Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Zpracování obrazu egyptských hieroglyfů metodami hlubokého učení

Petra Andrejsková

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Ing. Petra Andrejsková

Informatika

Název práce

Zpracování obrazu egyptských hieroglyfů metodami hlubokého učení

Název anglicky

Image processing of Egyptian hieroglyphs by deep learning methods

---

Cíle práce

Cílem práce je prokázat efektivitu moderních metod zpracování obrazu založenou na použití algoritmů hlubokého učení. Student si vybere úlohu, k jejímuž řešení je analýza obrazu nezbytná a pro kterou může získat vhodný datový soubor. Navrhne metodiku řešení úlohy založenou na algoritmech hlubokého učení z knihoven Tensorflow a Keras a na datovém souboru ověří její efektivitu.

Metodika

Student popíše současný stav výzkumu v oblasti zpracování obrazu metodami hlubokého učení a popíše některé typické aplikace. Dále popíše opensource software určený k jejich realizaci, především frameworky tensorflow a keras. Vybere si vhodnou úlohu, pro kterou jsou na internetu k dispozici dostupná data. Navrhne metodiku řešení úlohy, řešení úlohy realizuje ve frameworku keras a na datech ověří.

Doporučený rozsah práce

60

Klíčová slova

deep learning, zpracování obrazu, klasifikace, convolutional neural network, egyptské hieroglyfy

---

Doporučené zdroje informací

GÉRON, Aurélien. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems.* Beijing ; Boston ; Farnham ; Sevastopol ; Tokyo: O'Reilly, 2019. ISBN 978-1-4920-3264-9.

CHOLLET, F. Deep Learning with Python, Second Edition, 2021, ISBN: 9781617296864

Předběžný termín obhajoby

2022/23 ZS – PEF

Vedoucí práce

doc. Ing. Arnošt Veselý, CSc.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 31. 10. 2022

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 28. 03. 2024

Čestné prohlášení

Prohlašuji, že svou diplomovou práci Zpracování obrazu egyptských hieroglyfů metodami hlubokého učení jsem vypracovala samostatně pod vedením vedoucího diplomové 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 autorka uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušila autorská práva třetích osob.

V Praze dne 29. 3. 2024 _____

Poděkování

Ráda bych touto cestou poděkovala vedoucímu mé diplomové práce doc. Ing. Arnoštu Veselému CSc. za poskytované cenné rady a připomínky, kterými přispěl k vytvoření této diplomové práce.

Děkuji Bc. Martinu Jiřičkovi za neocenitelné rady a doporučení při zajištění a zprovoznění připojení ke vzdálenému počítači. Za všestrannou podporu a jazykové korektury děkuji mé matce Mgr. Janě Andrejskové a Dorotě Skřivanové.

Zpracování obrazu egyptských hieroglyfů metodami hlubokého učení

Abstrakt

Diplomová práce se soustředí na aplikaci konvoluční neuronové sítě, která bude schopna rozpoznat egyptské hieroglyfy. Hlavním smyslem a účelem závěrečné diplomové práce bude:

a) Vysvětlit teoretické zásady hlubokého učení v souvislosti s jeho uplatněním v oblasti identifikace a klasifikace egyptských hieroglyfů, popsat historický vývoj etap konvolučních sítí a poukázat na některé reálné příklady aplikace CNN,
b) analyzovat datový soubor pro lepší pochopení jeho struktury a následné upravení dat tak, aby byla vhodná pro řešení výše uvedené problematiky,
c) sestavit a optimalizovat sekvenční více klasifikační model konvoluční neuronové sítě,
d) ověřit funkčnost navrženého řešení a prezentovat výsledky,
e) generalizovat ověřené postupy pro další možné využití.

Stěžejními nástroji ke zpracování praktické části budou metody a techniky hlubokého učení, zejména konvoluční neuronové sítě. Realizace proběhne za pomoci programovacího jazyka Python, a především využití knihoven Keras a TensorFlow. Navrhované řešení bude brát v úvahu definované požadavky a očekávání spojená s řešením problematiky. Na základě syntézy teoretických poznatků a dosažených výsledků budou formulovány závěry diplomové práce a následně generalizovány pro další možné použití a aplikace.

Klíčová slova: hloubkové učení, zpracování obrazu, klasifikace, konvoluční neuronová síť, egyptské hieroglyfy

Image processing of Egyptian hieroglyphs by deep learning methods

Abstract

The master's thesis focuses on the application of a convolutional neural network capable of recognizing Egyptian hieroglyphs. The main purpose and objective of the final work will be:

a) To explain the theoretical principles of deep learning in relation to its application in the field of identification and classification of Egyptian hieroglyphs, describe the historical development stages of convolutional networks, and point out some real examples of CNN application,
b) To analyse the data set for a better understanding of its structure and subsequent data modification to be suitable for solving the above-mentioned issues,
c) To compile and optimize a sequential multiclassification model of a convolutional neural network,
d) To verify the functionality of the proposed solution and present the results,
e) To generalize the verified procedures for further possible use.

The key tools for processing the practical part will be methods and techniques of deep learning, especially convolutional neural networks. The implementation will be carried out with the help of the Python programming language, and mainly the use of Keras and TensorFlow libraries. The proposed solution will take into account the defined requirements and expectations associated with solving the issue. Based on the synthesis of theoretical knowledge and achieved results, the conclusions of the master's thesis will be formulated and subsequently generalized for further possible use and applications.

Keywords: deep learning, image processing, classification, convolutional neural network, Egyptian hieroglyphs

Obsah

1 Úvod

V současné době slýcháme o umělé inteligenci téměř každý den. Zvláště s rozvojem veřejně dostupných modelů se toto téma stalo číslem jedna v „technologickém vývoji“, přestože se jedná již o „starší“ koncept, jehož základy byly položeny už v roce 1950 Alanem Turningem. Postupným vývojem tak umělá inteligence prostupuje naše životy, až se nakonec stane jejich nevyhnutelnou součástí. Mnohé modely se však logicky soustředí spíše na aktuální záležitosti, které jsou ryze praktické, neméně působivé je však jejich využití vzhledem k historii. Zajímavou oblastí pro využití umělé inteligence je tak sféra egyptských hieroglyfů.

V této diplomové práci se budu soustředit na vytvoření modelu, který dokáže rozpoznávat jednotlivé hieroglyfy. Cílem je vyvinout takový model, který s maximální přesností správně určí, o který ze základních hieroglyfů se jedná. Pro pochopení celého vývoje modelu je nutné přiblížit z teoretického hlediska užité nástroje či jednotlivé kroky. Část Teoretická východiska tak obsahuje pojednání o umělé inteligenci obecně a o jednotlivých vrstvách, které jsou u těchto modelů typické. Vzhledem k tématu práce si zaslouží rozsáhlejší zpracování oblasti měření a vyhodnocování modelů. Podstatnou součástí všech modelů jsou také data a datasety, které de facto ovlivňují jejich celkovou kvalitu. Pro pochopení reálné podstaty konvolučních sítí je také vhodné popsat jejich vývoj, který je důležitý pro nastínění jejich aktuálních možností. Konkrétní aplikace CNN pak dokreslí reálný význam této technologie napříč vybranými sektory. Zcela klíčovou kapitolou je teoretické představení přímo použitého hardwaru a softwaru, který byl využíván v souvislosti s touto prací.

Vlastní práce je strukturována dle logických kroků zachycujících jednotlivé etapy vývoje modelu. Prvním krokem je představení vybraného datasetu a příprava dat k jejich využití. Následující pasáž se věnuje samotné tvorbě modelu, jejíž součástí je také tzv. trénování modelu. Další část se věnuje optimalizaci modelu, která obsahuje dílčí podkapitoly. Počáteční podkapitoly se věnují modelu 1 až 4, následně je popsána architektura modelu 4 a jeho trénink. Logickým pokračováním jsou podkapitoly týkající se metrik hodnocení modelu a jeho evaluaci. Závěrečnou podkapitolou je Matice záměn neboli tzv. confusion matrix. Navazující kapitolou je ověření funkčnosti predikce modelu na testovacích datech.

Přínosem této práce je propojení historie s moderní technologií na jedné straně a na druhé stráně představení další sféry pro využití umělé inteligence, která doposud není dostatečně odkryta, na straně druhé. Domnívám se, že vývoj daného modelu může pozitivně přispět k barvitosti spektra všech modelů umělé inteligence a napomoci rozšiřování obzorů tohoto technologického nástroje.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem diplomové práce je demonstrovat funkčnost a efektivitu klasifikačního modelu vytvořeného pro rozpoznání egyptských hieroglyfů. Představovaný model bude navržen na základě konvoluční neuronové sítě (dále jen „CNN“) a vytvořen pomocí knihoven Keras a TensorFlow za pomocí jazyka Python. Pro účely tréninku a testování modelu bude nalezena vhodná a specifická datová sada egyptských hieroglyfů.

Dílčími stěžejními kroky práce budou:

1. Výběr a příprava datové sady egyptských hieroglyfů,
2. návrh a implementace modelu pomocí CNN a knihoven Keras a TenserFlow,
3. trénink modelu a jeho postupné úpravy pro zlepšení funkčnosti,
4. testování finálního modelu.

Tato práce si též klade za cíl ověřit do jaké míry a kvality je navržený klasifikační model schopen efektivně rozpoznávat egyptské hieroglyfy a jak může přispět skrze zjištěné poznatky k lepšímu porozumění oblastem zpracování obrazu a strojového učení s využitím moderních technologií.

2.2 Metodika

Metodika použitá při řešení praktické části obsahuje několik částí, které lze poměrně snadno rozdělit do ucelených kroků, neboť i samotná podstata práce spočívá v několika na sebe navazujících stadiích:

- Vyhledání relevantních zdrojů vhodných ke zvolené problematice,
- studium dostupných odborných informačních zdrojů v oblasti hlubokého učení, konvolučních sítí a klasifikačních problémů,
- nalezení vhodného datového souboru pro trénování multiklasifikačního konvolučního modelu,
- analýza náležitostí obsahu datové sady a jejich charakteristik,
- upravení dat tak, aby byla vhodná pro multiklasifikační konvoluční model,

- programování a realizace prototypu modelu pro identifikaci a klasifikaci egyptských hieroglyfů,
- analýza experimentů a optimalizace modelu za pomocí vizualizace maticí záměn, metrik ztrátové funkce a hodnoty přesnosti identifikace egyptských hieroglyfů,
- evaluace predikce finálního modelu na testovacích datech.

3 Teoretická východiska

3.1 Umělá inteligence

Umělá inteligence, označována jinak také zkratkou AI – Artificial Intelligence (dále jen „AI“), je počítačový program, který je při vhodném nastavení schopen řešit operace a úkoly, které by ze své podstaty vyžadovaly použití lidské inteligence. Její užívání je spojeno nejen s prognózami v rozličných oblastech, ale také s optimalizací všeprostupující automatizace, a v neposlední řadě také s řešením nesčetného množství složitých úkolů, které vyžadující značné nasazení zainteresovaných osob. Zdůrazňuji, že v případě AI se jedná o nejširší termín, který lze jistě považovat za zastřešující pro mnoho souvisejících termínů. Pojem AI je znám široké veřejnosti, která ovšem mnohdy nerozlišuje další související pojmy a často je zaměňuje, přestože z povahy věci se nejedná o synonymní výrazy. Příznačné chybné zaměňování pojmů se velmi často objevuje v situacích, ve kterých se hovoří o strojovém učení a hlubokém učení, avšak tyto termíny jsou podmnožinou AI a nelze je tedy s tímto pojmem zaměňovat. (1)

3.1.1 Strojové učení

Strojové učení je součástí AI, jako takové představuje vytváření algoritmů a modelů, které mimo zpracovávání dat mají také schopnost se z těchto dat učit a zdokonalovat se, aniž by vyžadovaly provedení doplňujícího programování. Tato konkrétní schopnost umožňuje systémům založeným na strojovém učení dosáhnout adaptace na nová data a nové situace.

V oblasti strojového učení existuje široké spektrum technik, které disponují konkrétními metodikami zpracování. Mezi ně patří zejména rozhodovací stromy, které slouží k rozhodování založeném na hierarchické struktuře sestávající z jednotlivých podmínek a pravidel. Dále také lineární regrese, která představuje techniku modelování, jejímž účelem je definování vztahu mezi individuálními proměnnými, aby v závislosti na zjištěném vztahu umožnila následnou predikci nových hodnot. Za neopomenutelný prvek strojového učení je nutno považovat klasifikaci, která v sobě obsahuje funkci přiřazování objektů či dat do určitých tříd v závislosti na jejich charakteristikách.

Strojové učení má široké uplatnění v mnohých vědeckých či technických oblastech, které jsou častokrát zcela odlišné, ať už se jedná o oblast analýzy dat, užití při rozpoznávání vzorů; stále častější je v automatizaci rozhodovacích procesů nebo predikci budoucích

událostí. Ze své podstaty je klíčovým prvkem vývoje pokročilých AI systémů a zastává nepominutelnou roli při moderním zpracovávání dat a vytváření automatizace.

3.1.2 Hluboké učení

Za jádro pojmu hlubokého učení lze označit specifickou sféru strojového učení, jež se zaměřuje na vytváření neuronových sítí, které jsou typické velkým počtem vrstev. Lze tvrdit, že algoritmy, které jsou uzpůsobeny pro řešení konkrétní problematiky, jsou modelovány obdobným způsobem jako spojení mezi neurony v lidském mozku. Hlavním prvkem úspěšného hlubokého učení je nepopiratelně trénink modelů na rozsáhlých souborech dat, které umožňují konfiguraci propojení mezi různými vrstvami neuronů. Mezi přednosti modelů hlubokého učení patří schopnost analyzovat a zpracovávat data v reálném čase, což ovšem na druhé straně vyžaduje vysoký výpočetní výkon. Z toho důvodu je vhodné trénovat tyto modely na zařízeních, které disponují výkonnými grafickými procesory (GPU).

Hluboké učení nachází své využití v leckterých oborech, pro ilustraci lze zmínit například oblast rozpoznávání obrazů, zpracování přirozeného jazyka a řadu mnohých jiných úloh, skrze které je zajištěna potřeba zpracovávat a interpretovat velké a složitě datové soubory. (2)

3.1.3 CNN

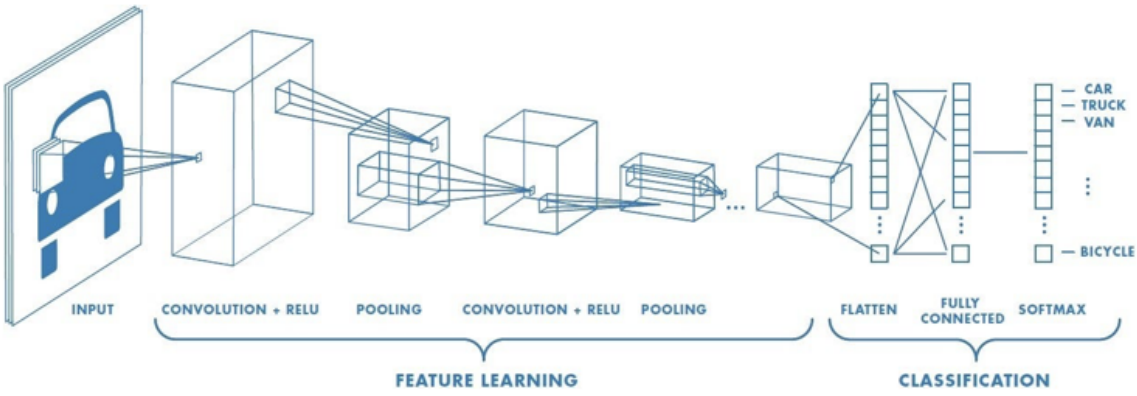
Integrální součástí hlubokého učení je konvoluční neuronová síť, dále jen CNN. Konvoluce je matematická operace v kontextu zpracování obrazu, konvoluce zahrnuje aplikaci tzv. konvolučního jádra (nebo filtru) na obrázek. Toto jádro je malá matice hodnot, která se „posouvá“ po celém obrázku. V každém kroku se hodnoty v jádru násobí hodnotami odpovídajících pixelů na obrázku a výsledky se sčítají. Tím se zformuje nový obrázek, který může zdůraznit určité vlastnosti původního obrázku, jako jsou hrany, rohy nebo textury.

Konvoluční neuronová síť je založena na podobné podstatě jako strojové učení, jelikož je také tvořena mnoha neurony, kterým jsou přiřazeny určité váhy. Je nutné poznamenat, že každý neuron zahrnutý v CNN přijímá vstup a provádí skalární součin, přičemž proces spočívá v užití libovolně nelineární cesty. Ve skutečnosti celá CNN představuje do určité míry jediný blok diferencovatelných skórovacích funkcí, které spojují surový obraz na jedné straně a hodnocenou třídu na straně druhé. (3)

CNN dosahují své specifičnosti použitím konvolučních vrstev a pooling vrstev, které jsou schopny efektivně zpracovávat vstupní data o velkých rozměrech a také vytvářet plně

propojené vrstvy. Majoritně se tyto modely vytváří se specializací pro rozpoznání obrazu a pro úlohy se zaměřením na dispozici s pixelovými daty. Dalším typickým znakem, kterým se také CNN vyznačuje, jsou bezpochyby ztrátové funkce a optimalizátory, které definují celý proces učení, tj. určují konkrétní povahu v jeho průběhu. (3)

Obrázek 1 – Proces CNN



Zdroj: https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/

3.2 Vrstvy

V úvodu do struktury CNN je nezbytné zdůraznit, že tyto modely jsou charakterizovány hierarchickým uspořádáním různých typů vrstev. Každá z těchto vrstev plní specifickou a nezastupitelnou roli v procesu zpracování vstupních dat. Správně umístění je poté klíčové pro identifikaci vzorců v datech. Nehledě na to, že vrstvy jsou vzájemně propojeny, což umožňuje interakci mezi nimi a přispívá ke zlepšování celého procesu učení modelu. V následujících podkapitolách budou jednotlivé vrstvy CNN podrobněji popsány a analyzovány s cílem poskytnou hlubší pochopení jejich funkcí a významu v rámci celkové architektury modelu. (4)

3.2.1 Konvoluční vrstvy

Hovoří-li se o konvolučních vrstvách, je nutné je označit jako stěžejní stavební kámen CNN. Jak již název napovídá, tyto konkrétní vrstvy zabezpečují výkon procesu konvoluce operací nad vstupními daty a poté předávají výsledek další vrstvě. Každou konvoluční vrstvu lze definovat počtem filtrů, u kterých je sledována jejich velikost. Typickou hodnotou pro velikost filtru je matice s rozměrem 3x3. Konvoluční vrstva dále definuje počet konvolučních filtrů, které stanovují jaké vzory a rysy jsou v datech hledány. Nelze též opomenout aktivační funkci, která určuje neutralitu konvoluční vrstvy.

Během procesu konvoluce jsou filtry aplikovány na různé segmenty vstupních dat, kde detekují širokou škálu vizuálních vzorců a charakteristiky. Výsledkem jsou poté konvoluční mapy, které zachycují specifické odezvy filtrů na různé části vstupních dat. Mapy mohou být následně zpracovány nebo modifikovány v dalších vrstvách. Tento postup je v praxi běžný, jelikož umožňuje modelu identifikovat a učit se složitější vzorce v datech a tím přispívat k jeho celkové efektivitě a přesnosti. (5)

Vstupní vrstva konvoluce obsahuje třírozměrnou strukturu, která představuje rozměr a hloubku korespondující s počtem rysů užitých ve vstupních dat. Zjištěné znaky odpovídají ve své hloubce barevným kanálům RGB (červená, zelená, modrá). Navazující vrstvy v konvoluční síti v sobě také zahrnují tuto hloubku, avšak mohou dosahovat různého prostorového rozsahu. Díky tomu se model flexibilně přizpůsobuje různým typům dat a úlohám.

Jedním z nejčastěji používaných typů konvolučních vrstev je Conv2D. Jeho charakteristikou je schopnost integrovat všechny výše zmíněné parametry, což ji činí univerzálním nástrojem pro řadu aplikací v rámci CNN. Právě pro svoji všestrannost je vrstva Conv2D často využívána v mnoha moderních modelech zaměřených právě na aplikaci obrazové klasifikace nebo na detekci objektů, což jen potvrzuje její zásadní význam v kontextu hloubkového učení. (5)

3.2.2 Pooling vrstvy

V návaznosti na výše zmíněné, je třeba objasnit i druhou nedílnou vrstvu konvolučního modelu, kterou představují pooling vrstvy. Ta postupně prochází skupinu výstupů z předchozí vrstvy, přičemž tyto výstupy jsou častokrát nastaveny na velikost 2x2. Kýženým cílem zmíněného postupu je redukce velikosti dat tak, že proběhne výběr důležitých znaků z konkrétně vybrané oblasti a ty se následně nahradí pouze jedním prvkem. (6)

3.2.2.1 Max pooling

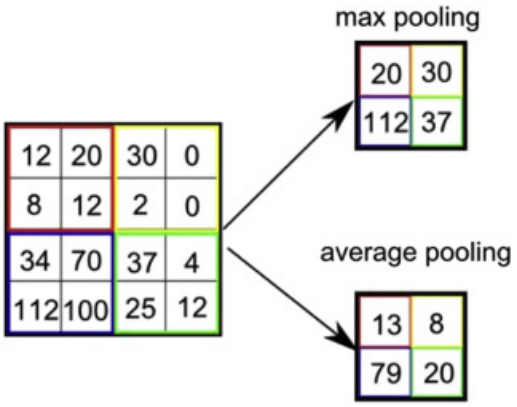
Technika max pooling používaná v konvolučních neuronových sítích sloužící k redukci rozměrnosti dat funguje tak, že prochází oblasti vstupních dat. Typicky, jak už bylo řečeno, o velikosti 2x2. Z každé takové oblasti poté vybere maximální hodnotu. Tímto způsobem se zmenšuje množství dat, které je třeba zpracovat, zatímco se zachovávají nejdůležitější informace.

Max pooling je zvláště efektivní při extrakci výrazných rysů, jako jsou například hrany v obrázcích. Umožňuje neuronové síti zaměření se na nejdůležitější rysy a tím pádem ignorovat ty méně důležité detaily. Takto se zajistí lepší výkonosti a přesnost rozpoznání výsledného modelu. (6)

3.2.2.2 Average pooling

Average pooling je také používána jako technika v CNN pro redukci rozměrnosti dat. Na rozdíl od předchozí metody (max pooling) nevybírá největší hodnotu, ale ze zadané oblasti počítá průměrnou hodnotu. Od max pooling, který se zaměřuje na extrakci nejvýraznějších rysů, se liší tím, že umožní hladší a méně nápadnou extrakci rysů. Použití této techniky může být výhodné v situacích, kdy je důležité zachovat více detailů v datech nebo pokud je potřeba snížit šum v datech. Na konkrétní aplikaci a na povaze dat závisí skutečnost, zda bude vybrána technika max pooling a average pooling. (6)

Obrázek 2 – Proces fungování average a max poolingu



Zdroj: https://www.sciencedirect.com/topics/mathematics/pooling-layer

3.2.3 Plně propojené vrstvy

V rámci architektury konvolučních neuronových sítí představuje plně propojená vrstva integrální a nezbytnou složku. Charakterizují ji dva hlavní prvky. První z nich je aktivační funkce Rectified Linear Unit (ReLu), její aplikace je v tomto kontextu typická. Druhým prvkem je softmaxová funkce, která je rovněž běžně využívána. Význam plně propojené vrstvy se projevuje v konečné fázi procesu, kdy dochází k interpretaci dat. Finální výstup vrstvy je prezentován ve formě procentuální přesnosti rozpoznání dat, které tak umožňuje kvantifikovat úspěšnost modelu v rámci jeho schopnosti správně klasifikovat vstupní data.

3.2.4 Dropout vrstvy

Široce používanou regulační technikou je Dropout. Metoda je založena na principu náhodného vypínaní neuronů v dané vrstvě během trénování modelu. Tímto způsobem se v každém kroku učení formuje mnoho různých „verzí" modelu, což zvyšuje jeho robustnost a schopnost generalizovat nová data. Dropout je efektivním nástrojem pro boj s přeučením, což je stav, kdy si model příliš dobře zapamatuje trénovací data a má problémy s předpovědí nových, dosud neviděných dat. Zabraňuje tak přílišné adaptaci na trénovací data a podporuje schopnost se efektivně učit. (4)

3.2.5 Aktivační funkce

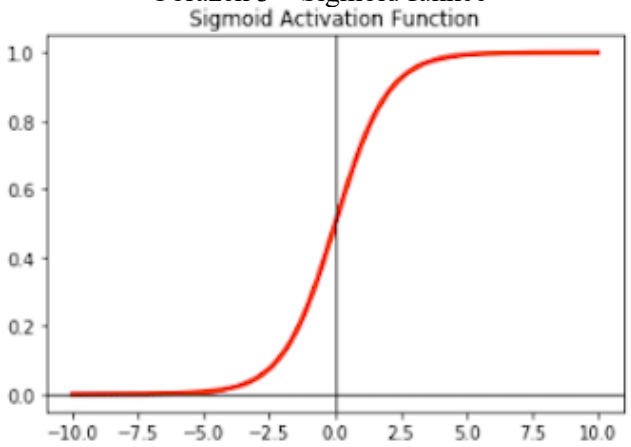
Účelem aktivačních funkcí je vnést do modelů nelineární vztah mezi vstupy a výstupy. Rozhodují, které neurony budou aktivovány a které nikoliv. Funkce se aplikují na různých vrstvách sítě za účelem vytvoření složitějších vzorců a schopností modelovat různorodé a komplexní data.

3.2.5.1 Sigmoid

Nelineární funkce, která transformuje reálnou hodnotu do intervalu mezi 0 a 1. Matematicky je sigmoidní funkce definována jako
sigmoid(x) = 1 / (1 + exp(-x)). Kde x je vstupní hodnota.

V kontextu neuronových sítí se sigmoidní funkce často používá jako aktivační funkce ve výstupní vrstvě pro binárně klasifikační nebo regresní úlohy. (7)

Obrázek 3 – Sigmoid funkce



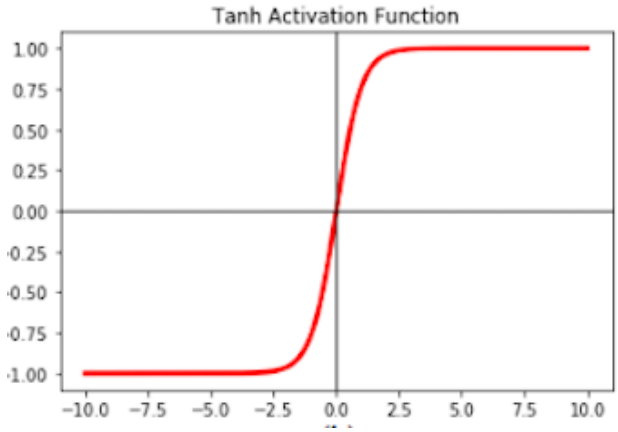
Zdroj: https://www.researchgate.net/figure/Plot-of-different-activation-functions-a-Sigmoid-activation-function-b-Tanh_fig4_339991922

3.2.5.2 Tanh

Hyperbolická tangent aktivační funkce transformuje vstupy mezi -1 a 1. Může být použita jako alternativa k sigmoid funkci zejména tam, kde je vhodné zachovat symetrický rozsah kolem nuly. (7)

tanh(x) = (exp(x) - exp(-x)) / (exp(x) + exp(-x)), kde x je vstupní hodnota.

Obrázek 4 – Tanh funkce



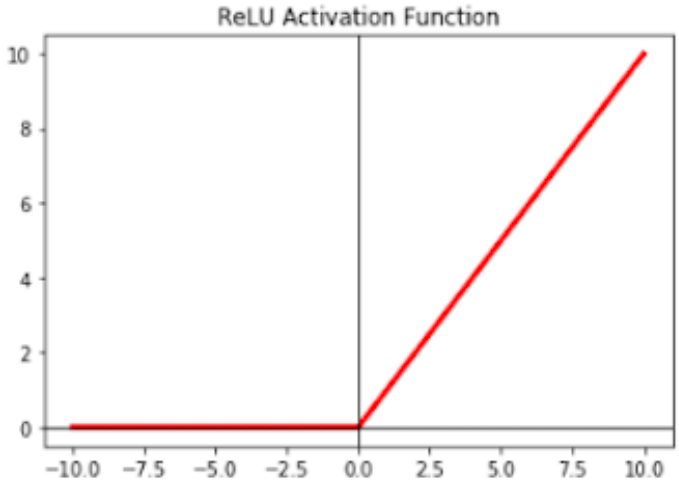
Zdroj: https://www.researchgate.net/figure/Plot-of-different-activation-functions-a-Sigmoid-activation-function-b-Tanh_fig4_339991922

3.2.5.3 ReLu

Tato aktivační funkce definuje nulu pro všechny negativní vstupy. Naproti tomu pro kladné vstupy nabývá ostrých lineárních hodnot. (4)(7)

standard ReLU activation: max(x, 0). Kde x je vstupní hodnota.

Obrázek 5 – ReLu funkce



Zdroj: https://www.researchgate.net/figure/Plot-of-different-activation-functions-a-Sigmoid-activation-function-b-Tanh_fig4_339991922

3.2.6 Softmax

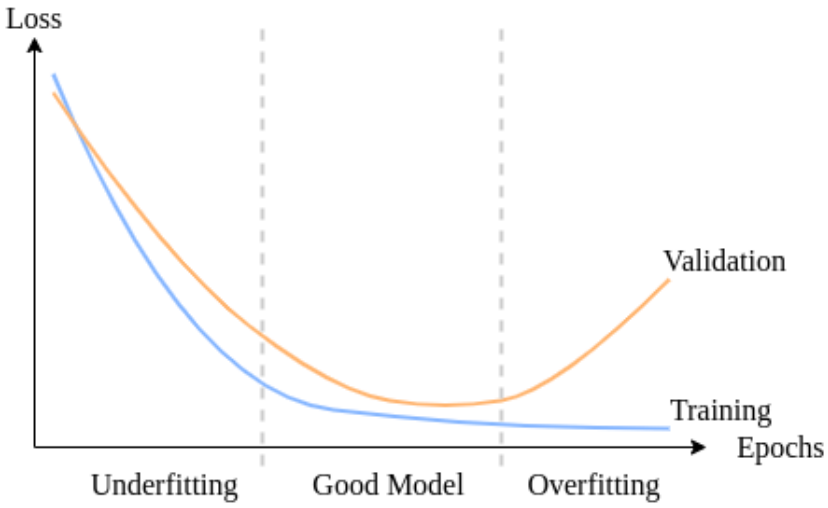
Softmax převádí hodnoty na pravděpodobnostní rozdělení. Prvky výstupního tensoru se nacházejí v intervalu (0,1) a jejich součet je roven jedné. Softmax je často používán jako aktivační funkce u poslední vrstvy multiklasifikačního modelu. (7)

3.3 Měření a vyhodnocení modelu

Důležitou složkou je monitorování a ověření průběhu učení modelu. Lze sledovat s jakou úspěšností se model naučil rozlišovat poskytnutá data, zda nedochází k nadměrné adaptaci dat nebo nedostatečné generalizaci. Nejefektivnějším výstupem vyhodnocení je vizualizace data, například do grafu, díky němuž se dá snadno sledovat a monitorovat vývoj učení. Výhodou je i jasná identifikace oblasti, která vyžaduje zlepšení a posléze je i důležitým faktorem pro jemné doladění modelu pro dosažení cílové optimální úspěšnosti modelu.

Velký význam vizualizace je zaměření na trénovací a validační data v rámci jednotlivých epoch. Důležité je sledovat hodnoty u ztrátové funkce a metriku přesnosti modelu. Tato pozorování mohou pomoci určit, zda se model nepřeučuje nebo naopak, zda dochází k nedotrénování. (8)

Obrázek 6 – Vyhodnocování průběhu učení modelu



Zdroj: https://www.baeldung.com/cs/ml-underfitting-overfitting

3.3.1 Overfitting

Běžným problémem, který při učení modelu nastává, je přizpůsobení modelu na tréninková data a ztráta schopnosti generalizovat nová, neviděná data. Pokud je model přetrénován (overfit), ztrátová funkce na tréninkových datech bude postupně klesat, zatímco ztrátová funkce validačních dat se bude zvyšovat. Existuje několik možností, jak se vypořádat s overfittingem. Jednou z nich je Early Stopping, který zastaví trénování modelu. Další z příčin může být nedostatek tréninkových dat. Pokud jde v konkrétním řešeném problému o práci s obrazovými daty, lze dosáhnout umělého rozšíření existujícího souboru pomocí augmentaci, která zajistí náhodné otáčení, přibližování nebo ořez dat. (8) (9)

3.3.2 Underfitting

Underfitting neboli nedotrénování modelu se projevuje vysokou ztrátovou funkci jak na validačních, tak trénovacích datech. Indikuje tak nedostatečné zachycení vzorů v datech. Jednou z příčin může být příliš jednoduchý model, který není schopen zachytit složitost struktury daného datasetu nebo, stejně jako u overfittingu, pokud není dostatečně velký soubor vstupních dat. Opravou problému může být přidání více parametrů do modelu nebo mohou být poskytnuty lepší vlastnosti učícímu algoritmu. (8) (9)

3.3.3 Early Stopping

Technika, která se používá při trénování modelu jako prevence přeučení, je Early Stopping. Ten monitoruje proces během učení na validačních datech a zastavuje trénink modelu podle určené sledované hodnoty. Tou mohou být přesnost nebo ztrátová funkce. Pokud se výkon modelu na validačních datech během jednotlivých epoch nezlepšuje, ukončí proces trénování a tím zamezí přetrénování. Je však nutné najít optimální bod, kdy se proces tréninku zastaví, aby naopak nedošlo k nedotrénování modelu. (9)

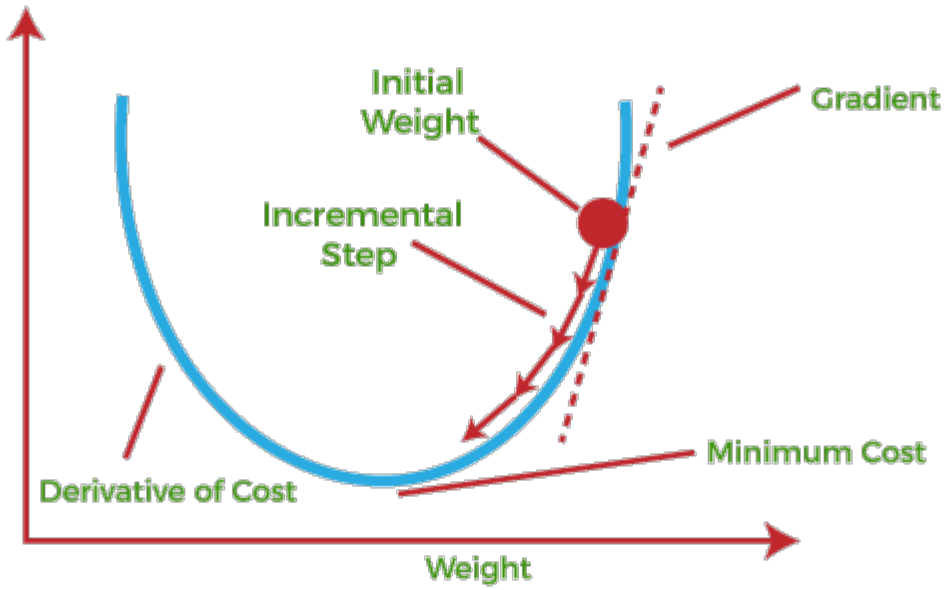
3.3.4 Optimalizátor a ztrátová funkce

Ztrátová funkce je matematická metrika, která kvantifikuje rozdíl mezi předpovězenými hodnotami modelu a skutečnými hodnotami na tréninkových datech. Jednoduše řečeno, tato matematická metoda vyhodnocuje, jak dobře algoritmus modelu predikuje datovou sadu. Během trénování modelu se vyhodnocuje pro každou epochu. Cílem je dosažení co nejnižší hodnoty. Čím menší je hodnota ztrátové funkce, tím lépe se data generalizují na nová data. Podle typu úlohy se v modelu používají různé typy ztrátové

funkce. V regresním modelu by mohla být použitá metoda nejmenších čtverců (MSE) nebo průměrná absolutní chyba (MAE). Zatímco v klasifikaci je možné použít Cross-entropy. (10)

Ruku v ruce se ztrátovou funkcí jde gradientní sestup. Ten je známý jako jeden s nejčastější optimalizačních algoritmů. Jeho hlavním cílem je minimalizovat konvexní funkci tak, že iterativně upravuje parametry modelu ve směru největšího poklesu. Nejdříve je třeba vysvětlit, co je gradient. Jde o vektor, jehož směr ukazuje nejrychlejšího nárůstu funkce, zatímco jeho opačný směr ukazuje největší pokles. Algoritmus gradientního sestupu zahájí iniciaci náhodných parametrů modelu. V každém kroku se vypočítá gradient ztrátové funkce pro aktuální hodnoty parametrů. Tento gradient určuje směr a velikost změny parametru tak, aby ztrátová funkce byla co nejmenší. Proces se opakuje, dokud není dosaženo určitého bodu zastavení nebo definovaného kritéria. (11)

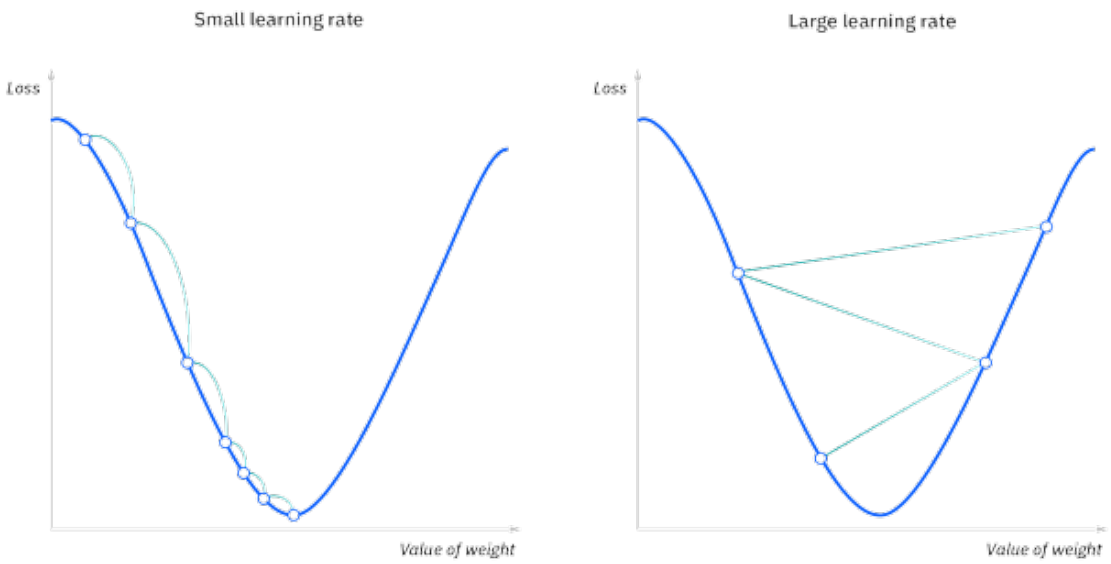
Obrázek 7 – Fungování gradient descent



Zdroj: https://www.javatpoint.com/gradient-descent-in-machine-learning

Nalezení minimálního bodu funkce lze ovlivnit hyperparametrem tzv. learning rate. To je velikost kroků potřebných k dosažení minimální hodnoty. Obvykle je tato hodnota malá a vyhodnocuje se a aktualizuje podle chování ztrátové funkce. Vysoká hodnota learning rate vede sice k velkým krokům, ale s rizikem překročení minimální hodnoty. Naopak nízká hodnota bude mít lepší a přesnější výsledek, ale bude časově a výpočtově náročnější, čímž poklesne celková efektivita. (12)

Obrázek 8 – Průběh nastaveného learning ratu



Zdroj: https://www.ibm.com/topics/gradient-descent

3.3.5 Metriky hodnocení

Metriky jsou dalším důležitým nástrojem pro zjištění kvality modelu. Objektivně poskytují měření toho, jak model funguje. Zároveň umožňují porovnávat různé modely a techniky, a to jak při trénování, tak i při testování datového souboru.

Různé typy úloh a modelů vyžadují charakteristické metriky, které odrážejí požadované vlastnosti dat a modelů. U klasifikačních modelů je obvyklé použití metriky jako je accuracy, precision, recall nebo F1 skóre. Tyto metriky pomáhají pochopit, do jaké míry model dobře klasifikuje různé třídy.

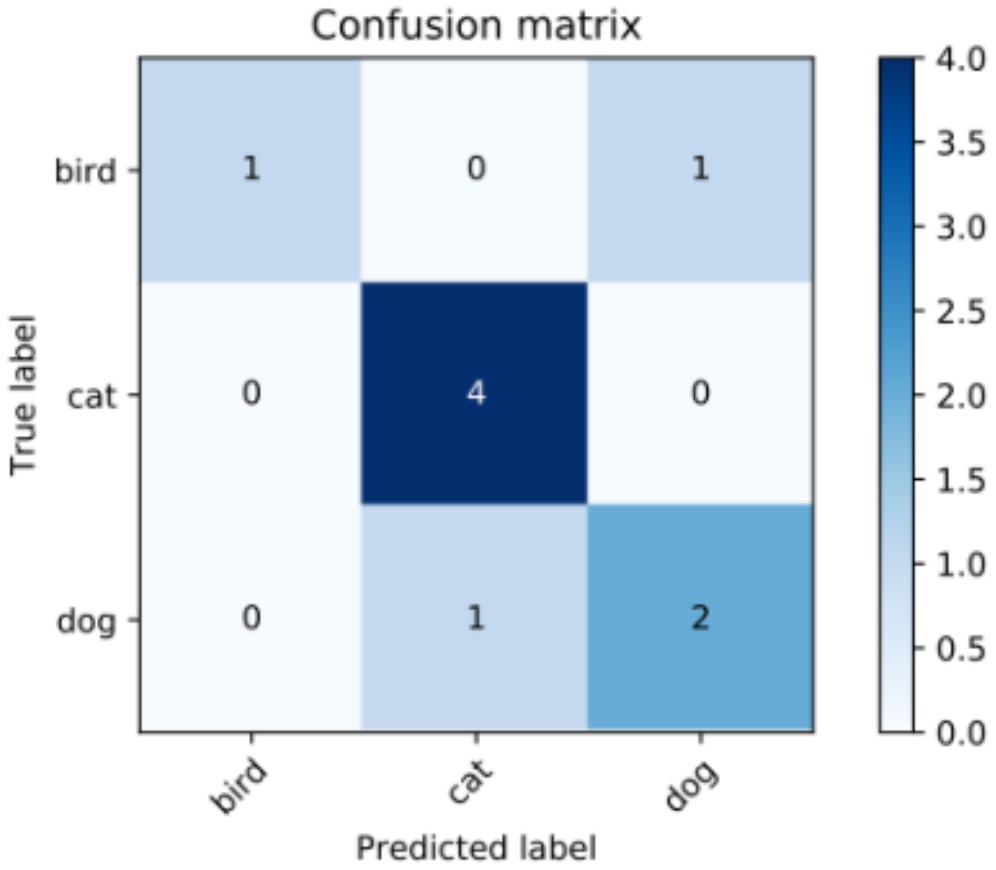
Naopak pro regresní modely (stejné jako u ztrátové funkce) je typické použití průměrné absolutní chyby (MAE) nebo metody nejmenších čtverců (MSE). Tyto metriky měří odchylky mezi předpovězenými a skutečnými hodnotami. Nutno dodat, že žádný ukazatel není univerzální. Správná volba výběru metriky závisí na tom, k jakému účelu je vytvořen model a jakého cíle má dosáhnout. (13) (14)

3.3.6 Confusion matrix

Confusion matrix neboli matice záměn je velmi užitečná při analýze klasifikačních modelů. Pomáhá identifikovat, jak model klasifikoval jednotlivé třídy. Za pomocí vizuálního zobrazení lze jasně určit, ve kterých třídách model dosahuje nejlepších výsledků, a kde jsou naopak výsledky nejhorší.

Na diagonále matice se nachází správně predikované hodnoty. Od diagonály nalevo jsou falešně pozitivní hodnoty, tedy příklady, které byly chybně klasifikovány jako pozitivní. Na pravé straně od diagonály se nacházejí falešně negativní predikce. (15) (16)

Obrázek 9 – Confusion matrix



Zdroj: https://www.mariakhalusova.com/posts/2019-04-17-ml-model-evaluation-metrics-p2/

3.3.7 Základní rozdělení modelů v neuronových sítích

Neuronové sítě představují základní stavební kámen pro mnoho moderních aplikací strojového učení. Modely se dokáží naučit složité vzorce v datech, čímž dokáží po natrénování rozpoznat a klasifikovat neznámé nebo neviděné vzorky. Podle typu úlohy jde modely neuronových sítí rozdělit do tří skupin, konkrétně regresní, binární klasifikační a multiklasifikační. Každý z vyjmenovaných typů modelů má své specifické využití a liší se i přistup k jejich trénování a evaluaci.

3.3.7.1 Regresní modely

Regresní model je statistická metoda, která zjišťuje vztah mezi závislou proměnnou a nezávislou proměnou. Regresivní analýza pomáhá zejména pochopit, jak se mění hodnota závislé proměnné (vysvětlovaná hodnota) na nezávislých proměnných (vysvětlující hodnoty). Pomáhá předpovídat skutečné hodnoty a hledá trendy v datech. Reálné použití regresního modelu může při hledání ceny nemovitostí, věku dožití nebo hodnoty Bitcoinu. V rámci strojového učení je na výstupní vrstvě použit právě jeden neuron, což je výsledná predikovaná hodnota. (17)

Následující tabulka poskytující ucelený pohled na klíčové aspekty architektury regresního modelu, které lze přizpůsobit konkrétním potřebám a povaze problému. (18)

Tabulka 1 – Typické parametry regresního modelu

Hyperparametr	**Typické hodnoty**
Tvar (shape) vstupní vrstvy	Jedna hodnota pro každou funkci
Skryté vrstvy	Záleží na specifikaci problému; min =1 a max je neomezené
Neurony ve skrytých vrstvách	Záleží na specifikaci problému, zpravidla 10 až 100
Tvar výstupní vrstvy	Tvar je stejný jako požadovaný tvar předpovědi
Aktivační funkce skryté vrstvy	Obvykle ReLu
Výstupní aktivační funkce	Žádná, ReLu nebo Tanh
Ztrátová funkce	Metoda nejmenších čtverců nebo průměrná absolutní chyba
Optimalizátor	SGD (Stochastic Gradient Descent), Adam

Zdroj: Geron Aurelien, Hand on machine learning with Scikit-Learn (2017)

3.3.7.2 Binární a multiklasifikační modely

Binární klasifikace se používá v modelech, jejichž hlavním účelem je klasifikovat data do dvou tříd nebo kategorií. Tento přístup může být použit buď při detekci spamu, nebo při klasifikaci obrazu, která se snaží určit přítomnost či nepřítomnosti určitého objektu v datech.

Přístup multiklasfikačního modelu je aplikován, v případě, že v datovém setu existuje více jak dvě třídy, které je potřeba detekovat. Zde nejde jen o jednoduché rozdělení na ano či ne (binární klasifikace), ale o přiřazení vstupních dat právě jedné kategorii. Multiklasifikace je využita v úlohách, jako je rozpoznání psaných čísel, detekce nálady podle obličeje, rozřazení textů do různých žánrů, anebo právě rozpoznání egyptských hieroglyfů.

Níže zobrazená tabulka poskytuje komplexní přehled o klíčových aspektech architektury modelu pro binární a multiklasifikační klasifikaci, jednotlivé parametry musí být upraveny dle konkrétních požadavků a charakteru řešeného problému. (18)

Tabulka 2 – Typické parametry klasifikačních modelů

Hyperparametr	**Typické hodnoty – Binární Klasifikace**	**Multiklasifikace**
Tvar (shape) vstupní vrstvy	Stejný jako počet funkcí	Stejné jako u binární
Skryté vrstvy	Záleží na specifikaci problému; min = 1, max je neomezené	Stejné jako u binární
Neurony ve skrytých vrstvách	Záleží na specifikaci problému, zpravidla 10 až 100	Stejné jako u binární
Tvar výstupní vrstvy	Tvar je jedna (jedna třída nebo druhá)	Jedna pro každou třídu
Aktivační funkce skryté vrstvy	Obvykle ReLu	Stejné jako u binární
Výstupní aktivační funkce	Sigmoid	Softmax
Ztrátová funkce	Cross entropy	CateforicalCrossEntropy
Optimalizátor	SGD, Adam	Stejné jako u binární

Zdroj: Geron Aurelien, Hand on machine learning with Scikit-Learn (2017)

3.4 Data a jejich úprava

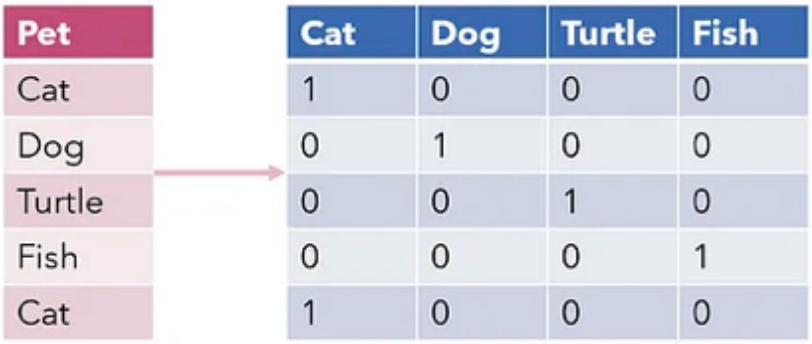
Příprava a zpracování dat je jedním z nejdůležitějších kroků při tvorbě a trénování modelu. Kvalita a přesnost dat hrají důležitou roli pro úspěch správného trénování modelu. Na internetu je k dispozici mnoho veřejně dostupných datových sad. Ty lze využít jak pro začátečníky, kteří se chtějí seznámit s umělou inteligenci, tak i v rámci zveřejněných soutěží, kde se soutěžící snaží sestavit co nejlepšího model. Pokud neexistuje datová sada, která by vyhovovala konkrétnímu modelu, je třeba shromáždit data tak, aby byly relevantní a odpovídaly cíli modelu. Data je třeba očistit, nahradit nebo ošetřit chybějící hodnoty a odstranit neplatné záznamy.

V případě, že nejsou ve vhodném formátu, je třeba je transformovat. Tento proces zahrnuje různé techniky a kroky odvíjející se od potřeb modelu a jeho cíle. Běžně používanou transformací je normalizace dat. Ta zajišťuje, že hodnoty v datové sadě mají interval hodnot mezi 0 a 1, což napomáhá a zjednodušuje model učení a trénování na datech. Dalším důležitým procesem je škálování, které zajistí, aby data byly ve stejném měřítku, což

je důležité při použití modelu na klasifikaci obrázků. Pro některé funkce je tento krok nezbytný, aby nedocházelo k dominanci některé z nich a zkreslení modelu.

Pokud jsou v datovém souboru nebo pro účely modelu textové štítky, které jsou zapotřebí k rozpoznání dat, je nutností provést převod z textové podoby na číselní formát, jinak není model schopen je zpracovat. Toho lze dosáhnout za pomocí one-hot kódováním, nebo jiné techniky jako je například indexace. (19)

Obrázek 10 – Příklad one-hot kódování



Pet	Cat	Dog	Turtle	Fish
Cat	1	0	0	0
Dog	0	1	0	0
Turtle	0	0	1	0
Fish	0	0	0	1
Cat	1	0	0	0

Zdroj: https://medium.com/analytics-vidhya/stop-one-hot-encoding-your-categorical-variables-bbb0fba89809

V mnoha případech je velikost datové sady nedostatečná a je třeba ji rozšířit. Především v úlohách pro zpracování obrazu se využívá metoda augmentace. Proces spočívá v použití různých augmentačních transformací jako je rotace, posun, zvětšení nebo zmenšení, avšak ponechává datům původní označení. K provedení augmentace dat je vhodné využit vestavěných funkcí Kerasu, jako je ImageDataGenerator, který zajistí větší rozmanitost trénovacích dat. (20)

Pro účely trénovaní a vyhodnocování modelu se datasety rozdělují na tři množiny: trénovací, validační a testovací. V ideálním případě by měly být modely hodnoceny na vzorcích, které nebyly použity při vývoji nebo ladění modelu, aby byl zajištěn nezkreslený odhad výkonnosti modelu.

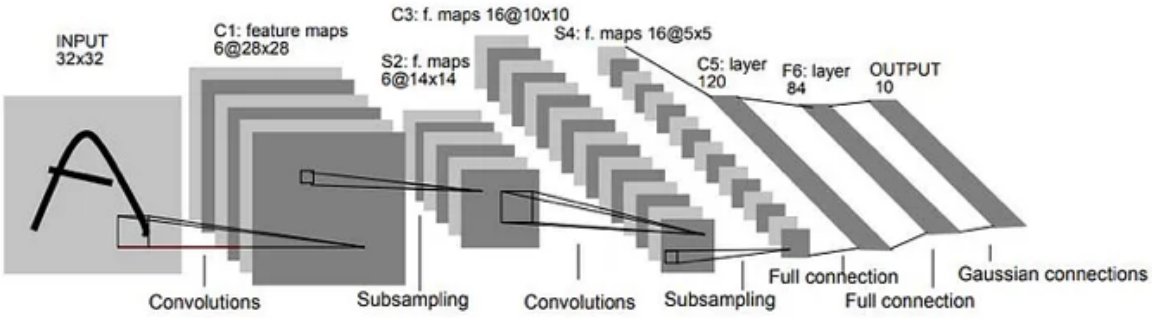
Trénovací datová sada slouží pro samotný trénink modelu. Validační sada se používá k odhadu, jak dobře model zobecňuje neznáma data. Zatímco testovací sada je oddělena od trénovacích a validačních dat a používá se až ke konečnému vyhodnocení výkonnosti modelu. Díky takovému rozdělení je model schopen správně předpovídat a odhadnout nová data, což je jeho hlavním cílem. (21)

3.5 Etapy vývoje konvolučních neuronových sítí

3.5.1 LeNet

LeNet je historicky první architekturou konvoluční neuronové sítě (CNN), která se specializovala na rozpoznávání ručně psaných čísel, a k tomu využívala slavný datový soubor MNIST. Tato architektura modelu je překvapivě jednoduchá, skládá se pouze z pěti vrstev, které obsahují 5 * 5 konvoluční a 2 * 2 maxpool vrstvy. I přes svou jednoduchost však LeNet položila základ pro vývoj sofistikovanějších a komplexnějších modelů v oblasti hlubokého učení. (22)

Obrázek 11 – Architektura LeNet



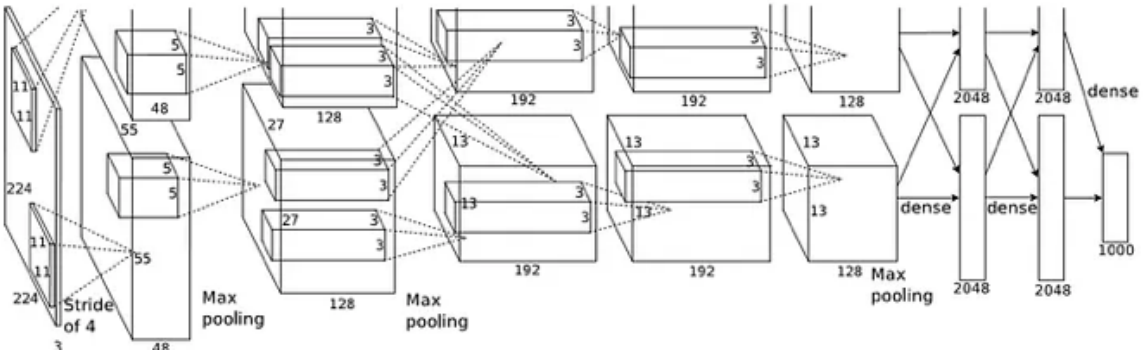
Zdroj: LeNet-5 - A Classic CNN Architecture - DataScienceCentral.com

3.5.2 AlexNet

AlexNet se dostal do povědomí díky úspěchu ve výzvě ImageNet (celoslosvětová soutěž v oblasti rozpoznávání obrazu). Šlo o první model, který byl založen na konvolučních neuronových sítích a tuto soutěž vyhrál. Oproti modelu LeNet byl výrazně složitější a obsahoval několik inovací.

Model obsahovat konvoluční vrstvy s různými velikostmi jádra, což umožnilo lépe zachytit různorodé rysy ve vstupních datech. Také používal aktivační funkce díky, kterým dosahoval mnohem lepších výsledku. Důležité je zmínit, že významným prvkem modelu bylo také jeho implementování a využití výkonu grafických karet (GPU). Touto inovací dokázal model rychle a efektivně zpracovávat velké množství dat, což se stalo jako zásadní pro jeho úspěch. (22)

Obrázek 12 – Architektura AlexNet



Zdroj: Architecture of AlexNet and its current use (opengenus.org)

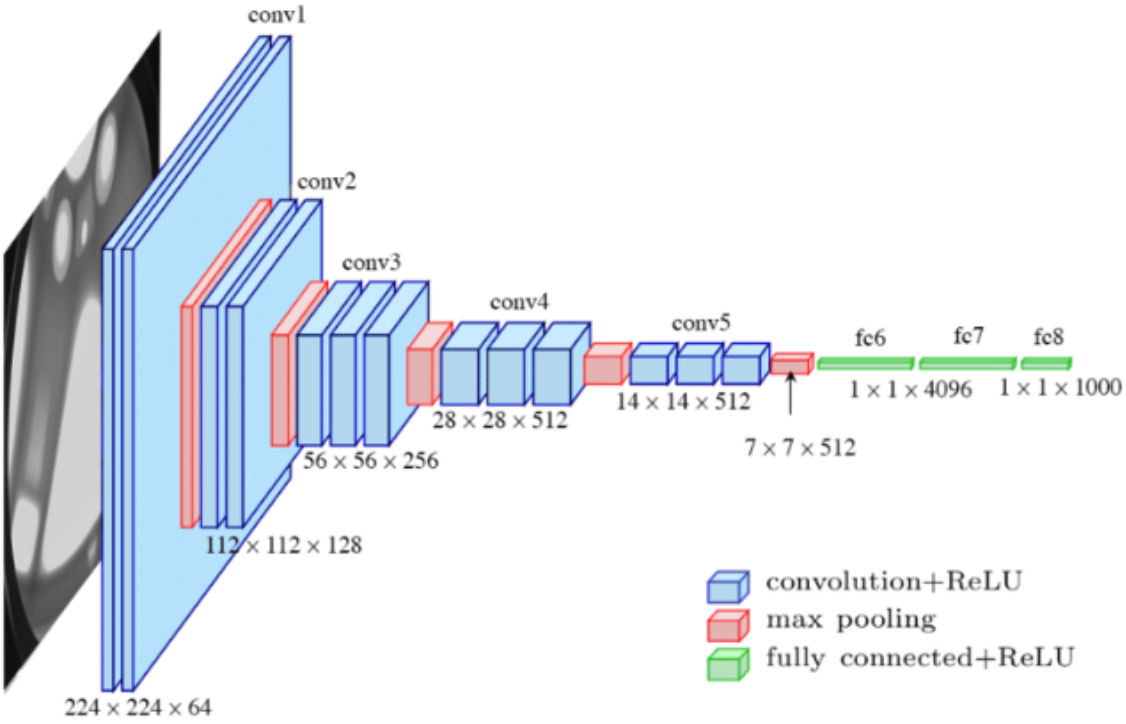
3.5.3 InceptionNet

Konstrukce modelu, také známá jako GoogLeNet, je jedním z pokročilých architektonických modelů konvolučních neuronových sítí vyvinutých firmou Google. InceptionNet byl zásadně hlubší a měl více parametrů než doposavad existující modely. Pro řešení problému tréninku takto hlubokých modelů byl zaveden koncept použití více vedlejších klasifikátorů, které byly umístěny uvnitř modelu. Tak se předešlo problému mizejícího gradientu. Jednou z hlavních myšlenek bylo použití paralelního zpracování jader různých velikostí, a tak modelu umožnit extrahovat různé prvky, ať už velké nebo malé, současně. (23)

3.5.4 VGG

Model VGG (Visual Geometry Group) je jedním z významných klasických konvolučních neuronových sítí vyvinutých pro účely počítačového vidění a rozpoznávání obrazů. Jedná se o významný a oblíbený přístup v oblasti klasifikace obrazů. VGG model se vyznačuje svou hloubkou. Konkrétně velmi úspěšná verze VGG16, kde se číslo 16 v názvu odkazuje na 16 vrstev, které obsahují váhy. Architektura VGG16 je charakterizována použitím malých konvolučních filtrů o velikosti 3x3 pixelů v celé síti s následným zdvojením počtu kanálů v konvoluční vrstvě po každé vrstvě s maxpoolingem. Model celkem zahrnuje 13 konvolučních, 5 maxpool a tři plně propojené vrstvy. Nicméně pouze 16 z nich má váhové parametry, které jsou naučitelné. Tímto způsobem se dosáhne velmi hlubokého modelu s relativně malými filtry. Nicméně verze VGG16 byla rozšířena na VGG19, která je hlubší a dosahuje vyšší přesnosti na úkor většího výpočetního výkonu a paměti. (24)

Obrázek 13 – Architektura VGG16



Zdroj: https://www.researchgate.net/figure/Fig-A1-The-standard-VGG-16-network-architecture-as-proposed-in-32-Note-that-only_fig3_322512435

3.5.5 MobileNet

MobilNet je typ mobilní architektury konvolučních sítí speciálně navržené pro výpočty na zařízeních s omezenými výpočetními zdroji jako jsou mobilní telefony a vestavěná zařízení. Architektura je optimalizována tak, aby modely dosahovaly vysoké přesnosti klasifikace obrazu při nízkém výpočetním a paměťovém nároku.

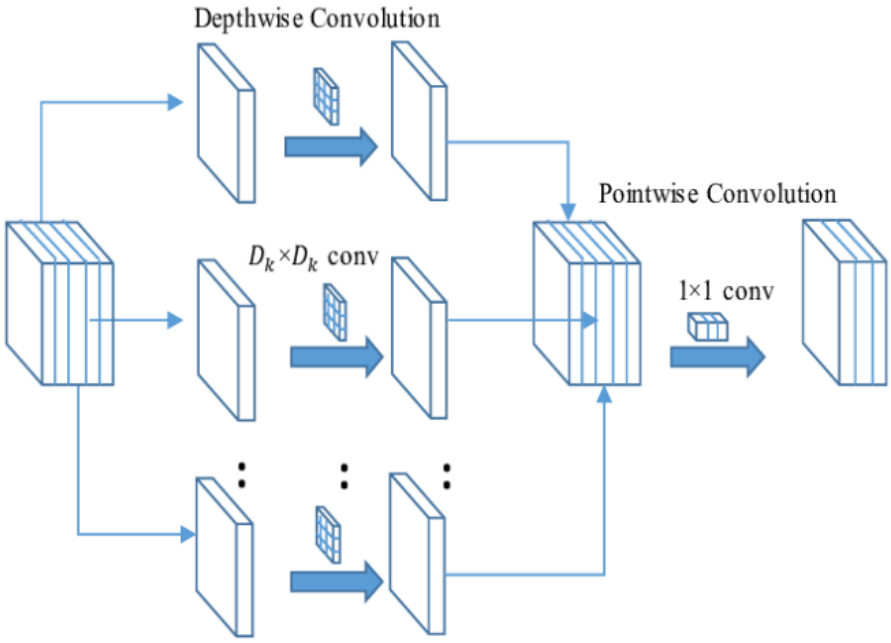
Klíčovou vlastností této architektury je hloubkově separovatelná konvoluce (Depthwise Seprable Convolution). Navržená technika rozkládá klasickou konvoluci na dvě části:

1. **Hloubková konvoluce**

   Tento díl modelu se zaměřuje na zachování hloubky dat. Zde se každý vstupní kanál filtruje samostatně. To znamená, že každý kanál prochází svým vlastním filtrem.

2. **Bodová konvoluce (konvoluce 1 × 1)**: Po provedení hloubkové konvoluce je použita konvoluce s velikostí 1 × 1, která spojí výstupy z předchozí části a zkombinuje je na nové výstupy. (25)

Obrázek 14 – Příklad hloubkové konvoluce



Zdroj: https://www.researchgate.net/figure/Depthwise-separable-convolution-block_fig1_343943234

3.5.6 EfficientNet

EfficientNet využívá koncept zvaný složené škálování, které hledá řešení v tradičním dilematu mezi velikostí modelu, přesností a efektivitou výpočtu. Ideou složeného škálování je upravit a optimalizovat tří klíčové parametry neuronové sítě: šířku, hloubku a rozlišení.

Koncept škálování šířky se zabývá manipulací s počtem kanálů v každé vrstvě neuronové sítě. Zvýšení šířky modelu umožňuje sítím lépe zachytit komplexní vzory a detaily v datech, což často vede k větší přesnosti modelu. Zmenšení šířky na druhou stranu produkuje lehčí modely, které jsou ideální pro nasazení v prostředích s omezenými výpočetními zdroji. (26)

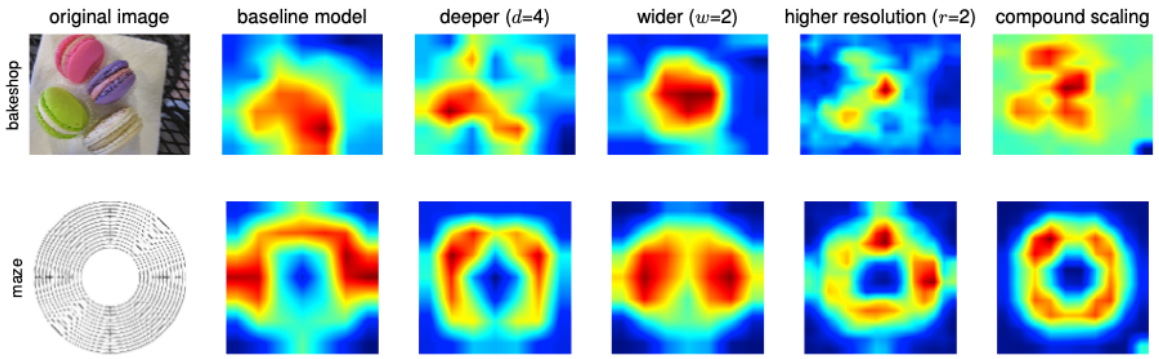
Měřítko hloubky modelu se týká celkového počtu vrstev v neuronové síti. Hlubší modely jsou schopny zachytit složitější reprezentace dat a abstraktnější vzory. To je často klíčem k dosažení výborné přesnosti v náročných úlohách. Avšak hlubší modely současně vyžadují více výpočetních zdrojů a dat pro trénink. Naopak mělčí modely jsou výpočetně efektivnější a snáze nasaditelné, ale nemusí nedosahovat takové přesnosti jako jejich hlubší protějšky.

Manipulace s rozlišením zahrnuje změnu velikosti vstupních obrázků. Obrázky s vyšším rozlišením poskytují detailnější a jemnější informace pro model, a to obvykle vede

k lepší výkonnosti. Bohužel vyšší rozlišení vyžaduje více paměti a výpočetního výkonu a nese s sebou problematické řešení pro zařízení s omezenými zdroji. Zatímco zajištění nižšího rozlišení vstupních dat bude šetrnější ke zdrojům, ale ztratí některé podrobnější prvky v obrazech. Z tohoto důvodu je hledání optimálního rozlišení kompromisem mezi výkonem a efektivitou. (27)

Lepší pochopení výhod metody složeného škálování je vidět na obrázku XY, který zobrazuje mapy aktivace škálování různými způsoby. Jak je vidět, model používající složené škálování má tendenci se zaměřovat na relevantnější části obrázků než na části s vyšším množstvím detailů týkajících se objektu. V porovnání s jinými modely, které buď nezachytí všechny objekty na obrázcích, nebo ztratí důležité detaily objektů, je tento přístup více efektivní a účinný. (26)

Obrázek 15 – Příklad rozpoznání obrazu za pomocí EfficientNet



Zdroj: https://arxiv.org/pdf/1905.11946.pdf

3.6 Aplikace CNN

Konvoluční sítě se staly nedílnou součástí současného světa. Díky své schopnosti efektivně zpracovávat obrazová data jsou základním nástrojem pro řadu aplikací v různých oblastech. Od rozpoznání obrazů a detekce objektů, přes rozpoznání tváři až k segmentaci a vylepšení rozlišení fotografií nebo obrázků. Následující příklady ukazují, jak je CNN využívána v praxi v reálných situacích a aplikacích.

3.6.1 Google Lens

Aplikace pro analýzu a interpretaci vizuální dat je vyvinutá společností Google. Konvoluční sítě jsou navržené tak, aby umožnil uživatelům prostřednictvím kamery smartphonu identifikovat a klasifikovat různé typy objektů ve svém okolí, včetně produktů, zvířat, rostlin, ale i textu. (28)

3.6.2 Detekce objektů – autonomní vozidla

Pokročilý systém asistence při řízení poskytuje řidiči nejnovější dostupné informace, co se právě děje v okolí, a to za pomocí kamer nebo sonarů umístěních v automobilu. Detekuje různé objekty v prostředí vozidla, prvky silniční infrastruktury, ostatní vozidla či chodce.

Nicméně stále existují výzvy. Detekce chodců a cyklistů je komplikovanější kvůli potencionální záměně. Dalším problémem jsou velké skupiny lidí nebo kol, zaparkovaná auta, které omezují viditelnost a tím tak brání detekci určitých objektů. Další průzkum a vývoje vyžaduje i oblast v přesnosti rozpoznání během různých denních dob nebo za nepříznivých povětrnostních podmínek. (29)

3.6.3 Analýza medicínských snímků – detekce onemocnění

Konvoluční neuronové sítě představují průlomovou technologii i v medicíně. Jejich schopnost segmentovat obraz může být klíčová při identifikaci specifických oblastí nádoru nebo sledování vývoje patologie v čase. Během nedávné pandemie COVID-19 se CNN staly významným nástrojem v boji proti viru. Efektivním zpracováním rentgenových snímků hrudníku a CT snímků plic mohou tyto sítě rychle a přesně diagnostikovat onemocnění, a to může být zásadní pro účinnou léčbu nebo kontrolu šíření viru.

Výzkum v oblasti CNN i nadále pokračuje s cílem zlepšit přesnost a spolehlivost těchto modelů. Zahrnuje řadu experimentů zaměřených na optimalizaci tréninkových procesů a vylepšení architektury konvolučních neuronových sítí. (30) (31)

3.6.4 Rozpoznání tváří – face ID

Technologie pro biometrickou autentizaci vyvinutá společností Apple, která je aktuálně běžně používaná na produktech společnosti – Face ID. V rámci systému se CNN používá k analýze obličejových rysů uživatele a porovnává je s uloženými daty v zařízení. Precizní a bezpečnou autentizaci zabezpečují různé fotonické komponenty, a to včetně běžné kamery, infračervené kamery, ale také dot projektoru, který vysílá více než 30 000 neviditelných infračervených bodů na tvář uživatele, a díky tomu je Face ID schopen rozpoznat jedinečné znaky a rysy obličeje. Získané hodnoty jsou předány neuronové síti pro analýzu a následnou autentizaci. (32) (33)

3.6.5 Segmentace obrazu – satelitní snímky

Konvoluční neuronové sítě se využívají i v analýze urbanizace prostřednictvím satelitních snímků. Pokročilé algoritmy strojového učení jsou schopné detekovat a klasifikovat různé urbanistické struktury, jako jsou budovy, silnice a parky. Umožňuje detailní sledování a průzkum růstu a vývoje měst, což je v hodné zejména v kontextu plánování měst a zároveň důležitá a cenná informace pro urbanisty nebo rozhodovací orgány. CNN může pomoci identifikovat oblasti rychlého urbanistického růstu, které by mohly vyžadovat další infrastrukturu a služby nebo detekuje nevyužitou oblast, která může být v budoucnu zastavěna. (34)

3.7 Hardware a software

Pro efektivní implementaci a trénování konvoluční neuronové sítě je často nezbytné využít výkonnější hardware, než jaký je k dispozici na běžných osobních počítačích. V tomto kontextu se nabízí využití vzdáleného počítače, který disponuje dostatečnou výpočetní kapacitou.

Pro získání informací o hardwaru na vzdáleném počítači lze použít příkaz nvidia-smi v terminálu. Tento příkaz poskytuje detailní přehled o dostupných grafických procesorech (GPU), jejich využití a dalších relevantních parametrech.

Obrázek 16 – Informace o hardwaru na vzdáleném počítači

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GeForce RTX 3090    Off  | 00000000:4C:00.0 Off |                  N/A |
|  0%   42C    P8    19W / 350W |  23650MiB / 24265MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A     44215      C   ...etra/b/venv/bin/python3.9      253MiB |
|    0   N/A  N/A     90779      C   ...etra/b/venv/bin/python3.9    23395MiB |
+-----------------------------------------------------------------------------+
```

Zdroj: Vlastní zpracování

3.7.1 Keras a TensorFlow

Keras a TensorFlow jsou dva softwarové nástroje, které hrají zásadní roli v implementaci a tréninku CNN. Jednou z nejpopulárnějších knihoven pro strojové a hloubkové učení navrženou pro programování v jazyce Python, je framework Keras. Při vývoji se autor Francoise Chollet řídil čtyřmi základními principy:

- Uživatelská přívětivost. Minimalistické uživatelské prostředí s jednoduchou a intuitivní API. Keras se pokouší eliminovat výskyt uživatelských chyb jasnou a stručnou zpětnou vazbou a zaměřuje se také na konzistenci v rámci API.
- Modularita. Celkový model lze chápat jako sadu nebo posloupnost samostatně konfigurovatelných menších modelů a ty tak bez omezení snadno propojit. Zejména neuronové vrstvy, aktivační funkce nebo optimalizátory může uživatel snadno vytvářet a upravovat podle své potřeby.
- Snadná rozšiřitelnost. Moduly, jako třeba třídy a funkce, jde snadno přidávat k již existujícím a tím snížit zbytečnou složitost kódu.
- Práce s jazykem Python. Keras se spoléhá na jazyk Python a díky tomu využívá všechny jeho výhody včetně výkonných nástrojů pro manipulaci s daty. (35)

Keras je navržen tak, aby umožnil rychlé experimentování s hlubokými neuronovými sítěmi, a to nejen pro účely práce výzkumných týmu, tak pro vývojáře v byznysových sférách. Díky následujícím vlastnostem ho tak činí jedním z nejpopulárnějších a nejvyužívanějším rámcům v oblasti hloubkového učení:

- Možnost provádět výpočet na procesoru (CPU anebo grafických kartách (GPU)),
- uživatelsky přívětivé rozhraní API usnadňující rychlé vytváření prototypů modelů hlubokého učení,
- podporuje zpracování konvoluční, rekurentní sítě, anebo jejich kombinaci,
- schopnost pracovat s různými síťovými architekturami, včetně modelů s více vstupy nebo výstupy. (4)

Protože Keras byl samostatnou knihovnou a neuměl pracovat a manipulovat s tensory a derivacemi využíval k tomu podpůrnou externí knihovnu TensorFlow.
Jde o otevřený software pro strojové učení navržen a spravován týmem Google Brain. Vzhledem k tomu, že je napsán v jazyce C++ dosahuje vysoké výkonnostní úrovně, důležité

pro výpočetně náročné úlohy hlubokého učení. Podporuje širokou škálu algoritmů a modelů strojového učení, včetně neuronových a konvolučních sítí.

Stejně tak jako Keras je TensorFlow navržen nejen pro výzkumné účely, ale také pro použití v produkčních systémech. Jeho velkou výhodou je aplikace na různých systém, a to i na mobilních zařízeních. Jde o velmi oblíbený nástroj pro hloubkové učení díky efektivní práci a výpočtech s multidimenzionálními poli. (35)

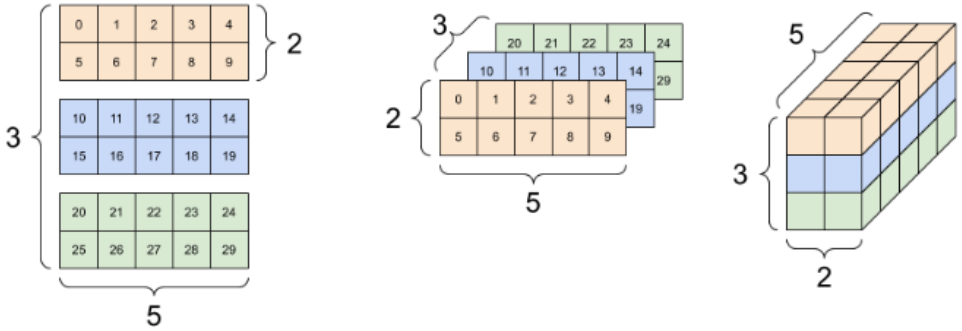
Název TensorFlow je odvozen z konceptu manipulace s tenzory (n-dimenzionální matice). Tenzory mohou existovat v různých dimenzích, které jsou charakterizovány jejich rankem. (36)

- Tensor rank 0 – Reprezentuje skalár
- Tensor rank 1 – Reprezentuje vektor
- Tensor rank 2 – Reprezentuje matici
- Tensor rank 3 – Reprezentuje třírozměrný tensor

Tenzory jsou základními prvky pro vstup a výstup modelů strojového a hloubkového učení. Každá dimenze tenzoru má svůj vlastní rozměr (shape), který určuje počet prvků obsažených v dané dimenzi. Například trojrozměrný tenzor s rozměry (3,2,5) se skládá z tří dimenzí, které mají rozměry 3, 2 a 5. Celkový počet prvků v tomto tenzoru je 60, což je výsledek násobení rozměrů (3*2*5). Tensory zaručují efektivní a flexibilní manipulaci s velkým množstvím dat, a to je důležité pro výpočetní operace v oblasti strojového učení.

Je důležité poznamenat, že od verze 2.0 je do TensorFlow již implementována knihovna Keras. Tato integrace umožňuje uživatelům využívat výhod obou knihoven na jedné platformě pro vývoj a implementaci modelů strojového učení. (36)

Obrázek 17 – Tensor



Zdroj: https://www.tensorflow.org/guide/tensor

3.7.2 Python

Keras a Tensorflow používají jako svůj hlavní programovací jazyk Python. A to hned z několika důvodů. Je to jazyk známý svou jednoduchou syntaxí, má bohatou škálu knihoven a nástrojů pro práci s modely umělé inteligence a analýzou dat. Především knihovny NumPy, pandas, scikit-learn, Matplotlib, které usnadňují zpracování a zobrazování dat po analýzu výsledků trénování model nebo vizualizaci vstupních dat. Bohatá dokumentace usnadňuje práci s Pythonem a napomáhá řešit problémy spojené s trénováním a vývojem modelů strojového učení. Důležité je zmínit i jeho multiplatformnost. Python podporuje i interaktivní vývoj prostřednictvím nástrojů jako je Jupyter Notebook, který je pro tyto účely ideální.

Nezbytností před samotným programováním je spouštění virtuálního prostředí. To zajistí izolaci knihoven a balíčků od ostatních projektů. Prostředí Pythonu se neustále vyvíjí, a proto by mohlo dojít ke konfliktům na již hotových projektech. Další výhodou je snadná migrace projektu mezi systémy. Pomocí vypsání požadavků (requirements) jsou známé všechny knihovny, balíčky a jejich verze potřebné pro projekt a nic nebrání spuštění na jiném zařízení. (37)

Instalaci virtuální prostředí (s předpokladem, že je již naistalován Python) je provedena v terminálu za pomocí příkazu (pro macOS)

```
$ python -m venv venv
```

Následná aktivace virtuálního prostředí:

```
$ source venv/bin/activate
```

Poté už stačí instalovat potřebné knihovny a balíčky pro konkrétní úlohu. Celkově lze říct, že kombinace jazyku Python s Keras a TensorFlow formuje výkonné prostředí pro vývoj a trénink modelů.

3.7.3 Jupyter notebook

Jupyter Notebook je vysoce ceněný nástroj pro správu a vývoj skriptů, který se stal standardem v oblasti datové analýzy a strojového učení. Poskytuje interaktivní webové rozhraní, které podporuje mnoho programovacích jazyků, včetně Pythonu. Uživatelé mohou psát a spouštět příkazy přímo v notebooku a okamžitě vidět jejich výstupy.

Jupyter Notebook je kompatibilní s řadou knihoven Pythonu, včetně NumPy a Matplotlib, které jsou široce používány pro numerické výpočty a vizualizaci dat. Integrace

umožňuje uživatelům snadno manipulovat s daty a vizualizovat výsledky přímo v notebooku. Na rozdíl od tradiční konzole, Jupyter Notebook uchovává historii příkazů a jejich výstupů, což usnadňuje jejich správu a úpravu. Kód lze také snadno organizovat a dokumentovat pomocí textových komentářů formátovaných v jazyce Markdown, a tak zvyšovat čitelnost a srozumitelnost kódu. (38)

Instalace Jupyter Notebooku je jednoduchá pomocí následujícího příkazu zadaného v terminálu:

```
$ python -m pip install notebook
```

Spuštění Jupyter Notebooku:

```
$ python -m notebook
```

4 Vlastní práce

Tato část práce se zaměřuje na detailní popis celého procesu, který zahrnuje přípravu a zpracování dat, sestavení konvolučního neuronového modelu pro rozpoznání egyptských hieroglyfů a prezentaci úspěšnosti predikce výsledného modelu. Následující pasáže jsou důležité pro porozumění celkové metodologii, která byla použita při vytváření a hodnocení modelu.

Nezbytné knihovny a moduly potřebné pro práci s daty a vytvoření modelu nejsou součástí skriptů prezentovaných v praktické části tohoto článku. Nicméně pro úplnost jsou tyto knihovny a moduly uvedeny v Příloze A (requirements.txt), která poskytuje kompletní přehled všech nástrojů a závislostí potřebných pro reprodukci prezentovaného postupu, který je v příloze B (Skript_klasifikace.ipynb).

4.1 Dataset a příprava dat

Po pečlivé analýze dostupných zdrojů byla pro účely rozpoznání egyptských hieroglyfů vybrána datová sada z platformy Kaggle. (https://www.kaggle.com/datasets/waleedumer/egyptian-hieroglyphics-datasets?resource=download).

Kaggle je renomovaná platforma pro datové vědce, která nabízí širokou škálu datových sad pro různé účely. Vybraná datová sada byla následně podrobena procesu přípravy dat. Tento proces je nezbytným krokem v jakémkoli projektu strojového učení a zahrnuje čištění dat, normalizaci, případnou augmentaci a další úpravy, které jsou podstatné pro efektivní trénování modelu. Příprava dat také zahrnuje rozdělení datové sady na trénovací, validační a testovací množinu, což umožňuje objektivní hodnocení výkonnosti modelu.

Po stažení datové sady bylo nezbytné provést její úpravu pro účely trénování modelu. Jedním z hlavních problémů, který se vyskytl, byla neúplnost dat. Některé .jpg soubory, které představovaly obrázky, neměly odpovídající .xml soubory, které by obsahovaly potřebná metadata. Aby se tento problém vyřešil, byla vytvořena funkce load_images, která byla nastavena tak, aby nejprve načítala .xml soubory a poté k nim dohledávala odpovídající .jpg soubory. Tímto způsobem bylo možné zajistit, aby všechny zpracované obrázky měly odpovídající anotace.

```
for file in os.listdir(image_dir):
 # print(file)
  if file.endswith('.xml'):
    annotation_path = image_dir + '/' + file
   # print(annotation_path)
    tree = ET.parse(annotation_path)
    root = tree.getroot()
```

Funkce kromě načtení .xml a jim odpovídající .jpg formátu, konvertuje obrázky do RGB a upravuje jejich velikost. Dále extrahuje labely a bounding boxy z .xml souborů a za pomocí hodnot z boundig boxů ořeže originální obrázky. Důvodem vytvoření pole s ořezanými obrázky byla předběžná visuální analýza, kdy .jpg soubory zbytečně obsahovaly nadbytečné pozadí s malými objekty, což by mohlo ztížit učení modelu a snížit jeho výkonost. Proto je výsledkem funkce návratová hodnota čtyř polí obsahující originální obrázky, anotace, bounding box a ořezané obrázky.

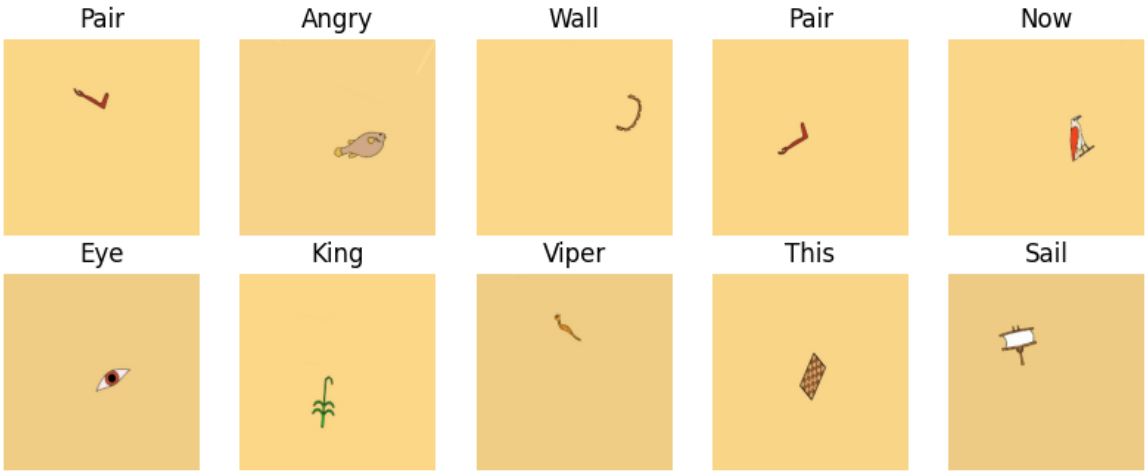
```
    for obj in root.findall('object'):
        # bounding box
        xmin = int(obj.find('bndbox/xmin').text)
        ymin = int(obj.find('bndbox/ymin').text)
        xmax = int(obj.find('bndbox/xmax').text)
        ymax = int(obj.find('bndbox/ymax').text)

        label = str(obj.find('name').text)
        labels.append(label)

        list_box = [xmin, ymin, xmax, ymax]
        list_boxes.append(list_box)

        cropped_image = image[ymin:ymax, xmin:xmax]
        #cropped_image = cv2.resize(image, target_shape)[ymin:ymax, xmin:xmax]
        cropped_image = cv2.resize(cropped_image, target_shape)

        cropped_images.append(cropped_image)
```

Pro zjištění správného zpracování dat byla implementována vizuální kontrola za použití speciálně vytvořené funkce show_img. Tato funkce vytváří mřížku obrázků o velikosti 5x5 a zobrazuje prvních x (dle zadaného vstupního parametru) obrázků z datové sady, doplněných svými odpovídajícími anotacemi.

Z celkového tréninkového souboru dat bylo náhodně vybráno 10 obrázků, které byly vizuálně zkontrolovány. Hlavním úkolem bylo ověřit, zda odpovídající anotace je správně přiřazena. Jakékoliv nesrovnalosti v načítání dat by mohly negativně ovlivnit proces tréninku a snížit přesnost modelu.

Vizuální kontrola také umožňuje hlubší seznámení s datovou sadou a posouzení její kvality. Pomáhá zjistit, zda jsou obrázky dostatečně kontrastní, zda neobsahují šum nebo jiné nežádoucí artefakty, které by mohly zkreslit výsledky. Na obrázku 18 je náhodně vybráno 10 obrázku z trénovacího souboru.

Obrázek 18 – Originální obrázky z datasetu



Zdroj: Vlastní zpracování

Data pro trénink a testování byla pečlivě předzpracována a rozdělena do odpovídajících sad. Celkový počet obrázků v datové sadě je 4937, z nichž 3889 obrázků je zařazeno do tréninkové sady a zbývajících 1048 obrázků tvoří testovací sadu. Velikost tréninkových a testovacích dat a jejich odpovídajících labelů byla ověřena promocí funkce len. Tato kontrola potvrdila, že data a titulky jsou správně spárovány a připraveny pro další zpracování a poskytují tak solidní základ pro trénink a testování modelu. Výpis celkového počtu a názvu tříd za pomocí následujícího skriptu je zobrazeno na obrázků 19.

Obrázek 19 – Velikost datasetů a jejich anotací

```
print('Train data size: ', len(train_data)),
print('Train labels size: ', len(train_labels)),
print('Test data size: ', len(test_data)),
print('Test labels size: ', len(test_labels))
```

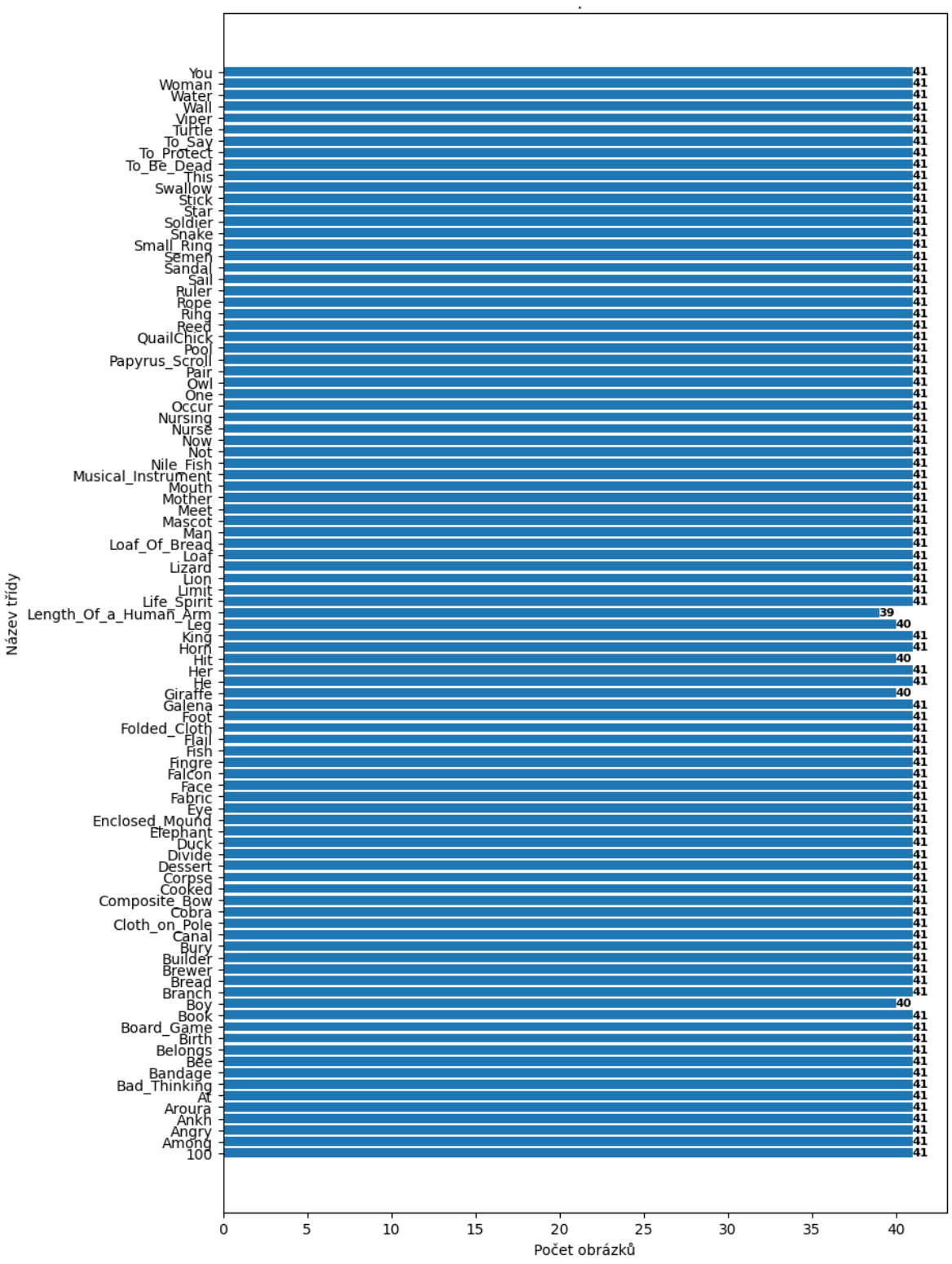
```
Train data size:  3889
Train labels size:  3889
Test data size:  1048
Test labels size:  1048
```

Zdroj: Vlastní zpracování

Pro získání přehledu o rozložení tréninkových a testovacích dat byla vytvořena funkce show_distr_dateset. Funkce generuje horizontální sloupcový graf, kde na ose Y jsou zobrazeny názvy tříd a na ose X odpovídající počty obrázků pro každou z nich. Labels jakožto vstupní parametr funkce využívá funkci np_unique s prametrem return_counts pro získání unikátních tříd a jejich počtů. Následně generuje sloupcový gram, kde každý sloupec reprezentuje počet obrázků pro danou třídu. Suma počtu je zobrazena přímo na sloupci pro snadnou orientaci.

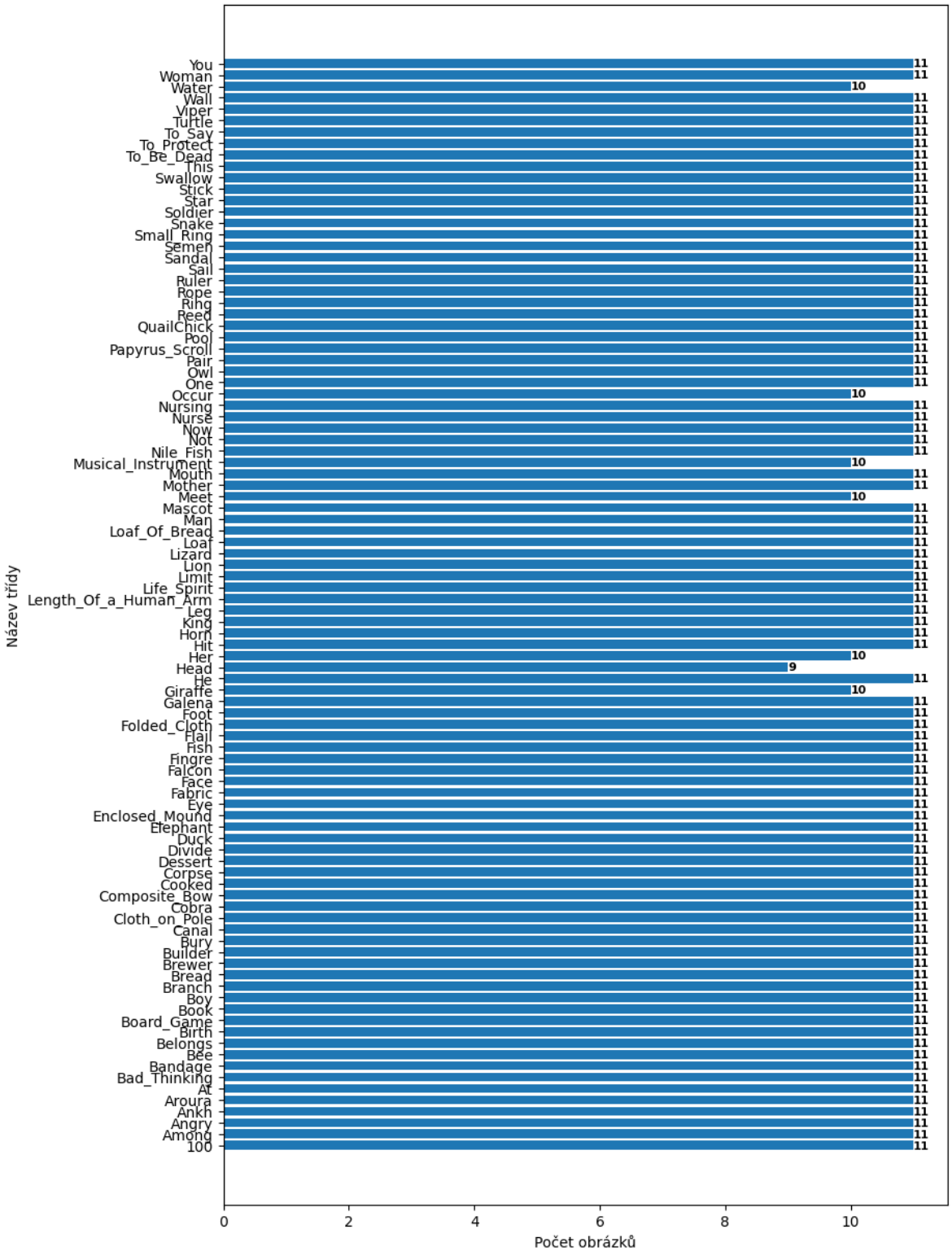
V rámci trénovacím souboru každé třídě náleží 39 až 41 různých obrázků. Zatímco u testovacího souboru, který je rozsahem menší, každá třída disponuje 9 až 11 obrázků. Toto vizuální zjištění předběžně zajistí, že bude model trénován a testován na reprezentativním vzorku dat.

Obrázek 20 – Rozdělení tréninkového souboru



Zdroj: Vlastní zpracování

Obrázek 21 – Rozdělení testovacího souboru



Zdroj: Vlastní zpracování

Pro hlubší pochopení rozmanitosti datové sady byla vytvořena funkce get_unique_name, která identifikuje a vrací seřazený seznam unikátních tříd v daném datasetu. Vstupní parametr funkce je seznam labelů, z nichž extrahuje jednotlivé třídy pomocí funkce set, následně je seřadí a vrátí jako výstup.

Z výsledku vyplývává, že obsahuje celkem 95 unikátních tříd. Informace není důležitá jen z hlediska analýzy dat, ale je to také parametr, který určuje počet neuronů v poslední vrstvě Dense při zpracování konvoluční sítě CNN, jakožto počet možných výstupů.

Obrázek 22 – Počet tříd a jejich seznam

```
unique_class_names= set(train_labels)
class_names = sorted(unique_class_names)
len(class_names), class_names
```

```
(95,
 ['100',
  'Among',
  'Angry',
  'Ankh',
  'Aroura',
  'At',
  'Bad_Thinking',
  'Bandage',
  'Bee',
  'Belongs',
  'Birth',
```

Zdroj: Vlastní zpracování

Nezbytným krokem pro potřeby modelu byla kategorizace názvů tříd. Model vyžaduje práci s číselnými hodnotami, proto byla vytvořena vlastní funkce **get_class_index**, která projde všechny názvy tříd z datasetu a přiřadí jim příslušní index. Výsledek této operace je pole **index_labels**, kde jsou uloženy veškeré indexované hodnoty. Jde o další stěžejní krok pro úspěšnou kategorizaci dat rozpoznávání hieroglyfů na základě indexované anotace.

```
def get_class_index(in_labels, in_class_names):
    for i, n in enumerate(in_labels):
        for j, k in enumerate(in_class_names):
            if n == k:
                in_labels[i] = j
    Labels = np.array(in_labels)
    return Labels
```

Posledním krokem u surové úpravy dat je ověření správné velikosti tvaru obrázků s indexovanými štítky. Pole obsahující obrázky train_data je čtyřdimenzionální: počet x šířka x výška x barevné kanály. To je velmi důležitá informace hlavně při volbě parametrů pro vstupní vrstvu modelu. Tímto ověřením lze říci, zda jsou data konzistentní a vhodná pro CNN model.

Obrázek 23 – Ověření správnosti vstupních dat

```
print(f'Shape of train images: {train_data.shape}')
print(f'Shape of train index labels: {train_labels_index.shape}')
print(f'Print train index labels: {train_labels_index}')
```

```
Shape of train images: (3889, 224, 224, 3)
Shape of train index labels: (3889,)
Print train index labels: [68  2 91 ...  5 43 54]
```

Zdroj: Vlastní zpracování

Běžnou praxí při tréninku a vyhodnocování konvolučních neuronových sítí je rozdělení trénovacích dat na další dvě sady: tréninkové a validační. Před zmíněným rozdělením proběhne ještě náhodné promíchání dat, což eliminuje přeučení a ovlivnění schopnosti modelu generalizovat nová data. To vše zajistí funkce **train_test_split** z knihovny scikit-learn. Podíl dat, který bude přiřazen validačním datům, je nastaven na 0.25. To znamená, že 25 % dat budou validační, a zbývajících 75 % budou data testovací. Důležité je dodat, že testovací data zůstávají oddělena. Po provedení operace odpovídají X_train a y_train tréninkovým datům a anotacím, zatímco X_val a y_val obsahují validační data s odpovídající anotací.

```
X_train, X_val, y_train, y_val = train_test_split (data_set,
                                                   in_labels,
                                                   shuffle = True,
                                                   test_size = 0.25,
                                                   random_state = 42)
```

4.2 Tvorba modelu

Tato část popisuje sestavení prvního modelu pro trénování klasifikace egyptských hieroglyfů. Model je postaven jako konvoluční neuronová síť a je prvním krokem k vytvoření sofistikovaného systému pro rozpoznání a klasifikaci hieroglyfů. Slouží jako

základ pro další iterace a vylepšení. V dalších částech práce bude dále laděn, upravován a zlepšován na základě zpětné vazby a výsledků z tréninku, aby dosáhl co nejlepších výsledků v klasifikaci.

Model je postaven jako sekvenční, takže vrstvy jsou uspořádány lineárně, jedna za druhou. Začíná vstupní konvoluční vrstvou. Ta přijímá vstupní obrazy o rozměrech 224x224 pixelů se třemi kanály (RGB) a 32 konvolučními filtry. Aktivační funkce ReLu je použita pro zvýšení nelinearity modelu.

Následují další konvoluční vrstvy, kde počet filtrů se postupně zvyšuje 64, 128, 128, 256. Po každé konvoluční vrstvě je aplikovanávrstva max pooling, která snižuje rozměry výstupů z předchozí vrstvy tím, že z každého okna o velikosti 2x2 vybere maximální hodnotu. Proces tak redukuje počet parametrů a snaží se zachovat důležité informace.

Po poslední vrstvě max pooling je vícekanálový vstup transformován na vektor pomocí vrstvy Flaten. Ten je důležitý pro plně propojené vrstvy, které očekávají jednorozměrný vstup.

První propojená vrstva (Dense) obsahuje 512 neuronů a její aktivační funkcí je také ReLu. Následuje konečná plně propojená vrstva, tentokrát s počtem neuronů odpovídajícím počtu tříd v datech. Jako aktivační funkci využívá softmax, kterou vypočítává pravděpodobnosti příslušnosti k jednotlivým třídám. Jako závěrečný klasifikátor přiřadí vstupní obraz k jedné z tříd na základě identifikovaných rysů.

Struktura modelu je následující:

Tabulka 3 – Architektura prototypu modelu

Pořadí	**Vrstvy**
1	Con2D 32 -> Pool
2	Con2D 64 -> Pool
3	Con2D 128 -> Pool
4	Con2D 128 -> Pool
5	Con2D 256 -> Pool
6	FLAT
7	Dense 512
8	Dense 95 (počet tříd)

Zdroj: Vlastní zpracování

Model je zkompilován optimalizačním algoritmem Adam, který je známý svou efektivitou a širokým využitím v různých úlohách strojového učení. Pro účely klasifikace s více třídami, kde jsou třídy označeny jako celá čísla, je jako ztrátová funkce nastavena na SparseCategoricalCrossentropy. Porovnává skutečné a předpovězené třídy a vypočítává ztrátu, která je následně využita pro aktualizaci vah modelu.

Následná výkonnost modelu je vyhodnocována pomocí metriky accuracy a poskytne rychlý přehled o úspěšnosti modelu, kdy určí poměr správně klasifikovaných příkladů vůči celkovému počtu.

Kód pro kompilaci:

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

4.2.1 Trénování modelu

Trénink modelu byl zahájen s použitím metody fit. Proces tréninku probíhal na původních, neupravených datech po dobu 65 epoch, přičemž každá epocha byla rozdělena na kroky odpovídající velkosti souboru.

Byl definován tzv. callback, speciální funkce, která sleduje hodnotu ztrátové funkce během tréninku. Pokud se hodnota ztrátové funkce nezlepší po dobu 5 epoch, callback zastaví trénink. Zamezí přetrénování modelu a zbytečnému dalšímu tréninku, který by nepřinesl významné zlepšení výsledků. Parametr verbose byl nastaven na hodnotu 1, proto budou v průběhu tréninku poskytovány podrobné informace. To zahrnuje oznámení i o ukončení tréninku v případě, že nedochází k dostatečnému zlepšení hodnoty ztrátové funkce. Výsledky jsou uloženy do proměnné histrory_model_1, pro snadnější zpětné zobrazení o vývoji ztráty a přesnosti modelu během tréninku.

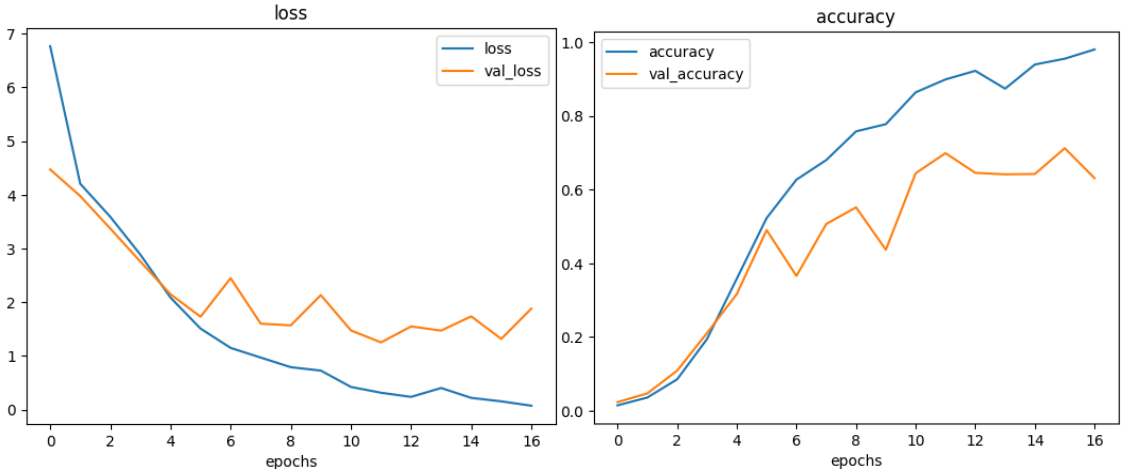
Kód pro trénink:

```
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=5, verbose=1)
history_model_1 = model_1.fit(origin_train_data,
                    epochs=65,
                    steps_per_epoch=len(origin_train_data),
                    validation_data = origin_val_data,
                    validation_steps=len(origin_val_data),
                    callbacks=[early_stopping_callback])
```

Jak už bylo zmíněno, metoda fit vrací objekt History, který obsahuje záznam o průběhu tréninku modelu. Pro lepší vizualizaci a analýzu těchto dat byla vytvořena funkce plot_loss_curves. Z historie tréninku zobrazuje dva grafy s vývojem a hodnotu ztrátové funkce a přesnosti na trénovacích a validačních datech během celého procesu tréninku.

Z těchto grafů lze jasně vyčíst, zda dochází k přeučení modelu. Indikací je, že model dosahuje vysoké korektnosti na trénovacích datech, ale jeho výkon na validačních datech je výrazné horší. To může naznačovat, že model se příliš přizpůsobil trénovacím datům a nedokáže dobře generalizovat nová data. Jednou z možných příčin tohoto problému může být malá velikost rozpoznávaného objektu na obrázku v porovnání s pozadím.

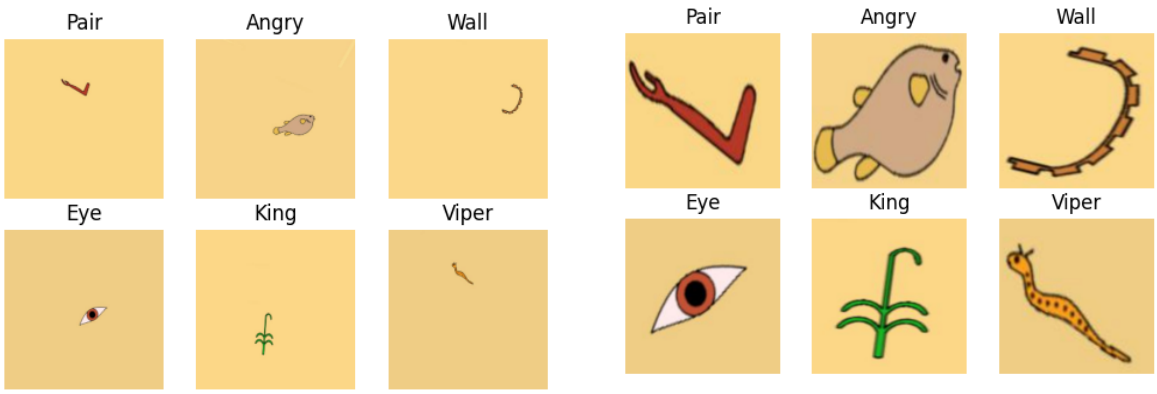
Obrázek 24 – Ztrátová funkce a přesnost klasifikace prototypu modelu na originálních datech



Zdroj: Vlastní zpracování

Na stejném modelu, který byl původně trénován na neupravených datech, je nyní provedena klasifikace s obrázky, které byly oříznuty. Tento postup pomůže modelu lépe se soustředit na relevantní části, což bylo identifikováno jako potenciální problém v předchozím tréninku modelu.

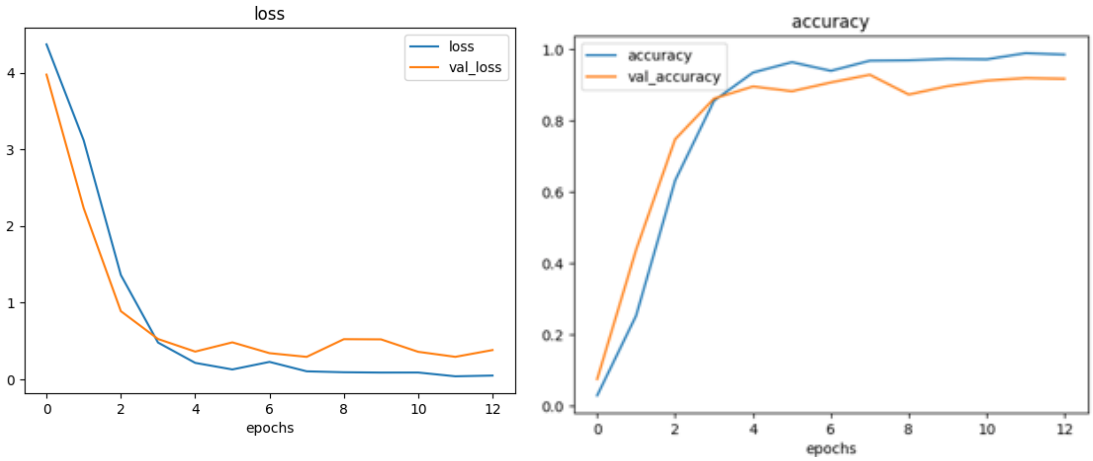
Obrázek 25 – Originální vs oříznuté obrázky z datasetu



Zdroj: Vlastní zpracování

Po klasifikaci s oříznutými obrázky se ukazuje zlepšení v grafech ztrátové funkce a přesnosti. Nicméně celkový výkon modelu je stále nedostatečný. S přesností rozpoznání kolem 88 % nejsou výsledky na očekávané úrovni pro tento druh úlohy, což signalizuje potřebu dalších úprav modelu.

Obrázek 26 – Ztrátová funkce a přesnost klasifikace prototypu modelu na upravených datech



Zdroj: Vlastní zpracování

Aby se zlepšila efektivita trénování modelu, byla provedena normalizace obrázků. Tento proces spočívá v převedení hodnot pixelů, které se pohybují v rozmezí 0 až 255, na nový rozsah 0 až 1. Cílového stavu se dosáhne dělením hodnot pixelů hodnotou 255. Upravený rozsah je vhodnější pro některé aktivační funkce a stabilitu tréninku. Jedním z důvodů je, že gradienty, které jsou základním mechanismem pro aktualizaci vah v procesu učení, se lépe chovají v blízkosti nuly. Dále se zjednodušuje interpretace dat pro model, protože všechny hodnoty pixelů jsou nyní v jednotném rozsahu.

Další optimalizace multiklasifikačního modelu pro rozpoznání egyptských hieroglyfů byla dosažena implementací augmentace dat pomocí třídy ImageDataGenerator z knihovny Keras. Rozšíření sady tréninkových dat vyjádřeno konkrétně:

- Náhodný úhel rotace, obrázky budou náhodně rotovat pod úhlem 10 stupňů,
- aplikace náhodného transverzního posunu do 10 %,
- obrázky jsou náhodně přiblíženy až o 10 %,
- obrázky jsou náhodně posunuty horizontálně až do 10 % své šířky.

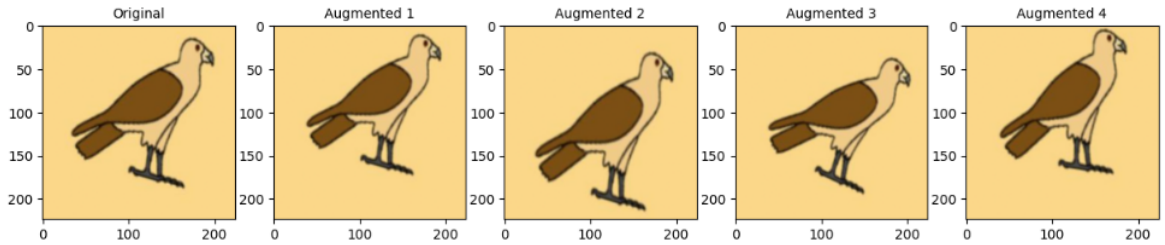
```
ImageDataGenerator (rotation_range=10,
                    shear_range=0.1,
                    zoom_range=0.1,
                    width_shift_range=0.1)
```

Nastavení garantuje téměř maximální diverzitu tréninkového souboru a snižuje riziko přizpůsobení datům a riziko přeučení. Je důležité poznamenat, že augmentace data se nepoužívá pro validační ani testovací dataset.

Výsledné instanci generátoru pro augmentaci tréninkových dat byla nastavena ještě batch_size=32, která určuje kolik obrázků bude v jedné dávce. Volba hodnoty 32 je často používaným standardem v oblasti hloubkového učení. Jde o kompromis mezi výpočetní efektivitou a přesností gradientu.

Pro ověření správné funkčnosti augmentace byla provedena vizualizace. Náhodně byl vybrán jeden oříznutý obrázek z tréninkového datasetu. V prvním sloupci se nachází originální verze obrázku a zbývající čtyři sloupce obsahují různé varianty augmentové verze.

Obrázek 27 – Originální obrázek s argumentovanými varianty



Zdroj: Vlastní zpracování

Následná tabulka reprezentuje výsledky čtyř různých experimentů s datasetem pro klasifikaci obrazu. Každý řádek reprezentuje jiný typ předzpracovaných dat a ukazuje hodnoty ztrátové funkce a přesnosti klasifikace pro daný experiment.

Tabulka 4 – Výsledky hodnot ztrátové funkce a přesnosti klasifikace modelů

Dataset	Ztrátová funkce	Přesnost klasifikace
Originální	1,8807	0,6310
Oříznutý	0,4060	0,8849
Oříznutý a normalizovaný	0,3806	0,9178
Oříznutý, normalizovaný a augmentovaný	0,1675	0,9342

Zdroj: Vlastní zpracování

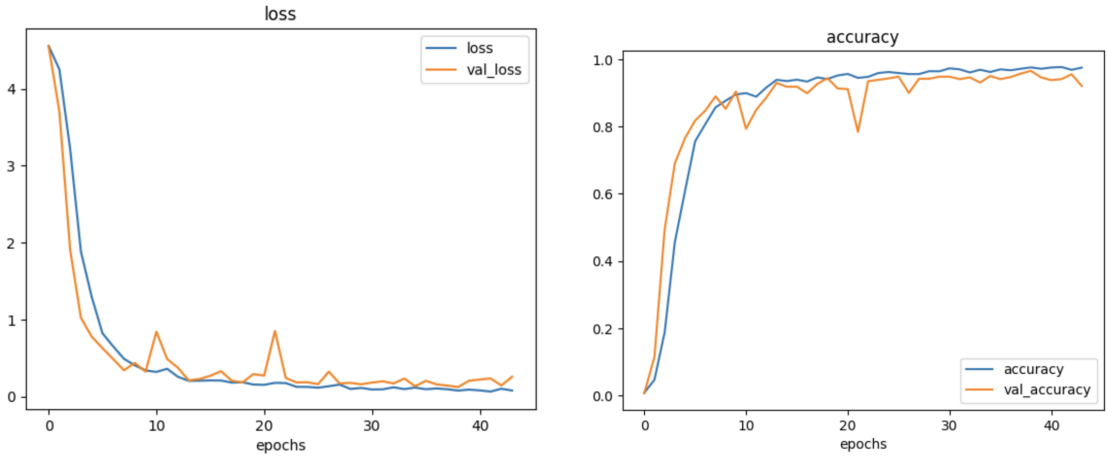
4.3 Optimalizace výkonu modelu

Před finálním stanovením architektury modelu proběhla řada pokusů a experimentů zaměřených na ladění a optimalizaci. Proces zahrnoval přidávání a odebírání různých vrstev včetně úpravy jejich filtrů. Dále byly prováděny úpravy parametrů v Dropout vrstvě a také pokusy s hodnotou learning rate v rámci optimalizačního algoritmu Adam.

4.3.1 Model 1

První upravený model se liší od přechozího ve dvou hlavních aspektech. První rozdíl spočívá v přítomnosti dvou vrstev Dropout s hodnotou 0,4, které v předchozím modelu chybí. Vrstvy pomáhají snižovat přeučení. Nastaví náhodně vybranou část vstupních neuronů na 0 při každé aktualizaci během tréninku. V tomto případě jde o 40 %. Druhý rozdíl spočívá v počtu neuronů v první vrstvě Dense. Nový model má v této vrstvě nastaveno 1024 jednotek, což je dvojnásobek oproti původnímu nastavení.

Obrázek 28 – Ztrátová funkce a přesnost klasifikace modelu 1



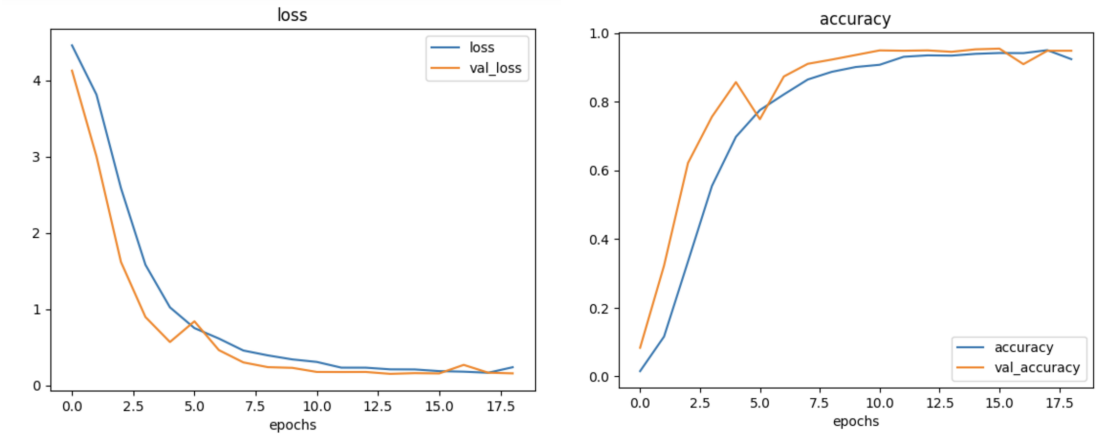
Zdroj: Vlastní zpracování

4.3.2 Model 2

Model 1 a 2 jsou si velmi podobné, avšak je mezi nimi jeden klíčový prvek, a to v konfiguraci jejich optimalizátorů. Zatímco model 1 má nastavenou standardní hodnotu optimalizátoru Adam, v modelu 2 jsou na základě experimentování upraveny parametry, které vedly k nejlepšímu výkonu modelu 2.

Konkrétně byla nastavena rychlost učení na 0,0001 a hodnota regularizační techniky weight decay na 1e-6, který v modelu 1 nebyl nastavena vůbec.

Obrázek 29 – Ztrátová funkce a přesnost klasifikace modelu 2

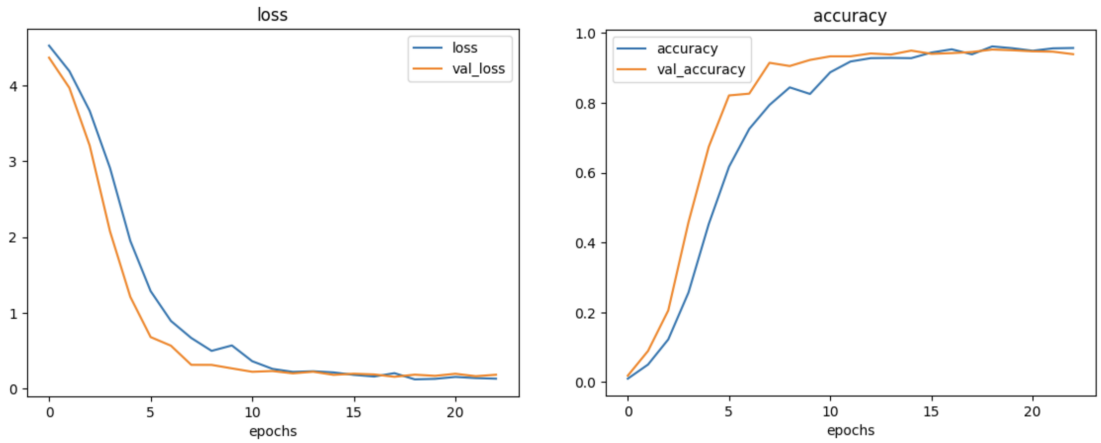


Zdroj: Vlastní zpracování

4.3.3 Model 3

Model 3 je od modelu 2 odlišný v architektuře konvolučních vrstev. Zatímco model 2 má jednu konvoluční vrstvu pro každou velikosti filtru (32, 64, 128, 256), v modelu 3 se nacházejí dvě konvoluční vrstvy pro velikosti filtrů 128, 256 a navíc jsou přidány dvě konvoluční vrstvy s 512 filtry. Díky tomu je model 3 schopen naučit se složitější vzorce v datech.

Obrázek 30 – Ztrátová funkce a přesnost klasifikace modelu 3



Zdroj: Vlastní zpracování

4.3.4 Model 4

Poslední model 4 představuje několik modifikací oproti modelu 3. První změnou je snížení dropout vrstvy z 0,4 na 0,2. Dále zahrnuje další dvojici konvolučních vrstev s 256 filtry a obsahuje ještě jednu Dense vrstvu s 1024 neurony. Poslední úpravou je implementace Batch Normalization vrstvy, která zlepšila výkon za pomocí normalizace vstupu. Avšak tyto úpravy vyžadují více výpočetních zdrojů, tudíž se nepatrně zvýšil čas tréninku v jednotlivých epochách. Přesná architektura modelu a výsledky tréninku budou popsány v další kapitole.

Tabulka 5 – výsledky hodnot ztrátové funkce a přesnosti klasifikace modelů

Název modelu	Ztrátová funkce	Přesnost modelu
Model 1	0,1205	0,9566
Model 2	0,1495	0,9455
Model 3	0,1557	0,9455
Model 4	0,0941	0,9609

Zdroj: Vlastní zpracování

4.3.5 Architektura modelu 4

Konečná architektura modelu, která dosáhla nejlepších výsledku v klasifikaci hieroglyfů, je rozdělena do sedmi bloků a je popsána níže.

První blok – vstupní:

- standardní konvoluční vrstvy s 32 filtry s velikostí jádra 3x3 a aktivační funkci ReLu, vrstva přijímá vstupní obrazy o rozměrech 224x224 s 3 kanály (RGB),
- vrstva Maxpoolingu s velikostí jádra 2x2.

Druhý blok:

- standardní konvoluční vrstvy s 64 filtry s velikostí jádra 3x3 a aktivační funkci ReLu,
- Maxpoolingu s velikostí jádra 2x2.

Třetí blok:

- standardní konvoluční vrstvy s 128 filtry s velikostí jádra 3x3 a aktivační funkci ReLu,
- standardní konvoluční vrstvy s 128 filtry s velikostí jádra 3x3 a aktivační funkci ReLu,
- Maxpoolingu s velikostí jádra 2x2.

Čtvrtý blok:

- standardní konvoluční vrstvy s 256 filtry s velikostí jádra 3x3 a aktivační funkci ReLu,
- standardní konvoluční vrstvy s 256 filtry s velikostí jádra 3x3 a aktivační funkci ReLu,
- Maxpoolingu s velikostí jádra 2x2,
- vrstva Dropout s dropoutem 0,2 (20 % vstupních neuronů bude během tréninku náhodně deaktivována).

Pátý blok:

- standardní konvoluční vrstvy s 256 filtry s velikostí jádra 3x3 a aktivační funkci ReLu,

- standardní konvoluční vrstvy s 256 filtry s velikostí jádra 3x3 a aktivační funkci ReLu,
- Maxpoolingu s velikostí jádra 2x2
- vrstva Dropout s dropoutem 0,2.

Šestý blok:

- standardní konvoluční vrstvy s 512 filtry s velikostí jádra 3x3 a aktivační funkci ReLu,
- standardní konvoluční vrstvy s 512 filtry s velikostí jádra 3x3 a aktivační funkci ReLu,
- Maxpoolingu s velikostí jádra 2x2,
- zakončen vrstvou Dropout s dropoutratem 0,2.

Sedmý blok – výstupní:

- Flatten vrstva převádí vstup do vektoru
- plně propojena vrstva Dense s 1024 neurony s aktivační funkcí ReLu,
- plně propojena vrstva Dense s 1024 neurony s aktivační funkcí ReLu,
- normalizační vrstva,
- Droupout vrstva s dropoutratem 0,2,
- Dense plně propojená vrstva s počtem neuronů odpovídajícím počtu tříd s aktivační funkcí Softmax.

Tabulka 6 – Architektura nejlepšího klasifikačního modelu

Pořadí	**Vrstvy**
1	Con2D 32 -> Pool
2	Con2D 64 -> Pool
3	Con2D 128
4	Con2D 128 -> Pool
6	Con2D 256
7	Con2D 256 -> Pool
8	Dropout 0,2
9	Con2D 256
10	Con2D 256 -> Pool
11	Dropout 0,2
12	Con2D 512
13	Con2D 512 -> Pool
14	Dropout 0,2
15	FLAT
16	Dense 1024
17	Dense 1024
18	BatchNormalization
19	Dropout 0,2
20	Dense 95

Zdroj: Vlastní zpracování

Kompilace modelu je součástí definované funkce create_model_tweak_4, která sestavuje model, aby byl připraven k tréninku a validaci. Model je kompilován s optimalizátorem Adam s rychlostí učení 0,0001 a váhovým úpadkem 1e-6. Během ladění a experimentování se ukázal tento optimalizátor jako nejlépe nastaven. Ztrátová funkce byla zvolena SparseCategoricalCrossentropy jako u úplně prvního modelu. V tomto případě jiná volba není možná.

Výstup architektury konvoluční neuronové sítě získaný spuštěním funkce model.summary v Kerasu je zobrazen na obrázku 31. Přehled uvádí názvy a typy jednotlivých vrstev modelu, rozměry výstupu každé vrstvy a počet trénovatelných a netrénovatelných parametrů v každé vrstvě.

Obrázek 31 – Výstup funkce model_sumary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_76 (Conv2D)           (None, 222, 222, 32)      896

max_pooling2d_54 (MaxPooli   (None, 111, 111, 32)      0
ng2D)

conv2d_77 (Conv2D)           (None, 111, 111, 64)      18496

max_pooling2d_55 (MaxPooli   (None, 55, 55, 64)        0
ng2D)

conv2d_78 (Conv2D)           (None, 55, 55, 128)       73856

conv2d_79 (Conv2D)           (None, 55, 55, 128)       147584

max_pooling2d_56 (MaxPooli   (None, 27, 27, 128)       0
ng2D)

conv2d_80 (Conv2D)           (None, 27, 27, 256)       295168

conv2d_81 (Conv2D)           (None, 27, 27, 256)       590080

max_pooling2d_57 (MaxPooli   (None, 13, 13, 256)       0
ng2D)

dropout_30 (Dropout)         (None, 13, 13, 256)       0

conv2d_82 (Conv2D)           (None, 13, 13, 256)       590080

conv2d_83 (Conv2D)           (None, 13, 13, 256)       590080

max_pooling2d_58 (MaxPooli   (None, 6, 6, 256)         0
ng2D)

dropout_31 (Dropout)         (None, 6, 6, 256)         0

conv2d_84 (Conv2D)           (None, 6, 6, 512)         1180160

conv2d_85 (Conv2D)           (None, 6, 6, 512)         2359808

max_pooling2d_59 (MaxPooli   (None, 3, 3, 512)         0
ng2D)

dropout_32 (Dropout)         (None, 3, 3, 512)         0

flatten_10 (Flatten)         (None, 4608)              0

dense_24 (Dense)             (None, 1024)              4719616

dense_25 (Dense)             (None, 1024)              1049600

batch_normalization_4 (Bat   (None, 1024)              4096
chNormalization)

dropout_33 (Dropout)         (None, 1024)              0

dense_26 (Dense)             (None, 95)                97375

=================================================================
Total params: 11716895 (44.70 MB)
Trainable params: 11714847 (44.69 MB)
Non-trainable params: 2048 (8.00 KB)
_________________________________________________________________
None
```

Zdroj: Vlastní zpracování

4.3.6 Trénink modelu

Obrátek 32 poskytuje vizuální reprezentaci výstupu metody fit, který obsahuje historii tréninku modelu, což umožňuje detailní analýzu průběhu tréninku. Výstupní data ukazují postup tréninku v každé epoše, včetně hodnoty ztrátové funkce a přesnosti na validační sadě. V první epoše model dosáhl na tréninkové sadě hodnoty ztráty 4,4666 a přesnosti 0,0209, zatímco na validační sadě dosáhl hodnoty ztráty 4,5112 a přesnosti 0,00195. Do 24 epochy se výkon modelu výrazně zlepšuje. Dosahující výsledky na tréninkové sadě ukazují hodnoty ztráty 0,007 a přesnosti 0,9709. Avšak na validační sadě se zvýšila na 0.2088, což může signalizovat potenciální přeučení. Aby se tomuto jevu zabránilo, model obnoví váhy z epochy, která dosáhla nejlepších výsledků (19. epocha) a ukončí trénink.

Obrázek 32 – Průběh tréninku modelu 4

```
Epoch 18/65
92/92 [==============================] - 22s 241ms/step - loss: 0.1040 - accuracy: 0.9643 - val_loss: 0.1156 - val_
accuracy: 0.9589
Epoch 19/65
92/92 [==============================] - 22s 241ms/step - loss: 0.0854 - accuracy: 0.9722 - val_loss: 0.0941 - val_
accuracy: 0.9609
Epoch 20/65
92/92 [==============================] - 22s 241ms/step - loss: 0.0811 - accuracy: 0.9729 - val_loss: 0.0970 - val_
accuracy: 0.9568
Epoch 21/65
92/92 [==============================] - 23s 244ms/step - loss: 0.0865 - accuracy: 0.9729 - val_loss: 0.1491 - val_
accuracy: 0.9517
Epoch 22/65
92/92 [==============================] - 22s 240ms/step - loss: 0.1115 - accuracy: 0.9643 - val_loss: 0.1008 - val_
accuracy: 0.9692
Epoch 23/65
92/92 [==============================] - 22s 243ms/step - loss: 0.1014 - accuracy: 0.9709 - val_loss: 0.1139 - val_
accuracy: 0.9599
Epoch 24/65
92/92 [==============================] - ETA: 0s - loss: 0.0787 - accuracy: 0.9709Restoring model weights from the
end of the best epoch: 19.
92/92 [==============================] - 22s 241ms/step - loss: 0.0787 - accuracy: 0.9709 - val_loss: 0.2088 - val_
accuracy: 0.9507
Epoch 24: early stopping
```

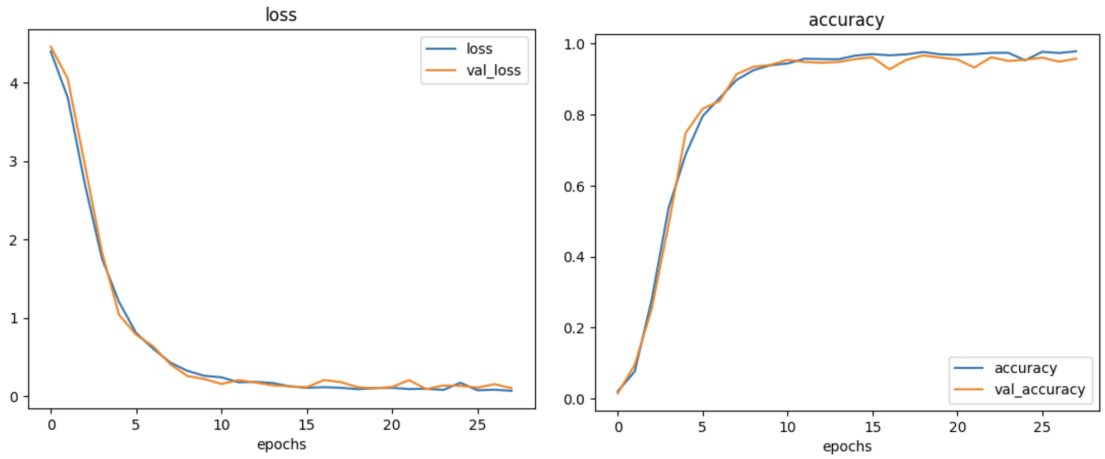
Zdroj: Vlastní zpracování

4.3.7 Metriky hodnocení modelu

Na obrázku 33 s grafem ztrátové funkce lze pozorovat, že hodnoty jak pro tréninková, tak validační data konverzují k nule. Křivky nevykazují žádné výrazné odchylky, což ukazuje na stabilní učení modelu bez přítomnosti přetrénování nebo podtrénování. Jak je vidět, trend naznačuje, že model se stále zlepšuje a minimalizuje chybu předpovědi.

Podobně je tomu na grafu týkající se hodnocení přesnosti. Tréninková i validační data se přibližují k hodnotě 1. Model tedy klasifikuje správně stále větší procento vstupních dat. Stejně jako u ztrátové funkce i zde nejsou patrné žádné výrazné výkyvy.

Obrázek 33 – Ztrátová funkce a přesnost klasifikace modelu 4



Zdroj: Vlastní zpracování

4.3.8 Evaluace modelu

Kód (viz níže) reprezentuje aplikaci metody evaluate na posledním nejlepším modelu. Metoda je zodpovědná za vyhodnocení výkonnosti modelu na validačních datech. Parametr steps je nastaven na hodnotu odpovídající velikosti validačních dat, což implikuje, že proces vyhodnocení bude proveden na celém souboru zahrnující všechny dávky a každý vzorek v datech bude použit právě jednou. Výsledek postupu umožňuje získat robustní odhad schopnosti modelu generalizovat nová, dosud neviděná, data, což je ideální pro posouzení jeho praktické aplikovatelnosti.

```
model_tweak_4.evaluate(aug_val_data, steps= len(aug_val_data))
```

Výstup výše zmíněné metody evaluate reprezentuje výsledky (obrázek 34) evaluace modelu na validačních datech. Celkový proces evaluace zahrnoval 31 kroků odpovídající počtu dávek v datasetu a byl dokončen za 2 sekundy. Hodnota ztrátové funkce, jakožto hlavní měřítko úspěšnosti modelu, dosáhla na validačních datech hodnoty 0,0941. Nižší hodnota indikuje lepší schopnost modelu předpovídat správné třídy. Přesnost modelu, která je vicemistrem v důležitosti měření, dosáhla hodnoty 0,9609, což odpovídá 96, 09 %. Tento výsledek svědčí o vysoké kvalitě modelu efektivně generalizovat neviděná data.

Obrázek 34 – Výsledek evaluace modelu

```
31/31 [==============================] - 2s 55ms/step - loss: 0.0941 - accuracy: 0.9609
[0.09411059319972992, 0.9609455466270447]
```

Zdroj: Vlastní zpracování

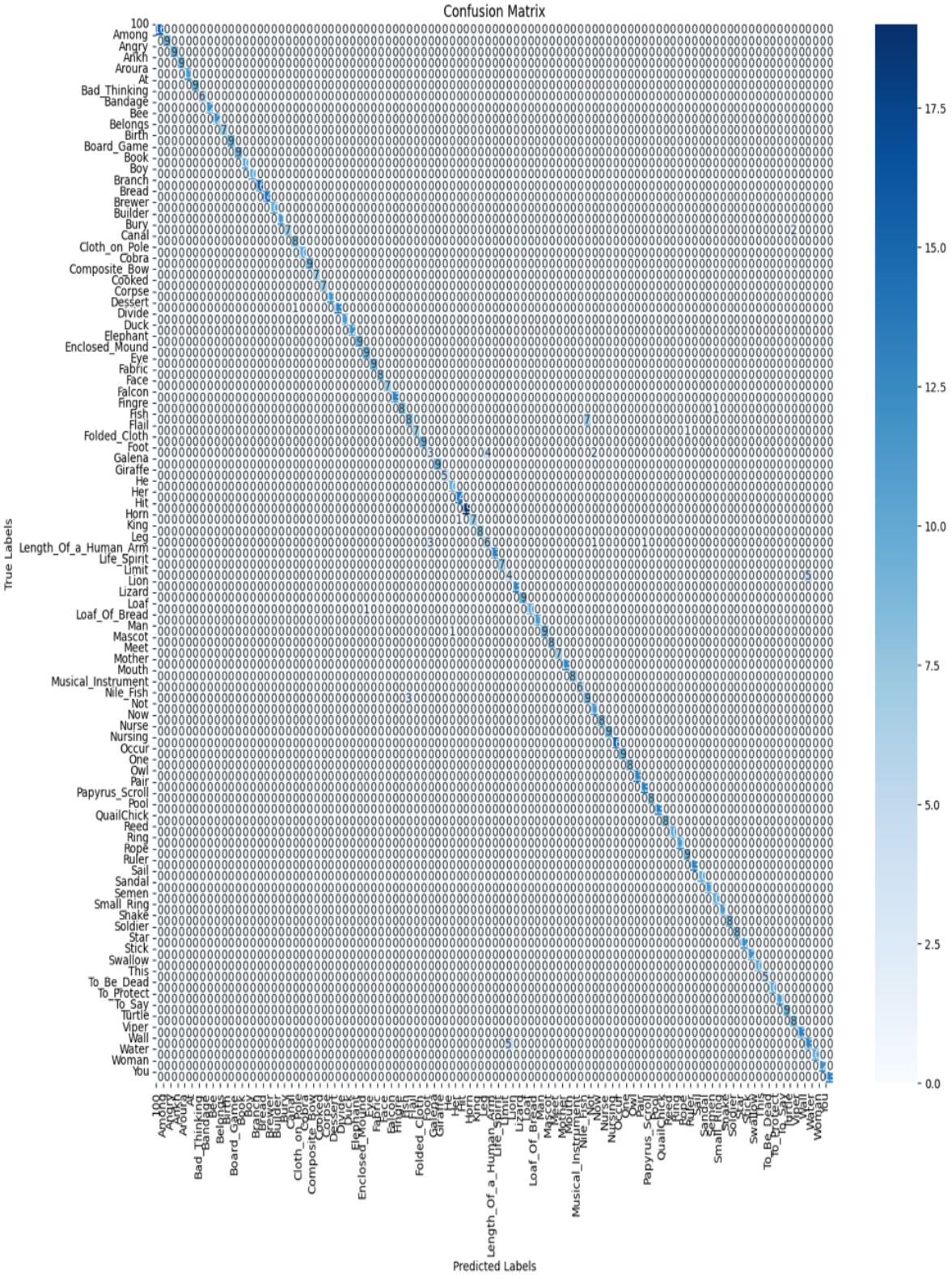
4.3.9 Matice záměn (Confusion matrix)

Pro detailní analýzu výkonu modelu a identifikace specifických oblasti, kde model správně či nesprávně klasifikoval data, byla implementována metoda maticí záměn (confusion matrix). Skript pro generování a vizualizaci této matice byl vytvořen s využitím metriky z knihovny sklearn.metrics a dalšího vlastního doladění. Výsledná matice je zobrazena na obrázku 35.

Matice formuje 95 tříd, které jsou textově vypsány na obou osách. Diagonála matice reprezentuje správně klasifikované objekty, které jsou zvýrazněny škálou modré barvy. Přesto je patrné, že model nesprávně klasifikoval některé objekty.

```
def get_matrix (in_X_val, in_y_val, class_names):
    y_pre_test = model_tweak_4.predict(in_X_val)
    y_pre_test = np.argmax(y_pre_test, axis=1)
    cm = confusion_matrix(in_y_val, y_pre_test)

    plt.figure(figsize=(15, 15))
    sn.heatmap(cm, annot=True, cmap='Blues')

    class_names_matrix = class_names
    plt.xticks(np.arange(len(class_names_matrix)), class_names_matrix, rotation=90)
    plt.yticks(np.arange(len(class_names_matrix)), class_names_matrix, rotation=0)

    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title('Confusion Matrix')
    plt.show()
    return cm, y_pre_test
```

Obrázek 35 – Confusion Matrix



Zdroj: Vlastní zpracování

Analýzy Confusion matrix může poskytnout podrobný vhled, proč model dosáhl úspěšnosti pouze 96 % při klasifikaci obrazu.

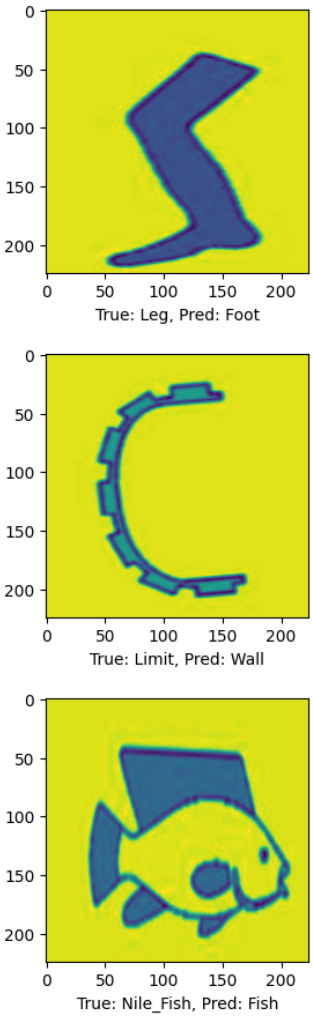
Skript zpracovaný níže je navržen pro rozpoznání a prezentaci skupin, které byly chybně označeny překračující určitý limit. Inicializovaný práh je nastaven na hodnotu 5, což naznačuje, že skript se zaměřuje na skupiny, u nichž došlo k nesprávnému označení vícekrát, než je tento limit.

Skript konstruuje pole korektně klasifikovaných příkladů pro každou skupinu využitím diagonály konfuzní matice. Následně generuje pole nesprávně označených příkladů pro každou skupinu, a to odečtením počtu správně klasifikovaných příkladů od celkového počtu příkladů pro danou skupinu.

Poté skript formuje seznam skupin, které byly nesprávně označeny častěji, než je stanovený limit. Pokud existují takové případy, skript generuje obrázek s popiskem osy se skutečnou a predikovanou třídou.

```
limit = 5

correctly_classified = np.diag(conf_matrix)

incorrectly_classified_counts = np.sum(conf_matrix, axis=1) - correctly_classified
incorrectly_classified_classes = [train_class_names[i] for i in
range(len(train_class_names))
                                  if incorrectly_classified_counts[i] >= limit]
print('incorrect identified classes more like', limit, incorrectly_classified_classes)

for incorrect_class in incorrectly_classified_classes:
    incorrect_indices = [i for i, is_incorrect in enumerate((y_pred_test - y_val != 0)
                         & (y_val == train_class_names.index(incorrect_class)))
                         if is_incorrect]
    if len(incorrect_indices) > 0:
        fig,ax=plt.subplots(1,1,sharey=False,figsize=(3,3))
        fig.tight_layout()
        random_index = np.random.choice(incorrect_indices)

        true_class = train_class_names[y_val[random_index]]
        pred_class = train_class_names[y_pred_test[random_index]]

        ax.imshow(X_val[random_index][:, :, 1])
        ax.set_xlabel('True: {}, Pred: {}'.format(true_class, pred_class))
        plt.show()
```

Obrázek 36 – Špatně predikované třídy

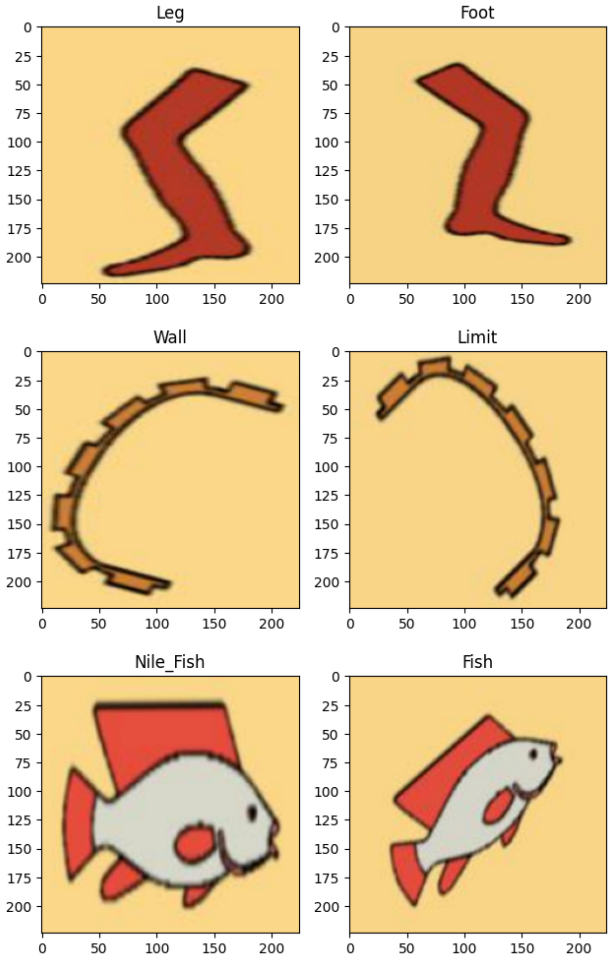


Zdroj: Vlastní zpracování

Průzkum dat ukázal, že nejvíce problematické jsou obrázky se štítky 'Foot', 'Wall' a 'Fish'. Proto byla vytvořena funkce show_inccorect_correct_img, která je navržena tak, aby umožnila přímé porovnání dvou správně a nesprávně klasifikovaných příkladů, náhodně vybraných z trénovacího datasetu s popisem s předchozího průzkum.

Z vizualizace (obrázek 37) je patrné, že některé obrázky, které byly nesprávně klasifikovány, jsou ve skutečnosti stejné jako ty, které byly klasifikovány korektně. Hlavní rozdíl mezi těmito obrázky spočívá v jejich anotaci. Tento nález naznačuje, že bude nutná další optimalizace zaměřená právě na tyto třídy.

Obrázek 37 – Obrázky z nesprávně a správně klasifikované třídy



Zdroj: Vlastní zpracování

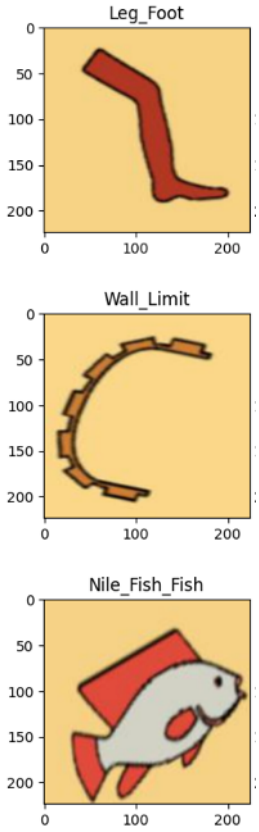
Ačkoliv obrázky vypadají stejně, mohou být interpretovány různými způsoby. Jako řešení problému se nabízí implementování funkce merge_classes, která sjednotí tyto obrázky do jedné třídy. Tímto způsobem tak vzroste schopnost modelu správně klasifikovat třídy.

Jak už bylo nastíněno. Funkce merge_classes je implementována s cílem sjednotit specifické třídy v datasetu. Přijímá jako vstupní parametry seznam popisků obrázků (dataset_labels) a seznam samotných obrázků (dataset_data). Dále definuje slovník merge_classes, který mapuje původní názvy tříd na nové názvy tříd, do kterých mají být původní třídy sloučeny. Pro každý obrázek v datasetu zjistí, zda třída má být sloučena, a pokud ano, přiřadí obrázku nový název. Pokud nový název třídy není definován ve slovníku, ponechá se původní označení.

Kategorie 'Leg' a 'Foot' byly sloučeny do třídy 'Leg_Foot'. 'Wall' a 'Limit' jsou nově sjednoceny do názvu ‘Wall_Limit'. A 'Nile_Fish' a 'Fish' náleží označení 'Nile_Fish_Fish'. Tímto způsobem byl počet tříd v datasetu redukován z původních 95 na 92.

```
def merge_classes(dataset_labels, dataset_data):
    new_labels = []
    new_images = []
    new_dataset_labels = []
    new_dataset_data = []

    merge_classes = {'Leg': 'Leg_Foot', 'Foot': 'Leg_Foot',
                     'Wall': 'Wall_Limit', 'Limit': 'Wall_Limit',
                     'Nile_Fish': 'Nile_Fish_Fish',
                     'Fish': 'Nile_Fish_Fish'}

    for m_label, m_image in zip (dataset_labels, dataset_data):
        new_label = merge_classes.get(m_label, m_label)
        new_labels.append(new_label)
        new_images.append(m_image)

    new_dataset_labels = np.array(new_labels)
    new_dataset_data = np.array(new_images)
    return new_dataset_labels, new_dataset_data
```

Obrázek 38 – Sjednocené třídy



Zdroj: Vlastní zpracování

Z výše uvedených důvodů bylo nutné znovu indexovat anotaci a upravit dataset dle předešlých výsledků, tzn. úprava originálních objektů na oříznuté, normalizované a argumentované.

Na znovu připraveném datasetu byl zpuštěn trénovací model 4. Díky korekci dat lze říct, že dosáhl výborných výsledků. Ztrátová funkce je 0,0097 a přesnost modelu 99,69 %.

Obrázek 39 – Průběh evaluace klasifikačního modelu na validačních datech

```
model_tweak_4.evaluate(merge_val_data, steps= len(merge_val_data))
31/31 [==============================] - 0s 9ms/step - loss: 0.0097 - accuracy: 0.9969
[0.009735777974128723, 0.9969167709350586]
```

Zdroj: Vlastní zpracování

4.4 Ověření predikce na testovacích datech

Výsledná predikce je provedena na testovacích datech. Ta zjistí, jak je model schopen generalizovat nová data. Stejně jako u trénovacích dat je potřeba, aby vstupy do modelu byly stejné. Z předchozích kroků je jasné, že je třeba nejprve provést normalizaci dat a nastavit generátor, který poskytne stejné sady jako u tréninkových dat. Nutné je i upravit anotace z textové podoby do číselné.

Ve finální fázi došlo ještě na úpravu testovacího souboru. Po spuštění funkce na vyhodnocení modelu **evaluate()** na testovacích datech, program hlásil chybu, z důvodu nálezu textového řetězec v souboru. Bylo nevyhnutelné znovu projít testovací data a upravit tak, aby byl model schopen spustit predikci.

Chyba byla zjištěna v anotaci obrázků za pomocí skriptu, který prošel již indexovanou anotaci. V tomto případě nezbylo nic jiného než si vypsat, o jaký případ jde. Na obrázku 40 je vidět, že většina tříd je indexovaná správně až na případ „Head“.

Obrázek 40 – Kontrolní skript pro hledaní datového typu string v datech

```
for idx, item in enumerate(test_labels_index):
    if isinstance(item, str):
        print(f"On indexu {idx} is string: {item}")
```

```
On indexu 154 is string: 28
On indexu 155 is string: 11
On indexu 156 is string: 31
On indexu 157 is string: 71
On indexu 158 is string: 47
On indexu 159 is string: Head
On indexu 160 is string: 81
On indexu 161 is string: 26
On indexu 162 is string: 58
On indexu 163 is string: 60
On indexu 164 is string: 7
On indexu 165 is string: 63
```

Zdroj: Vlastní zpracování

V testovacím souboru se nacházel obrázek s třídou, která nebyla definována v trénovacím souboru, proto byla tato třída s příslušnými obrázky vyňata ze souboru. Konečný počet dat je 1039.

Výsledky hodnocení modelu 4 na souboru final_test_data ukazují vynikající výkonnost. Model dosáhl hodnoty ztráty 0,0073 a přesnosti 0,9990.

Hodnota ztráty je velmi nízká, a to jen potvrzuje výsledky z trénování. Z toho vyplývá, že model má minimální chybovost při predikci tříd egyptských hieroglyfů a výsledek 0,0073 je výjimečně dobrý.

Přesnost modelu je téměř 1. Model správně klasifikoval téměř 100 % testovacích dat. Vysoká úroveň přesnosti ukazuje, že model je velmi spolehlivý při rozpoznávání a může být účinně využit pro další výzkum nebo aplikace v tomto oboru.

Obrázek 41 – Průběh evaluace modelu na testovacích datech

```
model_tweak_4.evaluate(final_test_data)
```

```
33/33 [==============================] - 0s 13ms/step - loss: 0.0073 - accuracy: 0.9990
[0.007278237957507372, 0.9990375638008118]
```

Zdroj: Vlastní zpracování

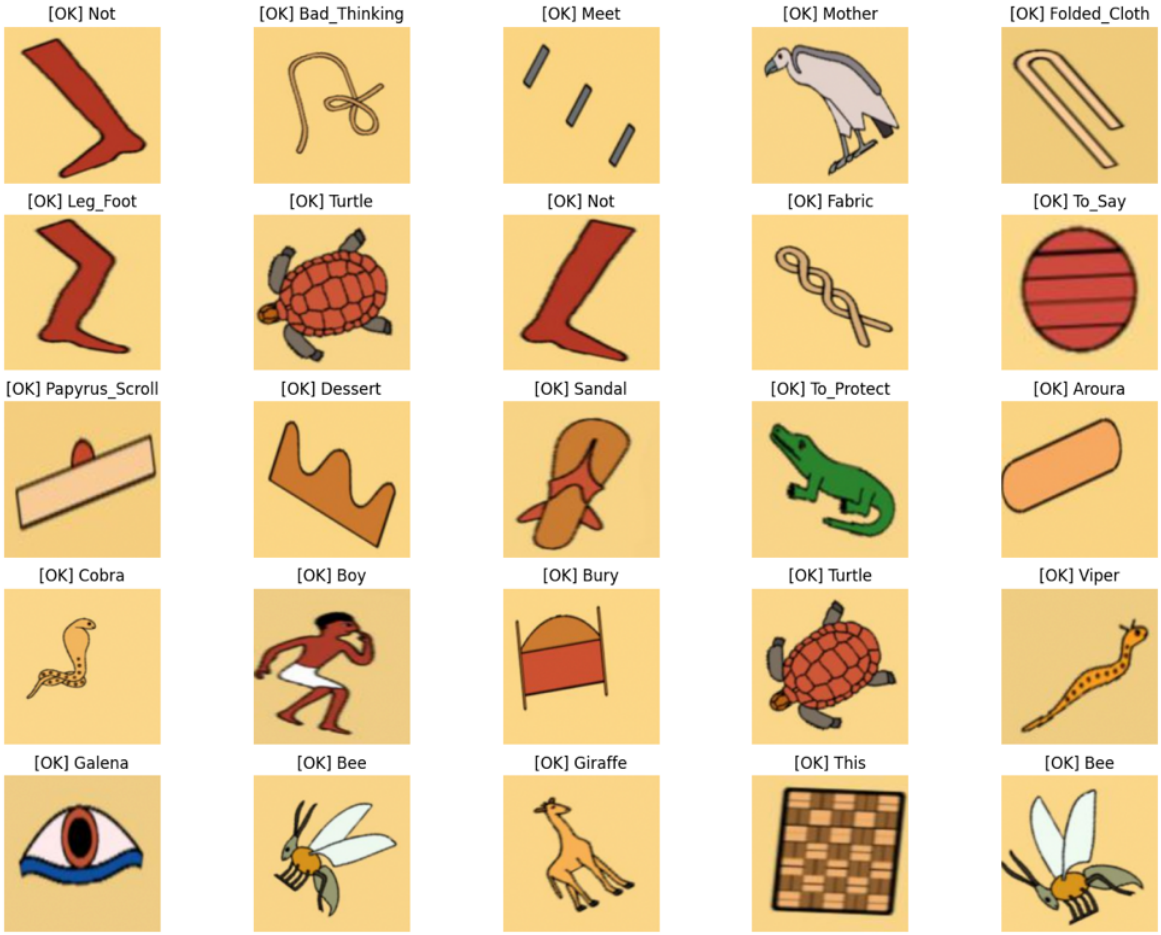
Kód zobrazený níže je nástrojem pro ověření schopnosti modelu správně klasifikovat obrázky, které jsou náhodně vybrány z testovací sady. Poskytuje vizuální sumarizaci výsledků a umožňuje detekci potenciálních chyb v predikcích, současně poskytuje informace o reálné výkonnosti modelu.

Skript generuje mřížku sestávající z 25 obrázků. Jak již bylo zmíněno, náhodně vybírá vzorky z testovacích dat. Každý vybraný obrázek je poté předložen modelu k analýze. Model stanoví třídu s nejvyšší pravděpodobností a skript poté ověří, zda se predikovaná třída shoduje se skutečnou.

Pokud je shoda potvrzena, titulek obrázku zobrazí název třídy s indikátorem OK. Pokud se třídy neshodují, název predikované třídy je zvýrazněn červenou barvou. Tento postup je aplikován na každý obrázek v mřížce. Výsledkem je vizualizace 25 náhodně vybraných obrázků z testovacích dat a jejich předpovědi.

```
fig, ax = plt.subplots(5, 5)
fig.subplots_adjust(0, 0, 2, 2)
for i in range(0, 5):
   for j in range(0, 5):
      random_number = random.randint(0, len(merge_test_data))
      pred_image = np.array([merge_test_data[random_number]])
      predictions = model_tweak_4.predict(pred_image)[0]
      pred_class = np.argmax(predictions)
      act = merge_test_classes[test_labels_index[random_number]]

      ax[i, j].imshow(merge_test_data[random_number])
      if(merge_test_classes[pred_class] !=
merge_test_classes[test_labels_index[random_number]]):
         title = '{} [{}]'.format(merge_test_classes[pred_class],
                                  merge_test_classes[test_labels_index[random_number]])
         ax[i, j].set_title(title, fontdict={'color': 'darkred'})
      else:
         title = '[OK] {}'.format(merge_test_classes[pred_class])
         ax[i, j].set_title(title)
      ax[i, j].axis('off')
```

Obrázek 42 – Predikované obrázky s označením správnosti klasifikace



Zdroj: Vlastní zpracování

5 Výsledky a diskuse

5.1 Příprava datasetu

První krok výzkumu spočíval v průzkumu a rozdělení datové sady. Datová sada byla komplexní a obsahovala soubory ve formátu JPG a XML. Každý obrázek byl doprovázen jedním souborem XML, který obsahoval anotace k obrázku.

Pro kontrolu správného přiřazení mezi obrázky a jejich odpovídajícími anotacemi byl vytvořen skript. Ten byl navržen tak, aby náhodně vybíral obrázky z datové sady a zobrazoval je spolu s příslušnými anotacemi z odpovídajících souborů XML.

Kontrolou funkčnosti bylo i sepsání skriptu, který náhodně vybral ze souboru obrázky s anotacemi a výsledkem bylo jejich zobrazení.

5.1.1 Rozdělení datasetu

Dalším krokem v procesu bylo rozdělení tréninkové datové sady, protože sada s testovacími daty již byla oddělena nebyla potřeba rozdělování i na něj. Nicméně soubor s tréninkovými daty v celkovém počtu 4937 datových jednotek byl rozdělen na sadu tréninkovou a validační, a to následujícím způsobem: Pro tréninkovou sadu bylo vybráno 3889 jednotek (75 %) a zbývajících 1048 (25 %) jednotek bylo použito pro validační sadu.

5.1.2 Identifikace tříd a rozdělení obrázků

Po rozdělení datové sady bylo provedeno další důležité zkoumání dat – identifikace počtu tříd. Třídy v tomto kontextu představují různé kategorie, do kterých mohou data spadat. Bylo definováno celkem 95 různých tříd.

Vytvoření histogramu umožnilo vizualizovat rozložení dat mezi třídami a poskytlo důležité informace pro další fáze výzkumu. Například pokud by bylo zjištěno, že některé třídy mají mnohem více obrázků než ostatní, mohlo by to ovlivnit rozhodnutí o dalším postupu, jako je potřeba vyvážení tříd před tréninkem modelu.

5.1.3 Úprava vstupů

Po identifikaci tříd a analýze rozložení dat byla provedena další klíčová fáze přípravy dat – úprava obrázků. Obrázky v datové sadě byly modifikovány tak, aby byly vhodné jako vstupní parametr pro model.

První úpravou byla změna velikosti obrázků. Cílová velikost byla nastavena na 224x224 pixelů za pomocí funkce resize, která je součástí knihovny OpenCV. Druhou úpravou bylo nastavení barevného prostoru obrázků do RGB prostřednictvím funkce cv2.COLOR_BGR2RGB, která je také součástí knihovny OpenCV. Nutností byl i převod anotací obrázků do číselné formy.

5.1.4 Prototyp modelu

Následně byl vytvořen první prototyp modelu. Vstupní vrstva měla parametry, jak už bylo řečeno, 224x224 se 3kanály (RGB), kterému byly k trénování předkládány různě upravené vstupní obrázky.

Struktura modelu:

Tabulka 7 – Architektura prototypu modelu

Typ vrstvy	Parametry
Conv2D	32 filtrů, jádro 3x3, aktivační funkce ReLU
MaxPooling2D	velikost poolu 2x2
Conv2D	64 filtrů, jádro 3x3, aktivační funkce ReLU
MaxPooling2D	velikost poolu 2x2
Conv2D	128 filtrů, jádro 3x3, aktivační funkce ReLU
MaxPooling2D	velikost poolu 2x2
Conv2D	128 filtrů, jádro 3x3, aktivační funkce ReLU
MaxPooling2D	velikost poolu 2x2
Conv2D	256 filtrů, jádro 3x3, aktivační funkce ReLU
MaxPooling2D	velikost poolu 2x2
Flatten	-
Dense	512 neuronů, aktivační funkce ReLU
Dense	95 neuronů, aktivační funkce softmax

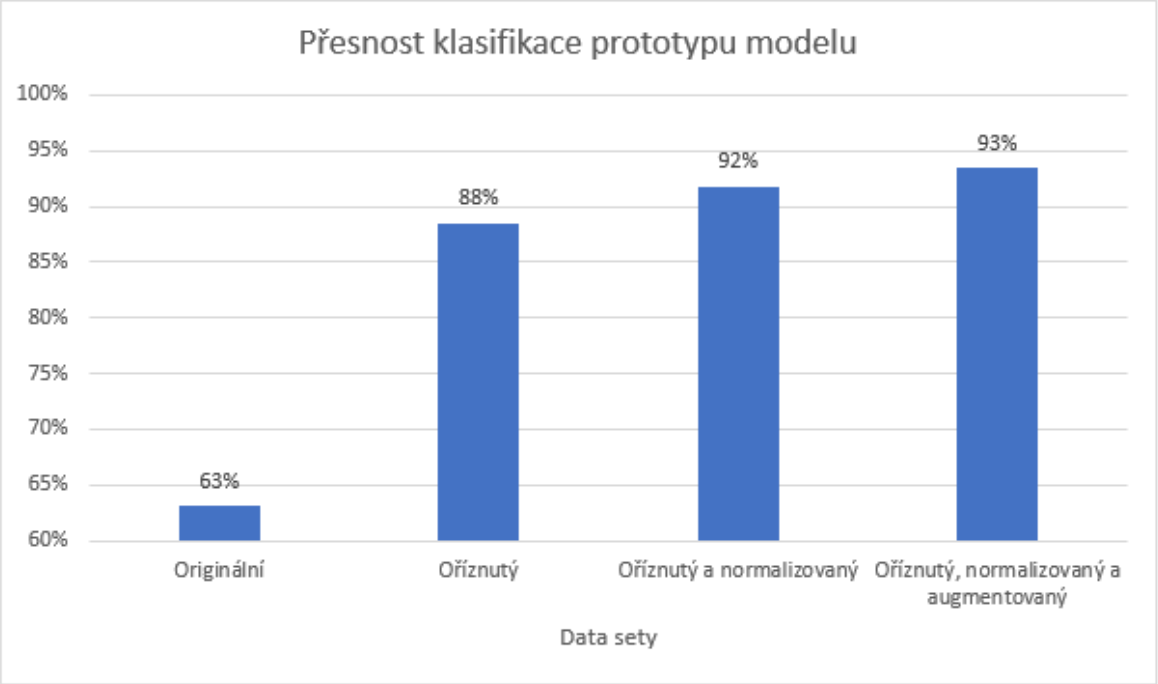
Zdroj: Vlastní zpracování

V průběhu experimentu byl prototyp modelu postupně trénován na různých typech dat:

- Originální data: Výchozí fáze, kde byl model trénován na původních, neupravených datech.
- Oříznutá data: fáze, kdy byla data oříznuta, aby se odstranily nepotřebné nebo nežádoucí části, což pomohlo modelu lépe se soustředit na relevantní rysy.
- Oříznutá a normalizovaná data: Po oříznutí byla data také normalizována. Díky tomu všechny vstupní rysy dostanou stejný rozsah hodnot.
- Oříznutá, normalizovaná a augmentovaná data: V poslední fázi byla data nejen oříznuta a normalizována, ale také augmentována. Tím bylo zajištěno rozšíření tréninkové datové sady.

Každá fáze experimentu představovala další úroveň předzpracování dat, která měla za cíl zlepšit výkon modelu.

Obrázek 43 – Přesnost klasifikace prototypu modelu na různě upravených datasetech



Zdroj: Vlastní zpracování

5.2 Optimalizace a ladění modelu

Došlo k ladění prototypu, aby dosahoval co nejlepších výsledků. Postupně došlo k vytvoření čtyř modelů.

5.2.1 Model 1

Model 1 představoval základní strukturu s dvěma Dropout vrstvami o hodnotě 0,4 a první Dense vrstvou s 1024 neurony.

Tabulka 8 – Architektura modelu 1

Typ vrstvy	Parametry
Conv2D	32 filtrů, jádro 3x3, aktivační funkce ReLU
MaxPooling2D	velikost poolu 2x2
Conv2D	64 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
MaxPooling2D	velikost poolu 2x2
Conv2D	128 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
MaxPooling2D	velikost poolu 2x2
Conv2D	128 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
MaxPooling2D	velikost poolu 2x2
Conv2D	256 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
MaxPooling2D	velikost poolu 2x2
Dropout	0.4
Flatten	-
Dense	1024 neurony, aktivační funkce ReLU
Dropout	0.4
Dense	95 neuronů, aktivační funkce softmax

Zdroj: Vlastní zpracování

5.2.2 Model 2

Model 2 byl podobný modelu 1, ale s upravenými parametry optimalizátoru Adam, konkrétně s rychlostí učení 0,0001 a hodnotou weight decay 1e-6.

5.2.3 Model 3

Model 3 přinesl změnu v architektuře konvolučních vrstev, kde byly přidány další konvoluční vrstvy pro filtry 128, 256 a nově 512.

Tabulka 9 – Architektura modelu 3

Typ vrstvy	Parametry
Conv2D	32 filtrů, jádro 3x3, aktivační funkce ReLU
MaxPooling2D	velikost poolu 2x2
Conv2D	64 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
MaxPooling2D	velikost poolu 2x2
Conv2D	128 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
Conv2D	128 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
MaxPooling2D	velikost poolu 2x2
Conv2D	256 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
Conv2D	256 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
MaxPooling2D	velikost poolu 2x2
Dropout	0.4
Conv2D	512 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
Conv2D	512 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
MaxPooling2D	velikost poolu 2x2
Dropout	0.4
Flatten	-
Dense	1024 neurony, aktivační funkce ReLU
Dropout	0.4
Dense	95 neuronů, aktivační funkce softmax

Zdroj: Vlastní zpracování

5.2.4 Model 4

Model 4 pak snížil hodnotu Dropout vrstvy na 0,2, přidal další konvoluční vrstvy s 256 filtry, další Dense vrstvu s 1024 neurony a implementoval vrstvu Batch Normalization.

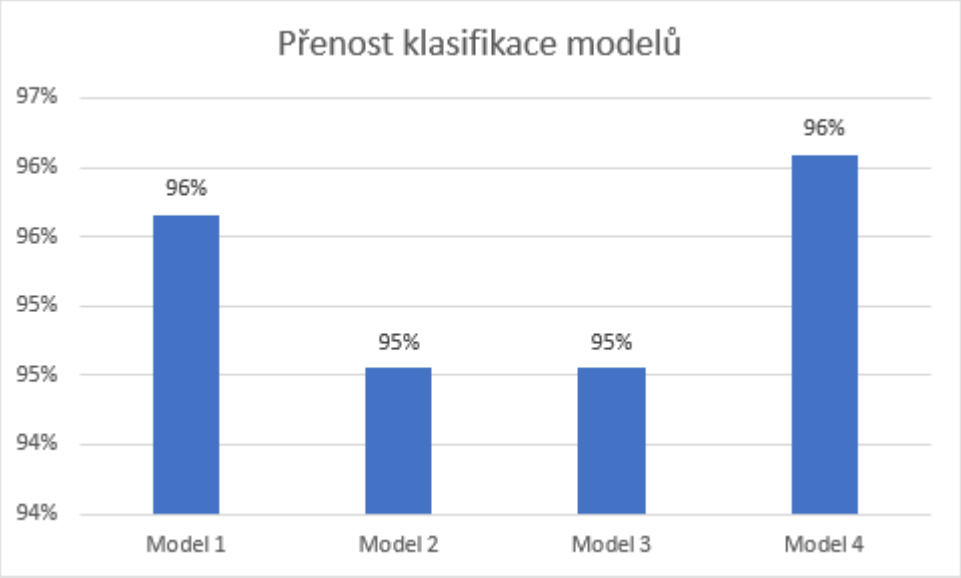
Tabulka 10 – Architektura modelu 4

Typ vrstvy	Parametry
Conv2D	32 filtrů, jádro 3x3, aktivační funkce ReLU
MaxPooling2D	velikost poolu 2x2
Conv2D	64 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
MaxPooling2D	velikost poolu 2x2
Conv2D	128 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
Conv2D	128 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
MaxPooling2D	velikost poolu 2x2
Conv2D	256 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
Conv2D	256 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
MaxPooling2D	velikost poolu 2x2
Dropout	0.2
Conv2D	256 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
Conv2D	256 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
MaxPooling2D	velikost poolu 2x2
Dropout	0.2
Conv2D	512 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
Conv2D	512 filtrů, jádro 3x3, aktivační funkce ReLU, padding "same"
MaxPooling2D	velikost poolu 2x2
Dropout	0.2
Flatten	-
Dense	1024 neurony, aktivační funkce ReLU
Dense	1024 neurony, aktivační funkce ReLU
BatchNormalization	-
Dropout	0.2
Dense	95 neuronů, aktivační funkce softmax

Zdroj: Vlastní zpracování

Tyto postupné úpravy vedly k vylepšení výkonu modelů, avšak zároveň zvyšovaly nároky na výpočetní zdroje a čas tréninku.

Obrázek 44 – Přesnost klasifikace vylepšených modelů



Zdroj: Vlastní zpracování

5.3 Evaluace výsledků na validačních datech

V průběhu experimentu byla použita matice záměn (Confusion Matrix) pro vizualizaci výsledků klasifikace modelu 4. To umožnilo identifikovat obrázky, které model klasifikoval nesprávně. Zjistilo se, že některé obrázky byly ve skutečnosti stejné, ale měly různé anotace. To vedlo k rozhodnutí sjednotit některé třídy:

- 'Leg' a 'Foot' byly sloučeny do třídy 'Leg_Foot',
- 'Wall' a 'Limit' byly sjednoceny pod názvem 'Wall_Limit',
- 'Nile_Fish' a 'Fish' byly sjednoceny pod označením 'Nile_Fish_Fish'.

Tímto způsobem byl počet tříd v datasetu redukován z původních 95 na 92. Po této korekci datasetu byl model znovu trénován. Díky těmto úpravám dosáhl model přesnosti 99,69 %, což ukazuje, že sjednocení tříd vedlo k výraznému zlepšení výkonu modelu.

5.4 Ověření predikce na testovacích datech

V závěrečné fázi byla provedena predikce na testovacích datech, aby se zjistilo, jak dobře model generalizuje nová data. Stejně jako u tréninkových dat bylo nutné zajistit stejné vstupy do modelu, což zahrnovalo úpravu velikosti, normalizaci dat a převod anotací z textové formy na číselnou.

Během finálního vyhodnocení modelu na testovacích datech došlo k chybě kvůli přítomnosti textového řetězce v souboru. Bylo tedy nutné znovu procházet testovací data a upravit je tak, aby bylo možné spustit evaluaci.

Chyba byla identifikována v anotaci obrázků pomocí skriptu, který procházel indexovanou anotaci. Bylo zjištěno, že většina tříd byla správně indexována, s výjimkou třídy „Head". Tento obrázek a jeho třída byly z testovacího souboru odstraněny, což vedlo ke konečnému počtu 1039 obrázků.

Model 4 vykázal vynikající výsledky, dosahujíc hodnoty ztráty 0,0073 a přesnosti 0,9990. Přesnost modelu je vynikající, vykazuje téměř 100 % správnou klasifikací testovacích dat. Vysoká úroveň přesnosti ukazuje, že model je velmi spolehlivý a může být účinně využit pro další výzkum nebo aplikace v tomto oboru.

5.5 Diskuse

Cílem této práce bylo vyvinout model rozpoznávající egyptské hieroglyfy pomocí konvoluční neuronové sítě. Tento cíl se podařilo splnit s pozitivními výsledky, neboť Model 4 vykazuje vynikající přesnost. V průběhu vývoje modelu se však vyskytly některé nežádoucí jevy, které jsou popsány výše a které bylo třeba odstranit skrze neočekáváný přídatný postup. Z hlediska celkové funkčnosti a architektury modelu se domnívám, že skrze vylepšení by mohl být konvoluční model solidním základem pro další vývoj modelu. Pro porovnání byly vybrány dva CNN modely, pro který byl použit obdobný dataset z platformy Kaggle.

5.5.1 Komparativní analýza modelů

První porovnaný model s přesností 95 % byl vytvořen uživatelem Soosmann. Při úpravě obrázků byla aplikována normalizace a ořez. Nicméně nebyla použita augmentace pro umělé navýšení datasetu. Při anotaci, kdy byl text převáděn na čísla, byla využita one-hot metoda, které je preferovanější místo indexace jednotlivých tříd. Soosmann se také snažil optimalizovat model, upravoval learning rate a jako optimalizátor použil SGD místo

optimalizátoru Adam. Jeho model by trénován po dobu 30 epoch. Neprovedl ale vizuální kontrolu, například za pomocí confusion matice, po natrénování. Tato kontrola by pomohla odhalit potřebu sjednocení některých tříd, což by zaručeně vedlo k lepšímu výsledku klasifikace. (39)

Porovnaný druhý sekvenční model konvolučních neuronových sítí byl vytvořen uživatelem s přezdívkou arturo-bandini-jr. Ten využil předtrénovaný model DenseNet169 s váhami z ImageNetu jako první vrstvu v sekvenčním modelu. Za ní následovaly další vrstvy, včetně GlobalAveragePooling2D a několika plně propojených (Dense) vrstev. Model je kompilován optimalizátorem Adam, s funkcí ztráty sparse_categorical_crossentropy a metrikou accuracy. Trénink probíhal na trénovacích datech po dobu pouhých 8 epoch. A to je v kontextu s učením konvolučních sítí značně omezený počet. Pro optimalizaci procesu trénování bylo použito zpětné volání EarlyStopping, které zastaví trénování, pokud se validační ztráta nezlepší o více než 0,0001 po dobu dvou po sobě jdoucích epoch. Hodnota pro zastavení trénování je výrazně nízká a může vést k předčasnému ukončení tréninku. Nicméně výsledky predikce dosahují pouze 85 %. V porovnání s předchozími modely jde o méně uspokojivý výsledek. Navíc zde nejsou naznačeny žádné další experimenty ani ladění modelu, což značně omezilo jeho potenciální výkonnost. (40)

5.5.2 Rozšíření aplikace modelu

Pro další praktické a náročnější uplatnění výsledného modelu se nabízí jeho využití jako základ pro aplikaci v oblasti detekce objektů, tzv. object detection. Místo toho, aby model pouze klasifikoval jednotlivé hieroglyfy, by mohl být rozšířen o detekci objektů. To by umožnilo modelu identifikovat a lokalizovat hieroglyfy v rámci většího obrazu. A bylo by užitečné pro analýzu skutečných archeologických nálezů.

Domnívám se, že pro účely této práce je daná architektura zcela dostačující. Pochopitelným a logickým prvkem je zejména zlepšení robustnosti celého modelu. Model by mohl být dále vylepšen tak, aby byl schopen lépe se vypořádat s různými styly psaní, kvalitou obrazu, osvětlením a dalšími faktory, které mohou ovlivnit kvalitu rozpoznávání.

Dalším upgradem, který by zahrnoval výše zmíněné vylepšení, by mohla být integrace do mobilní aplikace. Souvisejícím nástrojem by také mohl být komplexní překlad. Uživatel by použit svoje mobilní zařízení s fotoaparátem, naskenoval hieroglyfy a aplikace by je automaticky přeložila. Kromě rozpoznávání jednotlivých hieroglyfů by tak bylo možné vyvinout systém, který by dokázal přeložit celé věty nebo pasáže textu.

Touto aplikací by pak byl naplněn přínos nastíněný již v samém úvodu, kdy by se jednalo o zajímavé propojení historie a moderní technologie. Tak by byl zajisté zvýšen zájem o historické relikty a tím pádem by mohl být nápomocný i v oblasti vědy a zároveň by mohl urychlit výzkumné metody.

6 Závěr

Cílem této práce bylo vyvinout model rozpoznávající egyptské hieroglyfy pomocí konvoluční neuronové sítě. Tento cíl byl úspěšně splněn, konečný model vykázal vynikající přesnost. Během vývoje modelu se však vyskytly některé nežádoucí jevy, které bylo třeba odstranit.

V průběhu dosažení kýžených výsledků byla provedena řada kroků vedoucích k vytvoření a optimalizaci modelu pro klasifikaci obrázků. Začínalo se průzkumem a rozdělením komplexní datové sady, která obsahovala soubory ve formátu JPG a XML.

Datová sada byla následně rozdělena na tréninkovou a validační sadu, přičemž bylo identifikováno celkem 95 různých tříd. Obrázky v datové sadě byly upraveny pro optimální vstup do modelu, včetně změny velikosti a barevného prostoru. Model byl postupně trénován na různých typech dat, včetně dat originálních, oříznutých, normalizovaných a augmentovaných.

V další fázi byla provedena optimalizace a ladění modelu, což vedlo k vytvoření čtyř různých modelů s postupnými úpravami. Ty vedly k vylepšení výkonu modelů, avšak zároveň zvyšovaly nároky na výpočetní zdroje a čas tréninku.

V průběhu experimentu byla provedena evaluace výsledků na validačních datech, přičemž byla použita matice záměn pro vizualizaci výsledků klasifikace. Bylo zjištěno, že některé obrázky byly ve skutečnosti stejné, ale měly různé anotace, což vedlo k rozhodnutí sjednotit některé třídy. Výsledný počet tříd se snížil, a to na hodnotu 92. Po této korekci byl model znovu trénován a dosáhl přesnosti 99, 69 %.

V závěrečné fázi byla provedena predikce na testovacích datech. Během finálního vyhodnocení modelu na testovacích datech došlo k chybě vzhledem k přítomnosti textového řetězce v souboru. Po odstranění chyby model vykázal vynikající výsledky dosahující hodnoty ztrátové funkce 0,0073 a přesnosti kvalifikace obrazu téměř 100 %.

Tato vysoká úroveň přesnosti ukazuje, že model je velmi spolehlivý a může být účinně využit pro další výzkum nebo aplikaci. Výsledky diplomové práce představují solidní základ pro další výzkum a vývoj v oblasti automatického rozpoznávání egyptských hieroglyfů.

7 Seznam použitých zdrojů

(1)
AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the difference? Online. In: Ibm. 2023. Dostupné z: https://www.ibm.com/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks/. [cit. 2024-03-20].

(2)
What is Deep Learning? Online. In: Oracle. 2023. Dostupné z: https://www.oracle.com/hk/artificial-intelligence/machine-learning/what-is-deep-learning/. [cit. 2024-03-20].

(3)
MILLSTEIN, Frank. Convolutional Neural Networks In Python. Online. 2020. CreateSpace Independent Publishing Platform, 2020. Dostupné z: https://www.google.cz/books/edition/Convolutional_Neural_Networks_In_Python/a3nvDwAAQBAJ?hl=cs&gbpv=1. [cit. 2024-03-20].

(4)
CHOLLET, François. Deep learning v jazyku Python: knihovny Keras, Tensorflow. 2019. Knihovna programátora (Grada). Praha: Grada Publishing, 2019. ISBN 978-80-247-3100-1.

(5)
AGGARWAL, Charu C. Neural networks and deep learning: a textbook. Second Edition. Cham: Springer Nature Switzerland, [2023]. ISBN 978-3-031-29641-3.

(6)
Machine Learning for Biometrics. Online. 1. Academic Press, 2022. ISBN 978-0-323-85209-8. Dostupné z: https://doi.org/https://doi.org/10.1016/C2020-0-02002-5. [cit. 2024-03-20].

(7)
Layer activation functions. Online. Keras. 2024. Dostupné z: https://keras.io/api/layers/activations/. [cit. 2024-03-20].

(8)
Underfitting and Overfitting in Machine Learning. Online. TOKUÇ, A. Aylin a AIBIN, Michal. Baeldung. 2024. Dostupné z: https://www.baeldung.com/cs/ml-underfitting-overfitting. [cit. 2024-03-20].

(9)
GÉRON, Aurélien. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems. Third edition. Sebastopol, CA: O'Reilly, 2022. ISBN 978-1-098-12597-4.

(10)
Understanding Loss Function in Deep Learning. Online. Analyticsvidhya. 2023. Dostupné z: https://www.analyticsvidhya.com/blog/2022/06/understanding-loss-function-in-deep-learning/. [cit. 2024-03-20].

(11)
Gradient Descent in Machine Learning. Online. Javatpoint. 2021. Dostupné z: https://www.javatpoint.com/gradient-descent-in-machine-learning. [cit. 2024-03-20].

(12)
What is gradient descent? Online. IBM. 2022. Dostupné z: https://www.ibm.com/topics/gradient-descent. [cit. 2024-03-20].
Performance Metrics in Machine Learning. Online. Javatpoint. 2021. Dostupné z: https://www.javatpoint.com/performance-metrics-in-machine-learning. [cit. 2024-03-20].

(13)
Metrics. Online. Tensorflow. 2024. Dostupné z: https://www.tensorflow.org/api_docs/python/tf/keras/metrics. [cit. 2024-03-20].

(14)
Confusion Matrix. Online. Sciencedirect. Dostupné z: https://www.sciencedirect.com/topics/engineering/confusion-matrix. [cit. 2024-03-20].

(15)
SKIENA, Steven S. The data science design manual. 2017. Texts in computer science. Cham, Switzerland: Springer, [2017]. ISBN 978-3-319-55443-3.

(16)
Regression Analysis in Machine learning. Online. Javatpoint. 2021. Dostupné z: https://www.javatpoint.com/regression-analysis-in-machine-learning. [cit. 2024-03-20].

(17)
GÉRON, Aurélien. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems. Third edition. Sebastopol, CA: O'Reilly, 2022. ISBN 978-1-098-12597-4.

(18)
YADAV, Dinesh. Categorical encoding using Label-Encoding and One-Hot-Encoder. Online. Towardsdatascience. 2019. Dostupné z: https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd. [cit. 2024-03-20].

(19)
KUHN, Max a JOHNSON, Kjell. Applied predictive modeling. 2013. New York: Springer, 2013. ISBN 978-1-4614-6848-6.

(20)
BROWNLEE, Jason. What is the Difference Between Test and Validation Datasets? Online. Machinelearningmastery. 2020. Dostupné z: https://machinelearningmastery.com/difference-test-validation-datasets/. [cit. 2024-03-20].

(21)
From LeNet to EfficientNet: The evolution of CNNs. Online. GANESH, Prakhar. Towardsdatascience. 2020. Dostupné z: https://towardsdatascience.com/from-lenet-to-efficientnet-the-evolution-of-cnns-3a57eb34672f. [cit. 2024-03-20].

(22)
KRIZHEVSKY, Alex; SUTSKEVER, Ilya a HINTON, Geoffrey. ImageNet Classification with Deep Convolutional Neural Networks. Online. In: . S. 9. Dostupné z: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf. [cit. 2024-03-20].

(23)
Everything you need to know about VGG16. Online. ROHINI, G. Medium. 2021. Dostupné z: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918. [cit. 2024-03-20].

(24)
HOWARD, Andrew G.; WANG, Weijun; ZHU, Menglong; WEYAND, Tobias; CHEN, Bo et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. Online. In: .S. 9. Dostupné z: https://arxiv.org/pdf/1704.04861.pdf. [cit. 2024-03-20].

(25)
TAN, Mingxing a LE, Quoc. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Online. Arxiv.org. 2020, s. 9. Dostupné z: https://arxiv.org/pdf/1905.11946v5.pdf. [cit. 2024-03-20].

(26)
What is EfficientNet? The Ultimate Guide. Online. POTRIMBA, Petru. Roboflow. 2023. Dostupné z: https://blog.roboflow.com/what-is-efficientnet/. [cit. 2024-03-20].

(27)
TAN, Mingxing a LE, Quoc. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Online. Arxiv.org. 2020, s. 9. Dostupné z: https://arxiv.org/pdf/1905.11946v5.pdf. [cit. 2024-03-20].

(28)
SAGAR, Ram. These Machine Learning Techniques Make Google Lens A Success. Online. SAGAR, Ram. Analyticsindiamag. 2019. Dostupné z: https://analyticsindiamag.com/these-machine-learning-techniques-make-google-lens-a-success/. [cit. 2024-03-20].

(29)
A complete guide on object detection & its use in self-driving cars. Online. SINGH, Sumit. Labellerr. 2022. Dostupné z: https://www.labellerr.com/blog/how-object-detection-works-in-self-driving-cars-using-deep-learning/. [cit. 2024-03-20].
(30)
BREVE, Fabricio Aparecido. COVID-19 detection on Chest X-ray images: A comparison of CNN architectures and ensembles. Online. Sciencedirect. 2022. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0957417422008673?via%3Dihub. [cit. 2024-03-20].

(31)
A Study of CNN and Transfer Learning in Medical Imaging: Advantages, Challenges, Future Scope. Online. MDPI. 2023. Dostupné z: https://www.mdpi.com/2071-1050/15/7/5930. [cit. 2024-03-20].

(32)
Face Id: Deep learning for face recognition. Online. FENJIRO, Youssef. Medium. 2019. Dostupné z: https://medium.com/@fenjiro/face-id-deep-learning-for-face-recognition-324b50d916d1. [cit. 2024-03-20].

(33)
CYBELLIUM LTD. Mastering iOS Security. Online. 1. Cybellium, 2023. ISBN 9798861132169. Dostupné z: https://www.google.cz/books/edition/Mastering_iOS_Security/FjXZEAAAQBAJ?hl=cs&gbpv=0. [cit. 2024-03-20].

(34)
Adrian Albert; Jasleen Kaur a , Marta C. Gonzalez. Using Convolutional Networks and Satellite Imagery to Identify Paterns in Urban Environments at a Large Scale. Online. In: Arxiv. 2017. Dostupné z: https://arxiv.org/pdf/1704.02965.pdf. [cit. 2024-03-20].

(35)
Nguyen, G., Dlugolinsky, S., Bobák, M. et al. Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey. Artif Intell Rev 52, 77–124 (2019). https://doi.org/10.1007/s10462-018-09679-z

(36)
Introduction to Tensors. Online. Tensorflow. 2024. Dostupné z: https://www.tensorflow.org/guide/tensor. [cit. 2024-03-20].

(37)
Creation of virtual environments. Online. Python. 2024. Dostupné z: https://docs.python.org/3/library/venv.html. [cit. 2024-03-20].

(38)
Jupyter Notebook Documentation. Online. Jupyter. 2024. Dostupné z: https://jupyter-notebook.readthedocs.io/en/latest/. [cit. 2024-03-20].

(39)
Hieroglyphs | Image Cropping + NeuralNet (95% Acc). Online. Kaggle. 2023. Dostupné z: https://www.kaggle.com/code/soosmann/hieroglyphs-image-cropping-neuralnet-95-acc. [cit. 2024-03-28].

(40)
Hieroglyph Classification DenseNet (0.85 F1). Online. Kaggle. 2023. Dostupné z: https://www.kaggle.com/code/banddaniel/hieroglyph-classification-densenet-0-85-f1/notebook. [cit. 2024-03-28].

8 Seznam obrázků, tabulek, grafů a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

Přílohy

Příloha A Requirment.txt

Příloha B Skript_klasifikace.ipynb