Česká Zemědělská Univerzita v Praze

Provozně Ekonomická Fakulta

Katedra Informačního inženýrství

Bakalářská práce

# Computer Vision analýza a použití v průmyslu

Vlasov Illia

© 2021 ČZU v Praze

**ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE**

Provozně ekonomická fakulta

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Illia Vlasov

Systémové inženýrství a informatika
Informatika

Název práce

**Computer Vision – analýza a použití v průmyslu**

Název anglicky

**Computer Vision – Analysis and application in industry**

---

**Cíle práce**

Navrhněte softwarové řešení v programovacím jazyce Python, které rozpozná tři typu objektů, banán, pomeranč, rajče. Pro software je nutno vytvořit klasifikátor pomocí algoritmu YOLOv3 a použitím Darknet Frameworku. Software otestujte a definujte závěry.

**Metodika**

Metodika BP je založena na základě studia odborné literatury, odborných článků a existujících softwarových řešení.

- Instalujte a nastavte Darknet Framework, CUDA, OpenCV.

- Vytvořte sadu dat pro učení klasifikátoru.

- Pomocí architekutury hluboké neuronové sítě YOLOv3 vytvořte klasifikátor a s jeho pomocí Object Detector v programovacím jazyce Python.

- Detektor otestujte.

**Doporučený rozsah práce**

30-40

**Klíčová slova**

Computer Vision, Image Processing, Strojové vidění, Zpracování obrazu, Deep Learning, Python

---

**Doporučené zdroje informací**

Pavlíček, J. ; Jarolímek, J. ; Pavlíčková, P. ; Dvořák, S. ; Pavlík, J. ; Hanzlík, P.: Automated Wildlife Recognition

---

**Předběžný termín obhajoby**

2020/21 ZS – PEF (únor 2021)

**Vedoucí práce**

Ing. Josef Pavlíček, Ph.D.

**Garantující pracoviště**

Katedra informačního inženýrství

Elektronicky schváleno dne 11. 3. 2020

**Ing. Martin Pelikán, Ph.D.**

Vedoucí katedry

Elektronicky schváleno dne 11. 3. 2020

**Ing. Martin Pelikán, Ph.D.**

Děkan

V Praze dne 13. 03. 2021

**Čestné prohlášení**

Prohlašuji, že svou bakalářskou práci "Computer Vision analýza a použití v průmyslu" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne ____________________

Rád bych touto cestou upřímně poděkoval pánu Ing. Josefu Pavlíčkovi, Ph.D, za cenné připomínky, rady a čas, který mi věnoval při zpracování této bakalářské práce.

# Computer Vision analýza a použití v průmyslu

**Abstakt**

Cílem této práce je navrhnout softwarové řešení v programovacím jazyce Python, které rozpozná tři typu objektů, banán, pomeranč, rajče. Za tímto účelem tu budou uvedený všechny potřebné teoretické východiska, následně budou popsány již existující metody a přistupy. V praktické části bude vytvořená sada dat pro učení klasifikátoru a pomocí architektury hluboké neuronovésítě YOLOv3 bude vytvořen klasifikátor a s jeho pomocí Object Detector v programovacím jazyce Python.

**Klíčová slova:** Strojové vidění, Zpracování obrazu

# Computer Vision analysis and application in industry

**Abstract (en)**

The aim of this work is to design a software solution in the Python programming language that recognizes three types of objects, banana, orange, tomato. For this purpose, all the necessary theoretical background will be presented, followed by a description of existing methods and approaches. In the practical part, a set of data for learning the classifier will be created and using the YOLOv3 deep neural network architecture, a classifier will be created and with its help the Object Detector in the Python programming language.

**Keywords:** Computer Vision, Image Processing, Deep Learning, Python

# Obsah

**1 Úvod** **7**

**2 Cíl práce a metodika** **8**

2.1 Cíl práce . . . 8

2.2 Metodika . . . 8

**3 Teoretická Východiska** **9**

3.1 Zpracovaní Obrazu . . . 9

3.1.1 Typy Obrázků . . . 9

3.1.2 Geometrické Transformace obrázků . . . 10

3.1.3 Image Thresholding . . . 12

3.1.4 Image Smoothing . . . 13

3.1.5 Shrnutí Kapitoly . . . 14

3.2 Strojové Učení . . . 14

3.2.1 Definice Strojového Učení . . . 14

3.2.1.1 Úkol . . . 15

3.2.1.2 Interpretace . . . 15

3.2.1.3 Zkušenost . . . 16

3.2.2 Učení z učitelem . . . 16

3.2.2.1 Lineární regrese . . . 17

3.2.3 Učení bez učitelu . . . 19

3.2.4 Zpětnovazební učení . . . 19

3.2.5 Shrnutí Kapitoly . . . 20

3.3 Hluboké Učení . . . 20

3.3.1 Neuronovou Síť . . . 20

3.3.1.1 Sigmoidní funkce . . . 21

3.3.1.2 Architektura neuronové sítě . . . 22

3.3.2 Vícevrstvá Neuronová Síť . . . 24

3.3.3 Convolutional Neural Networks . . . 26

3.3.3.1 Konvoluční vrstva . . . 28

3.3.3.2 Pooling vrstva . . . 30

3.4 Object Detection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
3.4.1 Intersection over Union . . . . . . . . . . . . . . . . . . . . . . . . 32
3.4.2 YOLOv3 a Darknet Framework . . . . . . . . . . . . . . . . . . . . . 34
3.5 Závěr Teoretické části . . . . . . . . . . . . . . . . . . . . . . . . . . . 39

**4 Praktická čast 40**

4.1 Sběr Dat . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
4.2 Předzpracování Dat . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
4.3 Učení Modelu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
4.4 Testování Modelu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
4.5 Implementace . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
4.6 Výsledek . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47

**5 Závěr 48**

**Seznam použitých zdrojů 49**

# Seznam obrázků


1 Obrázek $5 \times 5$ pixelu . . . 9
2 Původní Obrázek $5 \times 5$ pixelů . . . 10
3 Původní Obrázek . . . 10
4 Přesunutý Obrázek . . . 10
5 Původní Obrázek . . . 11
6 Otočený Obrázek . . . 11
7 Původní Obrázek . . . 11
8 Afinní Transformace Obrázku . . . 11
9 Původní Obrázek . . . 11
10 Perspektivní Transformace Obrázku . . . 11
11 Špatně viditelný obrázek . . . 12
12 Hodnotu práhu o velikosti 110 . . . 12
13 Hodnotu práhu o velikosti 100 . . . 13
14 Původní Obrázek . . . 14
15 Obrázek po použití LPF . . . 14
16 Přizpůsobit data lineárním modelem, kde $y^{(i)}$ je jediné pozorování a $\hat{y}^{(i)}$ je jediný odhad ze datové sady **D**. [12] . . . 18
17 Jedná Vrstvá [12] . . . 20
18 Umělý Neuron, jako Biologický Neuron [11] . . . 21
19 Sigmoida graph. [12] . . . 22
20 Architektura neuronové sítě s více výstupy . . . 24
21 Dvouvrstvá Neuronová Síť . . . 25
22 prostorové schéma CNN `https://cs231n.github.io/assets/cnn/cnn.jpeg` . . . 28
23 Konvoluce, `https://cs231n.github.io/assets/cnn/depthcol.jpeg` 29
24 Pooling `https://cs231n.github.io/assets/cnn/pool.jpeg` . . . 31
25 Maxpooling `https://cs231n.github.io/assets/cnn/maxpool.jpeg` . 32
26 IoU [2] . . . 33
27 IoU příklad [2] . . . 33
28 Přístup YOLO [10] . . . 34

29 YOLO model [10] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
30 YOLO Bounding Boxes [6] . . . . . . . . . . . . . . . . . . . . . . . . . 38
31 YOLOv3 arch [8] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
32 Struktura Praktické Části . . . . . . . . . . . . . . . . . . . . . . . . . 40
33 Datová Sada Obrázků . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
34 Rozhraní programu LabelImg . . . . . . . . . . . . . . . . . . . . . . . . 41
35 Vytvoření labelů objektu v LabelImg . . . . . . . . . . . . . . . . . . . . 41
36 txt soubor pro obrázek . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
37 Testovací obrázek . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
38 Předpověd' . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
39 Náhodný obrázek z internetu https://upload.wikimedia.org/wikipedia/commons/thumb/8/89/Tomato_je.jpg/1280px-Tomato_je.jpg . . . . 43
40 Předpověd' . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43

# 1 Úvod

Bakalářská práca se skládá ze dvou části, teoretická a praktická, v teoretické části chtěl bych popsát základy Počítačového Vidění a Zpracování Obrazu, a popsat jejich základní metody, které později použiji v praktické části. Dále bych popsal základní metody Strojového učeni a to prodloužit Hlubokým učením, charakterizovat co to je Neuronová sít' a Konvoluční Neuronová sít'. A na konci popišu YOLOv3 algoritmus na kterém je navázaná praktická část bakalářské práce.

# 2 Cíl práce a metodika

## 2.1 Cíl práce

Cílem této práce je navrhnout softwarové řešení v programovacím jazyce Python, které rozpozná tři typu objektů, banán, rajče a pomeranč.Pro software je nutno vytvořit klasifikátor pomoci algoritmu YOLOv3 a použitím Darknet Frameworku. Software otestuji a deifinuji závěry.

## 2.2 Metodika

Metodika bakalářské práce je založena na základě stidua odborné literatury, odborných článků a existujících softwarových řešení.

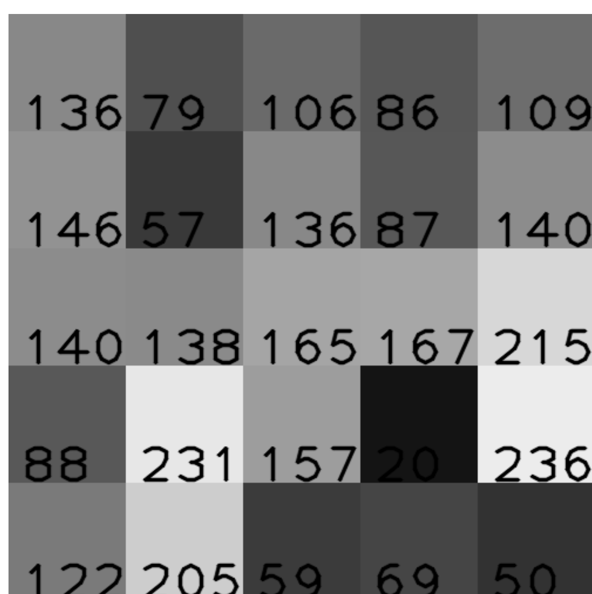
- Instaluji a nastavím Darknet Framework, CUDA, OpenCV
- Vytvořím sadu dat pro učení klasifikátoru
- Pomocí algoritmu YOLOv3 vytvořím klasifikátor a s jeho pomocí Object Detector v programovacím jazyce Python.
- Object Detector otestuji

# 3 Teoretická Východiska

## 3.1 Zpracovaní Obrazu

Zpracovaní obrazu se zabývá vývojem digitálních nástrojů, které provádí operace s digitálním obrazem. Takže před tím než popisovat základní metody Zpracovaní obrázu, je nutno uvést co to je Digitální obraz. [16]

Digitální obraz - je obraz který je složen z pixelu, které představuji bod obrázku z určitou intenzitou světla v každém dobu, například:




Obrázek 1: Obrázek 5 × 5 pixelu

Je to obrázek 5x5 pixelu, každé číslo v rozmezí 0-255 v čtverců udává intenzitu světla, čím je to číslo menší tím méně intenzita, například jestliže nastavíme nějakou proměnnou na 0, tak dostaneme úplně černý pixel, a naopak. [16]

### 3.1.1 Typy Obrázků

Je nutno ještě popsat nejvyužívanější typy obrázku.

- **Binary Image** má jenom dvě hodnoty pixelu, 1 nebo 0, používá se například ve funkce Thresholding.

- **8-bit Image Format** například Obrazek.1 je typicky 8-bit obrázek, protože vyjadřuje intenzitu šedé barvy v rozmezí mezi 0 a 255.

- **16-bit Image Format** nebo High Color Format, tady se už začíná varevní formát obrázku, tento formát má 65536 různých reprezentace barev, ale tato reprezentace není stejná jako u šedých obrázku, obraz se děli na tři různých kanálu, Red, Green a Blue neboli RGB, 5 bitů pro každý z kanálů a 1 bit pro Alpha kanál nebo 5 bitů pro Red a Blue kanály a 6 bitů pro Green.

- **24-bit Image Format** nebo True Color Format, distribuce bitu podle kanálů stejná jako u 16-bit obrázku, ale tento formát má 16.7 milionů barev. [16]

#### 3.1.2 Geometrické Transformace obrázků

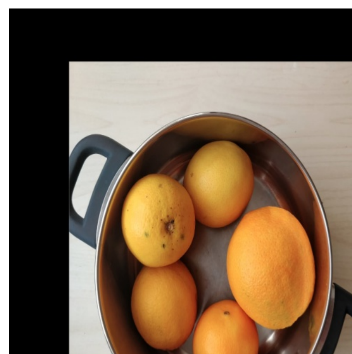
Cílem této sekce je vysvětlit takové pojmy jako posunuti, rotace, afinní transformace, image scaling, perspektivní transformace.

**Image scaling**, je pouze změna velikosti obrazu, například Obr.1 je zvětšený obrázek, jisty roměr tohoto obrázku vypadá tak [16]:

Obrázek 2: Původní Obrázek $5 \times 5$ pixelů

**Translation**, je posunuti polohy objektu. Pokud známe hodnoty posunu $(x, y)$, tak dostaneme $(t_x, t_y)$ a dosadíme této hodnoty do Matice $M$, a tím dostaneme **Matice Posunutí**, například[16]:

Obrázek 3: Původní Obrázek



Obrázek 4: Přesunutý Obrázek

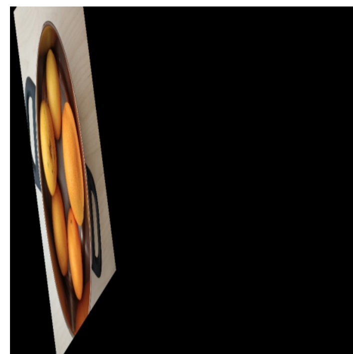
**Rotation**, je tu skoro stejný způsob jako u posunuti, ale místo hodnot posunu, je nutno zadat hodnotu uhlu a pak použít **Matice Rotace**, například[16]:

Obrázek 5: Původní Obrázek

Obrázek 6: Otočený Obrázek

**Affine Transformation**, používá tak zvanou Matice Transformace, které má dimenze $2 \times 3$, k nalezení matice potřebujeme tři body ze vstupního obrazu a jejich odpovídající umístěni ve vstupním obrazu, například[16]:

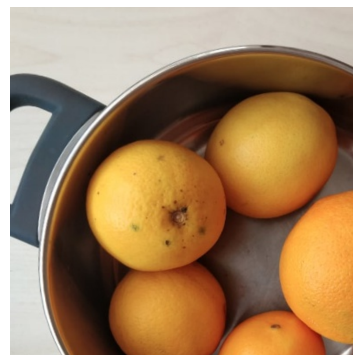
Obrázek 7: Původní Obrázek



Obrázek 8: Afinní Transformace Obrázku

**Perspective Transformation**, na rozdíl of **Affine Transformation**, tady je nutno použít $3 \times 3$ Matice transformace, k nalezení této matice potřebujeme 4 body na vstupním obrazu a odpovídajíci body na výstupním obrazu. Z těchto 4 bodů by 3 z nich **ne**měly by být kolineární, například[16]:

Obrázek 9: Původní Obrázek



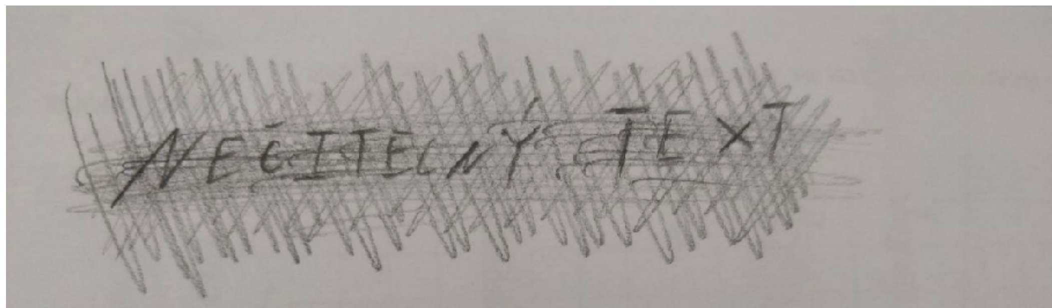
Obrázek 10: Perspektivní Transformace Obrázku

### 3.1.3 Image Thresholding

**Image Thresholding**, neboli Prahování, je metoda segmentace obrazu, která je založená na hodnocení barvy a jasu každého pixelu na obrázku. Principem je nalezení takové hodnoty, neboli **prahu**, v histogramu, pro kterou bude platit, že všechny hodnoty které jsou nižší než stanovený práh odpoviídají pozadí, zatímco hodnoty které jsou vyšší než stanovený práh odpovídají popředí. [16]

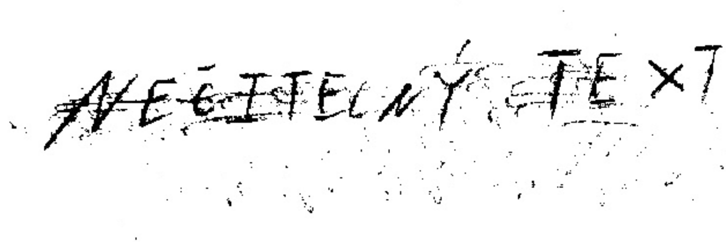
Nejjednodušší metoda v Prahování, je Binární prahování, stanovujeme pro každý pixel práh, a jestliže intenzita ve stanoveném pixelu je menší než tento práh, tak hodnota pixelu se bude rovnat 0, neboli černé barvě, a naopak, jestliže hodnota je vyšší než práh, tak hodnota pixelu se bude rovnat 255. [16]

Například, tady je obrázek z nějakým textem, který je těžko čitelný, protože obrázek má hodně šumu na pozadí, použiji Binární prahování, abych mohl ten text přečíst. [16]



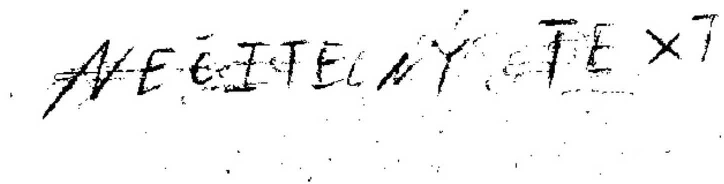
Obrázek 11: Špatně viditelný obrázek

Nastavím hodnotu práhu na 110, je to lepší ale na obrázku stále nějaký šum.




Obrázek 12: Hodnotu práhu o velikosti 110

Nastavím hodnotu práhu na 100, je to trochu lepší protože na obrázku se stalo méně šumu, zkisím to vylepšit s pomocí metod **Image Smoothing**. [16]



Obrázek 13: Hodnotu práhu o velikosti 100

### 3.1.4 Image Smoothing

Před tím než popisovat jak funguje Image Smoothing, je nutno uvést základní pojem, který zde bude použit, a je to **Convolution Matrix**, neboli **Kernel**, je to malá Matice, která se používá pro **Image Blurring, Image Sharpening, Edge Detection, atd.** A toho lze dosáhnout pomocí formy **matematicé konvoluce** mezi Kernelem a obrázkem. **Convolution**, neboli Konvoluce, je proces přidání každého **prvku obrázku** k **jeho místním sousedům**, váženého Kernelem. Pozor, konvoluce není tradiční násobení matice. [16] [19]

Matematická konvoluce měří míru překrytí mezi dvěma funkcemi. Lze jej považovat za směšovací operace, která intergruje bodové násobení jedné datové sady s druhou.

$$r(i) = (s * k)(i) = \int s(i-n)k(n)dn$$

Diskretně to lze zapsat jako:

$$r(i) = (s * k)(i) = \sum_n s(i-n)k(n)$$

V kontextu zpracování obrazu je Convolution Filter, neboli Konvoluční Filtr pouze skalárním součinem hmotnosti filtru se vstupními pixely v okně obklopujícím každý z výstupních pixelů. [19]

Rozlišujeme dvě skupiny filtru

- **Low-Pass Filters** zkracené **LPF**, se používá pro odstraněni šumu z obrázku, ve skutečnosti odstaní vysokofrekvenční obsah, např. šum, hrany, což má za následek vyhlazení hran při použití tohoto filtru.

$5 \times 5$ Kernel pro Image Averaging, nebolu Průmětovaní Obrazku

$$K = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- **High-Pass Filters** zkraceně **HPF** se používá pro například detekce hran. [16]

Příklad použití LPF z Kernelem $20 \times 20$ [16]

Obrázek 14: Původní Obrázek

Obrázek 15: Obrázek po použití LPF

### 3.1.5 Shrnutí Kapitoly

V této sekce byly popsány základní metody zpracování obrazu. Zpracování obrazu se používá v mnoha oblastech, nachází se uplatnění v oblastech, jako je strojové vidění, vytváření filtrů, používá se k vytváření nástrojů pro rozpoznávání textu, tváří, objektů atd.

## 3.2 Strojové Učení

### 3.2.1 Definice Strojového Učení

Především něž definovat co je Strojové učení, je nutno uvest co je algorithmus strojového učení, toto je takový algoritmus, který muže učit z dat, ale co myslíme pod pojmem *učení*, uvedu známou definici, která popisuje, co je Strojové Učení

> O počítačovém programu se říká, že se učí ze zkušenosti **E** s ohledem na nějakou třídu úkolů **T** a měření interpretace **P**, pokud se jeho interpretace při úkolech v **T**, měřený **P**, zlepšuje se zkušeností **E**. [13]

Dále je nutno určit, co je zkušenost:

- **T(ask)**, neboli úkol
- **P(erformance)**, neboli interpretace
- **E**(**xperience**), neboli zkušenost [9]

#### 3.2.1.1 Úkol

Úkoly Strojového učení jsou obvykle popsány z hlediska toho, jak by měl systém strojového učení zpracovat **vstupní data**. Jako vstupní data mužeme uvest nějakou **vlastnost** nebo **kolekce vlastnosti**, které byly kvantitativně změřeny z nějaké události, které chceme, aby systém učení zpracoval. Reprezentací vstupních dat je vektor $x \in \mathbb{R}^n$ kde každý vstup $x_i$ je další vlastnost.

Například, vlastnosti obrázku jsou obvykle **hodnoty pixelů** v tomto obrázku. [9]

Mužeme rozlišovat mezi třemi různými typy úkolů ve Strojovém učení, a to jsou:

1. **Supervised Learning**, neboli Učení z učitelem
2. **Unsupervised Learning**, neboli Učení bez učitelem
3. **Reinforcement Learning**, neboli Zpětnovazební učení

V této práci budu většínou pracovat s Učením z učitelem, a proto jej podrobně popíšu, ale budou popsány také Učení bez učitelem a Zpětnovazební učení, ale obecně.

#### 3.2.1.2 Interpretace

Například v takovém úkolu, jako je **Klasifikace** v Učení z učitelem, vždy musime měřit **accuracy**, neboli přesnost modelu. Přesnost modelu udává proporce dat pro které model produkuje správný výstup, takže opakem přesnosti modelu je **error rate**, neboli míra chyby, která udává proporce dat pro které produkuje NEsprávný výstup.

Nás zajímá, jak dobře, nebo špatně jestliže používáme míru chyb, algoritmus strojového učení interpretuje na datech, která dosud neviděl, protože to udává jak dobře ten model je zobecněn. Vyhodnotit interpretace mužeme pomoci testovací sady dat, neboli **data set**, která je oddělená od dat použitých k učení systemu. [9]

#### 3.2.1.3 Zkušenost

Jako zkušenost můžeme rozumět datovou sadu, ze které algorithmus učení může získávat *zkušenosti* a transformovat je na znalosti, neboli **zobecnění**. [9]

### 3.2.2 Učení z učitelem

Předpokládejme, že máme označený soubor vstupních dat:

$$\mathbf{D} = \{(x_n, t_n)\}_{n=1}^{N}$$

Kde $\mathbf{x}_n$ představuje vstupní, neboli vysvětlujíci proměnnou, která udává nějakou vlastnost, zatímco $\mathbf{t}_n$ je odpovídaící výstupní proměnnou, neboli odpověd'. Například proměnná $\mathbf{x}_n$ může představovat text emailu, plochu bytu, nebo kolik byt má pokojů, zatímco výstupní proměnna $\mathbf{t}_n$ může být například binární proměnná $\{0, 1\}$ označující, zda je email spam nebo ne.

Cílem Učení z učitelem je odhadnout hodnotu výstupní proměnne $\mathbf{t}_n$ pro nějaký vstup $\mathbf{x}_n$ který není v původním datovém souboru. Jinými slovy, učení z učitelem má za cíl dosahnout **generalizace**, neboli zobecnění modelu učení, aby ten mohl generovat nové znalosti na základě nových vstupů. Například algoritmus, naučený na sadě emailů by měl být schopen klasifikovat nový email (spam nebo ne), který se v sadě dat **D** nenachácí. [17]

Můžeme rozlišit dvě hlavní skupiny úkolů v Učení z učitelem:

1. **Classification** - v této úloze program požádán, aby určil, do které z **k** kategorií některý vstup patří. K vyřešení tohoto úkolu je obvykle vyžadován algoritmus učení k vytvoření funkce:

   $$f : \mathbb{R}^n \to \{1, \dots, k\}$$

   Když $y = f(x)$, model přiřadí vstup popsaný vektorem **x** kategorii označené výstupem **y**, který je označen číselným kodem. Jedním z nejdůležitějších typů klasifikačního úkolu například když je výstupem rozdělení pravděpodobnosti mezi kategorií. Příkladem

tohoto úkolu je **Object Detection**, neboli Rozpoznávání Objektů, toto je hlavním tématem této práce, kde vstupem je obraz a výstupem je číselný kod identifikující objekt v obrázku, nebo výstupem může být číseilná míra pravděpodobnosti, že se objekt na obrázku nachází. [9]

2. **Regression** - v této úloze program požádán, aby předpověděl číselnou hodnotu danou nějakým vstupem. K vyřešení tohoto úkolu je obvykle vyžadován algoritmus učení k vytvoření funkce:

$$f : \mathbb{R}^n \to \mathbb{R}$$

   Tento typ úkolu je podobný klasifikaci, ale formát výstupu je jiný. Dobrým příkladem regresní úlohy je predikce zítřejší teploty **t** na základě dnešních meteorologických pozorování **x**. [9] [17]

#### 3.2.2.1 Lineární regrese

Lineární regrese je nejjednodušší ze standardních nástrojů regrese, a je důležité ji popsat, protože je to nejjednodušší způsob, jak představit velmi důležité pojmy strojového učení, jako je [12]:

- **Cost/Loss function**, neboli Funkce Ztráty nebo Nákladová Funkce.
- **Gradient Descent**, neboli Gradientní sestup.
- **Overfitting**

Před popisem lineární regrese musíme udělat tří jednoduchých předpokladů [12]:

1. Nejprve předpokládáme, že vztah mezi nezávislými proměnnými $x$ a závislou proměnnou $y$ je lineární, tj. Že $y$ lze vyjádřit jako vážený součet prvků v $x$, vzhledem k určitému šumu na pozorováních.

2. Za druhé předpokládáme, že data mají normalní rozdělení, neboli Gaussovo rozdělení.

3. Za třetí, předpoklad linearity pouze říká, že vstup, neboli **hypoteza** lze vyjádřit jako vážený součet vlastnosti.

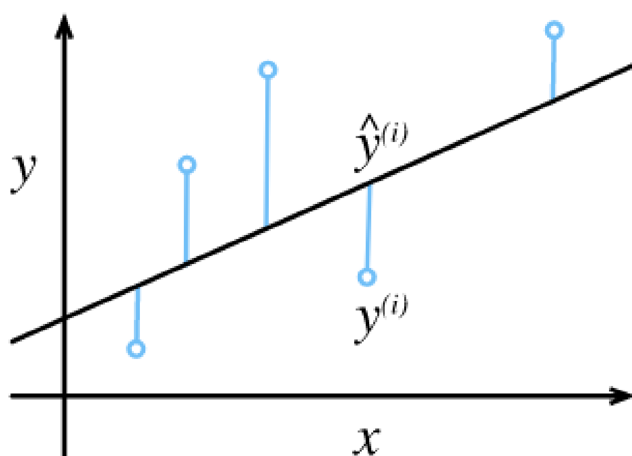
$$\hat{y} = b + w_1 x_1 + \cdots + w_k x_k$$

Kde $w$ je nějaká váha, neboli **weights**, $b$ je bias, váha určuje určuje vliv každé vlastnosti na předpověď' a bias určuje jakou hodnotu bude mít hypoteza, když všechny vlastnosti mají hodnotu 0. [12]

Takže můžeme stanovit, že vzhledem k datově sadě **D**, cílem lineární regrese je zvolit takové váhy $w$ a takový bias $b$, aby předpovědi provedené podle modelu, nejlépe odpovídaly pozorováni dat.

Ale, před tím než hledát nejlepší parametry $b, w$ je nutno uvést další tři důležité věci:

1. Funkce ztráty, je měřítko kvality pro daný model, protože před tím, než přizpůsobíme data našemu modelu, musíme určit míru přizpůsobení. Funkce ztráty měří vzdálenost mezi skutečnou hodnotou $y$ a předpokládanou hodnotou výstupu $\hat{y}$.




Obrázek 16: Přizpůsobit data lineárním modelem, kde $y^{(i)}$ je jediné pozorování a $\hat{y}^{(i)}$ je jediný odhad ze datové sady **D**. [12]

Nejpopulárnější ztrátovou funkcí je Střední Kvadratická Chyba, neboli **Mean Squared Error**, zkráceně **MSE**.

$$\mathbf{MSE} = L(b,w) = \frac{1}{N}\sum_{i=1}^{n}\frac{1}{2}(\hat{y}-y)^2$$

kde $\hat{y}$ je odhad, $y$ je pozorování, a $N$ je Počet pozorování v datové sadě **D**. Při učení modelu je nutné najít takové parametry $\{w,b\}$, které minimalizují celkovou ztrátu ve všech datových bodech.

2. Gradientní sestup, je proces aktualizace modelu, za účelem zlepšení jeho kvality, Gradientní sestup řeší optimalizaci tvaru
$$\min_{\theta} \sum_{i=1}^{n} f_n(\theta)$$
kde $\theta$ je vektor proměnných, které mají být optimalizovány, funkce ztráty $f_n(\theta)$ obvykle závisí na $n$-tom prvku v datové sadě **D**. [17]

3. Overfitting, případ přizpůsobení vstupních dat důkladněji, než odpovídá základnímu rozdělení. [12]

### 3.2.3 Učení bez učitelu

Předpokládejme, že máme NEoznačený soubor vstupních dat:

$$\mathbf{D} = \{(x_n)\}_{n=1}^{N}$$

Na rozdíl od dobře definovaného učení z učitelem, kde máme vztah mezi vstupními a výstupními daty, úkolem učení bez učitelu je obecně generovat tuto datovou sadu. Mezi konkétní úkoly patří například **Clustering**, ve kterém rozdělujeme datové sady do skupin z podobnými vlastostemí $x_n$. Algoritmy učení bez učitelu získávají zkušenosti z datové sady, která obsahuje mnoho vlastnosti, a poté se učí užitečné rysy struktury této datové sady.

**Semi-Supervised Learning** - je zobecněním učení z učitelem a učení bez učitelu, týká se případů, kdy ne všechny příklady jsou označeny, přičemž neoznačené příklady poskytují informace o distribuce proměnnych $x$. [17] [9]

### 3.2.4 Zpětnovazební učení

Zpětnovazební učení se týká problému vyvozování optimálních postupných rozhodnutí na základě odměn nebo trestů přijatých v důsledku předchozích akcí. V ramci učení, výstupy $t$ týká akce, která má být provedena, když je agent ve stavu získávání informace o prostředí danných proměnnou $x$. Po provedení akce $t$ ve stavu $x$ dostane agent zpětnou vazbu o okamžitě odměně, nebo trestu, získané tímto rozhodnutím a prostředí se přesune do jiného stavu. Jako příklad lze agenta naučit navigovat v nějakém prostředí s překážkami, musíme ho penalizovat, pokud narazí na jednu z těchto překážek

Zpětnovazební učení proto není ani učení z učitelem, protože agentovi nejsou poskytovány optimální akce $t$ pro výběr v daném stavu $x$, ani učení bez učitelu, vzhledem k dostupnosti zpětné vazby o kvalitě zvolené akce.

Zpětnovazební učení se také odlišuje od učení z učitelem a učení bez učitelu kvůli vlivu předchozích akcí na budoucí státy a odměny. [17]

### 3.2.5 Shrnutí Kapitoly

Na základě výše uvedené informace, můžeme usuzovat, že Strojové učení může nabidnout efektnivní a výkonnou alternativu k běžnému toku softwarového inženýrství, když může být problém příliš složitý na to, aby se dal studovat v celé své obecnosti.

Na druhé straně tento přístup má klíčové nevýhody spočívající v tom, že poskytuje obecně neoptimální interpretace nebo brání interpretovatelnosti řešení a aplikuje se pouze na omezený soubor problémů.

Dále budou popsány následující pojmy:

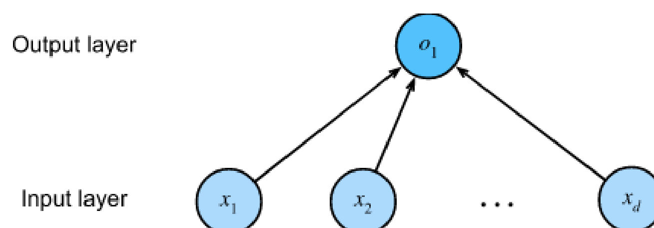
- **Deep Learning**, neboli Hluboké Učení.
- **Neural Network**, neboli Neuronová Sít'.
- **Convolutional Neural Network**, neboli Konvoluční neuronová sít'.

Což nás přivádí blíže k problému této práce, a to je Rozpoznávání Objektů.

## 3.3 Hluboké Učení

### 3.3.1 Neuronovou Síť

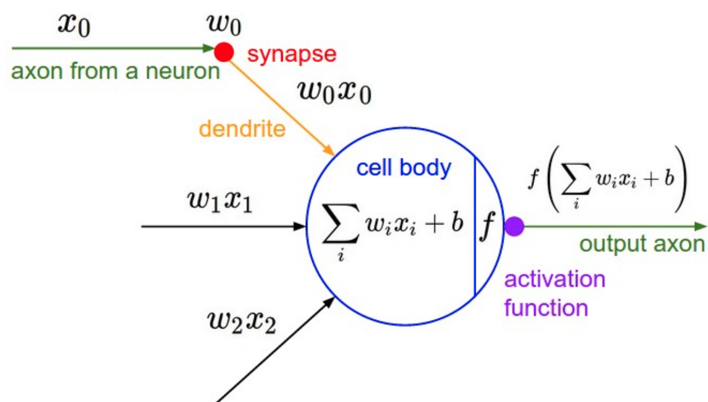
V předchozí části této práce byla představena a popsána lineární regrese, ale je možné ji zobrazit jako diagram neuronové sítě:




Obrázek 17: Jedná Vrstvá [12]

Pro neuronovou síť znázorněnou na obr.17, vstupy jsou $\{x_1, x_2, \ldots, x_d\}$, takže počet vstupů ve vstupní vrstvě je $d$. Výstup sítě označen jako $o_1$, takže počet výstupů ve výstupní vrstvě je roven 1. Jako poznámku je třeba uvést, že zde existuje pouze jeden neuron, protože se zaměřením na tom, kde výpočet probíhá, se obvykle nepovažujeme vstupní vrstvu při počítání vrstev. To znamená, že počet vrstev pro neuronovou síť na obr.17 je 1.

Model lineární regrese si můžeme představit jako neuronovou síť skládající se pouze z jednoho umělého neuronu nebo jako jednovrstvou neuronovou síť. A jak ukazuje diagram pro lineární regrese, každý vstup je připojen ke každému výstupu, lze jej považovat za plně spojenou vrstvu, neboli **fully-connected layer** nebo hustou vrstvu, neboli **dense layer**. [12]




Obrázek 18: Umělý Neuron, jako Biologický Neuron [11]

Umělý neuron můžeme představit jako biologický neuron, tak že Informace $x_i$ získané z prostředí jsou přijímány v *dendritech* nebo vstupech, zejmena jsou tyto informace váženy synaptickými váhami $w_i$, které určují účinek vstupů prostřednictvím produktu $w_i x_i$. Vážené vstupy přicházející z více zdrojů se shromažd'ují v *těle buňky* jako vážený součet $y = \sum_i x_i w_i + b$ a tato informace je poté odeslána k dalšímu zpracování v *axonu*, nebo výstupu $y$. [11] [12]

#### 3.3.1.1 Sigmoidní funkce

Sigmoidní funkce je zvláštním případem **logistické funkce**, kde
$L = 1, k = 1, x_0 = 0$. [12]

$$\sigma(z) = \frac{L}{1 + e^{-k(x - x_0)}}$$

kde $L$ je maximální hodnota, kterou může funkce nabývat, $k$ je parameter který řídí, jak prudká je změna z minimální na maximální hodnotu, $x_0$ je parameter který řídí, kde na na

ose $x$ by růst měl být. [12]

Sigmoidní neurony jsou podobné jednoduchým umělým neuronům, ale jsou upraveny tak, že malé změny v jejich váhy $w_i$ a jejich biasu $b_i$ způsobují jen malou změnu v jejich výstupu, je to důležité, protože namísto předpovědi přesně 0 nebo 1 sigmoid generuje hodnotu pravděpodobnosti mezi 0 a 1. Sigmoidní funkce pomáhá zpřesnit učení s každým krokem učení. [1]

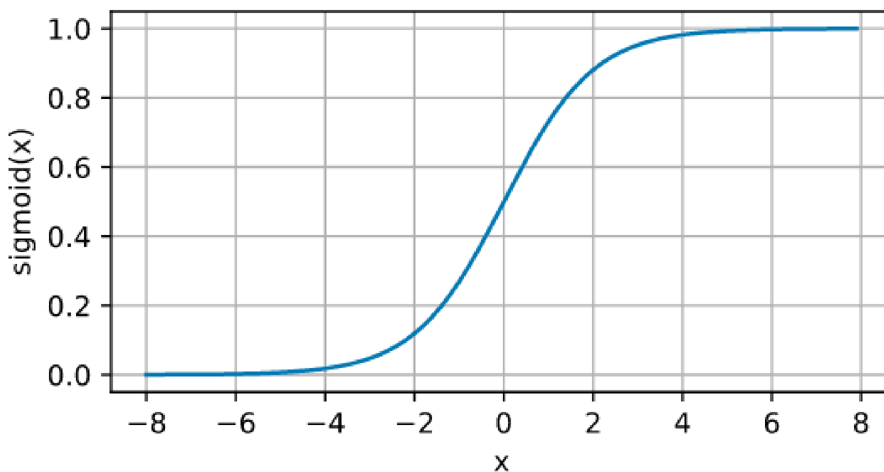
Tuto funkci můžeme reprezentovat jako:

$$\text{sigmoid}(x) = \sigma(z) = \frac{1}{1+e^{-z}}$$

kde $z = \sum_i x_i w_i + b$ a rozšířená funkce vypadá takto:

$$\text{sigmoid}(x) = \sigma(z) = \frac{1}{1+e^{-\sum_i x_i w_i - b}}$$

tvar funkce vypadá takto:




Obrázek 19: Sigmoida graph. [12]

#### 3.3.1.2 Architektura neuronové sítě

Například máme úlohu klasifikace obrázků, každý vstup se skládá z obrazu $320 \times 320$, můžeme představovat hodnoty každého pixelu jediným skalárem, což nám dává 102400 různých vlastnosti $x_1, x_2, \ldots, x_i$. Dále předpokládejme, že každý obrázek patří do jedné z kategorií obrázků $y \in \{0, 1, 2\}$, kde každé číslo představuje například **rajče, pomeranč a banán**. [12]

Nyní musíme představit naše kategorická data a metoda zvaná **one-hot encoding** se v tomto konkrétním úkolu hodí. Toto je metoda, ve které máme vektor s tolika prvky, kolik máme kategorií. Funguje to tak, že prvek ve vektoru odpovídající konkrétní kategorii je nastaven na 1 a všechny ostatní prvky jsou nastaveny na 0, například pokud má vstupní obrázek rajče, bude prvek výstupního vektoru vypadat takto $(1,0,0)$, a celý výstupní vektor bude trojrozměrný vektor, kde $(1,0,0)$ odpovídá **rajče**, kde $(0,1,0)$ odpovídá **pomeranču**, kde $(0,0,1)$ odpovídá **banánu**, můžeme výstup představovat jako $y \in \{(1,0,0),(0,1,0),(0,0,1)\}$. [12]

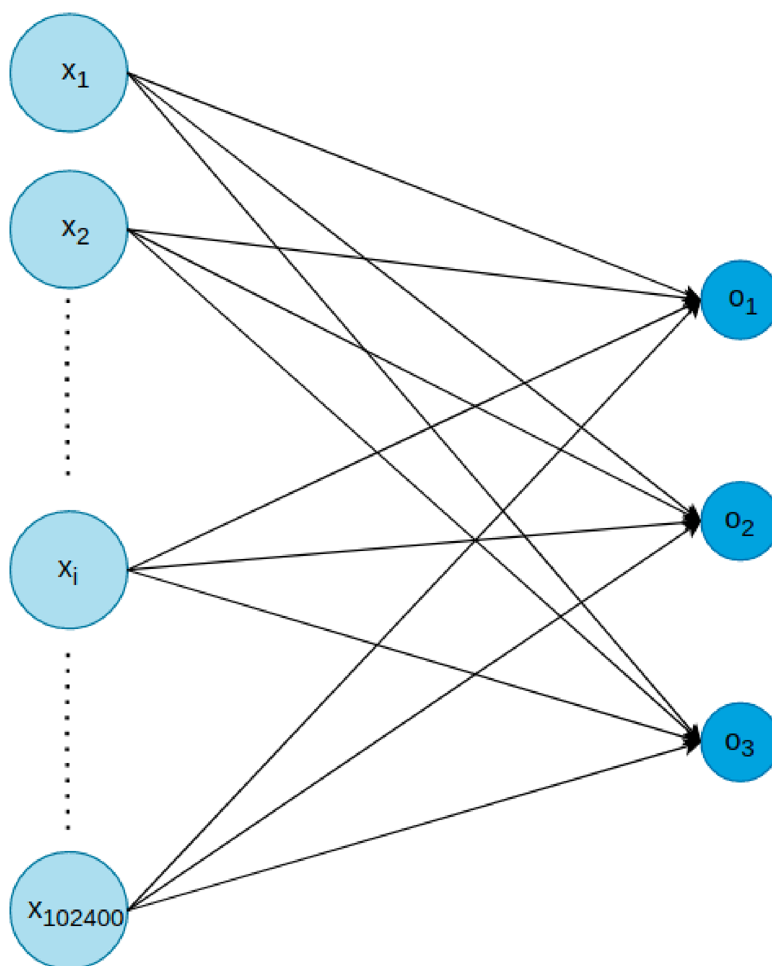
Abychom mohli odhadnout podmíněné pravděpodobnosti spojené se všemi možnými třídami, potřebujeme pro každou třídu model s více výstupy, a k tomu budeme potřebovat tolik funkcí, kolik máme výstupů, kde každý výstup bude odpovídat své vlastní funkci. [12]

V tomto příkladu máme 102400 vlastnosti a 3 výstupy, budeme potřebovat 307200 skalárů, které představují váhy $w_i$, a 3 skaláry, aby představovaly bias $b$. Takže pro každý vstup vypočítáme tři funkce $o_1, o_2, o_3$:

$$
\begin{aligned}
o_1 &= x_1 w_{1,1} + x_2 w_{1,2} + \cdots + x_i w_{1,i} + \cdots + x_{102400} w_{1,102400} + b_1, \\
o_2 &= x_1 w_{2,1} + x_2 w_{2,2} + \cdots + x_i w_{2,i} + \cdots + x_{102400} w_{2,102400} + b_2, \\
o_3 &= x_1 w_{3,1} + x_2 w_{3,2} + \cdots + x_i w_{3,i} + \cdots + x_{102400} w_{3,102400} + b_3.
\end{aligned}
$$

Je důležité si uvědomit, že se nejedná o logistickou regresní úlohu, ale o **softmaxovou regresní úlohu**. Softmax, jako je sigmoidní funkce, je zvláštním případem logistické funkce, ale pro více dimenzí, protože máme trojrozměrný výstupní vektor, bude použita softmax regrese. [12]

K znázornění tohoto výpočtu použijeme diagram neuronové sítě zobrazený na obr.20. Softmax regrese bude jednovrstvá neuronová síť. A protože výpočet každého výstupu, $o_1, o_2, o_3$, závisí na všech vstupech, $x_1, x_2, x_{102400}$, lze výstupní vrstvu softmax regrese také popsat jako plně připojenou vrstvu. [12]




Obrázek 20: Architektura neuronové sítě s více výstupy

### 3.3.2 Vícevrstvá Neuronová Síť

Vícevrstvé Neuronové Sítě, neboli **Multilayered Neural Networks**, zkráceně **MLP**, nazývané také Dopředné Neuronové Sítě, neboli **Feedforward Neural Networks**, zkráceně **FNN**, pravděpodobně jeden z nejdůležitějších konceptů v Deep Learning. Je užitečné, když lineární modely, jako je lineární regrese, logistická regrese nebo jednovrstvá neurální síť, nestačí k osvojení libovolné hranice rozhodnutí, protože kapacita modelu je omezena na lineární funkce, to znamená, že model nemůže porozumět interakci mezi dvěma libovolnými vstupy.

Cílem FNN je přiblížit nějakou funkci $f^*$, jako například klasifikátor $y = f^*(x)$ mapuje vstup $x$ na kategorii $y$. FNN definuje mapování $y = f(x;\theta)$ a učí se hodnotu parametrů $\theta$, která vede k nejlepší aproximaci funkce. FNN jsou v zásadě směrovaný acyklický graf, který

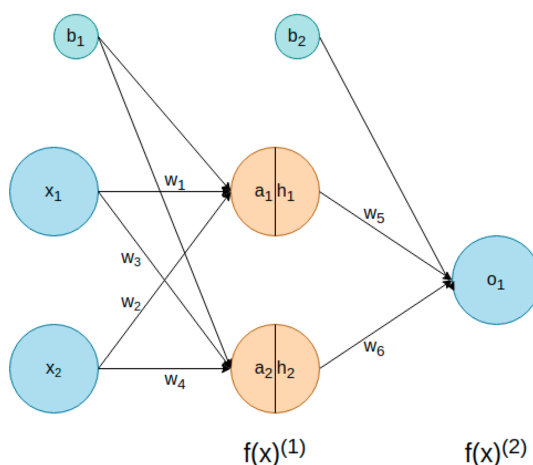
představuje sadu složených funkcí a popisuje, jak jsou tyto funkce složeny. Jako příklad máme dvě funkce $f^{(1)}, f^{(2)}$ spojené v řetězci, abychom vytvořili složenou funkce $f(x) = f^{(2)}(f^{(1)}(x))$, kde $f^{(1)}$ je první vrstva sítě a $f^{(2)}$ je druhá vrstva. Celková délka řetězu udává hloubku modelu. Délka tohoto řetězce určuje **hloubku** celého modelu. [9]

Na rozdíl od jednovrstvé neuronové sítě má FNN skryté vrstvy, neboli **hidden layes**, a jejich chování není přímo specifikováno vstupními daty. Algoritmy učení se musí rozhodnout, jak použít tyto vrstvy k vytvoření požadovaného výstupu, ale vstupy neurčují, co by každá jednotlivá vrstva měla dělat. Místo toho se algoritmus učení musí rozhodnout, jak použít tyto vrstvy k provedení aproximace $f^*$, proto je nazvali skryté vrstvy, protože vstupní data nezobrazují požadovaný výstup pro každou z těchto vrstev. Skryté vrstvy se skládají ze skrytých jednotek a každá z těchto jednotek přijímá vstup z mnoha dalších jednotek a počítá svou vlastní **aktivační hodnotu**, kterou lze vypočítat pomocí aktivační funkce. [9]

Aktivační funkce, neboli **Activation functions**, rozhodují o tom, zda by měl být neuron aktivován či nikoli, výpočtem váženého součtu a přidáním biasu. Sigmoidní funkce je široce používána jako aktivační funkce na výstupních jednotkách, když chceme interpretovat výstupy jako pravděpodobnosti pro klasifikační úkoly. [12]

FNN jsou velmi důležité, protože tvoří základ mnoha důležitých aplikací, jako například detekce objektů, ve které se používají **Konvoluční Neuronové Sítě**, což jsou zvláštní verze FNN. [9]

Můžeme zobrazit diagram FNN následovně:




Obrázek 21: Dvouvrstvá Neuronová Síť

Dopředné šíření, neboli **Forfward Propagation** FNN se týká výpočtu a ukládání přechodných proměnných pro FNN v následujícím pořadí, počínaje od vstupní vrstvy, přes skrytou vrstvu, až po výstupní vrstvu. Nyní se pokusíme vypočítat výstup $o_1$ z obr.21 krok za krokem. Nejprve musíme vypočítat aktivační hodnoty $h_1$ a $h_2$, jako aktivační funkce bude použita sigmoidní funkce, ale předtím musíme vypočítat vážený součet vstupů $a_1$ a $a_2$:

$$a_1 = w_1x_1 + w_2x_2 + b,$$
$$a_2 = w_3x_1 + w_4x_2 + b.$$
$$h_1 = \frac{1}{1+e^{-a_1}},$$
$$h_2 = \frac{1}{1+e^{-a_2}}.$$
$$o_1 = w_5h_1 + w_6h_2 + b_2.$$

Jak již bylo dříve definováno, můžeme funkci učení pro dvouvrstvou síť zobrazit jako složenou funkci: $f(x) = f^{(2)}(f^{(1)}(x))$ kde $f^{(1)}(x)$ je funkce naučená na první skryté vrstvě a $f^{(2)}(x)$ je funkce naučená na výstupní vrstvě. [12]

Zpětné šíření, neboli **Back Propagation** FNN se týká metody výpočtu gradientu parametrů neuronové sítě, tato metoda prochází sítí v opačném pořadí, od výstupu, přes skryté vrstvy, zpět do vstupní vrstvy, podle řetízkového pravidla z matematické analýzy. Algoritmus ukládá jakékoli přechodné proměnné, které jsou určeny (parciálními derivacemi) požadovanými při výpočtu gradientu s ohledem na některé parametry. Zpětné šíření vede k jemnému doladění, neboli **fine-tuning**, počátečních váh na základě hodnoty **ztráty**. [12].

### 3.3.3 Convolutional Neural Networks

Hlavní problém s neuronovými sítěmi, když začneme pracovat s obrazovými vstupy, že jsou hustě propojeny, každý neuron ve vrstvě je spojen s každým neuronem v další vrstvě. A když chceme z obrázku extrahovat množství vlastnosti, například ze vstupu, máme obrázek s rozlišením například $320 \times 320$, bude to obraz RGB, takže k němu musíme přidat 3 kanály RGB, nyní to vypadá takto $320 \times 320 \times 3$. Takže pro extrahování vlastnosti z tohoto obrázku si vezmeme $8 \times 8$ kousek tohoto obrázku, takže dostaneme obrázek, který je $40 \times 40$ a každý z těchto $8 \times 8$ kousků bude mít 36 vlastnosti spojené s jim, takže tento **mapa vlastnosti** je $40 \times 40 \times 36$. Pro přehlednost, **image patch**, neboli je obrazová záplata jakýmsi kontejnerem pixelů, například pokud máme obrázek $320 \times 320$, záplata obrázku $8 \times 8$ je 64 pixelů tohoto

obrázku a obrázek $320 \times 320$ má 1600 $8 \times 8$ záplat. Při zpracování obrazu jsou obrazové záplaty užitečné pro výpočet funkcí histogramu orientovaných přechodů (Histogram of oriented gradients nebo zkraceně HOG). Pokud se ale v Deep Learning pokusíme extrahovat vlastnosti ze vstupního obrazu pomocí hustě propojených neuronových sítí, bude naše vstupní vrstva hustě propojena se skrytými vrstvami, což znamená, že každá vstupní jednotka je připojena ke každé skryté jednotce a ve výsledku budeme mít 17694720000 spojení mezi vstupní vrstvou a skrytou vrstvou. A $320 \times 320$ je relativně malý obrázek a vyžaduje již jen obrovské množství výpočtu. Takže tam je místo, kde se hodí Convolutional Neural Networks. [5]

Proč tedy používat CNN?

1. Nejprve jsou CNN velmi podobné běžným neuronovým sítím, to znamená, že jsou tvořeny neurony, které mají váhy $w_i$ pro učení.

2. Každý neuron přijímá nějaké vstupy, provádí skalární součin a případně následuje ji nelinearity.

3. Celá síť stále vyjadřuje jednu funkci: od obrazku na vstupu až po detekce třídy na výstupu.

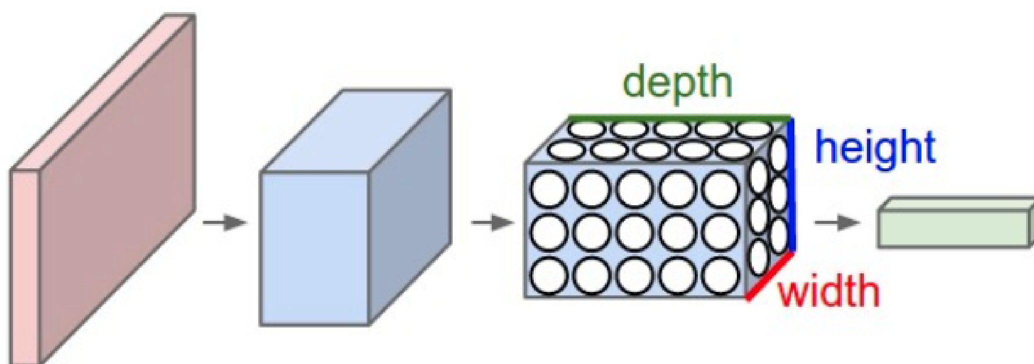
4. CNN stále mají funkci ztráty na poslední plně připojené vrstvě.

[11]

Architektury CNN výslovně předpokládají, že vstupy jsou obrázky, což umožňuje zakódovat do architektury určité vlastnosti. Ty pak zefektivňují implementaci funkce vpřed a výrazně snižují množství parametrů v síti. Zde se objevují dva hlavní typy vrstev:

- **Convolution Layer**, neboli Konvoluční vrstva, tento typ vrstvy je základním stavebním kamenem CNN, který provádí většinu výpočetního těžkého zvedání. [11]

- **Pooling Layer**, je pravidelně vložit Pooling vrstvu mezi následné vrstvy Convolution v architektuře, protože hlavní funkcí Pooling vrstvy je postupně zmenšovat prostorovou velikost reprezentace, snižovat množství parametrů a výpočtů v síti a kontrolovat *Overfitting*. [11]




Obrázek 22: prostorové schéma CNN `https://cs231n.github.io/assets/cnn/cnn.jpeg`

Jak již bylo řečeno, pravidelné neurální sítě se neškálí dobře na plné obrázky a CNN zde využívá výhody, protože na základě skutečnosti, že vstup se skládá z obrazů, dává nám trojrozměrný objem neuronů to znamená, že CNN neurony uspořádány ve 3 dimenzich. Například obrázek $320 \times 320 \times 3$ má 3 rozměry, kde $320 \times 320$ je šířka a výška a 3 je hloubka, která představuje barvu. Je nutno uvést, že hloubka zde není stejná jako ve ulně připojené sítě, protože hloubka zde odkazuje na třetí dimenzi, nikoli na celkový počet vrstev. Jak můžeme vidět na Obr.22, každá vrstva CNN transformuje objem trojrozměrný vstupu do svazku trojrozměrný výstup neuronových aktivací. Na obr. 22 červená vstupní vrstva drží obraz, takže jeho šířka a výška budou rozměry obrazu a hloubka bude 3.

#### 3.3.3.1 Konvoluční vrstva

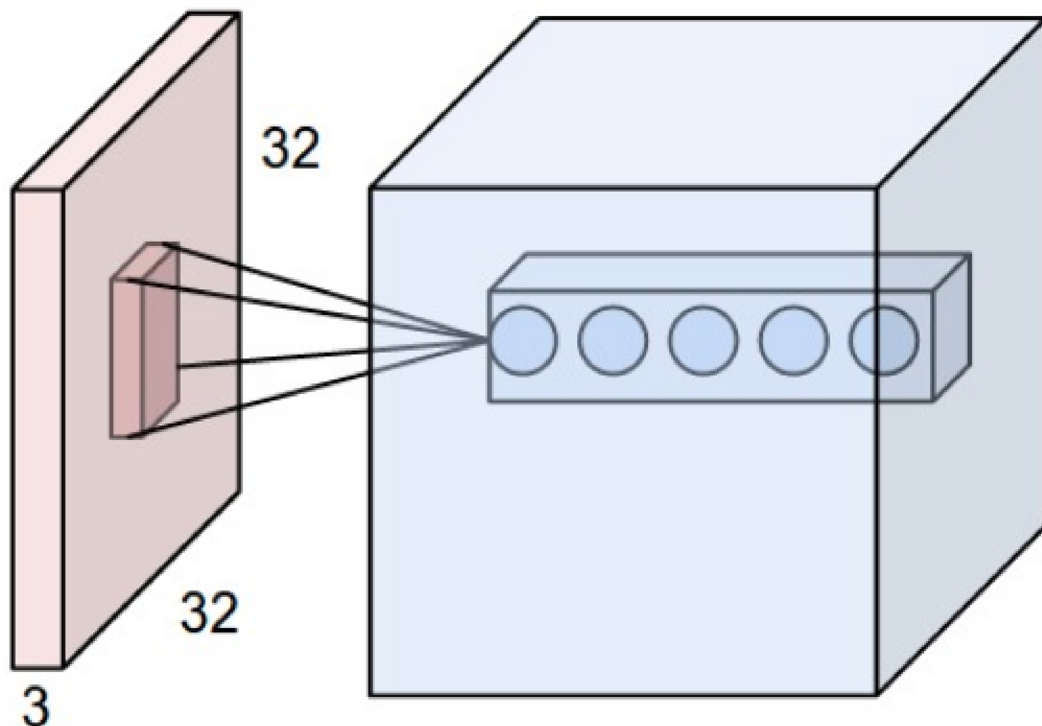
Jedním z nejdůležitějších konceptů v CNN je Konvoluční Vrstva, ve které se parametry skládají ze sady filtrů. Každý filtr je prostorově malý, viz šířka $\times$ výška, ale prochází celou hloubkou vstupního objemu. [11]

Jako příklad může mít typický filtr na první vrstvě CNN velikost $5 \times 5 \times 3$, tj. 5 pixelů na šířku, 5 pixelů na výšku a 3 je hloubka nebo barevné kanály. Během dopředného šíření konvertujeme každý filtr přes šířku a výšku vstupního objemu a počítáme skalární součiny mezi údaje filtru a vstupu v libovolné poloze. Když konvertujeme filtr na šířku a výšku vstupního objemu, vytvoříme dvourozměrnou aktivační mapu, která poskytne odpovědi daného filtru v každé prostorové poloze. Síť se naučí filtry, které se aktivují, když uvidí nějaký typ vizuální vlastnosti, poté budeme mít celou sadu filtrů v každé Konvoluční vrstvě a každý z nich vytvoří samostatnou dvourozměrnou aktivační mapu, pak spojíme tyto aktivační mapy

podél hloubky a vytváříme výstupní objem.[11]

Zajímavou částí konvolučních vrstev je, že mají **lokální konektivitu**, to znamená, že každý neuron spojíme pouze s lokální oblastí vstupního objemu, protože při práci s vícerozměrnými vstupy, jako jsou obrázky, je nepraktické připojovat neurony ke všem neuronům v předchozí vrstvě. Je také třeba poznamenat, že jediný rozdíl mezi plně spojenými vrstvami a konvolučnými vrstvami spočívá v tom, že neurony ve konvoluční vrstvě jsou připojeny pouze k místní oblasti na vstupu a že mnoho neuronů v parametrech sdílení objemu konvoluce. Neurony v obou vrstvách však stále počítají skalární součiny, takže jejich funkční forma se nezměnila.[11]

Prostorovým rozsahem této konektivity je hyperparametr nazývaný **receptivní pole** neuronu, můžeme říci, že receptivní pole je také velikostí filtru. Rozsah připojení podél hloubkové osy se vždy rovná hloubce vstupního objemu. Je důležité znovu zdůraznit tuto asymetrii v tom, jak vidíme prostorové dimenze a hloubkovou dimenzi, spojení jsou lokální v prostoru, ale vždy plná po celé hloubce vstupního objemu. [11]




Obrázek 23: Konvoluce, `https://cs231n.github.io/assets/cnn/depthcol.jpeg`

Jako příklad na obr. 23 je vstupní objem v červené barvě obrazem $32 \times 32 \times 3$. Každý neuron v konvoluční vrstvě je spojen pouze s lokální oblastí ve vstupním objemu prostorově, ale do celé hloubky. Je důležité si uvědomit, že zde několik neuronů podél hloubky, všechny

se dívají na stejnou oblast ve vstupu.[11]

Prostorové uspořádání nebo hyperparametry, které řídí velikost výstupního objemu:

1. **Depth**, neboli Hloubka je hyperparametr který odpovídá počtu filtrů, které bychom chtěli použít, přičemž každý se na vstupu učí hledat něco jiného. Například pokud první konvoluční vrstva vezme jako vstup obrázek, pak se mohou různé neurony podél hloubkové dimenze aktivovat v přítomnosti různě orientovaných okrajů nebo barevných bloků. Můžeme označit sadu neuronů, které se dívají na stejnou oblast vstupu jako sloupec hloubky.[11]

2. **Stride**, je hyperparametr, který posune filtr. Jako příklad, pokud je krok 1, pak filtry přesuneme o jeden pixel najednou.[11]

3. **Zero-padding**, nebo **padding**, je množství pixelů přidaných kolem hranice obrázku, když je zpracováván filtrem.[11]

Můžeme vypočítat prostorovou velikost výstupního objemu jako funkci velikosti vstupního objemu $W$, velikosti receptivního pole neuronů konvoluční vrstvy $F$, stride $S$, a množství Zero-padding na hranice $P$. Vzorec pro výpočet, můžeme představovat jako $(W - F + 2P)/S + 1$. Například pro vstup $7 \times 7$ a filtr $3 \times 3$ se stride 1 a Zero-padding 0 bychom dostali výstup $7 \times 7$, nebo $(7 - 3 + 2 * 0)/1 + 1$ se rovná 5, v tomto případě $5 \times 5$.[11]

Takže na závěr můžeme říci, že Convolution Layer:

- Přijímá objem o velikosti $W_1 \times H_1 \times D_1$.

- Vyžaduje čtyři hyperparametry, počet filtrů $K$, prostorový rozsah, nebo receptivní pole $F$, Stride $S$, nějaké množství Zero-padding $P$ (může být 0).

- Vytvoří objem o velikosti $W_2 \times H_2 \times D_2$, kde $W_2, H_2 = (W_1, H_1 - F + 2P)/S + 1$ and $D_2 = K$. [11]

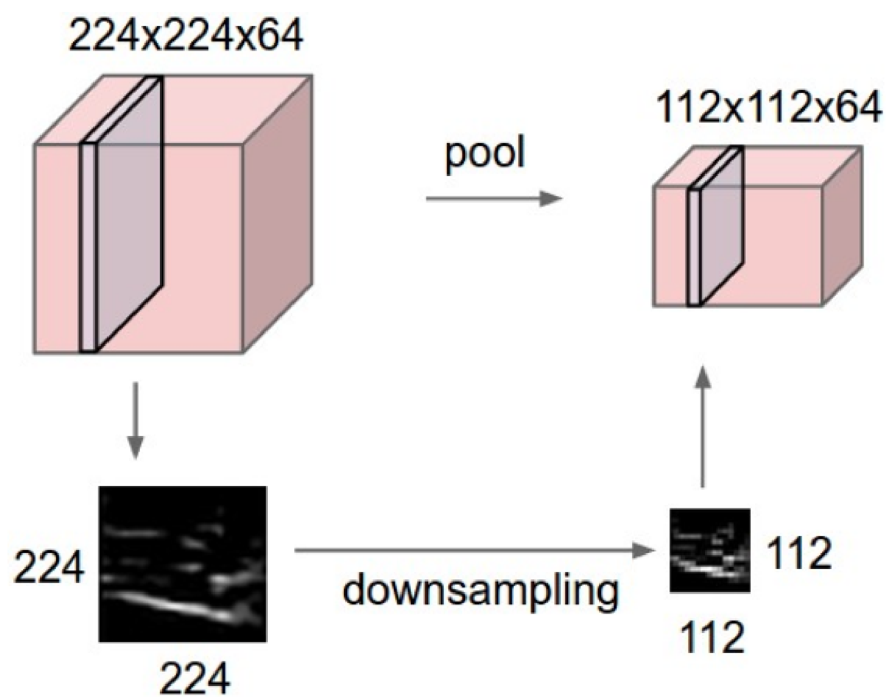
#### 3.3.3.2 Pooling vrstva

Cílem Pooling vrstva je postupně zmenšovat prostorovou velikost reprezentace, aby se snížilo množství parametrů a výpočtů v síti, a tudíž také kontrolovat overfitting. Pooling vrstva pracuje nezávisle na každém řezu hloubky vstupu a mění jej prostorově. Nejběžnější formou je sdružená vrstva s filtry o velikosti $2 \times 2$ aplikovanými s použitím stride o velikosti

2 a převzorkování (Downsampling) každého hloubkového řezu na vstupu o 2 podél šířky i výšky, zahodí cca 75 procentů aktivací. Každá operace by v tomto případě brala 4 čísla, a to kvůli oblasti hloubky $2 \times 2$. Dimenze hloubky zůstává nezměněna. [11]

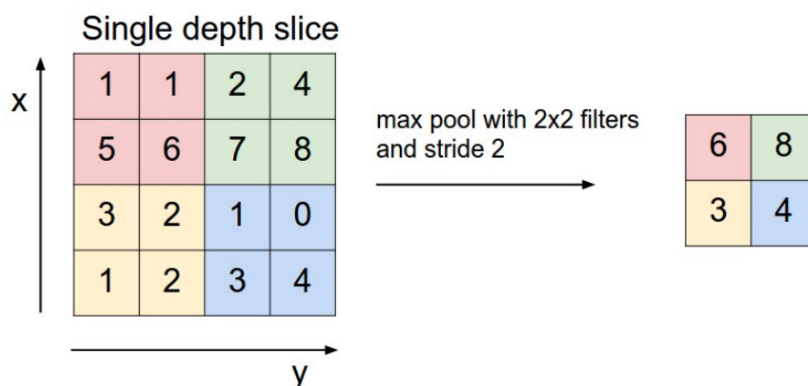
Takže na závěr můžeme říci, že Pooling vrstva:

- Přijímá objem o velikosti $W_1 \times H_1 \times D_1$.
- Vyžaduje dva hyperparametry, počet filtrů $F$ a stride $S$.
- Vytvoří objem o velikosti $W_2 \times H_2 \times D_2$, kde $W_2, H_2 = (W_1, H_1 - F)/S + 1$ and $D_2 = D_1$.
- při použití Pooling vrstvy, není běžné používat Zero-padding pro vstup. [11]




Obrázek 24: Pooling `https://cs231n.github.io/assets/cnn/pool.jpeg`

Pooling vrstva převzorkuje (Downsampling) objem prostorově, nezávisle na každém řezu hloubky vstupního objemu. Na obr.24 je vstupní objem o velikosti $224 \times 224 \times 64$ sloučen s velikostí filtru 2, stride 2 do výstupního objemu o velikosti $112 \times 112 \times 64$, jak můžete vidět, že hloubka objemu je zachována. [11]




Obrázek 25: Maxpooling `https://cs231n.github.io/assets/cnn/maxpool.jpeg`

Nejběžnější operace převzorkování (Downsampling) je operace MAXPOOLING, což vede k maximálnímu sdružování, na obr.25 je ukázáno se stride 2. To znamená, že každá max je převzata přes 4 čísla v čtverci $2 \times 2$ [11] [5]

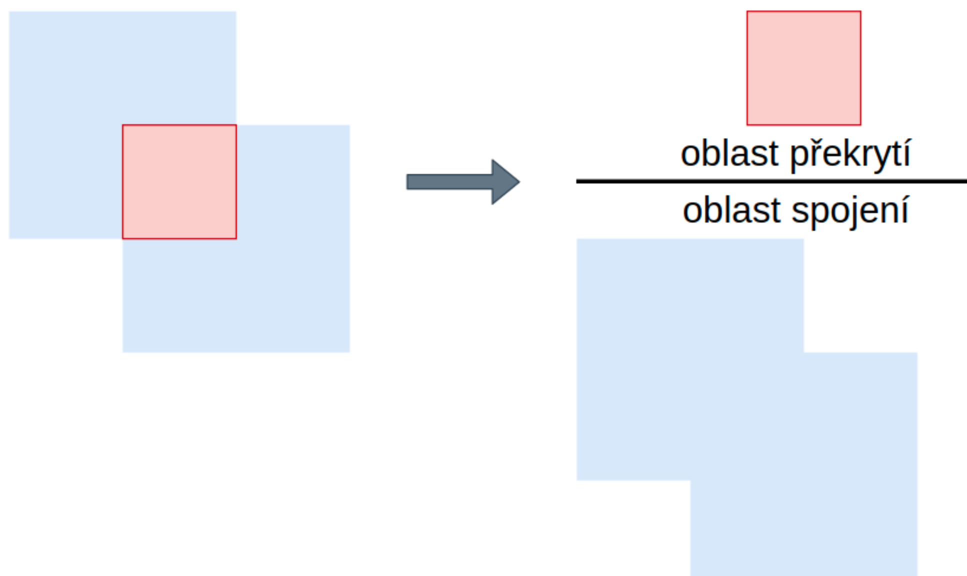
## 3.4 Object Detection

Detekce objektů je počítačová technologie související s počítačovým viděním a zpracováním obrazu, která se zabývá detekcí instancí sémantických objektů určité třídy (automobilů, ovoce, lidí) v digitálních obrazech. Domény detekce objektů zahrnují také detekci tváří a detekci chodců na ulici. Koncept detekce objektů je docela přímočarý, každá třída objektů má své vlastní speciální funkce, které pomáhají při klasifikaci třídy. Například všechny kruhy jsou kulaté a když hledáme kruhy, hledají se objekty, které jsou v určité vzdálenosti od bodu, například od středu.[14]

### 3.4.1 Intersection over Union

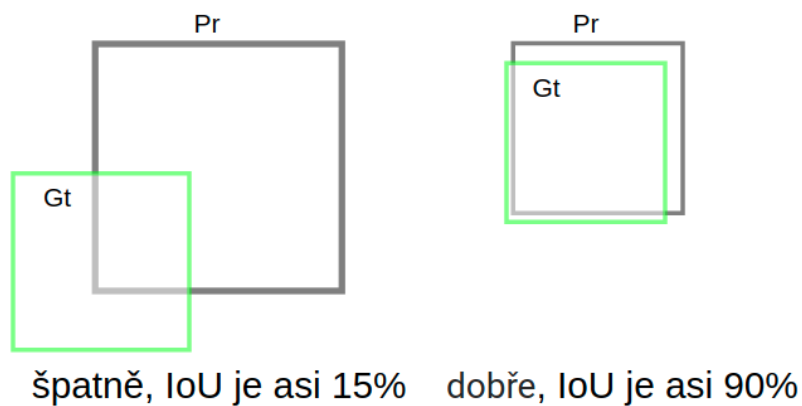
Před popisem prvků detekce objektů bychom měli nejprve probrat, jaké metriky se snažíme optimalizovat. Hlavním úkolem při detekci objektů, jak je znázorněno na obr.26, je umístit přesné ohraničující boxy (Bounding Boxes) kolem všech objektů zájmu a tyto objekty správně označit. Pro měření přesnosti každého ohraničujícího boxu je běžnou metrikou **Intersection over Union** nebo zkraceně **IoU**. IoU se počítá pomocí předpovězených ohraničujících boxů $B_{\text{pr}}$ a **ground truth** ohraničujících boxů $B_{\text{gt}}$ pro objekt a výpočet poměru jejich oblasti překrytí a jejich oblasti spojení, můžeme to znázornit jako [18]:

$$\frac{B_{\text{pr}} \cap B_{\text{gt}}}{B_{\text{pr}} \cup B_{\text{gt}}}$$




Obrázek 26: IoU [2]

Detektory objektů fungují tak, že nejprve navrhnou řadu přijatelných detekcí a poté každou detekci klasifikují a zároveň vytvoří skóre spolehlivosti. Tyto oblasti pak procházejí jakýmsi stupněm **non-maximal suppression**, zkraceně NMS, které odstraňuje slabší detekce, které se příliš překrývají se silnějšími detekcemi. [18]




Obrázek 27: IoU příklad [2]

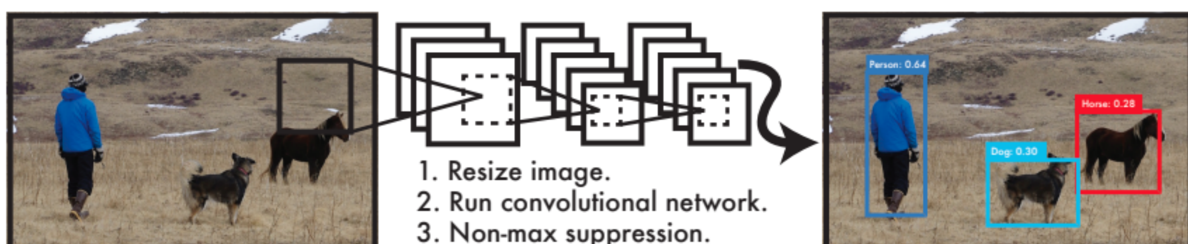
Abychom mohli vyhodnotit výkon detektoru objektů, projdeme všemi detekcemi, od nejdůvěryhodnějších po nejmenší, a klasifikujeme je *true positive* TP, je to správný label a vysoká hodnota IoU nebo *false positive* FP, je to nesprávný label. Pro každý nový klesající práh spolehlivosti můžeme vypočítat *precision* a *recall* jako [18]:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}},$$
$$\text{recall} = \frac{\text{TP}}{\text{P}}$$

kde P je počet pozitivních vzorků, to znamená že počet označených **ground truth** detekcí v obrazu, TP je True Positive a FP je False positive, jak bylo dříve definováno. Výpočet přesnosti a vyvolání při každém prahu spolehlivosti nám umožňuje naplnit křivku vyvolání přesnosti. Oblast pod touto křivkou se nazývá *average precision* nebo AP. Pro každou detekovanou třídu lze vypočítat samostatné skóre **AP** a výsledky zprůměrovat tak, aby vytvořily *mean average precision* nebo **mAP**. [18]

### 3.4.2 YOLOv3 a Darknet Framework

You Look Only Once, nebo zkraceně YOLO, je metoda detekce objektů, která využívá neuronové sítě.




Obrázek 28: Přístup YOLO [10]

YOLO je docela jednoduchá metoda, viz obr.28. CNN současně předpovídá více ohraničujících boxů a pravděpodobnosti tříd pro tyto boxy. YOLO se může učit na obrázcích v plném rozlišení a přímo optimalizuje výkon detekce. [10]

Můžeme tedy stanovit tři důležité body týkající se metody YOLO:

1. YOLO mění velikost vstupního obrazu, například na $320 \times 320$.

2. YOLO na obrázku spustí CNN.

3. YOLO prahuje výsledné detekce pomocí skóre spolehlivosti.

Tento přístup má několik výhod oproti jiným metodám detekce objektů:

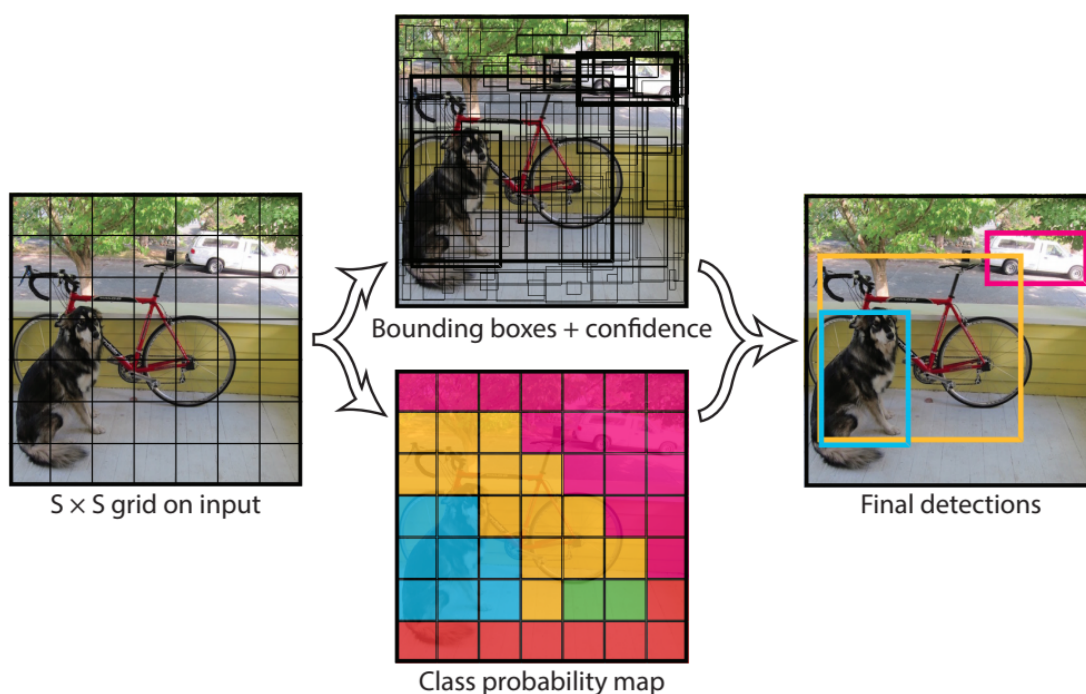
- YOLO je velice rychlý. Vzhledem k tomu, že detekci vytváříme jako regresní problém, nepotřebujeme složitou strukturu projektu a bude to ukázáno v praktické části této práce.

- YOLO přemýšlí o obrazu při vytváření předpovědí jako celku. To je ono, YOLO vidí celý obraz během učení a testování, takže kóduje kontextové informace o třídách a jejich objevu.

- YOLO se učí zobecnitelné reprezentace objektů, protože YOLO je vysoce zobecnitelné, je méně pravděpodobné, že se zlomí, když se použije na nové třídy nebo neočekávané vstupy. [10]

YOLO sjednocuje jednotlivé komponenty detekce objektů do jedné neuronové sítě. YOLO používá vlastnosti z celého obrázku k předpovědi každého ohraničujícího rámečku. Rovněž předpovídá všechny ohraničující boxy napříč všemi třídami pro obrázek současně.[10]

To znamená, že metoda YOLO myslí na celý obraz jako celek a všechny objekty v obrazu. Konstrukce YOLO umožňuje end-to-end učení a rychlost v reálném čase při zachování vysoké průměrné přesnosti. YOLO rozděluje vstupní obraz do mřížky $S \times S$. Pokud střed objektu spadne do buňky mřížky, je tato buňka mřížky zodpovědná za detekci tohoto objektu, to bude vysvětleno dále. [10]




Obrázek 29: YOLO model [10]

Každá buňka mřížky předpovídá B ohraničující boxy a skóre spolehlivosti pro tyto boxy. Tato skóre spolehlivosti odrážejí, jak jistý je model, že nějaký box obsahuje objekt. Spolehlivost je definována jako:

$$\mathrm{Pr}(\mathrm{Object}) * \mathrm{IOU}_{\mathrm{pred}}^{\mathrm{truth}}$$

Pokud v této buňce neexistuje žádný objekt, skóre spolehlivosti by mělo být nula. Jinak chceme, aby se skóre spolehlivosti rovnalo IOU mezi předpovězeným boxem $B_{\mathrm{pr}}$ a **ground truth** boxem $B_{\mathrm{gt}}$. [10]

Každá buňka mřížky také předpovídá podmíněnou pravděpodobnost třídy C, $\mathrm{Pr}(\mathrm{Class}_{\mathrm{i}}|\mathrm{Object})$. Tyto pravděpodobnosti jsou podmíněny buňkou mřížky obsahující objekt. Předpovídáme pouze jednu sadu pravděpodobností třídy na buňku mřížky, bez ohledu na počet boxu B. Pak vynásobíme pravděpodobnosti podmíněné třídy a předpovědi spolehlivosti jednotlivých boxu [10]:

$$\mathrm{Pr}(\mathrm{Class}_{\mathrm{i}}|\mathrm{Object}) * \mathrm{Pr}(\mathrm{Object}) * \mathrm{IOU}_{\mathrm{pred}}^{\mathrm{truth}} = \mathrm{Pr}(\mathrm{Class}_{\mathrm{i}}) * \mathrm{IOU}_{\mathrm{pred}}^{\mathrm{truth}}$$

Což nám dává konkrétní skóre spolehlivosti pro třídu a pro každý ohraničující box. Tato

skóre představují jak pravděpodobnost, že se tato třída objeví v boxu a jak dobře se předpokládaný box hodí k objektu.[10]

Jak vidíte na obr.29, detekce modelů YOLO jako regresní problém funguje tak, že:

- Rozdělí obraz do mřížky $S \times S$.
- Pak YOLO předpovídá B ohraničující boxy pro každou buňku mřížky, spolehlivost pro tyto boxy a a pravděpodobnosti třídy pro objekty, které jsou přítomny na obrázku C.
- Tyto předpovědi jsou reprezentovány jako $S \times S \times (B * 5 + C)$.[10]

Predikce ohraničujícího boxu funguje tak, že systém předpovídá ohraničující boxy pomocí sady předdefinovaných ohraničujících boxů známých jako **Anchor** boxy, síť předpovídá 4 souřadnice pro každý box $(t_x, t_y, t_w, t_h)$. Pokud je buňka odsazena od levého horního rohu obrázku o $(c_x, c_y)$ a ohraničující box před má šířku a výšku $(p_w, p_h)$, pak předpovědi odpovídají: [6]

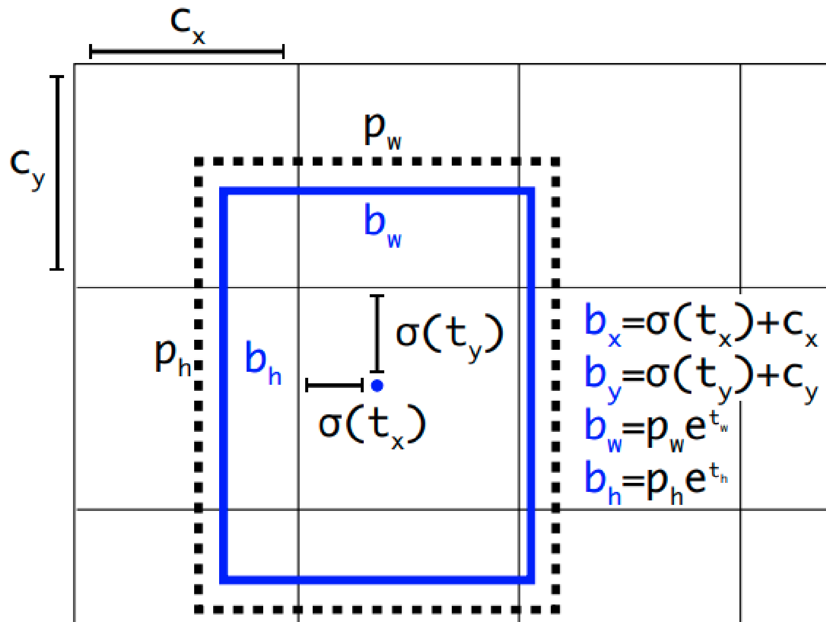
$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x, \\
b_y &= \sigma(t_y) + c_y, \\
b_w &= p_w e^{t_w}, \\
b_h &= p_h e^{t_h}.
\end{aligned}
$$

Jak vidíte na obr. 30, YOLO předpovídá šířku a výšku rámečku jako posunutí od geometrický středů, poté předpovídá středové souřadnice rámečku vzhledem k umístění filtru pomocí sigmoidní funkce. [6]

Na závěr tématu o předpovědí ohraničujícího boxu bychom mohli konstatovat:

- Souřadnice ohraničujícího boxu B, YOLO předpovídá čtyři souřadnice pro každé ohraničující rámeček $(b_x, b_y, b_w, b_h)$, kde $x$ a $y$ jsou nastaveny jako posuny umístění buňky. [6]
- YOLOv3 předpovídá skóre spolehlivosti pro každý ohraničující box pomocí logistické regrese. To by mělo být 1, pokud ohraničující rámeček dříve překrývá **ground truth** objekt o více než jakýkoli jiný ohraničující rámeček dříve. Pokud ohraničující box není nejlepší, ale překrývá **ground truth** objekt o více než určitou prahovou hodnotu, ignorujeme předpověď.[6]

- Pokud ohraničovací box obsahuje objekt, Ypředpovídá pravděpodobnost tříd K, kde K je celkový počet tříd. [6]



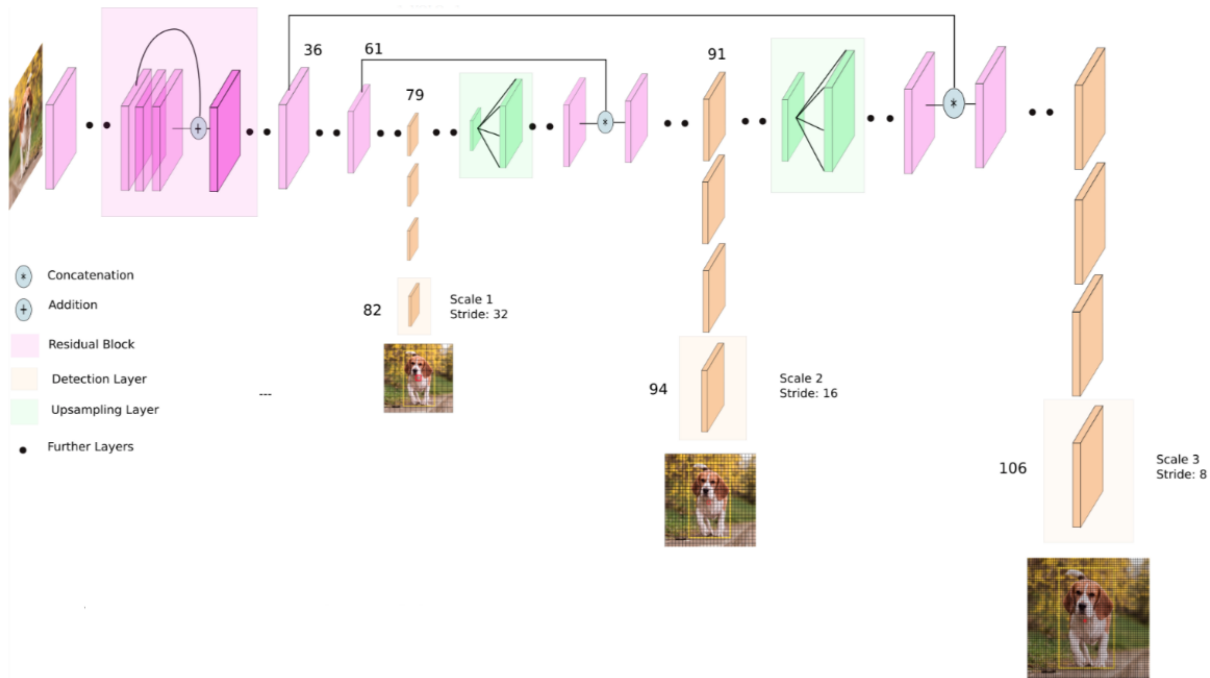

Obrázek 30: YOLO Bounding Boxes [6]

YOLO je neuronová síť, která sjednocuje detekci a klasifikaci objektů do jedné end-to-end sítě a této architektury se nazývá **DarkNet**. **DarkNet** má různé verze, například YOLOv2 používá **darknet-19**, to znamená, že tato architektura má v síti 19 konvolučních vrstev. YOLOv3 používá **darknet-53**, což znamená, že se skládá z 53 konvolučních vrstev. [8]

Jak vidíte na obr.31, Vstupní obraz prochází **extraktorem vlastnosti DarkNet-53** a poté je obraz převzorkován sítí (Upsampling), dokud vrstva 79. Síť se větví a pokračuje v převzorkování obrazu, dokud neprovede svou první předpověď ve vrstvě 82. Tato detekce se provádí na mřížce o velikosti $13 \times 13$, která odpovídá za detekci *velkých objektů*.[15] [8]

Dále je mapa prvků z vrstvy 79 převzorkována 2x na rozměry $26 \times 26$ a zřetězena s mapou prvků z vrstvy 61. Druhá detekce je provedena vrstvou 94 na mřížce $26 \times 26$, která je zodpovědná za detekci *středních objektů*. [15] [8]

Dále je mapa prvků z vrstvy 91 je podrobena několika převzorkováním konvolučních vrstev, než je hloubka zřetězena mapou prvků z vrstvy 36. Třetí předpověď je vytvořena vrstvou 106 na měřítku mřížky $52 \times 52$, který je znovu použitelný pro detekci *malých objektů*. [15] [8]



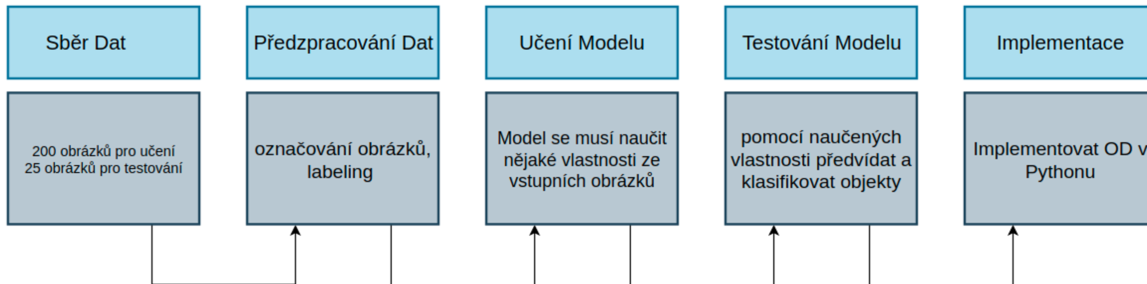

Obrázek 31: YOLOv3 arch [8]

[3] [7] [8]

## 3.5 Závěr Teoretické části

V této teoretické části práce byla popsána cesta detekce objektů jako celek, od základních principů, jako je zpracování obrazu a strojové učení, přes metody hlubokého učení, jako jsou MLP a CNN, až po nejmodernější metodu Detekce objektů, jako je YOLO.

# 4 Praktická čast

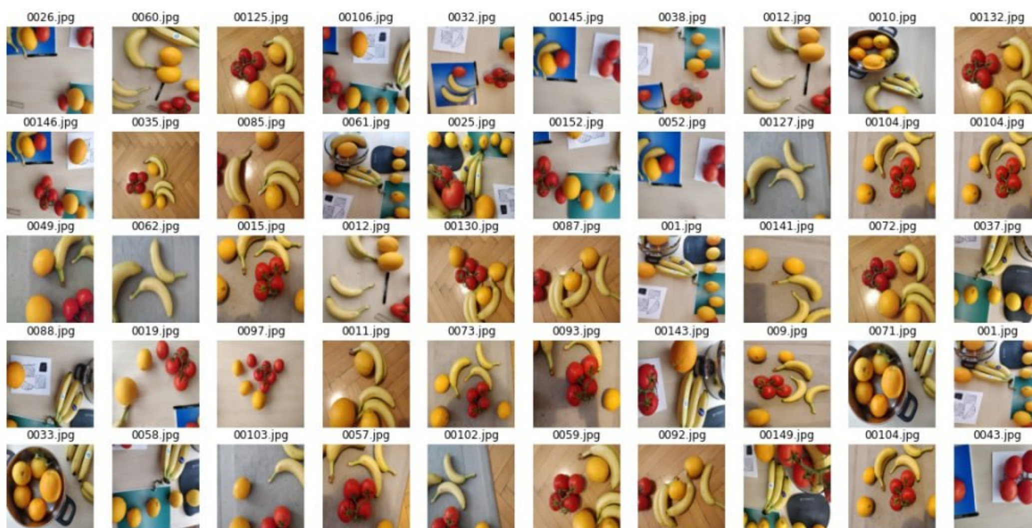



Obrázek 32: Struktura Praktické Části

Cílem této práce je navrhnout Object detector pomoci metody YOLO a použitím Darknet Frameworku a pak vytvořit implementace v programovacím jazyce Python, které rozpozná tři typu objektů, banán, rajče a pomeranč.

## 4.1 Sběr Dat

Jak je uvedeno na obr. 32, nejprve začnu sběrem dat.



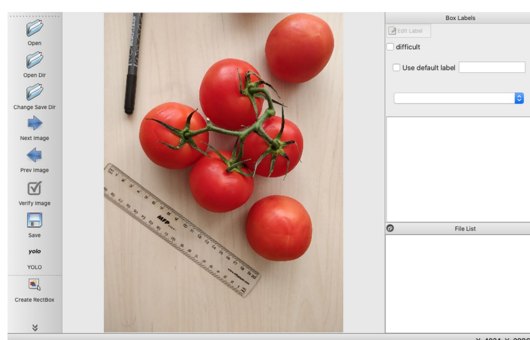

Obrázek 33: Datová Sada Obrázků

Jak již bylo dříve definováno, musím detekovat 3 typy objektů, to jsou rajče, pomeranč a banán. Vytvoření datové sady bylo docela jednoduché, právě jsem pomocí svého smartphonu

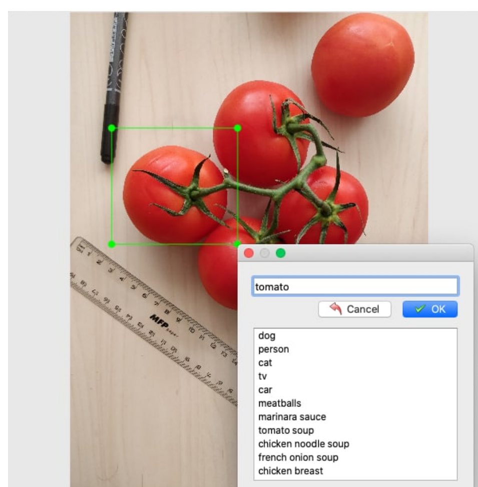
pořídil asi 200 snímků požadovaných objektů, v různých prostředích a s různými překážkami na obrázcích. Zde jsou spousta obrázků se skvělým rozlišením a kvalitou a obrázky se spoustou šumu nebo jiných faktorů. Vzorek 50 náhodných obrázků z datové sady je vynesen na obr.33

## 4.2 Předzpracování Dat

S pomocí open source nástroje, který se nazývá **LabelImg**, vytvořím **label** pro každý obrázek v datové sadě pro učení. Jak vidíte na obr.34 a obr.35. Nejprve obrázek načtu a poté vytvořím **label**. Je důležité si uvědomit, že musím zvolit možnost vytváření **label** speciálně pro algoritmus YOLO.




Obrázek 34: Rozhraní programu LabelImg




Obrázek 35: Vytvoření labelů objektu v LabelImg

Po uložení bude vytvořen *.txt* soubor. Tento soubor obsahuje pět proměnných, první proměnna je označení třídy, v tomto příkladu je již 14 tříd, a když jsem vytvořil novou třídu „Tomato“, objeví se třída 15. Další 4 proměnné jsou body pro **bounding box**, které bude algoritmus YOLO používat při učení. Důležité si uvědomit, že pro každý obrázek v datové sadě je vyžadován vyhrazený soubor *txt*. V tomto příkladu je 15 tříd, ale v mém detektoru budou pouze 3 třídy, protože budu učit klasifikátor pouze pro rajče, pomeranč, banán.

Obrázek 36: txt soubor pro obrázek

## 4.3 Učení Modelu

**yolov3-tiny.cfg** je konfigurační soubor a soubor **obj.names** je seznam názvů tříd, ve kterém rozhodujeme, jak bude probíhat učení našeho klasifikátoru. Níže ukážu některé změny v **yolov3-tiny.cfg**:

- batch = 16
- subdivision = 2
- width = 416, height = 416
- steps = 4800, 5400
- třídy (classes) v každé [yolo] vrstvě se rovnají 3, protože mám pouze 3 třídy
- filtr v každé [yolo] vrstvě se rovná 24, protože filtr = (třídy + 5) * 3

Rozhodl jsem se učit váhy pomocí konfigurace **yolov3-tiny**, kvůli problému s nízkou pamětí GPU (NVIDIA GTX 960 2gb), mám pouze 2gb video paměť, když je pro trénink na YOLOv3-320, 416, 608 vyžadováno alespoň 3gb.

Další kroky jsou:

- Stáhnout výchozí váhy yolov3-tiny
- Získejte **pre-trained weights**, yolov3-tiny.conv.15 pomocí příkazu:

  ```
  ./darknet partial cfg/yolov3-tiny.cfg
  yolov3-tiny.weights yolov3-tiny.conv.15 15
  ```

  Tyto váhy hsou velmi užitečné při provádění detekce objektů na nových třídách.
- Vytvoření vlastního modelu yolov3-tiny-obj.cfg na základě cfg yolov3-tiny-obj.cfg
- Pak se to učí:

  ```
  ./darknet detector train data/obj.data
  yolov3-tiny-obj.cfg yolov3-tiny.conv.15 -map
  ```
- Ukončit učení, když **average loss**, neboli průměrná ztráta bude menší než 1 (v mém případě 0,863960).

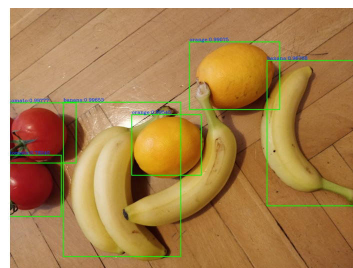
## 4.4 Testování Modelu

Ke spuštění tohoto programu a získání našich výsledků bych měl použít následující příkaz:

```
python app.py --img=test/<image.jpg> --out=output/<result.jpg>
```

Jako vstup použiji následující obrázek:

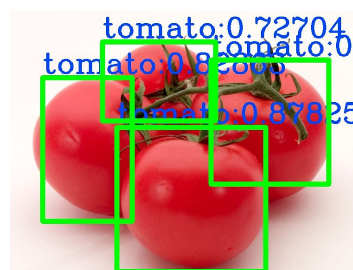
Obrázek 37: Testovací obrázek



Obrázek 38: Předpověď

Výsledkem je, že na testovacím obrázku můžete vidět, že vstupní obraz může nejprve pokračovat ve vstupním toku a za druhé nám program může poskytnout požadovaný výstupní obraz s ohraničujícími rámečky, třídami skóre spolehlivosti. Ale tento testovací obrázek vypadá jako obrázek z hlavní datové sady. Takže musím zkusit otestovat detektor objektů na náhodném obrázku z internetu, a to následovně:

Obrázek 39: Náhodný obrázek z internetu `https://upload.wikimedia.org/wikipedia/commons/thumb/8/89/Tomato_je.jpg/1280px-Tomato_je.jpg`




Obrázek 40: Předpověď

Výsledky testování nám ukazují, že i když máme náhodný obraz z internetu na vstupu, naučený detektor objektů, stále může produkovat dobrý výstup a správně předvídat třídy objektů a jejich skóre spolehlivosti.

A je to velmi úspěšný model, protože dokáže najít všechna rajčata na obrázku.

## 4.5 Implementace

```
import cv2
import argparse
import numpy as np

argparser = argparse.ArgumentParser()

argparser.add_argument('--img', type=str)
argparser.add_argument('--out', type=str)

args = argparser.parse_args()

confidence, threshold = 0.4, 0.3

labelPath = './obj.names'
labels = open(labelPath).read().strip().split('\n')
weightsPath = './yolov3-tiny.weights'
configPath = './yolov3-tiny.cfg'
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
```

Nejprve importuju požadované moduly a knihovny. Pak deklarujeme proměnné pro:

1. složku s obrázky
2. složku s třídami
3. váhy **yolov3-tiny.weights**
4. konfigurační soubor **yolov3-tiny.cfg**

Proměnná *net* je velmi užitečná funkce knihovny OpenCV, která načítá model serializované sítě z prostředí DarkNet a která je užitečná pro testování Neural Network. [16] Proměnná *confidence* představuje minimální pravděpodobnost filtrování slabých detekcí (weak detections) a proměnná *threshold* je maximální limit potlačení (non-maximum suppression). [3]

```python
layer_names = net.getLayerNames()
yolo_layers = []
for i in net.getUnconnectedOutLayers():
    yolo_layers.append(layer_names[i[0] - 1])

def draw_bb(img, boxes, confidences, classids, idxs, labels):
    if len(idxs):
        for i in idxs.flatten():
            x, y = boxes[i][0], boxes[i][1]
            w, h = boxes[i][2], boxes[i][3]
            cv2.rectangle(img, (x,y), (x+w, y+h), (0, 255, 0), 10)
            text = "{}:{:2.5f}".format(
                labels[classids[i]], confidences[i])
            cv2.putText(img, text,
                (x, y-10), cv2.FONT_HERSHEY_COMPLEX,
                2, (255, 70, 0), 3)
    return img

    boxes, confidences, class_ids = [], [], []
```

Dalším kódem je implementace funkce, která kreslí ohraničující rámečky (Boundig Box),a ještě více, tato funkce také přiřazuje skóre spolehlivosti a ID tříd k těmto rámečkům.

```python
def predict(net, layer_names, height, width, img, labels):
    blob = cv2.dnn.blobFromImage(img,
                1/255.0,
                (320, 320),
                swapRB=True,
                crop=False)
    net.setInput(blob)
    layerOutputs = net.forward(layer_names)
```

```
boxes, confidences, class_ids = [], [], []

for out in layerOutputs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        detect_confidence = scores[class_id]

        if detect_confidence > confidence:
            box = detection[0:4] * np.array(
                    [width, height, width, height]
                    )
            centerX, centerY, box_width, box_height = box.astype('int')

            x = int(centerX - (box_width / 2))
            y = int(centerY - (box_height / 2))

            boxes.append([x, y, int(box_width), int(box_height)])
            confidences.append(float(detect_confidence))
            class_ids.append(class_id)

idxs = cv2.dnn.NMSBoxes(boxes, confidences, confidence, threshold)

img = draw_bb(img, boxes, confidences, class_ids, idxs, labels)

return img, boxes, confidences, class_ids, idxs
```

Dalším kódem je implementace funkce, která interpretuje predikci a vrací šest proměnných, které plynou ve výstupu programu. Toto je hlavní funkce programu a používá předchozí funkci jako nástroj pro kreslení rámečků.

```
if args.img:
    img = cv2.imread(args.img)
    height, width = img.shape[:2]
    img, _, _, _, _ = predict(net,
                              yolo_layers,
                              height,
                              width,
                              img,
                              labels
                              )

    if args.out:
        cv2.imwrite(args.out, img)
    else:
        img = cv2.resize(img, (800, 800))
        cv2.imshow('Prediction', img)
        cv2.waitKey(0)
```

Některá část této implementace byla půjčena a upravena podle požadavků mého vlastního detektoru objektů odsud [3] a [4].

## 4.6 Výsledek

Takže mohu usuzovat, že detektor byl naučen s opravdu malým úsilím, pokud jde o množství obrázků v datové sadě a výpočetní výkon, protože jak jsem uvedl dříve, používál jsem pouze svou grafickou kartu, která se nazývá NVIDIA GTX 960 2gb.

Model se naučil od nuly a je dost dobrý na to, aby umožňoval jednoduchou detekci z toku vstupních dat, jak je znázorněno v části Testování Modelu.

# 5 Závěr

Na závěr musím říci, že v Detekci objektů stále existuje spousta zajímavých metod a YOLO jen jedna z nich, ale i tak to bylo velmi zajímavé pracovat.

Tato práce byla zaměřena na realizaci Object detectoru pomoci metody YOLO a použitím Darknet Frameworku a implementace tohoto detektoru v programovacím jazyce Python, které rozpozná tři typu objektů, banán, rajče a pomeranč, program uspěšně detekuje tyto objekty s velkou spolehlivosti.

V teoretické části této práce byly představeny a popsány základy počítačového vidění. Udělal jsem to záměrně, protože pro mě je velmi důležité vidět základní pojmy za novými metodami v technologiích, takže jsem začal od základu, abych lépe porozuměl následujícím metodám. Začal jsem zpracováním obrazu, kde byl představen a popsán koncept digitálních obrázků, poté následovaly koncepty strojového učení a hlubokého učení. Počítačové vidění je spojení těchto věcí, které vytváří další větev umělé inteligence. Na konci teoretické části byl na YOLO představen a popsán koncept detekce objektů.

V praktické části této práce byl celý proces popsán ne všemi, ale většinou podrobnostmi. Protože stále existuje spousta věcí ke studiu a výzkumu. Téma je obrovské, ale je velmi vzrušující.

K detekci objektů využívá počítačové vidění celá řada průmyslových odvětví. Používá se pro získávání obrazů, sledování, zabezpečení, kontrolu strojů a automatizaci systémů vozidel atd. Existují nekonečné možnosti použití tak výkonných metod detekce objektů, jako je YOLO.

# Seznam použitých zdrojů

[1] NIELSEN Michael A. *Neural Networks and Deep Learning [online]*. Dostupné z: http://neuralnetworksanddeeplearning.com/chap1.html. 2015.

[2] ROSEBROCK Adrian. *Intersection over Union (IoU) for object detection [online]*. Dostupné z: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. 2016.

[3] ROSEBROCK Adrian. *YOLO object detection with OpenCV*. Dostupné z: https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/. 2018.

[4] BOCHKOVSKIY Alexey. *Yolo v4, v3 and v2 for Windows and Linux [online]*. Dostupné z: https://github.com/AlexeyAB/darknet/.

[5] REDMON Joseph a ALI Farhadi. *The Ancient Secrets of Computer Vision [online]*. Dostupné z: https://pjreddie.com/courses/computer-vision/.

[6] REDMON Joseph a ALI Farhadi. *YOLOv3: An Incremental Improvement*. Dostupné z: https://pjreddie.com/media/files/papers/YOLOv3.pdf. 2018.

[7] KATHURIA Ayoosh. "How to implement a YOLO (v3) object detector from scratch in PyTorch". In: (2018). Dostupné z: https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/.

[8] KATHURIA Ayoosh. "What's new in yolo v3". In: (2018). Dostupné z: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b.

[9] GOODFELLOW Ian a BENGIO Yoshua a COURVILLE Aaron. *Deep Learning [online]*. Dostupné z: http://www.deeplearningbook.org. MIT Press, 2016.

[10] REDMON Joseph a DIVVALA Santosh a ROSS Girshick a ALI Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. Dostupné z: https://pjreddie.com/media/files/papers/yolo_1.pdf. 2016.

[11] LI Fei-Fei a KARPATHY Andrej a JOHNSON Justin. *CS231n: Convolutional Neural Networks for Visual Recognition [online]*. Dostupné z: https://cs231n.github.io/convolutional-networks/.

[12] ZHANG Aston a LIPTON C. Zachary a LI Mu a SMOLA Alexander J. *Dive into Deep Learning [online]*. Dostupné z: `https://d2l.ai`. 2020.

[13] MITCHELL Tom M. *Machine Learning*. [cit. 2021-03-01]. New York: McGraw-Hill, 1997.

[14] DASIOPOULOU Stamatia a MEZARIS Vasileios a KOMPATSIARIS Ioannis a PAPASTATHIS V-K a STRINTZIS Michael G. "Knowledge-assisted semantic video object detection". In: *IEEE Transactions on Circuits and Systems for Video Technology* 15.10 (2005), s. 1210–1224.

[15] ELGENDY Mohamed. *Deep Learning for Vision Systems*. [cit. 2021-02-09]. Manning Publications, 2020.

[16] *OPENCV PYTHON TUTORIALS*. Dostupné z: `https://docs.opencv.org/master/d9/df8/tutorial_root.html`. OpenCV. 2021.

[17] SIMEONE Osvaldo. *A Brief Introduction to Machine Learning for Engineers*. Dostupné z: `https://arxiv.org/pdf/1808.02342.pdf`. 2018.

[18] SZELISKI Richard. *Computer Vision: Algorithms and Applications*. 2. vyd. Springer, 2021. ISBN: 978-1848829343.

[19] PODLOZHNYUK Victor. "*Image convolution with CUDA*". In: *NVIDIA Corporation white paper, June* 2097.3 (2007). Dostupné z: `https://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/convolutionSeparable/doc/convolutionSeparable.pdf`.