Univerzita Pardubice
Fakulta Elektrotechniky a Informatiky

Webová aplikace pro rozpoznání architektonického stylu budov

Bakalářská práce
2022

David Kyncl

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**David Kyncl**
Osobní číslo:	**I19112**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Webová aplikace pro rozpoznání architektonického stylu budov**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem práce bude vytvořit webovou aplikaci, pro rozpoznání architektonických stylů z fotografie. Teoretická část se bude zabývat existujícími modely umělé inteligence pro rozpoznávání obrazu a jejich srovnáním. V praktické části student vybere vhodný model, natrénuje ho na požadovaný dataset obsahující označené fotografie jednotlivých budov (případně rozšíří dataset o další fotografie) a za dosažení dostatečně dobré úspěšnosti. Součástí webové aplikace budou nabízené informace o rozpoznaném architektonickém stylu.

Rozsah pracovní zprávy: **min. 30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

CHOLLET, François. Deep learning v jazyku Python: knihovny Keras, Tensorflow. Přeložil Rudolf PECINOVSKÝ. Praha: Grada Publishing, 2019. Knihovna programátora (Grada). ISBN 978-80-247-3100-1.
BISHOP, Christopher M. Pattern recognition and machine learning. [New York]: Springer, c2006. Information science and statistics. ISBN 0-387-31073-8.
GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. Deep learning. Cambridge, MA: MIT press, [2016]. Adaptive computation and machine learning series. ISBN 9780262035613.

Vedoucí bakalářské práce: **Ing. Jan Merta**
Katedra softwarových technologií

Datum zadání bakalářské práce: **17. prosince 2021**
Termín odevzdání bakalářské práce: **13. května 2022**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2022

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 8. 8. 2022

David Kyncl

PODĚKOVÁNÍ

Rád bych srdečně poděkoval vedoucímu práce, panu Ing. Janu Mertovi, který nabídl cenné rady a užitečné připomínky k vylepšení práce v celém jejím průběhu. Také bych chtěl poděkovat své rodině a přátelům, za rady a inspiraci.

ANOTACE

Cílem práce bude vytvořit webovou aplikaci pro rozpoznání architektonických stylů z fotografie. Teoretická část se bude zabývat existujícími modely umělé inteligence pro rozpoznávání obrazu a jejich srovnáním. V praktické části student vybere vhodný model, natrénuje ho na požadovaný dataset obsahující označené fotografie jednotlivých budov (případně rozšíří dataset o další fotografie) s cílem dosažení dostatečně dobré úspěšnosti. Součástí webové aplikace budou nabízené informace o rozpoznaném architektonickém stylu.

KLÍČOVÁ SLOVA

neurální síť, vstupní obrázek, konvoluční síť, síťové rozpoznávání obrázků, hluboká neuronová síť

ANNOTATION

Goal of this work will be to create a web application for recognizing architectural styles from images. The theoretical part will deal with existing models of artificial intelligence for image recognition and classification, and their comparison. In the practical part, the student selects a suitable model, trains it on the required dataset containing marked photos of individual buildings (or expands the dataset with other photos) to achieve sufficiently good estimates. The web application will offer information about the recognized architectural style.

KEYWORDS

Neural Network, Input Image, Convolutional Network, Network Image Recognition, Deep Neural Network

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM ZKRATEK

HTTP	Hypertext Transfer Protocol
CRUD	Create, Read, Update, Delete
JSON	JavaScript Object Notation
BSON	Binary JavaScript Object Notation
MLP	Multilayer Perceptrons
CNN	Convolution Neural Network
RNN	Recurrent Neural Network
DBN	Deep Belief Network
RBM	Restricted Boltzmann Machine
CORS	Cross-Origin Resource Sharing

ÚVOD

Technologie se vyvíjí rapidním tempem, každý den může přinést převrat hned v několika odvětvích. V minulosti mezi takové převraty patřil vynález knihtisku, vynález elektrických žárovek nebo také konstrukce prvního počítače. Umělá inteligence, a hlavně její využití v praxi, byl jedním z takových převratů. Digitalizace obrazu existuje již několik desetiletí, data o obraze jsou uložena v binárních hodnotách, které lze počítačem zpracovat. Avšak, možnost dát počítači digitalizovanou fotografii, u které poté rozezná co je jejím obsahem, je dlouho zkoumaný problém, který ještě do nedávna nebylo možné vyřešit.

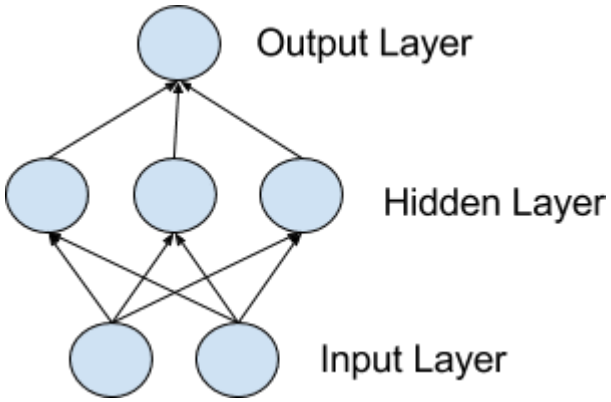
Řešením tohoto problému, stejně jako mnoho dalších se stala umělá inteligence také označována jako neurální síť. Přes sérii algoritmů, se snaží rozeznat spojitost mezi daty, podobně jako jsme je schopni rozeznat my lidé. Zpočátku byly neurální sítě považovány za technologii používanou pouze pro výzkumné týmy, a to z důvodu jejich náročnosti a komplexnosti. Stejně jako s každou jinou technologií, postupem času byly objeveny lepší postupy a efektivnější přístup k problémům, a to umožnilo umělé inteligenci rozšířit se do všech odvětví techniky. Pokrok těchto sítí umožnil řešit nejen problémy týkající se zpracování digitálního obrazu. Dnes najdeme tyto sítě skoro v mnoha aplikacích, od virtuálních asistentů po vyhledávače v prohlížeči.

Cílem bakalářské práce bude vytvořit webovou aplikaci, pro rozpoznání architektonických stylů z fotografie. Proto se práce zabývá uplatněním umělé inteligence na rozeznávání architektury budov z fotografie a bude uvedena problematika rozpoznávání obsahu obrazových dat umělou inteligencí, volby optimálního modelu a postup při tvorbě aplikace tohoto typu. A také svým výzkumem a tvorbou aplikace pomoci ostatním, kteří chtějí prozkoumat umělou inteligenci.

1 NEURONOVÉ SÍTĚ A ZPRACOVÁNÍ OBRAZU

1.1 Multilayer Perceptrons

Obor zabývající se umělými neurálními sítěmi se často zkráceně nazývá „neurální sítě“ nebo „víceúrovňové perceptrony“, tento název byl přebrán z jedné ze základních a pravděpodobně nejužitečnějších neurálních sítí. Síť víceúrovňových perceptronů se skládá z tří druhů vrstev. Vstupní vrstva, která přijímá signály nebo data, která má síť zpracovávat. Skrytá vrstva, v níž je obsaženo různé množství vrstev, které tvoří logickou výpočetní „sílu“ pro celý model. Poslední vrstva je výstupní, cílem této vrstvy je odhadnout nebo klasifikovat vstupní data, na základě informací daných skrytou vrstvou. [4][5][11]



Obrázek 1 - Vrstvy modelů, zdroj: [11]

Tyto sítě (dále nazývány MLP) se skládají z neuronů označované jako „perceptrony“.

Pod tímto pojmem si lze představit algoritmus, který funguje jako lineární klasifikátor. Ke klasifikaci využívá některou z lineárních funkcí, která kombinuje váhy a hodnotu vlastností nalezených ve vstupních datech. Původní myšlenkou perceptronu bylo podobat se neuronům v lidském mozku.

Perceptrony mohou přijímat *N* vlastností na vstupu, a každá tato vlastnost má vlastní váhu. Avšak vstupy do těchto vlastností musí být číselné, je-li vstupní hodnota jiná než číselná, je potřeba provést převod na číselnou reprezentaci této hodnoty. Nejčastější způsob převodu nečíselných vstupů pro vlastnost je reprezentace, že vstup existuje hodnotou 1, a v případě že neexistuje hodnotou 0. Náplní perceptronu je vzít vstupní vlastnosti a vypočítat jejich celkovou váhu. Tato hodnota je poté vyhodnocena další funkcí, a vrací 1 (pravda) překračuje-li vrácená hodnota funkce mezní hranici, nebo 0 (nepravda) je-li výsledek pod mezní hranicí. Učení této sítě spočívá v úpravě vah, aby tato hranice byla dobře nastavena. Pokud se skrytá vrstva skládá z několika podvrstev, vstup na jednotlivé perceptrony je vždy výstup předchozích perceptronů. Z důvodu problematiky

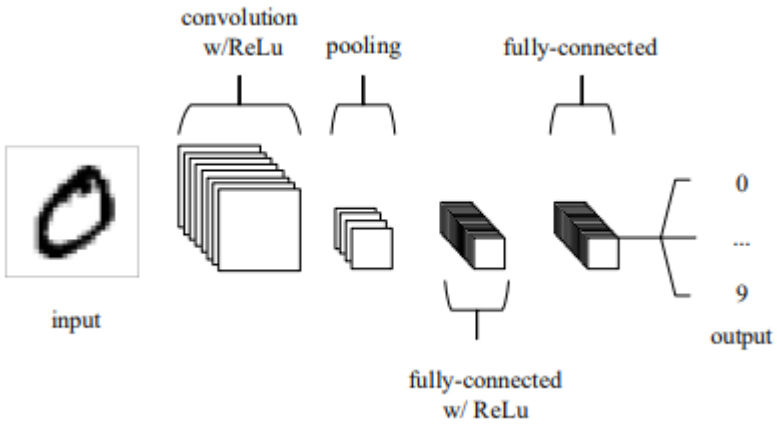
nastavení vah pro MLP, využívané pro rozeznávání a klasifikaci obrazových dat, způsobené náhodnými chybami nebo rušením (kazy v kvalitě apod.), se upřednostňují jiné druhy sítí, pro toto využití. [5][11][12]

1.1.1 Aktivační funkce

Aktivační funkce jsou důležitou částí jakéhokoliv modelu. Jejich využití v řadách vrstev uvnitř modelu umožňuje lepší zachycení nízkoúrovňových a vysokoúrovňových vlastností ze vstupních dat. Princip funkce spočívá v rozhodnutí, vyjádřeném matematickým výpočtem, které informace se mají poslat na další vrstvu v modelu. Zároveň pomáhají omezit velikost výstupu, která by bez nich exponenciálně rostla a vznikaly by velice vysoké nároky na výpočetní výkon. Aktivační funkce jsou považovány za nelinerání funkce, to je důležitá vlastnost v umělé inteligenci, protože většina klasifikačních problémů vzniká z nemožnosti řešit danou problematiku lineárními funkcemi. Mezi vlastnosti těchto funkcí patří omezení nebo vyřešení problému mizejícího přechodu, který nastává když výstup vrstvy projde aktivační funkcí, ale v další vrstvě se jeho hodnota nezmění a projde další aktivační funkcí, která dále výstup redukuje. Nastane-li tato situace několikrát po sobě, model ztrácí možnost se z této aktivace naučit užitečné informace a aktivace zaniká. Jedním z požadavků na aktivační funkce je jejich nenáročnost na výpočetní výkon. Ve většině modelů je aktivační funkce umístěna za každou vrstvou. Počet potřebných výpočtů funkce opět exponenciálně stoupá s hloubkou modelu a počtem cyklů. Mezi nejčastěji používané funkce patří Softmax a Relu, dříve byl používán také Sigmoid, ale tato funkce neřešila problém mizejícího přechodu. [8][9][24]

1.2 Convolution Neural Network

Konvoluční neurální sítě (dále označováno CNN) jsou tvořeny stejně jako jiné typy neurálních sítí, a to vrstvami které obsahují neurony přijímající vstupy, které jsou vyhodnoceny a následně je určena celková váha těchto vstupů. Narozdíl od jiných sítí, neurony v CNN se sami optimalizují při učení. Další z klíčových rozdílů neuronů CNN je jejich rozložení v prostoru. V tomto případě se jedná o dimenze výšky, šířky a hloubky, kde hloubka zastupuje část aktivačního objemu. To znamená, že neurony dané vrstvy přistupují pouze k malé části neuronů z předchozí vrstvy. Tyto vlastnosti dělají CNN excelentní pro tvorbu modelů zaměřených na rozpoznávání obrazových dat. Jiné typy sítí mají problémy zpracovat velký objem dat, který je potřeba například při práci s obrazovými daty. CNN se skládají ze vstupní a výstupní vrstvy a dále tří vnitřních typů vrstev: konvoluční, sdružovací a plně-propojené vrstvy. Vstupní vrstva je zodpovědná za přijímání dat. Ve většině případů se jedná o hodnotu pixelů. Cílem konvoluční vrstvy je určit výstup neuronů a váhy vstupů připojených k této vrstvě.



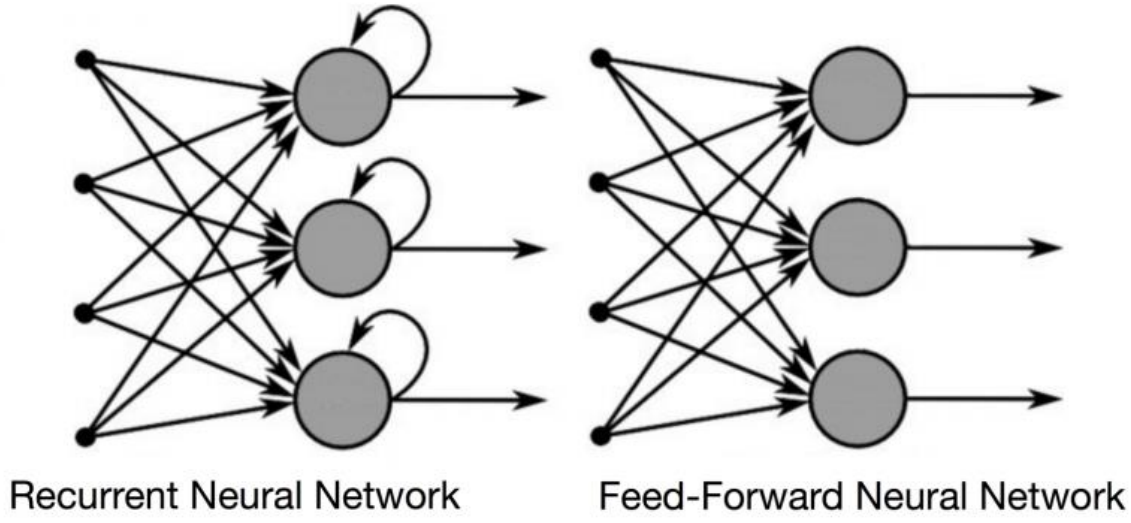
Obrázek 2 - Vrstvy modelu sítě CNN, zdroj: [15]

Sdružovací vrstva se snaží redukovat počet vstupních vlastností z celého vstupu. Plně-spojená vrstva zajišťuje stejnou funkcionalitu v CNN jako vrstvy určující váhu. Tato vrstva je poslední v řadě a zajišťuje získání výsledného odhadu, který můžeme dále zpracovat.

Síla CNN spočívá v jejich zaměření na problém rozpoznávání obrazových dat, na rozdíl od ostatních sítí, které nemají přímé zaměření a řeší tím tedy problém obecnosti řešení pomocí neurálních sítí. Díky těmto vlastnostem je CNN perfektní volbou pro tuto práci. [6][13][14][15]

1.3 Recurrent Neural Networks

Opakující se neurální sítě (dále pouze RNN) jsou založeny na feedforward sítích (dále pouze FFN), ve kterých data putují pouze jedním směrem. V FFN každý neuron zpracovává vlastnosti pouze jednou. Důsledkem toho je proces vyhodnocen v závislosti na nastavených vahách a předchozího trénování neurální sítě. V tomto se RNN liší, data jsou procházena v cyklu, při každém vyhodnocení je zvážen vstup, a vyhodnocení předchozích vstupů. Stejně jako předchozí modely obsahuje RNN vstupní, skrytou a výstupní vrstvu. Skrytá vrstva obsahuje mnoho po sobě jdoucích „opakujících se“ vrstev, které po zpracování dat ze vstupu výsledek opět vrací na vstup. Narozdíl od jiných sítí každý neuron RNN má N vstupů pro přítomnost a N vstupů pro blízkou minulost. Při vyhodnocení se používají váhy vypočítané v daném neuronu a také z předchozího vstupu.



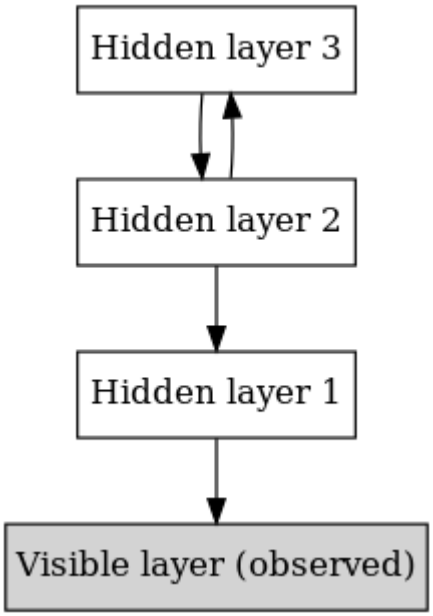
Obrázek 3 - Rozdíl mezi RNN a FFNN, zdroj: [34]

Toto umožňuje RNN efektivně určovat spojitosti informací v dané sekvenci dat, například v rozpoznávání ručního psaní nebo převodu textu na řeč. [16][17][18]

1.4 Deep Belief Network

Síť hluboké víry (dále pouze DBN) je označení pro síť, která se skládá z mnoha omezených Boltzmann strojů (dále pouze RBM), kde každý z nich má dvě vrstvy pro detekci vlastností z daného

vstupu. Hlavní funkce RBM je vytvářet pravděpodobnostní rozdělení na základě vstupního datasetu. Využití tohoto typu sítě v posledních letech rapidně stoupá. Hlavní příčinou je zvýšená potřeba rozpoznávání obrazových dat, rozpoznání hlasu, ručního psaní a podobných potřeb, se kterými mají jiné typy sítí potíže. Nejen že DBN jsou schopny řešit problémy, které jiné sítě nemohou, ale mají také mnohem vyšší přesnost v odhadu a klasifikaci dat. Narozdíl od typických sítí, jsou DBN schopny rozeznat hluboké vzory v datech, a díky tomu rozeznat detaily, které by jiné sítě přehlédly. Větší přesnost a rozpoznání vzorů nachází největší využití v odvětvích jako jsou autonomní vozidla, virtuální asistenti a další, kde jsou vyžadovaná přesná rozhodnutí. Typický způsob stavby DBN je spojování skupin RBM vrstev. Narozdíl od většiny ostatních sítí pracujících pouze s výstupem z předchozí vrstvy v DBN může každá vrstva ve skupině komunikovat s vrstvou předchozí, tak i nadcházející. Z tohoto důvodu se DBN neoznačuje jako vícevrstvá síť, ale síť tvořená více jednovrstvými sítěmi. Vstupní a výstupní vrstva (dále pouze VV) má stejnou roli jako v jiných sítích, ale každá vrstva mezi nimi má dva účely. Pro vrstvy před VV se chová jako skrytá vrstva, která zpracovává vstup a pro vrstvy za VV pracuje jako vstupní vrstva předávající svůj výstup jako vstup pro další vrstvu. Časté využití těchto vlastností je pro rozeznání dat, kategorizace a generování uměle vytvořených obrázků, videí a podobně. [19][20]



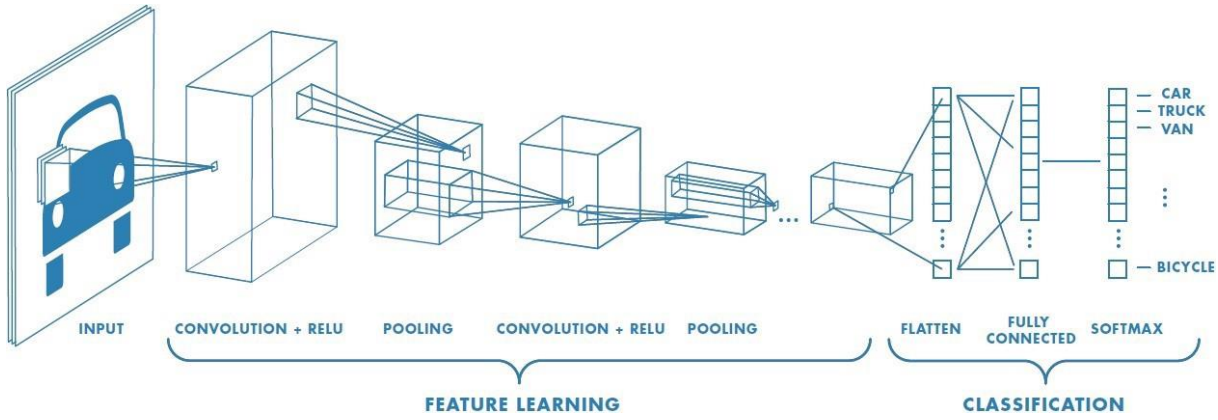
Obrázek 4 - Komunikace mezi skrytými vrstvami, zdroj: [19]

1.5 Restricted Boltzmann Machine

Omezený Boltzmannův stroj (nadále RBM) je jako jiné sítě složen z neuronů, které jsou mezi sebou propojeny ve vrstvách, a na základě vah produkují rozhodnutí. RBM lze učit bez dozoru nebo s dozorem, tato flexibilnost jim umožňuje najít různé využití. V dnešní době je nejčastější využití RBM spočívá v klasifikaci vstupních dat, učení rozpoznání vlastností a predikce na základě vstupu. RBM se skládají ze dvou vrstev, viditelné a skryté. Viditelná vrstva má vstupní body, které přijímají vstupní data. Skrytá vrstva se skládá z bodů, které vyhodnotí vstupní informace a produkuje celkovou váhu ze vstupní vrstvy. Oproti jiným sítím nemá RBM žádnou výstupní vrstvu a ani žádný binární výstup, který by v další iteraci byl použit pro další učení. RBM si automaticky odchytává vzory, parametry a všechny vazby mezi daty. Jsou-li správně nastaveny vstupní vrstvy, tak se o vše ostatní postará samotná síť. [21][22]

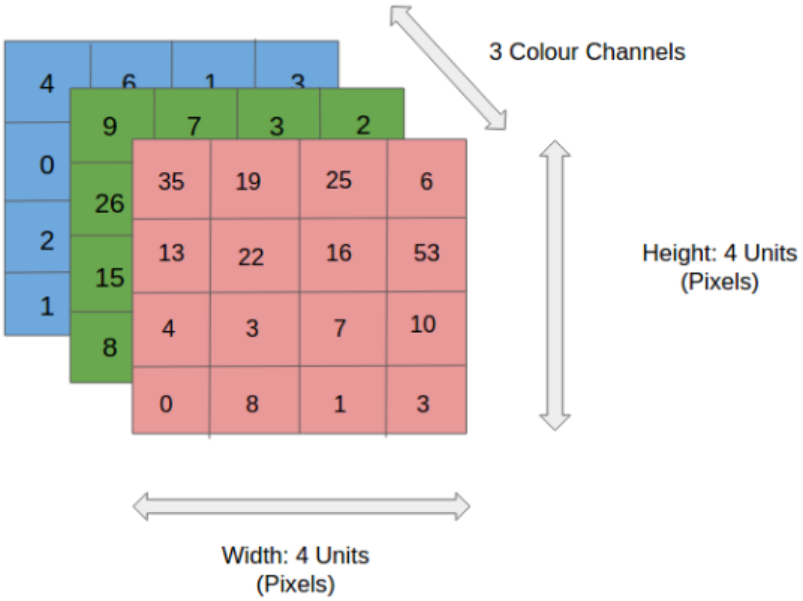
2 PRINCIP FUNKCE CNN

Jak již bylo zmíněno v předchozí kapitole, CNN také nazývány ConvNet, jsou jedny z nejlepších typů neurálních sítí pro klasifikaci nebo rozpoznávání objektů z grafických vstupů jako jsou fotografie nebo kresby. Grafický vstup je nejčastěji v podobě tří kanálů RGB s různou výškou a šířkou. Pro jiné typy sítí je tento objem vstupních parametrů příliš velký a bez aplikování úprav dat vzniká problém v poměru potřebného výpočetního výkonu s výslednou úspěšností modelu.



Obrázek 5 - Vrstvy a jejich typy v CNN, zdroj: [13]

První operací CNN je rozdělení vstupních grafických dat podle jejich barevného modelu, tím bývá nejčastěji RGB. Tento krok má velké výpočetní nároky, které exponenciálně rostou s rozměry vstupu. Pro snížení těchto nároků je na základě zvoleného modelu vstup zmenšen na jednodušeji zpracovatelné rozměry s prioritou zachování co nejvíce vlastností vstupu. Nejčastější rozměry jsou v rozmezí 200x200 pixel až 300x300 pixelů.



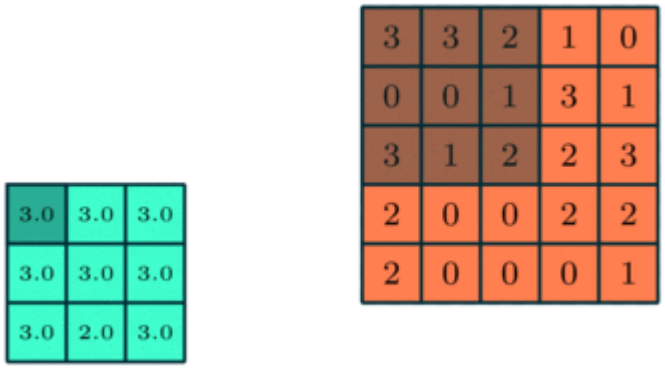
Obrázek 6 - Rozložení vstupních dat na barevné kanály, zdroj: [13]

Dále následuje jedna nebo více konvolučních vrstev, jejichž účelem je získat co nejvíce vysokoúrovňových vlastností vstupu. Jako vysokoúrovňové vlastnosti označujeme části fotografie nebo kresby, které vynikají na první pohled. Mezi ně nejčastěji patří rozdílné objekty, situace a jiné vlastnosti, které pomohou s výslednou klasifikací nebo detekcí vstupu. V některých případech mezi ně mohou patřit i okraje. Naopak jako nízkoúrovňové vlastnosti označujeme barvu, barevné přechody a jejich směrovou orientaci a další na první pohled méně výrazné vlastnosti.

Pro vytvoření matice se používá vzorec *n1 x n2 x l,* kde **n1** je výška, **n2** je šířka přes celou hloubku a **l** je počet kanálů pro daný barevný model.

Více komplexní modely využívají další dvě vrstvy připojené za konvoluční vrstvou, využívané ke zmenšení nebo rozšíření rozměrů originálních vstupních dat. Vrstva „Same padding“ se aplikuje, když je potřeba změnit rozměry a vrstva „Valid padding“ pro zachování stejných rozměrů matice. Posledním krokem vrstvy je zploštit výstup do jednoho rozměru V případě RGB je každá matice hodnot barevného kanálu vynásobena maticí kernelu patřící k danému barevnému kanálu. Výsledky těchto operací jsou sečteny a je k nim přičten bias. Bias nebo také „bias vector“ reprezentuje další sadu vah, kterou lze přidat k vlastním vahám. Bias nepotřebuje žádná vstupní data a vždy přidává hodnotu 1. Přidáním hodnoty bias je zaručen nenulová hodnota výstupu i v případě, že vypočtené hodnoty jsou výsledně nulové. Celkový součet výsledně tvoří jednorozměrný výstup konvoluční vrstvy.

I přes redukovanou hloubku ze všech barevných kanálu do jednoho je výstup stále objemný a také plný šumu, který vznikl v průběhu zpracování dat nebo byl přebrán z dat vstupních. Pro řešení rozměru a šumu existují dva typy sdružování, „max pooling“ a „average pooling“. Jak již vyplývá z názvu, max pooling bere nejvyšší hodnotu z *n x n* výstupu konvoluční vrstvy a average pooling tvoří z výstupu průměrnou hodnotu.



Obrázek 7 - 5x5 výstup konvoluční vrstvy, sdružení do 3x3 přes max pooling, zdroj: [13]

Nejčastější typ šumu je ve výstupu reprezentován nízkými hodnotami, max pooling redukuje rozměry a odstraní všechen šum, toto je však podmíněno rizikem ztráty informací, které se podobají šumu. Average pooling redukuje rozměry stejným způsobem jako max pooling, avšak šum je v tomto případě zakomponován do průměru hodnot při redukci rozměrů. Ve většině případů je max pooling efektivnější a má menší riziko ovlivnění výsledků šumem.

Počet konvolučních a sdružovacích vrstev v ConvNetu závisí na tom, jak složitá data bude model zpracovávat, čím více malých detailů vstup obsahuje, tím více je potřeba těchto vrstev. [6][7][13][23]

3 VÝBĚR MODELU

Vybrat správný model je velice důležitou částí jakéhokoliv projektu pracujícího s umělou inteligencí. Každý model se liší počtem vrstev, jaké vrstvy obsahuje, v jakém pořadí jsou uspořádány, jakou aktivační funkci používají a v jakých rozměrech pracují. Všechny tyto rozdíly ovlivňují výslednou úspěšnost modelu pro rozdílné datasety nebo úspěšnost řešení jiných problému, než je klasifikace. Tato kapitola se zaměří na to, odkud modely pocházejí, co je dělá rozdílnými od ostatních a na jejich očekávanou úspěšnost na základě statistik na ImageNet datasetu. [25]

3.1 ResNet50

ResNet50 je nástupcem menší verze stejné architektury, známé jako ResNet34. Jméno ResNet je odvozeno od reziduálních bloků, které jsou hlavním stavebním kamenem tohoto modelu. Celá architektura je postavena s cílem nejen odstranit ztrátu přesnosti, ale i získat přesnost s vyšším počtem vrstev. Jako reziduální blok označujeme skupinu vrstev, jejichž výstup nepostupuje pouze do další vrstvy, ale také do vrstvy hlouběji v bloku. V modelech bez reziduálních bloků je častým problémem zpočátku klesající chybovost, která ale začne stoupat, čím více vrstev je třeba zpracovat. [25][26][27]

3.2 ResNet152

Vývoj ResNet152 modelu začal v 2015, když skupina odborníků ze společnosti Microsoft začala řešit problémy se ztrátou úspěšnosti v modelech s velkým počtem vrstev. Jak bylo uvedeno u modelu ResNet50, reziduální bloky nezlepšují rychlost klesání chybovosti za počet iterací, ale zamezují opětovnému nárůstu. Díky této architektuře je možné trénovat modely přes velký počet iterací, s menším rizikem chyb, způsobených ztrátou informací po zpracování velkého množství vrstev. ResNet152 byl vytvořen jako ukázka efektivity těchto bloků v modelu s více jak sto vrstvami. S použitím datasetu ImageNet, modely ResNet se 152 vrstvami dosahují úspěšnosti pohybující se okolo 83 %. [25][26][27]

3.3 InceptionV2

Ve stejném roce, kdy ve společnosti Microsoft vznikal model ResNet, autoři z Londýnské univerzity Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe a Jonathon Shlens vydali práci zabývající se řešením problémů Inception V1 modelu, také známého jako GoogLeNet. Hlavními problémy modelu byl „bottleneck“ tvořený velkými změnami rozměrů vstupu a celkovou náročností na výpočetní výkon. Prvním krokem bylo rozdělení základní 5x5 konvoluce Inception modelu na

dvě zároveň prováděné 3x3 konvoluce, takto byla zachována úspěšnost, a zároveň snížena náročnost. Tyto 3x3, nebo jiné NxN konvoluce, lze také rozdělit na 1x3 a 3x1 nebo 1xN a Nx1, které následují za sebou. Na základě testování autoři zjistili, že tato metoda opět sníží náročnost bez žádné poznatelné ztráty úspěšnosti. Pro vyřešení bottlenecku se rozhodli rozšířit model do šířky, neboli zpracovávat více vrstev zároveň, místo řazení jedné za druhou. Inception V2 model má na rozšířeném ImageNet datasetu úspěšnost okolo 85 % a toto ho dělá dobrým kandidátem pro použití v projektu. [25][28][29][30]

3.4 InceptionV3

Tento model byl publikován zároveň s již zmíněným InceptionV2, autoři navrhli dvě verze stejné architektury s malými rozdíly. Mezi ně patří implementace RMSprop, „label smoothing regularizer“ určený k lepší regularizaci a normalizaci učení a pomocný klasifikátor pro zlepšení výsledků ve velice hlubokých modelech. Očekávaná úspěšnost je velice podobná jako pro InceptionV2, ale z důvodu těchto změn je zde šance pro zlepšení. Úspěšnost nad ImageNet datasetem se pohybovala v okolí 86 %.[25][30]

3.5 VGG16

VGG předchází již zmíněné modely, podobně jako oni i tento model vyhrál soutěž ILSVRC (ImageNet Large Scale Visual Recognition Challenge). Nápadem modelu bylo používat velice malé 3x3 konvoluční vrstvy, předchozí modely používaly například 11x11 nebo 7x7. Tento model byl přelomový díky své úspěšnosti i s malým počtem vrstev. Z názvu vyplývá, že model obsahuje šestnáct vrstev, doopravdy má však třináct konvolučních vrstev, pět max-pooling vrstev a tři plně propojené. Pouze konvoluční vrstvy a plně propojené se počítají do vrstev ovlivňující váhy, a proto VGG16.[25][31]

3.6 Xception

Další model od společnosti Google, který získal své jméno ze slovního spojení Extreme Inception. Z názvu vyplývá, že se jedná o model navazující na modely Inception. Cílem bylo dosáhnout lepšího využití dostupných parametrů modelu s použitím nových typů konvolučních vrstev. Nové typy byly pojmenovány „depthwise“ a „pointwise“. Tato architektura by měla dosahovat lepších výsledků než model InceptionV3, ze kterého převážně model Xception vycházel. S datasetem ImageNet se úspěšnost modelu blíží k 90 %. [25]

3.7 EfficientNet

V roce 2019 se tým ze společnosti Google rozhodl zvolit opačný přístup ke tvorbě modelů než bylo doposud typické. Většina modelů získává větší úspěšnost z rozšíření rozměrů, ať je to více vrstev nebo větší vstupní rozměry dat. EfficientNet není v tomto ohledu jiný, ale způsob, jak rozšiřuje tyto rozměry se liší. Každý z rozměrů je navyšován jednotně pomocí pevně daného koeficientu. Výsledkem je rychlejší a menší rodina modelů označovaná jako EfficientNet, která je schopna získat vyšší úspěšnost než dosud existující modely. Z výsledků testů nad ImageNet datasetem je tato skupina modelů schopna dosáhnout úspěšnosti vyšší jak 90 %. [25]

4 UČENÍ MODELU

4.1 Dataset na trénování modelu

S vybranou sítí a vybraným modelem je další složitou částí přípravy umělé inteligence volba takzvaného datasetu. Dataset je kolekce dat, v případě této aplikace kolekce fotografií budov, ze které model čerpá pro prvotní trénování, a poté případnou validaci. Jeden z největších nároků na dataset je jeho velikost. Neurální sítě vyžadují velké množství vstupních dat, aby dosáhly obstojné úspěšnosti ve svých odhadech. S dnešním rozšířením umělých inteligencí po celém světě není složité najít již existující dataset přímo pro požadavky projektu, případně obohatit jednoduchý dataset vlastními daty z dostupných zdrojů. Stavba vlastního datasetu od naprostého začátku je zdlouhavá a pro tento projekt nepraktická, všechny fotografie by měly být veřejně dostupné nebo kreditovat autora. V případě tohoto projektu bylo cílem nalézt vhodný dataset a dle možností ho rozšířit. Mnoho datasetů se nachází na archivních stránkách jako je například „Kaggle“ a „UCI Machine Learning Repository“, nebo v osobních/projektových repozitářích autorů na stránkách GitHub, GitLab, SourceForge nebo jejich vlastních. Jak již bylo zmíněno, tato práce se zabývá identifikací architektonických stylů budov, částečně z důvodu dostupnosti dat pro toto téma. Repozitář UCI bohužel neobsahoval vhodný dataset. Díky své popularitě Kaggle obsahoval více datasetů na tématiku budov, avšak jejich rozsah byl řádově malý, nebo jejich zaměření bylo příliš specifické pro danou geologickou oblast nebo obsahoval budovy nespadající do typických světových architektonických stylů. V repozitářích GitHub se nacházel veřejně dostupný projekt, s velice podobným cílem, identifikovat budovy na odvětví celosvětových stylů. Pro využití zde byly třídy rozděleny do podrobných odvětví stylů, ale díky rozsáhlosti datasetu, stačilo třídy spojit zpět dohromady do stylů, ze kterých se větví. Tento dataset splňuje rozsáhlost v počtu tříd i počtu jednotlivých fotografií pro každou třídu. Celkový počet 9166 fotografií, je jako základ dat pro umělou inteligenci dostatečně obsáhlý. Při průběhu učení je tento počet uměle rozšířen díky transformaci vstupních dat, jako je například částečná rotace, změna úrovně jasu, převrácení přes horizontální nebo vertikální osu a barevné posuny.

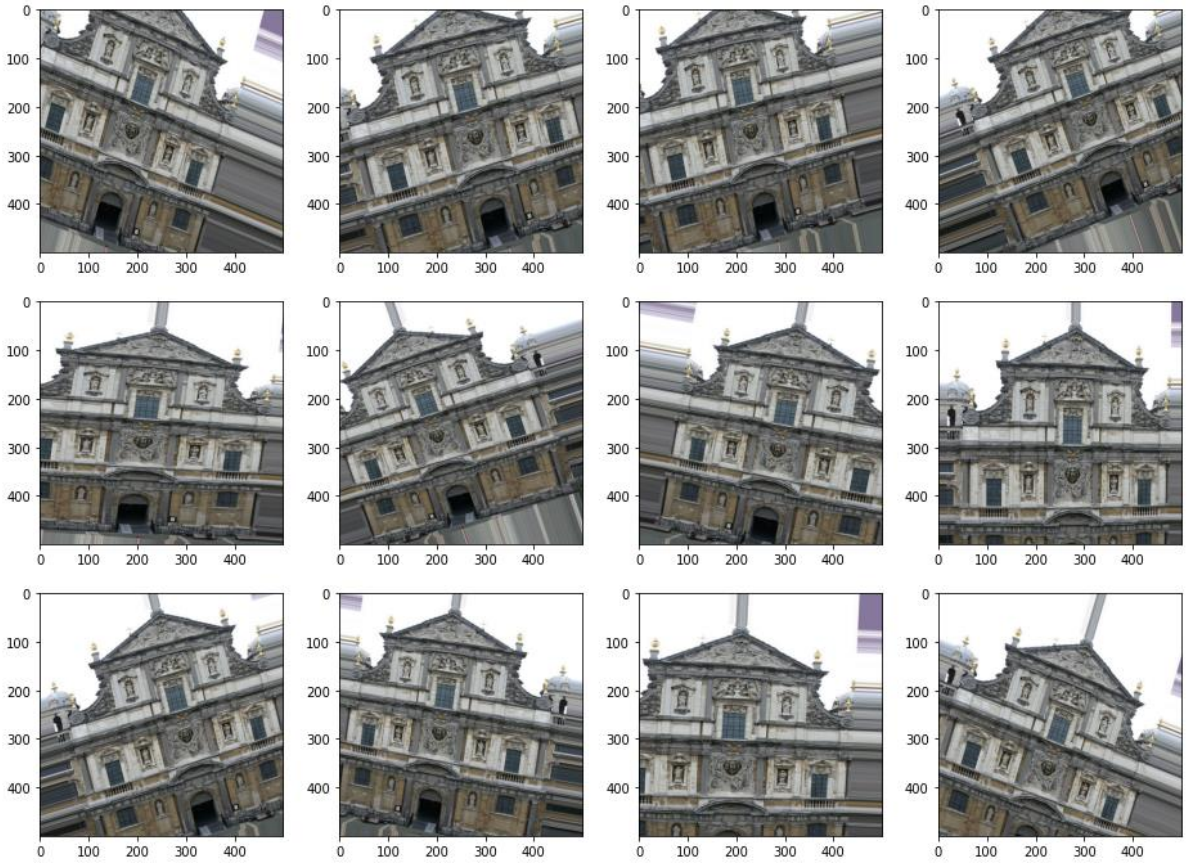
4.2 Augmentace vstupních dat

V oboru umělé inteligence se jako datová augmentace označuje úprava dat, převážně z důvodu rozšíření datasetu. Tyto úpravy jsou v podobě rotace, posunutí, převrácení přes horizontální nebo vertikální osu, změny světlosti a přiblížení. I přes podobný význam, je důležité neplést augmentaci dat s transformací dat. Transformace dat se zabývá zpracováním, takzvaným „očištěním“ datasetu před jeho použitím. Mezi takové transformace patří odstranění duplicitních dat, převod datových

typů na ty podporované modelem, odstranění nepodporovaných znaků v názvech a převod logických dat na jejich číselnou reprezentace (například ano/ne na 1 a 0). Jak již bylo zmíněno, hlavním cílem datové augmentace je rozšíření datasetu, protože i rozsáhlý dataset s tisíci fotografiemi není dostatečný na velký počet cyklů pro dnes používané modely. Typické nastavení modelu je 32 vstupů na jeden krok, označeno jako batch, a 256 kroků pro jeden cyklus. Potřebný počet cyklů se může lišit od modelu k modelu, jejich přesným počtem se bude zabývat kapitola Analýza průběhu cvičení, ale pro představu o objemu dat, řekněme že jejich počet je v řádech desítek. Minimální počet vstupních dat na cyklus můžeme tak získat výpočtem *batch * počet kroků*. Z tohoto vzorce můžeme určit, že se základním nastavením bude potřeba 8192 fotografií na jeden cyklus. Z předchozí kapitoly o datasetu víme, že samotný dataset obsahuje 9166 fotografií. I tento obsáhlý dataset by vystačil pouze na jeden cyklus a získaná úspěšnost by byla velice nízká. Z tohoto důvodu je dobré využít datovou augmentaci.

Datovou augmentaci lze rozdělit na dvě kategorie, augmentace za běhu a augmentace datasetu před použitím. Ve většině případů je datová augmentace za běhu jednodušší a efektivnější způsob rozšíření datasetu. Všechny úpravy lze provést automaticky za běhu aplikace, když model vyžaduje další data, a to bez potřeby ukládat nová augmentovaná data na disk. Nevýhodou však je náročnost na paměť a výkon zařízení, na kterém se model trénuje. V případě volby augmentace před použitím dat, jsou data augmentovaná stejným způsobem, ale výsledky jsou uloženy na disk. I s dostatečným úložným prostorem má tento typ augmentace problém. Zvolíme-li třicet cyklů, se stejným nastavením, dostává se potřebný počet fotografií na 245 760, s průměrnou dobou zpracování jednoho kroku v rozmezí 500 ms až 1 sekundy, vzniká takzvaný „bottleneck“ v rychlosti výběru dat z úložného prostoru.

Díky dostupnému vysokému výpočetnímu výkonu byl pro projekt zvolen přístup augmentace za běhu. V následujícím obrázku byly vygenerovány verze jedné fotografie na základě stejných parametrů augmentace, které používá model při učení.



Obrázek 8 - Ukázka aplikace augmentací, zdroj: vlastní

Pro model se chová každá tato úprava jako unikátní vstupní data. Augmentace má své limity, i s možností nastavit posun například na 90 % šířky fotografie, takováto augmentace poškodí výslednou úspěšnost modelu více, než by ji pomohla. Na základě doporučení a informací získaných z materiálů, byly hodnoty nastaveny na jednu čtvrtinu maximální hodnoty. Například rotace se pohybovala v rozmezí 20 stupňů a posun o 20 % šířky nebo výšky fotografie. Rozbor všech augmentací bude v kapitole o samotném kódu k vyučení modelu.

Existuje také možnost zdánlivého rozšíření datasetu přes opakování základního datasetu, tento způsob nelze označit jako augmentaci, a také způsobuje velice velký "overfitting".

4.3 Overfitting a underfitting

Overfitting je jeden z klíčových problémů učení umělé inteligence a označuje situaci, kdy model hlásí vysokou úspěšnost na trénovacích datech, ale úspěšnost na testovacích datech nebo na nových datech neobsažených v žádném použitém datasetu, je velice nízká. Když model trénuje moc dlouho na datech nebo se data opakují, model se naučí i šum obsažený ve vstupních datech. Má-li model dostatečný objem dat nebo dostatečnou augmentaci dat, tento problém se neprojeví, pokud model není trénován moc dlouho.

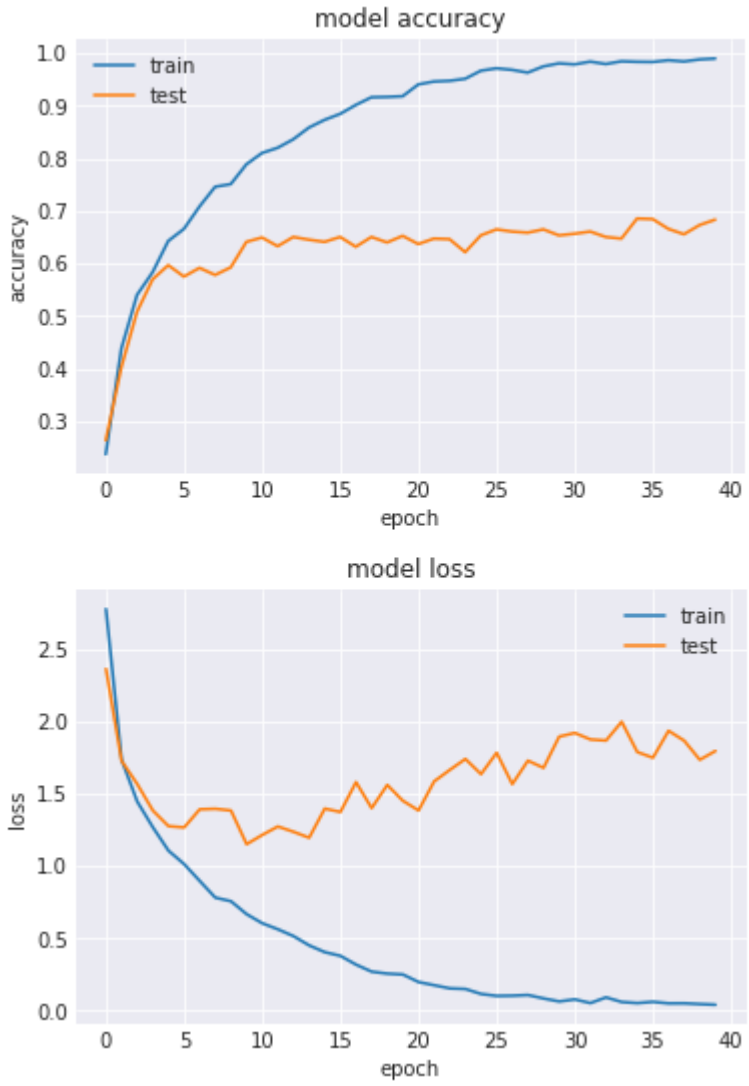
Protější strana tohoto problému je takzvaný „underfitting“, kde model neprošel dostatečným počtem cyklů, aby se naučil všechny potřebné detaily a vlastnosti dat k vytvoření správného odhadu. Overfitting se projevuje vysokou úspěšností nad trénovacím datasetem a nízkou úspěšností nad testovacím datasetem. Underfitting má nízkou úspěšnost nad oběma datasety. Z principu je jednodušší rozeznat underfitting než overfitting. Cílem je tedy zastavit učení v takzvaném „zlatém středu“ dříve, než model narazí na overfitting, ale zároveň, když už se naučil co nejvíce vlastností z dat. [33]

4.4 Analýza průběhu trénování

Tato kapitola se zabývá analýzou jednotlivých modelů s rozdílnou velikostí batch parametru, jiným počtem kroků na každý cyklus a také počtem cyklů pro celý průběh trénování. Každý graf přesnosti obsahuje statistiky počtu cyklů, označeno jako *epoch*, přesnost odhadu a přesnost odhadu nad validačními daty. Graf ztráty modelu obsahuje počet cyklů, ztrátu při učení a ztrátu při validaci. Všechny grafy byly generovány z historie modelu, která byla vytvořena po uběhnutí daného počtu cyklů. Grafy jsou tvořeny Pythonovskou knihovnou Matplot, kterou je možné importovat přímo do sešitu Google Collab. V případě hodnoty ztráty modelu, se jedná o jednotku, kterou lze měřit úspěšnost odhadu. Výpočet ztráty je prováděn jako průměr všech validačních pokusů v jednom cyklu. Ideální situací by byla validační ztráta s hodnotou 0, to znamená, že model vyhodnotil při validaci všechny vstupy správně. Každý model začíná na vysoké hodnotě ztráty a postupem skrze cykly klesá. Cílem každého modelu je dosáhnout na co nejnižší validační ztrátu (to znamená nejvyšší možnou procentuální validační přesnost). Bohužel i modely, které mají více jak 90 % v trénovací přesnosti, nemusí dosahovat na dobrou validační přesnost. Tato situace nastala hned v několika trénovaných modelech. Velká validační ztráta a malá validační úspěšnost může naznačovat hned několik problémů. Mezi ně patří nedostatek dat (underfitting) nebo naopak příliš mnoho dat, kde se model učí i šum (overfitting). Častým problémem je špatná generalizace modelu, to znamená, že z trénovacích dat se model naučil příliš mnoho detailů, které z většiny nejsou obsaženy ve validačních datech. Každý tento problém vyžaduje řadu experimentů ke zjištění, kterou část modelu je potřeba upravit.

4.4.1 VGG16

VGG16 byl vhodnou volbou pro první pokus trénování modelů. Díky své nízké hloubce šestnácti vrstev, probíhalo trénování značně rychleji v porovnání s ostatními. Cílem trénování VGG16 bylo zjistit, zda jsou modely s nižším počtem vrstev schopné získat veškeré potřebné vlastnosti ze vstupních dat, k získání dostatečné úspěšnosti odhadu.

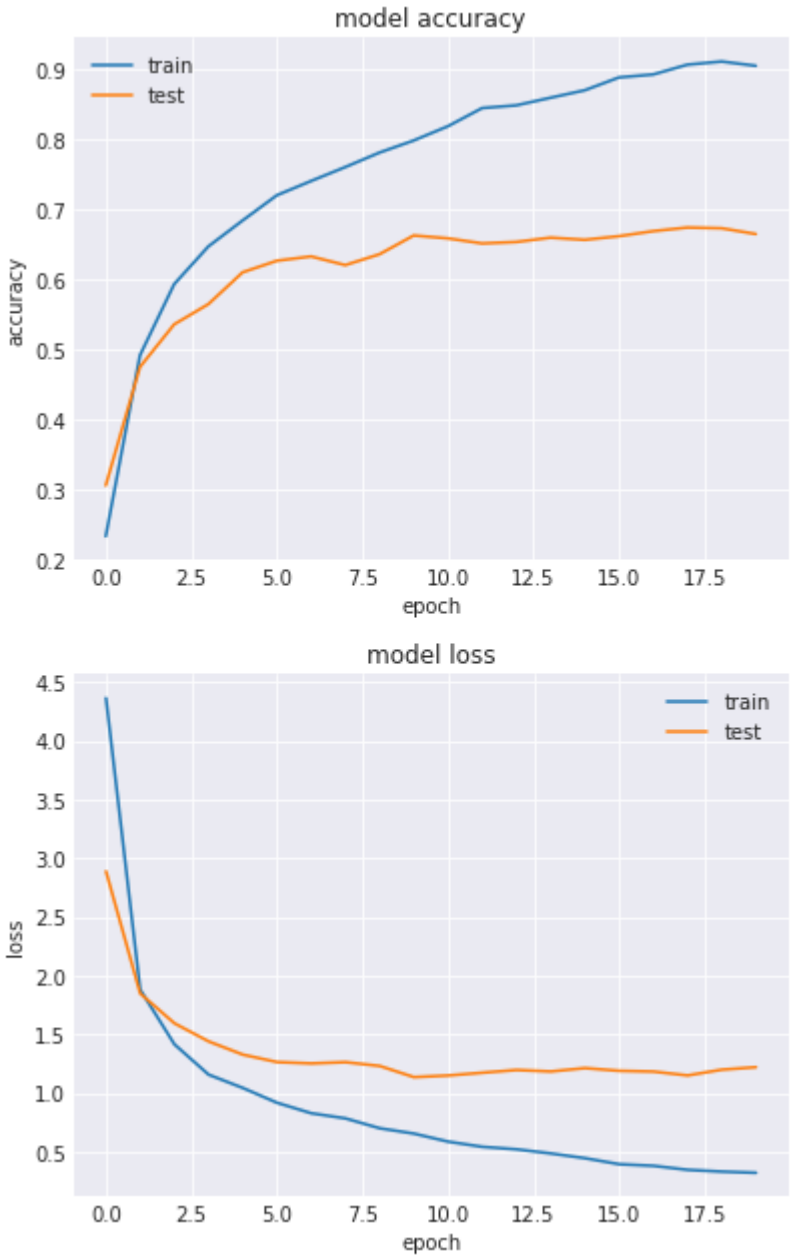


Obrázek 9 - Graf přesnosti a ztráty modelu VGG16, zdroj: vlastní

Dosažená přesnost v trénování a validaci na první pohled vypadá dobře, ale z grafu s hodnotami ztráty je jasné, že VGG16 není pro tento typ dat vhodnou volbou.

4.4.2 ResNet50

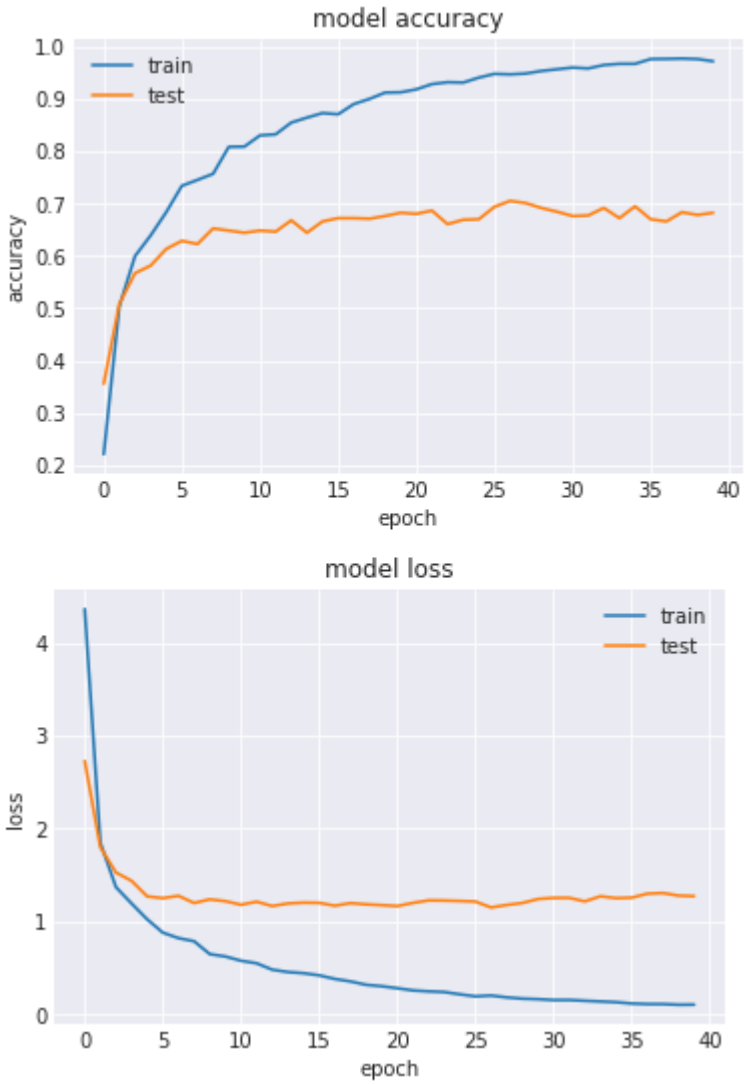
Druhým modelem byl ResNet50 (dále pouze R50), díky své poměrně vysoké úspěšnosti, ale zároveň menší komplexitě pouze padesáti vrstev. Dimenze vstupu pro R50 je 224 na 224 pixelů se třemi barevnými kanály. První pokus byl s batch size 24 a počet kroků 128 po dobu 20 cyklů. Z grafu vidíme, že s tímto počtem cyklů model ještě nedosáhl ustálených výsledků.



Obrázek 10 - Graf přesnosti a ztráty modelu VGG16 s upravenými parametry, zdroj: vlastní

Další průběh byl spuštěn s 24 batch size, 128 kroků a 40 cyklů. Zde můžeme vidět dosažení vyšší úspěšnosti a také ustálení úspěšnosti.

Výsledná trénovací přesnost dosáhla 98 %, se ztrátou 0,2, naopak validační přesnost se pohybovala v rozmezí 67-70 % a ztrátou 1.3.



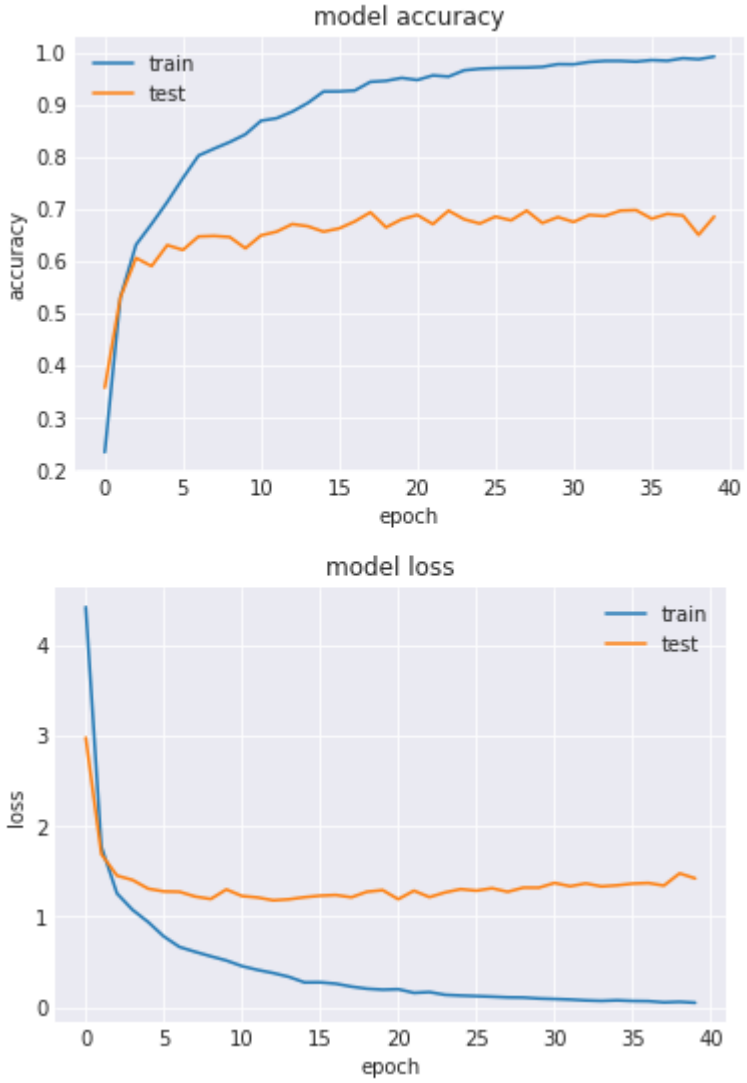
Obrázek 11 - Graf přesnosti a ztráty modelu ResNet50, zdroj: vlastní

Z výsledků je možné usoudit, že R50 v základní podobě nedosahuje dobré úspěšnosti. Pro ostatní modely zvýšíme základní počet cyklů také na 40.

4.4.3 ResNet152

Zkráceně R152, je další model z rodiny ResNet, přesnost by se měla tedy pohybovat v okolí hodnot modelu R50, s možností zlepšení díky větší hloubce modelu. Stejně jako předchozí experimenty byl model trénován s batch size 24, počet kroků 128 a 40 cyklů. V následujícím grafu (viz. obrázek

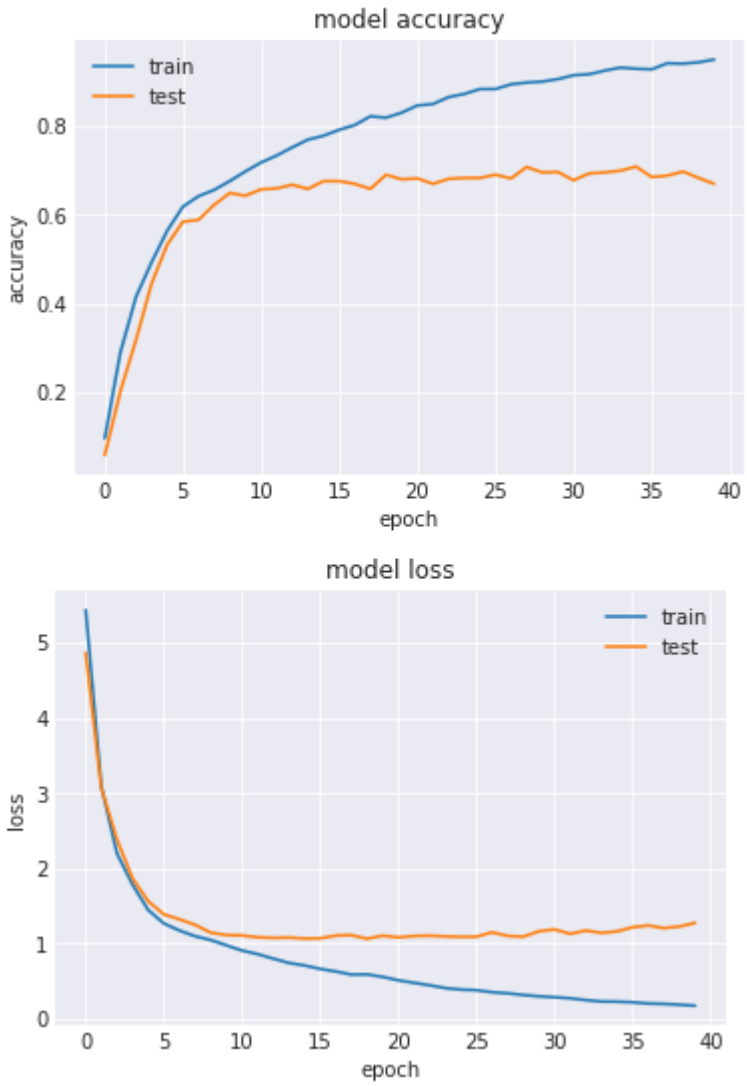
dole) lze vidět dřívější dosažení vysoké úspěšnosti, která se po 35 cyklech blíží ke 100 %. Validační přesnost, která reprezentuje úspěšnost na datech, které model nepoužíval pro vyučení, dosahuje pouze 70 %, se začínajícím úpadkem v posledních pár cyklech. U tohoto pokusu lze také zaznamenat větší nárůst ztráty pro validační data oproti modelu R50.



Obrázek 12 - Graf přesnosti a ztráty modelu ResNet152, zdroj: vlastní

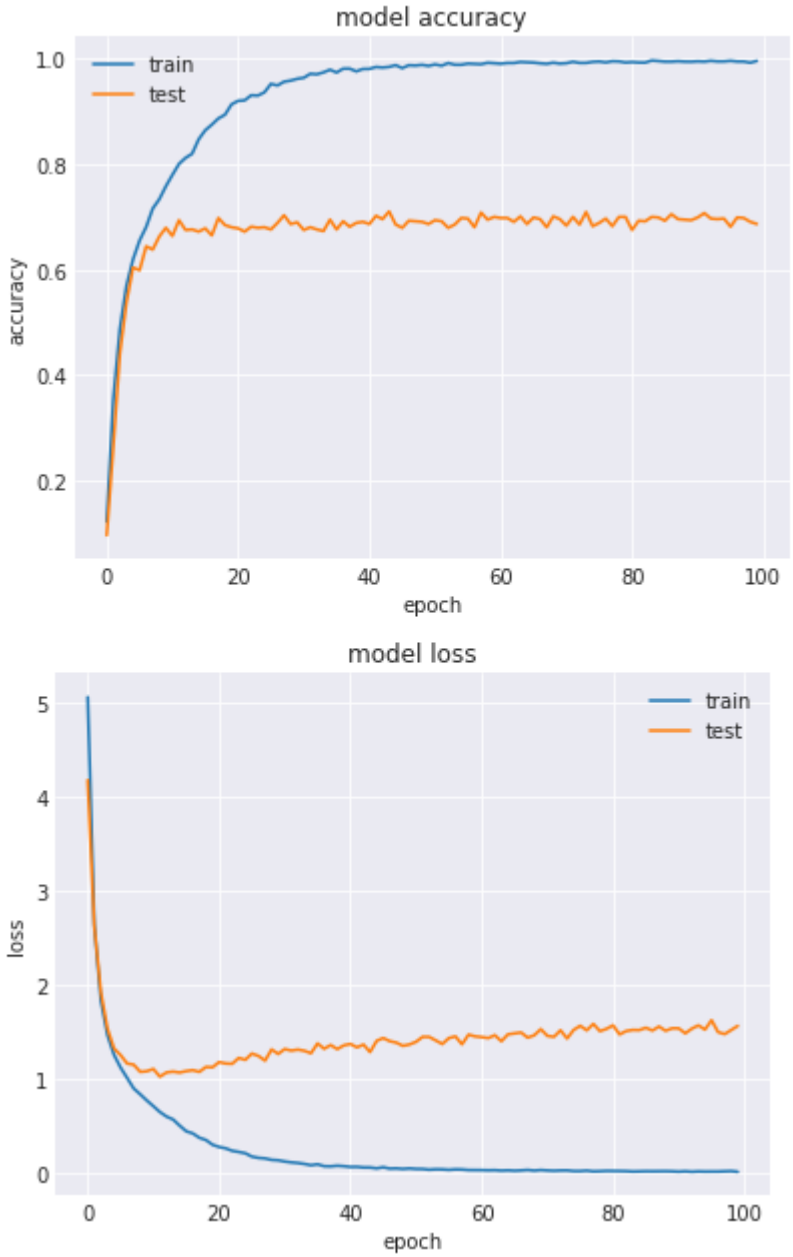
4.4.4 InceptionV2

Z rodiny modelů Inception, které jsou z části založeny na architektuře ResNet, je Inception Version 2 (zkráceně InV2) nejčastěji používaný model pro klasifikaci. Pro otestování této architektury modelu byly opět zachovány stejné parametry.



Obrázek 13 - Graf přesnosti a ztráty modelu InceptionV2, zdroj: vlastní

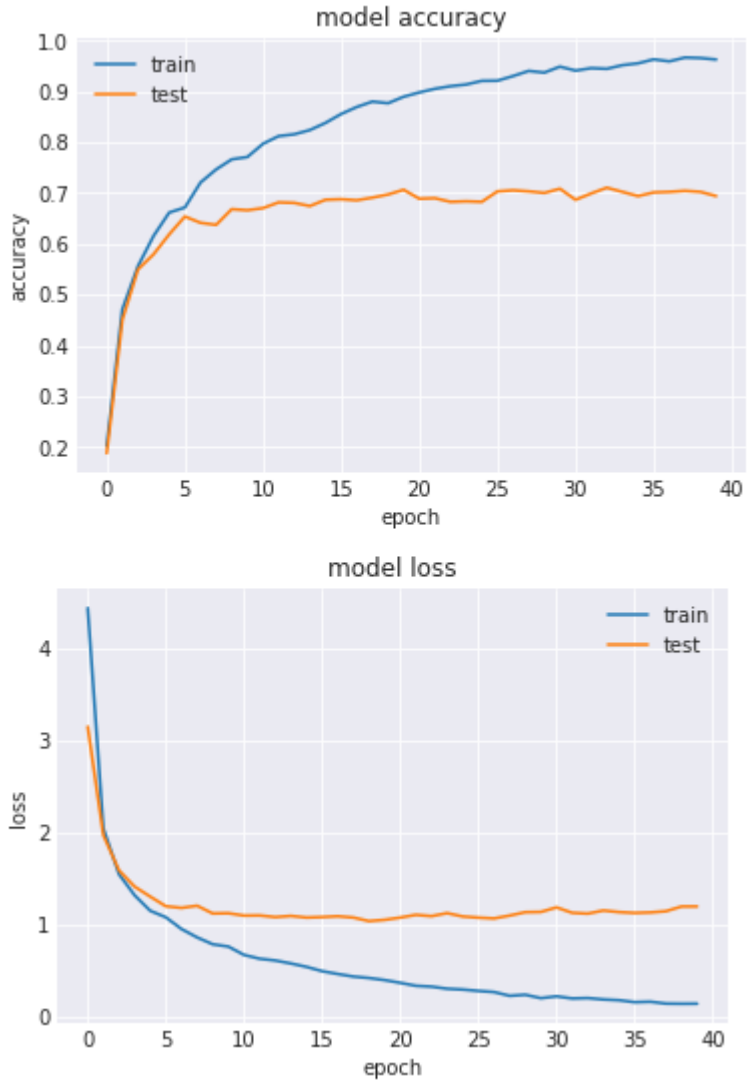
InV2 je první z modelů, u kterého bylo zapotřebí upravit parametry. Díky zvýšení počtu kroků na každý cyklus lze získat vyšší úspěšnost rychleji, bez potřeby zvyšovat počet cyklů. I v případě tohoto modelu se validační úspěšnost stále pohybuje okolo 70 %, zlepšení lze vidět ve validační ztrátě, která se udržuje okolo hodnoty 1,1 až těsně ke konci učení.



Obrázek 14 - Graf přesnosti a ztráty modelu InceptionV2 s upravenými parametry, zdroj: vlastní

4.4.5 InceptionV3

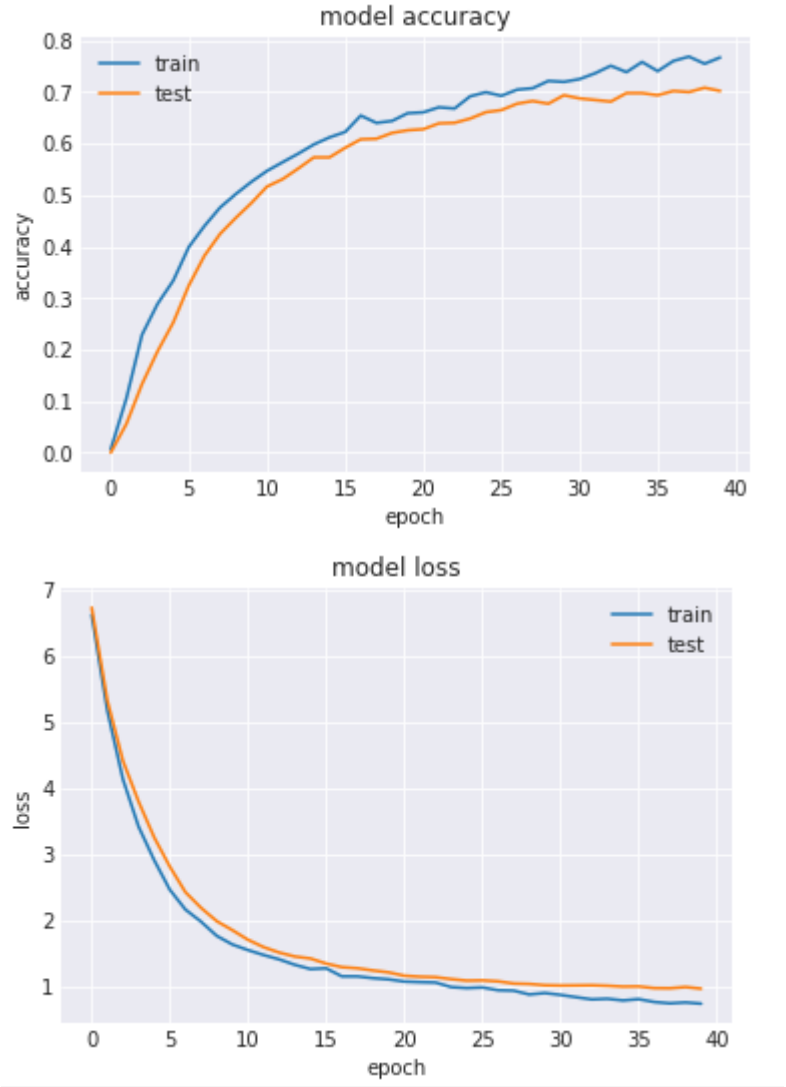
Jak již bylo zmíněno v kapitole výběru modelu, architektura InceptionV3 (dále InV3) byla objasněna ve stejné práci jako InceptionV2, tyto dva modely vycházejí ze stejného základního konceptu, ale liší se malými rozdíly. Tyto rozdíly mohou pomoci v určitých případech a typech problémů. Z grafu lze vidět zlepšení v rychlosti učení. V porovnání s InV2 dosáhl model InV3 trénovací úspěšnosti okolo 95 % již po základních 40 cyklech. Validační úspěšnost se také delší dobu pohybovala okolo 70 %. Bohužel v grafu zaznamenávajícím hodnoty ztráty, není vidět zlepšení oproti modelu InV2. Tyto výsledky byly očekávány na základě podobnosti těchto modelů.



Obrázek 15 - Graf přesnosti a ztráty modelu InceptionV3, zdroj: vlastní

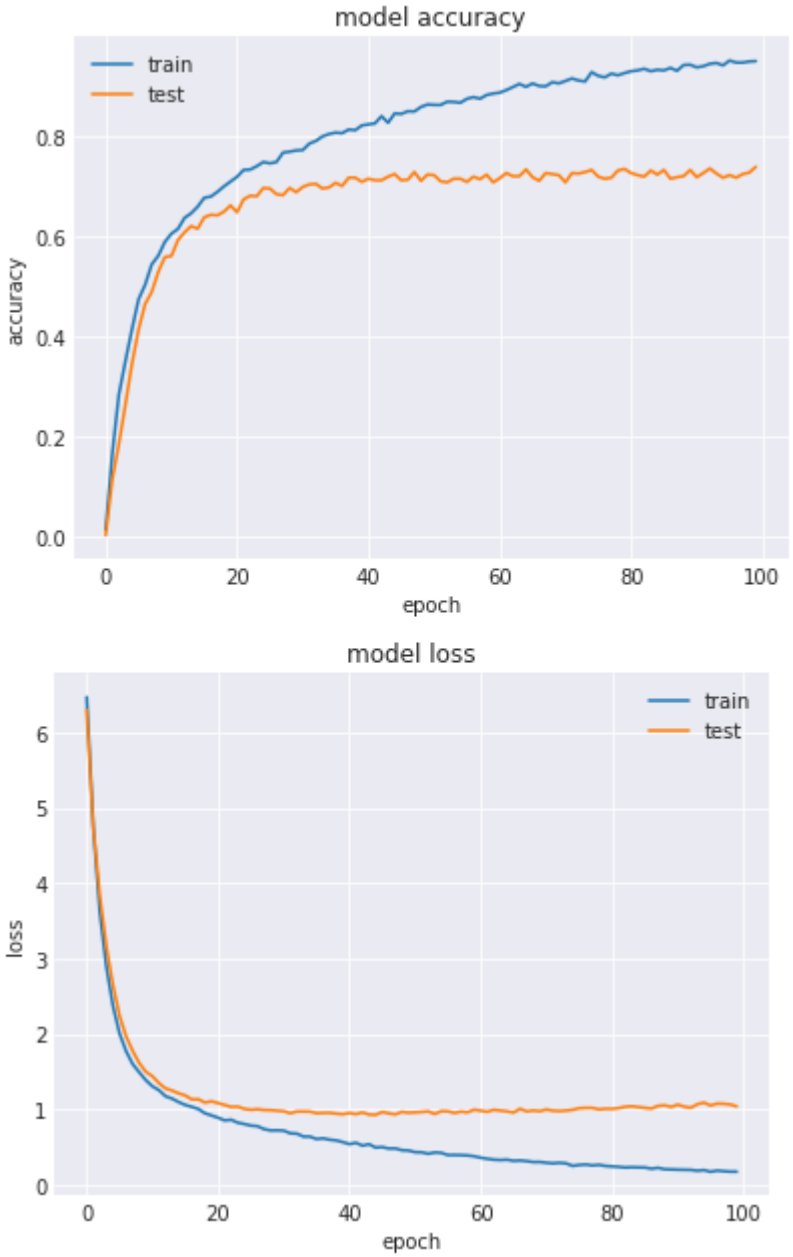
4.4.6 EfficientNetB3

Pro EfficientNet existují verze modelu od B0 po B7. Hlavními rozdíly jsou jiné rozměry vstupu a celkový počet vrstev. Nároky na výkon rostou s každou verzí modelu. Z důvodu limitace prostředí Google Colab a časové náročnosti práce s vyššími verzemi modelu, byl pro experiment zvolen model verze B3. První běh byl spuštěn se stejnými parametry jako předchozí experimenty.



Obrázek 16 - Graf přesnosti a ztráty modelu EfficientNetB3, zdroj: vlastní

Oproti všem předchozím výsledkům z jiných modelů, nedosáhl EfficientNet své maximální trénovací přesnosti a minimální ztráty po 40 cyklech. Také nelze určit, jestli se validační hodnoty ustálily nebo je stále možné získat lepší výsledky. Je tedy nutné upravit parametry, zvýšit batch size na 32, počet kroků navýšit na 160 a nastavit počet cyklů alespoň na 100.

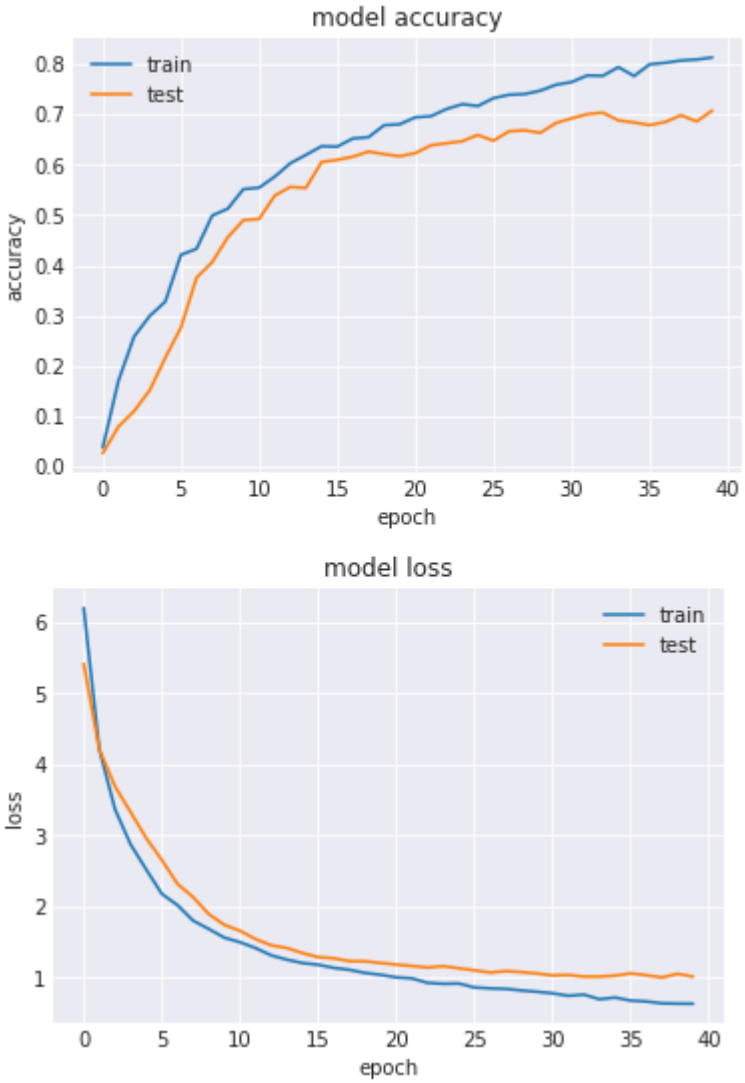


Obrázek 17 - Graf přesnosti a ztráty modelu EfficientNetB3 s upravenými parametry, zdroj: vlastní

Zde již bylo dosaženo vysoké úspěšnosti, ale i s delším průběhem a vyšším objemem vstupních dat na jeden cyklus, nebylo dosaženo lepší validační přesnosti než 70 %. Ztráty modelu se však blížily k 0,9, s malým nárůstem ke konci trénování.

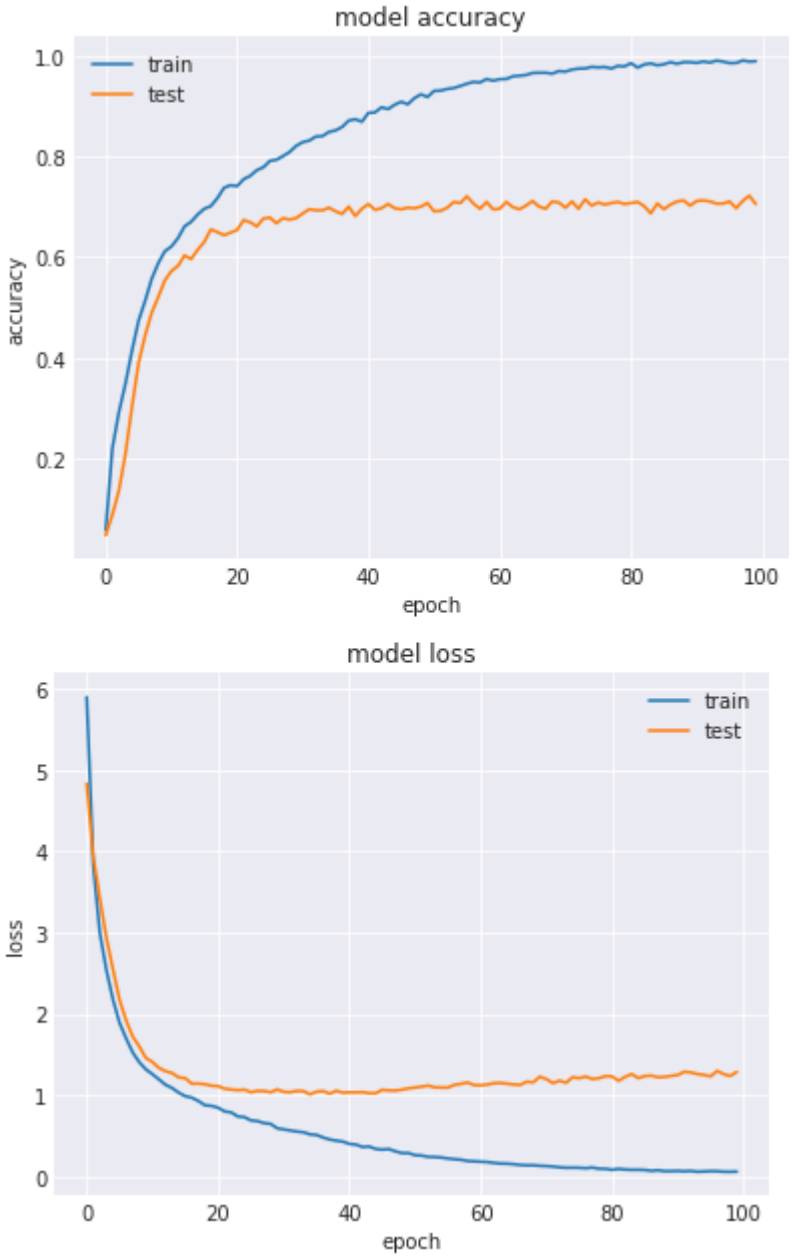
4.4.7 Xception

U experiment s Xception modelem byly očekávány podobné výsledky jako modely Inception, s pravděpodobností delšího zpracování díky vyšší komplexitě. Z následujícího grafu v obrázku dole je vidět stejná situace jako u modelu EfficientNet, běh se základním nastavením není dostatečně dlouhý nebo vyžaduje větší batch size a větší počet kroků.



Obrázek 18 - Graf přesnosti a ztráty modelu Xception, zdroj: vlastní

K získání ustálených výsledků byl běh prodloužen na 100 cyklů se zvýšenou batch size na 32 a počet kroků nastaven na 160.



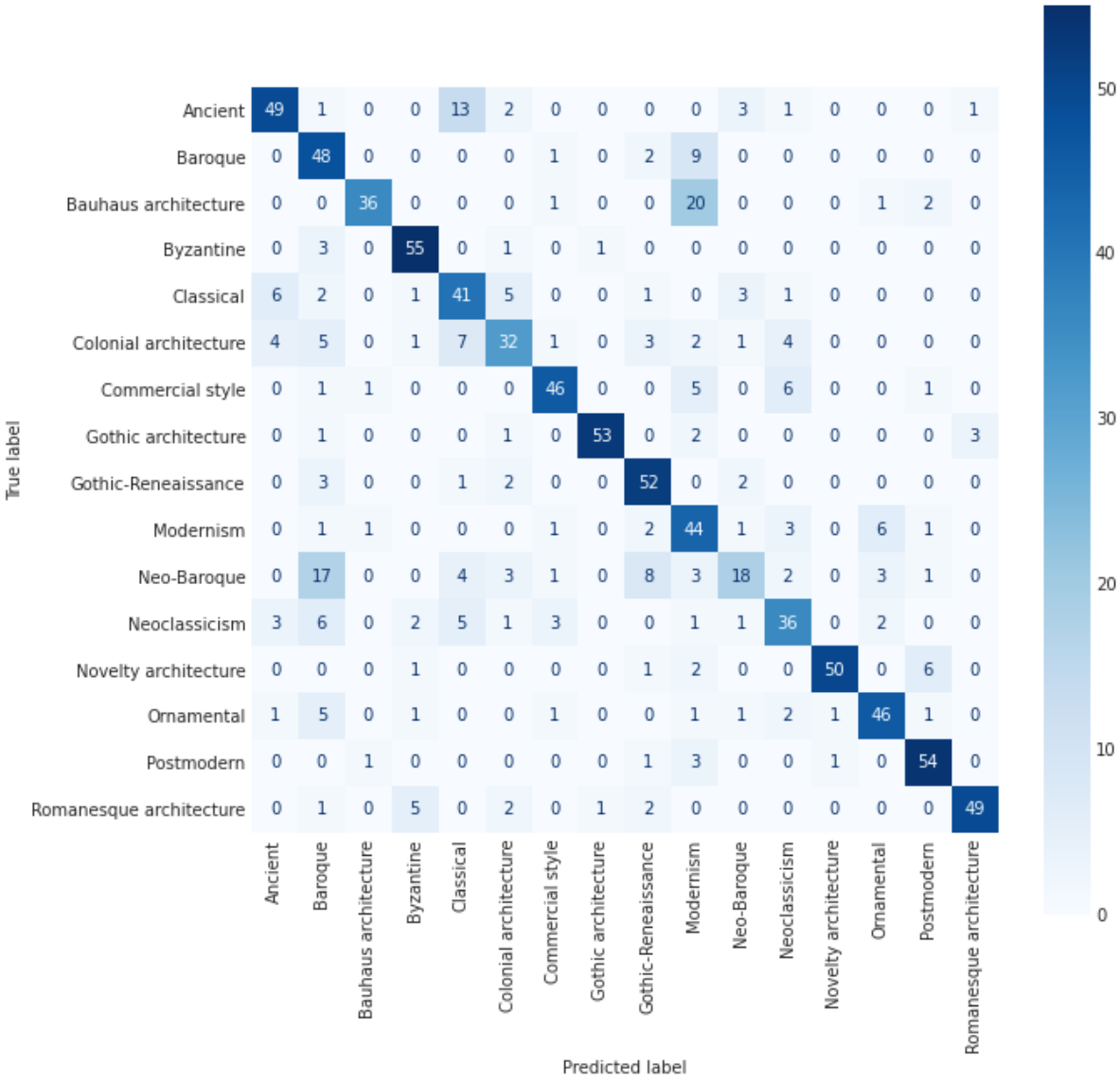
Obrázek 19 - Graf přesnosti a ztráty modelu Xception s upravenými parametry, zdroj: vlastní

Výsledná úspěšnost byla lepší než modely Inception, ale podobně jako EfficientNet i po dlouhém průběhu nedosáhl Xception model vyšší úspěšnosti než 70 %.

4.5 Vybraný model a jeho confusion matrix

Z pokusů s různými modely a parametry vyšlo najevo, že všechny modely jsou si podobné, ale konečné hodnoty modelu EfficientNetB3 byly stabilnější. Z tohoto důvodu byl tento model zvolen pro výslednou aplikaci.

Úspěšnost nebo naopak neúspěšnost modelu rozeznat rozdílné detaily ze vstupních dat, lze také reprezentovat takzvanou „maticí zmatení“, ve které obě osy X a Y obsahují jednotlivé třídy, osa X reprezentuje třídy odhadnuté umělou inteligencí a osa Y opravdové třídy, v každém protnutí os je číselná hodnota. Všechny hodnoty dohromady dávají celkový počet vstupních dat, nad kterými byl confusion matrix tvořen. Hodnoty uvnitř matice jsou součty odhadů dané třídy porovnané s opravdovou třídou. Například, má-li třída Modernism na ose X hodnotu 54 v bodě protínajícím se s třídou Modernism na ose Y, a hodnotu 16 v bodě protínajícím se s třídou Bauhaus architecture na ose Y a hodnoty 0 na všech ostatních místech znamená to, že ze 70 vstupních dat, které byly ze stylu Modernismus bylo 54 správně označeno a zbylých 16 bylo označeno jako špatná třída.



Obrázek 20 - Confusion Matrix modelu EfficientNetB3, zdroj: vlastní

4.6 Zhodnocení výsledků analýzy

Ve všech výsledných grafech můžeme vidět stejný problém, i když většina modelů dosahuje trénovací úspěšnosti vyšší než 95 %, u každého z nich se validační přesnost pohybuje kolem 70 % a validační ztráta se udržuje kolem hodnoty jedna. V grafech nezaznamenáváme klesající úspěšnost po větším počtu cyklů, to znamená, že problém overfittingu zmíněný v předchozí kapitole není příčinou nízké výsledné úspěšnosti. Protože validační přesnost nepřesahuje trénovací, a trénovací úspěšnost dosáhne dobrých výsledků, nejedná se o problém underfittingu. Díky využití datové augmentace a dostatečné velikosti počátečního datasetu, se nejedná ani o problém nedostatku dat. Jedinou zbývající možností je fundamentální problém v rozdílech mezi třídami datasetu. Podíváme-li se na confusion matrix (viz. obrázek nahoře) z předchozí kapitoly, můžeme vidět vysokou hodnotu zmatení mezi třídami Baroko a Neo-Baroko. Když si zobrazíme data z těchto tříd, pro třídu Neo-Baroko (viz. obrázek dole) a pro třídu Baroko (viz. obrázek dole), lze vidět podobnosti ve vlastnostech.

Obrázek 21 - Příklad dat z třídy Neo-Baroko, zdroj: vlastní

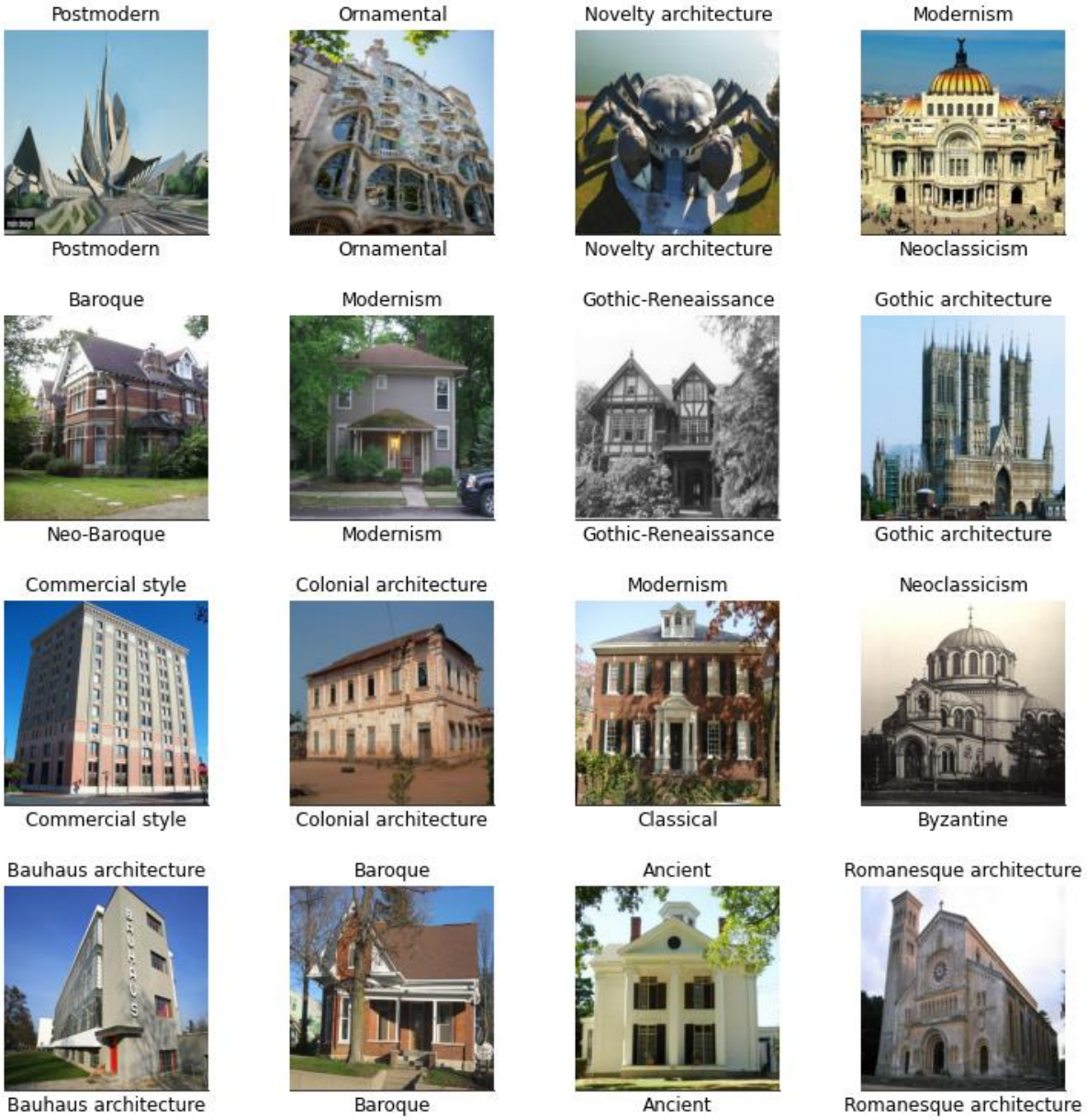
Model je schopný se naučit vlastnosti neobarokního stylu jako jsou například výrazné okenní rámy a vystouplé okraje. Avšak tyto vlastnosti jsou přeneseny z barokního stylu, ze kterého Neo-Barokní

styl vychází. A detailnější rozdíly, které jsou viditelné lidským okem, má model problém zaznamenat a naučit se je rozeznat.

Obrázek 22 - Příklad dat z třídy Baroko, zdroj: vlastní

4.7 Otestování naučeného modelu

Pro testování webu byly staženy dodatečné fotografie budov v různých architektonických stylech. Bylo důležité získat vstupní data, se kterými model dříve nepracoval, pro opravdové ověření výsledné úspěšnosti. Jak můžeme vidět v obrázku dole po provedení pokusného odhadu pro každou třídu, uspěl model ve dvanácti ze šestnácti odhadů. Tato úspěšnost je v rozmezí očekávaných hodnot. Z těchto výsledků lze odvodit, že model je schopen rozpoznat rozdíly mezi styly, a nejedná se pouze o slepé naučení všech originálních vstupních dat.



Obrázek 23 – Graf testovací predikce (opravdová třída dole, odhadnutá třída nahoře), zdroj: vlastní

5 TVORBA APLIKACE

Následující podkapitoly jsou zaměřeny na stavbu klientské aplikace, aplikace na serverové straně, potřebné knihovny a jiné části projektu, potřebné k jeho funkci.

5.1 Stavba webu

Pro tvorbu frontendu webové aplikace se nabízelo využití hned několik jazyků nebo frameworků. Webová aplikace by měla být lehká, responzivní a jednoduchá na ovládání. Musí také být napsaná v jazyce nebo frameworku, který umožňuje jednoduchý způsob tvorby vlastních http žádostí, nebo přímou komunikaci i s API, jako je například Flask. Volbu můžeme rozdělit do dvou hlavních kategorií. Webová aplikace, která poběží na serveru zároveň s backend částí projektu a webová aplikace, která poběží na straně klienta, a bude komunikovat s backend částí přes HTTP žádosti na danou webovou adresu. Pro frontend aplikace na straně serveru byl jedna z možností použít jazyk PHP a frameworky Django a Flask. Mezi možnosti pro web na straně klienta patří velice rozšířený jazyk JavaScript, nebo také populární framework React. V následujícím odstavci nastíníme výhody a nevýhody jednotlivých možností a odůvodníme následnou volbu. Hlavní výhodou PHP je odstranění jakékoliv potřeby klienta mít nainstalovaná různá prostředí jako je například Python. Veškerá komunikace je mezi klientem a serverem, kde poté na serveru jsou vytvořeny a zpracovány požadavky. Nevýhodou je závislost výkonu celé aplikace na vytíženosti serveru, kromě běhu samotné umělé inteligence musí server také tvořit požadavky na API, a jejich výsledky předat zpět klientovi. Z důvodu obav o vytíženost serveru a existující lepší volby, bylo PHP vyřazeno z výběru již brzy ve vývoji. Mezi další možnosti frontendu aplikace na straně serveru patřily frameworky Django a Flask. V posledních letech vznikají frameworky ve všech jazycích, s cílem zjednodušit a sjednotit stavbu webových aplikací. Oba tyto frameworky jsou napsány v jazyce Python a oba jsou převážně tvořeny pro běh na straně serveru. Tyto frameworky splňují požadavek na lehkost aplikace. Narozdíl od PHP, které může zatěžovat server v případě obdržení velkého počtu klientských žádostí. Zpočátku volba jednoho z těchto dvou frameworků vypadala jako nejlepší možnost. Po zamyšlení nad tím, proč tvořit aplikaci pouze na straně serveru, když lze využít dostupný výkon klientských zařízení, bylo rozhodnuto přesunout tuto část projektu na stranu klienta. I s tímto rozhodnutím, bude framework Flask využit v projektu, ale spíše jako API na komunikaci mezi klientem a samotným modelem umělé inteligence. Větší aplikace jsou dnes z velké části tvořeny frameworky, málokteré jsou psány přímo v jazyce, který daný framework používá. Bylo zapotřebí rozhodnout, bude-li aplikace na straně klienta dosahovat větších rozměrů, aby bylo výhodné nastudovat React framework narozdíl od sepsání aplikace v čistém JavaScriptu. Po

prostudování podkladů k frameworku React, vyšlo najevo, že sepsat lehkou klientskou aplikaci v čistém JavaScriptu bude časově výhodnější. Aplikace měla pouze několik cílů – jednoduché grafické rozhraní, responzivitu, svižnost a možnost vytvořit HTTP dotaz, který lze poslat na API pro komunikaci s umělou inteligencí. S dnešním výkonem uživatel nezaznamená, že se na jeho straně tvoří a posílají žádosti na API, které poté odpoví s výsledkem odhadu.

Výkonný kód aplikace na straně klienta lze rozdělit do tří hlavních kategorií – výběr a načtení obrázku, metoda pro tvorbu a odesílání HTTP žádostí a generování dodatečných HTML prvků na základě výstupu dotazu na databázi.

```
image_input = document.getElementById("#image-input");
image_input.addEventListener("change", function() {
  reader = new FileReader();
  reader.addEventListener("load", () => {
   uploaded_image = reader.result;
   document.getElementById("#display-image-IMG").src = `${uploaded_image}`;
  });
```

Obrázek 24 - Kód pro načtení souboru, zdroj: vlastní

Objekt uložený v proměnné *image_input* je HTML typu input, který akceptuje pouze soubory s obrazovými daty, specifikovány na formáty JPEG, JPG a PNG. Pokud je vybrán validní soubor, vytvoří se instance třídy FileReader, která dočasně nahraje soubor, převede ho na URI obsahující Base64. Dočasná adresa je použita pro zobrazení náhledu v okně aplikace. Nahraný soubor je potom přidán k HTTP žádosti, která je odeslána na API. Pro sestavení, odeslání a zpracování je využita lamba metoda *request*. Parametr *URL* je přesná adresa modulu ve Flask API, *method* je typ HTTP žádosti, v případě odesílání informací na server je tento parametr nastavený na POST, naopak pro získání dat, například z databáze, bude žádost typu GET. *Data* parametr je používaný pouze v případě posílání dat na server, kde je obsažen objekt typu FormData, který má klíč *predictionImage* a nahraný obrázek. Do *savedResponse* se zapíší informace obdržené z odpovědi na HTTP žádost, které lze uvnitř volání metody použít ke změně dat zobrazených uživateli.

```
let request = (url, method, data, savedResponse = (data) => {}) => {
  fetch(url, {
      method: method,
      body: data
    })
    .then(response => response.text())
    .then(data => {
      savedResponse(data);
    }).catch(exception => console.log(exception));
}
```

Obrázek 25 - Funkce pro stavbu žádostí, zdroj: vlastní

V přiloženém obrázku dole lze vidět postavenou žádost pro odeslání načteného souboru na server, který ho může zpracovat.

```
request("http://127.0.0.1/predict", "POST", formdata, (response) => {
```

Obrázek 26 - Žádost pro odeslání dat, zdroj: vlastní

Pro označení žádosti jako validní očekává funkce *fetch* v JavaScriptu návratovou hodnotu 200, která reprezentuje OK status. V případě jakéhokoliv jiného návratového kódu je chybová hláška zalogována do konzole a data na klientské straně se nezmění. Chybové hlášky, které vzniknou v případě validní žádosti, ale s problémem ve zpracování dat, jsou odeslány jako validní odpověď s kódem 200, na straně klienta jsou tyto chybové zprávy odchyceny a zobrazeny. Žádost pro dotaz na databázi je zobrazena v obrázku dole.

```
request("http://127.0.0.1/request?key=" + response, "GET", null, (response) => {
```

Obrázek 27 - Žádost pro získání dat ze serveru

V případě, že se nejedná o chybovou hlášku nebo neplatný návratový kód, je do adresy přidána odpověď z předchozího dotazu. Pro metodu GET nejsou přiložena data, potřebné argumenty jsou v hlavičce požadavku. Změna dat na straně klienta je poté zpracována JavaScriptem.

5.2 Běh umělé inteligence v prostředí Flask

Umělá inteligence byla vytrénována v jazyce Python na Google Colab a nejjednodušším způsobem, jak ji využít v rozpoznávání budov. je mít možnost komunikovat s ní ve stejném Python prostředí. I když framework Flask nebyl nejlepší volbou pro klientskou stranu aplikace, je naopak skvělou volbou jako prostředí, kde bude spuštěný model umělé inteligence, a zároveň jako API, které bude přijímat, zpracovávat a odpovídat na HTTP žádosti, příchozí z webového rozhraní aplikace na straně

klienta. Jak již bylo zmíněno, Flask je Python framework, načtení modelu a vah tedy spočívá pouze v importování správných knihoven Keras a TensorFlow a přiložení těchto souborů vedle Python souborů. Po provedení všech importů je potřeba převést model v JSON podobě, který byl vyexportován z prostředí Google Collab, zpět na modelový objekt z knihoven Keras. Stejně jako knihovny obsahují funkce pro převod modelu do JSON formátu, mají také možnost zpětného převodu. Po načtení modelu je zapotřebí vložit váhy, které vznikly v modelu při vyučení, bez vah by nebyl model schopný úspěšně tvořit odhady. Váhy jsou uloženy ve formátu H5, z tohoto důvodu lze použít váhy pouze pro Keras modely, jiné modely, například z knihoven FastAI nebo PyTorch, mají jinou strukturu vah a jednotlivé vrstvy modelu jsou také zpracovávány rozdílně v každé knihovně.

```
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
loaded_model.load_weights("modelWeights.h5")
```

Obrázek 28 - Kód pro načtení modelu, zdroj: vlastní

Ve frameworku Flask lze vytvořit rozdílné moduly, které jsou spuštěny, když na předem určenou adresu dorazí GET nebo POST HTTP žádost. Modul pro příjem HTTP žádosti obsahující data, která mají být zpracovávána, lze vidět v následujícím obrázku dole.

```
@app.route('/predict', methods=['POST'])
def predict_request():
    if flask.request.files.get('predictionImage'):
        file = flask.request.files['predictionImage']
        filename = secure_filename(file.filename)
        file.save(filename)
        resp = predict(filename)
        return CLASSES[resp]
```

Obrázek 29 - Flask modul pro zpracování příchozí žádosti, zdroj: vlastní

Řádek *@app.route('/predict', methods=['POST']),* za kterým následuje definice modulu *predict_request* značí, že každá příchozí POST HTTP žádost na adresu *adresaServeru/predict* má spustit tento modul, který poté zpracuje data přiložená v žádosti. Podmínka zaručuje, že žádost je souborového typu a obsahuje data s klíčem *predictionImage.* Není-li žádost typu „multipart/form-data“ nebo neobsahuje data se správným klíčem, server potvrzuje přijetí žádosti, ale odpovídá chybovou hláškou. V případě validních dat je název souboru převeden na bezpečnou podobu a soubor je dočasně uložen na server. Další zpracování je předáno modulu *predict.*

```
def predict(image_path, IMG_SIZE=224):
    image = Image.open(image_path)
    image = image.resize((IMG_SIZE, IMG_SIZE))
    image = np.asarray(image)
    image = np.expand_dims(image, axis=0)
    try:
        resp = loaded_model.predict(image)
        resp = np.argmax(resp[0])
    except:
        return "Error with image for prediction"
    return resp
```

Obrázek 30 - Flask modul pro provedení predikce, zdroj: vlastní

Modulu je předána cesta k lokálnímu umístění souboru a je nastavena velikost, na kterou je potřeba převést vstupní data. Po převedení dat na pole a jeho převedení na jednorozměrné pole, je volána funkce *predict*, nad načteným modelem. Z výstupu modelu je vybrána pouze první odpověď, která obsahuje odhad s nejvyšší pravděpodobností. V případě jakékoliv výjimky, která mohla vzniknout z důvodu chybného formátu dat nebo problému při tvorbě odhadu, modul vrací string s chybovou hláškou, který je předán dál a výsledně poslán jako odpověď na žádost z klientské aplikace.

5.2.1 Instalace prostředí Python a Flask

Pro spuštění backend části projektu, je zapotřebí mít nainstalované prostředí Python 3.7, knihovny pro framework Flask a rozšíření Flask-CORS. Pro usnadnění instalace knihoven byl použit správce balíčků Python PIP. Python lze nainstalovat přímo z webové stránky www.python.org nebo na systémech Windows z obchodu Microsoft. Na systémech Linux stačí podle používaného balíčkovacího systému nainstalovat balíček python3.7. Pro zjednodušení instalace knihovny Flask a jejího rozšíření, je vhodné na systému Windows nainstalovat správce balíčků PIP. Stačí vložit stažený soubor get-pip.py do složky, ve které je umístěn python.exe, v příkazovém řádku lze poté zavolat *python get-pip.py*, tento příkaz provede veškeré potřebné nastavení balíčkovacího systému PIP. Instalace Flask se provede příkazem *pip install Flask* a pro rozšíření Flask-CORS příkazem *pip install -U flask-cors.* Po instalaci je možné ověřit její správnost spuštěním souboru runserver.py, který by měl inicializovat model a připravit Flask API pro komunikaci s klientskou částí aplikace.

5.2.2 Flask CORS

V prohlížečích je systém CORS (Cross-Origin Resource Sharing) implementován pro omezení, ze kterých webů bude server přijímat a zpracovávat žádosti. Při tvorbě a testování aplikace, byl spuštně server na lokální adrese loopback 127.0.0.1 a klientská aplikace spuštěna přímo na zařízení. Tento rozdíl umístění zapříčinil, že žádosti na API nebyly zpracovány. Flask disponuje rozšířením Flask-CORS, které umožňuje vypnout CORS kontrolu. Pro projekt je toto řešení dostačující, v produkčním prostředí by bylo zapotřebí povolit adresu, ze které budou přicházet žádosti, nebo mít spuštěný web přímo na serveru s backend částí projektu.

5.3 Databáze pro klasifikaci fotografií

Databáze by měla stejně jako aplikace být lehká a responzivní a udržovat pouze potřebné informace o jednotlivých architektonických stylech. Relační databáze, jako je například SQL, jsou zbytečně komplikované a pomalé v porovnání s databázemi typu NoSQL, které lze nastavit na jakoukoliv aplikaci v řádu desítek minut. Mezi možnosti NoSQL patří MongoDB, Couchbase a Redis. MongoDB a Couchbase pracující s JSON formátem pro ukládání dat a Redis je takzvaná „in-memory“ databáze, kde data jsou skladována v paměti. Z důvodu minimální potřeby aktivního zápisu dat za běhu aplikace, je rychlost zápisu Redis databáze nepotřebná vlastnost. Flexibilita JSON formátu umožňuje navrhnout jakoukoliv strukturu databáze, a v případě MongoDB, i změnit strukturu v průběhu vývoje aplikace. Formát JSON umožňuje data upravovat i přímo v textovém editoru. Z těchto důvodů byla zvolena databáze MongoDB. Struktura databáze obsahuje položku *_id* držící unikátní klíč záznamu, který MongoDB přidává automaticky. Hlavní položkou je *name*, která nese název architektonického stylu, například baroko, renesance, gotický styl a další, a je používána jako klíč k získání dat z databáze. Položka *information*, obsahuje detailní informace o historii stylu, čím se vyznačuje a další. *YearOfOrigin* udržuje informaci, v jakém roce je zaznamenána první budova v daném stylu. *CountryOfOrigin* obsahuje zemi původu, kde styl vznikl. Pokud stále existují budovy zachovalé v jednom ze stylů, pole *existingBuildings* obsahuje jejich název. Dotaz je poslán na databázi p každé úspěšné žádosti na modul */predict* ve Flask API, který se vrátil s validní odpovědí. Pro komunikaci s databází je zapotřebí dalšího API rozhraní, protože čistý JavaScript, ve kterém je napsána klientská část aplikace, nemá přímou podporu jakékoliv databáze nebo komunikace s ní. První možností je implementace frameworku Node.js, který je postavený přímo nad jazykem JavaScript, a vybudovat klientskou stranu aplikace tímto frameworkem. Tento přístup má však několik problémů, hlavním z nich je potřeba přesunout aplikaci na stranu serveru. V předchozí kapitole bylo stanoveno, že cílem klientské aplikace a

grafického rozhraní je zůstat na straně klienta, server bude pouze zpracovávat žádosti přes Flask API a udržovat běh modelu. Druhým velkým problémem by bylo přidání dalšího frameworku, který musí být schopen komunikovat s Flask a stavět požadované HTTP žádosti. Oba tyto problémy byly vyřešeny stejnou knihovnou pro jazyk Python, díky volbě MongoDB jako databáze pro projekt, je možné nainstalovat knihovnu *pymongo*. Tato knihovna zpřístupňuje všechny funkce MongoDB pro Python nebo jakýkoliv framework postavený nad jazykem Python, v případě tohoto projektu framework Flask. Po instalaci přes správce balíčku PIP příkazem *pip install pymongo*, stačí importovat knihovnu a veškeré funkce pro CRUD operace jsou dostupné. K připojení do databáze stačí zavolat konstruktor MongoClient s parametry adresy a portu, na kterém běží MongoDBServer. Jako používaná databáze je zvolena *BPAI*, která v projektu obsahuje kolekci *StyleInformation*, ve které jsou uloženy veškeré informace ke všem architektonickým stylům, které je umělá inteligence schopna rozpoznat.

```
client = MongoClient('localhost', 27017)
db = client.BPAI
collectionOfStyles = db.StyleInformation
```

Obrázek 31 - Spojení s databází, zdroj: vlastní

K získání dat z databáze je používán modul *query_db*, který je zpracován při obdržení HTTP žádosti typu GET. V hlavičce žádosti je poslán argument *key*, ve kterém je uložen název stylu vrácený umělou inteligencí. Veškeré argumenty jsou uloženy v proměnné *style*. Funkce find_one nebo find pro vyhledávání v MongoDB vyžadují parametry v JSON formátu. Vyhledávací funkce vrací v případě find pole záznamů, funkce find_one vrací pouze první odpovídající záznam, obě funkce nevrací nic v případě, že databáze neobsahuje žádný odpovídající záznam. MongoDB pracuje s JSON formátem, ale protože používá ObjectID pro indetifikaci záznamů, výstup těchto funkcí je ve formátu BSON, který není implicitně podporován v HTTP žádostech nebo webových prohlížečích. Funkce dumps z knihovny json_util převede BSON do JSON formátu, který je prohlížeč schopný přijmout.

```
@app.route('/request', methods=['GET'])
def query_db():
    style = flask.request.args
    data = {"name" :style.get("key")}
    dbResult = collectionOfStyles.find_one(data)
    return json_util.dumps(dbResult)
```

Obrázek 32 - Flask modul pro dotaz na databázi, zdroj: vlastní

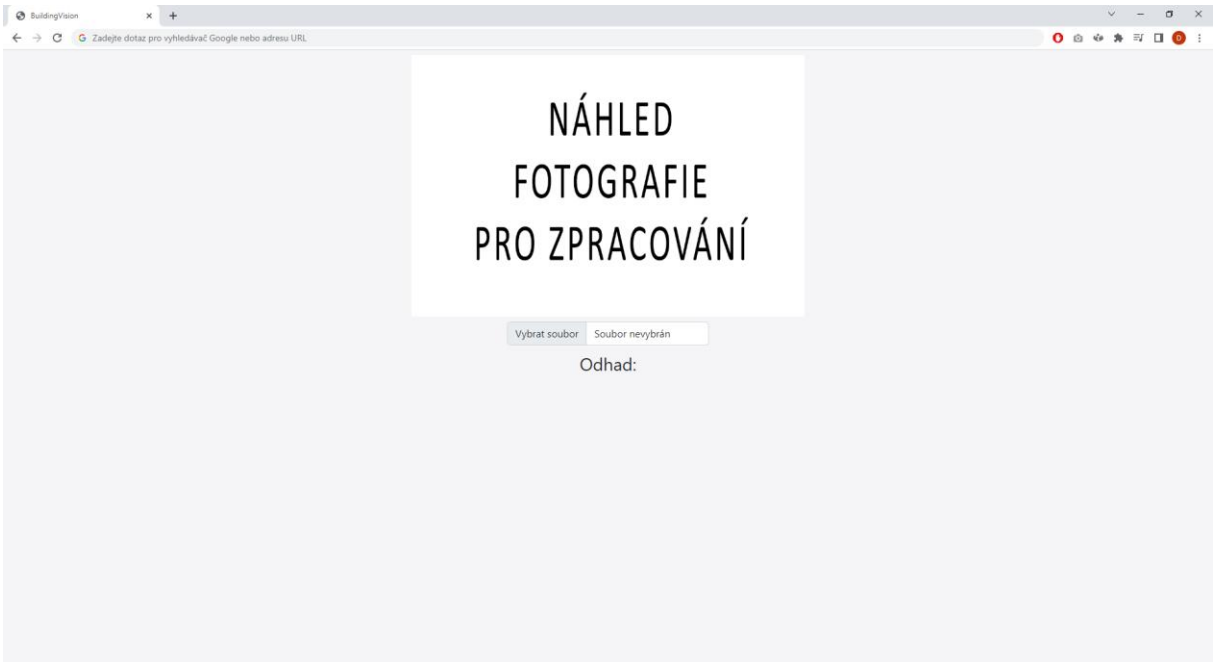
6 GRAFICKÉ ROZHRANÍ APLIKACE

6.1 Knihovna Bootstrap

Grafické rozhraní je tvořeno HTML prvky, které jsou stylizovány kaskádovými styly přes framework Bootstrap. Knihovna Bootstrap umožňuje stavět responzivní weby bez potřeby tvořit rozdílné kaskádové styly pro různá rozlišení. Díky takzvaným „breakpointům“ jsou všechny elementy webu správně zarovnány a zmenšeny nebo zvětšeny na základě rozlišení. Bootstrap obsahuje několik typů stylů, ze kterých je možné vybrat. Pro aplikaci byl zvolen základní minimalistický vzhled, pro přehlednost a jednoduchost.

6.2 Webová stránka před zpracováním fotografie

Úvodní stránka před zpracováním jakékoliv fotografie obsahuje pouze náhled, ve kterém se objeví nahraná fotografie, formulář pro nahrání souboru a textový element pro zobrazení výsledného odhadu.

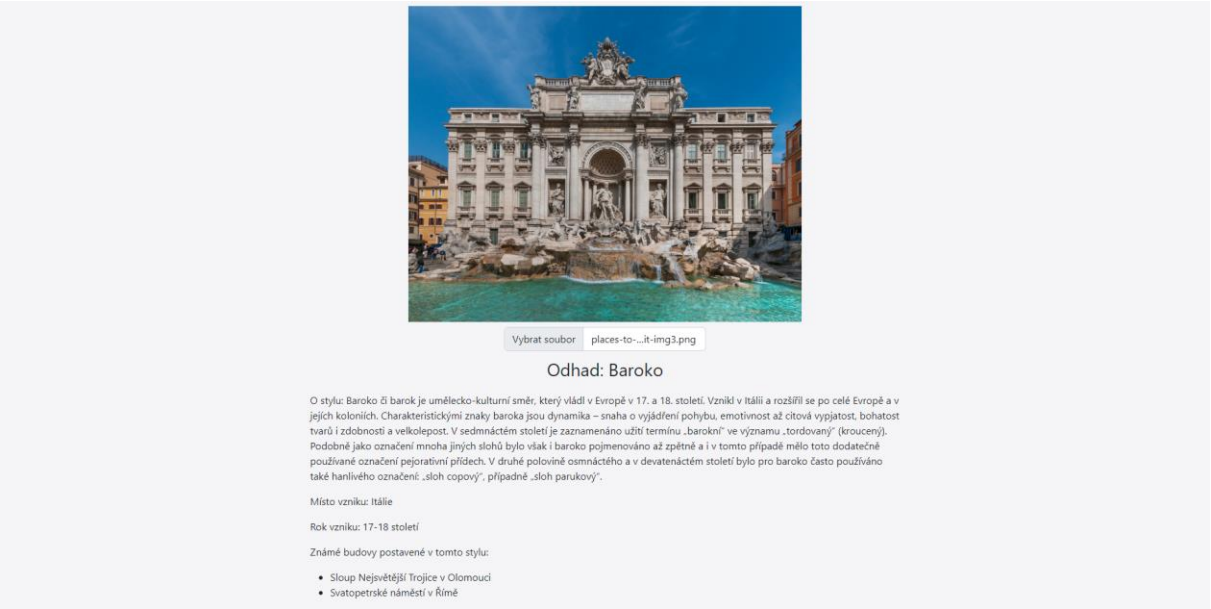


Obrázek 33 - Náhled stránky před predikcí, zdroj: vlastní

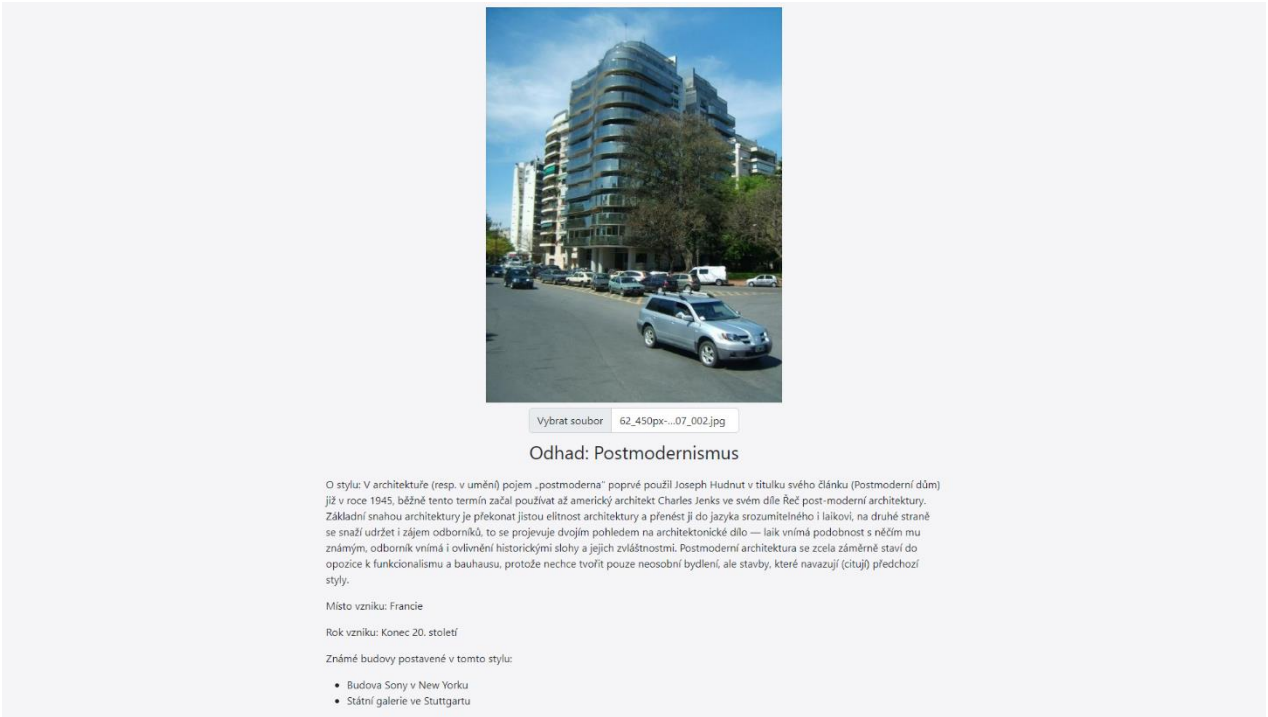
6.3 Webová stránka po zpracování fotografie

Po úspěšném nahrání souboru se náhled fotografie zobrazí na straně klienta, odešle se HTTP žádost a klientská aplikace čeká na odpověď serveru. Je-li server schopen zpracovat odeslanou žádost, vrácený odhad je zapsán do textového pole. V případě vrácení chybového stavu je textové pole

s odhadem nahrazeno chybovou hláškou a klient může nahrát nový soubor. V případě validního odhadu je odeslán dotaz na databázi. Získané informace se vloží do skrytých HTML elementů nebo vytvoří nové elementy, například v případě několika položek pro známé budovy. V obrázku dole je zobrazena celá stránka po zpracování všech žádostí. Seznam pro známé budovy je dynamicky rozšířen na základě počtu záznamů z databáze.



Obrázek 34 - Náhled stránky po provedení predikce baroka, zdroj: vlastní



Obrázek 35 - Náhled stránky po provedení predikce postmodernismu, zdroj: vlastní

Vybrat soubor 967_756px..._Lucca.JPG

Odhad: Románská architektura

O stylu: Románský sloh se prosadil ve stavitelství a výtvarném umění v zemích západní, jižní a střední Evropy v 11.-13. století. Východ Evropy byl v té době ovlivněn byzantským stylem. V architektuře šlo o postupně rozvíjenou, v rámci Evropy a dané epochy víceméně jednotnou tradici konstruování staveb a použití materiálu. Termín „románská“ v sobě zahrnuje pokus vyjádřit vztah tohoto stavebního slohu k stavebnímu slohu antického Říma, jímž byl do značné míry inspirován a sdílel s ním některé stavební prvky (viz klasickou řádovou architekturu).

Místo vzniku: 10-13. století

Rok vzniku: 6. století

Známé budovy postavené v tomto stylu:

- Románská katedrála ve Špýru
- Kostel svatého Michala v Hildesheimu

Obrázek 36 - Náhled stránky po provedení predikce románské architektury, zdroj: vlastní

ZÁVĚR

Cílem práce bylo vysvětlení teorie sítí umělé inteligence, vyučení modelu a tvorba aplikace pro využití vybraného modelu.

Část práce zaměřena na teoretické informace o existujících typech sítí, na strukturu a princip funkce CNN, využití aktivačních funkcí po výstupu vrstvy a dostupné modely použitelné pro klasifikaci. Po objasnění, které modely jsou dostupné pro projekt a popsání jejich výhod a nevýhod, bylo potřeba vysvětlit koncepty výběru správného datasetu a případnou augmentaci vstupních dat. U datasetu se jednalo o kritéria, která musí splňovat. Kapitola augmentace nastínila možnosti úprav vstupních dat, které umožní uměle zvýšit objem použitelných dat při vyučení modelu. Tyto úpravy jsou i vizuálně zobrazeny pro představu, jak ovlivňují data. S informacemi o datasetu a jeho úpravách, byly vysvětleny i problémy overfittingu a underfittingu.

Dále se práce zabývá implementací jednotlivých modelů v prostředí Google Colab, jejímž cílem je získat potřebná data, která jsou poté reprezentována grafy, ke zvolení nejlepšího z nich. Každý model má svoji vlastní podkapitolu, ve které jsou zmíněny použité parametry a úspěšnost. Výsledné zhodnocení obsahuje také confusion matrix, který umožňuje zobrazit, se kterými třídami má model největší potíže.

Poslední část práce je zaměřena na tvorbu aplikace na straně serveru a na straně klienta. Je zde objasněn kód potřebný ke tvorbě HTTP žádostí, které jsou využívány pro komunikaci mezi serverem a klientem, vysvětlena funkce frameworku Flask a jednotlivé rozdělení na výkonné moduly. Následuje krátké rozebrání grafického rozhraní, které demonstruje využití knihovny Bootstrap pro stylování webové stránky.

Výsledkem práce je analýza úspěšnosti modelů sítí CNN a funkční aplikace pro rozeznání architektonických stylů budov z fotografií. V části zabývající se teorií neurálních sítí jsou vysvětleny pojmy tak, aby čtenář porozuměl následujícím kapitolám, ale zároveň se neztratil v komplexitě, kterou mohou neurální sítě dosahovat. Za pomoci grafů jsou vysvětleny rozdíly modelů a potřebné parametry pro jejich trénování. Výsledná aplikace je poměrně úspěšná v rozpoznávání architektonických stylů, s možností budoucího zlepšení. Jak bylo zmíněno v rozboru, žádný z modelů nedosáhl předpokládaných procent úspěšnosti. Hlavním důvodem byly potíže v rozeznání malých detailů, které odlišovaly jednotlivé styly. V budoucnu je možné, že vznikne model, který bude lépe stavěný na problematiku, která byla řešena v této práci. Další z možností je rozšíření datasetu na desítky tisíc až statisíce unikátních fotografií. Samotný rozbor řady problémů, které způsobují nižší přesnost modelu, může být námětem na další práci.

POUŽITÁ LITERATURA

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

[2] BISHOP, Christopher M. Pattern recognition and machine learning. [New York]: Springer, c2006. Information science and statistics. ISBN 0-387-31073-8.

[3] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. Deep learning. Cambridge, MA: MIT press, [2016]. Adaptive computation and machine learning series. ISBN 9780262035613.

[4] KRISHNA, Sajja Tulasi a Kumar Kalluri HEMANTHA. Deep learning and transfer learning approaches for image classification. International Journal of Recent Technology and Engineering (IJRTE) [online]. February 2019, (Volume-7, Issue-5S4), 427-432 [cit. 2022-01-26]. ISSN 2277-3878. Dostupné z: https://www.researchgate.net/profile/Hemantha-Kumar-Kalluri/publication/333666150_Deep_Learning_and_Transfer_Learning_Approaches_for_Image_Classification/links/5cfbeeb9a6fdccd1308d6aae/Deep-Learning-and-Transfer-Learning-Approaches-for-Image-Classification.pdf

[5] SCHMIDHUBER, J. Multi-column deep neural networks for image classification. 2012 IEEE Conference on Computer Vision and Pattern Recognition [online]. IEEE, 2012, 2012, (Volume-7, Issue-5S4), 3642-3649 [cit. 2022-01-26]. ISBN 978-1-4673-1228-8. ISSN 2277-3878. Dostupné z: doi:10.1109/CVPR.2012.6248110

[6] TRIPATHI, Milan. Analysis of Convolutional Neural Network based Image Classification Techniques. Journal of Innovative Image Processing [online]. IEEE, 2021, 2012, 3(2), 100-117 [cit. 2022-01-26]. ISBN 978-1-4673-1228-8. ISSN 2582-4252. Dostupné z: doi:10.36548/jiip.2021.2.003

[7] ALBAWI, Saad, Tareq Abed MOHAMMED a Saad AL-ZAWI. Understanding of a convolutional neural network. 2017 International Conference on Engineering and Technology (ICET) [online]. IEEE, 2017, 2017, 3(2), 1-6 [cit. 2022-01-26]. ISBN 978-1-5386-1949-0. ISSN 2582-4252. Dostupné z: doi:10.1109/ICEngTechnol.2017.8308186

[8] JAVIDI, Bahram. Image Recognition and Classification. 2017 International Conference on Engineering and Technology (ICET) [online]. IEEE, 2017, 2002-6-14, 3(2), 1-6 [cit. 2022-01-26]. ISBN 978-1-5386-1949-0. ISSN 2582-4252. Dostupné z: doi:10.1201/9780203910962

[9] KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems 25,

editace F. Pereira; C. J. C. Burges; L. Bottou; K. Q. Weinberger, Curran Associates, Inc., 2012, s. 1097–1105.

[10] SEIDE, F.; AGARWAL, A.: CNTK: Microsoft's Open-Source Deep-Learning Toolkit. In Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, New York, NY, USA: ACM, 2016, ISBN 978-1-4503-4232-2, s. 2135–2135, doi:10.1145/2939672.2945397.

[11] BROWNLEE, Jason. *Crash Course on Multi-Layer Perceptron Neural Networks* [online]. 17.5.2016 [cit. 2022-08-02]. Dostupné z: https://machinelearningmastery.com/neural-networks-crash-course/

[12] ABIRAMI, S. a P. CHITRA. Energy-efficient edge based real-time healthcare support system. In: *The Digital Twin Paradigm for Smarter Systems and Environments: The Industry Use Cases* [online]. Elsevier, 2020, 2020, s. 339-368 [cit. 2022-08-02]. Advances in Computers. ISBN 9780128187562. Dostupné z: doi:10.1016/bs.adcom.2019.09.007

[13] SAHA, Sumit. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* [online]. 15.12.2018 [cit. 2022-08-02]. Dostupné z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[14] TRAORE, Boukaye Boubacar, Bernard KAMSU-FOGUEM a Fana TANGARA. Deep convolution neural network for image recognition. *Ecological Informatics* [online]. 2018, 48, 257-268 [cit. 2022-08-02]. ISSN 15749541. Dostupné z: doi:10.1016/j.ecoinf.2018.10.002

[15] O'SHEA, Keiron a Ryan NASH. *An Introduction to Convolutional Neural Networks* [online]. 2015 [cit. 2022-08-02]. Dostupné z: https://arxiv.org/pdf/1511.08458.pdf. Aberystwyth University, Lancaster University.

[16] Wikipedia contributors. *Recurrent neural network* [online]. Wikipedia, The Free Encyclopedia; 2022 Jun 26, 22:42 UTC [cit. 2022-08-02]. Available from: https://en.wikipedia.org/w/index.php?title=Recurrent_neural_network&oldid=1095187237

[17] IBM Cloud Education. Recurrent Neural Networks. *IBM* [online]. 14.9.2020 [cit. 2022-08-02]. Dostupné z: https://www.ibm.com/cloud/learn/recurrent-neural-networks

[18] DIPIETRO, Robert a Gregory D. HAGER. Deep learning: RNNs and LSTM. In: *Handbook of Medical Image Computing and Computer Assisted Intervention* [online]. Elsevier, 2020,

2020, s. 503-519 [cit. 2022-08-02]. ISBN 9780128161760. Dostupné z: doi:10.1016/B978-0-12-816176-0.00026-0

[19] Wikipedia contributors. *Deep belief network* [online]. Wikipedia, The Free Encyclopedia; 2021 Mar 17, 19:28 UTC [cit. 2022-08-02. Available from: https://en.wikipedia.org/w/index.php?title=Deep_belief_network&oldid=1012688873

[20] SHAJUN NISHA, S., M. MOHAMED SATHIK a M. NAGOOR MEERAL. Application, algorithm, tools directly related to deep learning. In: *Handbook of Deep Learning in Biomedical Engineering* [online]. Elsevier, 2021, 2021, s. 61-84 [cit. 2022-08-02]. ISBN 9780128230145. Dostupné z: doi:10.1016/B978-0-12-823014-5.00007-7

[21] ZHANG, Nan, Shifei DING, Jian ZHANG a Yu XUE. An overview on Restricted Boltzmann Machines. *Neurocomputing* [online]. 2018, 275, 1186-1199 [cit. 2022-08-02]. ISSN 09252312. Dostupné z: doi:10.1016/j.neucom.2017.09.065

[22] SODANI, Prerna. Restricted Boltzmann Machine and Its Application. *LatentView* [online]. 9.9.2020 [cit. 2022-08-02]. Dostupné z: https://www.latentview.com/blog/restricted-boltzmann-machine-and-its-application/#:~:text=Restricted%20Boltzmann%20Machines%20are%20stochastic,two%20layers%20visible%20and%20hidden

[23] CRUZ MARTINEZ, Juan. *Introduction to Convolutional Neural Networks CNNs* [online]. 8.12.2020 [cit. 2022-08-02]. Dostupné z: https://aigents.co/data-science-blog/publication/introduction-to-convolutional-neural-networks-cnns

[24] VANDIT, Jain. Everything you need to know about “Activation Functions” in Deep learning models. *Towards Data Science* [online]. 30.12.2019 [cit. 2022-08-04]. Dostupné z: https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253

[25] Image Classification on ImageNet. *Papers With Code* [online]. [cit. 2022-08-04]. Dostupné z: https://paperswithcode.com/sota/image-classification-on-imagenet

[26] M, Siddharth. Understanding ResNet and analyzing various models on the CIFAR-10 dataset. *Analytics Vidhya* [online]. 23.6.2021 [cit. 2022-08-04].

Dostupné z: https://www.analyticsvidhya.com/blog/2021/06/understanding-resnet-and-analyzing-various-models-on-the-cifar-10-dataset/

[27] OOMMEN, Prem. ResNets — Residual Blocks & Deep Residual Learning. *Towards Data Science* [online]. 28.11.2020 [cit. 2022-08-04]. Dostupné z: https://towardsdatascience.com/resnets-residual-blocks-deep-residual-learning-a231a0ee73d2#:~:text=A%20residual%20block%20is%20a,layer%20in%20the%20main%20path

[28] IOFFE, Sergey a Christian SZEGEDY. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift* [online]. New York, 2015 [cit. 2022-08-04]. Dostupné z: https://arxiv.org/pdf/1502.03167v3.pdf. Cornell University.

[29] BRITAL, Anas. Inception V2 CNN Architecture Explained. *Medium* [online]. 23.10.2021 [cit. 2022-08-04]. Dostupné z: https://medium.com/@AnasBrital98/inception-v2-cnn-architecture-explained-128464f742ce

[30] BHARATH, Raj. A Simple Guide to the Versions of the Inception Network. *Towards Data Science* [online]. 29.5.2018 [cit. 2022-08-04]. Dostupné z: https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

[31] THAKUR, Rohit. Step by step VGG16 implementation in Keras for beginners. *Towards Data Science* [online]. 6.8.2019 [cit. 2022-08-04]. Dostupné z: https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c#:~:text=VGG16%20is%20a%20convolution%20neural,vision%20model%20architecture%20till%20date.

[32] XU, Zhe, Dacheng TAO, Ya ZHANG, Junjie WU a Ah Chung TSOI. Architectural Style Classification Using Multinomial Latent Logistic Regression. In: FLEET, David, Tomas PAJDLA, Bernt SCHIELE a Tinne TUYTELAARS, ed. *Computer Vision* – ECCV 2014 [online]. Cham: Springer International Publishing, 2014, 2014, s. 600-615 [cit. 2022-08-04]. Lecture Notes in Computer Science. ISBN 978-3-319-10589-5. Dostupné z: doi:10.1007/978-3-319-10590-1_39

[33] IBM Cloud Education. Overfitting. *IBM* [online]. 3.4.2021 [cit. 2022-08-02]. Dostupné z: https://www.ibm.com/cloud/learn/recurrent-neural-networks

[34] DONGES, Niklas. A Guide to RNN: Understanding Recurrent Neural Networks and LSTM Networks. *Built In* [online]. 29.7.2021 [cit. 2022-08-04]. Dostupné z: https://builtin.com/data-science/recurrent-neural-networks-and-lstm