Detekcia a rozpoznávanie mikroskopických objektov v obraze

Lucia Hajduková

3AIb, 2018 - 2019

Abstrakt. Cieľom práce je vytvoriť program schopný lokalizovať mikroskopické objekty na upravených snímkach z mikroskopu. Problém bude riešený pomocou konvolučnej neurónovej siete zvanej U-sieť, ktorá bude objekty detegovať a vyznačovať dohodnutým spôsobom.

Kľúčové slová: strojové učenie, neurónové siete, konvolučné siete, U-siete, rozpoznávanie obrazu, TensorFlow, Keras

1. Úvod

1.1. Motivácia

Využitím techník rozpoznávania obrazu sa podarilo vyriešiť už mnoho komplexných úloh od jednoduchej zmeny kontrastu či jasu až po detekciu hrán a objektov. Týmto prístupom je v súčasnosti implementovaný aj program na detekciu častíc, ktorý chceme touto prácou upraviť použitím strojového učenia. S nárastom popularity strojového učenia sa totiž ukázalo, že dokáže k problémom pristupovať omnoho univerzálnejšie a pri vhodnej implementácii dosahuje porovnateľné výsledky ako klasický prístup rozpoznávania obrazu. Preto je našou víziou pristúpiť k detekcii mikroskopických častíc v snímkach z pohľadu strojového učenia a vytvoriť model rozpoznávajúci častice pomocou U-siete.

1.2. Úvod do problematiky

V posledných rokoch sa neurónové siete stali silným nástrojom na riešenie mnohých úloh. V oblasti rozpoznávania obrazu sa ako najlepšia voľba ukázali *konvolučné siete*. V ceste rozširovania ich použiteľnosti však stál zásadný problém so získavaním obrovského množstvu dát potrebného na natrénovanie siete. Okrem toho často boli potrebné siete s mnohými vrstvami a parametrami, čím sa úloha komplikovala.

Typicky konvolučné siete riešia problém *klasifikácie*, to znamená priradenie vstupu do prislúchajúcej triedy. Asi najznámejším príkladom je detekcia číslic na sade dát MNSIT [4]. Každému obrázku, na ktorom je vyobrazená ručne písaná číslica, je priradená hodnota 0-9, ktorá má zodpovedať napísanej číslici.

V našej práci je však úlohou klasifikovať každú bunku na snímke, nie obrázok ako celok. Očakávaný výstup teda má byť doplnený o lokalizáciu. Tento problém je známy pod názvom *segmentácia*. Cieľom je každému pixelu vstupného obrázka priradiť prislúchajúcu skupinu (segment). Výstupom je segmentovaný obrázok rovnakých rozmerov s vyznačením príslušnosti pixelov do skupín. Na tento problém sú v rozpoznávaní obrazu známe metódy *watershed*, metóda posuvného okna a práhovanie.

Neurónové siete získali svoje uplatnenie v oblasti segmentácie obrázkov v roku 2015, keď skupina nemeckých vedcov vyvinula *U-sieť*, ktorá prekonala výkonnosť dovtedy najspoľahlivejšej metódy posuvného okna (*sliding-window method*) [1]. Výhodou tejto siete je aj to, že rieši aj vyššie spomínaný problém s nutnosťou veľkej tréningovej množiny dát. U-sieť získala hlavnú cenu na *EM segmentation challenge* v ISBI 2012.

V tomto článku vysvetlíme pojmy, s ktorými sa v našej bakalárskej práci stretneme a uvedieme zaujímavé časti implementácie, ktoré prispejú k ujasneniu problematiky i prístupu riešenia.

1.3 Základné pojmy

Na to aby sme mohli modelovať neurónovú sieť a rozumieť jej správaniu, potrebujeme poznať niekoľko dôležitých pojmov.

Neurónová sieť

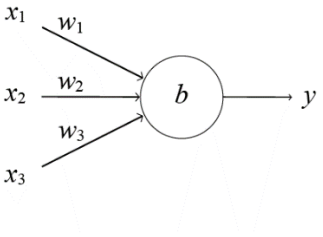
K rozpoznávaniu obrazu sa dá pristupovať tak, že definujeme určité pravidlá, na základe ktorých bude stroj schopný rozhodnúť, či sa na obrázku nachádza hľadaný vzor. V reálnom svete však ani taký jednoduchý objekt ako je jablko nevieme jednoznačne popísať. To, že je okrúhle a červené ešte nemusí znamenať, že je to jablko a jablko vlastne ani nie je presne okrúhle, ale každé je trochu inak deformované.

Pre tento typ problému je vhodný prístup učenia neurónovými sieťami, čo je v súčasnosti rýchlo rozvíjajúca sa oblasť informatiky. Neurónová sieť je algoritmus, ktorý sa dokáže učiť správne riešiť požadovanú úlohu. Po dostatočne dlhom trénovaní by mala byť sieť schopná správne odpovedať vo väčšine prípadov.

Neurón

Ako sme opísali vyššie, od neurónovej siete očakávame istý druh inteligencie – schopnosť vedieť správne rozoznávať povahu vstupov. Keďže náš mozog v úlohách tohto typu funguje výborne, stal sa vzorom pre vznik neurónových sietí.

V mozgu sa informácie prenášajú prostredníctvom neurónov, ktoré sú schopné prijať niekoľko vstupov a určiť vhodný výstup. Podľa [2] najjednoduchší model neurónu (*perceptrónu*) navrhol americký psychológ Frank Rosenblatt a vyzerá takto:



Obrázok 1. $x_1 \ldots x_n$ sú binárne vstupy s váhami $w_1 \ldots w_n$, kruh reprezentuje neurón, b bázu a y výstup.

Každému vstupu $x_1 \ldots x_n$ je priradená váha $w_1 \ldots w_n$ reprezentujúca dôležitosť vstupu. Výstup perceptrónu y môže byť len 1 (pravda) alebo 0 (nepravda) a je určený podľa jednoduchého pravidla:

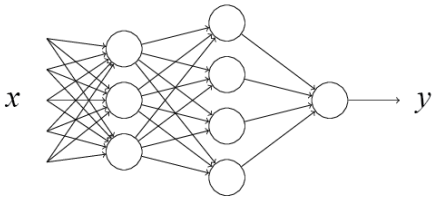
$$y = \begin{cases} 0, & \sum_j w_j x_j < t \\ 1, & inak \end{cases}$$

Rovnica 1. Hodnota t je prah, podľa ktorého sú vstupy zaradené do jednej z dvoch tried.

Samozrejme, nie vždy je hľadaná binárna odpoveď, ale sieť môže napríklad triediť vstupy medzi niekoľko rôznych tried. Vtedy je potrebný zložitejší model neurónu, napríklad sigmoidálny, ktorý vracia výstup v rozmedzí 0 a 1, pretože je na výstup aplikovaná sigmoidálna aktivačná funkcia. Aktivačným funkciám sa budeme venovať neskôr.

Vrstvový model neurónovej siete

Neuróny sú v ľudskom mozgu pospájané do komplexnej siete, čo umožňuje vytvárať široké súvislosti, posilňovať niektoré väzby a podobne. Na rovnakom princípe je vybudovaný aj model neurónovej siete v informatike. Často sa používa model, ktorý tvoria *vrstvy* zoskupujúce neuróny zaoberajúce sa spoločným problémom.



Obrázok 2. Rad neurónov nad sebou na obrázku predstavuje príslušnosť k tej istej vrstve. Hodnota x je viacnásobný vstup a y výstup.

Model na Obrázok 22 má tri vrstvy. Prvá robí jednoduché rozhodovanie na základe vstupov a jej výstup je poslaný na vstup druhej vrstvy. Tá je už zodpovedná za komplexnejšie rozhodnutie a podobne to funguje medzi každou ďalšou dvojicou vrstiev. To naznačuje, že čím viac vrstiev má sieť, tým presnejšie dokáže rozhodovať.

Dáta

Dáta pre neurónovú sieť sú rozdelené do nasledujúcich skupín:

1. Tréningové vzorky. Sú to dvojice vstup a očakávaný výstup, na ktorých je sieť trénovaná.
2. Testovacie vzorky. Tieto dáta sa použijú na testovanie siete. Sú to vlastne vstupy (rôzne od vstupov v tréningovej vzorke), ku ktorým sieť vypočíta prislúchajúce výstupy.
3. Vyhodnocovacie vzorky. Na týchto dátach odlišných od predošlých dvoch sa overuje, či sieť dokáže správne reagovať na celkom nové vstupy.

Počet dávok

V anglickej literatúre sa v súvislosti s trénovaním siete stretávame s pojmom *batch size* (označme *bs*), ktorý označuje počet vzoriek, s ktorými sieť pracuje pred tým než upraví nastavenie parametrov. Po každých *bs* vzorkách sú výsledky porovnané s očakávaným výsledkom, upravia sa parametre a prepočíta chyba. Od *bs* závisí, na koľko krokov prejdeme všetky tréningové vzorky.

$$k = ceil(n / bs), \qquad 1 \leq bs \leq n$$

Rovnica 2. Počet krokov *k* (za 1 epochu) je rovný pomeru veľkosti tréningovej množiny *n* a počtu vzoriek *bs* zaokrúhleného nahor.

Epocha

Počet epoch hovorí o tom, koľko krát algoritmus učenia použije celú sadu tréningových vzoriek. Jedna epocha môže použiť jednu alebo viac dávok. Epoch zvykne byť veľa, pretože po každej z nich sa sieť niečo nové naučí a je vhodné, aby to aplikovala na vylepšenie výstupu. Epocha aj dávka sú hyperparametre, teda parametre pre proces učenia, nie interné dané výsledkom učenia. Porovnanie dávky a epochy približuje zdroj [6].

Stochastický pokles gradientu

Algoritmu optimalizácie na trénovanie algoritmov strojového učenia sa hovorí stochastický pokles gradientu. Je to iteratívny proces úpravy parametrov počas trénovania. Každou iteráciou by sa mala zmenšovať chyba. Názov vyjadruje, že počítame gradient (svah, vývoj) chyby, ktorý sa zmenšuje (klesá). Pre neurónové siete sa najčastejšie používa *algoritmus spätnej propagácie (backpropagation)*.

Aktivačná funkcia

Každý neurón počíta vážený súčet vstupov a pripočítava k nemu bázu. Výsledkom toho je reálne číslo, no neurón sa potrebuje binárne rozhodnúť, či sa aktivuje alebo nie, takže na tento výsledok potrebujeme aplikovať funkciu, ktorá rozhodne, či sa neurón má aktivovať. Túto funkciu nazývame aktivačná, pretože rozhoduje o aktivovaní neurónu.

$$y = \sum_j w_j x_j + b$$

Rovnica 3. Výstup neurónu (bez aktivačnej funkcie) y. $x_1 \ldots x_n$ sú binárne vstupy s váhami $w_1 \ldots w_n$, b je báza neurónu.

Najjednoduchší typ aktivačnej funkcie je *kroková*, ktorá vráti 0 (neaktivovať) alebo 1 (aktivovať). To však spôsobuje nerozhodnosť v prípade, že v jednej vrstve viaceré neuróny vrátia 1, takže nie je jasné, ktorý neurón sa má aktivovať (mal by sa len jeden zodpovedajúci správnemu výsledku). Preto je výhodnejšie, aby aktivačná funkcia vrátila percentuálnu hodnotu vyjadrujúcu pravdepodobnosť, že daná odpoveď je správna.

$$A(y) = max(0, y)$$

Rovnica 4. Aktivačná funkcia *ReLu*.

V našej implementácii používame aktivačnú funkciu *ReLu (Rectified Linear Unit)*, ktorá je odporúčaná pre všetky skryté vrstvy. Jedným z dôvodov je to, že je to nelineárna funkcia, takže aj jej skladaním dostaneme nelineárnu funkciu. To je dobré, pretože to znamená, že výstup nezávisí len od vstupu, ale aj od výpočtov v skrytých vrstvách (ak by bola funkcia lineárna, viacvrstvový model by nemal zmysel). Ďalšou výhodou *ReLu* je, že zredukuje vstupy, tým, že každej zápornej hodnote priradí nulu. To znamená, že stačí spracovať menšie množstvo čiastočných výstupov. Neurónová sieť je vďaka tomu redšia (ľahšia) a rýchlejšia.

Nevýhodou je, že keď neurónu priradíme 0, gradient bude vždy 0, pretože váhy sa neprepočítajú. To znamená, že neuróny v tejto oblasti prestanú reagovať na zmeny chyby – tomu sa anglicky hovorí *dying ReLu problem*. Preto sa niekedy používa obmenená *ReLu* funkcia zvaná preskakujúca *ReLu (leaky ReLu)*. Záporná časť krivky je jemne naklonená, aby nastávali aspoň malé zmeny.

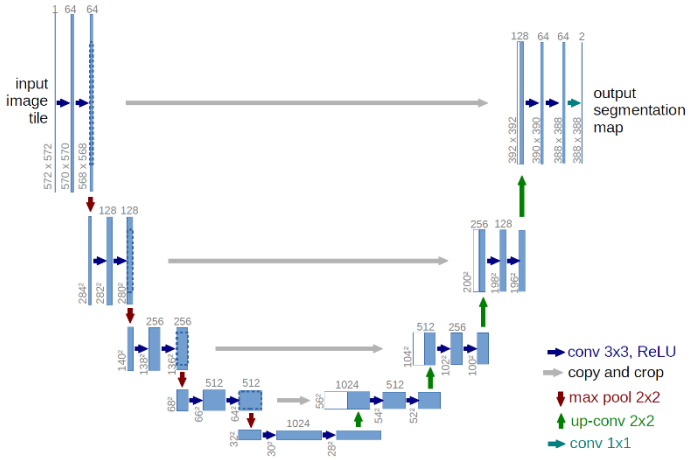
Overfitting

Sieť je trénovaná len na určitom počte vstupov (v našom prípade obrázkov), pretože vystaviť ju všetkým možným vstupom by bolo nielen výpočtovo náročné, ale aj nemožné. Keďže trénuje len na zlomku z možných dát, ktoré potenciálne môže dostať na vstupe, môže sa stať, že bude vedieť dobre reagovať na vstupy totožné alebo veľmi podobné tým, na ktorých bola natrénovaná, no vo všeobecnosti bude dosahovať nízku presnosť. Tomuto nežiadúcemu javu sa hovorí *overfitting*. Hovoríme, že sme sa dopustili *overfittingu*, ak je úspešnosť siete na testovacích dátach nižšia ako jej úspešnosť na tréningových dátach. Ako uvádza [9] o*verfitting* je bežný najmä u modelov s viacerými nelineárnymi vrstvami, pretože to umožňuje naučiť model

veľmi komplikovaným vzťahom medzi vstupmi a výstupmi. Na obmedzenie tohto javu sme použili metódu vynechania, ktorá je priblížená v časti Tvorba modelu.

1.3 U-sieť

U-sieť je vystavená na princípe plne prepojenej konvolučnej siete. Jej architektúra sa skladá z 2 častí, prvá funguje práve tak ako konvolučná sieť a druhá implementuje spätný chod, ktorého výsledkom je segmentovaný obrázok. Náčrt architektúry (viď. Obrázok 3. Architektúra U-siete (príklad pre 32x32 pixelov). Modré rámce znázorňujú viackanálovú mapu čŕt, nad nimi je uvedený počet kanálov. Rozmery *x*, *y* sú uvedené v ľavom dolnom rohu rámcov. Biele rámce reprezentujú skopírovanú mapu čŕt a šípky predstavujú jednotlivé operácie. Zdroj: [**1**]Obrázok 3) pripomína písmeno U, preto sieť dostala názov U-sieť.



Obrázok 3. Architektúra U-siete (príklad pre 32x32 pixelov). Modré rámce znázorňujú viackanálovú mapu čŕt, nad nimi je uvedený počet kanálov. Rozmery *x*, *y* sú uvedené v ľavom dolnom rohu rámcov. Biele rámce reprezentujú skopírovanú mapu čŕt a šípky predstavujú jednotlivé operácie. Zdroj: [1]

U-sieť v *prvej fáze* postupuje podobne ako konvolučná sieť. Striedajú sa vrstvy vykonávajúce aplikáciu *konvolúcie* a *zhromažďovania*. Zhromažďovaním sa redukuje veľkosť vstupu s ponechaním najdôležitejších znakov. Takýmto spôsobom je získaný jednoduchý výstup označujúci triedu, do ktorej podľa výpočtu obrázok patrí. Prvá fáza zodpovedá ľavej časti nákresu architektúry.

Cieľom *druhej fázy* je doplniť výstup o pixely zredukované v prvej časti. Postup je symetrický s prvou fázou s tým rozdielom, že v každom kroku je miesto konvolúcie aplikovaná *opačná konvolúcia* a zhromažďovanie je nahradené *nadvzorkovaním*. Týmito dvoma operáciami je získaný predbežný výstup, ktorý je ešte potrebné kombinovať s mapou čŕt z prvej fázy na tej istej úrovni (v i-tom kroku 2. fázy je výstup kombinovaný s mapou čŕt z *(n-i)*-teho kroku 1. fázy, kde *n* je počet krokov a $i = \{1,2,...n\}$). Vďaka tomu sú zachované dôležité črty pôvodného obrázka.

Tento proces nemusí byť aplikovaný na celý obrázok naraz. V praxi sa používa *stratégia prekrývajúcich sa dlaždíc* (z anglického *overlap-tile strategy*). Na výpočet pixelov vybranej časti obrázku sú použité aj pixely z jej okolia, aby bol zachovaný kontext. Ak je vybraná časť na okraji obrázku, okolie je vytvorené doplnením zrkadlového obrazu tejto časti. Vďaka tomu U-sieť dokáže rýchlo segmentovať aj veľké snímky.

Dáta pre U-sieť

Pre neurónové siete vo všeobecnosti platí, že je to silný nástroj na riešenie rôznych komplexných úloh, ktorý však za svoju presnosť vďačí obrovskému množstvu dát. Naproti tomu U-sieť si postačí s omnoho menšou tréningovou množinou. To je možné vďaka *rozšíreniu* každého tréningového vzoru aplikovaním elastických deformácií [8]. To znamená, že z jedného vzoru je vytvorených niekoľko nových vzorov, pričom sieti stačí pamätať si len originálne vzory a deformácie, ktorými vzniknú nové. Takýto spôsob rozšírenia dát je aplikovateľný predovšetkým pre biomedicínske snímky, ktoré sú si navzájom veľmi podobné a preto je možné aj generovaním vytvoriť vierohodnú snímku.

2. Návrh riešenia

Všeobecný postup pri tvorbe siete podľa [11] vyzerá nasledovne:

1. Príprava dát
2. Tvorba modelu
3. Trénovanie
4. Vyhodnotenie
5. Predikcia

2.1 Príprava dát

Mikroskopické snímky na trénovanie a testovanie modelu nám poskytli kolegovia z Neurobiologického ústav na SAV. Tieto snímky sú predspracované a rozdelili sme si ich do 3 skupín (viď. Tabuľka 1).

1. Tréningová množina vstupných obrázkov
2. Tréningová množina vzorových výstupných obrázkov
3. Testovacia množina vstupných obrázkov, pre ktoré budú sieťou získané výstupné obrázky

Vzorový vstup	Vzorový výstup	Vstup na testovanie

Tabuľka 1. Typy použitých obrázkov.

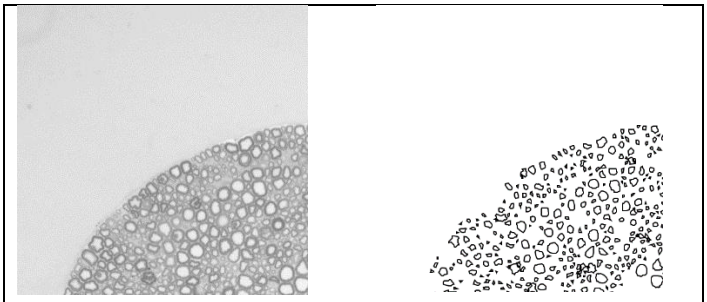
V snímkach bol odstránený šum, vstupy sme previedli na 8 bitové čiernobiele obrázky o rozmeroch 512 x 512 pixelov a výstupy na 32 bitové čiernobiele obrázky o rozmeroch 512 x 512 pixelov.

Rozšírenie dát

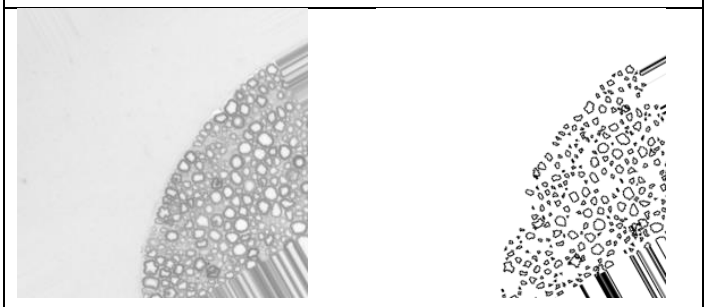
Skôr než sme dáta použili na trénovanie, aplikovali sme na nich rôzne deformácie a tým si rozšírili množinu dát. Výhodou je, že na disku reálne nepotrebujeme mať veľké množstvo obrázkov, ale rozšírením dát sme získali viac vzorov bez toho, aby sme si museli ukladať skonštruované obrázky, stačilo si zapamätať prevedené deformácie.

Použili sme na to `ImageDataGenerator`, triedu rozhrania Keras [3]. Na vstupe dostane slovník s vlastnosťami deformácie, napríklad, o aký uhol môže byť obraz otočený, o koľko pixelov posunutý alebo priblížený. Zvolené deformácie aplikuje použitím metódy `flow_from_directory(directory)`. Aby boli v sieti použité aj tieto rozšírené dáta, použijeme na trénovanie modelu miesto obvyklej metódy `fit()` metódu `fit_generator()`, ktorej predáme vytvorený generátor a ďalšie parametre k trénovaniu, ako je veľkosť dávky a počet krokov za epochu.

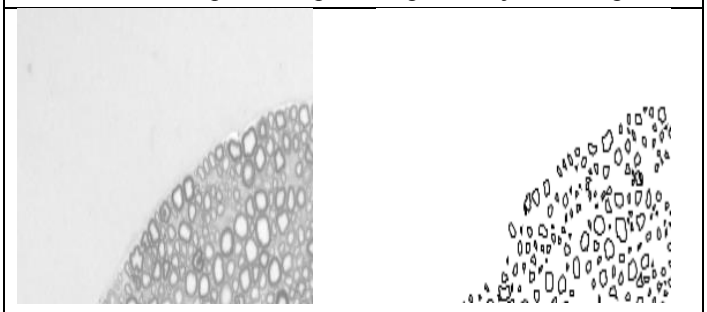
Dôvodom, prečo si vôbec môžeme dovoliť takto umelo vytvoriť ďalšie obrázky je, že všetky naše obrázky sú veľmi podobného charakteru – vždy sú na nich vyobrazené bunky pomerne okrúhleho tvaru. Ak takýto obrázok buniek otočíme o 30 stupňov a priblížime o 0.5 pixela, dostaneme obrázok dostatočne odlišný od pôvodného na to, aby sa ním vylepšil model našej siete. Na nasledujúcich obrázkoch v Tabuľka 2 si ukážeme, aké transformácie sme s obrázkami robili.



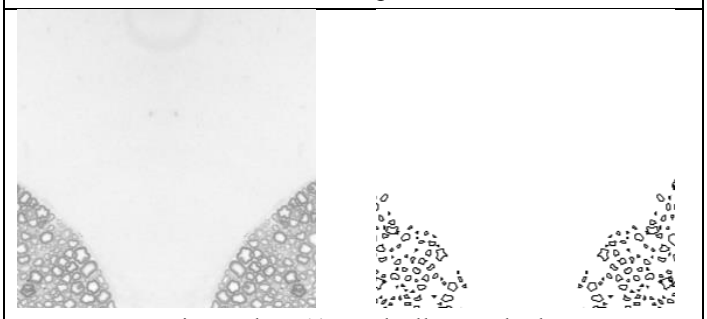
Pôvodný obrázok a jeho maska



Otočenie o 30 stupňov s doplnením podľa najbližšieho pixelu



Priblíženie o polovicu



Posunutie v šírke o ½ so zrkadlovým doplnením

Tabuľka 2. Rozšírenie dát aplikovaním jednotlivých deformácií.

2.2 Tvorba modelu

Model neurónovej siete sa skladá z niekoľkých vrstiev, pričom každá by mala byť zameraná na riešenie konkrétneho problému. Úlohou prvej vrstvy je vždy načítať vstup. Na to sa v Kerase používa vrstva `Input`, ktorej môžeme predať parameter veľkosti vstupu, ktorý má načítať.

```
inputs = Input(input_size)
```

Ďalej sme štruktúru modelu vytvorili podľa schémy u-siete na Obrázok 3. V 1. fáze bolo potrebné niekoľko krát vytvoriť trojicu vrstiev: dvakrát konvolučnú vrstvu a raz vrstvu zhromažďovania maxím. V ukážke kódu sú pre prehľadnosť vynechané vstupné parametre vrstiev.

```
conv1 = Conv2D(...)(inputs)
conv1 = Conv2D(...)(conv1)
pool1 = MaxPooling2D(...)(conv1)
```

Rozhodli sme sa túto trojicu vytvoriť 4-krát, a teda 1. fázu sme ukončili vytvorením vrstvy `conv5`.

Skôr ako sme prešli do 2. fázy tvorby u-siete, sme použili vrstvu `Dropout` (viď. [9]) ktorá náhodne nastaví zadanému počtu (je zadaný percentuálnou hodnotou) vstupných zložiek hodnotu na 0. To je užitočné preto, aby sme sa vyhli *overfittingu* spomínanému v Základné pojmyoch. Chceme ponechať len čiastočnú informáciu toho, čo sa sieť naučila, pretože celá informácia je príliš špecifická, kým my chceme naučiť sieť reagovať na ľubovoľný vstup.

```
conv5 = Conv2D(...)(pool4)
conv5 = Conv2D(...)(conv5)
drop5 = Dropout(0.5)(conv5)
```

V 2. fáze sme vykonali proces spätných konvolúcií a nad-vzorkovania opäť v skupinách po jednu vrstvu nad-vzorkovania a dve vrstvy spätnej konvolúcie. Ako je to znázornené v schéme, po aplikácii nad-vzorkovania sme ešte výslednú vrstvu skombinovali s mapou čŕt z 1. fázy na rovnakej úrovni. Až na túto vrstvu sme použili dvakrát spätnú konvolúciu.

```
up6 = Conv2D(...)(UpSampling2D(size = (2,2)(drop5)
merge6 = concatenate([drop4,up6], axis = 3)
conv6 = Conv2D(...)(merge6)
conv6 = Conv2D(...)(conv6)
```

Druhú fázu sme ukončili vytvorením vrstvy `conv10` a už ostáva len vrstvy pospájať do modelu (v Kerase ako trieda `Model`), čím je model hotový a pripravený na použitie.

```
model = Model(input = inputs, output = conv10)
```

2.3 Trénovanie

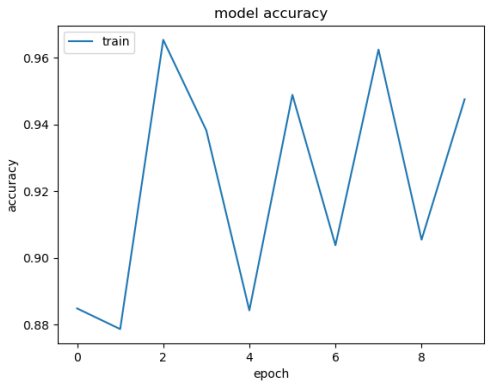
Pripravený model skompilujeme a môžeme trénovať. Pri kompilácii uvedieme, aký optimalizačný algoritmus a algoritmus výpočtu chyby bude použitý a tiež, na základe čoho budeme ohodnocovať model.

Podľa [10] je doteraz najlepším optimalizačným algoritmom Adamov optimizér, preto sme vybrali ten. Výber algoritmu výpočtu chyby ovplyvní to, podľa akých kritérií bude presnosť nášho modelu označená za vysokú či nízku. Vybrať si môžeme z ponuky Kerasu alebo implementovať vlastný. Pre náš typ úlohy je vhodný algoritmus `binary_crossentropy`, pretože každý pixel obrázku má byť vo výsledku čierny alebo biely, ide teda o binárne rozhodovanie. Ako metriku sme zvolili presnosť, vďaka tomu budeme môcť ohodnotiť náš model na základe toho ako presne vyhodnocuje.

```
model.compile(optimizer = Adam(lr = 1e-4), loss =
'binary_crossentropy', metrics = ['accuracy'])
```

Na spustenie trénovania sa v Kerase používajú metódy `fit()` a `fit_generator()`. Druhá uvedená sa používa v prípade, že používateľ chce sieť trénovať na dátach vytvorených generátorom, čo je aj náš prípad, pretože sme generátor použili na rozšírenie dát. Funkcia dostane na vstupe spomínaný generátor, počet epoch a krokov v rámci epochy a prípadne `ModelCheckpoint` užitočný na ukladanie medzivýsledkov. Metóda `fit_generator()` dáva na výstup inštanciu triedy `History`, v ktorej je uložený progres modelu, a vďaka tomu vieme vizualizovať zmeny v presnosti výpočtu vzhľadom na epochu, v ktorej sa výpočet nachádzal (Obrázok 4).

```
history = model.fit_generator(myGene,steps_per_epoch=6,epochs=1,
callbacks=[model_checkpoint])
```



Obrázok 4. Presnosť v jednotlivých epochách.

Keď model dokončí trénovanie, môžeme ho uložiť a znovu použiť, prípadne ho vylepšiť. Ukladáme do formátu .hdf5, ktorý dokáže uchovať informáciu o štruktúre modelu a hodnotách váh medzi vrstvami.

3. Zhrnutie

V tomto článku sme si priblížili dôležité pojmy týkajúce sa neurónových sietí ako aj konvolučnej u-siete, ktorú používame v našej práci. Okrem toho sme uviedli prvé tri body návrhu siete, do ktorých spadá príprava dát, návrh modelu a trénovanie. Ďalším bodom návrhu, experimentom a prezentácii výsledkov sa budeme venovať v našej bakalárskej práci.

4. Literatúra

1. Ronneberger, O. Fischer, P. Brox, T.: U-Net: Convolutional Networks for Biomedical Image Segmentation. Eprint arXiv:1505.04597. (2015) Dostupné na https://arxiv.org/abs/1505.04597

2. Michael A. Nielsen, "Neural networks and deep learning!", Determination Press, 2015

3. Keras: The Python Deep Learning Library, https://keras.io/

4. Machine Learning is Fun! Part 3: Deep Learning and Convolutional Neural Networks, https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721

5. Understanding Activation Functions in Neural Networks, https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

6. What is the Difference Between a Batch and an Epoch in a Neural Network?, https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

7. Building a Convolutional Neural Network (CNN) in Keras, https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5

8. Keras and Convolutional Neural Networks (CNNs), https://www.pyimagesearch.com/2018/04/16/keras-and-convolutional-neural-networks-cnns/

9. Srivastava, N. Hinton, G. Krizhevsky, A. Sutskever, I. Salakhutdinov, R.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15 (2014) 1929-1958. Dostupné na http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

10. How to pick the best learning rate for your machine learning project, https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2

11. Train your first neural network: basic classification, https://www.tensorflow.org/tutorials/keras/basic_classification