját tanulmányozva válik teljesen világossá. Amennyiben a szoftverkeretrendszer telepítve van, a sablon alkalmazása és a dolgozat szerkesztése LaTeX-ben a sablon segítségével tapasztalataink szerint jóval hatékonyabb, mint egy WYSWYG (*What You See is What You Get*) típusú szövegszerkesztő esetén (pl. Microsoft Word, OpenOffice).

9.1. Címkék és hivatkozások

A LaTeX dokumentumban címkéket (`\label`) rendelhetünk ábrákhoz, táblázatokhoz, fejezetekhez, listákhoz, képletekhez stb. Ezekre a dokumentum bármely részében hivatkozhatunk, a hivatkozások automatikusan feloldásra kerülnek.

A sablonban makrókat definiáltunk a hivatkozások megkönnyítéséhez. Ennek megfelelően minden ábra (*figure*) címkéje `fig:` kulcsszóval kezdődik, míg minden táblázat (*table*), képlet (*equation*), fejezet (*section*) és lista (*listing*) rendre a `tab:`, `eq:`, `sec:` és `lst:` kulcsszóval kezdődik, és a kulcsszavak után tetszőlegesen választott címke használható. Ha ezt a konvenciót betartjuk, akkor az előbbi objektumok számára rendre a `\figref`, `\tabref`, `\eqref`, `\sectref` és `\listref` makrókkal hivatkozhatunk. A makrók paramétere a címke, amelyre hivatkozunk (a kulcsszó nélkül). Az összes említett hivatkozástípus, beleértve az `\url` kulcsszóval

bevezetett web-hivatkozásokat is a `hyperref`[1] csomagnak köszönhetően aktívak a legtöbb PDF-nézegetőben, rájuk kattintva a dokumentum megfelelő oldalára ugrik a PDF-néző vagy a megfelelő linket megnyitja az alapértelmezett böngészővel. A `hyperref` csomag a kimeneti PDF-dokumentumba könyvjelzőket is készít a tartalomjegyzékből. Ez egy szintén aktív tartalomjegyzék, amelynek elemeire kattintva a nézegető behozza a kiválasztott fejezetet.

9.2. Ábrák és táblázatok

Használjunk vektorgrafikus ábrákat, ha van rá módunk. PDFLaTeX használata esetén PDF formátumú ábrákat lehet beilleszteni könnyen, az EPS (PostScript) vektorgrafikus képformátum beillesztését a PDFLaTeX közvetlenül nem támogatja (de lehet konvertálni, lásd később). Ha vektorgrafikus formában nem áll rendelkezésünkre az ábra, akkor a veszteségmentes PNG, valamint a veszteséges JPEG formátumban érdemes elmenteni. Figyeljünk arra, hogy ilyenkor a képek felbontása elég nagy legyen ahhoz, hogy nyomtatásban is megfelelő minőséget nyújtson (legalább 300 dpi javasolt). A dokumentumban felhasznált képfájlokat a dokumentum forrása mellett érdemes tartani, archiválni, mivel ezek hiányában a dokumentum nem fordul újra. Ha lehet, a vektorgrafikus képeket vektorgrafikus formátumban is érdemes elmenteni az újrafelhasználhatóság (az átszerkeszthetőség) érdekében.

Kapcsolási rajzok legtöbbször kimásolhatók egy vektorgrafikus programba (pl. CorelDraw) és onnan nagyobb felbontással raszterizálva kimenthatők PNG formátumban. Ugyanakkor kiváló ábrák készíthetők Microsoft Visio vagy hasonló program használatával is: Visio-ból az ábrák közvetlenül PDF-be is menthetők.

Lehetőségeink Matlab ábrák esetén:

- Képernyőlopás (*screenshot*) is elfogadható minőségű lehet a dokumentumban, de általában jobb felbontást is el lehet érni más módszerrel.

[1] Segítségével a dokumentumban megjelenő hivatkozások nem csak dinamikussá válnak, de színezhetők is, bővebbet erről a csomag dokumentációjában találunk. Ez egyúttal egy példa lábjegyzet írására.

- A Matlab ábrát a `File/Save As` opcióval lementhetjük PNG formátumban (ugyanaz itt is érvényes, mint korábban, ezért nem javasoljuk).
- A Matlab ábrát az `Edit/Copy figure` opcióval kimásolhatjuk egy vektorgrafikus programba is és onnan nagyobb felbontással raszterizálva kimenthatjük PNG formátumban (nem javasolt).
- Javasolt megoldás: az ábrát a `File/Save As` opcióval EPS *vektorgrafikus* formátumban elmentjük, PDF-be konvertálva beillesztjük a dolgozatba.

Az EPS kép az `epstopdf` programmal[2] konvertálható PDF formátumba. Célszerű egy batch-fájlt készíteni az összes EPS ábra lefordítására az alábbi módon (ez Windows alatt működik).

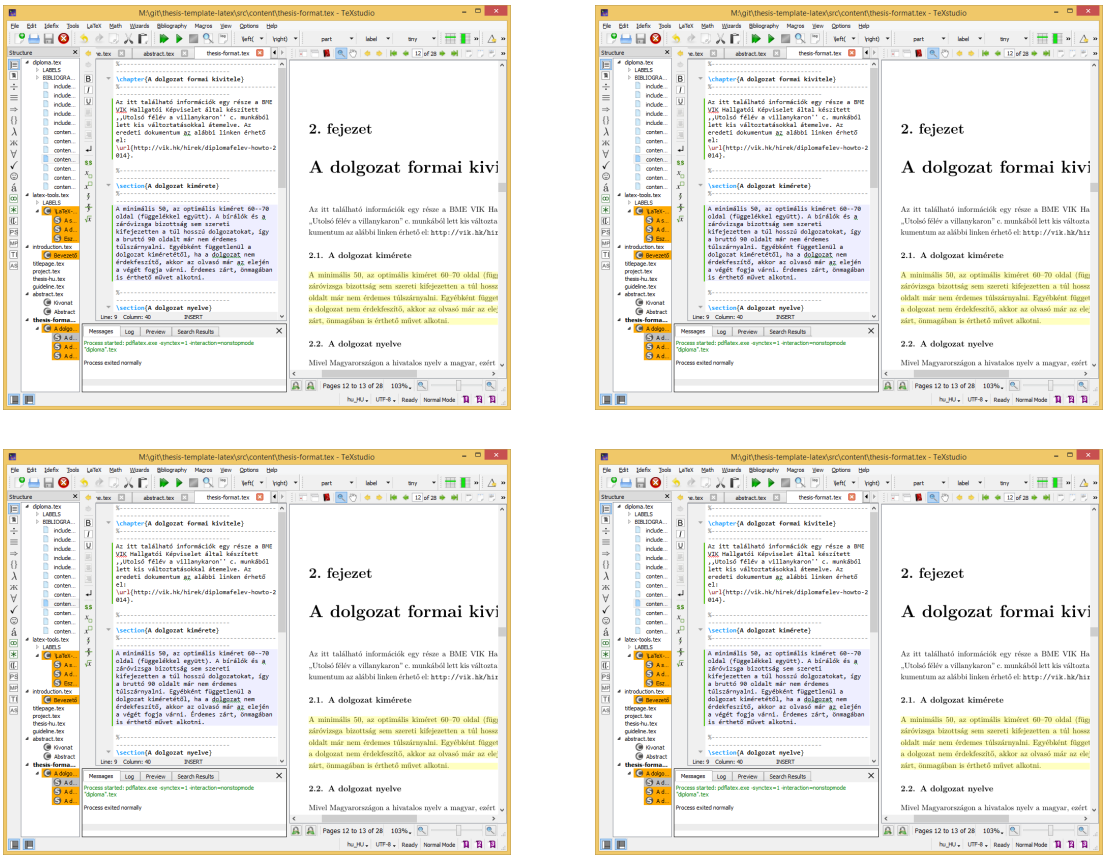
```
@echo off
for %%j in (*.eps) do (
  echo converting file "%%j"
  epstopdf "%%j"
)
echo done .
```

Egy ilyen parancsfájlt (`convert.cmd`) elhelyeztük a sablon `figures\eps` könyvtárába, így a felhasználónak csak annyi a dolga, hogy a `figures\eps` könyvtárba kimenti az EPS formátumú vektorgrafikus képet, majd lefuttatja a `convert.cmd` parancsfájlt, ami PDF-be konvertálja az EPS fájlt.

Ezek után a PDF-ábrát ugyanúgy lehet a dokumentumba beilleszteni, mint a PNG-t vagy a JPEG-et. A megoldás előnye, hogy a lefordított dokumentumban is vektorgrafikusan tárolódik az ábra, így a mérete jóval kisebb, mintha raszterizáltuk volna beillesztés előtt. Ez a módszer minden – az EPS formátumot ismerő – vektorgrafikus program (pl. CorelDraw) esetén is használható.

A képek beillesztésére a 7. fejezetben mutattunk be példát (7.1. ábra). Az előző mondatban egyúttal az automatikusan feloldódó ábrahivatkozásra is láthatunk példát. Több képfájlt is beilleszthetünk egyetlen ábrába. Az egyes képek közötti horizontális és vertikális margót metrikusan szabályozhatjuk (9.1. ábra). Az ábrák elhelyezését számtalan tipográfiai szabály egyidejű teljesítésével a fordító maga végzi, a dokumentum írója csak preferenciáit jelezheti a fordító felé (olykor ez bosszúságot is okozhat, ilyenkor pl. a kép méretével lehet játszani).

[2] a korábban említett LaTeX-disztribúciókban megtalálható



9.1. ábra. Több képfájl beillesztése esetén térközöket is érdemes használni.

A táblázatok használatára a 9.1 táblázat mutat példát. A táblázatok formázásához hasznos tanácsokat találunk a `booktabs` csomag dokumentációjában.

Órajel	Frekvencia	Cél pin
CLKA	100 MHz	FPGA CLK0
CLKB	48 MHz	FPGA CLK1
CLKC	20 MHz	Processzor
CLKD	25 MHz	Ethernet chip
CLKE	72 MHz	FPGA CLK2
XBUF	20 MHz	FPGA CLK3

9.1. táblázat. Az órajel-generátor chip órajel-kimenetei.

9.3. Felsorolások és listák

Számozatlan felsorolásra mutat példát a jelenlegi bekezdés:

- *első bajusz:* ide lehetne írni az első elem kifejését,

- *második bajusz:* ide lehetne írni a második elem kifejését,
- *ez meg egy szakáll:* ide lehetne írni a harmadik elem kifejését.

Számozott felsorolást is készíthetünk az alábbi módon:

1. *első bajusz:* ide lehetne írni az első elem kifejését, és ez a kifejtés így néz ki, ha több sorosra sikeredik,
2. *második bajusz:* ide lehetne írni a második elem kifejését,
3. *ez meg egy szakáll:* ide lehetne írni a harmadik elem kifejését.

A felsorolásokban sorok végén vessző, az utolsó sor végén pedig pont a szokásos írásjel. Ez alól kivételt képezhet, ha az egyes elemek több teljes mondatot tartalmaznak.

Listákban a dolgozat szövegétől elkülönítendő kódrészleteket, programsorokat, pszeudo-kódokat jeleníthetünk meg (9.1. kódrészlet).

```
\begin{enumerate}
  \item \emph{első bajusz:} ide lehetne írni az első elem kifejését,
  és ez a kifejtés így néz ki, ha több sorosra sikeredik,
  \item \emph{második bajusz:} ide lehetne írni a második elem kifejését,
  \item \emph{ez meg egy szakáll:} ide lehetne írni a harmadik elem kifejését.
\end{enumerate}
```

9.1. lista. A fenti számozott felsorolás LaTeX-forráskódja

A lista keretét, háttérszínét, egész stílusát megválaszthatjuk. Ráadásul különféle programnyelveket és a nyelveken belül kulcsszavakat is definiálhatunk, ha szükséges. Erről bővebbet a `listings` csomag hivatalos leírásában találhatunk.

9.4. Képletek

Ha egy formula nem túlságosan hosszú, és nem akarjuk hivatkozni a szövegből, mint például a $e^{i\pi} + 1 = 0$ képlet, *szövegközi képletként* szokás leírni. Csak, hogy másik példát is lássunk, az $U_i = -d\Phi/dt$ Faraday-törvény a $\operatorname{rot} E = -\frac{dB}{dt}$ differenciális alakban adott Maxwell-egyenlet felületre vett integráljából vezethető le. Látható, hogy a LaTeX-fordító a sorközöket betartja, így a szöveg szedése esztétikus marad szövegközi képletek használata esetén is.

Képletek esetén az általános konvenció, hogy a kisbetűk skalárt, a kis félkövér betűk ($\mathbf{v}$) oszlopvektort – és ennek megfelelően $\mathbf{v}^T$ sorvektort – a kapitális félkövér betűk ($\mathbf{V}$) mátrixot jelölnek. Ha ettől el szeretnénk térni, akkor az alkalmazni kívánt jelölésmódot célszerű külön alfejezetben definiálni. Ennek megfelelően, amennyiben $\mathbf{y}$ jelöli a mérések vektorát, ϑ a paraméterek vektorát és $\hat{\mathbf{y}} = \mathbf{X}\vartheta$ a paraméterekben lineáris modellt, akkor a *Least-Squares* értelemben optimális paraméterbecslő $\hat{\vartheta}_{LS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ lesz.

Emellett kiemelt, sorszámozott képleteket is megadhatunk, ennél az `equation` és a `eqnarray` környezetek helyett a korszerűbb `align` környezet alkalmazását javasoljuk (több okból, különféle problémák elkerülése végett, amelyekre most nem térünk ki). Tehát

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \tag{9.1}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x}, \tag{9.2}$$

ahol $\mathbf{x}$ az állapotvektor, $\mathbf{y}$ a mérések vektora és $\mathbf{A}$, $\mathbf{B}$ és $\mathbf{C}$ a rendszert leíró paramétermátrixok. Figyeljük meg, hogy a két egyenletben az egyenlőségjelek egymáshoz igazítva jelennek meg, mivel a mindkettőt az & karakter előzi meg a kódban. Lehetőség van számozatlan kiemelt képlet használatára is, például

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u},$$

$$\mathbf{y} = \mathbf{C}\mathbf{x}.$$

Mátrixok felírására az $\mathbf{A}\mathbf{x} = \mathbf{b}$ inhomogén lineáris egyenlet részletes kifejtésével mutatunk példát:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}. \tag{9.3}$$

A `\frac` utasítás hatékonyságát egy általános másodfokú tag átviteli függvényén keresztül mutatjuk be, azaz

$$W(s) = \frac{A}{1 + 2T\xi s + s^2T^2}. \tag{9.4}$$

A matematikai mód minden szimbólumának és képességének a bemutatására természetesen itt nincs lehetőség, de gyors referenciaként hatékonyan használhatók a következő linkek:

`http://www.artofproblemsolving.com/LaTeX/AoPS_L_GuideSym.php`,

`http://www.ctan.org/tex-archive/info/symbols/comprehensive/symbols-a4.pdf`,

`ftp://ftp.ams.org/pub/tex/doc/amsmath/short-math-guide.pdf`.

Ez pedig itt egy magyarázat, hogy miért érdemes `align` környezetet használni:

`http://texblog.net/latex-archive/maths/eqnarray-align-environment/`.

9.5. Irodalmi hivatkozások

Egy LaTeX dokumentumban az irodalmi hivatkozások definíciójának két módja van. Az egyik a `\thebibliograhy` környezet használata a dokumentum végén, az `\end{document}` lezárás előtt.

```
\begin{thebibliography}{9}

\bibitem{Lamport94} Leslie Lamport, \emph{\LaTeX: A Document Preparation System}.
Addison Wesley, Massachusetts, 2nd Edition, 1994.

\end{thebibliography}
```

Ezek után a dokumentumban a `\cite{Lamport94}` utasítással hivatkozhatunk a forrásra. A fenti megadás viszonylag kötetlen, a szerző maga formázza az irodalomjegyzéket (ami gyakran inkonzisztens eredményhez vezet).

Egy sokkal professzionálisabb módszer a BiBTeX használata, ezért ez a sablon is ezt támogatja. Ebben az esetben egy külön szöveges adatbázisban definiáljuk a forrásmunkákat, és egy külön stílusfájl határozza meg az irodalomjegyzék kinézetét. Ez, összhangban azzal, hogy külön formátumkonvenció határozza meg a folyóirat-, a könyv-, a konferenciacikk- stb. hivatkozások kinézetét az

irodalomjegyzékben (a sablon használata esetén ezzel nem is kell foglalkoznia a hallgatónak, de az eredményt célszerű ellenőrizni). felhasznált hivatkozások adatbázisa egy `.bib` kiterjesztésű szöveges fájl, amelynek szerkezetét a A 9.2 kódrészlet demonstrálja. A forrásmunkák bevitelekor a sor végi vesszők külön figyelmet igényelnek, mert hiányuk a BiBTEX-fordító hibaüzenetét eredményezi. A forrásmunkákat típus szerinti kulcsszó vezeti be (`@book` könyv, `@inproceedings` konferenciakiadványban megjelent cikk, `@article` folyóiratban megjelent cikk, `@techreport` valamelyik egyetem gondozásában megjelent műszaki tanulmány, `@manual` műszaki dokumentáció esetén stb.). Nemcsak a megjelenés stílusa, de a kötelezően megadandó mezők is típusról-típusra változnak. Egy jól használható referencia a `http://en.wikipedia.org/wiki/BibTeX` oldalon található.

```
@book{Wettl04,
  author    = {Ferenc Wettl and Gyula Mayer and Péter Szabó},
  publisher = {Panem Könyvkiadó},
  title     = {\LaTeX~kézikönyv},
  year      = {2004},
}

@article{Candy86,
  author       = {James C. Candy},
  journaltitle = {{IEEE} Trans.\ on Communications},
  month        = {01},
  note         = {\doi{10.1109/TCOM.1986.1096432}},
  number       = {1},
  pages        = {72--76},
  title        = {Decimation for Sigma Delta Modulation},
  volume       = {34},
  year         = {1986},
}

@inproceedings{Lee87,
  author    = {Wai L. Lee and Charles G. Sodini},
  booktitle = {Proc.\ of the IEEE International Symposium on Circuits and Systems},
  location  = {Philadelphia, PA, USA},
  month     = {05~4--7},
  pages     = {459--462},
  title     = {A Topology for Higher Order Interpolative Coders},
  vol       = {2},
  year      = {1987},
}

@thesis{KissPhD,
  author      = {Peter Kiss},
```

```
  institution = {Technical University of Timi\c{s}oara, Romania},
  month       = {04},
  title       = {Adaptive Digital Compensation of Analog Circuit Imperfections for Cascaded Delta-
    Sigma Analog-to-Digital Converters},
  type        = {phdthesis},
  year        = {2000},
}

@manual{Schreier00,
  author       = {Richard Schreier},
  month        = {01},
  note         = {\url{http://www.mathworks.com/matlabcentral/fileexchange/}},
  organization = {Oregon State University},
  title        = {The Delta-Sigma Toolbox v5.2},
  year         = {2000},
}

@misc{DipPortal,
  author       = {{Budapesti űMszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai
    Kar}},
  howpublished = {\url{http://diplomaterv.vik.bme.hu/}},
  title        = {Diplomaterv portál (2011. február 26.)},
}

@incollection{Mkrtychev:1997,
  author    = {Mkrtychev, Alexey},
  booktitle = {Logical Foundations of Computer Science},
  doi       = {10.1007/3-540-63045-7_27},
  editor    = {Adian, Sergei and Nerode, Anil},
  isbn      = {978-3-540-63045-6},
  pages     = {266-275},
  publisher = {Springer Berlin Heidelberg},
  series    = {Lecture Notes in Computer Science},
  title     = {Models for the logic of proofs},
  url       = {http://dx.doi.org/10.1007/3-540-63045-7_27},
  volume    = {1234},
  year      = {1997},
}
```

9.2. lista. Példa szöveges irodalomjegyzék-adatbázisra BibTEX használata esetén.

A stílusfájl egy `.sty` kiterjesztésű fájl, de ezzel lényegében nem kell foglalkozni, mert vannak beépített stílusok, amelyek jól használhatók. Ez a sablon a BiBTEX-et használja, a hozzá tartozó adatbázisfájl a `mybib.bib` fájl. Megfigyelhető, hogy az irodalomjegyzéket a dokumentum végére (a `\end{document}` utasítás elé) beillesztett `\bibliography{mybib}` utasítással hozhatjuk létre, a stílusát pedig

ugyanitt a `\bibliographystyle{plain}` utasítással adhatjuk meg. Ebben az esetben a `plain` előre definiált stílust használjuk (a sablonban is ezt állítottuk be). A `plain` stíluson kívül természetesen számtalan más előre definiált stílus is létezik. Mivel a `.bib` adatbázisban ezeket megadtuk, a BiBTEX-fordító is meg tudja különböztetni a szerzőt a címtől és a kiadótól, és ez alapján automatikusan generálódik az irodalomjegyzék a stílusfájl által meghatározott stílusban.

Az egyes forrásmunkákra a szövegből továbbra is a `\cite` paranccsal tudunk hivatkozni, így a 9.2. kódrészlet esetén a hivatkozások rendre `\cite{Wettl04}`, `\cite{Candy86}`, `\cite{Lee87}`, `\cite{KissPhD}`, `\cite{Schreirer00}`, `\cite{Mkrtychev:1997}` és `\cite{DipPortal}`. Az egyes forrásmunkák sorszáma az irodalomjegyzék bővítésekor változhat. Amennyiben az aktuális számhoz illeszkedő névelőt szeretnénk használni, használjuk az `\acite{}` parancsot.

Az irodalomjegyzékben alapértelmezésben csak azok a forrásmunkák jelennek meg, amelyekre található hivatkozás a szövegben, és ez így alapvetően helyes is, hiszen olyan forrásmunkákat nem illik az irodalomjegyzékbe írni, amelyekre nincs hivatkozás.

Mivel a fordítási folyamat során több lépésben oldódnak fel a szimbólumok, ezért gyakran többször is le kell fordítani a dokumentumot. Ilyenkor ez első 1-2 fordítás esetleg szimbólum-feloldásra vonatkozó figyelmeztető üzenettel zárul. Ha hibaüzenettel zárul bármelyik fordítás, akkor nincs értelme megismételni, hanem a hibát kell megkeresni. A `.bib` fájl megváltoztatáskor sokszor nincs hatása a változtatásnak azonnal, mivel nem mindig fut újra a BibTeX fordító. Ezért célszerű a változtatás után azt manuálisan is lefuttatni (TeXstudio esetén `Tools/Bibliography`).

Hogy a szövegbe ágyazott hivatkozások kinézetét demonstráljuk, itt most sorban meghivatkozzuk a [**?**], [**?**], [**?**], [**?**], [**?**] és a [**?**][3] forrásmunkát, valamint a [**?**] weboldalt.

Megjegyzendő, hogy az ékezetes magyar betűket is tartalmazó `.bib` fájl az `inputenc` csomaggal betöltött `latin2` betűkészlet miatt fordítható. Ugyanez a `.bib` fájl hibaüzenettel fordul egy olyan dokumentumban, ami nem tartalmazza a

[3]Informatikai témában gyakran hivatkozunk cikkeket a Springer LNCS valamely kötetéből, ez a hivatkozás erre mutat egy helyes példát.

`\usepackage[latin2]{inputenc}` sort. Speciális igény esetén az irodalmi adatbázis általánosabb érvényűvé tehető, ha az ékezetes betűket speciális latex karakterekkel helyettesítjük a `.bib` fájlban, pl. á helyett `\'{a}`-t vagy ő helyett `\H{o}`-t írunk.

Irodalomhivatkozásokat célszerű először olyan szolgáltatásokban keresni, ahol jó minőségű bejegyzések találhatók (pl. ACM Digital Library,[4] DBLP,[5] IEEE Xplore,[6] SpringerLink[7]) és csak ezek után használni kevésbé válogatott forrásokat (pl. Google Scholar[8]). A jó minőségű bejegyzéseket is érdemes megfelelően tisztítani.[9] A sablon angol nyelvű változatában használt `plainnat` beállítás egyik sajátossága, hogy a cikkhez generált hivatkozás a cikk DOI-ját és URL-jét is tartalmazza, ami gyakran duplikátumhoz vezet – érdemes tehát a DOI-kat tartalmazó URL mezőket törölni.

9.6. A dolgozat szerkezete és a forrásfájlok

A diplomatervsablonban a TeX fájlok két alkönyvtárban helyezkednek el. Az `include` könyvtárban azok szerepelnek, amiket tipikusan nem kell szerkesztenünk, ezek a sablon részei (pl. címoldal). A `content` alkönyvtárban pedig a saját munkánkat helyezhetjük el. Itt érdemes az egyes fejezeteket külön TEX állományokba rakni.

A diplomatervsablon (a kari irányelvek szerint) az alábbi fő fejezetekből áll:

1. 1 oldalas *tájékoztató* a szakdolgozat/diplomaterv szerkezetéről (`include/guideline.tex`), ami a végső dolgozatból törlendő,

2. *feladatkiírás* (`include/project.tex`), a dolgozat nyomtatott verzójában ennek a helyére kerül a tanszék által kiadott, a tanszékvezető által aláírt feladatkiírás, a dolgozat elektronikus verziójába pedig a feladatkiírás egyáltalán ne kerüljön bele, azt külön tölti fel a tanszék a diplomaterv-honlapra,

3. *címoldal* (`include/titlepage.tex`),

[4] `https://dl.acm.org/`
[5] `http://dblp.uni-trier.de/`
[6] `http://ieeexplore.ieee.org/`
[7] `https://link.springer.com/`
[8] `http://scholar.google.com/`
[9] `https://github.com/FTSRG/cheat-sheets/wiki/BibTeX-Fixing-entries-from-common-sources`

4. *tartalomjegyzék* (`thesis.tex`),

5. a diplomatervező *nyilatkozat*a az önálló munkáról (`include/declaration.tex`),

6. 1-2 oldalas tartalmi *összefoglaló* magyarul és angolul, illetve elkészíthető még további nyelveken is (`content/abstract.tex`),

7. *bevezetés*: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása (`content/introduction.tex`),

8. sorszámmal ellátott *fejezetek*: a feladatkiírás pontosítása és részletes elemzése, előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések, a tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása, a megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek,

9. esetleges *köszönetnyilvánítás*ok (`content/acknowledgement.tex`),

10. részletes és pontos *irodalomjegyzék* (ez a sablon esetében automatikusan generálódik a `thesis.tex` fájlban elhelyezett `\bibliography` utasítás hatására, a 9.5. szakaszban leírtak szerint),

11. *függelékek* (`content/appendices.tex`).

A sablonban a fejezetek a `thesis.tex` fájlba vannak beillesztve `\include` utasítások segítségével. Lehetőség van arra, hogy csak az éppen szerkesztés alatt álló `.tex` fájlt fordítsuk le, ezzel lerövidítve a fordítási folyamatot. Ezt a lehetőséget az alábbi kódrészlet biztosítja a `thesis.tex` fájlban.

```
\includeonly{
  guideline,%
  project,%
  titlepage,%
  declaration,%
  abstract,%
  introduction,%
  chapter1,%
  chapter2,%
  chapter3,%
  acknowledgement,%
```

```
  appendices,%
}
```

Ha az alábbi kódrészletben az egyes sorokat a `%` szimbólummal kikommentezzük, akkor a megfelelő `.tex` fájl nem fordul le. Az oldalszámok és a tartalomjegyék természetesen csak akkor billennek helyre, ha a teljes dokumentumot lefordítjuk.

9.7. Alapadatok megadása

A diplomaterv alapadatait (cím, szerző, konzulens, konzulens titulusa) a `thesis.tex` fájlban lehet megadni.

9.8. Új fejezet írása

A főfejezetek külön `content` könyvtárban foglalnak helyet. A sablonhoz 3 fejezet készült. További főfejezeteket úgy hozhatunk létre, ha új TEX fájlt készítünk a fejezet számára, és a `thesis.tex` fájlban, a `\include` és `\includeonly` utasítások argumentumába felvesszük az új `.tex` fájl nevét.

9.9. Definíciók, tételek, példák

Definíció 1 (Fluxuskondenzátor térerőssége). Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. ▪

Példa 1. *Példa egy példára. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.*

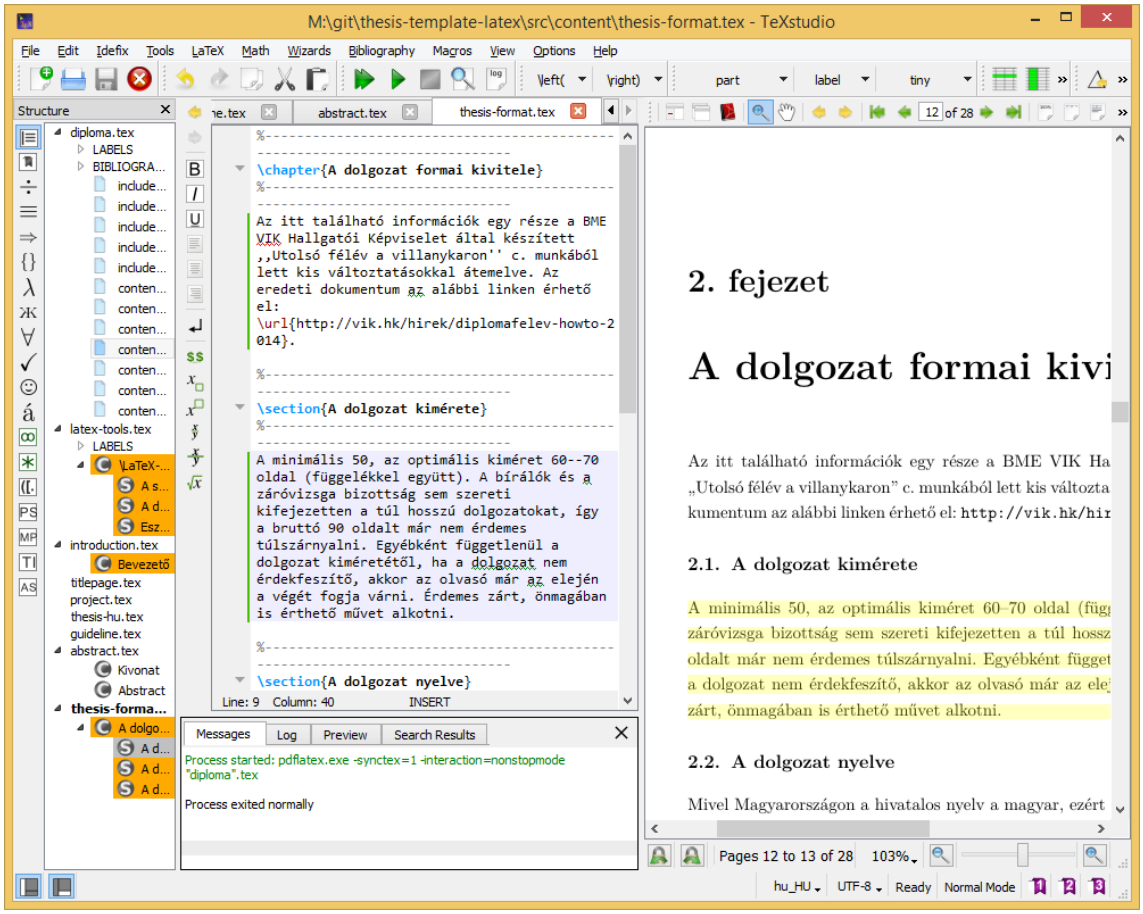
Tétel 1 (Kovács tétele). Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. ▪

Köszönetnyilvánítás

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Függelék

F.1. A TeXstudio felülete



F.1.1. ábra. A TeXstudio LaTeX-szerkesztő.

F.2. Válasz az „Élet, a világmindenség, meg minden” kérdésére

A Pitagorasz-tételből levezetve

$$c^2 = a^2 + b^2 = 42. \tag{F.2.1}$$

A Faraday-indukciós törvényből levezetve

$$\operatorname{rot} E = -\frac{dB}{dt} \qquad \longrightarrow \qquad U_i = \oint_{\mathbf{L}} \mathbf{Edl} = -\frac{d}{dt} \int_{A} \mathbf{Bda} = 42. \tag{F.2.2}$$