Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Semestrální práce

Automatická detekce chyb na základních deskách

Plzeň 2017 Michal Medek

Obsah

1 Úvod

Cílem práce je vytvořit program, který za pomoci konvoluční neuronové sítě, bude s co možná největší úspěšností rozhodovat zda jsou základní desky televizorů v pořádku nebo ne.

Neuronové sítě jsou v posledních letech více a více používány pro automatizování výroby. Jejich hlavní předností je schopnost učit se. Síť je schopná se v čase zdokonalovat a dosahovat tak vyšší úspěšnosti na základě předchozích špatných rozhodnutí.

Výsledný program by v případě vysoké úspěšnosti mohl být reálně nasazen a působit tak jako takzvaný druhý čtenář pro dodatečnou kontrolu desek. Může se totiž stát, že se pracovník kvality přehlédne a nebo si chyby nevšimne. Další možné využití programu je ještě před první kontrolou zúžit množinu potenciálně vadných desek a upozornit tak pouze na tyto desky. Dostatečně zdokonalený program by v budoucnu pracovníka kvality mohl nahradit úplně.

2 Analýza práce

V následujících odstavcích bude čtenář ve stručnosti seznámen se základní problematikou neuronových sítí.

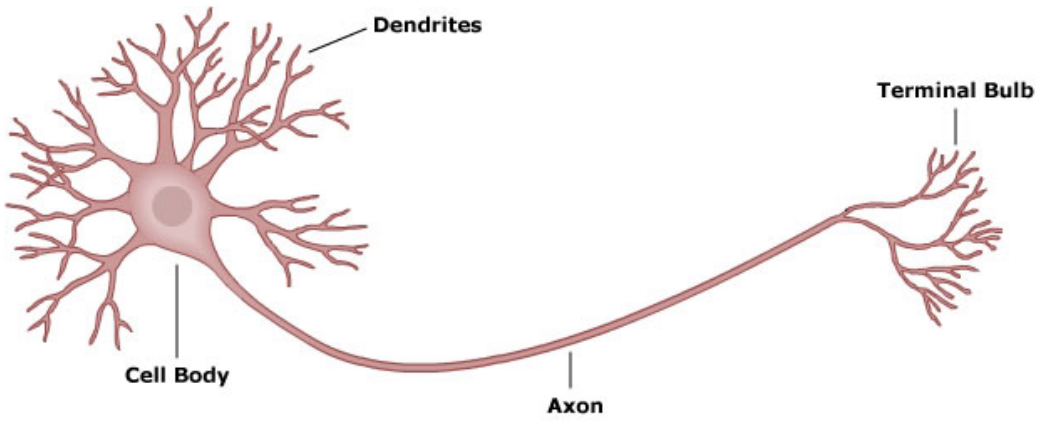
2.1 Neuronová síť

Teorie neuronových sítí vychází z chování biologických struktur. Základním prvkem neuronové sítě je neuron. Síť pak tvoří množina neuronů, které mají mezi sebou vazby.

2.1.1 Biologický neuron

Každá síť je tvořena jednotlivými neurony, které představují samostatné mezi sebou navzájem komunikující jednotky. Biologický neuron (obr. 2.1) je tvořen dendrity, které slouží pro napojení na ostatní neurony (neurony připojené na dendrity můžeme brát jako vstupy). Spojením neuronů říkáme synapse. Dendrity je koncentrováno okolo jádra neuronu z kterého vede axon (lze si jej přestavit jako výstup).

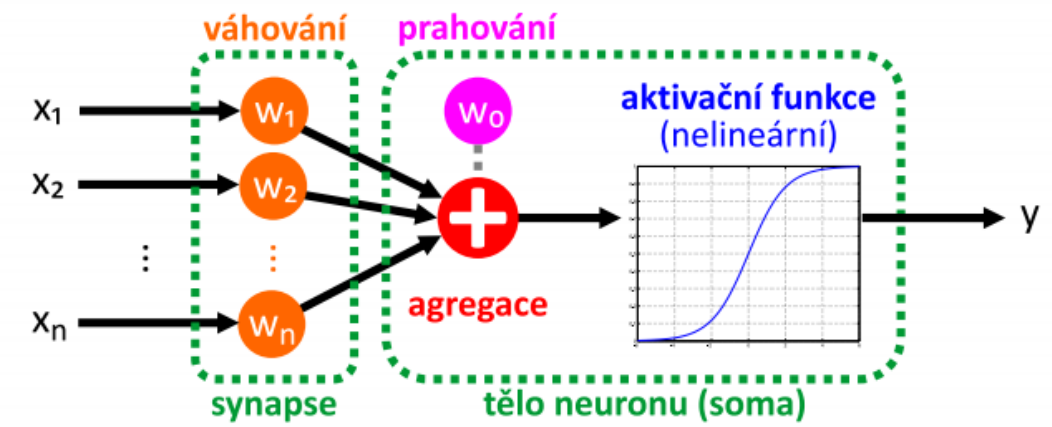
Obrázek 2.1: Biologický neuron [6]



2.1.2 McCulloch-Pittsův neuron

Umělý neuron poprvé matematicky popsali Warren McCulloch a Walter Pitts roku 1943. Jedná se o zjednodušený model biologického neuronu. McCulloch-Pittsův neuron (obr. 2.1) je složením vstupů, které jsou na začátku neuronové sítě a nebo aktivací některého z neuronů v předcházející vrstvě. Následuje váhování, kde je nejčastěji konkrétní vstup vynásoben určitou váhou (právě váhy poté představují adaptivní parametry). Všechny vstupy jsou po aplikaci váhové funkce přivedeny do agregační funkce. Pokud je práh větší než výstup funkce neděje se nic. Pokud je práh nižší je poté výstup agregační funkce přiveden do aktivační funkce.

Obrázek 2.2: McCulloch-Pittsův neuron [4]



2.1.3 Aktivační funkce

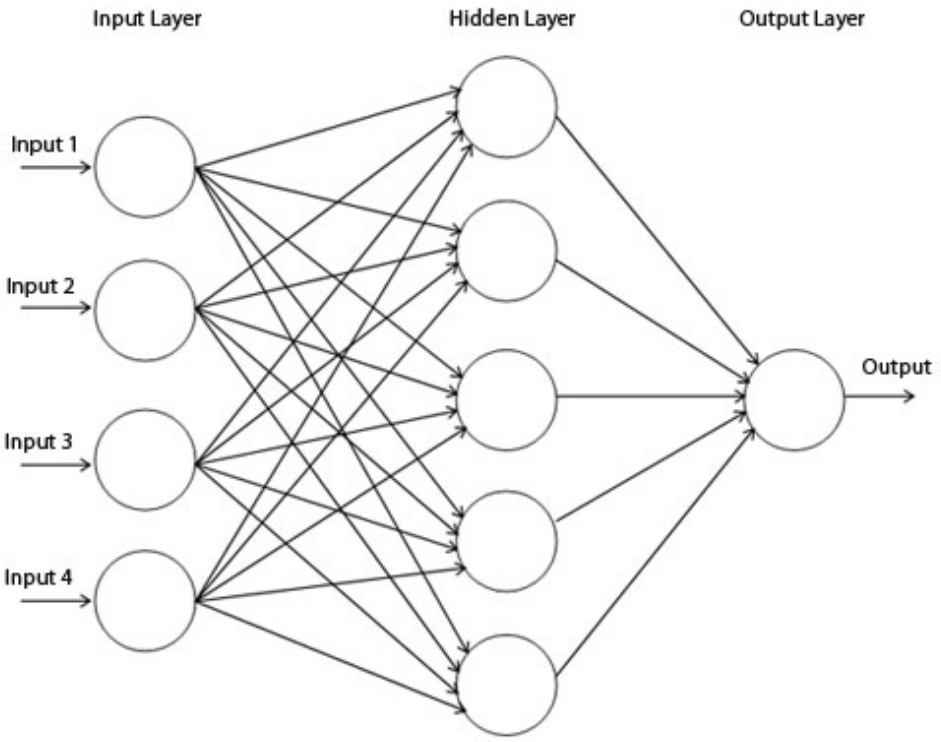
Aktivační funkce neuronu slouží pro finální nastavení aktivace. Funkce musí být derivovatelná, abychom byli schopni při procesu trénování neuronové sítě rozdistribuovat chybu po síti. Nejčastěji používáme následující funkce:

- `sigmoida` $f(n) = \dfrac{1}{1+e^{-x}}$,
- `gaussova funkce` $f(n) = e^{-x^2}$,
- `tangens hyperbolický` $f(n) = tangh$,
- `lineární aktivační funkce` $f(n) = x$.

2.1.4 Vícevrstvý perceptron

Vícevrstvý perceptron je jedna z nejjednodušších a nejpoužívanějších neuronových sítí. Síť se skládá z jedné vstupní vrstvy, která má počet neuronů roven počtu vstupů. Následuje libovolný počet tzy. skrytých vrstev o různém počtu neuronů a poslední vrstvou je vrstva výstupní, která slouží buď pro klasifikaci `n` tříd (počet neuronů výstupní vrstvy je roven `n`) a nebo pro binární úlohy, kdy je ve výstupní vrstvě neuron pouze jeden. Hodnota na vstupu neuronu je počítána jako suma propojených neuronů z předchozí vrstvy.

Obrázek 2.3: Vícevrstvý perceptron

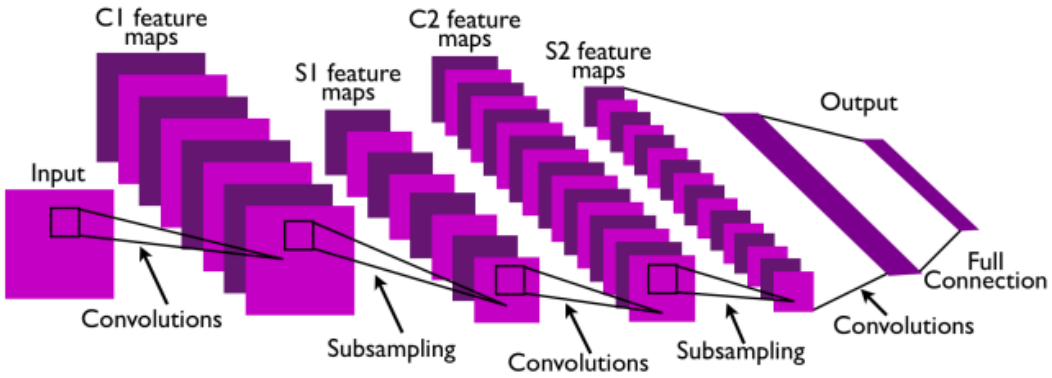


2.2 Konvoluční neuronová síť

Konvoluční neuronová síť je speciální typ vícevrstvé neuronové sítě, kdy na vstupu očekáváme obrázek, který je postupně předzpracován až v samotné síti. Tato varianta neuronových sítí se používá především pro identifikaci objektů.

První tzv. konvoluční vrstva, na kterou je přiveden obrázek, slouží pro extrakci příznaků pomocí konvolučního filtru, který nazýváme recepční pole. Výstupem konvolučního filtru jsou tzv. příznakové mapy, které jsou výsledkem konvoluce určité části obrázku s recepčním polem. Druhá vrstva pak většinou slouží ke snížení počtu příznaků pomocí vzorkování. Obvykle pro vzorkování používáme metody jako max-pooling (hledání maxima) a nebo průměrování. V následujících vrstvách se pak obvykle tyto zmíněné vrstvy opakují. Z toho důvodu abychom byli schopni zachytit i složitější obrazce než jen základní hrany. Po této extrakci příznaků následuje neuronová síť tak jak ji známe z předchozí kapitoly.

Obrázek 2.4: Konvoluční neuronová síť [5]



2.2.1 Proces učení

Z předchozího popisu konvoluční neuronové sítě můžeme předpokládat, že za každou konvoluční vrstvou l následuje vrstva se vzorkováním $l + 1$. Proto musíme každou příznakovou mapu, před počítáním chyby zvětšit do velikosti použitého konvolučního filtru. Zvětšení probíhá tak, že pixel, který reprezentuje výsledek filtru `20x20` jednou hodnotou bude nyní představovat matici `20x20` stejných hodnot původního výsledku (funkce g rovnice (2.1)). Následně vynásobíme zvětšenou příznakovou mapu derivací aktivační funkce konvoluční vrstvy:

$$\theta_j^l = \beta_j^{l+1}(\partial f(u_j^l) \cdot g(\theta_j^{l+1})) \tag{2.1}$$

kde β_j^{l+1} představuje váhy z navzorkované vrstvy $l+1$ a funkce $f(u_j^l)$ derivaci aktivační funkce neuronu u_j^l. Tuto akci opakujeme pro každou příznakovou mapu. Poté můžeme vypočítat gradient prahu jako součet jednotlivých

chyb θ_j^l:

$$\frac{\partial E}{\partial b_j} = \sum_{u,v} (\theta_j^l)_{u,v} \tag{2.2}$$

kde u, v jsou souřadnice právě počítaného elementu.
Gradient vah budeme počítat velmi podobně pomocí vztahu:

$$\frac{\partial E}{\partial k_{i,j}^l} = \sum_{u,v} (\theta_j^l)_{u,v} (p_i^{l-1})_{l-1} \tag{2.3}$$

kde (p_i^{l-1}) představuje bod v příznakové mapě.

3 Popis implementace

Celá práce se skládá ze tří nezávislých programů:

- `ROI Selector` – program pro zajištění extrakce trénovacích a testovacích dat z obdržených obrázků,
- `CNNs` – jednotlivé skripty, které slouží pro natrénování a uložení konvolučních neuronových sítí, pro jednu chybu existuje jeden skript,
- `Recognizer` – program, který využívá natrénované sítě a pomocí nich rozpoznává chyby na základních deskách.

3.1 ROI Selector

Program je napsán v jazyce C++ za pomoci knihovny OPENCV. Slouží ke generování ROI snímků na základě svg snímků, které jsou nejprve naparsovány a následně je díky informaci o poloze chyb vybrán jeden vzor pro každou chybu. Tento vzor slouží k získání všech ostatní pozitivních a negativních snímků. Tento přístup byl použit, aby všechny ROI byly v ideálním případě stejně vycentrované.

V pozdější fázi projektu se ukázalo, že máme velmi málo pozitivních snímků a byla proto implementována základní metoda rozšíření datového korpusu pomocí geometrických transformací.

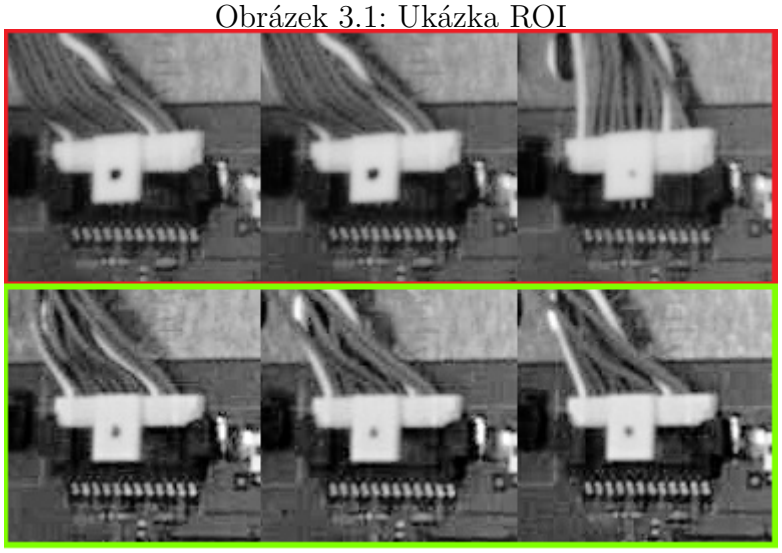
3.1.1 Popis tříd

Program tvoří několik tříd, které si nyní v krátkosti přiblížíme:

- `main.cpp` – hlavní třída, která obsahuje metodu `preparePostiveROIs` a `prepareNegativeROIs`, kde se volají metody nad instancí objektu třídy `ROISelector`.
- `ROISelector.cpp` – nejdůležitější třída, která má na starosti řízení toku programu. Využívá se zde volání instance třídy `XMLParser.cpp` pro práci s svg snímky a následné vybrání ROI je realizováno pomocí knihovny `OPENCV`.
- `XMLParser.cpp` – třída pro rekurzivní naparsování `SVG` obrázku.

- `ImageRecord.cpp a Line.cpp` – slouží pouze jako přepravky, image record pro uchování informací o snímku a line pro informace o objektu čáry v `SVG` snímku

Obrázek 3.1: Ukázka ROI



3.2 CNNs

Jádro práce tvoří konvoluční neuronové sítě, které jsou implementovány pomocí knihovny Keras a Theano v programovacím jazyce Python 2.7. Vzhledem k tomu, že má výběr sítě zásadní vliv na funkčnost celého systému bude mu zde věnována největší pozornost.

3.2.1 Načtení trénovacích a testovacích dat

První část programu tvoří načítání obrázků do velké matice z níž každý řádek představuje vektor reprezentující jeden ROI. Za sebou tak jsou v matici seřazené nejdříve pozitivní ROI a pak negativní. Na závěr se k matici přidá poslední sloupec, který značí do jaké skupiny trénovacích dat obrázek spadá (1 - pokud se jedná o chybu, 0 - pokud je správně). Poté se data rozdělí na trénovací a testovací množinu podle našich požadavků, prvky matice se převedou na datový typ float a matice se normalizuje vydělením 255.

3.2.2 Realizace neuronové sítě

Druhá část obsahuje vytvoření neuronové sítě, která je tvořená konvoluční vrstvou následovanou `ReLU` aktivační funkcí, poté se opakuje stejná kombinace konvoluční vrstvy a `ReLU` aktivace. Další vrstva je `max-pooling` následovaný `dropout` vrstvou, která zapříčiní vynulování všech neuronů s aktivační funkcí jejímž výsledkem je hodnota 0,5. Takto je sestavená část pro extrakci příznaku. Klasifikaci provádí plně propojená neuronová síť s `ReLU` aktivační funkcí. Poslední vrstvu tvoří dva neurony reprezentující klasifikované třídy a jako aktivační funkce je použit `softmax`, který se pro výstupní vrstvu obvykle používá.

```
model = Sequential()

model.add(Convolution2D(nb_filters, nb_conv, nb_conv,
                        border_mode='valid',
                        input_shape=(1, img_rows, img_cols)))
convout1 = Activation('relu')
model.add(convout1)
model.add(Convolution2D(nb_filters, nb_conv, nb_conv))
convout2 = Activation('relu')
model.add(convout2)
model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
model.compile(loss='binary_crossentropy', optimizer='adadelta',
    metrics=["accuracy"])
```

3.2.3 Zjištění úspěšnosti na trénovací množině

Poslední část kódu pouze spustí test nad testovací množinou a vrátí celkovou úspěšnost. Následuje uložení sítě a vah.

```
score = model.evaluate(X_test, Y_test, verbose=1)
print('Test lost:', score[0])
print('Test accuracy:', score[1])
```

```
# serialize model to JSON
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model.h5")
print("Saved model to disk")
```

3.3 Recognizer

Recognizer je program pro rozpoznávání chyb na základních deskách. Na vstupu programu je obrázek ve formátu `.jpeg` tento obrázek se nejprve zařadí na základě jeho jména – které je standardizováno a můžeme tak desku zařadit dle typu. Díky tomu jsme schopni otestovat části desky pro, které byly natrénovány konvoluční sítě, protože se tam často nachází chyby. Výstupem programu je pak obrázek s potenciálně označenými chybami.

3.3.1 Popis jednotlivých skriptů

Celý program je tvořen několika skripty (jedná se o třídy):

- `run.py` – spustitelný skript, který řídí tok programu,
- `record.py` – třída sloužící jako přepravka pro záznam obrázku (obsahuje umístění obrázku, jméno komponenty, desky, apod.),
- `data_loader.py` – třída pro načítání a parsování obrázků a CNNs, které se ukládají jako instance třídy `evaluation_data.py`,
- `cnn_runner.py` – hlavní třída pro načtení neuronových sítí a zahájení evaluace, při nalezení chyby je chyba uložena jako instance třídy ze skriptu `error.py` .

4 Dosažené výsledky

Následující sekce obsahuje přehled získaných výsledků a provedených experimentů při práci s konvolučními sítěmi (CNNs) pro jednu konkrétní chybu, stejný experiment byl proveden pro každou z chyb. Bylo použito 32 konvolučních filtrů, pooling-area rovna dvěma a konvoluční jádro mělo velikost 3x3. Experimenty byly provedeny na zařízení s konfigurací Windows 10, 8GB RAM a procesorem Intel Core i7-270QM s frekvencí 2.2Ghz.

4.1 Experiment 1

Při prvním experimentu byla použita původní pozitivní množina dat čítající 23 obrázků, která byla doplněna o 23 snímků negativních, abychom používali vyváženou množinu dat. Z toho bylo 40 snímků použito pro trénování a 6 pro testování.

Na základě tabulky 4.1, kde bylo použito pro trénování 15 epoch, že bychom počet epoch ještě mohli zvýšit vzhledem k tomu, že úspěšnost na trénovací množině má celkem velké výkyvy. To je spojené s malým počtem trénovacích vzorků. Přesto jsme však našli model, který má úspěšnost i 95

Tabulka 4.1: Experiment 1

Epocha	Doba běhu [s]	Úspěšnost [%]
1.	6	45
2.	5	42.5
3.	6	60
4.	6	72.5
5.	6	87.5
6.	6	77.5
7.	6	67.5
8.	6	80
9.	6	85
10.	6	72.5
11.	6	87.5
12.	6	85
13.	6	95
14.	6	77.5
15.	6	95

Úspěšnost na testovací množině je 100%, ale bylo použito jen 6 testovacích snímků.

Na základě toho experimentu jsme se rozhodli použít geometrické transformace pro zvětšení množiny pozitivních snímků a následné další experimenty.

4.2 Experiment 2

Pří druhém experimentu jsme rozšířili pomocí transformací obrázku datovou množinu negativních dat a mohli jsme tak provést pokus s množinou čítající 1000 testovacích a 150 trénovacích vzorků v poměru 1:1.

V tabulce 4.2 znázorňující trénování sítě můžeme vidět, že s rostoucím počtem epoch roste i úspěšnost klasifikace. Jedna epocha trvala vždy do tří minut, takže trénování této konvoluční neuronové sítě při deseti epochách trvalo přibližně třicet minut. Dále je vidět, že jsme dosáhli 100% úspěšnosti na trénovací množině.

Tabulka 4.2: Experiment 2

Epocha	Doba běhu [s]	Úspěšnost [%]
1.	158	89.5
2.	151	98.4
3.	152	99.6
4.	149	99.9
5.	147	99.9
6.	148	100
7.	150	100
8.	152	99.9
9.	154	100
10.	164	99.9

Dosažená úspěšnost na testovací množině byla také 100% a klasifikace 150 vzorků trvala tři sekundy. Na základě těchto výsledků můžeme říct, že se nám podařilo získat kvalitní model, který umí vhodně generalizovat.

5 Závěr

V oborovém projektu jsem se seznámil s konvolučními neuronovými sítěmi a jazykem Python. Podařilo se mi dosáhnout velmi dobré úspěšnosti při klasifikaci úlohy z reálného prostředí. Vznikl velmi dobře rozšiřitelný systém, který poskytuje velmi dobré výsledky, proto svou práci považuji za splněnou.

Do budoucna je vhodné refactorovat skripty pro vytváření konvolučních sítí, protože se jednalo o mou první zkušenost jak s knihovnou Keras tak s jazykem Python kód není tak přehledný jak by mohl být.

Vzhledem k tomu, že konvoluční sítě jsou obtížnou problematikou a neměl jsem dostačující matematický aparát na jejich rychlé pochopení tak mi projekt zabral podobné množství času jako bakalářská práce.

Literatura

[1] Mária Krajčovičová *Konvoluční neuronová síť pro zpracování obrazu*, Diplomová práce VUTBR, 2016.

[2] Adit Deshpande *A Beginner's Guide To Understanding Convolutional Neural Networks*, 2016, https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

[3] Andrej Karpathy *Convolutional Neural Networks (CNNs / ConvNets)*, http://cs231n.github.io/convolutional-networks/ .

[4] Kamil Ekštein *Umělé neuronové sítě*, 2016

[5] Nicholas Leonard *Torch 7: Applied Deep Learning for Vision and Natural Language*, 2015

[6] The Pennsylvania State University *Neurons*, 2012

[7] Nando de Freitas *Deep Learning Lecture 10: Convolutional Neural Networks*, University of Oxford, 2015