UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Experimenty s OCR na báze hlbokého učenia

DIPLOMOVÁ PRÁCA

2020 Peter Kulcsár Szabó

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Experimenty s OCR na báze hlbokého učenia

DIPLOMOVÁ PRÁCA

Študijný program:	Aplikovaná informatika
Študijný odbor:	11378 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Vedúci práce:	RNDr. Andrej Lúčny, PhD.

Bratislava 2020 **Peter Kulcsár Szabó**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Peter Kulcsár Szabó
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Experimenty s OCR na báze hlbokého učenia
Experiments with OCR using Deep Learning

Anotácia: Práca má výskumný charakter a je z oblasti hlbokého učenia a vychádza z jednoduchšej architektúry hlbokej neurónovej siete použitej pre OCR individuálneho písmena. Skúša analógiu triku, ktorý pomáha zvýšiť úspešnosť klasického OCR na OCR pomocou hlbokého učenia.

Cieľ: Cieľom práce je vyskúšať trénovanie OCR na báze hlbokej neurónovej siete a vyhodnotiť či rozšírenie klasifikácie znakov o príznaky ako sú počet veľkých dier v stredovej vertikálnej línii, prítomnosť vonkajších a vnútorných oblúkov a rohov, zúženie v strede zľava a sprava, záseky v obvodovej línii zľava v dolnej polovici a sprava v dolnej polovici, spojitosť vnútra s ľavou či pravou stranou a podobne zlepší dosiahnutú úspešnosť. Pritom príznaky možno požadovať na výstupnej vrstve zo siete alebo priamo v jej latentom priestore. Cieľom práce nie je dosiahnuť čo najvyššiu úspešnosť, ale zistiť či danou úpravou architektúry úspešnosť klesá alebo stúpa.

Literatúra:
Chollet, F.: Deep learning v jazyku Python, Grada, 2019
Learning OpenCV 3, Computer Vision in C++ with the OpenCV Library By Gary Bradski, Adrian Kaehler, O'Reilly Media, 2016
Smith, R.: An Overview of the Tesseract OCR Engine. ICDAR '07 Proceedings of the Ninth International Conference on Document Analysis and Recognition, Volume 02, pp. 629-633
Lúčny, A: Čítanie textu na pneumatike, KUZ 2018, M.U. Brno
learnopencv.com

Poznámka: Platforma: OpenCV, Keras

Kľúčové slová: OCR, hlboké učenie, počítačové videnie, OpenCV, Keras

Vedúci: RNDr. Andrej Lúčny, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 08.10.2019

Abstrakt v štátnom jazyku

KULCSÁR SZABÓ, Peter: Experimenty s OCR na báze hlbokého učenia [Diplomová práca], Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, Katedra aplikovanej matematiky a štatistiky; školiteľ: RNDr. Andrej Lúčny, PhD., Bratislava, 2021, XX s

Lorem ipsum

Klúčové slová: xx

Abstract

KULCSÁR SZABÓ, Peter: Experiments with OCR using Deep Learning [Master Thesis], Comenius University in Bratislava, Faculty of Mathematics, Physics and Informatics, Department of Applied Mathematics and Statistics; Supervisor: RNDr. Andrej Lúčny, PhD., Bratislava, 2021, XX p.

Lorem ipsum

Key words: xx

Obsah

Úvod

Úvod do problematiky: OCR ťažko rozpoznáva písmená v niektorých prípadoch ako je popísané v článku [2]. Napírklad z pneumatík.

OCR budeme riečiť na báze deep learningu a pozorovaním latentného spacu popísaného v článku [1]

1 Prehlad problematiky

Optické rozpoznávanie znakov (OCR - optical character recognioton) je rozšírená technológia na rozpoznávanie textu na obrázkoch, ako sú naskenované dokumenty a fotografie. Technológia OCR sa používa na konverziu prakticky všetkých druhov obrázkov obsahujúcich písaný text (písaný, písaný rukou alebo vytlačený) do digitálnej formy.

Technológia OCR sa stala populárnou na začiatku 90. rokov, keď sa pokúšali digitalizovať historické noviny. Odvtedy táto technológia prešla niekoľkými vylepšeniami. V súčasnosti poskytujú riešenia takmer dokonalú presnosť.

Pravdepodobne najznámejším prípadom použitia OCR je konverzia tlačených papierových dokumentov na strojovo čitateľné textové dokumenty. Keď naskenovaný papierový dokument prejde spracovaním OCR, text dokumentu sa dá upravovať textovými procesormi ako Microsoft Word alebo Google Docs. Predtým, ako bola dostupná technológia OCR, bola jedinou možnosťou digitalizácie tlačených papierových dokumentov ručné prepisovanie textu. Nielenže to bolo časovo náročné, ale aj nepresnosti a preklepy boli časté.

OCR sa často používa ako „skrytá“ technológia, ktorá poháňa mnoho známych systémov a služieb v našom každodennom živote. Medzi menej známe, ale rovnako dôležité prípady použitia technológie OCR patrí automatizácia zadávania údajov, indexovanie dokumentov pre vyhľadávače (google), automatické rozpoznávanie ŠPZ, ako aj pomoc nevidiacim a slabozrakým osobám.

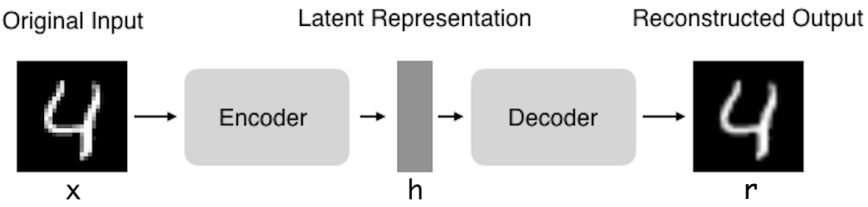
Technológia OCR sa ukázala ako nesmierne užitočná pri digitalizácii historických novín a textov, ktoré sa teraz zmenili na plne prehľadateľné formáty a uľahčili a urýchlili prístup k týmto historickým textom.

1.1 Autoenkóder

Autoenkódery (AE) sú neurónové siete, ktorých cieľom je kopírovať ich vstupy do ich výstupov. Fungujú tak, že komprimujú vstup do reprezentácie latentného priestoru a potom rekonštruujú výstup z tejto reprezentácie. Tento druh siete sa skladá z dvoch častí:

- Enkóder: Toto je časť siete, ktorá komprimuje vstup do reprezentácie latentného priestoru. Môže to byť reprezentované kódovacou funkciou h = f (x)
- Dekóder: Cieľom tejto časti je rekonštruovať vstup zo znázornenia latentného priestoru. Môže to byť reprezentované dekódovacou funkciou r = g (h)

Autoenkóder ako celok teda môžeme opísať funkciou $g(f(x)) = r$, kde chceme, aby r bolo čo najbližšie pôvodnému vstupu x.



Obr. 1: Základný princíp autoenkóderov

Ak by jediným účelom autoenkóderov bolo kopírovať vstup na výstup, boli by zbytočné. Predpokladáme, že trénovaním autoenkódera na kopírovanie vstupu na výstup získa latentná reprezentácia h užitočné vlastnosti.

To sa dá dosiahnuť vytvorením obmedzení pre úlohu kopírovania. Jedným zo spôsobov, ako získať užitočné vlastnosti z trénovacích dát, je obmedziť h tak, aby malo menšie rozmery ako x, v tomto prípade sa autoenkóder nazýva nedokončený (undercomplete). Trénovaním neúplného autenkódera prinútime, aby sa naučil najdôležitejšie vlastnosti údajov z trénovacích dát. Ak má autoenkóder príliš veľkú kapacitu, môže sa naučiť kopírovať bez extrahovania akýchkoľvek užitočných informácií o distribúcii údajov. Môže to tiež nastať, ak je rozmer latentnej reprezentácie rovnaký ako vstup, a v prípade neúplného (overcomplete) uskutočnenia, kde je dimenzia latentnej reprezentácie väčšia ako vstup. V týchto prípadoch sa dokonca aj lineárny kódovač a lineárny dekodér môžu naučiť kopírovať vstup na výstup bez toho, aby sa dozvedeli niečo užitočné o distribúcii údajov. V ideálnom prípade by sa dalo úspešne trénovať ľubovoľnú architektúru autoenkódera, pričom by sa vybrala dimenzia kódu a kapacita kódovacieho zariadenia a dekodéra na základe zložitosti distribúcie, ktorá sa má modelovať.

1.2 Klasifikátor

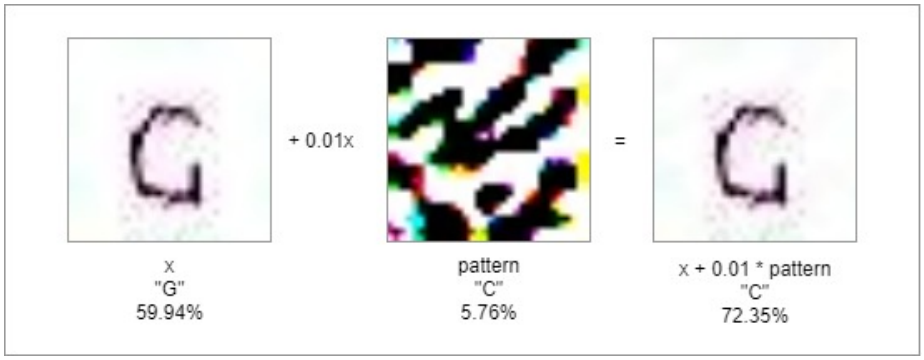
Ako funguje klasifikátor =¿ odskenutia decodera

1.3 Adversarial samples

V roku 2013 Szegedy navrhol v článku [8] koncept kontradiktórnych (adversarial) vzoriek. Tieto vzorky sa konštruujú pridaním k pôvodnej vzorke drobný šum, ktorý ľudské oko nedokáže rozpoznať, tak že cielený model klasifikuje do nesprávnej triedy kontratiktórnu vzorku.

Predpokladajme, že máme model M a pôvodnú vzorku x, ktorú model dokáže správne klasifikovať; $M(x) = y_{true}$. Pridaním malej perturbácie k tejto vzorke vzniká vzorka x', ktorá je podobná vzorke x, ale model M ju nevie klasifikovať správne; $M(x') \neq y_{true}$. Príklad je znázornený na obrázku č. 2, kde sa pôvodná

vzorka klasifikovala správne ako písmeno "Gä po pridaní šumu sa klasifikuje ako písmeno "C". Môžeme si všimnúť že pre vzorku x' je si model istejší s klasifikovanou hodnotou.



Obr. 2: Príklad kontradiktórnej vzorky

Príčina exitencie

Sem treba dopísať text

Generovanie kontradiktórnych vzoriek

```
def create_adversarial_pattern(model, image, label):
    image = tf.cast(image, tf.float32)

    with tf.GradientTape() as tape:
        tape.watch(image)
        prediction = model(image)

        loss = tf.keras.losses.MSE(label, prediction[0])

    gradient = tape.gradient(loss, image)

    signed_grad = tf.sign(gradient)

    return signed_grad

def generate_adversarial(model, image, true_label, epsilon = 0.1,
    isFeature = False):
    return image + epsilon * create_adversarial_pattern(model,
    image, true_label, isFeature)
```

Úryvok 1: Generovanie kontradiktórnych vzoriek

2 Implementačné prostriedky

2.1 Tensorflow a Keras

Tensorflow

Pri prací s neurovnovymi sieťami potrebujeme framework v ktorom zadefinujeme ich architektúru a dokážeme ich efektívne spúštať na GPU. My na tento používame Tensorflow. Tento framework vyvinuli vo firme Google. Tensorflow je založený na grafovej štruktúre tak, že každý uzol v grafe reprezentuje jednu operáciu a hrany grafu popisujú vzťah medzi uzlami. Tensorflow je napísaný v programovacom jazyku C++, ale podporuje aj aplikačné rozhranie v jazyku Python, ktoré sme využili v našej práci.

Keras

Pri implementácii našich experimentov sme použili knižnicu Keras, ktorá je nadstavba nad frameworkom Tensroflow. Využitím Kerasu sa jednoducho definovali modeli a ľahko sa nám experimentovalo s neurónovými sieťami. Pri vytváraní modelov môžeme použiť dve triedy, a to Sequential a Graph. Zavolaním metódy add sa do týchto objektov pridajú jednotlivé vrstvy. Pomocou Kerasu sa dá vytvoriť dopredná, konvolučná a rekurentná neurónová sieť. V našej práci sme implementovali konvolučnú neurónovú sieť. Vrstvy ktoré sme využili sú konvolučné, podvzorkovacie, aktivačné a úplne prepojené (fully-connected).

Toto preformulvoať ešte: Keras okrem pridávania regularizačných parametrov umožňuje používať aj rôzne typy inicializácií váh, ďalej povoľuje ukladanie, načítanie aj vizualizáciu modelov. Keras taktiež obsahuje základné metódy pre narábanie so sekvenciami a prirodzeným jazykom. Pokiaľ chce človek počas trénovania využiť monitorovanie a logovanie štatistík, v Kerase sú dostupné nastaviteľné funkcie-callbacky. Pre model vieme napríklad nastaviť trénovanie tak, aby trvalo pokiaľ sa bude chybová funkcia zmenšovať - tento callback sa volá EarlyStopping. Funckie callbacku vieme nastaviť v metóde fit objektu Sequential alebo Graph, a to použitím parametru callback.

2.2 Konvolučná neurónová sieť

Konvolučné neurónové siete sa používajú hlavne v oblasti počítačového videnia, a preto sa zaoberáme týmto typom neurónovej sieti v našej práci. Tieto siete sú biologicky inšpirované vizuálnym coretxom u cicavcov [6].
Niečo o konv neur sietach

Konvolučná

Základným prvkom konvolučných sietí je konvolučná vrstva. Cieľom konvolučnej vrstvy je extrahovať zo vstupného obrazu prvky vysokej úrovne, ako sú napríklad hrany. Konvolučné siete nemusia byť obmedzené iba na jednu konvolučnú vrstvu.

Prvá konvolučná vrstva je zodpovedná za zachytenie nízkoúrovňových príznakov, ako sú okraje, farba, orientácia gradientu atď. S pridanými vrstvami sa architektúra prispôsobuje aj príznakom vysokej úrovne, čo nám poskytuje sieť, ktorá má porozumenie obrázkov v datasete, podobne ako človek.

Pooling

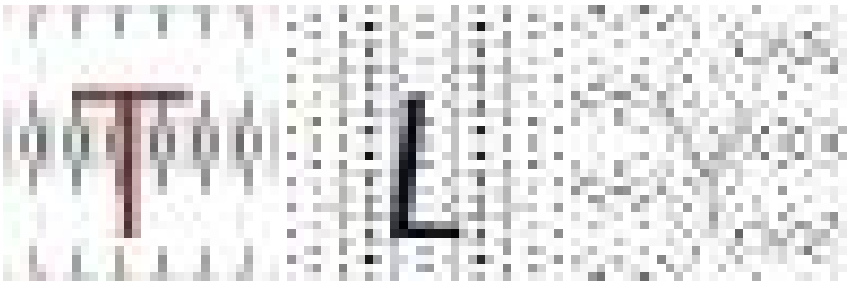
Ďalšou dôležitou vrstvou v konvolučných sieťach je Pooling vrstva. Táto vrstva sa používa pre zmenšenie vstupov, a teda na podvzorkovanie vstupu z predchádzajúcich vrstiev. Dané zmenšenie je spôsobené znížením počtov parametrov v ďalších vrstvách. Znížený počet parametrov vo vrstvách má za dôsledok urýchlenie celkového tréningu takejto siete. Generalizáciu modelu zlepšujú menšie vrstvy a to z hľadiska počtu parametrov. Podobne ako v konvolučnej vrstve aj pri tejto sú špecifikované dva parametre, a to krok a veľkosť okna/filtru. Aby sa oblasti neprekrývali, používa sa veľkosť filtra 2 s rovnakou veľkosťou kroku. Pooling vrstva môže byť definovaná podľa typu filtrov. Ak filter z určitej oblasti vyberá maximálnu hodnotu, ide o MaxPooling vrstvu. Pri vrstve AvgPooling ide o filter, ktoré jednotlivé hodnoty spriemeruje. Podobne ako pri aktivačnej vrstve aj táto vrstva slúži iba na transformáciu vstupu na výstup, žiadne váhy sa neuchovávajú.

2.3 Dáta

Pre experimenty s optickým rozpoznávaním znakov sme si vytvorili dáta pomocou Pythonovskej knižnice TextRecognitionDataGenerator [7]. Všetky dáta majú rozmer 32x32 pixlov a obsahujú tri kanály pre červenú, zelenú a modrú farbu. Celkovo sme vygenerovali 10700 dát, z ktorých 1800 dát je testovacích.

Knižnica, ktorú sme využili nám umožnila generovať dáta, ktoré sú z rôznych typov písmen, sú farebné, majú rušivé pozadie alebo pridaný šum na celkový obraz. Ukážky z vygenerovaných dát sú na obraze č. 3 a 4. Pre účely experimentov sme vygenerovali aj dáta, ktoré sú aj pre ľudské oči ťažko rozpoznateľné. (napríklad tretí znak na obraze č. 4, kde sa nachádza písmeno Y)

Obr. 3: Dáta bez pozadia



Obr. 4: Dáta s pozadím

2.4 Prostredie

sem príde info ako je postavené prostredie + ako sa používa

3 Výskum

V tejto kapitole sa budeme venovať výskumnej práci, od plánu až po realizáciu.

3.1 Plán

Naša výskumná práca pozostávala z nasledovných častí:

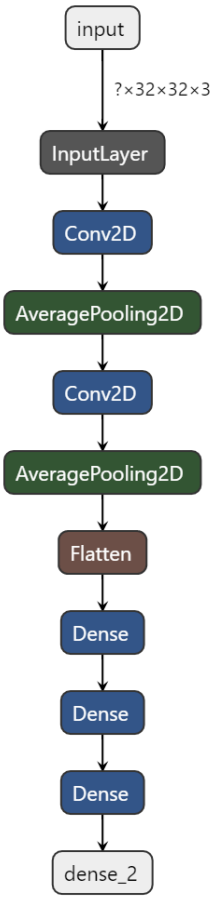
- vytvorenie prostredia
- príprava datasetu
- výber základného modelu
- definícia charakteristík znakov
- rozšírenie základného modelu
- experimenty s rozšíreným modelom
- Generovanie ADV samples s FFT

3.2 Výber základného modelu

V tejto časti práci si predstavíme základné modely neurónových sietí, s ktorými sme pracovali. Pri výbere modelu sme brali nasledovné aspekty do úvahy: hĺbku modelu, počet parametrov, zložitosť modelu. Keďže naša práca sa zaoberá s experimentom s hlbokými neurónovými sieťami, tak sme si vybrali sieť ktorá má viacero vrstiev, ktoré sa hĺbkou zmenšujú. Počet parametrov hralo rolu, kvôli trénovaniu modelu. Našim cieľom je zistiť či vieme pomôcť modelu, tým že vyžiadame od neurónovej sieti extrakciu viditeľných príznakov pre ľudské oči. Aby sme tento jav vedeli odsledovať, tak sme si vybrali jednoduché siete. Vrámci našej práci sme použili dva základné modeli. Ako prvý model je LeNet, ktorý bol uvedený v článku [3], ktorý bol publikovaný v roku 1998. Druhý základný model sme vytvorili na základe modelu LeNet s pridaním aktivačných funkcii, ďalšej konvolučnej vrstvy a zmenou parametrov. V nasledujúcich častiach si detailne opíšeme základné modely.

3.2.1 Model LeNet-5

Táto časť podrobnejšie popisuje architektúru modelu LeNet-5, konvolučná neurónová sieť použitá v našich experimentoch ako základný model, ktorý sa následne rozšíri. Architektúra je znázornená na grafe 5.



Obr. 5: Architektúra LeNet-5

LeNet-5 obsahuje 7 vrstiev, nerátajúc vstupnú vrstvu, pričom všetky obsahujú trénovateľné parametre (váhy). Vstupom je obraz v rozlíšení 32 x 32 pixelov.

V nasledujúcom texte sú konvolučné vrstvy označené ako Cx, čiastkové vzorkovacie vrstvy sú označené ako X a úplne spojené vrstvy sú označené ako Fx, kde x je index vrstvy.

Vrstva C1 je konvolučná vrstva so šiestimi výstupnými mapami. Každá jednotka na mape je spojená k vstupným pixelom s okolím 5x5. Veľkosť výstupných máp je 28x28. Vrstva C1 obsahuje 456 trénovatelných parametrov. Výstupom vrstvy je 6 máp s veľkosťou 28x28.

Vrstva S2 je podvzorkovacia so šiestimi výstupnými mapami veľkosťou 14x14. Každá jednotka na výstupe je spojená s 2x2 okolím na vstupe. Tieto podoblasti sa neprekrývajú a to zaručí že sa výška a šírka máp zníži na polovicu oproti vrstve C1 na 14x14.

Nasledujúca vrstva C3 je konvolučná a používa rovnakú veľkosť podoblasti ako vrstva C1. Oproti vrstve C1, táto vrstva obsahuje až 16 výstupných máp a teda celkový počet trénovatelných parametrov v tejto vrstve je 2416. Výstupom vrstvy je 16 máp s veľkosťou 10x10.

Rovanko ako po prvej konvolučnej vrstve, tak aj po druhej konvolučnej vrstve nasleduje poolingová vrstva S4. Rovanko ako vrstva S2, aj táto vrstva požíva okolie veľkosťou 2x2 a teda výstupom vrstvy je 16 máp s veľkosťou 5x5.

Po tejto vrstve nasleduje flatten vrstva, ktorá výstup z vrstvy S4 pretransformuje do vektora veľkosti 400.

Nasledujú úplne prepojené vrstvy s počtom neurónov 120, 84 a 36. Na poslednej vrstve je 36 neurónov, lebo máme 36 tried, do ktorých klasifikujeme vstup. Počty trénovatelných parametrov prislúchajúci k úplne prepojeným vrstvám sú 48120, 10164 a na poslednej vrstve 3060. Výstup z druhej úplne prepojenej vrstvy budeme chápať ako latentný priestor, ktorý budeme skúmať v ďalších kapitolách našej práce. Na poslednú vrstvu sa aplikuje aktivačná funkcia softmax, ktorá pretransformuje výstup poslednej vrstvy na vektor obsahujúci pravdepodobnosti, akou daný vstup patrí do danej triedy. Suma výstupu po aktivačnej funkcie je 1.

Celkový počet trénovatelných parametrov je 64216.

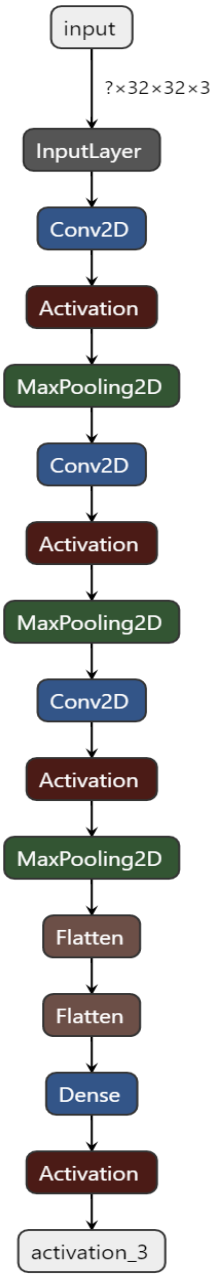
3.2.2 Vlastný model

V tejto časti si detailne popíšeme architektúru nášho vlastného modelu, s ktorým budeme tiež experimentovať s rovnakým postupom ako pri modeli LeNet-5 detailne popísaný v kapitole 3.2.1. Pri tvorbe architektúry nášho modela sme vychádzali z architektúry modelu LeNet. Pridali sme ďalšiu konvolučnú vrstvu, pridali sme aktivačné vrstvy a zmenili hyperparametre konvolučných vrstiev. Architektúra je znázornená na grafe 6.

Rovnako ako aj v základom LeNet-5 modeli, aj v našom modeli je prvá vrstva V1 konvolučná. Táto vrstva má na výstupe 32 máp s rozmermi 30x30. Pri konvolúcii bolo využité okolie veľkosti 3x3. V tejto vrstve sa nachádza 896 trénovatelných parametrov.

Po konvolučnej vrstve nasleduje aktivačná vrstva s funkciou ReLU (rectified linear unit) a následne podvzorkovacia vrstva pomocou funkcie maximum s rozmerom 2x2. Na výstupe vrstvi S2 je 32 máp s rozmerom 15x15.

Nasleduje ďalšia konvolučná vrstva s rovnakým počtom máp a veľkosťou konvolučnej mapy 3x3 ako vo vrstve C1. Na výstupe je 32 máp s veľkosťami 13x13. Počet trénovatelných parametrov je 9248.



Obr. 6: Architektúra Vlastného modelu

Po druhej konvolučnej vrstve nasleduje aktivačná a podvzorkovacia vrstva s rovnakými hyperparametrami ako po prvej konvolučnej. Po nej nasleduje posledná konvolučná vrstva C3, ktorá má na vstupe z maxppolingovej vrstvy 32 máp veľkosti 6x6 a na výstupe 64 máp s rozmerom 4x4. Tretia konvolučná vrstva má 18496 trénovatelných parametrov.

Po poslednej podvzorkovacej vrstve je výstup 64 máp s rozmerom 2x2. Po nej nasleduje flatten vrstva, ktorá pretransformuje výstup na vektor veľkosti 256. Tento vektor reprezentuje latentný priestor. V našom výskume sa budeme zameriavať na tento latentný priestor. Po latentnom priestore je úplne prepojená vrstva s výstupným vektorom veľkosti 36 na ktorý sa aplikuje rovnako ako pri modeli LeNet-5 softmax funkcia.

Model má celkovo 37892 trénovatelných parametrov.

Implementácia LeNet-5 modelu

Na nasledujej vzorke kódu č. 2 je zobrazené ako sa vytvára neurónova sieť pomocou knižnice Keras. Na prvom riadku sa inicializuje model typu Sequential. Trieda sequential umožnuje vytvoriť model, ktorý sa skladá z vrstiev a každá vrstva má jeden vstupný a jeden výstupný tensor. Na riadkoch 3 a 7 sa pomocou funkcie add pridáva konvolučná vrstva. Na riadkoch 4 a 8 sa pridávajú podvzorkovacie vrstvy a následne sa pridávajú ešte vrstvy flatten a dense (úplne prepojená vrstva). Ako posledná vrstva je úplne prepojená vrstva s 84 neurónmi, ktorej výstup je latentný priestor.

```
1  model = Sequential()
2  # First convolution Layer
3  model.add(Conv2D(6, (5, 5), input_shape=input_shape))
4  model.add(AveragePooling2D(pool_size=(2,2)))
5
6  # Second convolution Layer
7  model.add(Conv2D(16, (5, 5)))
8  model.add(AveragePooling2D(pool_size=(2, 2)))
9
10 model.add(Flatten())
11 model.add(Dense(120))
12
13 # Latent space
14 model.add(Dense(84))
```

Úryvok 2: Implementácia modelu LeNet-5 v Keras-e

Implementácia vlastného modelu

Na ďalšej vzorke č. 3 je znázornené, ako sme vytvorili základ nášho modelu. Rovnako sa využíva trieda Sequential pre vytvorenie modelu. Oproti vzorke č. 2 sa tu nachádzajú aj aktivačné vrstvy na riadkoch 4, 9 a 14. Latentný priestor je výstupom vrstvy flatten na riadku č. 18.

```
model = Sequential()
# First convolution Layer
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# First convolution Layer
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Third Convolution Layer
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Latent space
model.add(Flatten())
```

Úryvok 3: Implementácia vlastného modelu v Keras-e

3.3 Charakteristika písmen

V našej práci sme sa zameriavali na príznaky písmen, ktoré sú ľudskými očami rozpoznateľné. Na základe toho sme si vybrali 13 príznakov, pre ktoré sme vytvorili tabuľku 1, ktorá obsahuje dáta o tom, či dané písmeno má príznak alebo nie. V prípade počte dier, príznak vyjadruje počet a nie existenciu príznaku.

Stĺpce tabuľky 1 v poradí sú nasledovné: roh vľavo hore, roh vpravo hore, roh vľavo dole, roh vpravo dole, zaoblený roh vľavo hore, zaoblený roh vpravo hore, zaoblený roh vľavo dole, zaoblený vpravo dole, počet dier, zúženie a intersection.

Znak	TL	TR	BL	BR	TL	TR	BL	BR	#H	RL	RR	IL	IR
0					1	1	1	1	1				
1													
2					1	1						1	1
3					1	1	1	1				1	
4									1				
5	1						1	1				1	1
6					1	1	1	1	1				1
7												1	
8					1	1	1	1	2	1	1		
9					1	1	1	1	1			1	
A									1				
B	1		1			1		1	2		1		
C					1		1						1
D	1		1			1		1	1				
E	1		1										1
F	1												1
G					1	1	1						1
H													
I													
J							1	1					
K											1		
L			1										
M	1												
N													
O					1	1	1	1	1				
P	1		1			1			1				
Q					1	1	1	1	1				
R	1					1			1		1		
S					1	1	1	1				1	1
T												1	1
U							1	1					
V													
W													
X										1	1		
Y										1	1		
Z												1	1

Tabuľka 1: Charakteristika znakov

3.4 Data generátor

V našej práci sa zaoberáme s dátami typu obraz. Tieto obrazy majú tri kanály pre tri základné farby, a to červené, zelená a modrá. Pre trénovanie neurónovej sieti sa načítavajú tieto dáta do pamäte. Aby sa pamäť počítača nepreplnila kvôli veľkému množstvu načítaných dát sa používa pri trénovaní dáta generátor. Keďže sa neurónová sieť trénuje v batchoch, tak je vhodné načítavať dáta ad-hoc keď je potrebné pre trénovanie.

V našom prípade sme implementovali vlastný data generátor na základe Keras triedy Sequence. Existuje viacero implementácii data generátorov. Data generátor založený na Tensorflowe dokáže násobne zrýchliť načítanie obrázkov oproti generátoru zaleženom na Kerase. Porovnanie je dostupné v článku [5]. Pre naše účely, ale tieto generátory boli nevhodné, lebo generovali duplicitné dáta. Pre vyhodnotenie našich experimentov sme teda nedostávali skutočné výsledky. Z týchto dôvodov sme implementovali generátor na základe triedy Sequence, ktorá zaručuje, že dáta sa pre vyhodnotenie trénovanie neurónovej sieti načítajú presne raz a všetky. Výsledná implementácia je znázornená na 4.

Implementácia generátora

```
class CustomGenerator(tf.keras.utils.Sequence):

    def __init__(self, image_filenames, labels, batch_size):
        self.image_filenames, self.labels = image_filenames, labels
        self.batch_size = batch_size

    def __len__(self):
        return math.ceil(len(self.image_filenames) / self.
    batch_size)

    def __getitem__(self, idx):
        batch_x = self.image_filenames[idx * self.batch_size:(idx +
     1) * self.batch_size]
        batch_y = self.labels[idx * self.batch_size:(idx + 1) *
    self.batch_size]

        return np.array([imread(file_name) / 255 for file_name in
    batch_x]), np.array(batch_y)
```

Úryvok 4: Stratová funkcia pre klasfikačnú vetvu

3.5 Rozšírený model

V tejto kapitole sa oboznámime s rozšíreným modelom, ktorý je priamym rozšírením základného modelu LeNet-5 z kapitoly č. 3.2.1 a vlastného modelu opísaný v kapitole č. 3.2.2. Zadefinujeme si požiadavky na latentný priestor, oboznámime sa s rozvetvením výstupu modelu, chybovými funkciami a funkciami na výpočet úspešnosti.

Zo základného modelu pre naše experimenty si vytvoríme dva rozšírené modely. V jednom modeli z latentného priestoru sú dva priame výstupy pre klasifikátor znaku a vyhodnocvač extrakcii príznakov. V druhom modeli pred výstup do klasifikátora je vložená fully conntected vrstva.

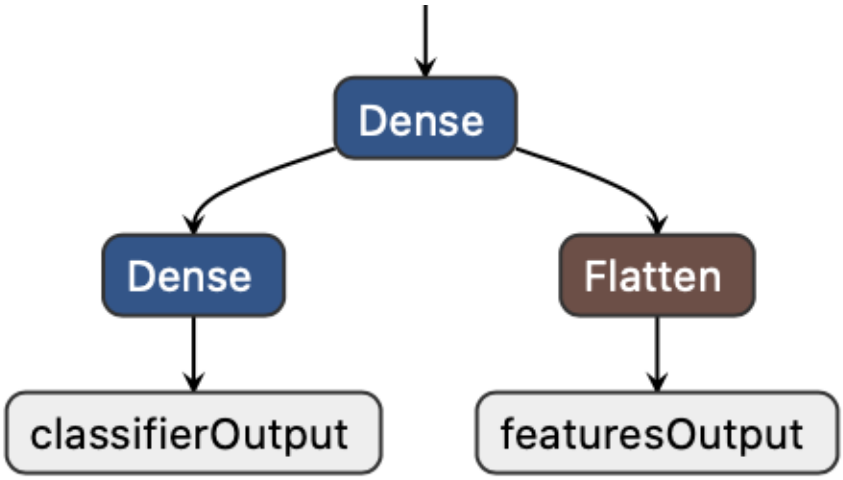
Požiadavky na latentný priestor

V našej práci sa zameriavame na to, aké príznaky zo vstupných dát je schopná neurónová sieť extrahovať. V základných modeloch sa v latentnom priestore nachádzajú čísla, ktoré reprezentujú príznaky, ale nevieme ich interpretovať. Našim cieľom je donútiť neurónovú sieť aby v latentnom priestore na daných pozíciach boli nami žiadané príznaky. V kapitole č. 3.3 sme si zadefinovali príznaky, ktoré chceme pozorovať. Naša požiadavka je, aby sa neurónova sieť naučila extrahovať tieto príznaky tak, že budú na začiatku latentného priestoru. Pri modeli LeNet-5 má latentný priestor veľkosť 84 a v našom modeli má veľkosť 256. V oboch prípadoch teda sa rezervuje prvých 13 hodnôt pre príznaky.

Trénovanie modelu

Popísať ako prebieha trénovanie modelu v princípe =¿ back propagation, treba loss function

V našej práci sme implementovali dve stratové (loss) funkcie, ku ktorým sa rátal gradient a vstupovali rovnakou váhou do trénovania neurónovej sieti. Váhy ktoré definujú ako daný gradient vplýva na trénovanie je možné definovať. Vďaka týmto hyper parametrom rastie komplexita neurónovej sieti. Pre obe stratové funkcie sú definované osobitné výstupné vrstvy z latentného priestoru. Tieto dve výstupné vrstvy sa rozvetvia z latentného priestoru. Rozvetvenie je znázornené na obrázku č. 7. Latentný priestor sa nachádza v strede, z ktorého na ľavo je výstupný vrstva pre klasifikátor a na pravo je výstupná vrstva pre stratovú funkciu pre príznaky.



Obr. 7: Rozvetvenie latentného priestoru

V nasledujej vzorke kódu č. 5 si ukážeme, ako sme implementovali v Kerase rozvetvenie výstupu z latentného priestoru pomocou Keras Functional API.

Keras Funcitonal API umožnuje vytvoriť modely, ktoré sú flexibilnejšie ako rozhranie tf.keras.Sequential API. Functional API dokáže pracovať s modelmi s nelineárnou topológiou, zdieľanými vrstvami a dokonca s viacerými vstupmi alebo výstupmi.

Na riadku č. 1 sa inicializuje model podľa kódu, ktorý sme si ukázali v 2 alebo 3. Na riadku č. 4 je voliteľná vrstva dense. Následne sa definujú dva výstupné vrstvy na riadkoch č. 4 a 8. Na riadku č. 11 je definovanie nového modelu pomocou Functional API, kde sa definuje vstup zo základného modelu ako model.inputs a výstupy sa definujú ako pole vrstiev, ktoré sme vytvorili na riadkoch č. 6 a 8.

```
1  model = getDefaultModel(input_shape)
2
3  # Optional
4  dense1 = Dense(units=64, activation="relu")(model.layers[-1].
       output)
5
6  classifierOutput = Dense(num_classes, activation="softmax", name="
       classifierOutput")(dense1)
7
8  featuresOutput = Flatten(name="featuresOutput")(model.layers[-1].
       output)
9
10 # define the new model
11 model = Model(inputs=model.inputs, outputs=[classifierOutput,
```

```
    featuresOutput])
```

Úryvok 5: Rozvetvenie latentného spacu implemetnované v Keras-e

Pre trénovanie modelu sme pre obe vetvy definovali stratovú funkciu a funkciu pre vyhodnotenie úspešnosti. Obe funkcie sa definujú štandardne s dvoma vstupnými parametrami. Prvý parameter je pravdivý znak(label) a druhý parameter je odhad neurónovej sieti pre znak.

3.5.1 Klasifikačná vetva

Stratová funkcia

Príznaková vetva má na vstupe výstup z latentného priestoru. Do funkcie vstupy sú y_{true} pravdivá hodnota veľkosti $n + k$, kde k je počet tried a n je počet príznakov a y_{pred} predikovaná hodnota neurónovou sieťou.

Keďže pri vytváraní modelu ako výstupná aktivačná funkcia je softmax:.

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

a pre vypočítanie straty sme použili vstavanú funkciu CategoricalCrossentropy:

$$CE = -\sum_i^C t_i \log(f(s)_i)$$

, ktorá v prípade že label je vo formáte one-hot-encoding, teda daný vstup patrí presne do jednej triedy, tak sa výpočet dá zredukovať na:

$$CE = -\log \frac{e_p^s}{\sum_j^C e_j^s}$$

, kde p je index tej triedy do ktorej tréningová vzorka patrí. Keďže do funkcie na vstupoch sú matice ktoré nemajú zhodné veľkosti, treba z pravdivej hodnoty čítať iba prvých k hodnôt, čo zodpovedá informácii o tom, že dáta na vstupe do ktorej skupiny patria. Výsledná implementácia je zobrazená na **??**

```
1 def classifierLossFunction(y_true, y_pred):
2     k = 36 # number of classes
3     cce = tf.keras.losses.CategoricalCrossentropy()
4     return cce(y_true[:,0:k], y_pred)
```

Úryvok 6: Stratová funkcia pre klasfikačnú vetvu

Vyhodnocovacia funkcia

V klasifikačnej vetve sme ako vyhodnocovaciu funkciu použili zabudovanú funkciu categorical_accuracy, ktorá vyhodnocuje či sa pravdivá hodnota rovná s predikovanou hodnotou. Ak sa hodnoty rovnajú tak je hodnota 1, v inom prípade 0.

```
def classifierAccuracy(y_true,y_pred):
    k = 36 # number of classes
    return tf.keras.metrics.sparse_categorical_accuracy(tf.math.
    argmax(y_true[:,0:k],1), y_pred)
```

Úryvok 7: Vyhodnocovacia funkcia pre klasfikačnú vetvu

3.5.2 Príznaková vetva

Stratová funkcia

Príznaková vetva má na vstupe výstup z latentného priestoru. Základom stratovej funkcie sme použili nasledujúcu funkciu:

$$L(y_{true}, y_{pred}) = \frac{1}{m}(y_{pred} - y_{true})^2$$

Keďže v našom prípade vstup y_{pred} je vektor veľkosti m, a príznakov máme n, kde platí že n je menšie ako n ($n < m$). V trénovacích dátach y_{true} je vektor veľkosti $k + n$, kde n je počet príznakov a k je počet tried do ktorých klasifikujeme vstupné dáta. Príznaky sa nachádzajú na konci vektora y_{true}

Kvôli nezhodám vo veľkostiach a umiestnenia informácii vo vektoroch, sme zmenili funkciu na nasledovnú:

$$L(y_{true}, y_{pred}) = (y_{pred}[: n] - y_{true}[k :])^2$$

Ďelenie s počtom dát v latentom priestore sme odobrali, lebo efekt na trénovanie je regulovaný váhami stratových funkcii znázornený na riadku X v XX.
Vstupy do funkcii majú odlišný typ. Pradikovaná hodnota má typ float32. Kvôli výpočtom je potrebné pravidvú hodnotu y_{true} transformovať do tohto typu. Výsledná implementáciá stratovej funkcii pre príznakovú vetvu je zobrazená na 8

```
def featuresLossFunction(y_true, y_pred):
    k = 36 # number of classes
    n = 13 # number of features
    return K.square(y_pred[:,:n] - tf.cast(y_true, tf.float32)[:,k
    :])
```

Úryvok 8: Stratová funkcia pre príznakovú vetvu

Vyhodnocovacia funkcia

Vyhodnocovacia funkcia pre príznakovú vetvu je založené na stratovej funkcii. Zmena oproti stratovej funkcii je, že sa zosumujú. Čím nižšie je výstupová hodnota funkcie, tým lepšie vie neurónová sieť extrahovať príznaky zo vstupných dát. Výsledná implementáciá vyhodnocovacej funkcii pre príznakovú vetvu je zobrazená na 9

```
def featuresAccuracy(y_true, y_pred):
    k = 36 # number of classes
    n = 13 # number of features
    return K.sum(K.square(y_pred[:,:n] - tf.cast(y_true, tf.float32
    )[:,k:]))
```

Úryvok 9: Vyhodnocovacia funkcia príznakovú vetvu

3.5.3 Implementácia

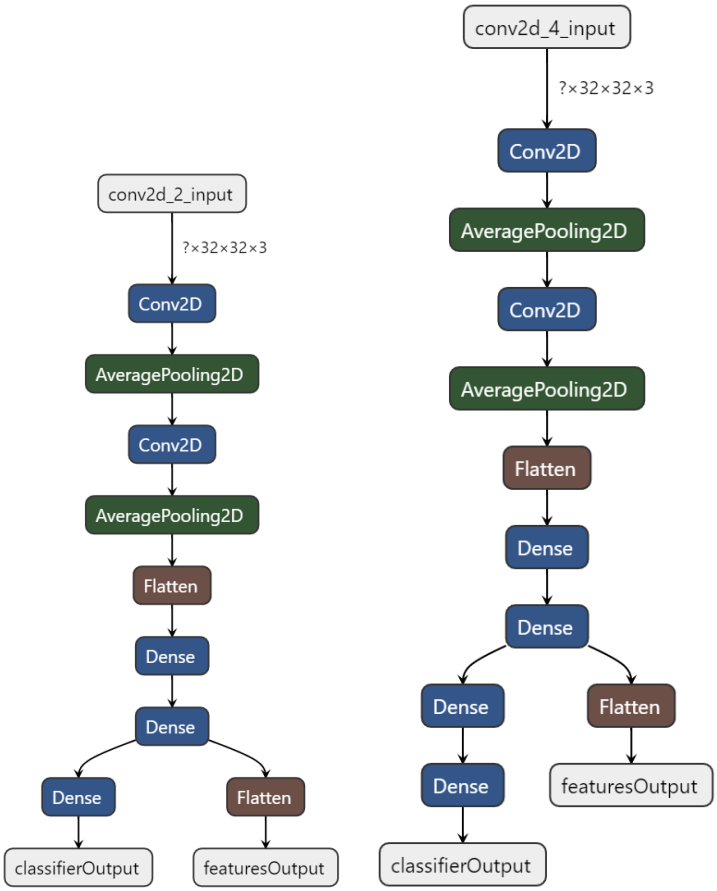
V tejto časti si ukážeme ako sa implementuje rozvetvenie pomocou knižnice Kerasu. Pre trénovanie modelu potrebujeme definovať stratové funkcie, vyhodnocovacie funkcie a váhy pre stratové funkcie. Definícia stratových funkcii je na riadku č. 1, kde použijeme funkcie z predošlých kapitol č. 3.5.1 a č. 3.5.2. Podobne sa definujú vyhodnocovacie funkcie na riadku č. 2. Na riadku č. 3 je definícia váh. V našom prípade do trénovacieho procesu vstupujú stratové funkcie s rovnakou váhou.

Riadky č. 4 až 8 už boli vysvetlené v kapitole č. 3.5. Na riadku č. 10 sa volá funkcia compile, ktorá definuje pre model stratové funkcie, optimalizačnú metódu a vyhodnocovacie funkcie.

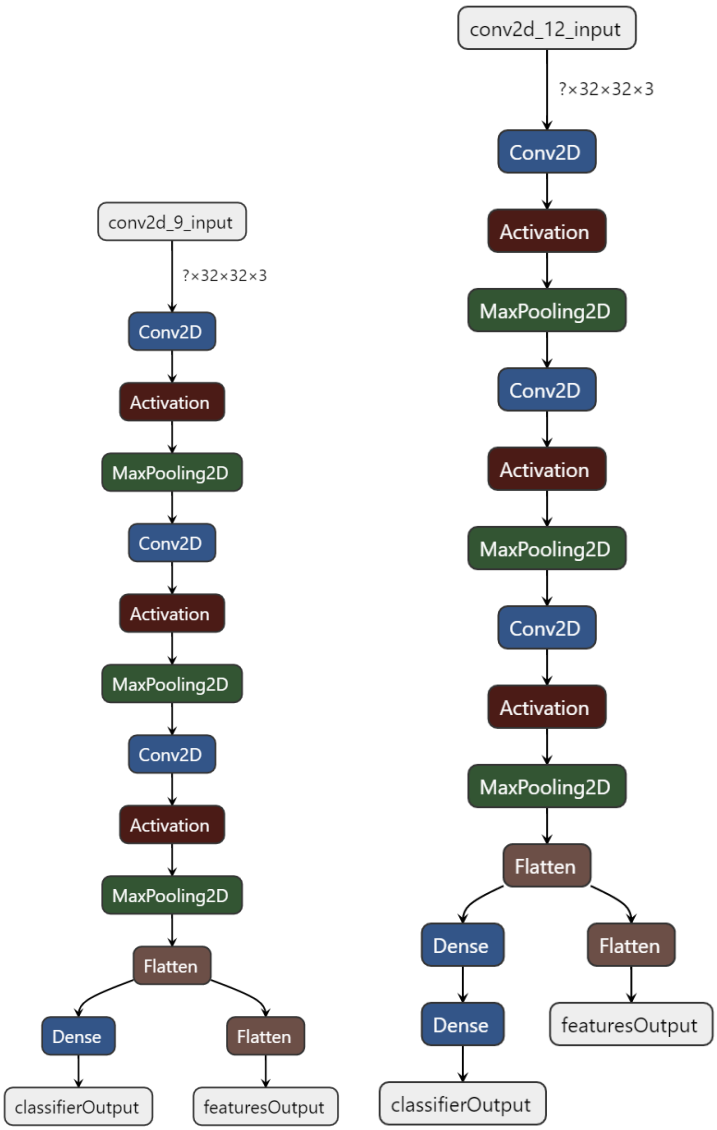
```
losses = {'classifierOutput': classifierLossFunction, '
    featuresOutput': featuresLossFunction}
metrics = {'classifierOutput': classifierAccuracy, 'featuresOutput'
    : featuresAccuracy}
weights = {'classifierOutput':1.0, 'featuresOutput':1.0}

model = getDefaultEnhancedModel(input_shape)
classifierOutput = Dense(num_classes, activation='softmax', name='
    classifierOutput')(model.layers[-1].output)
featuresOutput = Flatten(name='featuresOutput')(model.layers[-1].
    output)
model = Model(inputs=model.inputs, outputs=[classifierOutput,
    featuresOutput])

model.compile(loss=losses,  metrics=metrics, optimizer='rmsprop',
    run_eagerly=True, loss_weights=weights)
```

Úryvok 10: Implementácia

3.5.4 Architektúry modelov graficky znázornené



Obr. 8: Rozšírený model na základe Lenet modelu



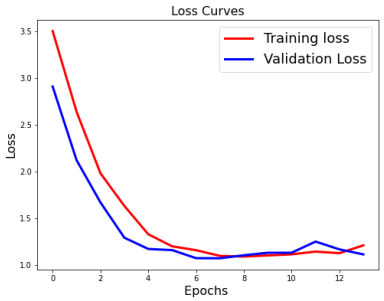
Obr. 9: Rozšírený model na základe vlastného modelu

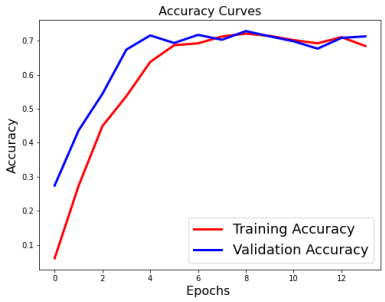
4 Zhodnotenie

Popísať čo sa dá odvodiť z obrázkov, pridať tabulku s percentami + opísať javy + obranyschopnosiť voči adv attack

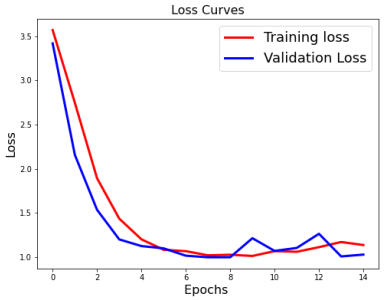
4.1 Výsledky

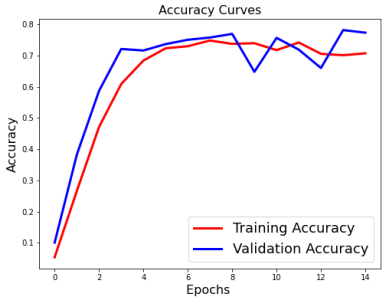
LeNet-5



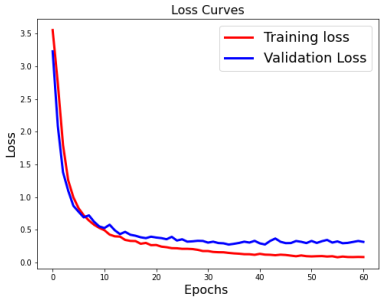


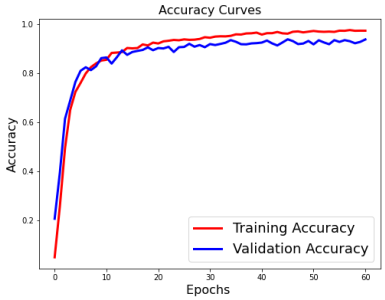
LeNet-5 + Features



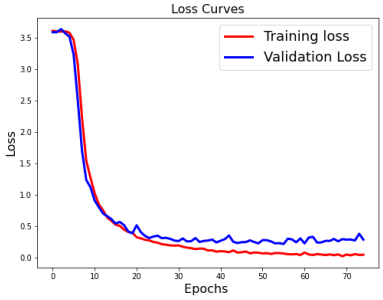


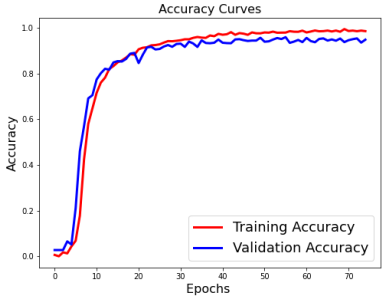
LeNet-5 + Features + Dense



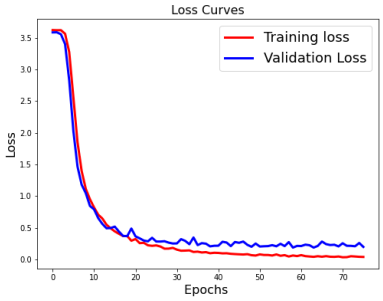


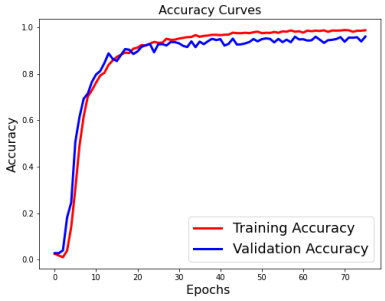
Vlastný model



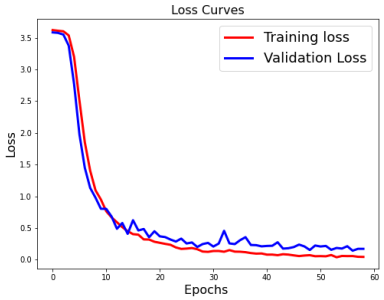


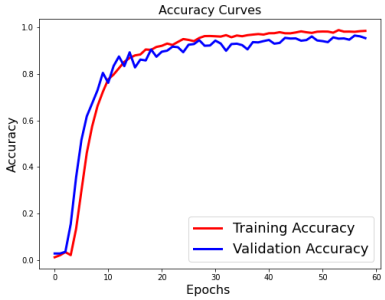
Vlastný model + Features





Vlastný model + Features + Dense





5 Záver

Lorem ipsum

Literatúra

[1] Andrej Lúčny. *Nové možnosti fyzickej interakcie hráča s počítačovou hrou.*. Dostupné na internete 2020 (http://www.agentspace.org/andy/avfx-lucny.pdf).

[2] Andrej Lúčny. *Čítanie textu na pneumatike*. Dostupné na internete 2020 (http://www.agentspace.org/andy/kuz2018-lucny.pdf).

[3] Yann LeCun Leon Bottou Yoshua Bengio and Patrick Haffner *GradientBased Learning Applied to Document Recognition*. Dostupné na internete 2020 (http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf).

[4] Bc. Michal Lukáč *Hlboké neurónové siete pre spracovanie multimédií*. Dostupné na internete 2020 (https://is.muni.cz/th/r1pur/thesis.pdf).

[5] Sunny Chugh *Dump Keras-ImageDataGenerator. Start Using TensorFlow-tf.data*. Dostupné na internete 2020 (https://medium.com/swlh/dump-keras-imagedatagenerator-start-using-tensorflow-tf-data-part-1-a30330bdbca9).

[6] D. H. Hubel and T. N. Wiesel *Receptive fields and functional architecture of monkey striate cortex*. The Journal of Physiology, vol. 195, no. 1, pp. 215–243, 1968.

[7] Edouard Belval *TextRecognitionDataGenerator*. Dostupné na internete 2020 (https://github.com/Belval/TextRecognitionDataGenerator)

[8] Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; Fergus, R. *Intriguing properties of neural networks*. arXiv 2013, arXiv:1312.6199