Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Systémové inženýrství a informatika

Bakalářská práce

Vývoj neuronové sítě pro rozpoznávání objektu v digitálním obrazu

Denis Voinkov

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Denis Voinkov

Systémové inženýrství a informatika
Informatika

Název práce

Vývoj neuronové sítě pro rozpoznávání objektu v digitálním obrazu

Název anglicky

Development of a neural network for object recognition in a digital image

---

Cíle práce

Cílem práce je naprogramovat a naučit neuronovou síť, jejímž úkolem bude rozpoznávat objekty v digitálním obrazu.

Metodika

V první teoretické části práce bude seznámení se s konceptem neuronové sítě a umělé inteligence, principem strojového vidění. Bude popsán přehled nástrojů a zdrojů, které se používají pro učení neuronových sítí, včetně toho, jaký dopad mají neuronové sítě na popkulturu, a také se seznámit s rozšířeným používáním neurových sítí v našem životě. V druhé praktické části práci bude softwarová dokumentace naprogramované neuronové sítě ve vyšším jazyce Python s využitím frameworku Keras, a naučení sítě pomocí datové sady CIFAR-10, vypracovaná podle standardů softwarového inženýrství. Proces učení modelu neuronové sítě bude mít čtyři fáze: 1) Příprava dat, ze kterých se síť bude učit. 2) Rozhodnutí o počtu vrstev použitých v modelu, jaká bude velikost vstupní a výstupní vrstvy, jaké aktivační funkce budou použity. 3) Učení modelu a jeho přizpůsobení datům. Při tréninku modelu je největší pozornost věnována množství času potřebného k tréninku sítě. 4) Vyhodnocení účinnosti modelu na takové testovací sadě, která nebyla použita k učení.

Doporučený rozsah práce

40-80 stran

Klíčová slova

neuronová síť; strojové vidění; rozpoznávání objektu; Keras; Python

---

Doporučené zdroje informací

GÉRON, A. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems.* Beijing ; Boston ; Farnham ; Sevastopol ; Tokyo: O'Reilly, 2019. ISBN 978-1-492-03264-9.

GOODFELLOW, I. – BENGIO, Y. – COURVILLE, A. *Deep learning.* Cambridge, Massachusetts: The MIT Press, 2016. ISBN 978-0-262-03561-3.

CHOLLET, F. (2018) Deep learning with Python. Saint Petersburg, ISBN 978-5-4461-0770-4

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

doc. Ing. Vojtěch Merunka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2021

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 23. 11. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 30. 11. 2022

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Vývoj neuronové sítě pro rozpoznávání objektu v digitálním obrazu" jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.11.22 _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu své bakalářské práce, doc. Ing.Vojtěchu Merunkovy, Ph.D. za vedení bakalářské práce a ochotnou spolupráci. Rád bych také poděkoval Ing. Adéle Hamplové za poskytnutí souboru dat.

Vývoj neuronové sítě pro rozpoznávání objektu v digitálním obrazu

Abstrakt

Cílem této bakalářské práce je vytvořit a natrénovat neuronovou síť pro rozpoznávání digitálních obrazů palmýrské abecedy.

Teoretická část popisuje základní pojmy umělé inteligence a neuronových sítí, typy jejich architektur a principy fungování. Uvádí také, v jakých oblastech a jaké úlohy dnes neuronové sítě pomáhají řešit.

Praktická část se skládá z přípravy dat a jejich rozšíření pro trénování, sestavení architektury neuronové sítě, trénování modelu a analýzy výsledků.

Klíčová slova: neuronová síť; strojové vidění; rozpoznávání objektu; Keras; Python

Development of a neural network for object recognition in a digital image

Abstract

The aim of this bachelor thesis is to build and train a neural network for recognizing digital images of the Palmyrene alphabet.

The theoretical part describes the basic concepts of artificial intelligence and neural networks, types of their architectures and principles of operation. It also shows in which fields and what tasks neural networks help to solve today.

The practical part consists of preparing data and augmenting it for training, building the neural network architecture, training the model, and analysing the results.

Keywords: neural network; computer vision; object recognition; Keras; Python

Obsah

1 Úvod

Umělé neuronové sítě se od svého vzniku používají v mnoha oblastech lidského života, což je zvláště patrné v posledních letech, a to díky rostoucímu počtu specialistů na neuronové sítě a také díky vzniku nástrojů a frameworků, které jsou dostupnější a srozumitelné pro začátečníky a používají se k trénování umělých neuronových sítí. Umělé neuronové sítě se úspěšně používají v široké řadě úloh, jako je rozpoznávání vzorů, prognózování, detekce závislostí, komprese dat, řídicí úlohy a mnoho dalších. Jednou z důležitých oblastí praktické aplikace neuronových sítí jsou rozpoznávací úlohy. V moderním světě se rozpoznávání vzorů stále více používá v každodenním životě lidí. Metody a algoritmy teorie rozpoznávání jsou široce používány v medicíně (diagnostika lékařských snímků), geologii (zkoumání přírodních zdrojů na Zemi), robotice (robotické vidění), astronomii, analýze obrazu, identifikaci lidí, automatickém návrhu atd.

V souvislosti s výše uvedeným je téma bakalářské práce zaměřená na zkoumání a vývoj modelu neuronové sítě, která rozpoznává obrazy. Relevance tohoto programu je způsoben širokým použitím podobných technologií, a v důsledku toho je velmi populární na trhu program pro rozpoznávání obrazu.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je naprogramovat a naučit neuronovou síť, jejímž úkolem bude rozpoznávat objekty v digitálním obrazu.

2.2 Metodika

V první teoretické části práce bude seznámení se s konceptem neuronové sítě a umělé inteligence, principem strojového vidění. Bude popsán přehled nástrojů a zdrojů, které se používají pro učení neuronových sítí, včetně toho, jaký dopad mají neuronové sítě na popkulturu, a také se seznámit s rozšířeným používáním neuronových sítí v našem životě. V druhé praktické části práci bude softwarová dokumentace naprogramované neuronové sítě ve vyšším jazyce Python s využitím frameworku Keras, a naučení sítě pomocí datové sady palmýrské abecedy, vypracovaná podle standardů softwarového inženýrství. Proces učení modelu neuronové sítě bude mít čtyři fáze: 1) Příprava dat, ze kterých se síť bude učit. 2) Rozhodnutí o počtu vrstev použitých v modelu, jaká bude velikost vstupní a výstupní vrstvy, jaké aktivační funkce budou použity. 3) Učení modelu a jeho přizpůsobení datům. Při tréninku modelu je největší pozornost věnována množství času potřebného k tréninku sítě. 4) Vyhodnocení účinnosti modelu na takové testovací sadě, která nebyla použita k učení, a také vytvoření matice záměna její analýza pomocí klasifikačních metrik.

3 Teoretická východiska

3.1 Umělá inteligence

Historie vzniku pojmu Umělá Inteligence se datuje od léta roku 1956, kdy se na Dartmouth College konal seminář, kde se setkávali zájemci o modelování lidské mysli.

Seminář v Dartmouthu trval dva měsíce, během nichž byly schváleny hlavní body nové vědecké oblasti a název navržený Johnem McCarthym "artificial intelligence". Semináře se zúčastnilo 10 lidí, kteří byli spojeni s problematikou výzkumu inteligence, neuronových sítí, teorie her.

Hlavním cílem bylo: simulovat uvažování, inteligenci a tvůrčí procesy pomocí počítačů.

Během semináře byl formulován jeden z hlavních principů tvorby umělé inteligence – změna reakcí na proměnlivé prostředí.

Takovouto metodu učení lze nazvat „pokusem a omylem“, to znamená, že stroj umístěný v určitém prostředí a s přihlédnutím ke kritériím selhání / úspěch lze vycvičit k účelnému chování. [1]

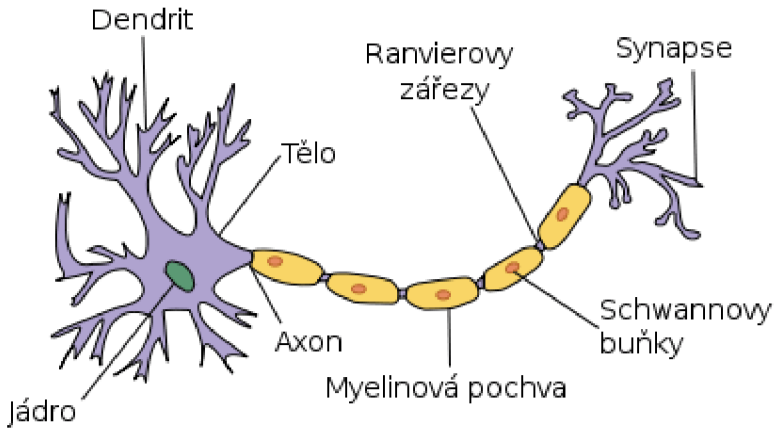
Hlavním problémem je, že vědci v současnosti nejsou schopni přesně určit, jak aktivita neuronů v mozku přispívá k řešení problémů. A přestože se optimistické prognózy nenaplnily a během těchto dvou měsíců se seminář nestal místem žádných velkých objevů, ale byl to právě on, kdo dal start vědeckým objevům v oblasti umělé inteligence.

V roce 1976 Joseph Weizenbaum, v té době profesor počítačových věd na Massachusetts university of technology, napsal knihu „Computer Power and Human Reason“, jejímž tématem bylo rozlišovat mezi lidskou myslí a strojem, že síla počítačových výpočetních systémů nikdy nepřeroste do lidské mysli, protože jsou zásadně odlišné. Tam, kde se člověk řídí opatrností a moudrostí, má stroj pouze algoritmy. [2]

I po více než půl století mají inteligentní systémy spíše úzké oblasti použití. Například programy vytvořené porazit osobu ve videohře, není schopen odpovědět na otázky nebo identifikovat předměty na fotografii.

3.2 Neuronová síť

Biologická neuronová síť je sekvence neuronů spojených synapsemi.



obrázek 1 - struktura biologické neuronové sítě

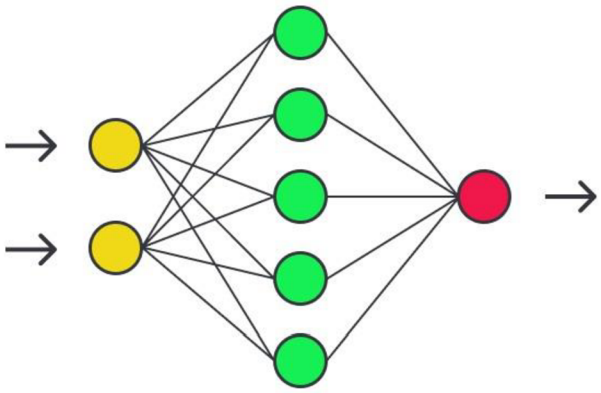
Buňka neuronu je navržena tak, aby přijímala zvenčí, zpracovávala, ukládala, přenášela a vydávala informace ven pomocí elektrických a chemických signálů.

Synapse jsou spojení, přes které výstupní signály některých neuronů vstupují do vstupů jiných, čímž spojují neuronové buňky a tvoří neuronové sítě. Každé takové spojení je charakterizováno svou váhou

Umělé neuronové sítě jsou matematickým modelem, stejně jako jeho softwarová či hardwarová implementace, postavené na principu biologických neuronových sítí. To znamená, že jsou zjednodušeným modelem biologického analogu a nejsme schopni pomocí moderních technologií přesně vytvořit mechanismus lidského mozku. [3]

3.2.1 Neuron a Bias

Strukturální jednotkou neuronové sítě je neuron. Každý neuron přijímá data, zpracovává je a poté je předává dalšímu neuronu. Dělí se na tři hlavní typy: vstup (žlutá), skrytý (zelená), výstup (červená).



obrázek 2 - neuronová síť

Neuronová síť se může skládat z mnoha neuronů, takzvaných „vrstev“.

- První vrstva je vstup, který přijímá informace, může to být např. obraz, zvuk.
- Skrytá vrstva, skládající se z n-počtu skrytých vrstev. Data se dostávají do skrytých vrstev, kde jsou jim přiřazeny určité váhy.
- Na konci je výstupní vrstva, která by měla vydávat odpovědi, například to, co je znázorněno na obrázku.

Bias neuron je typ neuronu používaný ve většině neuronových sítí. Zvláštností tohoto typu neuronu je, že jeho vstup a výstup vždy se rovná 1 a nikdy nemají vstupní synapse. Bias neurony mohou být buď přítomny v neuronové síti, jeden na vrstvu, nebo zcela chybět. Spojení bias neuronů je stejné jako u běžných neuronů – se všemi neurony další úrovně, s výjimkou, že synapse mezi dvěma bias neurony nemůže být. Proto je lze umístit na vstupní vrstvu a všechny skryté vrstvy, ale ne na výstupní, protože prostě nemají s čím vytvářet spojení.

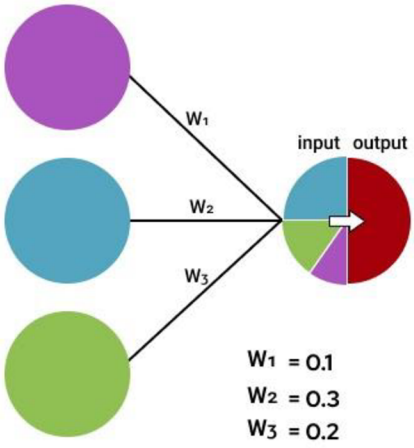
Bias neuron je potřeba k tomu, aby bylo možné získat výstupní výsledek posunutím grafu aktivační funkce.

Také bias neuron pomáhá v případě, kdy všechny vstupní neurony přijmou jako vstup 0 a bez ohledu na to jakou mají váhu, jsou všechny 0 budou převedeny na další vrstvu, ale ne v případě přítomnosti bias neuronu.

Přítomnost nebo nepřítomnost bias neuronů je hyperparametr. [4]

3.2.2 Synapse

Synapse je spojení mezi dvěma neurony a má pouze jeden parametr – hmotnost. Díky hmotnosti se vstupní informace mění, když se přenáší z jednoho neuronu do druhého.



obrázek 3 - interakce vah

Neuron, který má větší váhu, jeho informace bude dominantní v dalším neuronu. Váhy jsou distribuovány náhodně při aktivaci neuronové sítě. [4]

3.2.3 Datová sada, Epocha, Batch, Iterace

Datová sada jsou data, na kterých se neuronová síť trénuje. Běžně se data rozdělují na trénovací množinu, validační množinu a testovací množinu.

Pro efektivní zpracování velkého množství dat je třeba použít parametry (epocha, velikost dávky, iterace), protože často není možné načíst všechna data do zpracování najednou.

Tento problém se řeší tak, že se data rozdělí na menší části, které se postupně načítají, a na konci každého kroku se aktualizují váhy neuronové sítě a přizpůsobují se datům.

Epocha se zvyšuje pokaždé, když neuronová síť projde celou trénovací sadou.

Protože jedna epocha je pro počítač příliš velká, je soubor dat rozdělen do malých dávek (batch).

Iterace jsou počtem dávek potřebných k dokončení jedné epochy.

3.2.4 Hyperparametry

Hyperparametry jsou ručně vybrané hodnoty, a jsou založeny na metodě pokusů a omylů.

Můžete rozlišit několik takových hodnot:

- Rychlost učení
- Počet skrytých vrstev
- Počet neuronů ve vrstvě

Správný výběr hyperparametrů pozitivně ovlivní výsledek výstupu. Nicméně přesný jejich výběr, přichází se zkušenostmi a nejlepším řešením by bylo nastudovat různé příklady neuronových sítí na internetu.

3.3 Aktivační funkce

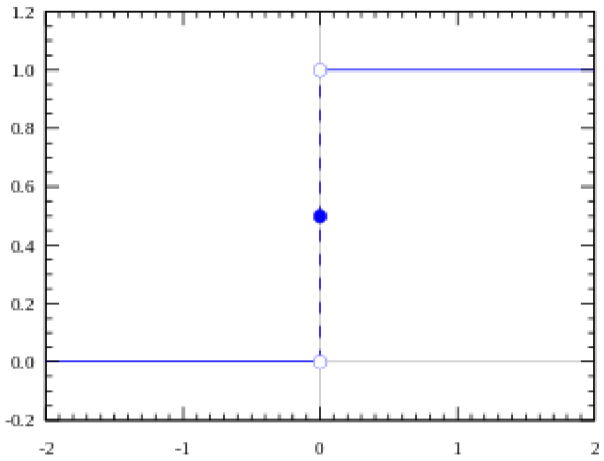
Aktivační funkce je způsob, jak normalizovat vstupy. S velkým počtem vstupů jej provedeme aktivační funkcí a získáme výstup v požadovaném rozsahu.

Je neuron: H = Σ (váha * vstup)

Hodnota H může být libovolná v rozmezí od mínus nekonečna a plus nekonečna. Ve skutečnosti neuron nezná hranici, po které následuje aktivace. Pro tento účel se rozhodli přidat aktivační funkci. Aktivační funkce kontroluje hodnotu H vytvořenou neuronem, aby zjistila, zda by vnější spojení měla považovat tento neuron za aktivovaný, nebo jej lze ignorovat. [5]

3.3.1 Heavisideova funkce

První věc, kterou je třeba rozhodnout, je to, co považovat za aktivační hranici pro aktivační funkci. Pokud je hodnota H větší než nějaká prahová hodnota, považujeme neuron za aktivovaný. Jinak říkáme, že neuron je neaktivní.



obrázek 4 - heavisideova funkce

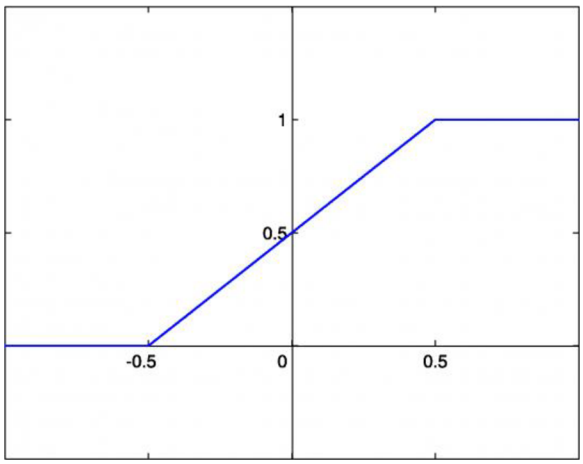
$$H(x) := \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

Funkce přijímá hodnotu 1 (aktivována), když je H> 0 (hranice) a hodnota 0 (není aktivována) jinak.

Je to jednoduchý způsob, ale má své nevýhody. Představte si případ, kdy je potřeba více neuronů pro klasifikaci mnoha tříd: třída1, třída2, třída3 atd. Co by se stalo, kdyby aktivovaných bylo více než 1 neuron? Všechny neurony budou mít z aktivační funkce výstup 1. V tomto případě se objevují otázky, jaká třída by měla být nakonec získána pro daný objekt. [5]

3.3.2 Lineární funkce

Lineární funkce má dva lineární úseky, kde se aktivační funkce rovná minimální a maximální dovolené hodnotě.



obrázek 5 - lineární funkce

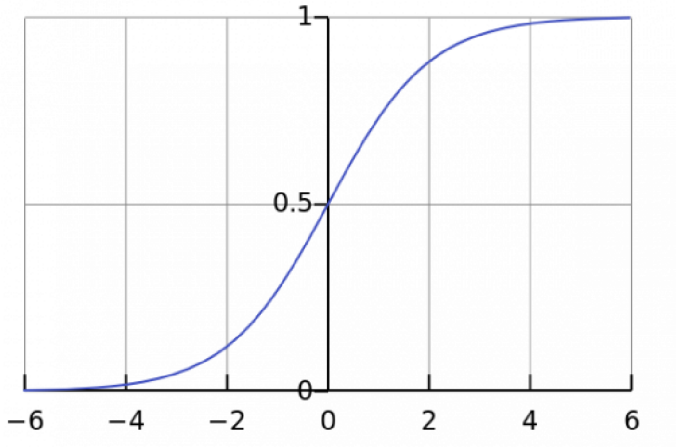
Tato volba aktivační funkce umožňuje získat spektrum hodnot, a to nejen binární odpověď. Je také možné spojit několik neuronů dohromady a pokud je aktivován více neuronů, rozhoduje se na základě maximální hodnoty. Ale i tady jsou problémy.

Nezáleží na tom, kolik vrstev máme. Pokud jsou všechny od přírody lineární, pak poslední aktivační funkce v poslední vrstvě bude pouze lineární funkcí od vstupů na první vrstvě.

To znamená, že dvě vrstvy (nebo n-vrstvy) mohou být nahrazeny jednou vrstvou. Ztratili jsme možnost dělat sestavy z vrstev. [5]

3.3.3 Sigmoidní funkce

Sigmoid je nelineární a kombinace těchto funkcí produkuje také nelineární funkci. Tímto způsobem můžeme spojit vrstvy.



obrázek 6 - sigmoidní funkce

$$f(x) = \frac{1}{1 + e^{-x}}$$

Další výhodou takové funkce je, že je nebinární, takže aktivace je na rozdíl od Heavisideovy funkce analogová. Sigmoid má také hladký gradient.

Sigmoid skutečně vypadá jako vhodná funkce pro klasifikační úkoly. Má za cíl přinést hodnoty na jednu stranu křivky (například na horní x=2 a dolní x=-2). Toto chování nám umožňuje najít jasné hranice při předpovědi.

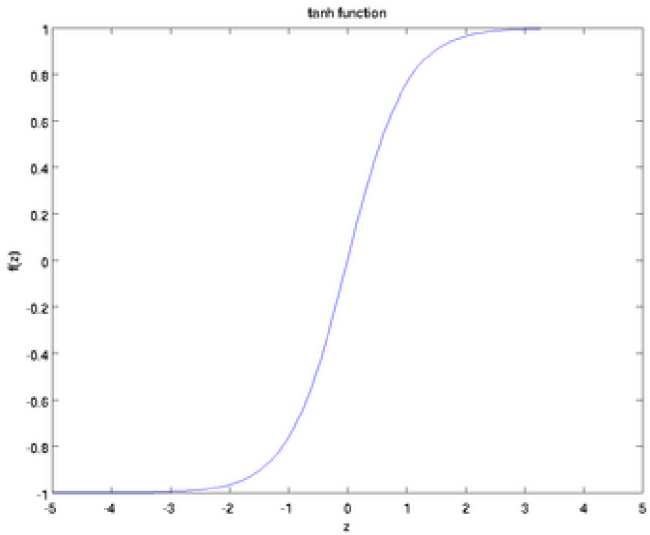
Další výhodou sigmoidy nad lineární funkcí je následující. V prvním případě máme pevný rozsah hodnot funkce [0,1], zatímco lineární funkce se mění v mezích (- inf, + inf). Tato vlastnost sigmoidů je velmi užitečná, protože nevede k chybám v případě velkých aktivačních hodnot.

Dnes je sigmoid jednou z nejčastějších aktivačních funkcí v neuronových sítích. Ale i ona má nedostatky.

Když se blíží konce, sigmoidy mají tendenci reagovat na změny slabě. To znamená, že gradient v takových oblastech má malé hodnoty. A to zase vede k problémům s gradientem zmizení. V takovém případě je hodnota přechodu malá nebo zmizí. Neuronová síť se odmítá dále vzdělávat, nebo to dělá velmi pomalu. [5]

3.3.4 Hyperbolická tangenta

Hyperbolický tangens je velmi podobný sigmoidovi. A opravdu je to upravená sigmoidní funkce. Proto má tato funkce stejné vlastnosti jako u sigmoidů, které byly přezkoumány dříve.



obrázek 7 - hyperbolická tangenta

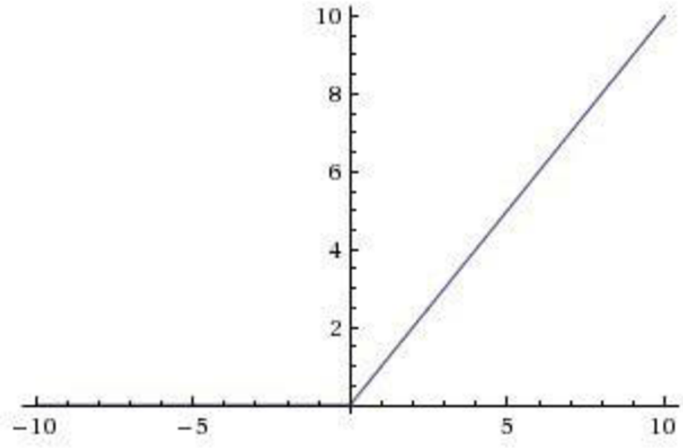
$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

Tato funkce je také nelineární, dobře se hodí pro kombinaci vrstev a rozsah funkčních hodnot je [-1, 1]. Nemá proto smysl se obávat, že aktivační funkce bude přetížena velkými hodnotami.

Rozhodnutí, zda zvolit sigmoid nebo tangens, závisí na požadavcích na amplitudu gradientu. Stejně jako sigmoid má hyperbolický tangens problém se zmizením gradientu. [5]

3.3.5 Rektifikovaná lineární jednotka (ReLu)

ReLu je nelineární funkce a jakákoli jiná funkce může být aproximována kombinací ReLu. Vrátí hodnotu x, pokud je x kladné, jinak 0. Oblast platných hodnot ReLu je [0, inf).



obrázek 8 - ReLu

$$f(x) = x^+ = \max(0, x)$$

ReLu je nelineární funkce a jakákoli jiná funkce může být aproximována kombinací ReLu. Vrátí hodnotu x, pokud je x kladné, jinak 0. Oblast platných hodnot ReLu je [0, inf).

Nejdůležitějším plusem ReLu je jeho matematický vzhled, funkce vyžaduje méně výpočetních zdrojů, což urychluje fyzickou dobu učení neuronové sítě, je to dobré pro vytváření hlubokých neuronových sítí. Také pomocí ReLU můžeme deaktivovat některé neurony, což opět zrychlí naši neuronovou síť.

Ale, jako u každé funkce, u ReLU existují nedostatky, je to způsobeno tím, že záporné vstupní hodnoty budou dávat na výstupu nulový gradient, což negativně ovlivní proces učení neuronové sítě z důvodu nedostatku aktualizace váhových koeficientů. Tento problém je částečně vyřešen modifikací ReLU, v níž při záporných hodnotách vstupních parametrů je přiřazena určitá lineární funkce s malým koeficientem proporcionality, který umožňuje zbavit se nulového gradientu a v procesu učení upravit váhy. [5]

3.3.6 Jak vybrat aktivační funkci?

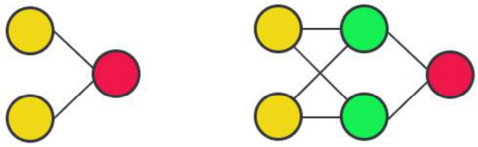
Na tuto otázku neexistuje jednoznačná odpověď. Každá z uvažovaných funkcí má své charakteristické vlastnosti, výhody a nevýhody. Žádná z funkcí není univerzální. Nelze jednoznačně říct, ve kterém případě se má použít ReLu, sigmoid nebo tangens. Když znáte některé vlastnosti funkce, kterou se snažíte aproximovat, zvolte aktivační funkci, která aproximuje požadovanou funkci lépe a vede k rychlejšímu učení.

3.4 Typy architektury neuronových sítí

V současné době existují desítky typů architektur neuronových sítí, rozebereme několik z nich.

3.4.1 Perceptron

Perceptron je prvním modelem neuronové sítě, skládá se ze dvou vrstev, vstupní vrstvy, do které dodáváme data, a výstupní vrstvy, která dává jako odpověď 0 nebo 1, podle toho, zda vstup dosáhl určitého úroveň nebo ne.

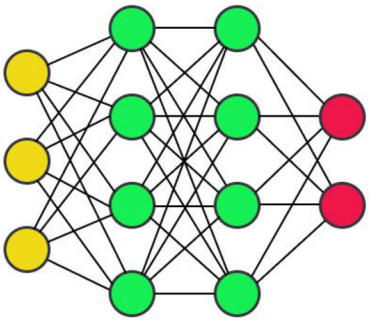


obrázek 9 - perceptron

Taková architektura nemohla pracovat s některými typy logických úloh a bylo potřeba ji vylepšit. Řešením bylo zavést skrytou vrstvu, v podstatě je to několik perceptronů připojeno postupně. [6]

3.4.2 Vícevrstvý perceptron

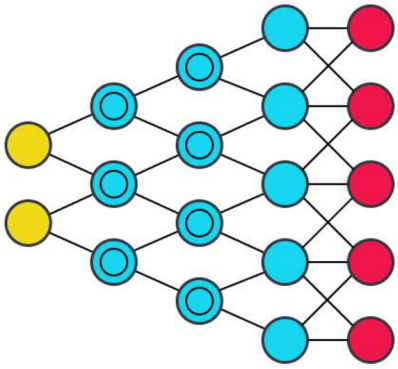
Vícevrstvý perceptron – skládá se ze tří nebo více vrstev. Každý uzel ve vrstvě je připojen ke každému uzlu v další vrstvě, čímž je síť plně propojena. Pokud existují více než dvě skryté vrstvy, považuje se to za hlubokou neuronovou síť (Deep Feed Forward, DFF). [6]



obrázek 10 - vícevrstvý perceptron

3.4.3 Konvoluční neuronová síť

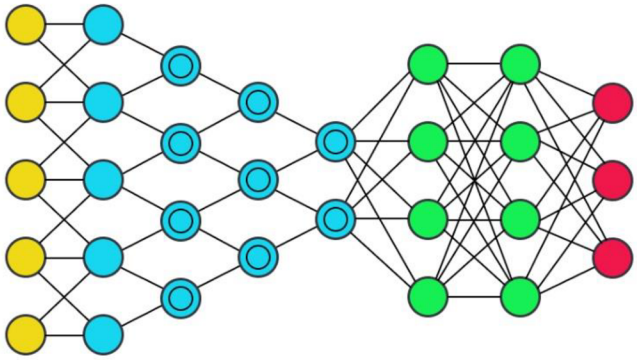
Konvoluční neuronová síť (Convolutional neural network, CNN) – využívá variaci vícevrstvého perceptronu. První vrstvy komprimují informace, pak normální perceptron zpracovává komprimované informace a vydává odpověď. [6]



obrázek 11 - konvoluční neuronová síť

3.4.4 Dekonvoluční neuronová síť

Existují také Dekonvoluční sítě (Deconvolution Network) – ve skutečnosti tyto neuronové sítě provádějí operace v obráceném pořadí než CNN. [6]



obrázek 12 - dekonvoluční neuronová síť

3.5 Princip fungování neuronových sítí

Pojďme si na příkladu rozebrat princip fungování neuronových sítí.

Existuje síť se dvěma vstupními neurony (A1 a A2), jejich odpovídajícími váhami (w1 a w2) a skrytým neuronem (H).

Podáme na vstup:

A1 = 1

A2 = 0

w1 = 0,4

w2 = 0,7

Musíme vypočítat váženou částku (Weighted Sum). Jedná se o součet vstupních signálů vynásobených jejich hmotností.

Používáme vzorec: $H_{input} = (A1*w1) + (A2*w2)$

To znamená vstupní data neuronu $H_{input} = 1 * 0,4 + 0 *0,7 = 0,4$.

Vzhledem k vstupním datům můžeme získat výstupní data dosazením vstupní hodnoty do aktivační funkce.

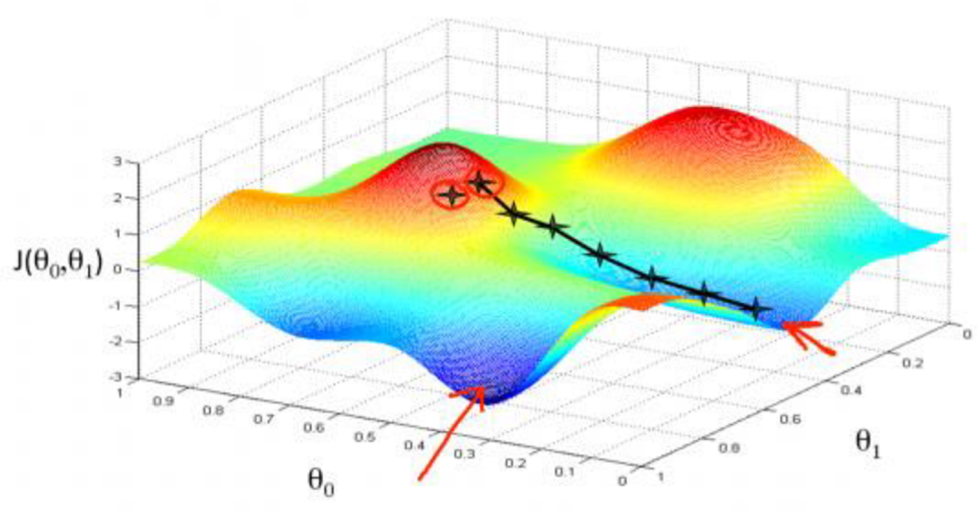
Aktivační funkce je funkce, která bere jako argument vážený součet. Hodnota této funkce je výstupní neuron H_{output}.

Používáme vzorec: $H_{output} = f_{activation}(H_{input})$

A tak budeme opakovat pro všechny vrstvy, dokud se nedostaneme k výstupnímu neuronu. [7]

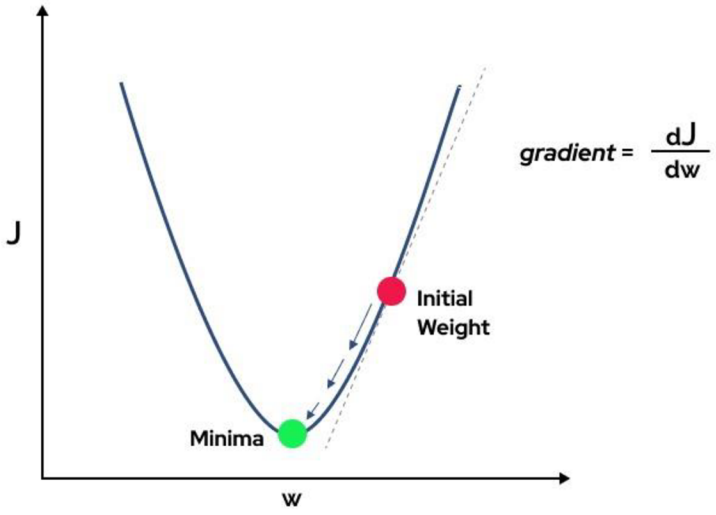
3.5.1 Gradientní sestup

Gradient sestup je metoda nalezení minimální hodnoty funkce ztráty. Minimalizace jakékoli funkce znamená nalezení nejhlubšího údolí v této funkci. Funkce se používá ke kontrole chyby v prognózách modelu strojového učení. Najít minimum znamená dostat co nejmenší chybu nebo zvýšit přesnost modelu. Zvyšujeme přesnost tím, že procházíme trénovací datovou sadu při nastavení parametrů našeho modelu (váhy a bias).



obrázek 13 - graf ztrátové funkce

Aby najít nejnižší chybu (nejhlubší údolí) ve funkci, musíme upravit parametry modelu. V tom pomůže matematická analýza. Prostřednictvím analýzy víme, že sklon grafu funkce je derivací funkce vzhledem k proměnné. Tento sklon vždy ukazuje k nejbližší prohlubni.



obrázek 14 - gradientní sestup

Na obrázku 3.14 vidíme graf ztrátové funkce s jednou váhou. Nyní, když spočítáme sklon (označíme to ∂J/ ∂w) ztrátové funkce vzhledem k jedné váze, dostaneme směr, kterým se musíme pohybovat, abychom dosáhli lokálních minim.

Když procházíme všechna tréninková data, neustále přidáváme hodnoty ∂J/ ∂w pro každou váhu. Protože ztráta závisí na příkladu tréninku, ∂J/ ∂w se také neustále mění. Shromážděné hodnoty pak vydělíme počtem tréninkových příkladů, abychom získali průměr. Tento průměr pak použijeme k úpravě každé hmotnosti.

Kromě ztrátové funkce vyžaduje sestup gradientu také gradient, který je ∂J/ ∂w (derivace ztrátové funkce s ohledem na jednu váhu, provedená pro všechny váhy). ∂J/ ∂w závisí na volbě ztrátové funkce. Nejběžnější ztrátovou funkcí je střední kvadratická chyba.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Derivace této funkce s ohledem na jakoukoli váhu (tento vzorec ukazuje výpočet gradientu pro lineární regresi):

$$\frac{\partial}{\partial\theta_j}\mathrm{MSE}\,(\theta) = \frac{2}{m}\sum_{i=1}^{m}(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)})\,x_j^{(i)}$$

Problém u většiny modelů nastává s koeficientem rychlosti učení. Faktor rychlosti učení si lze představit jako „krok správným směrem“, kde směr vychází z ∂J/ ∂w.

$$\text{Repeat until convergence } \{$$

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$$\}$$

alpha – je koeficient rychlosti učení, j – leží v rozmezí od 0 do množství váhy a Théta-j je váha j ve vektoru vah.

Vypočítáme ∂J / Théta-j (gradient hmotnosti Théta-j) a pak krok velikosti alpha v tomto směru. To znamená, že klesáme po gradientu.

Je důležité nastavit rychlost učení na vhodnou hodnotu, která pomůže gradientu sestupu dosáhnout lokálních minim. Proto je lepší nevolit příliš vysoké nebo nízké hodnoty. Pokud je velikost kroku příliš velká, překonáme minimum: to znamená, že mineme minimum. Pokud je krok příliš malý, používáme velmi mnoho iterací, abychom se dostali na minimum. [8]

3.6 Metody učení, Chyby, Matice záměn

Existují dvě hlavní metody trénování neuronové sítě – je to s učitelem (supervised) a bez učitele (unsupervised).

3.6.1 Supervised a Backpropogation

Supervised – je metoda učení, kde budeme v roli učitele, a neuronová síť je student. Dáváme vstupní data a hlásíme požadovaný výsledek, takže neuronová síť pochopí, o co se musí v procesu učení snažit.

Zabývá se takovou problematikou jak:

- Regrese – dělat předpovědi na základě dat.
- Klasifikace – Na základě dat předpovězte jednu z uvedených odpovědí.

Algoritmus zpětného šíření chyby (Backpropogation) — populární algoritmus učení vícevrstvých perceptronů. Odkazuje se na metody výuky s učitelem, takže vyžaduje, aby učební příklady měly cílové hodnoty. Je také jedním z nejznámějších algoritmů strojového učení.

Pro posouzení chyby konkrétního příkladu tréninku bereme to, co se ukáže na výstupní vrstvě v síti a to, co podle nás síť měla vydat, a skládáme čtverec rozdílu mezi složkami. Tím, že to zopakujeme pro všechna data ze vzorku a zprůměrujeme výsledek, dostaneme celkovou chybu celé sítě. Zajímáme se o negativní gradient funkce chyby, která informuje o tom, jak je třeba změnit váhy a posunout jejich spojení, aby bylo dosaženo co nejlepšího výsledku. Backpropogation je algoritmus pro výpočet tohoto gradientu.

Jádrem algoritmu je použití výstupní chyby neuronové sítě pro výpočet hodnot korekce hmotnosti neuronů ve skrytých vrstvách.

$$E = \frac{1}{2}\sum_{i=1}^{k}(y - y')^2$$

k — počet výstupních neuronů sítě, y — cílová hodnota, y′ — skutečná výstupní hodnota.

Při každé iteraci probíhají dva průchody sítí – vpřed a vzad. Po přímce se vstupní vektor šíří ze síťových vstupů na její výstupy a tvoří nějaký výstupní vektor odpovídající aktuálnímu (skutečnému) stavu vah. Potom se vypočítá chyba neuronové sítě jako rozdíl mezi skutečnými a cílovými hodnotami. Na opačném průchodu se tato chyba rozšiřuje z výstupu sítě na její vstupy a provádí se korekce váhy neuronů podle pravidla:

$$\Delta w_{j,i}(n) = -\eta\frac{\partial E_{av}}{\partial w_{ij}}$$

$w_{j,i}$ — je váha *i* spojení *j* neuronu, η — parametr rychlosti učení, která je umožňuje dále řídit hodnotou kroku korekce $\Delta w_{j,i}$ s cílem více přesné nastavení na minimum chyby a volí se experimentálně v procesu učení (se mění v intervalu od 0 do 1).

Výstupní součet *j* neuronu je:

$$S_j = \sum_{i-1}^{n} w_{ij}x$$

Můžeme ukázat, že:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial S_j}\frac{\partial S_j}{\partial w_{ij}} = x_i\frac{\partial E}{\partial S_j}$$

Z posledního výrazu vyplývá, že diferenciál ∂S_j aktivační funkce neuronů v síti f(S) musí existovat a nesmí být v žádném bodě roven nule, tj. aktivační funkce musí být diferencovatelná na celé číselné ose. Proto se pro Backpropogation používají sigmoidní aktivační funkce, například hyperbolický tangens.

Algoritmus tedy využívá tzv. stochastický gradientní sestup, „pohybující se“ v multidimenzionálním váhovém prostoru ve směru anti-gradientu, aby bylo dosaženo minima chybové funkce.

V praxi trénink nepokračuje do jemného nastavení sítě na minimum chybových funkcí, ale pouze do dosažení poměrně přesného přiblížení. To umožní na jedné straně snížit počet iterací učení a na druhé straně se vyhne přeškolování sítě. [9][10]

3.6.2 Unsupervised

Unsupervised je vzácnější druh učení. Zde se neptáme neuronové sítě na požadovanou odpověď, neuronová síť má pouze vstupní dataset.

Používají tento druh učení především pro clustering, tj. neuronová síť se nezávisle snaží naučit seskupovat data, která jsou si podobná nebo jsou spojena různými proměnnými.

Non clustering, někdy používají termín „koktejlová party“ – k nalezení struktury v chaotickém prostředí, příkladem je definice konkrétního hlasu v davu lidí.

3.6.3 Chyby underfitting a overfitting

Chyba je procentní hodnota, která ukazuje rozdíl mezi požadovanou odpovědí a výsledným výsledkem. Chyba se počítá každou epochu a měla by klesat. Pokud se tak nestane, znamená to, že dochází k chybnému výpočtu v tréninku. Řešením může být zadání hyperparametrů, ale pokud to nepomůže, pak je problém v architektuře neuronové sítě.

Nedoučení (underfitting) je ukončení trénovacího procesu neuronové sítě předtím, než dosáhne stavu, který poskytne dostatečně malou chybu na trénovací datové sadě.

K nedoučení může dojít z různých důvodů. Uživatel může například nastavit neúspěšné podmínky ukončení tréninku (počet iterací, hodnota tréninkové chyby atd.). Navíc je možné, že konfigurace modelu, která neodpovídá řešenému problému, nebo neúspěšná trénovací sada, prostě nedovolí algoritmu přiblížit se optimu.

Možné řešení při underfitting:

- Přidání nových parametrů modelu
- Použijte k popisu modelu, funkcí s vyšším stupněm

- Snížení regularizačního koeficientu

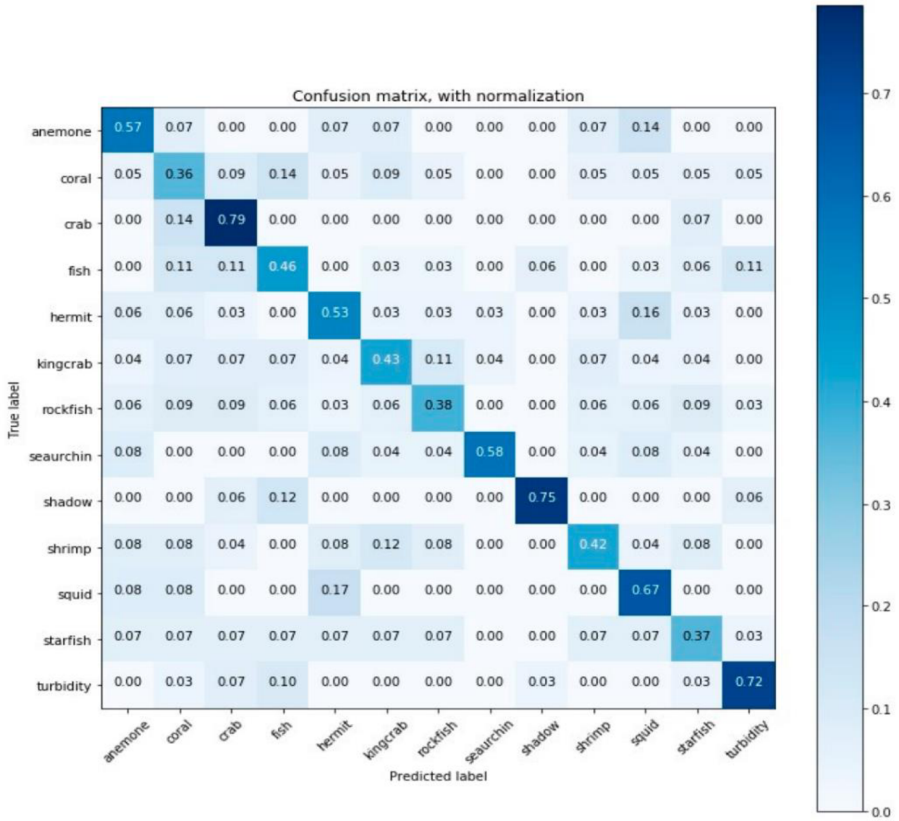
Přeučení (overfitting) – je případ, kdy je neuronová síť přesycena daty, v důsledku čehož "neuč se" na datech, ale "pamatuj" je. Podáním nových vstupních dat narazíme na chybu, která ovlivní výsledek. Například dáme datovou sadu s obrázkem černé kočky a řekneme, že je to kočka, a když neuronová síť uvidí bílou kočku, nebude schopna určit, že je to kočka, nebo klasifikovat černého psa jako kočku.

Existuje několik způsobů, jak vyřešit přeučení. Zde jsou některé z nich:

- Netrénovat neuronovou síť dlouho na stejných datech nebo velmi podobných
- Snížit počet parametrů
- Samostatně zobrazit všechny parametry a nechat jen důležité
- Regularizace – nechat všechny parametry, ale snížit jejich hmotnost nebo velikost [11]

3.6.4 Matice záměn a Klasifikační metriky

Matice záměn (matice chyb) je tabulka, která umožňuje vizualizovat účinnost klasifikačního algoritmu porovnáním předpovězené hodnoty cílové proměnné s její skutečnou hodnotou.



obrázek 15 - příklad matice chyb

Případy na diagonále matice záměn jsou klasifikovány správně, případy mimo diagonálu jsou chyby.

Na matice záměn stojí za to věnovat pozornost nejprve diagonálním hodnotám, které jsou zvýrazněny tmavší barvou. Tato čísla ukazují, v kolika případech se předpověď modelu a skutečná hodnota shodovaly, tj. shoda mezi tím, jak model identifikoval data, a tím, co jsou data ve skutečnosti.

Všechna čísla mimo diagonálu jsou chybné předpovědi modelu, tato čísla ukazují, kolikrát se model mýlil a která data zaměnil.

Z matice chyb pak lze vypočítat několik metrik souvisejících s různými potřebami. V této práci jsou vypočteny následující metriky: Precision, recall и F-score.

Precision (přesnost) – lze interpretovat jako podíl položek, které klasifikátor označí za pozitivní a které jsou skutečně pozitivní.

Recall (úplnost) - ukazuje, jaký podíl objektů pozitivní třídy ze všech objektů pozitivní třídy algoritmus nalezl.

Čím vyšší Precision a Recall, tím lépe. V reálném životě však nelze dosáhnout maximální přesnosti a úplnosti současně a je třeba hledat nějakou rovnováhu.

F-score – je harmonický průměr mezi dvěma hlavními metrikami Precision a Recall. F-skóre redukuje tyto metriky na jediné číslo, které dosahuje maxima při maximální úplnosti a přesnosti a blíží se nule, pokud je jeden z argumentů blízký nule. Na základě tohoto hodnocení bude mnohem snazší rozhodnout, zda jsou změny algoritmu k lepšímu, či nikoli. [12]

3.7 Rozpoznávání obrazu

Teorie rozpoznávání obrazu je obor informatiky, který se zabývá vývojem identifikaci a klasifikaci konkrétních objektů, procesů, situací z celkové hmotnosti na obrázku.

V naší době informačního přetížení, kdy člověk nemůže vyrovnat se s tímto hustý tok informací, je problém rozpoznávání obrazů získává stále větší význam, a stále více přitahuje pozornost různých průmyslových odvětví.

Existují tři hlavní metody rozpoznávání grafických obrazů:

- Iterační metoda, když procházíte pohled na objekt pod různými úhly, měřítky, posuny. U písmen iterujte písmo, vlastnosti písma atd.
- Hledat obrys objektů a prozkoumat jeho vlastnosti, jako je přítomnost úhlů, spojitost atd.

- Použití umělé neuronové sítě. Tato metoda bude vyžadovat velkou sadu příkladů úloh rozpoznávání se správnou odpovědí nebo vytvoření speciální struktury neuronové sítě, která zohledňuje specifika úlohy.

S rozpoznáváním obrazů pracují i další podobné oblasti, jako je počítačové vidění, zpracování obrazu, a strojové vidění. [13][14]

3.7.1 Počítačové vidění

Počítačové vidění je soubor technik pro výuku strojů k získávání informací z obrázku nebo videa. Aby počítač našel na obrázcích určité předměty, musí se to naučit. K tomu je vytvořen trénovací vzorek, z nichž část obsahuje hledaný objekt a druhá část naopak neobsahuje. Dále přijde strojové učení, počítač analyzuje obraz ze vzorku, určuje, jaké znaky a jejich kombinace ukazují na přítomnost hledaných objektů, a vypočítá jejich význam. Po dokončení školení, počítačové vidění lze použít v praxi. Používá se v oblastech, jako jsou: video dohled, systémy modelování objektů a prostředí (topografie), systémy interakce stroje a člověka, rozšířená realita atd. [14][15]

3.7.2 Zpracovávání obrazu

Zpracování obrazu – nebo analýza obrazu, se zaměřuje na práci s dvojrozměrnými obrázky.

Používá se jak pro příjem obrazu na výstupu, například televizního vysílání, a pro získání dalších informací, jako je definice textu, identifikace osoby atp. [13]

3.7.3 Strojové vidění

Strojové vidění je aplikace počítačového vidění pro průmysl a výrobu a od počítačového vidění se liší tím, že oblast zájmu a aplikace je zaměřena na vstupně-výstupní zařízení a počítačové sítě, které jsou určeny k řízení výrobních zařízení. Jednou z nejběžnějších aplikací je kontrola průmyslového zboží, jako jsou automobily, léky, potraviny. Lidé pak kontrolují část zboží a vyvozují závěry o kvalitě kontroly. [15]

3.7.4 Princip rozpoznávání obrazu

Pro počítač je obraz sadou pixelů, z nichž každá má svou vlastní hodnotu jasu nebo barvy. Aby stroj získal představu o obsahu obrázku, je zpracován pomocí speciálních algoritmů. Nejprve jsou identifikována potenciálně významná místa, což lze provést několika způsoby.

Jednou z takových metod Gaussian Blur je dolnopropustný filtr, který vyhlazuje nerovnoměrné hodnoty pixelů v obrázku ořízнutím nejvyšších hodnot. Na obrázek se používá několikrát pomocí různých poloměrů rozostření, pak se výsledky porovnávají mezi sebou, a to umožní identifikovat nejkontrastnější momenty, světlé body a zalomení čar. Jakmile jsou nalezena významná místa, počítač je popisuje v číslech.

Záznam fragmentu obrázku v číselné podobě se nazývá deskriptor. S jeho pomocí můžete přesně porovnávat fragmenty obrazu bez použití samotných fragmentů.

Pro urychlení výpočtu počítač provádí clustering, tedy rozdělení deskriptorů do skupin.

Do stejného clusteru spadají podobné deskriptory z různých obrázků.

Po clusteringu se důležité stává pouze číslo clusteru s deskriptorem, který je nejvíce podobný datům.

Přechod od deskriptoru k číslu clusteru se nazývá kvantizace (Quantization) a samotné číslo clusteru je kvantované číslo. Kvantizace výrazně snižuje množství dat, která je třeba zpracovat v počítači.

Na základě kvantování deskriptory, počítač může porovnat obraz a rozpoznat objekty na nich. Stroj porovnává sady kvantovaných deskriptorů z různých obrázků a vyvozuje, jak jsou oni nebo jejich jednotlivé fragmenty podobné. [13][14][16]

3.8 Programy pro vytváření a učení neuronových sítí

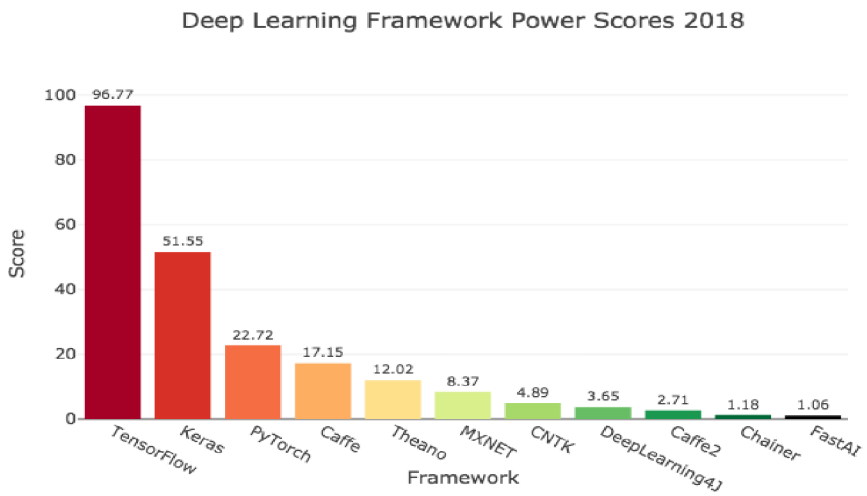
Neuronové sítě jsou vytvářeny a trénovány především v Pythonu, ale často schopnosti jazyka nestačí na efektivní implementaci. Pro práci s neuronovými sítěmi by tedy byla dobrou pomůckou znalost Matlabu, OpenCL nebo C++ pro psaní na nižší úrovni. Obecně platí, že neuronovou síť můžete napsat v jakémkoli Turingově kompletním programovacím jazyce. Ale popularita určitých jazyků pro tento úkol je způsobena skutečností, že pro určitý jazyk bylo napsáno více pomocných knihoven, což snižuje množství práce. Takže v Pythonu se takový vývoj dostal tak daleko, že neuronové sítě mohou být napsány v několika řádcích kódu.

3.8.1 Framework a knihovny

Framework je sada nástrojů a standardních implementací, které umožňují rychlejší vývoj jakéhokoli softwarového produktu. Termín "framework" je poměrně vágní a může znamenat mnoho různých věcí v závislosti na kontextu. Frameworkem může být cokoli, co

se používá při vývoji aplikací: knihovna, soubor mnoha knihoven, soubor skriptů, nebo jakýkoli software potřebný k vytvoření aplikace. [17]

Nejpopulárnější frameworky pro práci s hlubokým učením:



obrázek 16 - nejpopulárnější frameworky roku 2018

3.8.2 TensorFlow

TensorFlow je open source knihovna vytvořená pro Python týmem Google Brain. Zpracovává mnoho různých algoritmů a modelů, což umožňuje uživateli realizovat hluboké neuronové sítě pro použití v takových úkolech, jako je rozpoznávání a klasifikace obrazu, a zpracování přirozeného jazyka.

TensorFlow funguje tak, že implementuje řadu uzlů zpracování, z nichž každý představuje matematickou operaci, a celá řada uzlů se nazývá „graf“. Tato knihovna pracuje se statickým výpočetním grafem. To znamená, že nejprve definujeme graf, pak spustíme výpočty a v případě potřeby provedeme změny v architektuře, znovu vycvičíme model. Tento přístup je zvolen v zájmu efektivity, ale mnoho moderních nástrojů neuronových sítí je schopno zohlednit upřesnění v procesu učení bez významné ztráty rychlosti učení. [18][19]

3.8.3 PyTorch

PyTorch – pracuje na dynamicky aktualizovaném grafu. To znamená, že vám umožňuje provádět změny v architektuře v procesu. PyTorch může používat standardní debuggery jako pdb nebo PyCharm. Proces trénování neuronové sítě je jednoduchý a jasný. Ale na rozdíl od TensorFlow je popsaný framework hlubokého učení mnohem méně

flexibilní při podpoře různých platforem. PyTorch také nemá nativní nástroje pro vizualizaci dat. [18]

3.8.4 Keras

Keras je doplněk TensorFlow, CNTK a Theano. Lze jeho však považovat za framework, protože se používá k vytváření kompletních řešení, jako je Mask R-CNN a face_classification. Také pokud jde o řešení úloh rozpoznávání obrazů a strojového překladu, Keras je vždy považován za alternativu k TensorFlow a PyTorch, protože se snadno používá, snadno se učí a má dobrou dokumentaci. Je vhodný pro malé projekty, protože je těžké na něm vytvořit něco velkého, Keras jasně prohraje TensorFlow ve výkonu neuronových sítí. [18][19]

3.8.5 Caffe

Caffe je systém hlubokého učení zesílený C, C++, Pythonem a MATLABEM. Mezi ostatními frameworky vyniká svou rychlostí a mobilitou, stejně jako svou užitečností při zobrazování konvolučních neuronových systémů. Použití knihovny C++ poskytuje přístup k systémům hlubokého síťového archivu Caffe Model Zoo. [19]

3.8.6 MXNet

MXNet je systém hlubokého učení. Podporuje Python, R, C++ a Julia. Framework umožňuje klientovi kódovat v různých programovacích dialektech. Backend je napsán v C++ a CUDA (hardwarová a softwarová paralelní výpočetní architektura), takže MXNet může škálovat a pracovat s více GPU. [19]

3.8.7 Microsoft Cognitive Toolkit

Microsoft Cognitive Toolkit (CNTK) je open source sada nástrojů pro hluboké učení. Framework popisuje neuronové sítě jako sérii výpočetních kroků prostřednictvím orientovaného grafu. Poskytuje uživateli možnost snadno implementovat a kombinovat oblíbené typy modelů, jako jsou dopředné DNN, konvoluční neuronové sítě a rekurentní neuronové sítě. [19]

3.9 Neuronové sítě v našem životě

Stroje jsou v dnešní době schopné řešit stále více procesů, za které lidé dříve odpovídali. Navíc je kvalitnější a v mnoha případech levnější. Neuronové sítě jsou již nyní

pevně integrovány do našeho každodenního života. Vyhledávací algoritmy od společnosti Google a Apple jsou postaveny na neuronových sítích. Online překladatelé využívají při své práci neuronovou síť. Služby doporučení produktů, které vidíme na mnoha webových stránkách, jsou také založeny na neuronových sítích.

Popularita neuronových sítí zvláště rychle roste po roce 2010. Až do roku 2010 prostě neexistovala žádná dostatečně velká databáze, která by kvalitativně trénovala neuronové sítě k řešení určitých úkolů, zejména souvisejících s rozpoznáváním a klasifikací obrazu. Neuronové sítě proto často chybovaly: pletly si kočku se psem, nebo ještě hůř, obrázek zdravého orgánu s obrázkem orgánu postiženého nádorem.

V roce 2010 se ale objevila databáze ImageNet obsahující 14 milionů obrázků ve 22 000 kategoriích. ImageNet byl mnohokrát větší než existující databáze obrázků a byl přístupný každému specialistovi. S takovými objemy dat se neuronové sítě mohly naučit dělat téměř bezchybná rozhodnutí. [20]

Dále jsou uvedeny některé příklady úspěšného použití neuronových sítí v různých oblastech lidského života.

3.9.1 Zdravotnictví

Největší šíření neuronových sítí v medicíně se dostalo právě v oblasti práce s obrázky. Pracovní postupy lékařských organizací jsou neoddělitelně spojeny se sběrem, zpracováním a analýzou různých medicínských obrazových dat: x-ray, počítačová tomografie a tak dále.

Prvním lékem vytvořeným pomocí neuronové sítě a zařazeným do fáze klinických zkoušek byl lék DSP-1181. Byl vyvinut společností Exscientia ve spolupráci s japonskou farmaceutickou společností a je určen pro léčbu pacientů s obsedantně-kompulzivní poruchou. Výzkumníkům obvykle trvá asi pět let, než takové léky vyvinou. Umělá inteligence si s tímto úkolem poradila za jeden rok.

Analýza DNA je perspektivní a aktivně se rozvíjející oblastí aplikace neuronových sítí. Například nástroj vyvinutý Michiganskou státní univerzitou provádí genetický výzkum a umožňuje lidskému genomu určit jeho výšku s přesností na tři centimetry, předpovídat vývoj tak závažných onemocnění, jako je rakovina, infarkt a identifikovat mutace, které ovlivňují hustotu kostní tkáně, a dokonce i předpovědět úroveň vzdělání, kterého může dosáhnout člověk.

Program Voice2Med šetří čas lékaře vyplňovat různé druhy dokumentace. Díky hlasovému vyplňování zdravotní dokumentace. Služba obsahuje specializované slovníky, které umožňují správné rozpoznání lékařských termínů. [21]

3.9.2 Dopravní prostředek

Bezpilotní automobily – koncept, na kterém pracuje většina velkých korporací, stejně jako technologické společnosti (Google, Uber a další) a startupů, které při své práci spoléhají na neuronové sítě. Neuronové sítě jsou zodpovědné za rozpoznání okolních objektů, ať už je to jiné auto, chodec nebo jiná překážka. Potenciál umělé inteligence v této oblasti není omezen na autopilota. Auta, integrovaný do internetových věcí, bude shromažďovat informace o preferencích cestujících a automaticky regulovat teplotu v kabině, hlasitost rádia, polohy sedadel a další parametry.

3.9.3 Zemědělství

Inženýři společnosti Microsoft společně s vědci z ICRISAT používají umělou inteligenci k určení optimálního času výsevu v Indii. Aplikace, která používá Microsoft Cortana Intelligence Suite, také sleduje stav půdy a vybírá potřebná hnojiva. Původně se na programu podílelo jen 175 zemědělců ze sedmi vesnic. Začali s výsevem až po příslušném SMS oznámení. V důsledku toho sklízeli o 30-40 % více než obvykle. [24]

3.9.4 Bezpečnost

Vývojový tým z University of Technology Sydney představil drony pro hlídkování pláží. Hlavním úkolem dronů bude hledání žraloků v pobřežních vodách a varování lidí na plážích. Analýzu video dat provádějí neuronové sítě, což výrazně ovlivnilo výsledky: vývojáři tvrdí, že pravděpodobnost detekce a identifikace žraloků je až 90 %, zatímco operátor sledující video z dronů úspěšně rozpozná žraloků pouze v 20-30 % případů. [22]

Analýza provedená společností Deep Instinct ukázala, že nové verze virových programů se prakticky neliší od předchozích, změna je od 2 % do 10 %. Self-learning model vyvinutý Deep Instinct, na základě těchto informací je schopen s vysokou přesností určit infikované soubory. [23]

3.9.5 Umění a zábava

Dříve se mělo za to, že strojové učení nikdy nemůže nahradit člověka v umění. Ale současné neuronové sítě mohou kreslit obrazy, skládat hudbu, restaurovat filmy a psát články.

Následuje několik příkladů použití neuronových sítí v umění a zábavě.

Deepfake – je speciální technologie, která je pomocí neuronových sítí a dat schopna vytvářet nové obrázky založené na jiných. S jeho pomocí můžete změnit stávající tváře na úplně jiné.

DALL-E – tuto službu vytvořila nekomerční společnost OpenAI, založená na algoritmu gpt3. Služba vytvoří obrázek na základě anglických popisných slov. Při jejich vytváření bere neuronová síť v úvahu 12 miliard parametrů, mezi které patří barva, výška, umístění a názvy objektů. Díky tomu je DALL-E schopen vytvářet realistické fotografie neexistujících objektů. [25]

Generative Pre-trained Transformer 3 (gpt3) – jeden z nejdokonalejších programů, který chápe jazyky, stejně je vyvinut společností OpenAI. Program je schopen na základě prvních napsaných vět, dopsat celý článek nebo malý příběh. Je schopen napodobit partnera v textovém chatu na sociální síti, přičemž odborníci potvrzují, že gpt3 rozumí i poměrně specializovaným otázkám. [26]

A to byla jen malá část toho, čeho jsou moderní neuronové sítě schopny. Každý rok vytváří více a více databází a tréninkových sad, díky čemuž se vývoj a trénink neuronových sítí stává snadnějším a dostupnějším, což vede k nárůstu zájemců o toto odvětví a vzniku nových typů neuronových sítí pro řešení nejrůznějších každodenních i globálních problémů.

4 Vlastní práce

4.1 Cíl

Cílem praktické části je vytvořit a natrénovat neuronovou síť pro rozpoznávání digitálních obrázků palmýrské abecedy. Vzhledem k malému počtu obrázků v datovém souboru je jedním z úkolů augmentovat obrázky, tj. rozšířit objem dat pomocí jejich modifikace. Nakonec se provede vyhodnocení modelu neuronové sítě pomocí Matice záměn.

Neuronová síť byla vytvořena a natrénována v programovacím jazyce Python (verze 3.9.6) ve vývojovém prostředí Jupyter Notebook, kde je možné okamžitě vidět výsledek provádění kódu a jeho jednotlivé fragmenty.

Celý datový soubor pro učení a testování neuronové sítě byl získán od Ing. Adéla Hamplová.

4.2 Příprava dat na školení

Prvním krokem při práci s daty, ať už jde o analýzu dat nebo trénování neuronových sítí, je příprava těchto dat. Hlavní kroky tohoto procesu zahrnují shromažďování, filtrování a označování nezpracovaných dat do podoby vhodné pro algoritmy strojového učení a následné zkoumání a vizualizaci dat. Tyto postupy zabírají většinu času věnovaného strojovému učení.

Pro tuto bakalářskou práci byl poskytnut soubor dat rozdělený do 28 složek, přičemž každá složka odpovídala názvu jednoho z písmen palmýrské abecedy. Každá složka obsahovala přibližně 390 až 500 obrázků příslušného písmene a velikost každého obrázku byla 224x224 pixelů.

Při přípravě těchto dat jsem se rozhodl vytvořit tři datové sady:

- Tréninkový soubor dat
- Ověřovací soubor dat
- Soubor dat pro testování

K trénování se používá tréninková sada dat, na které se model učí skryté příznaky a vzorce v datech.

K ověření výkonnosti modelu během trénování se používá ověřovací soubor dat. Tento proces poskytuje informace, které pomáhají adekvátně upravit hyperparametry a konfigurace modelu.

Testovací sada je samostatný soubor dat, který se používá k testování modelu po dokončení trénování. Poskytuje objektivní konečnou metriku výkonnosti modelu z hlediska přesnosti předpovědi.

Každý soubor obsahuje 28 složek s písmeny, ale trénovací soubor nyní obsahuje 279 obrázků v každé složce, ověřovací a testovací sada obsahují po 60 obrázků.

Takové nerovnoměrné rozložení dat je nezbytné pro správné trénování neuronové sítě, kdy většina dat jde na trénování a pouze 10-20 % celkových dat jde na ověřování a testování.

4.3 Připojení potřebných knihoven

Po dokončení přípravy dat je možné začít vytvářet neuronovou síť.

Nejdříve je nutné zapojit všechny potřebné knihovny, které budou v této práci použity.

```
import os
import tensorflow as tf
import seaborn as sn
import numpy as np
from keras.models import load_model
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout,
Activation
import matplotlib.pyplot as plt
from keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import confusion_matrix, classification_report
%matplotlib inline
```

Zde jsou propojeny Tenserflow, Keras, Seaborn, Matplotlib a Sklearn.

Tenserflow a Keras pomohou pracovat se samotnou neuronovou sítí a poskytnou různé komfortní funkce pro trénování a načítání modelu a také funkci pro monitorování trénování, která pomůže uložit model po každé epoše a zastavit trénování v případě, že dojde k přeškolení.

Numpy, Seaborn, Matplotlib, Sklearn jsou potřebné pro práci s grafy, maticemi, stejně jako schopnost podporovat matematické funkce na vysoké úrovni.

4.4 Propojení datových souborů

Dalším krokem je připojení dat.

```
train_dir = ('D:/Neural_network_(Denis_Voinkov)/train')
val_dir = ('D:/Neural_network_(Denis_Voinkov)/val')
test_dir = ('D:/Neural_network_(Denis_Voinkov)/test')
image_size = (224, 224)
batch_size = 20
```

Bylo rozhodnuto vytvořit tři proměnné: `train_dir`, `val_dir`, `test_dir` které obsahují cestu k příslušné složce s daty.

Zde jsem také vytvořil proměnnou `image_size` obsahující velikost obrázku, která je v tomto případě 224x224 pixelů.

Protože nemůžeme prohnat celou sadu dat neuronovou sítí najednou, musíme ji rozdělit do několika dávek, proměnná `batch_size` určuje velikost jedné dávky, protože máme malou sadu dat, použijeme velikost 20 obrázků na dávku.

4.5 Vytvoření generátoru obrázků

4.5.1 Rozšíření dat

Rozšíření dat je vytvoření nových dat ze stávajících dat. Rozšíření lze použít na jakýkoli typ dat, od čísel po obrázky. Stávající fotografie lze například deformovat, otáčet a ořezávat, aby se zvýšil počet snímků.

Generátor obrázků je vytvořen na základě třídy `ImageDataGenerator`.

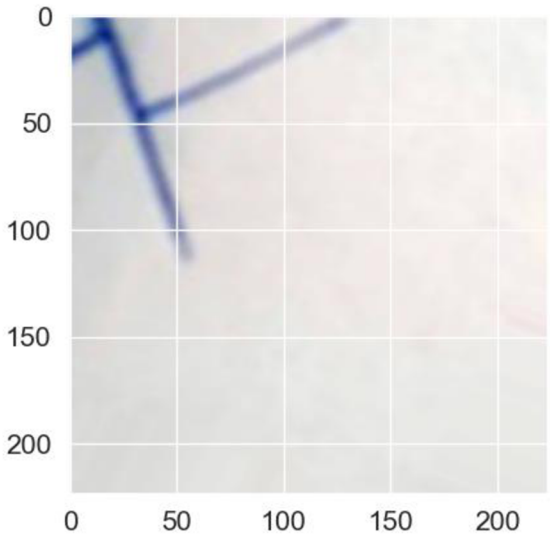
```
train_datagen = ImageDataGenerator(rescale=1. / 255,
                                   rotation_range=20,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   zoom_range=0.2,
                                   fill_mode='nearest')
```

- `rotation_range` určuje maximální počet stupňů, o které lze obrázek otočit
- `width_shift_range` posune obrázek o 20 % na šířku
- `height_shift_range` posune obrázek o 20 % na výšku
- `zoom_range` zvětší obrázek o 20 %

Keras při použití tohoto generátoru náhodně vybere několik těchto variant a použije je na původní obrázek.

Bylo testováno několik variant, jak silně bude generátor deformovat obrázek. Nastavení se silnější deformací poskytovala výsledky nevhodné pro trénování neuronové sítě, protože písmena často přesahovala hranice obrazu.

Příklad augmentace písmene Aleph, která není vhodná pro použití při školení:



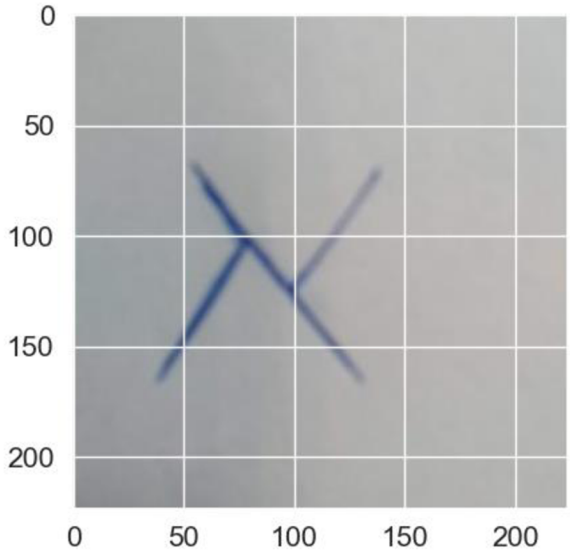
obrázek 17 - příklad špatně augmentovaného obrazu

4.5.2 Kontrola augmentace obrazu

Pro kontrolu správné funkce generátoru je třeba vybrat libovolný obrázek z trénovacího souboru dat, v tomto případě bylo vybráno písmeno Aleph.

```
image_file_name = train_dir + '/Aleph//Aleph (105).jpg'
img = tf.keras.utils.load_img(image_file_name, target_size=(224, 224))
plt.imshow(img)
```

Výsledek je následující:



obrázek 18 - písmeno Aleph z trénovacího souboru dat

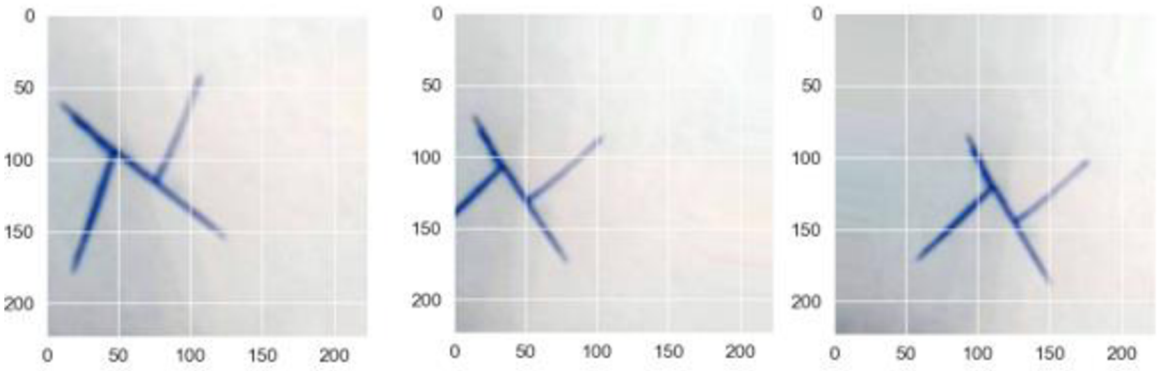
Toto písmeno Aleph z trenovaciho souboru dat neni zadnym zpusobem rozšířeno.

Dále provedeme augmentaci vybraného obrázku pomocí generátoru.

```
x = tf.keras.utils.img_to_array(img)
x = x.reshape((1,) + x.shape)
i = 0
for batch in train_datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(tf.keras.utils.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0:
        break
plt.show()
```

Protože generátor může běžet neomezeně dlouho, je třeba zadat ukončení cyklu.

Na základě toho bylo získáno několik výsledků augmentace obrazu:



obrázek 19 - příklad augmentace obrazu

Výsledné obrázky vykazují posuny na výšku, šířku a otočení o několik stupňů, i když se obrázky příliš neliší, k tréninku neuronové sítě to stačí. Z toho můžeme vyvodit závěr, že generátor funguje správně a je připraven k použití na celou trénovací sadu dat.

4.5.3 Generátory pro trénování, ověřování a testování

Na základě augmentovaných obrázků z trénovacího souboru dat je vytvořen speciální generátor trénovacích dat.

```
train_generator = train_datagen.flow_from_directory(

    train_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical')
```

`class_mode` používá `categorical`, protože tento dokument má 28 tříd pro predikci, třídu pro každé písmeno, což je pro vzájemně se vylučující označení. Například: pes nemůže být kočka, člověk není pes atd.

Vzhledem k tomu, že během validace a testování není rozšíření dat potřeba, je speciálně pro tyto operace vytvořen další samostatný generátor.

```
test_datagen = ImageDataGenerator(rescale=1. / 255)
```

Tento generátor dělí hodnotu každého pixelu číslem 255, takže vstup do neuronové sítě je mezi 0 a 1.

Z tohoto generátoru jsou pak vygenerovány následující dva, které slouží k ověření a otestování modelu.

Generátor dat pro ověření na základě obrázků z příslušného katalogu:

```
val_generator = test_datagen.flow_from_directory(
    val_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical')
```

Generátor dat pro testování na základě obrázků z příslušného katalogu:

```
test_generator = test_datagen.flow_from_directory(
    test_dir,
    shuffle = False,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical')
```

4.6 EarlyStopping

Keras nabízí velmi užitečnou funkci zpětných volání (`callbacks`), pomocí které lze implementovat mnoho praktických funkcí. V této práci jsou využity dvě funkce, a to `EarlyStopping` a `ModelCheckpoint`.

```
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.001,
                           patience=10, mode='auto')
chkpt = ModelCheckpoint("D:/Neural_network_(Denis_Voinkov)/model_1.h5",
                        monitor='val_loss',
                        save_best_only=True,
                        mode='auto')
callbacks = [early_stop, chkpt]
```

`EarlyStopping` pomáhá automaticky zjistit přetrénování neuronové sítě a zastavit učení.

Funguje to následovně: po každé epoše se zkontroluje hodnota `val_loss`, tj. ztráta na ověřovacích datech, pokud se tato hodnota sníží, vše je v pořádku, učení pokračuje a funkce checkpoint po úspěšné epoše uloží model. V případě, že se `val_loss` nezmění, nedojde po takové epoše k uložení. Parametr `patience` (trpělivost) určuje, kolik epoch musí projít bez snížení hodnoty `val_loss` před ukončením tréninku.

V důsledku toho bude zachován nejlepší model s nejnižší hodnotou ztráty a trénink neuronové sítě bude ukončen před zahájením jejího přeškolování.

4.7 Vytvoření neuronové sítě

Dalším krokem, jakmile jsou sady dat připraveny a nastaveny, je vytvoření architektury neuronové sítě.

V případě této práce byla použita Konvoluční neuronová síť, protože tento typ neuronové sítě je nejvhodnější pro úlohy klasifikace digitálních obrazů.

Samotná architektura CNN, počet skrytých a plně propojených vrstev se může lišit a jejich parametry mohou být různé a nejsou přesně definovány.

Pro řešení úkolu bylo natrénováno celkem 6 různých modelů neuronových sítí s různou architekturou, z nichž byl vybrán nejúspěšnější model, jehož architektura bude použita v této bakalářské práci.

```
model = Sequential()
model.add(Conv2D(16, (5, 5), input_shape=(224, 224, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (5, 5)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (5, 5)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (5, 5)))
model.add(Activation('relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(28))
model.add(Activation('softmax'))
```

Tato architektura je poměrně tradiční pro problémy klasifikace obrazu. Střídá konvoluční vrstvy a max pooling vrstvy, přičemž jedna konvoluce má velikost 5x5. Na konci je část s plným propojením pro klasifikaci a výstupem je 28 neuronů podle počtu tříd v datové sadě.

Síť vytvořená k řešení problému je hluboká a využívá 11 vrstev:

- 4 konvoluční vrstvy
- 4 max pooling vrstvy
- 3 plně propojené vrstvy

4.7.1 Konvoluční neuronová síť

Podívejme se podrobněji na architekturu konvoluční neuronové sítě.

`model = Sequential()` - vytvoří se řetězová síť, kde vrstvy následují jedna za druhou.

`model.add(Conv2D(16, (5, 5), input_shape=(224, 224, 3)))` - konvoluční vrstva, která pracuje s dvourozměrnými daty, vrstva má 16 map příznaků, velikost konvolučního jádra 5x5, velikost vstupních dat 224x224x3, což odpovídá 3 obrazovým kanálům pro barevný kód RGB a velikosti obrazu 224x224 pixelů.

`model.add(Activation('relu'))` - jako aktivační funkce se používá ReLu.

`model.add(MaxPooling2D(pool_size=(2, 2)))` - vrstva max pooling s redukcí dimenzionality 2x2.

Následuje několik dalších podobných vrstev, kde se mění pouze velikost map příznaků.

Po konvoluční kaskádě následuje klasifikátor, který na základě znaků nalezených konvoluční sítí určí, do které třídy objekt na obrázku patří.

4.7.2 Plně propojená část

Podívejme se nyní blíže na architekturu plně propojené části neuronové sítě.

`model.add(Flatten())` - převede síť z dvourozměrné reprezentace na plochou.

`model.add(Dense(1024))` - plně propojená vrstva obsahující 1024 neuronů a využívající aktivaci ReLu. Poté následuje další vrstva s 256 neurony.

`model.add(Dropout(0.2))` - technika prevence overfittingu, je náhodně vypnuto několik neuronů s pravděpodobností 20 %, takže zbývající neurony v síti musí samy vybrat váhy.

`model.add(Dense(28))` - výstupní vrstva s 28 neurony podle počtu tříd v souboru dat.

`model.add(Activation('softmax'))` - na této vrstvě používá aktivační funkci softmax, která odpovídá pravděpodobnosti výskytu určité třídy, přičemž celková výstupní hodnota všech 28 neuronů je rovna jedné.

4.7.3 Kompilace modelu

Jakmile je síť nastavena, měla by být zkompilována.

```
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])
```

Volá se metoda `model.compile`, která jako chybovou funkci používá kategoriální crossentropii, což je vhodné, pokud je výstupem pravděpodobnost výskytu tříd.

Optimalizovat pomocí Adam – jednoho z nejefektivnějších optimalizačních algoritmů pro trénování neuronových sítí.

Jako měřítko používáme `accuracy` (přesnost).

4.7.4 Trénování neuronové sítě

```
history = model.fit(
    train_generator,
    steps_per_epoch=100,
```

```
epochs=100,
validation_data=val_generator,
validation_steps=25,
callbacks = callbacks)
```

Pro trénování sítě voláme metodu `fit`. Trénujeme síť pomocí `train_generator`, jehož výstupní data již zohledňují augmentaci obrazu, a protože generátor může běžet nekonečně dlouho, je nutné zadat počet kroků na epochu `steps_per_epoch`.

Dále je třeba zadat počet epoch. Vzhledem k tomu, že zpětná volání EarlyStopping byla propojena, bylo zadáno 100 epoch, protože neexistuje možnost přetrénování.

Pro ověření `val_generator` je rovněž určen generátor dat, a protože tento generátor může běžet neomezeně dlouho, je určen konkrétní počet kroků - 25.

Výsledek učení:

```
Epoch 32/100
100/100 [==============================] - 100s 993ms/step - loss: 0.6186 - accuracy: 0.8100 - val_loss: 0.4307 - val_accuracy: 0.8960
Epoch 33/100
100/100 [==============================] - 99s 991ms/step - loss: 0.5661 - accuracy: 0.8170 - val_loss: 0.5657 - val_accuracy: 0.8840
Epoch 34/100
100/100 [==============================] - 99s 986ms/step - loss: 0.6264 - accuracy: 0.8060 - val_loss: 0.5056 - val_accuracy: 0.8660
Epoch 35/100
100/100 [==============================] - 98s 982ms/step - loss: 0.5110 - accuracy: 0.8350 - val_loss: 0.5586 - val_accuracy: 0.8700
Epoch 36/100
100/100 [==============================] - 99s 986ms/step - loss: 0.5526 - accuracy: 0.8270 - val_loss: 0.5610 - val_accuracy: 0.8840
Epoch 37/100
100/100 [==============================] - 101s 1s/step - loss: 0.4957 - accuracy: 0.8550 - val_loss: 0.6492 - val_accuracy: 0.8380
Epoch 38/100
100/100 [==============================] - 99s 989ms/step - loss: 0.5436 - accuracy: 0.8365 - val_loss: 0.5299 - val_accuracy: 0.8780
Epoch 39/100
100/100 [==============================] - 99s 985ms/step - loss: 0.5416 - accuracy: 0.8345 - val_loss: 0.5214 - val_accuracy: 0.8780
Epoch 40/100
100/100 [==============================] - 99s 992ms/step - loss: 0.4987 - accuracy: 0.8502 - val_loss: 0.4912 - val_accuracy: 0.8900
Epoch 41/100
100/100 [==============================] - 99s 986ms/step - loss: 0.4643 - accuracy: 0.8630 - val_loss: 0.6576 - val_accuracy: 0.8580
Epoch 42/100
100/100 [==============================] - 108s 1s/step - loss: 0.4482 - accuracy: 0.8627 - val_loss: 0.4996 - val_accuracy: 0.8820
```

obrázek 20 - výsledek učení neuronové sítě

V epoše 32 byl uložen model s nejnižší hodnotou val_loss a přesností predikce 89,6 % na ověřovacích datech. Další epochy nevykazovaly zlepšení výsledku a nebyly uloženy, takže trénink byl ukončen v epoše 42.

5 Testování a analýza výsledků

5.1 Testování modelu

Po vytvoření a učení modelu je třeba provést jeho testování a analyzovat výsledky.

Vyhodnocení kvality sítě pomocí metody `evaluate`.

```
score = model.evaluate(test_generator)
print("Accuracy on test data: %.2f%%" % (score[1]*100))
```

```
Accuracy on test data: 78.33%
```

Výsledek přesnosti predikce na testovacích datech: 78.33 %

Ačkoli se výsledek testu neblíží ideálním 100 %, lze ho stále považovat za poměrně slušný.

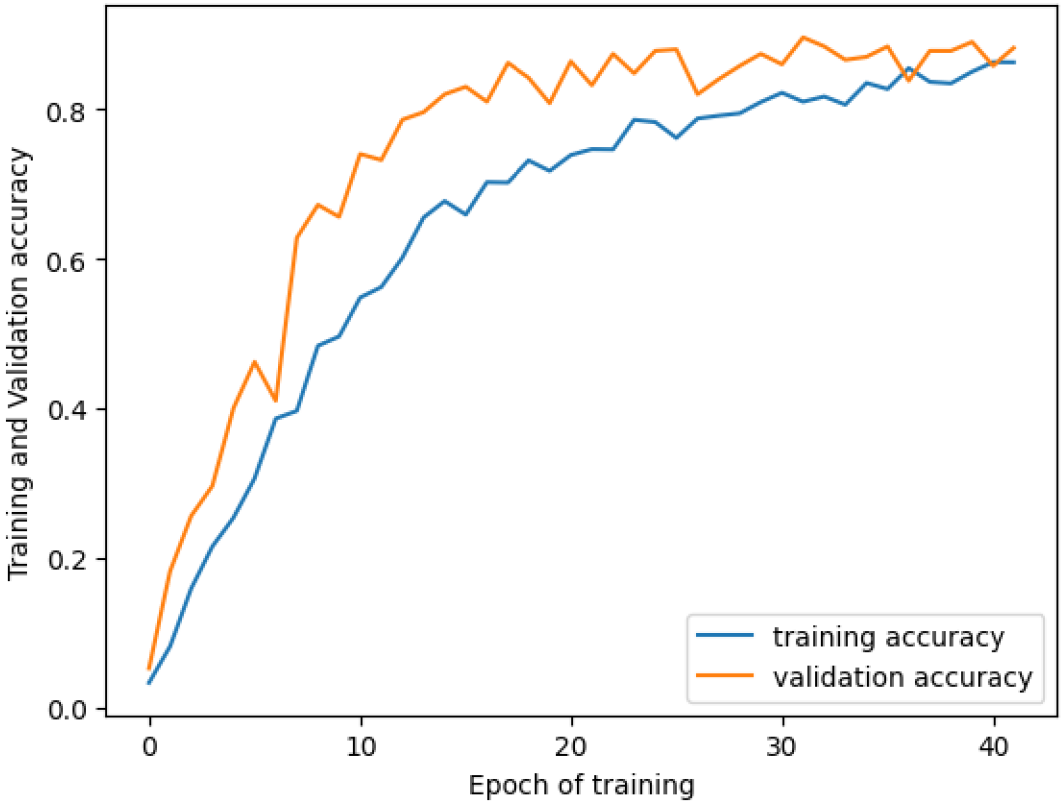
5.2 Grafy přesnosti a ztráty

Grafy kvality učení pomohou lépe si představit, jak probíhalo školení, jak se zlepšila přesnost a se snížily ztráty.

Zobrazení podílu správných odpovědí na trénovací a validační sadě dat:

```
plt.plot(history.history['accuracy'],
         label='training accuracy')
plt.plot(history.history['val_accuracy'],
         label='validation accuracy')
plt.xlabel('Epoch of training')
plt.ylabel('Training and Validation accuracy')
plt.legend()
plt.show()
```

Výsledek:



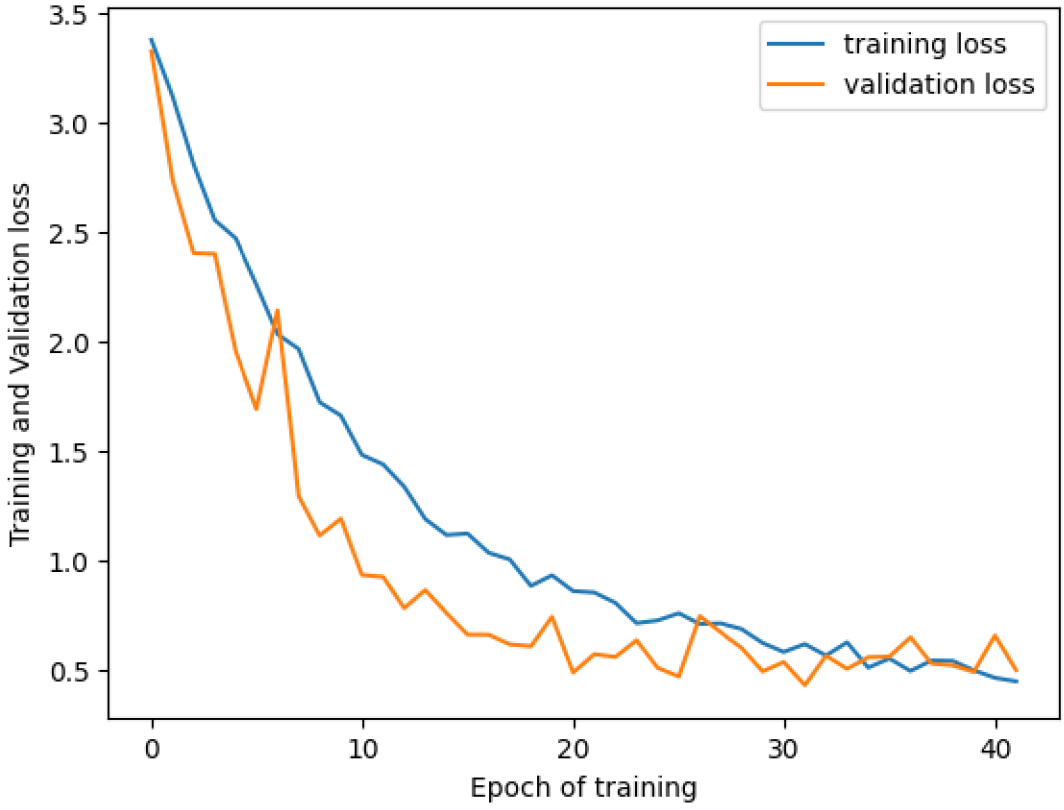
obrázek 21 - graf přesnosti

Graf ukazuje plynulý nárůst přesnosti predikce sítě, což je dobrým ukazatelem toho, že model byl správně natrénován. Neexistují ani žádné známky přetrénování, které se obvykle projevuje jako plató na grafu.

Zobrazení grafu chyb na trénovací a validační sadě dat:

```
plt.plot(history.history['loss'],
         label='training loss')
plt.plot(history.history['val_loss'],
         label='validation loss')
plt.xlabel('Epoch of training')
plt.ylabel('Training and Validation loss')
plt.legend()
plt.show()
```

Výsledek:



obrázek 22 - graf ztrát

Graf ztráty je podobný předchozímu a ukazuje, jak se hodnota loss s každou epochou snižovala.

Výsledek vypadá přijatelně a lze soudit, že učení proběhlo správně, ale po 32. epoše hodnota val_loss přestala klesat a naopak vzrostla, což může ukazovat na to, že začalo přetrénování.

Model byl však uložen naposled v epoše 32, proto nebyl model přeškolován.

5.3 Matice záměn

V dalším kroku se podíváme na matici záměny a na hodnoty Precission, Recall a F1-Score, které poskytuje knihovna SciKit Learns.

Před konstrukcí matice chyb je však nutné vytvořit generátor predikcí:

```
predictions = model.predict(test_generator,
                            steps=None,
                            max_queue_size=10,
                            workers=0,
```

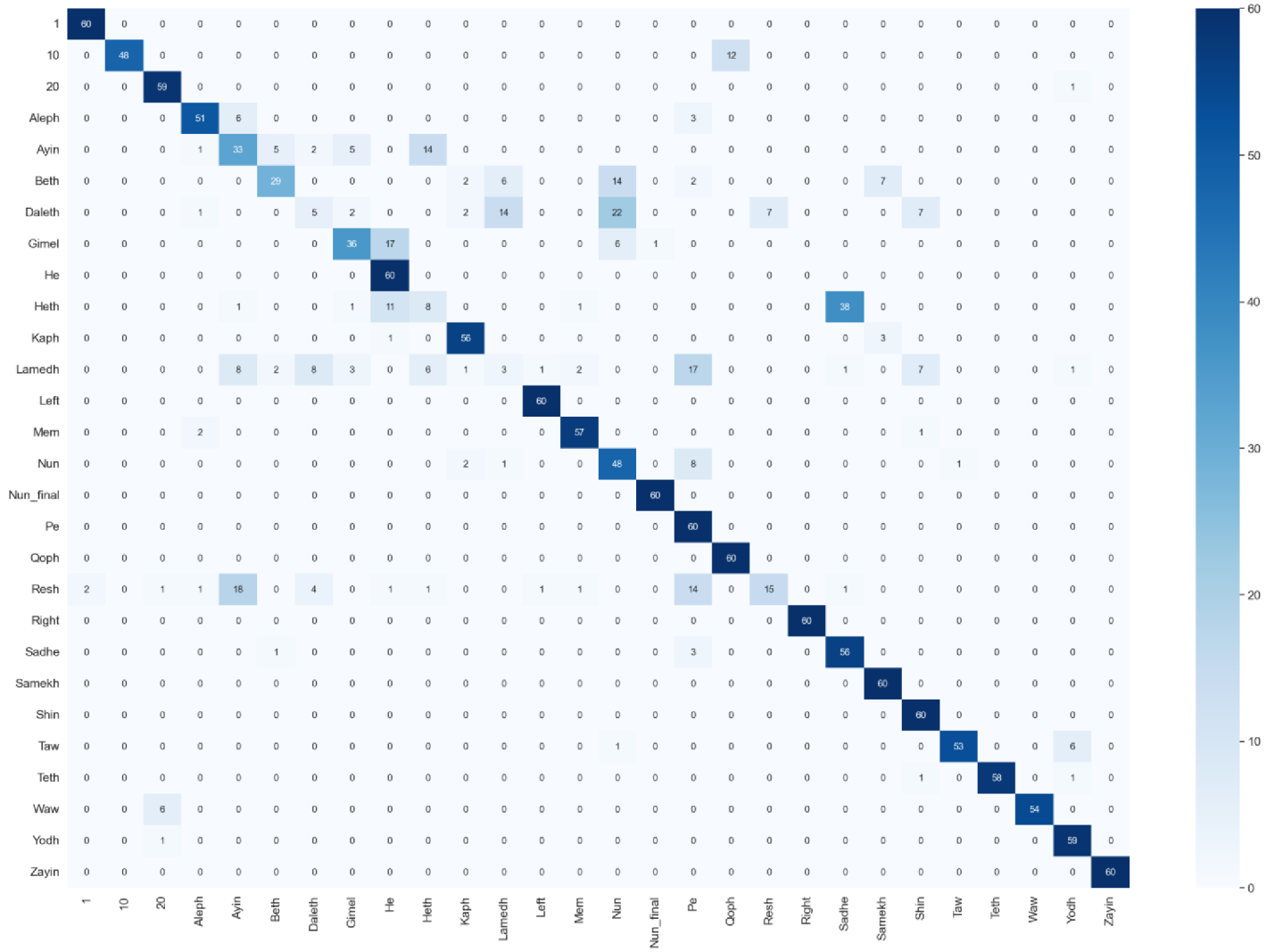
```
                                use_multiprocessing=False,
                                verbose=1)

    predicted = np.argmax(predictions, axis=1)
    trueClass = test_generator.classes[test_generator.index_array]
    classLabels = list(test_generator.class_indices.keys())
```

Kde proměnná `predicted` uchovává predikované hodnoty a proměnná `trueClass` uchovává referenční hodnoty.

Poté se vygeneruje matice chyb a klasifikační zpráva:

```
    print('Confusion Matrix')
    cm = (confusion_matrix(y_true=trueClass, y_pred=predicted))
    plt.figure(figsize=(30,20))
    sn.set(font_scale=1.4)
    sn.heatmap(cm, annot=True, annot_kws={"size": 12}, cmap='Blues',
xticklabels=classLabels, yticklabels=classLabels)
    plt.show()
    print()
    print('Classification Report')
    print(classification_report(test_generator.classes, predicted,
                                target_names=classLabels))
```

Výsledky:



obrázek 23 - matice záměn

Jak je vidět z matice, klasifikátor rozpozná většinu obrázků správně. Diagonální prvky matice jsou jasně zvýrazněny. V rámci některých tříd (Daleth, Heth, Lamedh, Resh) však klasifikátor vykazuje nízkou přesnost.

Pro podrobnější analýzu matice chyb použijeme klasifikační metriky:

Classification Report

	precision	recall	f1-score	support
1	0.97	1.00	0.98	60
10	1.00	0.80	0.89	60
20	0.88	0.98	0.93	60
Aleph	0.91	0.85	0.88	60
Ayin	0.50	0.55	0.52	60
Beth	0.78	0.48	0.60	60
Daleth	0.26	0.08	0.13	60
Gimel	0.77	0.60	0.67	60
He	0.67	1.00	0.80	60
Heth	0.28	0.13	0.18	60
Kaph	0.89	0.93	0.91	60
Lamedh	0.12	0.05	0.07	60
Left	0.97	1.00	0.98	60
Mem	0.93	0.95	0.94	60
Nun	0.53	0.80	0.64	60
Nun_final	0.98	1.00	0.99	60
Pe	0.56	1.00	0.72	60
Qoph	0.83	1.00	0.91	60
Resh	0.68	0.25	0.37	60
Right	1.00	1.00	1.00	60
Sadhe	0.58	0.93	0.72	60
Samekh	0.86	1.00	0.92	60
Shin	0.79	1.00	0.88	60
Taw	0.98	0.88	0.93	60
Teth	1.00	0.97	0.98	60
Waw	1.00	0.90	0.95	60
Yodh	0.87	0.98	0.92	60
Zayin	1.00	1.00	1.00	60
accuracy			0.79	1680
macro avg	0.77	0.79	0.76	1680
weighted avg	0.77	0.79	0.76	1680

obrázek 24 - klasifikační metriky

Výsledky modelu budeme analyzovat pomocí metriky F1-Score jako nejlepšího odhadu kvality klasifikátoru.

Na základě této metriky můžeme konstatovat, že většina tříd je odhadována v rozmezí 70 % až 98 %, což je ukazatel dobře vycvičené neuronové sítě.

Existuje však i několik tříd s výrazně nízkým hodnocením:

- Daleth s výsledkem 13 %
- Heth s výsledkem 18 %

- Lamedh s výsledkem 7 %
- Resh s výsledkem 37 %

5.4 Shrnutí výsledků

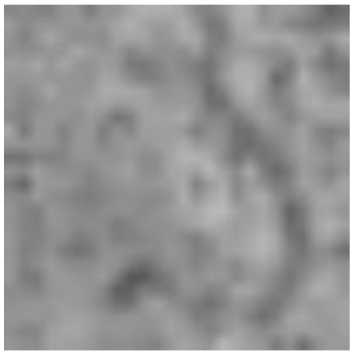
Po otestování dat a analýze grafů platnosti a ztrát lze konstatovat, že samotné školení proběhlo správně a nebyly zjištěny žádné zjevné chyby.

Co se týče kvality predikce, je na přijatelné úrovni a s vysokou pravděpodobností dokáže odhalit většinu písmen palmýrské abecedy.

Největším problémem pro model jsou písmena Daleth, Heth, Lamedh, Resh. Neuronová síť většinou nedokáže tato písmena identifikovat a dosahuje nízkých výsledků.

Řešením tohoto problému a celkovým zlepšením kvality učení může být zvětšení datového souboru. V práci je použita augmentace dat, která pomáhá uměle zvětšit množství dat, ale není ideálním řešením a nemůže plně nahradit velké množství dat, což ukazují výsledky zkoumaného modelu. Kvalitativní soubor dat obvykle obsahuje přibližně 80 000 obrázků, z nichž 60 000 je určeno k trénování a zbytek ke kontrole a testování modelu.

Také je důležité věnovat pozornost nejen objemu dat, ale i jejich kvalitě, aby model mohl během učení správně rozpoznat obrázek a jeho detaily.



obrázek 25 - nevhodný obraz pro učení

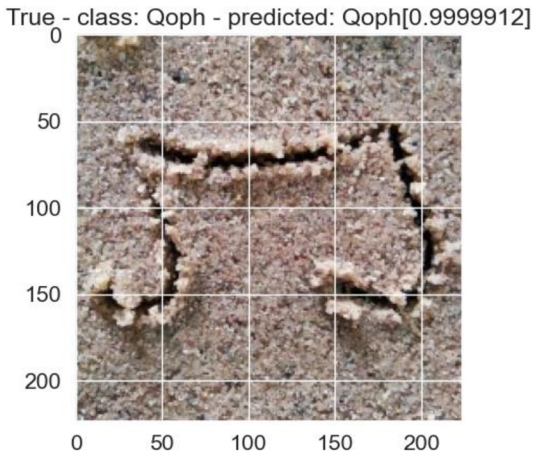
5.5 Obrázek a předpověď

Nakonec můžeme výsledky predikce vizualizovat a podívat se na několik obrázků s jejich třídou a predikcí.

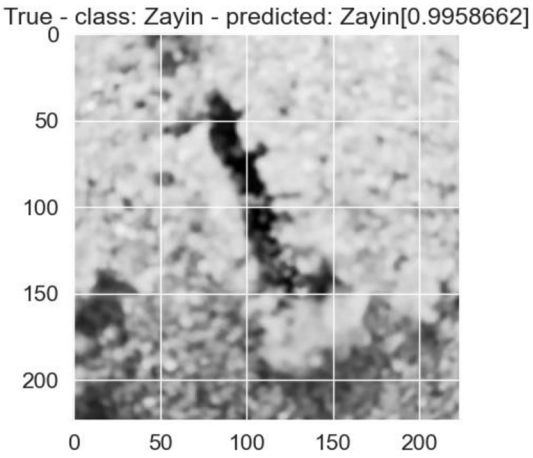
```
def predict_one(model):
    image_batch, classes_batch = next(test_generator)
    predicted_batch = model.predict(image_batch)
    for k in range(0,image_batch.shape[0]):
      image = image_batch[k]
      pred = predicted_batch[k]
      the_pred = np.argmax(pred)
```

```
        predicted = target_names[the_pred]
        val_pred = max(pred)
        the_class = np.argmax(classes_batch[k])
        value = target_names[np.argmax(classes_batch[k])]
        plt.figure(k)
        isTrue = (the_pred == the_class)
        plt.title(str(isTrue) + ' - class: ' + value + ' - ' + 'predicted: ' +
predicted + '[' + str(val_pred) + ']')
        plt.imshow(image)
predict_one(model)
```

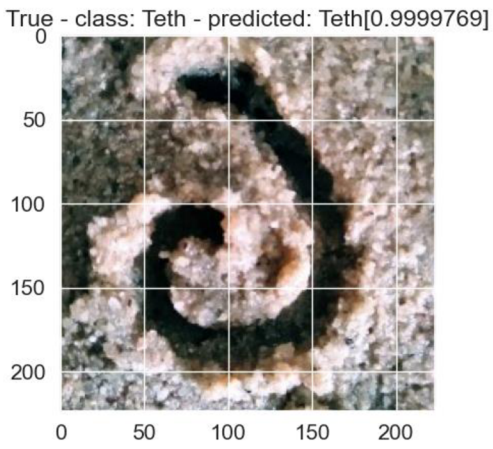
Výsledky:



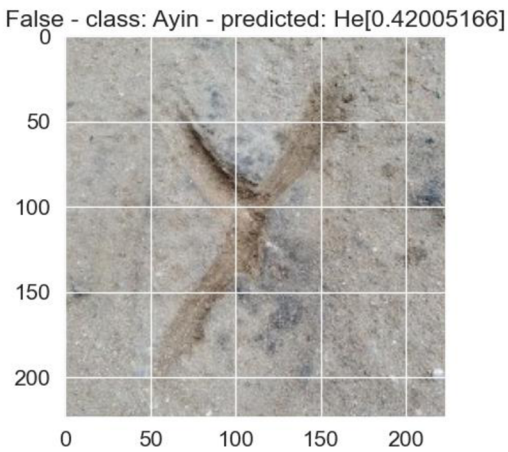
obrázek 27 - příklad predikce č.1



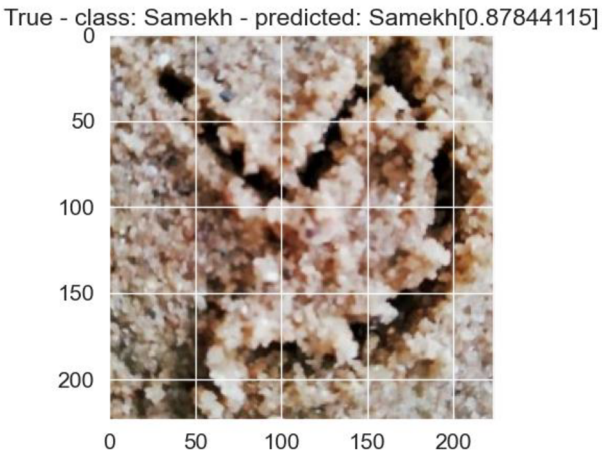
obrázek 26 - příklad predikce č.2



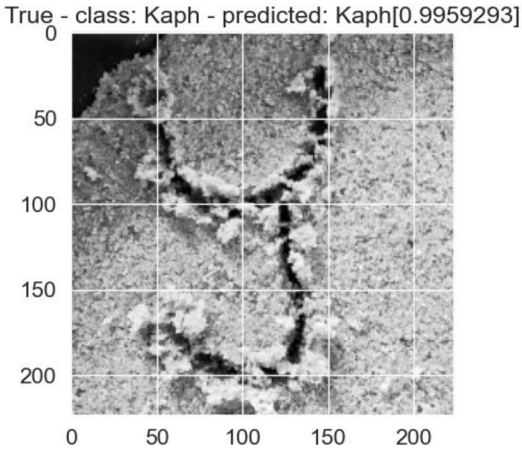
obrázek 29 - příklad predikce č.3



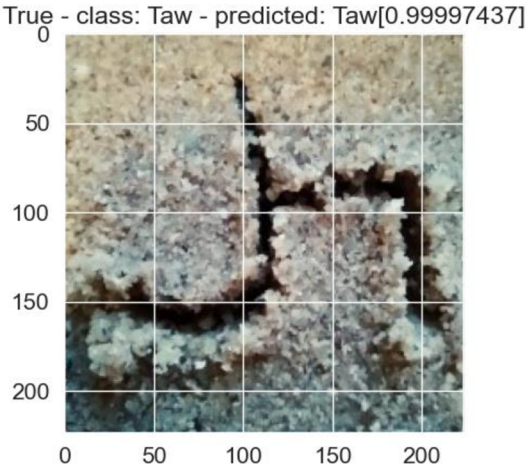
obrázek 28 - příklad predikce č.4



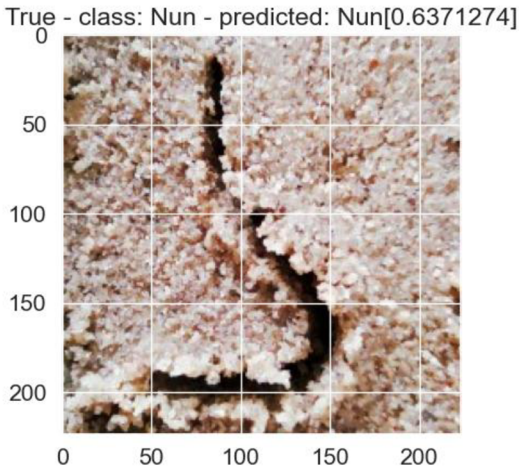
obrázek 31 - příklad predikce č.5



obrázek 30 - příklad predikce č.6



obrázek 32 - příklad predikce č.7



obrázek 33 - příklad predikce č.8

Závěr

Cílem této práce bylo vytvořit a natrénovat neuronovou síť pro rozpoznávání palmýrské abecedy, připravit trénovací soubor dat a vybrat nejvhodnější architekturu neuronové sítě pro úlohu klasifikace obrazu.

Domnívám se, že cíle byly splněny, protože konečný model neuronové sítě vykazuje 78% úspěšnost predikce na testovacích datech a je schopen rozpoznat většinu písmen palmýrské abecedy s vysokou přesností, což podporují i výsledky hlubší analýzy modelu pomocí matice chyb, metrik, grafů ztrát a přesnosti. Nicméně tento model není ideální a má problémy s rozpoznáním čtyř písmen abecedy: Daleth, Heth, Lamedh, Resh. Nejedná se o kritický problém a lze jej vyřešit dodatečným tréninkem na větším objemu dat.

Na základě analýzy tohoto problému lze dojít k závěru, že nejdůležitější částí při práci s neuronovými sítěmi jsou data, na kterých bude probíhat trénování a testování. Architekturu a hyperparametry modelu lze snadno a rychle měnit a samotný proces trénování trvá obvykle několik minut až hodin, ale nejvíce času zabere vytvoření datového souboru. Shromažďování velkého množství dat je pracné a časově náročné, zejména pokud jde o data, jako je starověká abeceda, která se používala již před naším letopočtem. Právě dostupnost velkého množství dat shromážděných různými institucemi, korporacemi a výzkumnými organizacemi, způsobila v posledních letech zájem o neuronové sítě a jejich široké využití jak pro komerční účely, tak pro výzkum, lékařství, a dokonce i pro umění.

Při psaní této práce jsem získal mnoho teoretických znalostí v oblasti analýzy dat a návrhu neuronových sítí. Tyto znalosti jsem dokázal využít v praxi, například při výběru správné architektury neuronové sítě, aktivačních a ztrátových funkcí, interpretaci výsledků.

6 Seznam použitých zdrojů

1. MCCARTHY, John. A PROPOSAL FOR THE DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE. *Formal Reasoning Group* [online]. Apr 3 19:48:31 PST 1996 [cit. 2021-12-11]. Dostupné z: http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html
2. WEIZENBAUM, Joseph. *Computer Power and Human Reason*. 1976. San Francisco: WH Freeman and Company, 1976. ISBN 978-0716704645.
3. Forklog Team. Co je to neuronová síť?. *Forklog* [online]. 29.09.2021 [cit. 2021-12-10]. Dostupné z: https://forklog.com/cryptorium/chto-takoe-nejronnaya-set
4. SHAIKHUTDINOV, Arnis. Neural networks for beginners. Part 2. *Habr* [online]. February 12, 2017 at 04:38 pm [cit. 2021-12-13]. Dostupné z: https://habr.com/ru/post/313216/
5. SHARMA V, Avinash. Understanding Activation Functions in Neural Networks. *Medium* [online]. Mar 30, 2017 [cit. 2021-12-13]. Dostupné z: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0
6. VEEN, Van. Co je to neuronová síť?. *The Asimov Institute* [online]. 22 April 2019 [cit. 2021-12-13]. Dostupné z: https://www.asimovinstitute.org/neural-network-zoo/
7. BALAKIREV, Sergey. Structure and principle of operation of fully connected neural networks. *Proproprogs* [online]. 19 Jun 2020 [cit. 2021-12-15]. Dostupné z: https://www.asimovinstitute.org/neural-network-zoo/
8. S., Suryansh. Gradient Descent: All You Need to Know. *Hackernoon* [online]. March 10th 2018 [cit. 2021-12-15]. Dostupné z: https://hackernoon.com/gradient-descent-aynk-7cbe95a778da
9. MAZUR, Matt. A Step by Step Backpropagation Example. *Mattmazur* [online]. March 10th 2018 [cit. 2021-12-18]. Dostupné z: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
10. SETH, Neha. How does Backward Propagation Work in Neural Networks?. *Analytics Vidhya* [online]. June 8th, 2021 [cit. 2021-12-18]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/06/how-does-backward-propagation-work-in-neural-

networks/#:~:text=Forward%20Propagation%20is%20the%20way,is%20called%20the%20Backward%20Propagation.

11. VORONTSOV, Konstantin. Underfitting and overfitting in machine intelligence. *Postnauka* [online]. 02.06.2020 [cit. 2021-12-28]. Dostupné z: https://postnauka.ru/video/154955
12. NARKHEDE, Sarang. Understanding Confusion Matrix. *Towards Data Science* [online]. May 9, 2018 [cit. 2021-12-27]. Dostupné z: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62
13. LEGOTSKOY, Evgeny. Image recognition in Python with TensorFlow and Keras. *EVILEG* [online]. May 13, 2020 [cit. 2021-12-09]. Dostupné z: https://evileg.com/ru/post/619/
14. MALLICK, Satya. Image Recognition and Object Detection : Part 1. *LearnOpenCV* [online]. NOVEMBER 14, 2016 [cit. 2021-12-09]. Dostupné z: https://learnopencv.com/image-recognition-and-object-detection-part1/
15. ZUYKOVA, Asya. What is computer vision and where is it used?. *RBC Trends* [online]. 25.02.2021 [cit. 2021-12-09]. Dostupné z: https://trends.rbc.ru/trends/industry/5f1f007e9a794756fafbfa83
16. POZDNYAKOV, Dmitry. Effective Geometric Hashing of the Feature Hyperspace for a Quick Accurate Search of the Most Similar Descriptors. *Habr* [online]. October 11, 2021 [cit. 2021-12-09]. Dostupné z: https://habr.com/ru/post/582708/
17. Skillfactory Team. Framework. *Skillfactory.Blog* [online]. December 17, 2021 [cit. 2021-12-28]. Dostupné z: https://blog.skillfactory.ru/glossary/framework/
18. GLAYBORODA, Marina. Top 10 Frameworks for Artificial Intelligence: Part One. *Skillfactory.Blog* [online]. August 23, 2019 [cit. 2021-12-28]. Dostupné z: https://vc.ru/ml/80391-top-10-freymvorkov-dlya-iskusstvennogo-intellekta-chast-pervaya
19. Technostacks Team. Top 7 Frameworks That Have Enhanced Machine Learning. *Technostacks* [online]. Nov 13 2018 [cit. 2021-12-28]. Dostupné z: https://technostacks.com/blog/machine-learning-frameworks/
20. GRIMOV, Ruslan. Neural networks prefer textures and how to deal with it. *Habr* [online]. May 29, 2019 [cit. 2022-01-07]. Dostupné z: https://habr.com/ru/company/ods/blog/453788/

21. Celsus Team. Neural networks in medicine: what are they and how do they work?. *Celsus* [online]. 22.09.2020 [cit. 2022-01-07]. Dostupné z: https://celsus.ai/about-us/
22. Reuters Team. Shark-detecting drones to patrol Australian beaches. *Reuters* [online]. August 25, 2017 [cit. 2022-01-07]. Dostupné z: https://www.reuters.com/article/us-australia-sharkdrone-idUSKCN1B51KB
23. LUNDEN, Ingrid. Deep Instinct nabs $43M for a deep-learning cybersecurity solution that can suss an attack before it happens. *Techcrunch* [online]. February 12, 2020 [cit. 2022-01-07]. Dostupné z: https://techcrunch.com/2020/02/12/deep-instinct-nabs-43m-for-a-deep-learning-cybersecurity-solution-that-can-suss-an-attack-before-it-happens/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAACmcfhzQvRHFk4sVZh-BwXowiZymjXjkNk1m3hieBnrV-eNKT6kG6pxHVpecIbQrCSi_TgSVkbBl6e-jvPCjFzNUAM2f59aZu4CbIqup-wZ8p6SQRbOVHTm7S8W0mjNEVwXRMCR74DUvZ8nxu-_qz99tRFauMzxtX5J_bAb9dMys
24. RÖSSLER, Claudia. Empowering passionate people to tackle world's food challenges. *Microsoft Industry Blogs* [online]. 18.05.2017 [cit. 2022-01-07]. Dostupné z: https://cloudblogs.microsoft.com/industry-blog/en-gb/cross-industry/2017/05/18/empowering-passionate-people-tackle-worlds-food-challenges/
25. Openai Team. DALL·E: Creating Images from Text. *OpenAI* [online]. January 5, 2021 [cit. 2022-01-08]. Dostupné z: https://openai.com/blog/dall-e/
26. IVTUSHOK, Elizabeth. The third generation of the OpenAI algorithm has learned how to perform text tasks by a few examples. *N+1* [online]. 29.05.20 [cit. 2022-01-08]. Dostupné z: https://nplus1.ru/news/2020/05/29/gpt-3

7 Seznam obrázků a zkratek

7.1 Seznam obrázků

7.2 Seznam použitých zkratek

CNN – A convolutional neural network (Konvoluční neuronová síť)

ReLu – Rectified linear unit (Rektifikovaná lineární jednotka)

Přílohy

Odkaz na projekt na GitHubu:

DenisVoinkov/neural-network-for-object-recognition: Creation and training of a neural network for recognition of digital images of the Palmyra alphabet. (github.com)