**STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST**

**Obor č.18: Informatika**

Hluboké učení pro rozpoznání obrázků

**Tadeáš Přikryl**

**Vysočina Jaroměřice nad Rokytnou 2024**

**STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST**

**Obor č.18: Informatika**

**Hluboké učení pro rozpoznání obrázků**

**Deep learning for image classification**

**Autoři:** Tadeáš Přikryl

**Škola:** Katolické gymnázium Třebíč, Otmarova 30/22, 674 01 Třebíč

**Kraj:** Vysočina

**Konzultant:** Mgr. Lucie Malachová

Jaroměřice nad Rokytnou 2024

**Prohlášení**

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

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

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

V Jaroměřicích nad Rokytnou dne 15. 2. 2024 ………………………………………………

Tadeáš Přikryl

**Poděkování**

Rád bych poděkoval své konzultantce Mgr. Lucii Malachové, která mi pomáhala s korekcí této práce. Velmi oceňuji její rychlou a vstřícnou komunikaci, díky níž jsem mohl vylepšovat svoji práci.

**Anotace**

Moje práce se zabývá naprogramováním modelu hlubokého učení pro rozpoznání obrázků. Postup mé práce se snažím vysvětlit tak, aby i programátor bez zkušeností s umělou inteligencí byl schopný model z replikovat. Pracuji v jazyku Python a využívám JupyterLab. Program se snaží rozpoznat hady a ještěrky na základě vzhledu a určit o co se jedná. V průběhu práce získám data, zpracuji je a podle nich vytvořím model pro rozpoznání obrázků. Při trénování modelu použiji 70 % dat k trénování, 20 % pro validaci a 10 % pro testování. To pomůže k přesnosti modelu, ale to také záleží na množství dat, které posbírám.

**Klíčová slova**

Umělá inteligence, hluboké učení, neuronové sítě, plazy, rozpoznání obrázků

**Annotation**

My work deals with programming a deep learning model for image recognition. I try to explain the process of my work in such a way that even a programmer with no experience in artificial intelligence will be able to replicate the model. I work in Python and use JupyterLab. The program tries to recognize snakes and lizards based on their appearance and determine what they are. In the course of my work, I collect the data, process it, and use it to create a model for image recognition. When training the model I will use 70 % of the data for training, 20 % for validation and 10 % for testing. This will help the accuracy of the model, but it also depends on the amount of data I collect.

**Keywords**

Artificial intelligence, deep learning, neural networks, reptiles, image recognition

Obsah

[Úvod 6](#_Toc161171509)

[1 Příprava Virtuálního Prostředí 7](#_Toc161171510)

[1.1 Nástroje pro vytvoření virtuálního prostředí 7](#_Toc161171511)

[1.1.1 Jupyter Lab 7](#_Toc161171512)

[1.2 Vytvoření virtuálního prostředí 8](#_Toc161171513)

[1.2.1 Vytvoření místa 8](#_Toc161171514)

[1.2.2 Vytvoření prostředí 8](#_Toc161171515)

[2 Dataset 10](#_Toc161171516)

[2.1 Vytvoření datasetu 10](#_Toc161171517)

[2.2 Třídění 11](#_Toc161171518)

[2.3 Umístění datasetu 11](#_Toc161171519)

[3 Programování modelu 12](#_Toc161171520)

[3.1 Vytvoření Jupyter Notebooku 12](#_Toc161171521)

[3.2 Importování knihoven 12](#_Toc161171522)

[3.2.1 Knihovna TensorFlow 13](#_Toc161171523)

[3.2.2 Knihovna os 14](#_Toc161171524)

[3.2.3 Knihovna OpenCV 14](#_Toc161171525)

[3.2.4 Modul pyplot 14](#_Toc161171526)

[3.2.5 Knihovna NumPy 14](#_Toc161171527)

[3.3 Příprava datasetu 15](#_Toc161171528)

[3.3.1 Upravení datasetu 16](#_Toc161171529)

[3.3.2 Rozdělení datasetu 18](#_Toc161171530)

[3.4 Sestavení a trénování modelu 19](#_Toc161171531)

[3.4.1 Sestavení modelu 20](#_Toc161171532)

[3.4.2 Trénování modelu 22](#_Toc161171533)

[3.5 Analýza grafů performancí 23](#_Toc161171534)

[3.6 Testování a exportování 24](#_Toc161171535)

[3.6.1 Testování 24](#_Toc161171536)

[3.6.2 Exportování 25](#_Toc161171537)

[3.6.3 Využití exportovaného modelu 25](#_Toc161171538)

[4 Závěr 26](#_Toc161171539)

[5 Zdroje 27](#_Toc161171540)

[6 Seznam obrázků a tabulek 28](#_Toc161171541)

# Úvod

Tato seminární práce je zaměřena na programátory, kteří již mají určité zkušenosti   
s programováním, ale zatím nemají žádné znalosti o umělé inteligenci. I když jste nikdy neprogramovali, tak přesto můžete mé práci rozumět. Veškeré informace jsou detailně vysvětleny, s výjimkou drobností, které by mohl snadno pochopit každý programátor i takový začátečník. Po dokončení čtení byste měli být schopni sami naprogramovat podobný model   
a rozumět základům fungování umělé inteligence pro rozpoznání dat.

Cílem mé práce je vytvořit model hlubokého učení v jazyku python, který bude schopen rozpoznávat obrázky na základě jejich vizuálního vzhledu. Model bude schopen klasifikovat obrázky do dvou kategorií, a to znamená, že se jedná o binární model. Konkrétně se zaměřím na rozpoznání ještěrek a hadů, což jsou dvě kategorie, které mají podobný vizuální charakter, jenž činí úkol pro model těžší než např. rozeznání ptáků od ryb. Všechny kroky vytváření tohoto modelu budou podrobně zdokumentovány a vysvětleny tak, aby byly srozumitelné   
i pro programátory začínající s umělou inteligencí. Kód v mé práci je buď napsaný v specifickém stylu písma nebo je vložen jako screenshot.

V mé práci naleznete systematické uspořádání postupů, které je třeba krok za krokem dodržet. Prvním krokem je vytvoření a aktivace virtuálního prostředí. Následně se věnuji tvorbě datasetu, kde podrobně popisuji jak a kde získat dostatek dat pro model. Jako poslední část mé práce se zaměřuji na využití všech předchozích témat k samotnému programování modelu hlubokého učení pro binární klasifikaci obrázků.

Toto téma jsem si vybral z důvodu mé fascinace umělou inteligencí. Jelikož programuji pro zábavu a rád bych s tím pokračoval jako budoucím povoláním, bylo mi toto téma velmi blízké. Možná vás překvapí, že když jsem začínal s tímto projektem, nevěděl jsem vůbec nic   
o programování umělé inteligence. Všechny zkušenosti jsem nabral během vytváření tohoto projektu. Měl jsem výhodu v tom, že jsem byl už zběhlý v tomto programovacím jazyku, takže jsem po přečtení většiny funkcí rozuměl tomu, co dělají.

Všechny informace k této práci jsem posbíral z YouTube videí a z fóra Stack Overflow[[1]](#footnote-1). Jelikož se jednalo o programování, není mnoho knih na toto téma k dispozici, a řekl bych, že videa jsou pro tuto situaci lepší, protože někdy je efektivnější vidět, jak se věci vytváří než o nich pouze číst. Video, které mi pomohlo nejvíce, je toto: <https://www.youtube.com/watch?v=jztwpsIzEGc&ab_channel=NicholasRenotte>. Ve videu jsou všechny koncepty jednoduše vysvětlené, a přesto obsahuje všechny podstatné informace.

# Příprava Virtuálního Prostředí

Virtuální prostředí v programování je izolovaný prostor, ve kterém může běžet program   
se svými vlastními závislostmi a knihovnami. Tato izolace umožňuje vývojářům spravovat různé projekty s různými verzemi knihoven a zabránit konfliktům mezi různými verzemi.

Moje prostředí jsem vytvořil s pomocí programu Anaconda. Virtuální prostředí jsem poté spároval s JupyterLab, což je webová aplikace, ve které jsem vyvíjel celý projekt.

## Nástroje pro vytvoření virtuálního prostředí

Jak jsem již zmínil, k vytvoření toho virtuálního prostředí jsem použil program Anaconda. Ten můžeme stáhnout z jejich oficiálních stránek[[2]](#footnote-2). Program podporuje všechny platformy (Windows, Mac a Linux). Po nainstalování programu můžeme otevřít aplikaci Anaconda Prompt, v níž jsem pracoval při vytváření virtuálního prostředí pro můj projekt. Anaconda Prompt je velmi podobný Příkazovému řádku,  
 ale na rozdíl od něj má své speciální příkazy (např. conda create nebo conda activate). Dalším nástrojem je JupyterLab, který nyní mohu spustit přes Anaconda Prompt. Stačí pouze napsat: jupyter lab, což nastartuje lokální JupyterLab server ve složce, ze které jsme spustili Anaconda Prompt.

### Jupyter Lab

Jupyter Lab je vývojové prostředí pro programování v jazyce Python a dalších programovacích jazycích. Jedná se o rozšíření původního projektu Jupyter Notebook   
a poskytuje mnoho vylepšení a nových funkcí. Jupyter Lab umožňuje psát kód v buňkách,  
 což znamená, že můžete spouštět kód část po části a okamžitě vidět výsledky. Tato možnost je užitečná pro rychlé experimentování a ladění kódu, což může být méně pohodlné   
v tradičních textových editorech, jako jsou Visual Studio nebo IntelliJ IDE. Dále umožňuje vkládat grafy a další vizualizace přímo do dokumentů, což je užitečné pro vědecké výzkumníky, datové vědce a vývojáře pracující na datových projektech. Kromě kódu lze   
v Jupyter Lab vkládat i textové buňky formátované pomocí Markdown, což umožňuje vytvářet nadpisy a poznámky s vysvětlením kódu. Jupyter Lab dále umožňuje instalovat   
a spravovat balíčky přímo z uživatelského rozhraní, což usnadňuje správu prostředí   
a závislostí pro projekty.

## Vytvoření virtuálního prostředí

K vytvoření potřebuji ještě místo někde na svém počítači. A potom se můžu vrhnout   
na programování a importování knihoven.

### Vytvoření místa

Nejdříve potřebujeme místo na disku, a to může být buď na C: nebo D: či na nějakém dalším disku. Já jsem pracoval na disku C:. Pro umístění prostředí jsem vybral složku „seminary“   
na ploše disku C:. Následně jsem jako správce spustil Anaconda Prompt. K navigaci mezi složkami jsem použil příkaz cd. Začal jsem psát cd /, což mě přesunulo na disk C:. Poté jsem použil cd Users a následně cd thadd, protože to je název mého uživatelského účtu na počítači. Poté jsem se přesunul na plochu do složky „seminary“, pomocí příkazů cd Desktop a cd seminar.

Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automaticky

Obrázek 1: Příkazy v Anacondě. Autor fotografie: Tadeáš Přikryl

### Vytvoření prostředí

Poté, co jsem vše připravil, začal jsem příkazem python -m venv Seminary, který vytvoří základní prostředí pomocí jazyka Python. „Seminary“ je název mého prostředí. Následně aktivuji prostředí příkazem .\Seminary\Scripts\activate. Po aktivaci   
se mi vedle základního prostředí (base) ukáže i (Seminary), což mi říká, že mám aktivovaná prostředí Anacondy a svoje vlastní. Abych spustil mé prostředí, je potřeba vypnout základní prostředí Anacondy příkazem conda deactivate, teď se mi už ukazuje jenom (Seminary). Nyní se mi zobrazuje pouze (Seminary). Po zapnutí Anacondy je možné otevřít Jupyter Lab. V této chvíli dochází k problému, že se nové prostředí nespáruje s Jupyter Lab. Proto nechávám zapnuté pouze mé prostředí a nezapínám Jupyter Lab.

Prvním krokem k propojení prostředí s Jupyter Lab je instalace samotného Jupyteru pomocí příkazu pip install jupyterlab. Tento příkaz nainstaluje vše potřebné pro spuštění Jupyter serveru, včetně knihovny ipykernel, která slouží k propojení prostředí s Jupyter Lab. Příkaz k instalaci Jupyter Lab je k dispozici na jejich oficiálních stránkách[[3]](#footnote-3), stejně jako pro další software, například Jupyter Notebook nebo Voilà.

Nyní vytvořím kernel z mého prostředí pomocí příkazu python -m ipykernel install --name=Seminary. Jméno může být libovolné; já jsem použil stejné jako   
mé prostředí. Pro ověření provedených kroků mohu použít příkaz jupyter kernelspec list, který zobrazí všechny aktivované kernely. Po spuštění Jupyter serveru příkazem jupyter lab mohu změnit kernel Python3 na Seminary v kategorii „Start Other Kernel“. Jupyter server můžeme vypnout stiskem Ctrl+C.[[4]](#footnote-4)

Dále potřebuji několik knihoven, takže je nutné zavřít Jupyter server a pokračovat v instalaci. Tentokrát instaluji knihovnu TensorFlow příkazem pip install tensorflow. TensorFlow je knihovna pro strojové učení vyvinutá společností Google. Poskytuje nástroje pro vytváření a trénování modelů strojového učení, zejména pro hluboké učení. Existuje také možnost instalace pro grafické karty, které pomáhají při trénování modelu. Já jsem to zkusil, protože mám grafickou kartu kompatibilní s CUDA, ale přesto se mi nedařilo kartu rozpoznat. Informace o instalaci jiných verzí jsou k dispozici na oficiálních stránkách TensorFlow[[5]](#footnote-5). Nakonec tedy pokračuji pouze s verzí pro CPU; rozdíl mezi verzemi s a bez grafické karty je asi třetina výkonu, což však záleží na konkrétní grafické kartě.[[6]](#footnote-6)

Potom instaluji ještě několik knihoven v průběhu projektu, a tak všechny shrnuji dohromady. První z nich je OpenCV, rozsáhlá knihovna pro práci s obrazovými a videodatovými sadami, často používaná pro úpravu obrazu, zpracování videa, počítačové vidění a strojové učení. Pro instalaci jsem použil příkaz pip install opencv-python. Pokračuji instalací Matplotlib, oblíbenou knihovnu pro vizualizaci dat, pomocí příkazu pip install matplotlib.

To jsou kroky, které jsem provedl v průběhu instalace prostředí a potřebných knihoven pro můj projekt.

Citace uváděj vždy u konkrétního odstavce, ne až na konci textu (leda by celý text byl citován). Pokud chceš zapsat dva zdroje, zadej dvě poznámky pod čarou, aby každá byla číslovaná zvlášť – poznámka pod čarou má specifický formát – velikost 10 – neměň.

# Dataset

Dataset je soubor dat, který může obsahovat informace seskupené do určité struktury. Tato struktura může být tabulková, hierarchická nebo jiná a závisí na typu dat a jejich organizaci. Datasety jsou základním prvkem v oblasti datové analýzy, strojového učení a vědeckého výzkumu, protože poskytují základní informace, na kterých lze provádět různé analýzy   
a modelování.

Jedním z klíčových aspektů datasetu jsou datové body, což jsou jednotlivé záznamy nebo řádky obsažené v tomto izolovaném prostoru. Každý datový bod reprezentuje konkrétní entitu nebo událost a obsahuje různé atributy, které jsou charakteristiky nebo vlastnosti, jež lze pozorovat nebo měřit. Atributy mohou zahrnovat číselné hodnoty, kategorické informace   
a další.

Dalším důležitým prvkem datasetu je cílová proměnná. Cílová proměnná představuje hodnotu, kterou model snaží předpovědět na základě ostatních atributů. Tento aspekt je klíčový pro tvorbu a trénování modelů v rámci strojového učení.

Můj dataset se skládá z obrázků, protože model bude rozeznávat na základě vizuálního vzhledu. Cílem je shromáždit obrázky odpovídající mému zaměření – ještěrky a hadi.   
To znamená, že tento dataset je rozdělen do dvou kategorií.

## Vytvoření datasetu

V mém případě se jedná o dataset s obrázky. Existuje několik možných způsobů, jak ho vytvořit. Jednou z cest je vyhledávání obrázků na Google, ale zde může vzniknout problém   
s autorskými právy. Při rozeznávání obrázků to však nemusí být takový problém; odlišuje se to například od situace při generaci obrázků, kde v minulosti vznikly problémy při učení modelů z děl významných malířů.

Alternativou je fotografovat obrázky v realitě, což však může vyžadovat značné množství času, zejména pokud jde o zvířata nebo jiné objekty, které je třeba najít v přírodě nebo na různých místech na Zemi. Proto jsem se rozhodl pro využití již předem vytvořeného datasetu. Tyto datasety lze nalézt na různých platformách jako jsou například Kaggle nebo Google Dataset Search.

Můj dataset pochází z Kaggle[[7]](#footnote-7). Tento dataset není explicitně licencován, avšak může vzniknout problém, pokud by byl využit za účelem komerčního zisku. Většina fotografií pochází z pixabay.com, některé však z flickr.com, a to může představovat potenciální problém s autorskými právy.[[8]](#footnote-8)

## Třídění

Po bližším zkoumání mého datasetu jsem zjistil, že je velmi nesrozumitelný, s mnoha složkami, z nichž některé nejsou ani potřeba. Rozhodl jsem se tedy přeorganizovat všechny fotografie do dvou hlavních složek: lizard a snake. Je to obzvláště důležité, protože pracuji na binárním modelu a potřebuji pouze dvě kategorie.

Do složky lizard jsem zařadil složky Chameleon, Gecko, Iguana a Lizard. U složky snake to bylo poněkud snazší, protože tam jsem pouze zařadil složku Snake. Ostatní složky jsem   
v rámci tohoto projektu nepoužil.

Poté je nutné projít všechny fotografie a zkontrolovat, zda neobsahují nechtěné prvky, například brouky, žáby nebo fotografie bez ještěrek či hadů. Při procházení složky lizard jsem objevil celkem dost fotografií jiných živočichů, přesněji asi 10. Ve složce snake jsem však našel pouze jednu takovou fotografii. Celkové prohledávání této složky bylo jednodušší, protože hadi se lépe rozeznají.

Prohledávání složek v tomto objemu fotek je lepší provádět ručně, přičemž je dobré nastavit velké ikony, aby byly fotografie dobře rozpoznatelné na monitoru. S ohledem na to, kolik fotografií bylo v jedné kategorii (kolem 500), tak celý proces mi zabral asi hodinu a půl. Nakonec jsem skončil s 524 fotografiemi pro ještěrky a 497 pro hady.

Je důležité mít podobný počet fotografií pro každou kategorii, abychom předešli odchylkám při trénování modelu. Z tohoto důvodu jsem musel odstranit několik fotek ze složky lizard.

## Umístění datasetu

Ještě je potřeba umístit dataset, tak aby byl snadno dosažitelný. Proto jsem tyto dvě kategorie umístil do složky images a celou složku do místa, kde jsem vytvořil svoje prostředí, takže finální lokace je C:\Users\thadd\Desktop\seminary\images. To znamená, že při programování, když budu odkazovat na umístění souboru, použiji pouze images.

# Programování modelu

V této kapitole se věnuji naprogramování celého modelu a využití všech předchozích nástrojů. Vše je naprogramováno v Jupyter Lab pomocí dvou Jupyter Notebooků. Tento speciální typ souboru je určen pro programování v Jupyter Lab nebo Jupyter Notebook a má koncovku .ipynb, soubor nelze otevřít jinde než v Jupyter Lab nebo Jupyter Notebook.

Jak jsem zmínil v předchozích kapitolách, tento typ souboru je specifický tím, že kód je psán do tzv. buněk. Každá buňka funguje jako normální python soubor. Výhodou je, že mám celý kód na jednom místě a mohu aktivovat buňky jednu po druhé a vidět okamžitý výsledek buňky. I když bych mohl psát kód i v běžných editorech, jako je Visual Studio, pro tento účel je výhodnější použít Jupyter.

Na konci této kapitoly budu mít plně funkční model pro rozpoznání obrázků mezi ještěrkami a hady. Cílím na alespoň 95 % přesnost modelu, i když by v takovém projektu bylo přijatelné dosáhnout i přesnosti kolem 90 %, protože tento model slouží spíše jako ukázka. Nicméně, pokud bych vytvářel model pro nějakou společnost, bylo by nezbytné dosáhnout co nejvyšší přesnosti.

## Vytvoření Jupyter Notebooku

Když tedy spustím Jupyter Lab server přes Anacondu, je třeba vytvořit místo pro psaní kódu. Tato část mé práce je asi nejlehčí, protože stačí v Jupyter Labu jenom kliknout na New Notebook. Můj notebook vytvářím ve stejné složce, kde jsem se pohyboval v rámci virtuální prostředí. Takže cesta k němu je seminary\sem.ipynb, ale nesmíme zapomenout zkontrolovat, zda je prostředí aktivované (v Jupyteru je to Kernel).

## Importování knihoven

Než začnu se samotným programováním, je vždy nejdůležitější importovat všechny knihovny. Rád importuji všechny knihovny na začátku, i když některé z nich přidávám v průběhu projektu. Tento přístup dělá můj kód čitelnější, jak pro mě, tak i pro ostatní, kteří budou číst kód později.

Obsah obrázku text, snímek obrazovky, Písmo

Popis byl vytvořen automaticky

Obrázek 2: Importování knihoven v Jupyter Lab. Autor fotografie: Tadeáš Přikryl

### Knihovna TensorFlow

TensorFlow je open-source knihovna pro strojové učení vyvinutá společností Google. Je navržena tak, aby umožňovala vytváření a trénování modelů strojového učení, zejména neuronových sítí, na různých zařízeních. TensorFlow nabízí flexibilitu, výkonnost a širokou podporu pro vývojáře, kteří pracují na projektech strojového učení.

Pro importování TensorFlow se často používá konvence, kde se knihovna importuje pod zkráceným názvem tf. To umožňuje psát kód jednodušeji a zároveň odlišuje funkce a třídy TensorFlow od jiných částí kódu. Jak je vidět na obrázku nahoře, kód pro importování je import tensorflow as tf. Dále pod prvními importy jsou pak tři řádky kódu, které se vztahují   
k TensorFlow a jsou z balíčku Keras. Keras je balíček pro stavbu a trénování neuronových sítí, který je vestavěn do TensorFlow. Když si tyto řádky přečteme dávají smysl sami o sobě, přesto je stejně vysvětlím.

Jako první importuji Sequential, který umožňuje snadno vytvářet a konfigurovat modely neuronových sítí vrstvu po vrstvě. Tato knihovna se používá k vytváření sekvenčních modelů.

Dalším importem jsou různé druhy vrstev (Conv2D, MaxPooling2D, Dense, Flatten, Dropout). Každá z těchto vrstev má své specifické účely a využití při sestavování modelu neuronové sítě.

Conv2D představuje 2Dsa konvoluci, která je základní operací pro zpracování obrazu   
v konvolučních neuronových sítích (CNN). Tato vrstva umožňuje filtrování obrazu pomocí malých okenních filtrů s cílem extrahovat různé funkce obsažené ve vstupním obraze.

MaxPooling2D provádí tzv. max pooling operaci. Tato operace redukuje rozměry prostoru zpracovávaného obrazu, a tím snižuje celkovou výpočetní náročnost. Max pooling extrahuje dominantní rysy z každé oblasti obrazu.

Dense představuje plně propojenou vrstvu, kde každý neuron je propojen s každým neuronem v následující vrstvě. Její použití je typické v tzv. „fully connected“ vrstvách, které následují po konvolučních a pooling vrstvách.

Flatten slouží k převedení vstupu do jednorozměrného vektoru. Tato vrstva je často používána před plně propojenou vrstvou.

Dropout provádí techniku dropout, což je metoda regularizace používaná ke snížení přeučení modelu tím, že náhodně vypíná (nastavuje na nulu) některé vstupní jednotky během tréninku.

A nakonec se importují metriky, které jsou často využívány při trénování a vyhodnocování binárních klasifikačních modelů.

Precision je metrika, jež měří přesnost klasifikace pozitivních případů. Je to poměr správně klasifikovaných pozitivních případů k celkovému počtu případů, které byly klasifikovány jako pozitivní. Vyšší hodnota precision indikuje lepší schopnost modelu identifikovat skutečné pozitivní případy.

Recall udává poměr správně klasifikovaných pozitivních případů k celkovému počtu skutečných pozitivních případů. Vyšší hodnota recall ukazuje lepší schopnost modelu zachytit všechny skutečné pozitivní případy.

BinaryAccuracy je metrika měřící celkovou přesnost binární klasifikace. Celková přesnost znamená poměr všech správně klasifikovaných případů (pozitivní i negativní) k celkovému počtu případů. Jedná se o jednoduchou metriku, která hodnotí celkovou schopnost modelu správně klasifikovat instance.

### Knihovna os

Knihovna os poskytuje funkce pro interakci s operačním systémem, což umožňuje manipulovat s různými aspekty souborového systému, pracovat s proměnnými prostředí   
a provádět další operace na úrovni operačního systému. Například funkce os.listdir(cesta) slouží k získání seznamu souborů a adresářů v dané cestě, zatímco os.path.join(cesta1, cesta2) spojuje dvě cesty dohromady, a to usnadňuje práci s adresářovou strukturou.

### Knihovna OpenCV

Import cv2 slouží k importování knihovny OpenCV (Open Source Computer Vision Library). OpenCV je výkonná knihovna, která poskytuje nástroje pro zpracování obrazu a videa, počítačové vidění, strojové učení a další. Přesněji řečeno, slouží k detekci hran, segmentaci objektů, transformaci obrazu, analýze videa a mnoha dalším úlohám. OpenCV poskytuje širokou škálu funkcí a algoritmů, což ji činí oblíbenou knihovnou pro vývoj aplikací   
v oblastech počítačového vidění a zpracování obrazu.

### Modul pyplot

Tento příkaz slouží k importování modulu pyplot z knihovny Matplotlib a přiřazení zkráceného jména plt. Matplotlib je knihovna pro vizualizaci dat v Pythonu, a pyplot je jedním z jejích modulů, který poskytuje nástroje pro vytváření různých typů grafů   
a vizualizací. V mém projektu vytvářím grafy performancí a k tomu využívám modul pyplot.

### Knihovna NumPy

NumPy (Numerical Python) je knihovna pro vědecké výpočty v Pythonu, která poskytuje podporu pro efektivní práci s multidimenzionálními poli, lineární algebru, matematické funkce a další. Například mohu vytvořit jednorozměrné nebo dvourozměrné pole, provádět matematické operace s maticemi nebo generovat náhodná data.

## Příprava datasetu

Přestože jsem dataset již vytvořil a umístil, je třeba jej dodatečně zpracovat tak, aby ho model mohl využít. Důvod, proč jsem toto zpracování nevykonal již při tvorbě datasetu, spočívá   
v tom, že k provedení těchto úprav je nutné použít kód v Jupyteru. Tento proces zahrnuje vytvoření tzv. batche a změnu velikosti obrázků.

### Upravení datasetu

Prvním krokem bude načtení umístění obrázků. Použiji zkratku „data\_dir“ (v obrázku buňka   
s číslem 5) pro adresář s daty. Jak jsem již dříve zmínil, všechny obrázky jsou umístěny   
ve složce „images“, takže k přístupu k datům stačí pouze označit tuto složku. Poté pro kontrolu zjistím, jaké složky, z nichž se vytvoří třídy, jsou v mém datasetu (v obrázku buňka   
s číslem 6). Konkrétně to jsou složky „lizard“ a „snake“.

Obsah obrázku text, snímek obrazovky, Multimediální software, software

Popis byl vytvořen automaticky

Obrázek 3: Příprava datasetu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl

Nejdříve krátce nastíním, co vlastně batch představuje. Batch je skupinou vzorků, které jsou zpracovávány najednou během trénování modelu. Trénování neuronových sítí je často výpočetně náročný proces a práce s jednotlivými vzorky by mohla být neefektivní z hlediska výpočetního výkonu. Namísto toho se používá koncept dávkového (batch) zpracování, kde se vzorky dělí do skupin a tyto skupiny se používají k aktualizaci parametrů modelu.

Poté pod variablem „data“ rozdělím dataset do dvou tříd (v obrázku buňka s číslem 7), vytvořím batche a určím velikost obrázků. Když se ale podíváme na kód, můžeme vidět,  
 že jsem nijak tyto veličiny nedefinoval. To je kvůli tomu, že TensorFlow již má tyto hodnoty nastavené ve výchozím nastavení. Velikost batche je v základu 32 a velikost obrázků je redukována na 256x256 pixelů. Tyto hodnoty můžeme upravit, ale pro mé účely jsou výchozí hodnoty dostačující. Shuffle je nastaveno na True, což znamená, že data jsou automaticky promíchána do batchů.

Momentálně jsou ale moje data nepoužitelná a nejjednodušší cestou, jak je udělat čitelná, je   
z nich udělat numpy iterator (čísla). To udělám v další buňce (v obrázku buňka s číslem 8). Poté pod variablem „batch“ beru z mého datasetu nový batch. Pokaždé, když spustím tuto buňku, tak vezmu nový batch (v obrázku buňka s číslem 14). V další buňce (v obrázku buňka s číslem 15) použiji funkci .shape, která mi řekne, jak můj batch vypadá. Můj batch má velikost 32 obrázků, ty jsou velké 256x256 pixelů a jsou ve 3 kanálech, což znamená, že jsou barevné (RGB).

Potom následují dvě buňky (v obrázku buňky s čísly 16 a 17), které ukazují, jak vypadají moje obrázky v batchi a také zjišťují, pod jakým číslem jsou skryty jednotlivé třídy. První buňka ukáže batch v 1 a 0, kde jednička v hranatých závorkách reprezentuje třídy. V jedničce v hranatých závorkách jsou všechny třídy a v nule jsou všechny obrázky v numpy. Když se podíváme na output této buňky, můžeme si všimnout, že je jich přesně 32. Teď je potřeba zjistit, pod jakým číslem jsou jaké třídy. To uděláme další buňkou, kde si necháme zobrazit první 4 obrázky s čísly jejich tříd. Když se podíváme na buňku s číslem 16, vidíme, že zpětně jsem si napsal, jaké třídy jsou za jakým číslem.

V další buňce (v obrázku buňka s číslem 18) škáluji svoje data tak, aby byla mezi jedničkou   
a nulou. V kódu x zastupuje numpy obrázku a y za třídu, a proto měníme pouze x. Obrázky mají hodnoty mezi 0 a 255, protože to jsou čísla barev v RGB. Celkově tedy tento kód provádí normalizaci hodnot pixelů obrázků v datasetu. V poslední buňce (v obrázku buňka   
s číslem 19) můžeme vidět, že output této buňky jsou čísla mezi 1 a 0.[[9]](#footnote-9)

### Rozdělení datasetu

V této části rozdělíme dataset na tři části. První část je trénovací, což je část, kterou model využívá nejvíce, protože zde probíhá veškeré trénování modelu. Druhou část tvoří validační data; tato část je menší a během trénování ji model neuvidí až do konce každé epochy. Na konci každé epochy se model bude snažit vyhodnotit validační data, což je podobné testovacím datům, které během trénování nebyly použity, a tím nám umožňuje získat představu o výkonnosti modelu. Poslední částí jsou testovací data, která model nevidí během trénování a jsou použita k finálnímu vyhodnocení modelu.

Obsah obrázku snímek obrazovky, text, software, Multimediální software

Popis byl vytvořen automaticky

Obrázek 4: Rozdělení datasetu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl

Nejprve tedy nechám spočítat všechna data (v obrázku buňka s číslem 12). Tato operace není nutná pro žádné úpravy, ale poskytuje mi informaci o tom, kolik batchů dat jsem nasbíral. Následně jsem data rozdělil (v obrázku buňka s číslem 13) pod pojmy train\_size, val\_size   
a test\_size. Tyto velikosti jsou vypočteny násobením 0,7, 0,2 a 0,1, takže 70 % dat je trénovacích, 20 % validačních a 10 % testovacích. Tento způsob výpočtu umožňuje jednoduché přizpůsobení pro jiná data v budoucnosti místo ručního zadávání konkrétních hodnot, například 3 pro test\_size.

V další buňce (v obrázku s číslem 14) se kontroluje, kolik batchů bude mít test\_size. Nakonec jsou batche přiřazeny ke svým skupinám (v obrázku buňka s číslem 15) s označením train, val a test. Tato operace se provádí tak, že pro každou skupinu vezmu odpovídající počet batchů   
z celkového datasetu. Pokud je potřeba, přeskočím předchozí batche, které jsem již použil.

## Sestavení a trénování modelu

Sestavení a trénování modelu patří mezi jednu z nejtěžších částí vývoje. Tato fáze může trvat až desítky hodin, protože vyžaduje mnoho experimentování a zkoušení různých konfigurací, zejména pokud se vám nedaří dosáhnout optimálních výsledků. Sám jsem strávil přibližně 6 hodin čistého času při trénování a sestavování modelu. I přes toto úsilí bych řekl, že jsem nedosáhl úplného maximálního výkonu. Po dosažení určité úrovně výkonnosti modelu jsem stále pokračoval v experimentování a pokusech o jeho zdokonalení. Někdy jsem však zjistil, že některé změny mohou výsledky spíše zhoršit. Tato fáze vývoje je náročná nejen kvůli samotnému kódování, ale zejména kvůli časové náročnosti. Trénování modelu je procesorově náročné a váš počítač může být během něj zpomalen.

Během trénování modelu jsem často nechal proces běžet ve chvílích, kdy jsem se věnoval jiným úkolům, protože jeden tréninkový cyklus modelu může trvat od 30 do 60 minut,   
v závislosti na velikosti modelu. Možná by bylo výhodné zvažovat použití grafické karty, pokud by byla kompatibilní, protože může značně zvýšit efektivitu trénování. Doporučil bych zejména spárovat grafickou kartu, pokud pracujete na větším projektu.

### Sestavení modelu

Pro sestavení modelu je potřeba vybrat typ modelu. Můj model je sekvenčně konvoluční. Sekvenční znamená, že je lineární, kde každá vrstva má předchůdce a následovníka. Konvoluční znamená, že se model specializuje na zpracování dat z obrázků. Pro sestavení modelu jsou použity vrstvy složené z neuronů. Nakonec celý model skládám dohromady pomocí funkce .compile().

Obsah obrázku text, snímek obrazovky, software, Multimediální software

Popis byl vytvořen automaticky

Obrázek 5: Sestavení modelu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl

Obsah obrázku řada/pruh, diagram, Vykreslený graf

Popis byl vytvořen automatickyObsah obrázku diagram, text, Vykreslený graf, řada/pruh

Popis byl vytvořen automaticky

Obrázek 6: Funkce ReLU v grafu. Autor fotografie: PyTorch Contributors

Obrázek 7: Funkce sigmoid v grafu. Autor fotografie: YanisaHS

V první buňce (v obrázku buňka s číslem 43) jsem vytvořil sekvenční model pod proměnnou model. I když by bylo možné napsat vše do jedné buňky, rozhodl jsem se rozdělit vytváření   
a přidávání vrstev. Tento přístup mi umožnil postupně přidávat vrstvy, což bylo výhodné, protože jsem nepředpokládal, že dosáhnu optimálního modelu na první pokus.

Druhá buňka (v obrázku buňka s číslem 44) se zaměřuje na přidávání neuronů. Začínám s 16 Conv2D neurony, jejich vstup popisuji jako obrázky o rozměrech 256x256 pixelů ve 3 kanálech (barevný obrázek). Aktivační funkcí je ReLU. Následně přidávám MaxPooling2D vrstvu, na kterou ihned navazuje Dropout vrstva s 50% pravděpodobností vypnutí neuronu. Tyto tři kroky (Conv2D, MaxPooling2D, Dropout) opakuji znovu, tentokrát s 32 Conv2D neurony. Poté přidávám další Conv2D vrstvu s 16 neurony, na kterou navazuje MaxPooling2D. Na konci modelu používám Flatten vrstvu následovanou Dense vrstvou s 256 neurony a ReLU aktivací. Poté vytvářím outputovou vrstvu, která obsahuje Dense vrstvu   
s jedním neuronem a sigmoid aktivací, což odpovídá binární klasifikaci (1 pro hady, 0 pro ještěrky).

Celý model následně kompiluji funkcí .compile() (v obrázku buňka s číslem 45). Jako optimalizační algoritmus jsem zvolil Adam (Adaptive Moment Estimation), který je populární optimalizační algoritmus pro binární modely. Jako loss funkci používám BinaryCrossentropy, jež je běžná pro binární klasifikaci. Metrikou sleduji přesnost (accuracy) během trénování.

Na závěr používám funkci .summary() (v obrázku buňka s číslem 46), která poskytuje přehled o počtu neuronů, vrstvách a celkové velikosti modelu. Celkem má model 3 696 625 neuronů a je připraven k trénování.

### Trénování modelu

Tato fáze vyžaduje hodiny čekání a může být někdy poněkud nepříjemná. Můj aktuální model vyžaduje přibližně 33 minut na dokončení trénování (jedna epocha trvá přibližně 20 sekund).

Obsah obrázku text, snímek obrazovky, software, Multimediální software

Popis byl vytvořen automaticky

Obrázek 8: Trénování modelu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl

V první buňce (v obrázku buňka s číslem 47) jsem vytvořil proměnou logdir a potom jsem vytvořil složku „logs“ ve stejné složce jako je mé virtuální prostředí a dataset. Následně jsem v další buňce (v obrázku buňka s číslem 48) vytvořil TensorFlow callback pro TensorBoard, což jsou funkce spouštěné na konci nebo začátku každé epochy. TensorBoard ukládá průběh trénování do složky „logs“.

Poslední buňka (v obrázku buňka s číslem 49) provedla celé trénování pomocí funkce .fit()   
a výsledky byly uloženy do proměnné „hist“. Prvním argumentem této funkce jsou trénovací data, která jsem v předchozích kapitolách připravil do proměnné „train“. Důležitým argumentem je „epochs“, což určuje, kolik epoch model provede před ukončením učení. Během jedné epochy je každý trénovací vzorek použit k aktualizaci vah modelu jednou. Na konci každé epochy se provede validace modelu na validačním datasetu. V outputu této buňky můžeme vidět čtyři hodnoty loss, accuracy, val\_loss a val\_accuracy. Loss slouží k měření rozdílu mezi předpovězenými hodnotami modelu a skutečnými hodnotami cílové proměnné, takže je ideální mít loss a val\_loss co nejblíže k nule. Accuracy a val\_accuracy udávají přesnost modelu na trénovacích a validačních datech. To souvisí s dalším argumentem, který se týká toho, jaká data použiji jako validační, a to opět využiji již připravenou proměnnou „val“. Nakonec nastavím callback, aby zapisoval průběh tréninku do složky „logs“.

Ve výsledku jsem dosáhl accuracy 99 % a loss 0,022 na trénovacích datech, avšak s validací to bylo horší. Získal jsem průměrné val\_accuracy kolem 90 % a val\_loss mezi 0,2 a 0,3.

## Analýza grafů performancí

Obsah obrázku text, snímek obrazovky, software, Multimediální software

Popis byl vytvořen automatickyObsah obrázku text, snímek obrazovky, software, displej

Popis byl vytvořen automatickyKdyž jsem dokončil trénování modelu, vytvořil jsem dva grafy pomocí knihovny Matplotlib. První z nich je zaměřený na ztrátu (loss) v závislosti na epochách a druhý na přesnost   
v závislosti na epochách. Tímto jsem mohl analyzovat, co modelu chybí a jak ho vylepšit.

Obrázek 9: Loss graf modelu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl

Obrázek 10: Accuracy graf modelu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl

První graf se věnuje ztrátě (loss) během trénování a validace. Ideální stav je, když jsou tyto dvě hodnoty co nejblíže u sebe. Z mého grafu je patrné, že mají odchylku přibližně 0,3. To naznačuje, že bych měl zvážit přidání další formy regularizace, například více dropout vrstev nebo rozšíření datasetu. Experimentoval jsem s oběma přístupy, ale žádný z nich nejevil známky zlepšení. Na začátku grafu také můžeme vidět, že ztráta rychle klesá. To je způsobeno tím, že při první epoše je ztráta kolem 2,5 a model se rychle učí z nových dat. [[10]](#footnote-10)

Druhý graf ukazuje, že přesnost tréninku se po 100 epochách téměř dostala na 100%, zatímco přesnost validace zůstává kolem 90%. To naznačuje, že model má tendenci příliš se přizpůsobit trénovacím datům a nezobecňuje dobře. Můj nejlepší model dosahuje kolem 95% přesnosti na validačních datech. Pokud bych tento model trénoval znovu, mohl bych očekávat odchylku kolem 2,5 %, což by dávalo průměrnou přesnost okolo 93 %.   
V porovnání s ostatními modely, které jsem trénoval, je tento nejslabší. Nicméně, jak jsem již zmínil, tato práce slouží spíše jako ukázka a přesnost 90 % je více než dostačující. [[11]](#footnote-11)

## Testování a exportování

V poslední části programování celého modelu otestuji model a exportuji, aby model byl použitelný i později. Následně vytvořím Jupyter Notebook, který nebude vyžadovat žádné specifické virtuální prostředí pro vyzkoušení modelu.

### Testování

Prvním krokem bude otestovat model na třech batech z testovací části datasetu a změřit jeho výkonnost v třech kategoriích: přesnost (precision), úplnost (recall) a celková přesnost (accuracy).

Obsah obrázku text, snímek obrazovky, software, Multimediální software

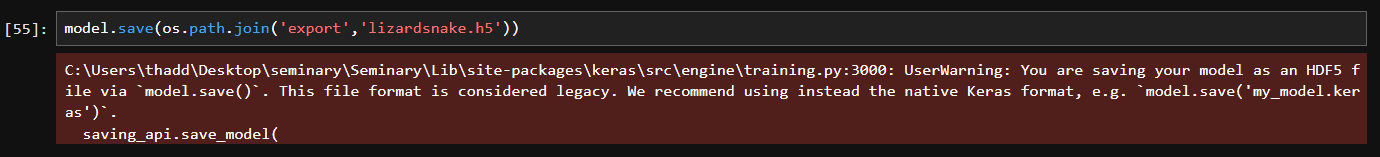
Popis byl vytvořen automaticky

Obrázek 11: Testování modelu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl

V první buňce (v obrázku buňka s číslem 52) jsem jednoduše vytvořil tři kategorie pomocí tří proměnných. Následně v další buňce (v obrázku buňka s číslem 53) jsem nechal model klasifikovat všechny testovací batche a na základě těchto výsledků jsem aktualizoval hodnoty tří kategorií. Precision měří, kolik z pozitivně klasifikovaných příkladů skutečně patří do této třídy. Recall měří, kolik skutečně pozitivních příkladů byl model schopen najít. Accuracy měří celkovou přesnost modelu, což je podíl správně klasifikovaných příkladů k celkovému počtu příkladů. Poslední buňka (v obrázku buňka s číslem 54) zobrazuje procentuální hodnoty pro všechny tři kategorie, které se pohybují kolem 93 %. To není vůbec špatné, zejména když předchozí validace dosahovala okolo 90 %.

### Exportování

Pokud jsem spokojen s výkonem modelu, mohu ho exportovat pomocí funkce .save().   
V tomto konkrétním případě používám také knihovnu os a ukládám model do složky   
s názvem „export“ jako soubor s názvem „lizardsnake.h5“. Tento uložený model mohu kdykoli znovu načíst a využít ho k predikci na nových obrázcích.



Obrázek 12: Exportování modelu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl

### Využití exportovaného modelu

Pokud bych se někdy rozhodl znovu využít model, budu mu muset nějak sdělit, co od něj požaduji. Vytvořil jsem tedy nový notebook, který obsahuje veškerý kód v jediné buňce a lze ho spustit bez specifického virtuálního prostředí. Tento notebook využívá dvou dalších souborů: exportovaný model a obrázek, který chci identifikovat.

Obsah obrázku text, snímek obrazovky, software, Multimediální software

Popis byl vytvořen automaticky

Obrázek 13: Využití modelu v novém notebooku v Jupyter Lab. Autor fotografie: Tadeáš Přikryl

Začnu tím, že importuji knihovny TensorFlow, OpenCV, NumPy a funkci load\_model   
z knihovny TensorFlow. Poté načtu můj exportovaný model do proměnné model. Dále importuji obrázek, který zmenším na rozměry 256x256 pixelů. V proměnné y nechám model provést binární predikci obsahu obrázku. Následně vytvořím výstup na základě toho, zda hodnota y je větší nebo menší než 0,5. Pokud je větší, předpokládáme, že se jedná o hada,   
v opačném případě o ještěrku. Obrázek musí být pojmenován „image“, nebo bych ho musel jinak definovat. Pokud bych chtěl určit jiný obrázek, musel bych ho přejmenovat a vymazat starý obrázek.

# Závěr

Má práce měla za cíl naučit mě a ostatní, jak funguje a jak se programuje umělá inteligence, která se zabývá zpracováním dat z obrázků. Řekl bych, že tento cíl jsem splnil, avšak určitě to nebylo lehké, někdy poněkud stresující. Zejména případy, kdy jsem upravil model a nechal ho někdy i hodiny trénovat, abych zjistil, že je horší než předtím. Nakonec se mi podařilo sestavit svůj nejlepší model s přesností až 98 % na testovací části, takže bych řekl, že v mém programování nebyl takový problém.

Jako slabinu své práce považuji nedostatečnou komunikaci s jinými programátory o tom, jak vše funguje. Neměl jsem moc příležitostí nechat někoho moji práci přečíst a nalezl jsem jenom dva mladé programátory, kteří byli mojí cílovou skupinou k přečtení práce, protože jsem mířil na specificky úzkou skupinu lidí.

S výsledky, kterých jsem dosáhl, jsem určitě spokojen. Vím, že kdybych věnoval trénování   
a vylepšování modelu více času, tak bych mohl dosáhnout ještě přesnějších výsledků. Rád bych se v budoucnu zaměřil i na nebinární modely tohoto typu, protože z toho, co jsem zkoušel, vím, že to bude mnohem náročnější. Kdyby se někdo rozhodl také pokračovat s vývojem umělé inteligence, tak všechen můj kód najdete na stránce github (<https://github.com/TadeasPrikryl/Seminar>).

Toto téma bavilo a určitě hodlám rozšířit své znalosti v této oblasti. Myslím si, že vzhledem   
k aktuálnímu vzestupu umělé inteligence rozhodně neuškodí znát o ní něco více.

# Zdroje

Nicholas Renotte, 2022, Getting Started with Python Deep Learning for Beginners, YouTube video. [2024-02-09]. Dostupné z: <https://www.youtube.com/watch?v=19LQRx78QVU&ab_channel=NicholasRenotte>.

Nicholas Renotte, 2022, Build a Deep CNN Image Classifier with ANY Images, YouTube video. [2024-02-09]. Dostupné z: <https://www.youtube.com/watch?v=jztwpsIzEGc&ab_channel=NicholasRenotte>

Reptiles and Amphibians Image Dataset 🦎🐢. MARIONETTE 👺. [online]. [cit. 2024-02-09]. Dostupné z: <https://www.kaggle.com/datasets/vencerlanz09/reptiles-and-amphibians-image-dataset/data>

RELU. Pytorch Contributors. [online]. [cit. 2024-02-09]. Dostupné z: <https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html>

Sigmoid Activation Function. YanisaHS. [online]. [cit. 2024-02-09]. Dostupné z: <https://www.codecademy.com/resources/docs/ai/neural-networks/sigmoid-activation-function>

Build from source on Windows. TensorFlow. [online]. [cit. 2024-02-09]. Dostupné z: <https://www.tensorflow.org/api_docs/python/tf>

# Seznam obrázků a tabulek

[Obrázek 1: Příkazy v Anacondě. Autor fotografie: Tadeáš Přikryl 9](#_Toc158921019)

[Obrázek 2: Importování knihoven v Jupyter Lab. Autor fotografie: Tadeáš Přikryl 13](#_Toc158921020)

[Obrázek 3: Příprava datasetu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl 17](#_Toc158921021)

[Obrázek 4: Rozdělení datasetu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl 19](#_Toc158921022)

[Obrázek 5: Sestavení modelu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl 21](#_Toc158921023)

[Obrázek 6: Funkce ReLU v grafu. Autor fotografie: PyTorch Contributors 22](#_Toc158921024)

[Obrázek 7: Funkce sigmoid v grafu. Autor fotografie: YanisaHS 22](#_Toc158921025)

[Obrázek 8: Trénování modelu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl 23](#_Toc158921026)

[Obrázek 9: Loss graf modelu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl 24](#_Toc158921027)

[Obrázek 10: Accuracy graf modelu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl 24](#_Toc158921028)

[Obrázek 11: Testování modelu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl 25](#_Toc158921029)

[Obrázek 12: Exportování modelu v Jupyter Lab. Autor fotografie: Tadeáš Přikryl 26](#_Toc158921030)

[Obrázek 13: Využití modelu v novém notebooku v Jupyter Lab. Autor fotografie: Tadeáš Přikryl 26](#_Toc158921031)

1. <https://stackoverflow.com> [↑](#footnote-ref-1)
2. <https://www.anaconda.com/download> [↑](#footnote-ref-2)
3. <https://jupyter.org/install> [↑](#footnote-ref-3)
4. (Nicholas Renotte, 2022, Getting Started with Python Deep Learning for Beginners, YouTube video. [2024-02-09]. Dostupné z: <https://www.youtube.com/watch?v=19LQRx78QVU&ab_channel=NicholasRenotte>.) [↑](#footnote-ref-4)
5. [https://www.tensorflow.org/install/pip#windows-native\_1](https://www.tensorflow.org/install/pip%23windows-native_1) [↑](#footnote-ref-5)
6. (Build from source on Windows. TensorFlow. [online]. [cit. 2024-02-09].   
   Dostupné z: <https://www.tensorflow.org/api_docs/python/tf>.) [↑](#footnote-ref-6)
7. <https://www.kaggle.com/datasets/vencerlanz09/reptiles-and-amphibians-image-dataset/data> [↑](#footnote-ref-7)
8. (Reptiles and Amphibians Image Dataset 🦎🐢. MARIONETTE 👺. [online]. [cit. 2024-02-09]. Dostupné z: <https://www.kaggle.com/datasets/vencerlanz09/reptiles-and-amphibians-image-dataset/data>.) [↑](#footnote-ref-8)
9. (Nicholas Renotte, 2022, Build a Deep CNN Image Classifier with ANY Images, YouTube video. [2024-02-09]. Dostupné z: <https://www.youtube.com/watch?v=jztwpsIzEGc&ab_channel=NicholasRenotte>.) [↑](#footnote-ref-9)
10. (Nicholas Renotte, 2022, Build a Deep CNN Image Classifier with ANY Images, YouTube video. [2024-02-09]. Dostupné z: <https://www.youtube.com/watch?v=jztwpsIzEGc&ab_channel=NicholasRenotte>.) [↑](#footnote-ref-10)
11. (Nicholas Renotte, 2022, Build a Deep CNN Image Classifier with ANY Images, YouTube video. [2024-02-09]. Dostupné z: <https://www.youtube.com/watch?v=jztwpsIzEGc&ab_channel=NicholasRenotte>.) [↑](#footnote-ref-11)