STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Aplikace 3N

Jan Klivan
Jihočeský kraj

Dačice 2021

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Aplikace 3N
Application 3N

Autoři: Jan Klivan
Škola: Gymnázium Dačice, Boženy Němcové 213, 380 01 Dačice
Kraj: Jihočeský kraj

Dačice 2021

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Dačicích dne 2. 4. 2021 ……

Jan Klivan

Poděkování

Chtěl bych poděkovat Mgr. Marku Krajíčkovi za pomoc při korekturách výsledné práce.

Anotace

Práce se zabývá naprogramováním aplikace pro učení se cizích slov. Aplikace je rozdělena na studentskou část, která umožňuje studentovi se učit slova prostřednictvím sady testů a přidávat si svá slova z webových stránek nebo skenováním textu. Učitelská část umožňuje učitelům zadávat slova k naučení skrze domácí úkoly. Práce se také zabývá vytvořením neuronové sítě pro rozpoznání řeči na mobilních zařízeních.

Klíčová slova

Mobilní aplikace, databáze, neuronová síť, Tensorflow, PyTorch

Annotation

This project deals with making of an application for learning foreign words. The application is divided to a student part, where students can learn new words through set of tests. They can also add new words from web pages or by scanning of text. In the teacher's part of the application, teachers can give their students homework to learn new words. This project also deals with making of neural network for speech recognition, which can be run on mobile devices.

Keywords

Mobile application, database, neural network, Tensorflow, PyTorch

1 Úvod

Tato práce se zabývá naprogramováním aplikace 3N(najdi, napiš, naskenuj), jejímž účelem je pomoci studentům s učením cizích slov. Cílem je udělat jednoduchou, intuitivní aplikaci, která umožní studentovi si přidávat slova například prohlížením stránek nebo skenováním z učebnice. Následné otestování znalosti slov je provedeno prostřednictvím testů. Celá aplikace je vázána na hlavní server, na který si ukládá data, zároveň však funguje i offline a případné změny synchronizuje s databázovým serverem později. Druhá část aplikace je určena pro učitele, kteří můžou studentům zadat slova k naučení za domácí úkol.

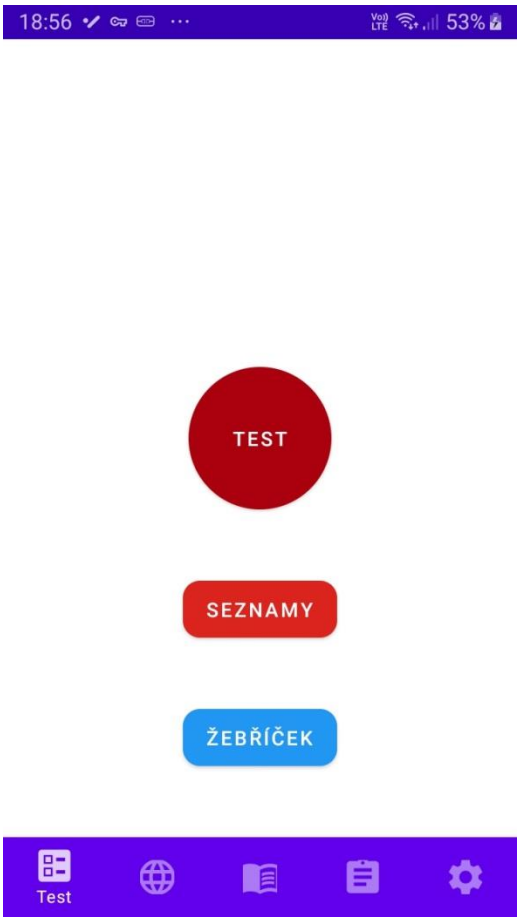
Druhým cílem práce je vytvoření neuronové sítě pro kontrolu výslovnosti v rámci poslechového testu, která bude dostatečně malá, aby byla schopná běžet na omezeném výpočetním výkonu. V rámci toho jsou porovnány dva nejpopulárnější frameworky pro učení neuronový sítí Tensorflow a PyTorch.

2 POPIS ŘEŠENÍ

2.1 Studentská část aplikace

2.1.1 Úvodní obrazovka

První z pěti sekcí je úvodní obrazovka, ze které je možné spustit test nebo vybrat seznam slov, která budou do testu zařazována. Poslední položkou je stránka s žebříčkem úspěšností ostatních studentů.

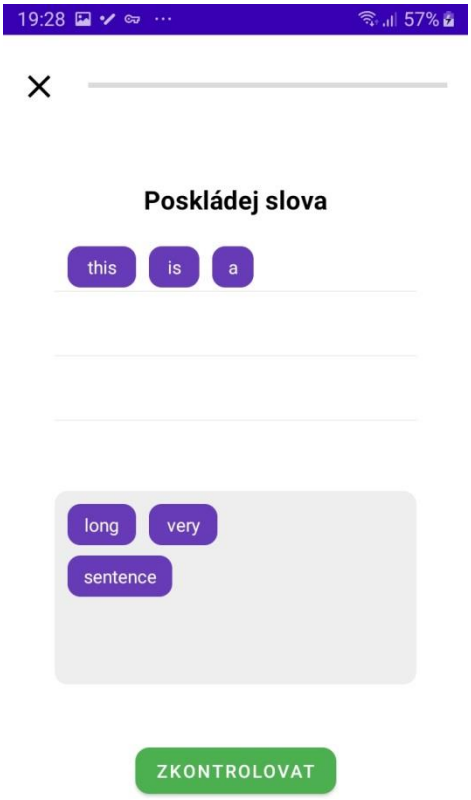


Obrázek 1: Úvodní obrazovka studentské části

Aplikace obsahuje pět druhů testů, které jsou náhodně vybrány při spuštění testu. Konkrétně to jsou: výběr ze čtyř možností, poskládání věty, rozhodnutí zdali je překlad správně, přímý překlad a poslechový test.

Obrázek 2: Výběr ze čtyř možností

Obrázek 3: Rozhodnutí o správnosti překladu

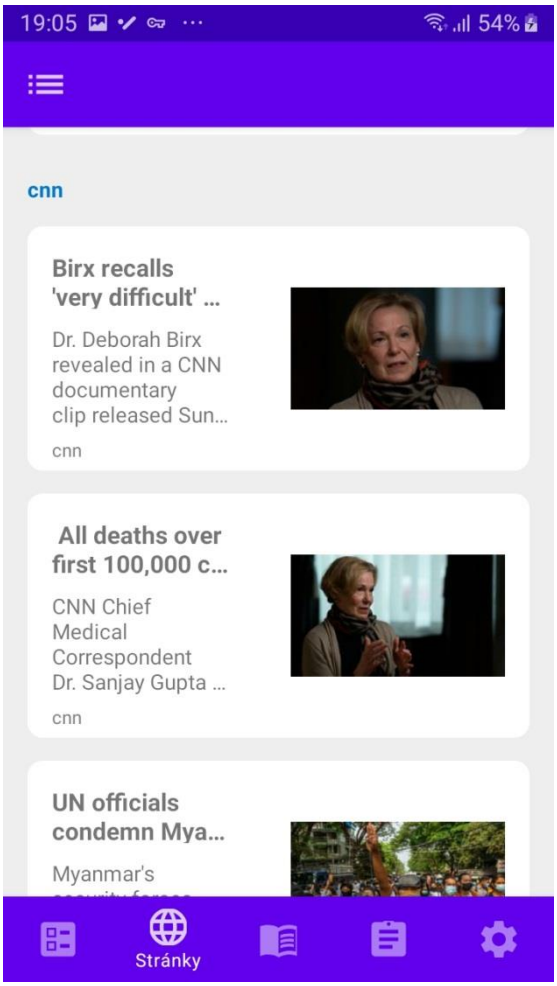


Obrázek 4: Poskládání věty

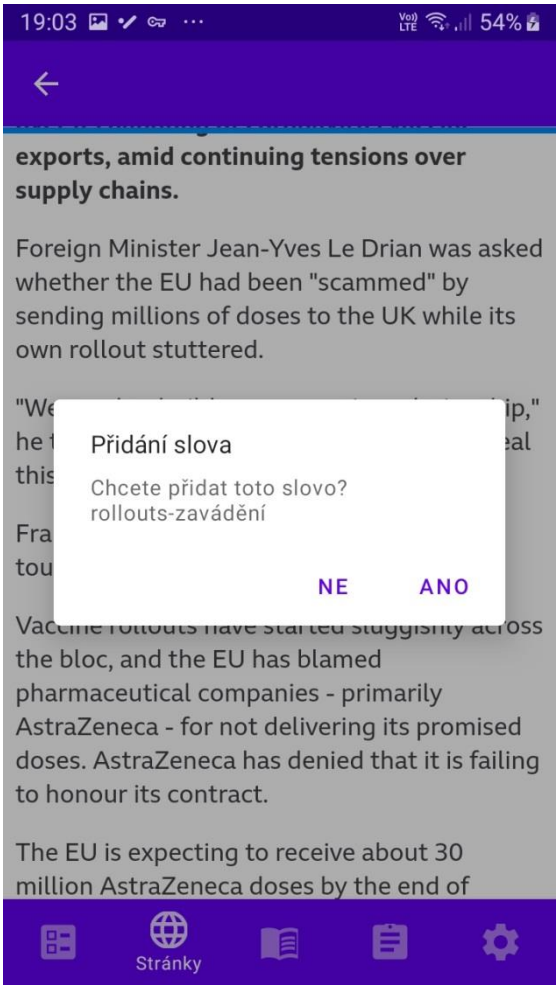
Obrázek 5: Přímý překlad

2.1.2 Webové stránky

V této sekci je zobrazen přehled článků z webových stránek, které si student přidal. Při prohlížení stránky může přidat slovo podržením prstu na něm spolu s automatickým překladem.



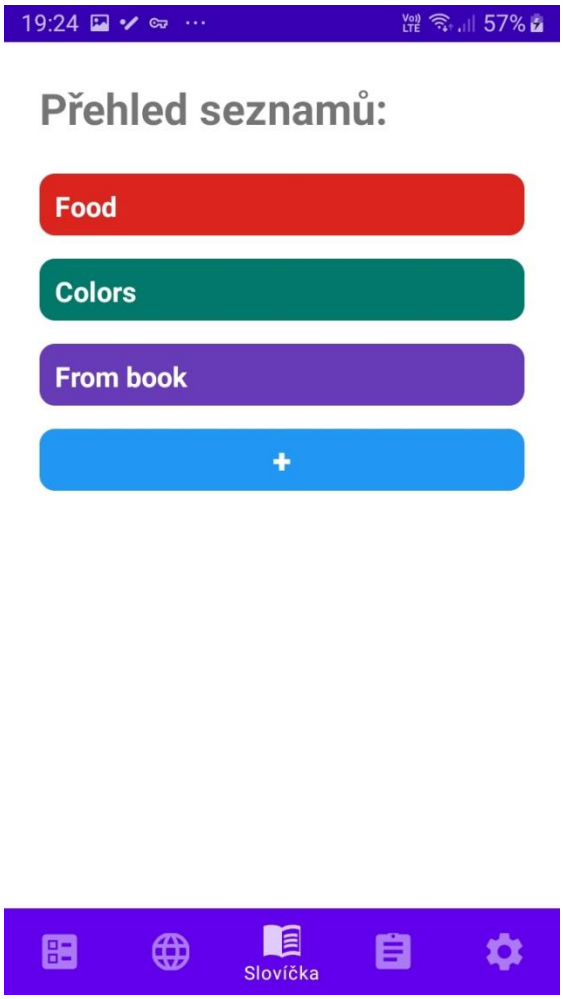
Obrázek 6: Přehled článků



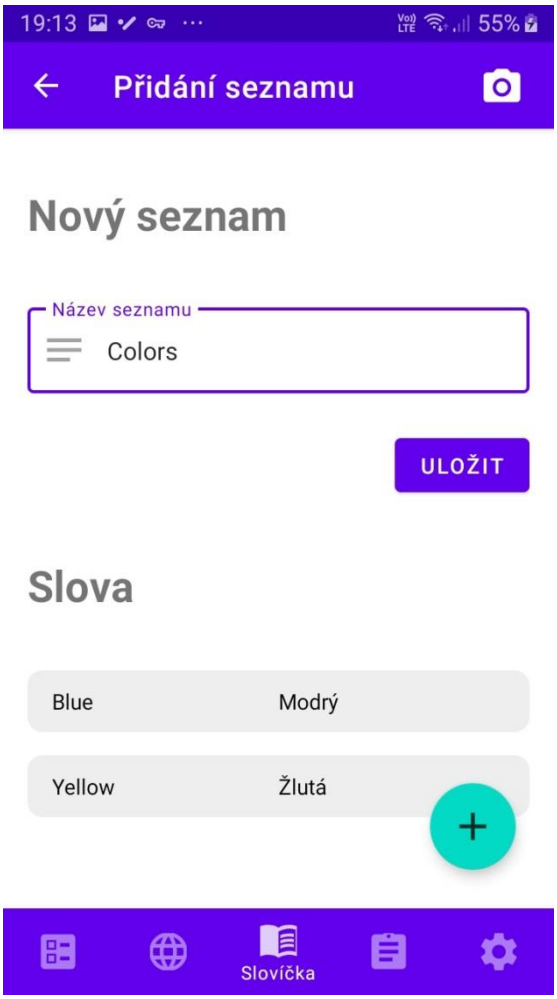
Obrázek 7: Přidání slova

2.1.3 Slovník

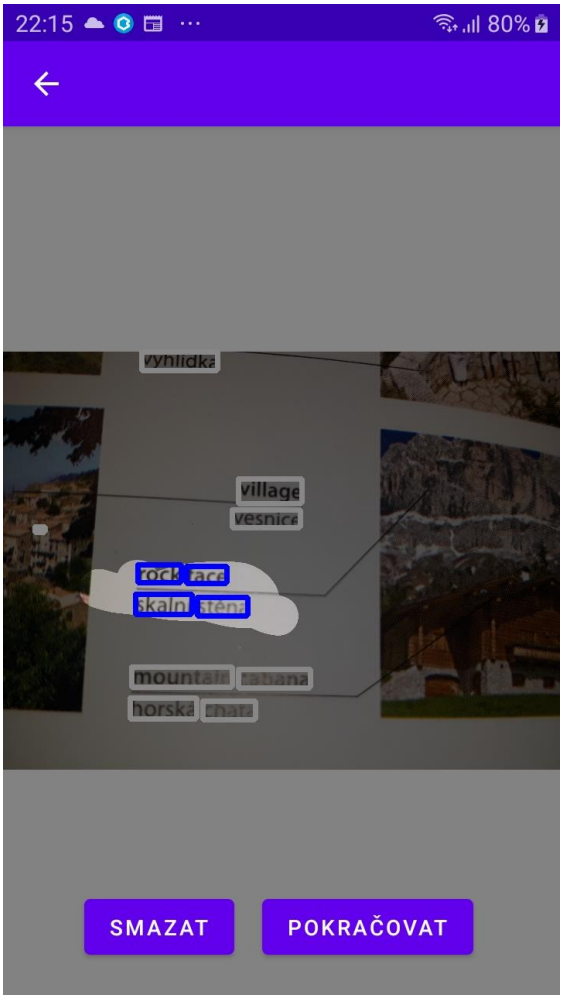
Slova jsou zde rozdělena do seznamů slov, který si student sám vytvoří, případně mu jsou zadány učitelem. Slova může přidávat klasickou formou s nabídkou překladu nebo skenováním. Skenování má dvě možné podoby: buďto skenování textu bez překladu, nebo skenování textu spolu s překladem například z učebnice.



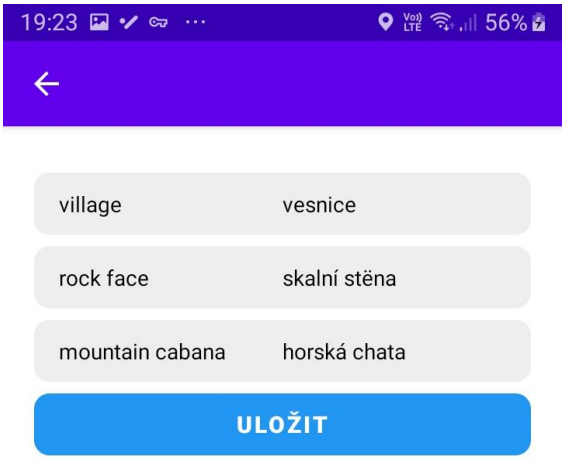
Obrázek 8: Seznamy slov



Obrázek 9: Přidání nového seznamu slov



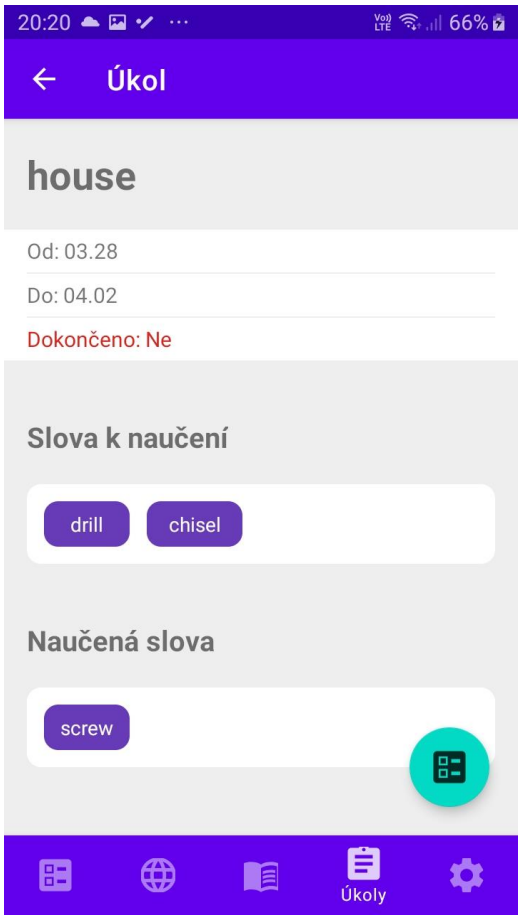
Obrázek 10: Výběr naskenovaných slov



Obrázek 11: Potvrzení naskenovaných výsledků

2.1.4 Úkoly

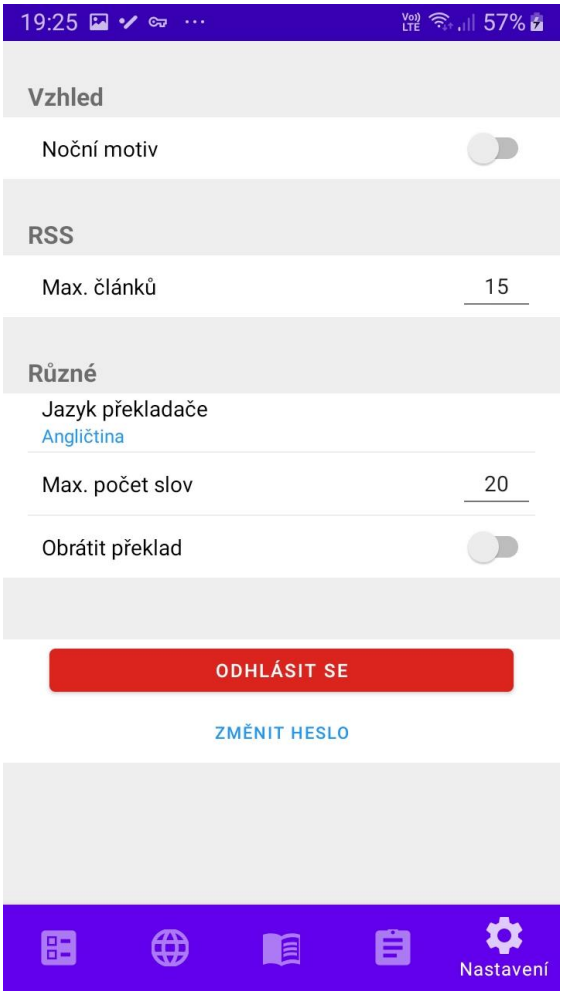
Předposlední sekce zobrazuje seznam úkolu, které jsou studentovi zadány. V detailu každého úkolu jsou zobrazeny slova k naučení a slova, které ještě zbývá se naučit, spolu s tlačítkem ke spuštění testu, jenž bude obsahovat právě slova z daného úkolu.



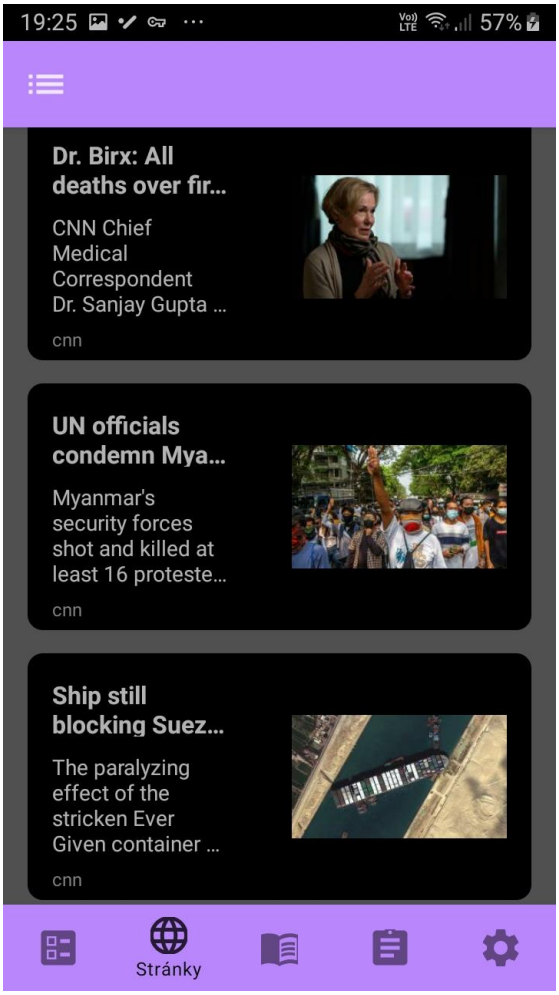
Obrázek 12: Detail úkolu

2.1.5 Nastavení

Sekce s nastavením zahrnuje obecné možnosti přizpůsobení aplikace, například noční režim nebo jazyk překladače, popřípadě možnost změnit, zdali bude testován překlad z cizího jazyka nebo do něho (možnost „Obrátit překlad“).



Obrázek 13: Nastavení aplikace



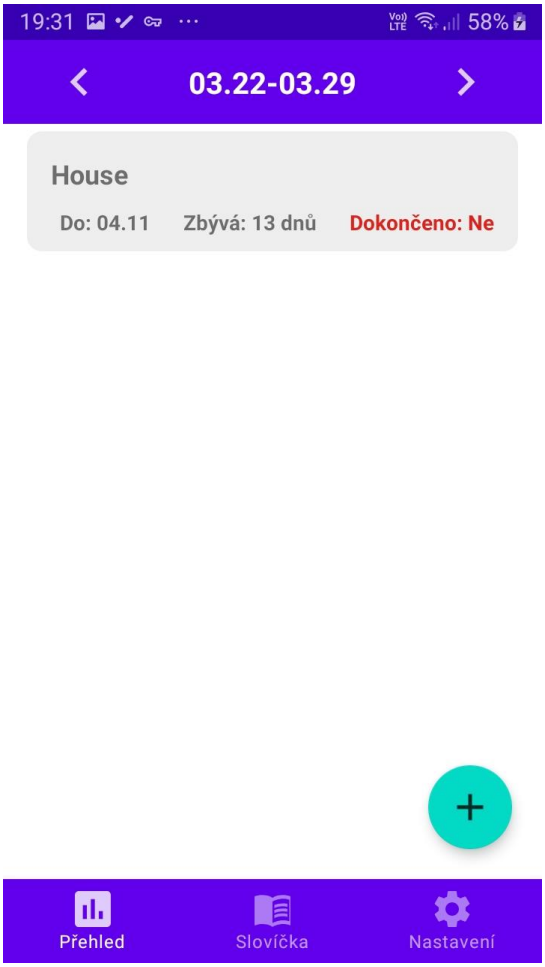
Obrázek 14: Noční motiv

2.2 Učitelská část aplikace

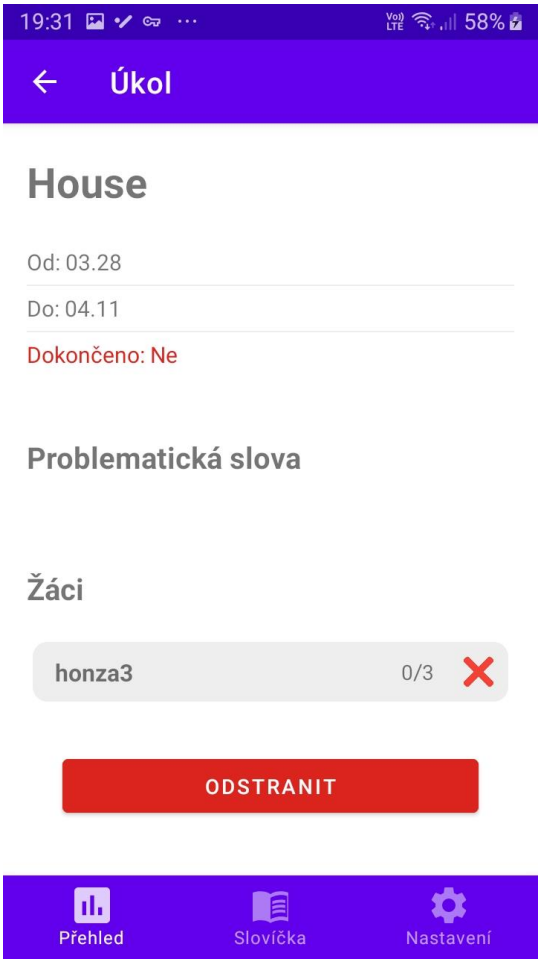
Před samotným vstupem do aplikace si učitel vybere při zapnutí aplikace třídu.

2.2.1 Přehled

Zobrazuje přehled úkolu, které učitel zadal. V detailu úkolu následně vidí, kteří studenti úkol nesplnili a kolik slov jim zbývá. V horní části se zobrazují slova, která dělala studentům největší problém se naučit, aplikace toto zjišťuje na základně toho, kolikrát v testu s daným slovem udělal student chybu.



Obrázek 15: Přehled



Obrázek 16: Detail úkolu

2.2.2 Slovník

Tato část je společná se studenty s tím rozdílem, že učitel slova zadává celé třídě. Slova přidaná v této části jsou studenty procvičována dobrovolně. Ostatní možnosti jsou stejné, například skenování.

2.3 Webové rozhraní

Webové rozhraní je určené pro administrátory k přidávání učitelů, studentů a tříd. Případně k jejich úpravě.

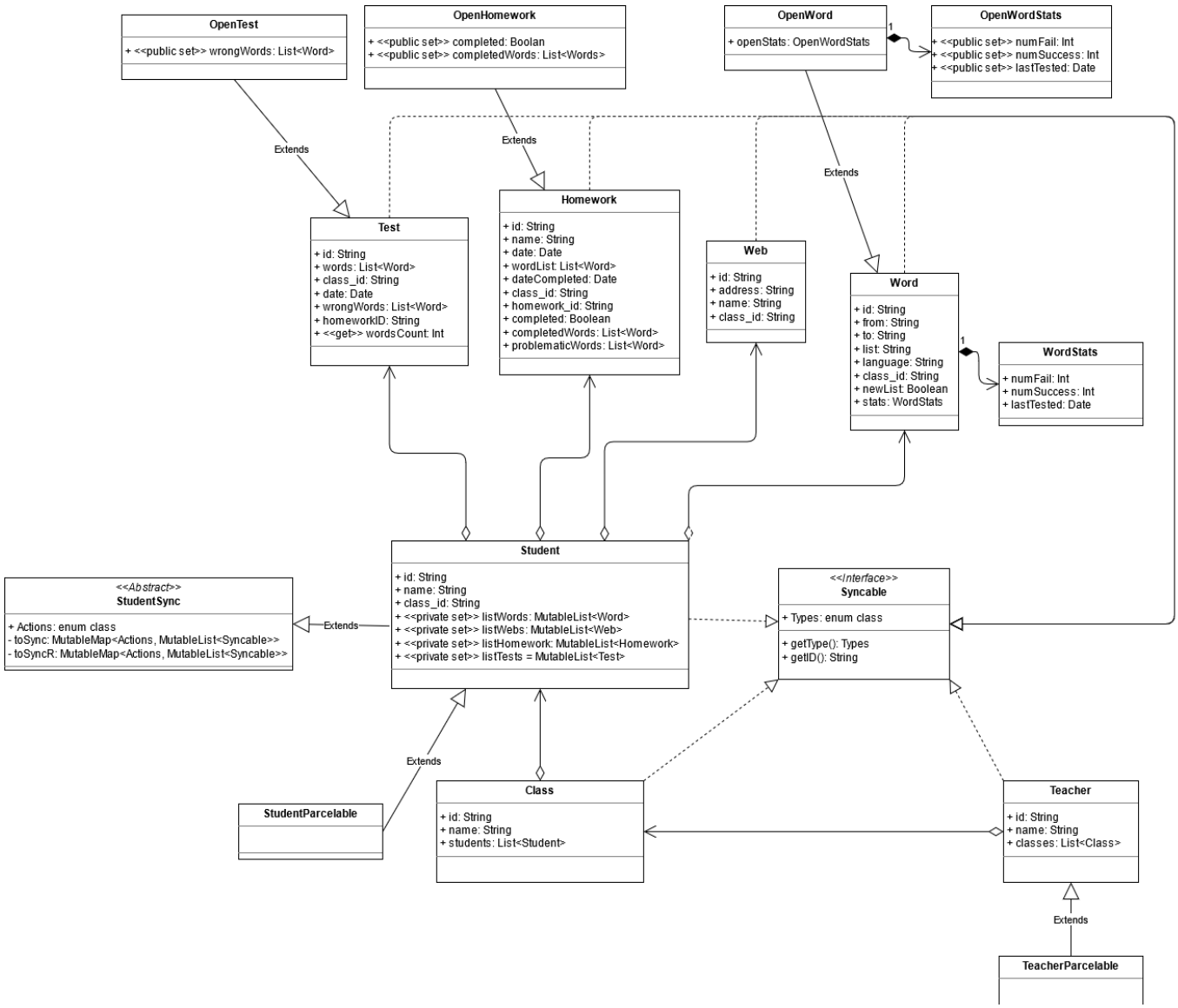
3 DATOVÉ TŘÍDY

Pro to, aby aplikace mohla fungovat v offline režimu si zaznamenává, které objekty byly přidány nebo odebrány. K tomuto účelu slouží třída syncStudent, ze které dědí třída Student, ta implementuje metody pro přidávání úkolů, testů, webových stránek a slov.

Všechny třídy implementují rozhraní Syncable, které zjednodušuje práci s těmito třídami při synchronizaci dat, aby nedocházelo k úpravám objektů těchto tříd mimo třídu Student, jsou všechny tyto třídy konstantní a k jejich úpravě je třeba je převést na otevřené verze. Tento převod se provádí jednoduše pomocí konstruktoru.

Třídy StudentParcelable a TeacherParcelable slouží k přenosu tříd Student a Teacher mezi aktivitami aplikace.

Na straně serveru jsou ve webovém rozhraní implementovány pouze zjednodušeně třídy Student, Teacher a Class.



Obrázek 17: Zjednodušené schéma tříd, funkce jsou zapsány pouze u rozhraní Syncable

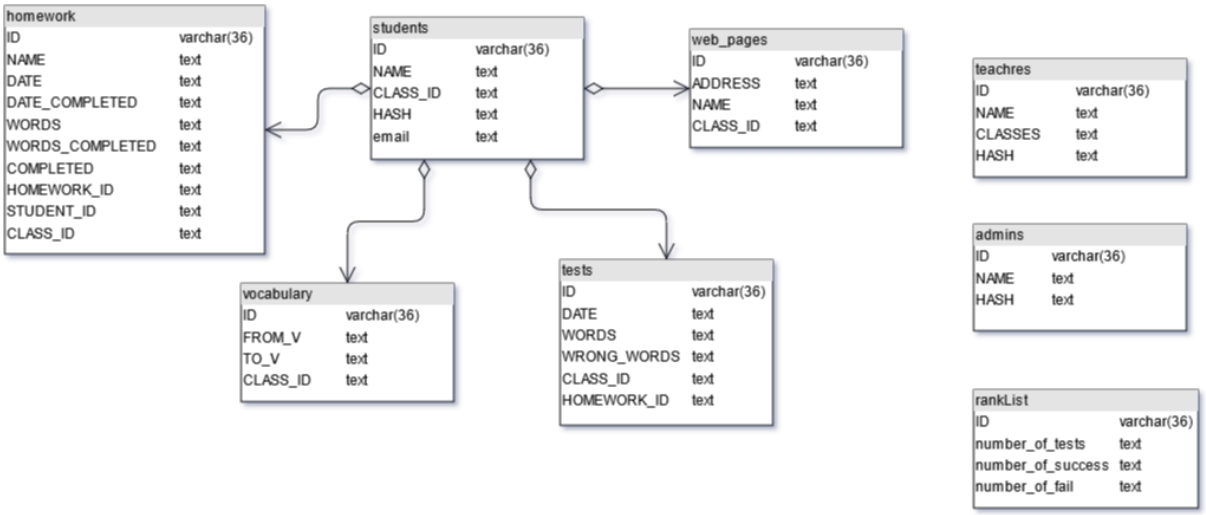
4 BACKEND SERVERU

Backend serveru tvoří dvě paralelně běžící služby, jedna pro obsluhu databáze pomocí RESTful API a druhá pro obsluhu administrátorského rozhraní.

4.1 Databáze

4.1.1 Návrh

Návrh databáze reflektuje návrh datových tříd, výjimku tvoří tabulka rankList, která uchovává údaje pro tvorbu žebříčku studentů. Není ani přítomna třída class, ta je totiž tvořena pouze jejím ID a jménem, které je odvozeno z identifikačního čísla, proto pro ni není třeba speciální tabulka. U tabulky slov chybí položka pro jméno seznamu, ten je odvozen z názvu slova, kde slovo je uloženo ve formátu „slovo#název seznamu“.



Obrázek 18: Tabulky serverové databáze

4.1.2 Komunikace

S databází probíhá komunikace prostřednictvím RESTful API po HTTPS spojení. Požadavky jsou rozděleny do dvou základních kategorií posílání dat a posílání zpráv. Zprávy pak slouží k získání dat nebo jejich smazání, případně jako informace, zdali proběhla operace úspěšně.

4.1.3 Implementace

Databáze je implementována pomocí tří základních tříd. První databaseJson převádí JSON požadavky na volání funkcí třídy databaseObj. Třída databaseObj má na starosti převod slovníků získaných z JSON požadavků na SQL příkazy, které sestavuje pomocí třídy database, zároveň kontroluje, zdali má uživatel oprávnění ke čtení nebo zápisu pro daný typ dat.

```
{
    'id': '262e73b46a848b1f',
    'type': 'SM-J530F',
    'content': {
        'id': 'student3',
        'type': 'data',
        'homework':{
            '1f5769e4-18d0-47eb-a888-1444bf2c55f7': {
                'completed': 'true',
                'words_completed': 'screw#House;chisel#House;drill#House'
            }
        },
        'web_pages': {},
        'class_id': 'septima_1',
        'name': 'honza3',
        'words': {},
        'tests': {}
    },
    'time': '28.03.2021 20:15:25',
    'version': '0.1a',
    'os_v': '28'
    'hash': 'ee26b0dd4af7e749aa1a8ee3c10ae9923f618980772e473f8819a5d4940e0db27a
    'manu': 'samsung',
    'id_user': 'student3'
}
```


Obrázek 19: Uložení dat studenta pomocí JSON požadavku

```
def addHomework(self, data):
    result = True

    if data and self.__isSubSet(data.getHeaders(), [self.DBTeacherNamespace.homework.lower()]):
        if self.__ring > self.Ring.Student:
            for id in data.getData()[self.DBTeacherNamespace.homework.lower()].keys():
                homework = data.getData()[self.DBTeacherNamespace.homework.lower()][id]

                if self.__isSubSet(homework.keys(), self.DBHomeworkNamespace.headersTeacher):
                    if not self.__Database.insert(self.DBType.homework.value,
                                                  [id,
                                                   homework[self.DBHomeworkNamespace.name.lower()],
                                                   homework[self.DBHomeworkNamespace.date.lower()],
                                                   homework[self.DBHomeworkNamespace.dateCompleted.lower()],
                                                   json.dumps(homework[self.DBHomeworkNamespace.words.lower()]),
                                                   '',
                                                   'false',
                                                   homework[self.DBHomeworkNamespace.homeworkID.lower()],
                                                   homework[self.DBHomeworkNamespace.studentID.lower()],
                                                   homework[self.DBHomeworkNamespace.classID.lower()]]):
                        result = False
                else:
                    result = False
        else:
            result = False
    else:
        result = False

    return result
```

Obrázek 20: Uložení nového úkolu v databaseObj

4.2 Webové rozhraní

Rozhraní je postavené stejně jako databáze na python frameworku Flask. Implementace se skládá ze zjednodušených datových tříd a programovacího kódu pro každou podstránku.

5 MOBILNÍ APLIKACE

Pro mobilní aplikaci byla zvolena cílová platforma Android, pro jeho velkou popularitu, k naprogramování byl použit jazyk Kotlin.

5.1 Databáze

Správu dat v aplikaci zajišťují čtyři třídy: Database, DB_rem, DB_loc a HttpConnector.

HttpConnector zajišťuje komunikaci se serverem a mapování slovníku na JSON požadavky. Nad touto třídou je třída DB_rem, která mapuje objekty na slovníky, tato třída má naprogramovány všechny metody jako tzv. suspend funkce. Suspend funkce spolu s Coroutines tvoří v Kotlinu základní synchronizační primitiva. Suspend funkce se po zavolání uspí a čekají na probuzení, proto je tyto funkce nutné spouštět v Corountines, což jsou odlehčené verze vláken. Výhodou tohoto systému na rozdíl od používání jiných prostředků je velká flexibilita. Programátor může velmi jednoduše zvolit, v jakém druhu vlákna chce Corountines spustit, například na hlavním vlákně nebo vlákně speciálně určeném pro procesy pracující s diskem nebo sítí.

```
suspend fun remove(obj: Syncable): DBmessage{
    val connection = HttpConnector(userId,userHash,content)

    connection.openMessage(server,"DEL")
    when(obj.getType()){
        Syncable.Types.Homework -> connection.addMessage(
            mapOf<String,String>(
                "TYPE" to obj.getType().type,
                "HOMEWORK_ID" to (obj as HomeWork).homework_id
            )
        )
        else -> connection.addMessage(
            mapOf<String,String>(
                "TYPE" to obj.getType().type,
                "ID" to obj.getID()
            )
        )
    }

    return DBmessage(
        connection.sendResponse()?.sections?.get("content")?.values?.get("text")
    )
}
```

Obrázek 21: Suspend funkce pro odstranění objektů ze serverové databáze

Jelikož aplikace může fungovati i v offline režimu je v aplikaci přítomna lokální databáze SQLite se kterou se komunikuje pomocí DB_loc, návrh databáze opět reflektuje datové třídy.

Přidány byly pomocné tabulky, jež uchovávají, které objekty je třeba synchronizovat se serverem.

Serverovou i lokální databázi dohromady spravuje třída Database, která se pokouší ukládat data zároveň do lokální i do serverové databáze. Pokud ukládání dat na server selže, zaznamená do lokální databáze, které objekty bude při příštím pokusu o uložení synchronizovat. Všechny metody pro načítání dat jsou naprogramovány synchronně, jelikož se data stahují pouze při spuštění aplikace, metody pro ukládání dat jsou pak implementovány asynchronně. Podobně jako serverová implementace kontroluje přihlášení uživatele a jeho oprávnění k přístupu k datům.

5.2 Výměna dat v rámci aplikace

Mobilní aplikace je vytvořena pomocí návrhového vzoru MVVM (Model-view-viewmodel), ten nejen usnadňuje vývoj aplikace, ale v Androidu je přímo nutný. MVVM se skládá ze tří vrstev, modelu v tomto případě databáze, viewmodel a view, což je samotné zobrazení. Viewmodel je v aplikaci rozdělen do dvou vrstev globalModel a viewmodel pro každý fragment (viz 5.3). GlobalModel zajišťuje získávání a ukládání dat, je uložen v kontextu aktivity aplikace, jinak by muselo dojít k opakovanému načítání dat každým fragmentem. Samotné viewmodely fragmentů pak zajišťují přípravu dat pro fragmenty. Mají společné rozhraní zděděné z třídy BaseViewModel, které umožňuje dělat změny v datech bez jejich zápisu do databáze, fragment se pak může rozhodnout, zdali data funkcí commit zapíše do databáze, nebo je zruší.

```
val id = fragmentModel.requestAccess()
    ?: throw Exception("access to homeworkModel denied")
fragmentModel.save(
    BaseHomework(
        homework
    ), id)
fragmentModel.commit()
fragmentModel.releaseAccess()
```

Obrázek 22: Příklad ukládání dat do fragmentModelu, pokud by nedošlo k zavolání funkce commit a byla zavolána jen funkce releaseAccess byli by změny vytvořené funkcí save zrušeny.

Aktualizace všech dat ve fragmentech je navázána na viewmodels přes LiveData, tato třída, kterou Kotlin nabízí, umožňuje jednoduše sledovat aktualizace dat pomocí observerů.

K výměně dat mezi fragmenty lze použít právě již zmíněný viewmodel, nebo nově přidanou metodu která umožňuje zaznamenat data do tzv. FragmentManageru. Data se zapisují se specifickým klíčem a fragmenty v rámci FragmentManageru můžou na tyto klíče připojit své observery, tento způsob aplikace používá v místech, kde je potřeba poslat informaci jedním směrem například stisknutí tlačítka, nebo změna data v kalendáři.

```
override fun onResume(){
    super.onResume()

    parentFragmentManager.setFragmentResultListener(keyToListen, requireActivity(),
        FragmentResultListener{ _, _ ->
            val result = checkAndPost()
            onResult(result)
        }
    )
}
```

Obrázek 23: Metoda třídy GeneriTest, která kontroluje, zdali bylo stisknuto tlačítko „zkontrolovat“, které je uloženo v jiném fragmentu

Posledním způsobem je výměna dat mezi aktivitami aplikace (viz 5.3), toho lze dosáhnout dvěma způsoby v případě objektů, buďto implementací rozhraní Serializable, nebo rozhraní Parcelable. V aplikace je použita varianta s rozhraním Parcelable, kterou zajišťují třídy TeacherParcelable a StudentParcelable.

```
fun parcelListOfWords(listOfWords: List<Word>, parcel: Parcel, withToSyncParameter: Boolean = false){
    for(word in listOfWords){
        parcel.writeString(word.from)
        parcel.writeString(word.to)
        parcel.writeString(word.list)
        parcel.writeString(word.language)
        parcel.writeString(word.class_id)
        parcel.writeString(word.id)
        parcel.writeString(word.newList.toString())
        parcel.writeInt(word.stats.numSucces)
        parcel.writeInt(word.stats.numFail)
        parcel.writeLong(word.stats.lastTested.time)

        if(withToSyncParameter) {
            parcel.writeString(isToSync<Word>(word, add = true, remote = false).toString())
            parcel.writeString(isToSync<Word>(word, add = false, remote = false).toString())
            parcel.writeString(isToSync<Word>(word, add = true, remote = true).toString())
            parcel.writeString(isToSync<Word>(word, add = false, remote = true).toString())
        }
    }
}
```

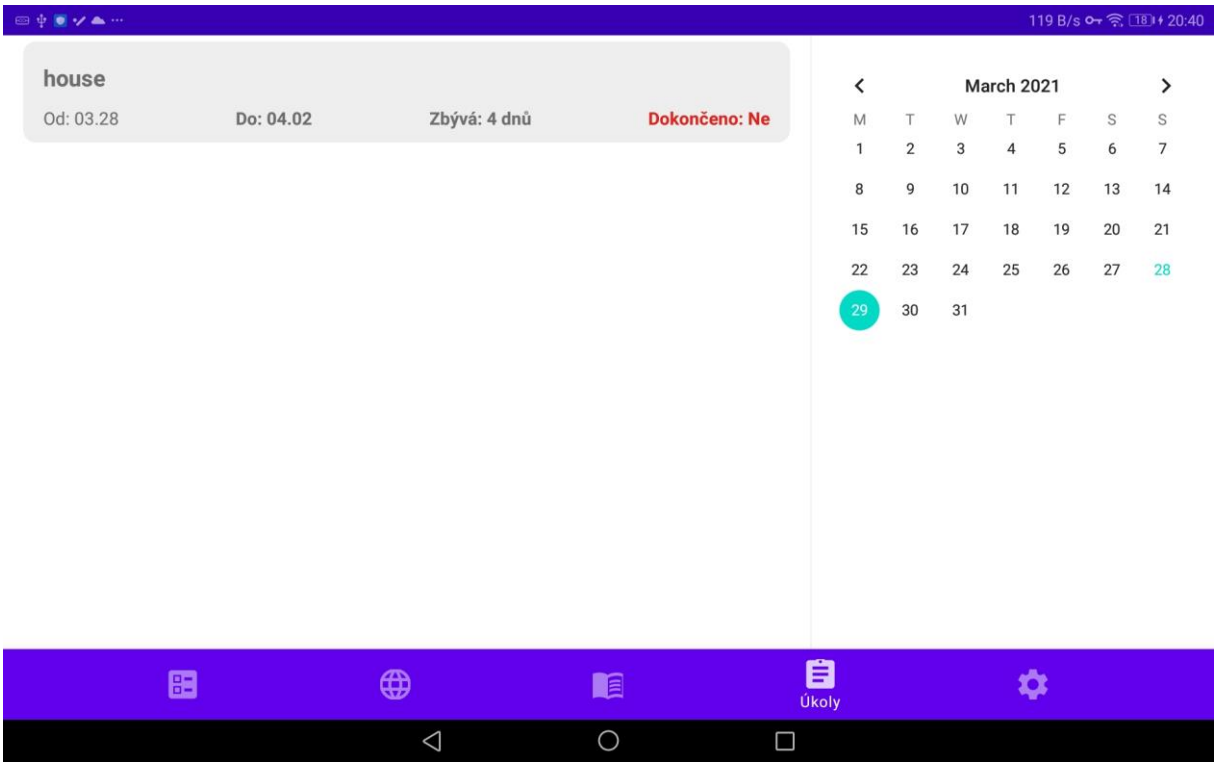
Obrázek 24: Parcelable třídy zapisují svoji vnitřní strukturu do tzv. parcel pomocí primitivních datových typů

5.3 Frontend

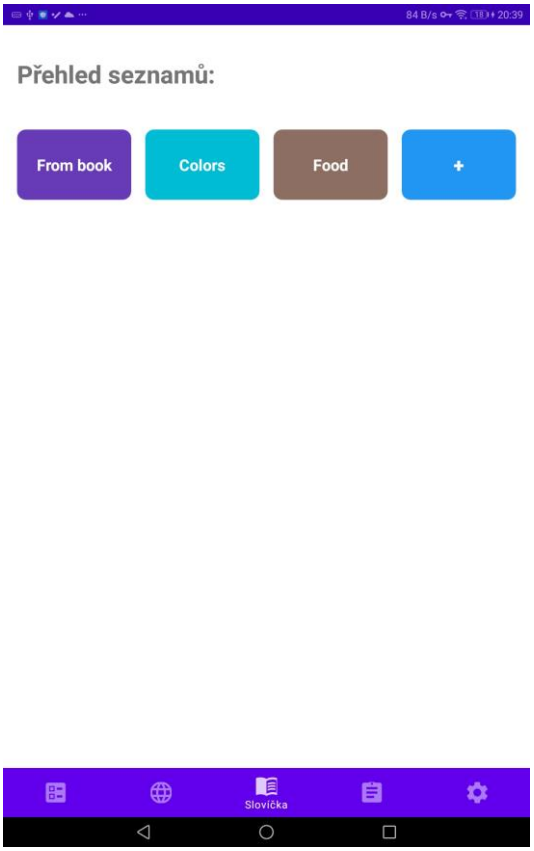
Frontend aplikace je rozdělen na resources, které obsahují popis rozložení grafických prvků, animace a obrázky, a na samotný řídící kód, který je propojen s GUI prvky pomocí ViewBinding a DataBinding.

5.3.1 Návrh

Aplikace byla navržena, aby byla co nejvíce intuitivní a jednoduchá, snaží se využívat orientace zařízení. V případě mobilu je aplikace ve fixní pozici orientována vertikálně, v případě tabletu může být orientována jak vertikálně, tak horizontálně. V případě vertikální orientace jsou například úkoly zobrazovány na týden, v případě horizontální jsou zobrazovány na den, kde datum se vybírá prostřednictvím kalendáře. Podobně se mění i zobrazení seznamů slov (viz níže). Pro lepší používání v noci je možno přepnout do nočního motivu, ten Android má defaultně zaimplementován a stačí jen definovat barvy pro noční a denní režim. Jeho přepínání probíhá přes funkci AppCompatDelegate.setDefaultNightMode.



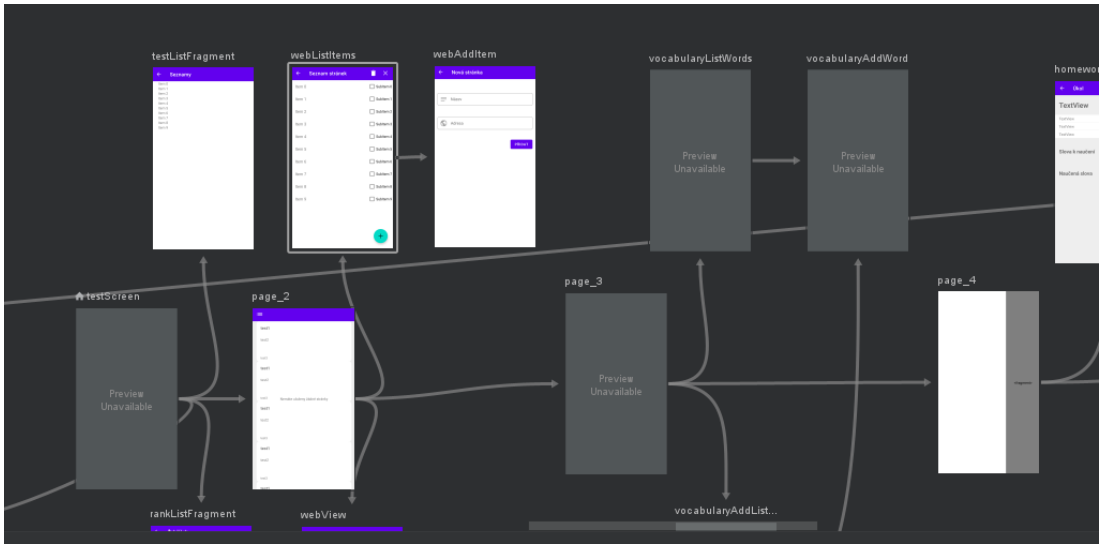
Obrázek 25: Horizontální zobrazení úkolů



Obrázek 26: Změna zobrazení buněk listů v případě tabletu

5.3.2 Navigace

Navigace v android aplikacích probíhá pomocí tzv. navigačního stacku. O tento navigační stack se stará NavigationController, který umožňuje definovat cesty mezi jednotlivými fragmenty v XML formuláři. V aplikaci je NavigationController propojený s appbarem, který je umístěný ve spodu aplikace a umožňuje přecházet mezi jednotlivými sekcemi.



Obrázek 27: Navigation controller

V případě potřeby tzv. popup stránek, které překryjí veškerou nabídku včetně appbaru, se spouští kód jako aktivita. Tento proces je tvořen jenom programově a není zaznamenán v XML formuláři. V aplikaci jsou jako popup aktivity naprogramovány třídy TestRunActivity (zprostředkovává test), Scanner (skenování textu), ale i aktivita učitelské části aplikace, která je spuštěna z hlavní aktivity, jež je pak následně odstraněna a přenechá řízení učitelské aktivitě.

5.3.3 BaseListAdapter

RecycledView je hlavní součástí celé aplikace a je prakticky implementován téměř ve všech sekcích. Jedná se o grafický prvek, který umožňuje zobrazení jiných prvků v seznamu, konkrétně pak RecycledView má na rozdíl od klasického listu vylepšenou správu paměti a prvky, které nejsou zobrazeny na displeji, uvolňuje z paměti. Pro správnou funkci RecycledView na Androidu je potřeba mít dvě třídy, jedna, která se stará se o zobrazování prvků v listu (Adapter), a druhá, která reprezentuje View prvku v listu (ViewHolder).

```
data class BaseHeader(private val title: String, private val height: Int): BaseItem<FragmentVocabularyHeaderBinding> {
    override val layoutId: Int = R.layout.fragment_vocabulary_header
    override val itemID: Any = title

    override fun initViewBinding(view: View): FragmentVocabularyHeaderBinding {
        return FragmentVocabularyHeaderBinding.bind(view)
    }

    override fun getBinding(binding: FragmentVocabularyHeaderBinding) {
        binding.header.text = title
        binding.header.gravity = Gravity.CENTER_VERTICAL

        binding.root.LayoutParams.height = height
    }
}
```

Obrázek 28: Třída BaseHeader, která slouží pro zobrazování nadpisů sekcí implementována pomocí BaseItem

Jelikož by se ale často implementace opakovala pro každý list, byla použita obecná implementace Adaptéru (BaseListAdapter), která pracuje s obecnou třídou pro prvek (BaseViewHolder). Aby mohl BaseListAdapter jednoduše pracovat s různými typy dat, bylo definováno rozhraní BaseItem, které je implementováno každým prvkem, jejž je možno přidat do listu, například BaseButton (tlačítko), BaseHomework (buňka zobrazující informace o úkolu) atd. Pro fragmenty tak stačí jednoduše vytvořit seznam prvků, které se adaptéru pošlou funkcí submitList.

5.4 Přidávání slov

5.4.1 Přidání z webové stránky

Přidávání slov podržením prstu je naprogramováno pomocí JavaScriptu, který je spouštěn ve WebView, který stránku zobrazuje. Skript prvně přepočítá souřadnice z WebView na souřadnice se kterýma pracuje knihovna jQuery a následně získá nejvnitřnější HTML element ze zadaných souřadnic. Jelikož však element může obsahovat například celý odstavec, musí být nejprve každé slovo z elementu přiřazeno do vlastního elementu, následně skript zopakuje

prohledávání a extrahuje z elementu dané slovo. Jako tag, do kterého je slovo zabaleno, byl zvolen span, ten se nejvíce osvědčil a způsoboval nejméně potíží se zobrazováním stránky. Samotný výběr elementů je pak filtrován na elementy, které zpravidla obsahují text.

```
private fun getScript(x: String, y: String): String{
    var script = "var inX = $x * (window.innerWidth / ${binding.web.width});\n"
    script += "var inY = $y * (window.innerHeight / ${binding.web.height}); \n"
    script += """
    var theWindow = $(window)
    var theElement = document.elementFromPoint(inX, inY);
    var sentence = theElement.textContent;
    var tagName = theElement.tagName;
    var words = sentence.split(' ');
    if(words.length > 1 &&
        (tagName == 'P' || tagName == 'A' || tagName == 'H1' || tagName == 'H2'
        || tagName == 'H3' || tagName == 'H4' || tagName == 'H5' || tagName == 'UL' || tagName == 'LI'
        || tagName == 'B')){
        theElement.innerHTML = '';
        $.each(words, function(i, w){theElement.innerHTML += ("<span>" + w + " </span>")});
        theElement = document.elementFromPoint(inX, inY);
        theElement.textContent.split(' ')[0];
    }
    else if(words.length == 2 && tagName == 'SPAN'){
        theElement.textContent.split(' ')[0];
    }
    """

    return script
}
```

Obrázek 29: Funkce generující skript pro získání slova na daných souřadnicích

5.4.2 Skenování

Rozpoznávání textu je umožněno díky MLKitu. MLKit je framework, které využívá neuronové sítě a zároveň umožňuje jednodušeji vytvářet vlastí neuronové sítě. Nabízí širokou škálu nástrojů, například detekci obličejů, tracking objektů, skenování čárových kódu atd.

API frameworků je velmi jednoduché, rozpoznání textu probíhá přes jednu funkci, která bere jako parametr obrázek a vrací naskenovaný text rozdělený na bloky, řádky a jednotlivá slova. Toho je využito v aplikaci, která buďto nabízí skenování slov, ke kterým automaticky připojí překlad, nebo skenování jednotlivých řádků například v učebnici, k nimž pak uživatel sám připojí překlady.

5.4.3 Překlad slov

Aplikace umožňuje překlad slov během přidávání, v případě přidávání z webové stránky dochází k překladu automaticky. Pokud aplikace může, využívá online překladu, pokud není zrovna připojená k internetu, využije opět MLKit, který nabízí omezený překlad.

6 ROZPOZNÁNÍ ŘEČI

Jedním z testů je i poslechový test, který kontroluje, zdali student správně vyslovuje slovo nebo větu. Pro tento účel byla vytvořena a natrénována neuronová síť, která může běžet na omezeném výpočetním výkonu. Síť byla následně testována na dvou nejrozšířenějších frameworcích pro trénování neuronových sítí: Tensorflow (s použitím Kerasu) a PyTorch.

6.1 Výběr technologií

6.1.1 Tensorflow

Je nejpopulárnější Framework pro práci s neuronovými sítěmi, byl vytvořen společností Google a v roce 2015 byla poprvé vydána první verze. Tensorflow pracuje se statickými grafy, to znamená, že před začátkem samotného trénování Tensorflow převede grafy do statické podoby a poté až na nich spustí trénování. Díky tomuto přístupu odhalí Tensorflow případné nesrovnalosti ještě před samotným trénováním, umožňuje to i rychlejší průběh trénování.

Obrázek 30: Logo projektu Tensorflow

6.1.2 Keras

Je Framework, který zjednodušuje práci s neuronovými sítěmi. Keras v současné době (verze 2.4) podporuje už pouze Tensorflow jako backend. Díky Kerasu je práce s Tensorflow rychlejší, protože má implementovanou řadu funkcí a není nutné znát podrobně práci s Tensorflow, který má poměrně složité API.

Obrázek 31: Logo projektu Keras

6.1.3 PyTorch

PyTorch je Framework který byl vyvinut převážně společností Facebook, jeho první verze byla vydána v roce 2016. Na rozdíl od Tensorflow pracuje PyTorch s dynamickými grafy, které spouští až za běhu, díky tomu je snadnější debugging a zároveň i jednoduší práce s ním pro začátečníky, důsledkem toho je i pomalejší trénování, a odhalení chyb až za běhu.

Obrázek 32: Logo projektu PyTorch

6.1.4 Torchaudio

Je částí projektu PyTorch, obsahuje celou řadu nástrojů pro práci se zvukem, mimo jiné i pro práci s datovými sadami jako například LibriSpeech nebo CommonVoice. Nabízí i metody pro data augmentation v podobě časového a frekvenčního maskování (viz 6.4). Tato knihovna byla použita nejen v PyTorch ale i při práci s daty v Tensorflow.

6.1.5 Librosa a jlibrosa

Jsou knihovny určené pro práci se zvukovými daty, jedna z nich pro python a druhá pro Android. Využity byly především kvůli tomu, že torchaudio nemá podporu v případě Androidu. Jelikož jsou neuronové sítě velmi závisle na způsobu předzpracování dat, nemohou být spolu kombinovány, proto je pro generování mel spektrogramu použita knihovna Librosa v případě trénování a pro následný běh na Androidu jlibrosa.

6.1.6 Hardware

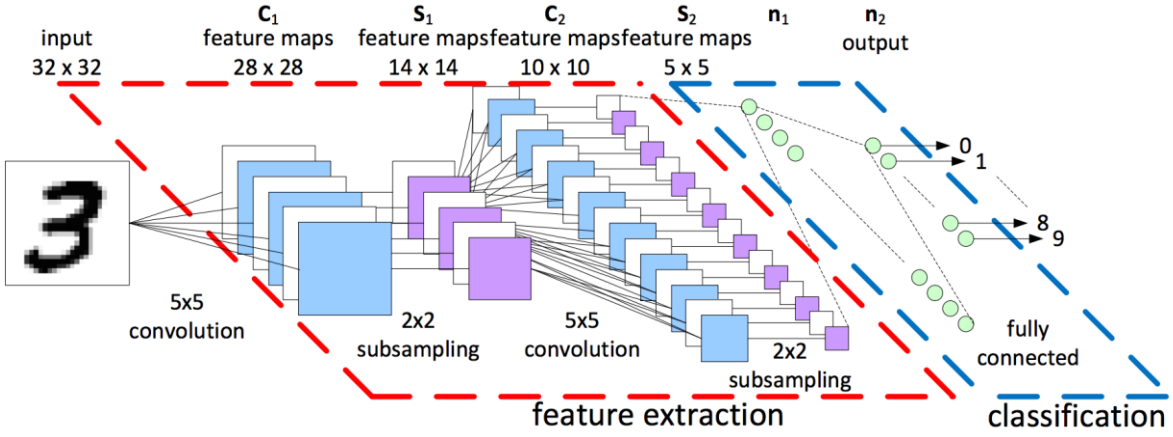
Tensorflow i PyTorch podporují akceleraci pomocí výpočtů na grafických kartách, které podporují technologii CUDA, Tensorflow podporují i další technologie například SYCL nebo ROCm, které přidávají podporu i pro jiné grafické karty. V tomto případě bylo využito technologie CUDA, konkrétně s grafickými kartami NVIDIA GTX 1060 a NVIDIA GTX 950 s třemi a dvěma gigabajty grafické paměti.

6.2 Popis použitých pojmů

6.2.1 Konvoluční vrstva

Aplikuje masku (kernel), jejíž velikost je možno zvolit, maskou postupně prochází celá vstupní data a hodnoty v dané oblasti se maskou vynásobí a následně sečtou a uloží jako jedna

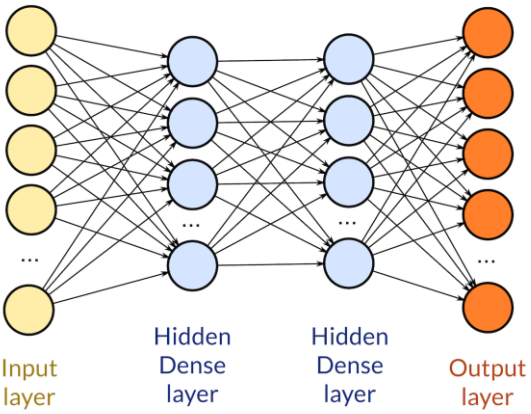
hodnota. Konvoluční vrstva může mít několik těchto masek, záleží na výstupním počtu kanálů konvoluční vrstvy. Krok (stride) konvoluční sítě určuje, kolik hodnot konvoluce přeskočí při posunu masky, a tím i o kolik se zmenší výstupní data. V případě kroku rovnu jedné se rozměr zachovává, jelikož konvoluce proběhne pro každou hodnotu (netýká se počtu kanálů).



Obrázek 33: Znázornění konvolučních vrstev spolu s klasifikátorem

6.2.2 Plně propojená vrstva

Je vrstva, ve které každý neuron je připojen na každý neuron předchozí vrstvy. Tyto vrstvy se můžou velmi dobře učit různé vzory ve vstupních datech, ale můžou se učit až příliš konkrétní vzory, což by vedlo k horším výsledkům. Proto se zpravidla tyto vrstvy používají uvnitř a na konci sítě.



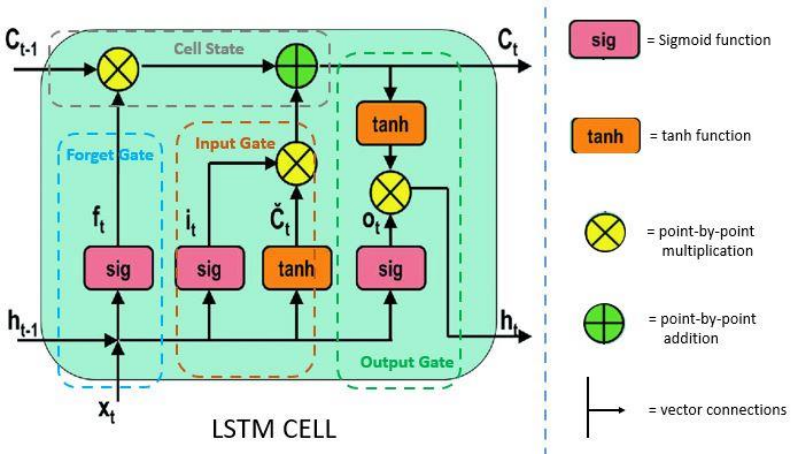
Obrázek 34: Jednoduchá neuronová sít s dvěma plně propojenými vrstvami

6.2.3 Rekurentní vrstva

Na rozdíl od jiných vrstev, které vyhodnocují data z časové osy nezávisle na sobě, je výsledek výstupu rekurentní vrstvy ovlivněn předchozími vstupy, díky tomu se může učit vzory v závislosti na čase, proto je využívána zejména pro sekvenční data, kde je časová návaznost například u rozpoznání řeči. Mezi dvě nejpoužívanější implementace patří LSTM (Long short-term memory) a GRU (Gated recurrent unit).

LSTM síť řeší problém mizejících gradientů, který byl problémem pro jednoduché rekurentní sítě. Její návrh byl představen v roce 1996. LSTM buňka se rozděluje na tzv. brány, které řídí tok dat v buňce. Konkrétně na:

- Vstupní bránu (input gate) – řídí průchod informací do paměťového bloku
- Zapomínající brána (forget gate) – určuje, která data jsou nepodstatná
- Výstupní brána (output gate) – určuje data, která jsou propagována do další buňky
- Stav paměťového bloku (cell state) – uchovává informaci

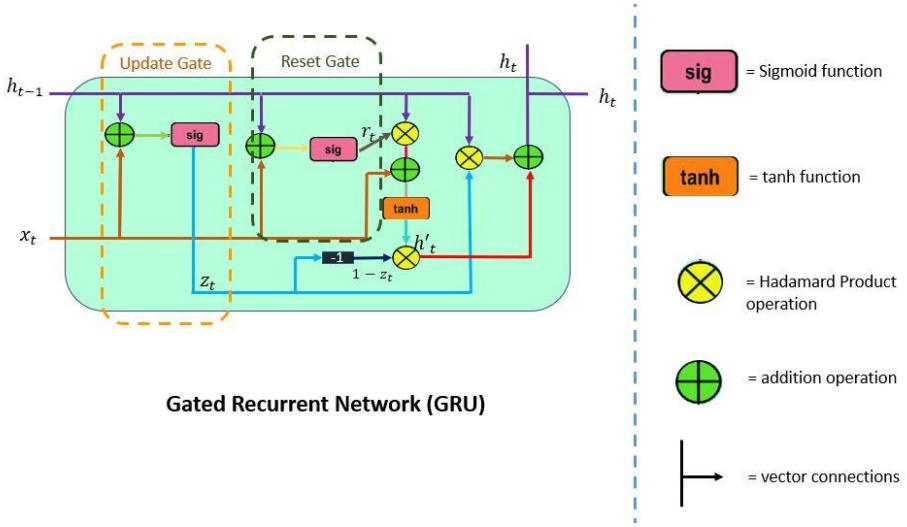


Obrázek 35: LSTM buňka

GRU (2014) síť je jednodušší a tím pádem i rychlejší, výsledky jsou s LSTM téměř srovnatelné. Vnitřní struktura se liší:

- Aktualizační brána (update gate) – určuje, množství dat, které bude z předchozí buňky využito
- Resetovací brána (reset gate) – plní podobnou funkce jako forget gate, určuje, které informace jsou nepodstatné

GRU postrádá stav paměťového bloku.



Obrázek 36: GRU buňka

6.2.4 Problém mizejících gradientů

Problém mizejících gradientů se projevuje při zpětné propagaci (backpropagation), kdy postupným derivováním výsledku loss funkce klesne hodnota k nule a neprovádí žádné výrazné změny ve váhách neuronů. Tento problém postihuje zejména hluboké sítě s velkým počtem vrstev sekvenčně zapojených za sebe.

Problém lze řešit několika možnými způsoby, nejčastěji použitím jiné aktivační funkce, jejíž derivace neklesá tak rychle jako například u sigmoid funkce. Dalším způsobem jsou tzv. skip connections, kde se vrstvy nezapojí sekvenčně za sebe, ale paralelně k sobě.

6.2.5 Problém exploze gradientů

Je opačný problém, kdy naopak velké hodnoty chyb z loss funkce můžou způsobit nastavení vah sítě způsobem, že výpočty budou probíhat ve velmi velkých číslech. Počítáním s velmi velkými čísly se stane síť nestabilní, a po čase se přestane učit, úplně například v důsledku přetečení proměnných.

Tento problém se nejčastěji řeší pomocí tzv. clipingu, kdy hodnoty výpočtů se mohou v síti pohybovat jen v rámci nějakého rozmezí, nebo použitím dynamického learning rate, kdy ze začátku, kdy se síť učí nejvíce, je učení zpomaleno.

6.2.6 Normalizační vrstva

Normalizační vrstva se snaží zobecnit data, aby se mohly ostatní vrstvy lépe učit vzory a nebyly rozhozeny extrémy v datech. Používají se dva druhy Batch a Layer normalizace. Batch normalizace provádí normalizaci napříč vrstvami v mini-batchi, kdežto Layer normalizace provádí normalizaci napříč vrstvami v celém průběhu trénování. Pro trénování rekurentních sítí nelze použít Batch normalizaci, protože ignoruje časovou závislost dat, proto je použita Layer normalizace.

6.2.7 Dropout vrstva

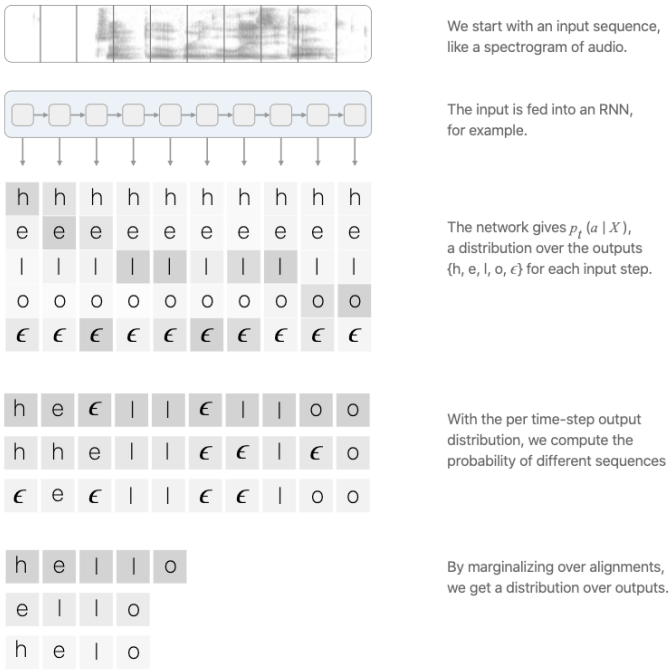
Zajišťuje, aby se síť nepřeučovala tím, že ignoruje výstupy některých neuronů předchozí vrstvy. A zbylé hodnoty pouští dál.

6.2.8 Softmax aktivace

Převádí hodnoty z neuronové sítě na hodnoty v rozmezí mezi nulou a jedničkou (pravděpodobnost).

6.2.9 CTC loss funkce

CTC (Connectionist Temporal Classification) je operace, která se snaží pro každý časový úsek z výstupu neuronové sítě určit konkrétní znak, jelikož může být znak přítomný ve více časových úsecích za sebou provede následně kolaps opakujících se znaků. Aby bylo možné zakódovat dva stejné po sobě jdoucí znaky, používá CTC tzv. blank znak, který slouží pro oddělení znaků.



Obrázek 37: CTC algoritmus

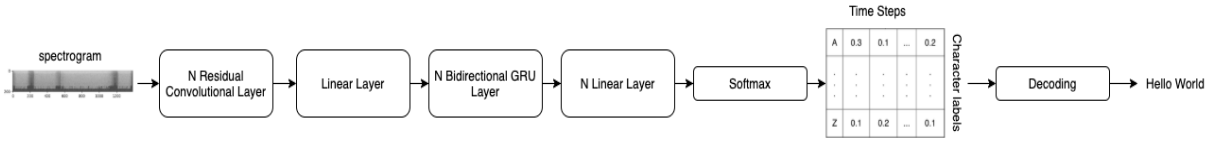
CTC loss funkce počítá chybu sítě po dopředné propagaci dat, učí síť nejen správně předpovídat znaky, ale i naučit jejich správně rozložení.

6.2.10 Levenshteinova vzdálenost

Též taky jako editační vzdálenost, je definována jako vzdálenost dvou řetězců na základě toho jaký je mimální počet operací vkládání, mazání a substituce. Slouží obecně k vyhodnocení výsledků algoritmů nebo neuronových sítí při práci s textem.

6.3 Návrh sítě

Při návrhu sítě bylo vycházeno ze sítě DeepSpeech2, která slouží právě k účelu rozpoznávání řeči, tato síť je však příliš výpočetně náročná pro mobilní zařízení, a proto musela být zjednodušena při snaze zachovat maximálně její úspěšnost.



Obrázek 38: Zjednodušený diagram DeepSpeech2

Prvním krokem bylo zmenšení počtu rekurentních vrstev (GRU) z pěti na čtyři a snížení počtu jejich parametrů. Druhým krokem bylo výměna prvního bloku sítě konvolučních vrstev se skip connections za různé kombinace konvolučních a plně propojených sítí (viz tabulka).

Struktura sítě - PytToch	CER	WER
2xCL, 2xDENSE, 4xRL, 1xTDD - Dropout: 0.1	29%	63%
2xCL, 2xDENSE, 4xRL, 1xTDD - Dropout: 0.2	30%	65%
2xCL,, 1xDENSE, 4xRL, 1xTDD - Dropout: 0.1	30%	64%
2xCL, 1xDENSE, 4xRL, 1xDENSE - Dropout: 0.1	30%	64%
CL(32 channels, stride=2), CL(32 channels) 2x DENSE, 4xRNN, 1xTDD - Dropout: 0.1	30%	64%
CL(16 channels, stride=2), CL(32 channels) 2x DENSE, 4xRNN, 1xTDD - Dropout: 0.1	30%	65%
1xCL, 2xDENSE, 4xRL, 1xTDD - Dropout: 0.1	31%	66%
2xCL, DENSE, CL(1D), DENSE, 4x RL, 1xTDD Dropout: 0.1	28%	61%
3xCL, 2xDENSE, 4xRL, 1xTDD - Dropout: 0.1	30%	65%

Tabulka 1: Vysvětlivky: CL – Convolutional layer, RL – Recurrent Layer, TDD – TimeDistributedDense

Struktura sítě - Tensorflow(Keras)	CER	WER
2xCL, 2xDENSE, 4xRL, 1xTDD - Dropout: 0.1 learning rate: 0.001	42%	82%
2xCL, 2xDENSE, 4xRL, 1xTDD - Dropout: 0.1	30%	65%
CL(16 channels, stride=2), CL(32 channels) 2x DENSE, 4xRNN, 1xTDD - Dropout: 0.1	31%	66%
2xCL, DENSE, CL(1D), DENSE, 4x RL, 1xTDD Dropout: 0.1	30%	64%

Tabulka 2: Výsledky pro stejnou síť, trénovanou v Kerasu

Nejlepší možností se po testování ukázalo použití na vstupu dvou konvolučních sítí s krokem jedna a následně jedné oddělovací plně propojené vrstvy společně s 1D konvoluční vrstvou a opět oddělovací plně propojenou vrstvou. Konvoluce 1D zlepšuje vlastnosti sítě při zpracovávání sekvenčních dat, jako je například text nebo řeč. Na konci sítě je místo samotné

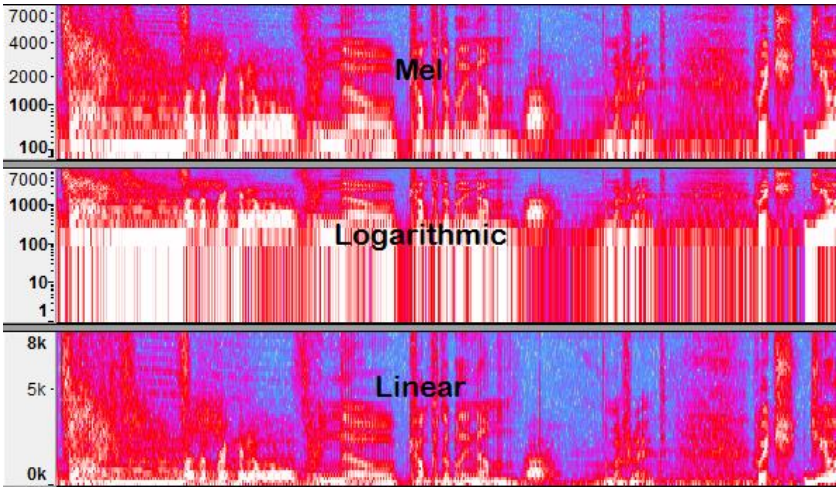
plně propojené vrstvy přítomna TimeDistributedDense (TDD) vrstva, ta aplikuje plně propojenou vrstvu (se stejnými váhami) na každý časový úsek v datech, proto musí být vstupní data alespoň 3D (batch, time, features).

V případě Kerasu byl zvolen learning rate poloviční oproti PyTorch, protože byl spuštěn na menší grafické kartě s menší batch velikostí.

Implementace sítě na obou platformách je velmi podobná, rozdíl je pouze v aktivačních funkcích, v případě PyTorch byla použita aktivační funkce GeLU, ta však není přítomna v Kerasu a proto byla nahrazena velmi podobnou funkcí SeLU. Liší se také aktivační funkce Softmax, ta je v případě PyTorch logaritmická, jelikož zlepšuje učení sítě, pro větší chyby je větší hodnota loss funkce než v případě běžného Softmaxu. Logaritmická implementace však tvořila v případě Kerasu problémy s explozí gradientů a hodnota loss funkce časem dosáhla hodnoty NaN, což znamenalo, že síť je nestabilní a neučí se. Přesné příčiny se bohužel nepodařilo odhalit a nepomohly ani techniky k opravení tohoto problému (viz 6.2.5). I přes tyto změny dosáhly obě sítě velmi podobných výsledků (viz tabulka).

6.4 Předzpracování dat

Neuronová síť se může učit rozpoznání řeči z různých forem vstupních dat. Jednou z nejznámějších variant je spektrogram, který vyjadřuje změnu hlasitosti a frekvence v čase, spektrogram je však příliš podrobný a zachycuje často i zvuky, které nejsou pro neuronovou síť podstatné. Pro účely trénování jsou data zpracována do tzv. Mel spektrogramu, nemá na rozdíl od klasického spektrogramu lineární stupnici, ale stupnici logaritmickou, speciálně pak mel spektrogram se řídí podle mel stupnice, která převádí frekvenci zvuku na výšku, jejíž jednotkou je mel. Mel stupnice je navržená tak, aby pro posluchače byl stejný rozdíl mezi například 50 a 100 mely a 1000 a 1050 mely, jelikož to samé neplatí pro klasickou stupnici v Hertzech v důsledku toho, že lidské vnímání zvuků není lineární.



Obrázek 39: Porovnání klasického a mel spektrogramu

Protože neuronová síť byla trénována na různých datových sadách, jsou data ještě převedením na mel spektrogram, pokud je nutné převzorkována na 16KHz.

Poslední operací provedenou nad daty před samotnou dopřednou propagací je tzv. data augmentation. Data augmentation je sada technik, která do dat přidává šum nebo je různým způsobem upravuje, aby vytvořila větší sadu, než ve skutečnosti je, tím má síť více dat na učení a může dosáhnout lepších výsledků. V případě zvuku jsou v náhodných úsecích jak na časové, tak na frekvenční ose vytvořeny mezery v datech (viz obrázek).

Souběžně s úpravou zvukového záznamu dochází k převodu transkripce z písmen na čísla například „a“ – 0.

6.5 Trénování

Obě implementace sítě byly trénované na třech datových sadách. Prvně byly obě sítě trénovány na datové sadě LibriSpeech, ta se skládá z tisíc hodin mluvené angličtiny, převzaté z audioknih. Konkrétně byla pro první trénink použitá sada train-clean-100, která obsahuje sto hodin mluvené angličtiny s převážně americkým přízvukem. Důvodem, proč je sada nazvána clean, je to, že obsahuje naprosté minimum chyb v transkripci textu k nahrávkám, což je pro počáteční trénování důležité. Tabulky uvedené výše byly zhotoveny po trénování na této sadě. Druhou sadou byla opět z LibriSpeech sada train-other-500, která obsahuje pět set hodin nahrávek s více přízvuky a s více chybami v transkripci. Poslední trénovací sadou byla sada z projektu CommonVoice organizovaným společností Mozilla, Common Voice Corpus 6.1. Ta obsahuje celkem 2181 hodin nahrávek, s větším množstvím přízvuků než LibriSpeech, právě kvůli tomu byla zvolena až jako poslední, jelikož klade na síť největší nároky.

Později se vyplatilo prohodit toto pořadí, protože datová sada Common Voice Corpus 6.1 je obecnější, a proto se síť neučila nechtěné vzory, jako je například přízvuk.

6.5.1 Porovnání

Oba frameworky byly porovnány z hlediska rychlosti a jednoduchosti.

Tensorflow byl z hlediska rychlosti při trénování pomalejší než PyTorch, což je však v rozporu s tvrzením v 6.1.1. Důvodem tohoto zpomalení bylo pravděpodobně vlastní kompilace Tensorflow, která byla na sestavě nutná z důvodu že release verze Tensorflow potřebovaly některé instrukce, které nebyly procesorem podporovány. Díky tomu byla vlastní kompilace hůře optimalizovaná než release verze. Konkrétně byl PyTorch schopen při mini-batch size velikosti pět projít pět set vzorků za tři čtvrtě minuty, v případě Tensorflow tomu bylo průměrně okolo dvou minut.

Z hlediska jednoduchosti si byly oba frameworky podobné, Tensorflow s Kerasem však vyžadoval méně programovacího kódu než PyTorch, důvodem je, že v Kerasu je už implementována trénovací a evaluační smyčka, kdežto v PyTorch musí být naprogramována. Keras zároveň nevyžaduje uvádět, kromě u vstupní vrstvy, vstupní počty parametrů vrstev, jelikož je dopočítá při převádění na statický graf. Problém nastává ve chvíli, kdy je potřeba implementovat v Kerasu nízkoúrovňovější kód, kdy je programátor často konfrontován se složitým API Tensorflow. Debugging v PyTorch je příjemnější, než v případě Tensorflow,

v PyTorch může programátor během trénování vypisovat libovolně obsah dat nebo váhy vrstev, stanovovat podmínky, tyto věci jsou v Tensorflow složitější, pro potřeby debuggingu při spuštěném trénování je potřeba používat speciální nástroje. V neposlední řadě je důležitá i uživatelská základna, ta je větší v případě Tensorflow, což může být rozhodující v případě začátků s neuronovými sítěmi.

```
def train(model, device, train_loader, criterion, optimizer, scheduler, epoch, iter_meter):
    model.train()
    data_len = len(train_loader.dataset)
    for batch_idx, _data in enumerate(train_loader):
        spectrograms, labels, input_lengths, label_lengths = _data
        spectrograms, labels = spectrograms.to(device), labels.to(device)

        optimizer.zero_grad()

        output = model(spectrograms)
        output = F.log_softmax(output, dim=2)
        output = output.transpose(0, 1)

        loss = criterion(output, labels, input_lengths, label_lengths)
        loss.backward()

        optimizer.step()
        scheduler.step()
        iter_meter.step()

        #torch.nn.utils.clip_grad_value_(model.parameters(), 20)

        if batch_idx % 100 == 0 or batch_idx == data_len:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(spectrograms), data_len,
                100. * batch_idx / len(train_loader), loss.item()))
```

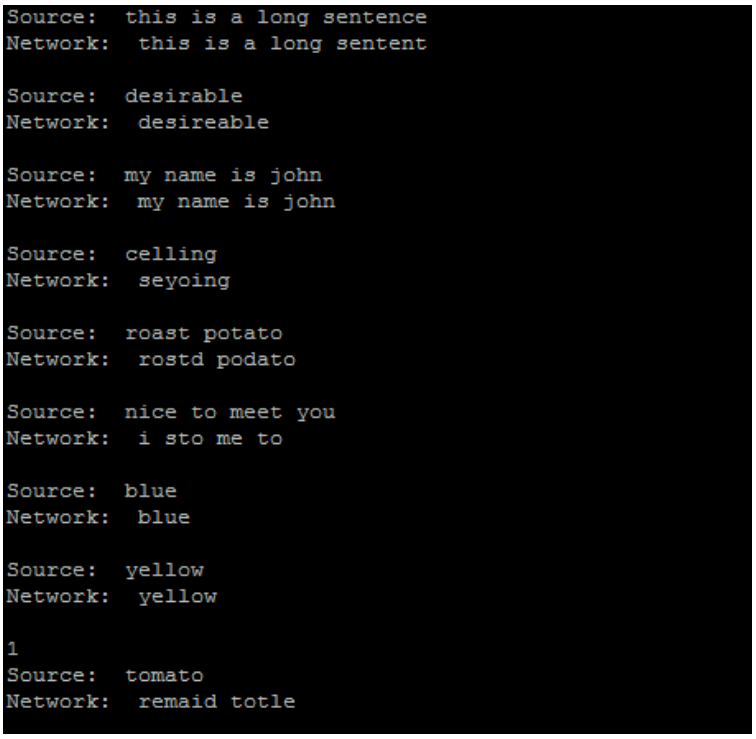
Obrázek 40: Python trénovací smyčka musí provést dopřednou propagaci a následně samostatně spustit loss funkci a optimizér, to provádí keras automaticky

6.5.2 Výsledky

Po natrénování na obou sadách dosáhla PyTorch implementace chybovosti pro CER 8% a pro WER 24%. Po prohození pořadí datových sad dosáhla síť chybovosti pro CER 5,5% a pro WER 18%. Pro Keras nejsou výsledky v době psaní této práce zatím známy.

Pro lepší srovnání byla vytvořena vlastní malá testovací sada, namluvená autorem práce. Tato sada, vzhledem k českému přízvuku, který byl v obou sadách zastoupen minimálně, dosáhla nejhorší úspěšnosti a to pro CER 28% a pro WER 67%, v tomto případě je více vypovídající CER parametr, jelikož sada obsahovala pouze přibližně dvacet slov/frází, a proto každé chybné slovo se citelně projevilo na WER parametru.

Níže je ukázka frází + jejich interpretace neuronovou sítí.



Obrázek 41: Příklad frází a jejich implementací neuronovou sítí. V případě slova „tomato“ lze vidět, že síť může v případě rušivých podmínek vyhodnotit slovo, které si s originálem není vůbec podobné, i díky tomuto dosahuje na této testovací sadě velké chybovosti v CER

6.5.3 Interpretace výstupu

Výstup sítě je ve formátu trojdimenzionálního pole, kde první osou je batch druhou čas a třetí pole o 29 prvcích, kde každý prvek reprezentuje pravděpodobnost znaku daného pozicí hodnoty v poli, například pro pozici nula je to znak „a“. Tento výstup je poté zpracován podobně, jako ho zpracovává CTC operace.

6.5.4 Evaluace

Evaluace probíhá pomocí dvou parametrů WER (Word error rate), která popisuje kolik procent slov je sítí špatně určeno a CER (Character error rate), který počítá, kolik procent znaků bylo sítí špatně určeno. Hodnoty uvedené v tabulkách jsou po trénování na první sadě, proto jsou tak vysoké.

6.6 Mobilní implementace

Oba frameworky nabízí velmi podobné API pro práci s neuronovými sítěmi na Androidu. Pro PyTorch je to implementace PyTorch mobile a pro Tensorflow je to TFLite. Z hlediska rychlosti je rychlejší PyTorch i jeho převod modelu do modelu podporovaného mobilní verzí byl jednoduší, než v případě Tensorflow. V případě PyTorch trvalo zpracování čtyř sekundového záznamu čtyři až čtyři a pul sekundy, v případě Tensorflow mezi šesti až šesti a půl sekundami. V případě PyTorch byla možnost, jednoduchého zrychlení pomocí kvantování modelu, konkrétně pak bylo použito tzv. dynamické kvantování, které je optimalizované především pro rekurentní sítě. Princip funkce spočívá v převedení vah rekurentních sítí na

celá čísla, přičemž aktivační vrstvy si tato čísla převádějí zpět do desetinné čárky. Tím se zejména ušetří čas na načítání dat z paměti. Kvantování modelu vždy snižuje jeho přesnost, v tomto případě bylo snížení zanedbatelné a jednalo se o řádově desetiny procent. Díky kvantovaní se vyhodnocovací doba zkrátila na 1,5 sekundy. Oba frameworky nabízí experimentálně možnost spustit síť na GPU[13],[14], tato možnost je však v raném stádiu vývoje u obou frameworků a nebylo ji možno použít kvůli rekurentním sítím.

7 ZÁVĚR

Hlavním cílem práce bylo vytvořit jednoduchou a intuitivní aplikaci, která pomůže studentům s učením se cizích slov. Druhým cílem bylo pak vytvoření neuronové sítě pro kontrolu výslovnosti, která dokáže běžet i na mobilním zařízení.

Aplikace je funkční jak pro práci offline, tak online. Nicméně stále ji čeká dlouhé odlaďování, aplikace občas nepředvídatelně spadne, nebo udělá nečekanou akci. Většina operací funguje bez problému, avšak některé je stále třeba ještě odladit.

Druhý cíl práce bylo vytvoření neuronové sítě, i tento cíl se domnívám, že byl splněn, neuronová síť je funkční jak ve vývojovém, tak v mobilním prostředí, její úspěšnost je dostačují pro účely kontroly výslovnosti (viz 6.5.2), avšak výsledky jsou o něco horší, než jsem z počátku očekával. I zde mohu říct, že síť může být vyvíjena dále, do budoucna bych rád experimentoval s nastavením různých parametrů, případně i se strukturou sítě, případně bych síť implementoval do sofistikovanějšího systému, jako je například Kaldi.

SEZNAM POUŽITÝCH ODKAZŮ

1. openslr.org. openslr.org [online]. Dostupné z: https://www.openslr.org/12
2. Common Voice. *Common Voice* [online]. Dostupné z: https://commonvoice.mozilla.org/
3. Tutorial — Flask Documentation (1.1.x). [online]. Dostupné z: https://flask.palletsprojects.com/en/1.1.x/tutorial/
4. Dropout in (Deep) Machine learning | by Amar Budhiraja | Medium. *Medium – Where good ideas find you.* [online]. Dostupné z: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5
5. Introduction to LSTM Units in RNN | Pluralsight. *Pluralsight | The tech workforce development company* [online]. Dostupné z: https://www.pluralsight.com/guides/introduction-to-lstm-units-in-rnn
6. Normalization Techniques in Deep Neural Networks | by Aakash Bindal | Techspace | Medium. *Medium – Where good ideas find you.* [online]. Dostupné z: https://medium.com/techspace-usict/normalization-techniques-in-deep-neural-networks-9121bf100d8
7. Building an end-to-end Speech Recognition model in PyTorch. *AssemblyAI: The #1 Speech-to-Text API for Developers* [online]. Dostupné z: https://www.assemblyai.com/blog/end-to-end-speech-recognition-pytorch
8. Create a basic coroutine – tutorial | Kotlin. *Kotlin Programming Language* [online]. Dostupné z: https://kotlinlang.org/docs/coroutines-basic-jvm.html#let-s-run-a-lot-of-them
9. A detailed example of data generators with Keras. [online]. Dostupné z: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly
10. Any PyTorch function can work as Keras' Timedistributed? - #4 by miguelvr - vision - PyTorch Forums. *PyTorch Forums* [online]. Dostupné z: https://discuss.pytorch.org/t/any-pytorch-function-can-work-as-keras-timedistributed/1346/4
11. XML Parsing in Android using SAX Parser - GeeksforGeeks. *GeeksforGeeks | A computer science portal for geeks* [online]. Dostupné z: https://www.geeksforgeeks.org/xml-parsing-in-android-using-sax-parser/
12. Pytorch Mobile Performance Recipes — PyTorch Tutorials 1.8.1+cu102 documentation. [online]. Copyright © Copyright 2017, PyTorch. [cit. 31.03.2021]. Dostupné z: https://pytorch.org/tutorials/recipes/mobile_perf.html?highlight=mobile
13. [online]. Dostupné z: https://pytorch.org/blog/prototype-features-now-available-apis-for-hardware-accelerated-mobile-and-arm64-builds/
14. TensorFlow Lite on GPU. *TensorFlow* [online]. Dostupné z: https://www.tensorflow.org/lite/performance/gpu_advanced
15. [online]. Copyright © 2012 [cit. 31.03.2021]. Dostupné z: https://blog.miguelgrinberg.com/post/running-your-flask-application-over-https

16. Stack Overflow - Where Developers Learn, Share, & Build Careers. *Stack Overflow - Where Developers Learn, Share, & Build Careers* [online]. Dostupné z: https://stackoverflow.com/
17. Documentation | Android Developers. *Android Developers* [online]. Dostupné z: https://developer.android.com/docs
18. PyTorch documentation — PyTorch 1.8.1 documentation. [online]. Copyright © Copyright 2019, Torch Contributors. [cit. 31.03.2021]. Dostupné z: https://pytorch.org/docs/stable/index.html
19. How to implement a dark theme on Android | by Kamal Faraj | Ideas by Idean | Medium. *Medium – Where good ideas find you.* [online]. Dostupné z: https://medium.com/ideas-by-idean/how-to-implement-a-dark-theme-on-android-bc615fccf2d8
20. Material Design [online]. Dostupné z: https://material.io/develop/android
21. A detailed example of data generators with Keras. [online]. Dostupné z: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly
22. MITRENGA, Michal. *Konvoluční neuronová síť pro segmentaci obrazu* [online]. Brno, 2018 [cit. 2021-03-31]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=171096. Bakalářská práce. Vysoké učení technické Brně.
23. NOVÁČIK, Bc. Tomáš. *Rekurentní neuronové sítě pro rozpoznávání řeči* [online]. Brno, 2016 [cit. 2021-03-31]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=132425. Diplomová práce. Vysoké učení technické Brně.
24. HAMERNÍK, Bc. Pavel Hamerník. *Využití hlubokého učení pro rozpoznávání textu v obrazu grafického uživateského rozhraní* [online]. Brno, 2019 [cit. 2021-03-31]. Dostupné z: https://dspace.vutbr.cz/xmlui/bitstream/handle/11012/187290/final-thesis.pdf?sequence=3&isAllowed=y. Diplomová práce. Vysoké učení technické v Brně.

Použité obrázky:

1. [ONLINE]. DOSTUPNÉ Z: HTTPS://ASSETS-GLOBAL.WEBSITE-FILES.COM/5FBD459F3B05914CF70496D7/5FDFA5D8F483A0EEF6EF7CB1_BHOBFDVTCGCQKTTP.PNG

2. *Audacity Manual* [online]. Copyright © [cit. 31.03.2021]. Dostupné z: https://manual.audacityteam.org/m/images/e/ed/mel_log_linear_spectrogram_annotated.png
3. [online]. Dostupné z: https://assets-global.website-files.com/5fbd459f3b05914cf70496d7/5fdfa5d90d23543010468e25_mQEJyPYmfQAHalLK.png
4. *imgix - Image Processing On-Demand, Served By CDN* [online]. Copyright ©Z [cit. 31.03.2021]. Dostupné z: https://pluralsight2.imgix.net/guides/8a8ac7c1-8bac-4e89-ace8-9e28813ab635_3.JPG
5. *imgix - Image Processing On-Demand, Served By CDN* [online]. Dostupné z: https://pluralsight2.imgix.net/guides/c02c6196-7452-4095-9215-c4d57a9dd1a4_1.JPG

6. [online]. Copyright © [cit. 31.03.2021]. Dostupné z: https://stackabuse.s3.amazonaws.com/media/deep-learning-in-keras-building-a-deep-learning-model-1.png
7. [online]. Dostupné z: https://camo.githubusercontent.com/269e3903f62eb2c4d13ac4c9ab979510010f8968/68747470733a2f2f7261772e6769746875622e636f6d2f746176677265656e2f6c616e647573655f636c617373696669636174696f6e2f6d61737465722f66696c652f636e6e2e706e673f7261773d74727565
8. [online]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/thumb/a/ae/Keras_logo.svg/1200px-Keras_logo.svg.png
9. [online]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/thumb/1/11/TensorFlowLogo.svg/800px-TensorFlowLogo.svg.png

Seznam obrázků:

Seznam tabulek: