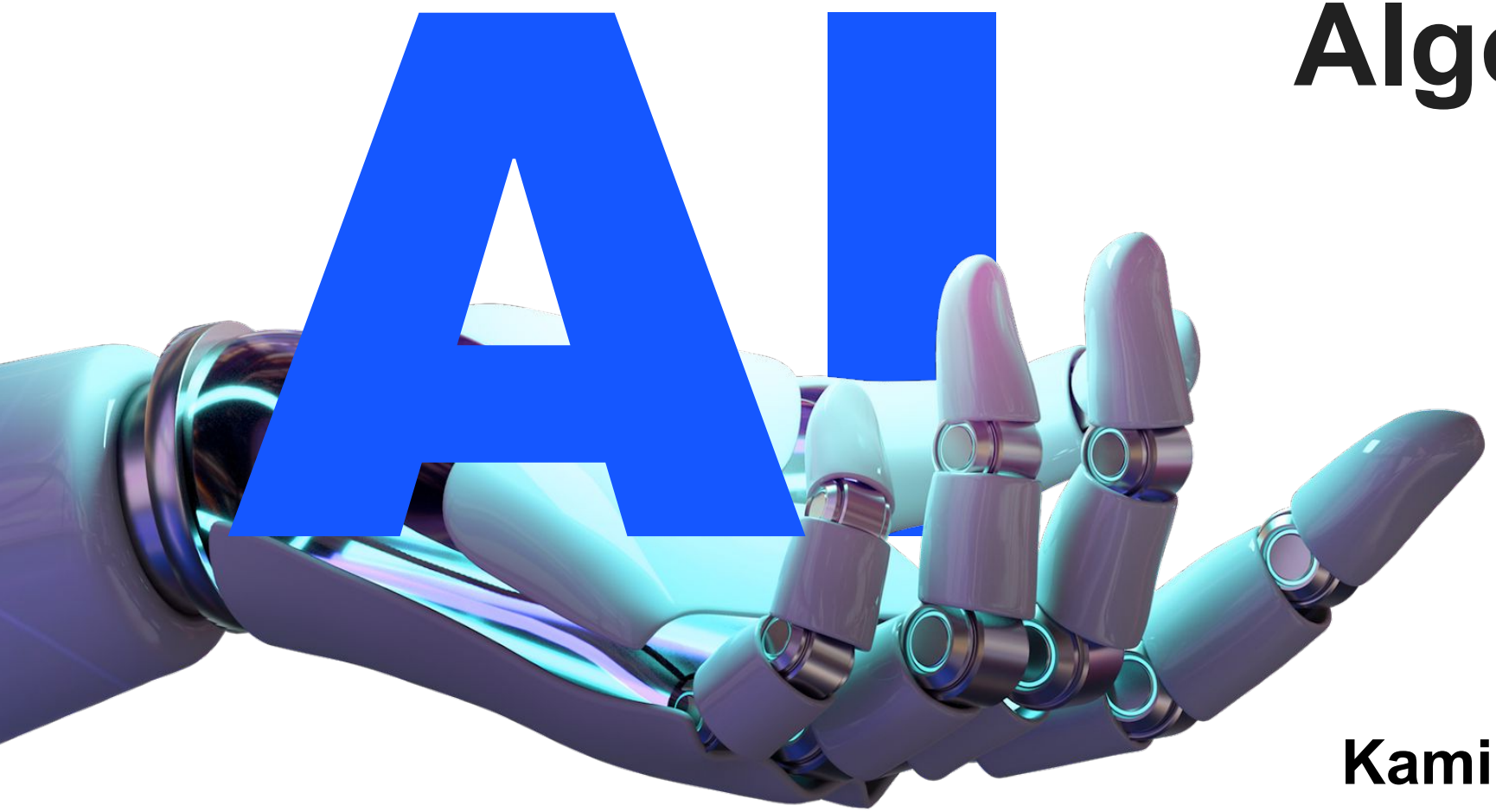
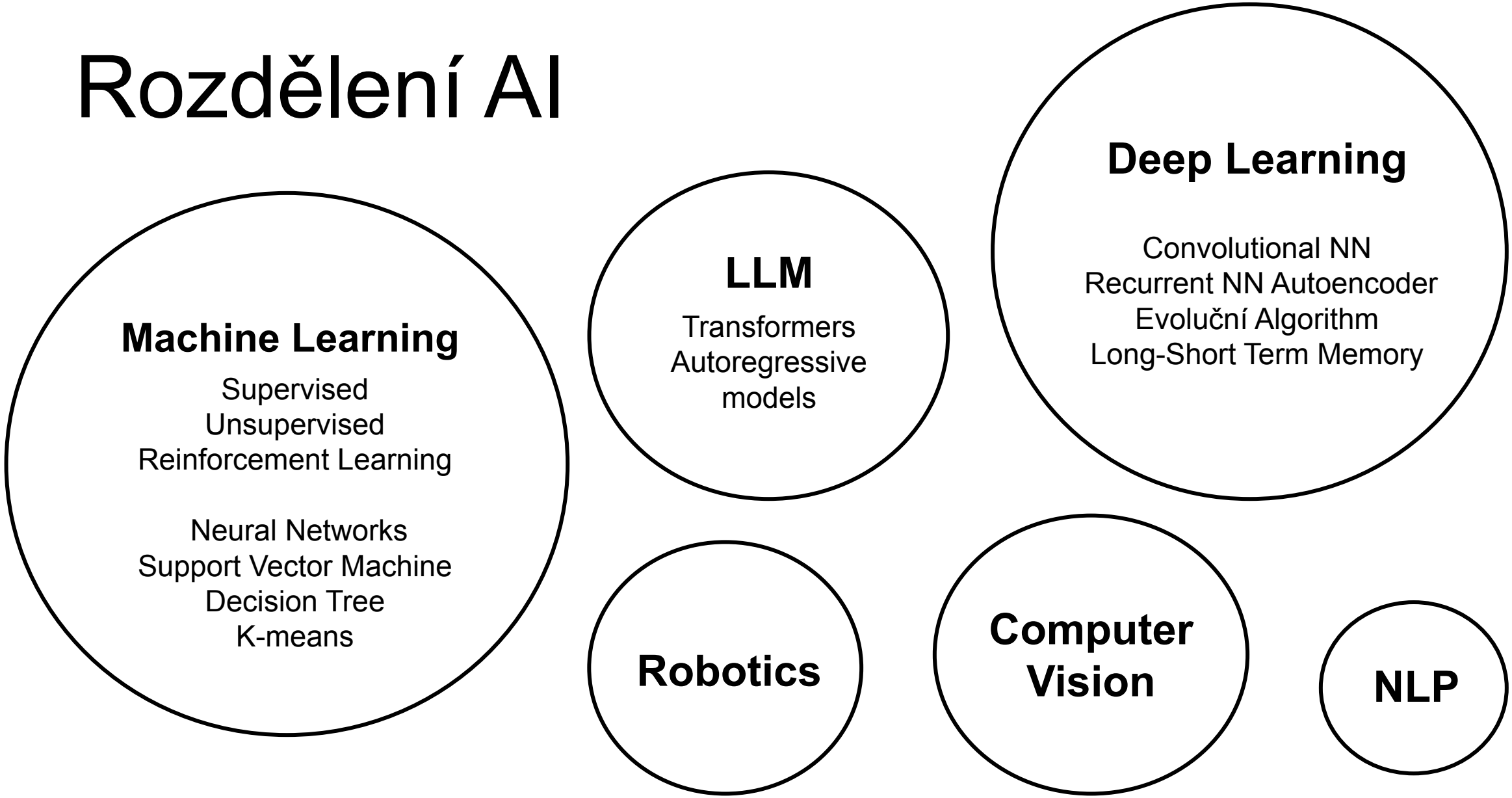
Pokročilé Algoritmy AI



Kamila Lepkova

Rozdělení AI



Machine Learning (Strojové učení)

- Odvětví umělé inteligence (AI) a informatiky
- Využití dat a algoritmů k napodobování způsobu, jakým se lidé učí, postupně zlepšuje svoji přesnost.
- Statistické metody
- Strojové učení je při učení více závislé na lidském zásahu

Typy Machine Learning

- **Supervised Machine Learning**: používá označené datové sady k trénování (Logistická regrese, Random Forest, Support Vector Machine). Algoritmy se učí vztah mezi vstupy a výstupy
- **Unsupervised Machine Learning**: analýza a seskupování neoznačených datových sad (Analýza hlavních komponent (PCA), Singulární rozklad (SVD), k-means). Algoritmy se snaží najít skryté struktury a vzorce
- **Semi-supervised Machine Learning:** během tréninku se použije menší označená datová sada, která vede ke klasifikaci neoznačené datové sady (Semi-supervised support vector machine, klastrování)
- **Reinforcement Learning:** algoritmus se učí optimálním akcím na základě zpětné vazby z prostředí. Interaguje s dynamickým prostředím a učí se ze zkušeností.

Aplikace Supervised Machine Learning

- Bioinformatika (struktura duhovky, otisky prstů)
- Rozpoznání řeči
- Diagnostika nemocí
  - Algoritmy pro detekci nádorů
  - Klasifikace srdečních arytmií
- Predikce výsledků léčby
- Personalizovaná medicína
  - Doporučující personalizované léčebné plány

Aplikace Unsupervised Machine Learning

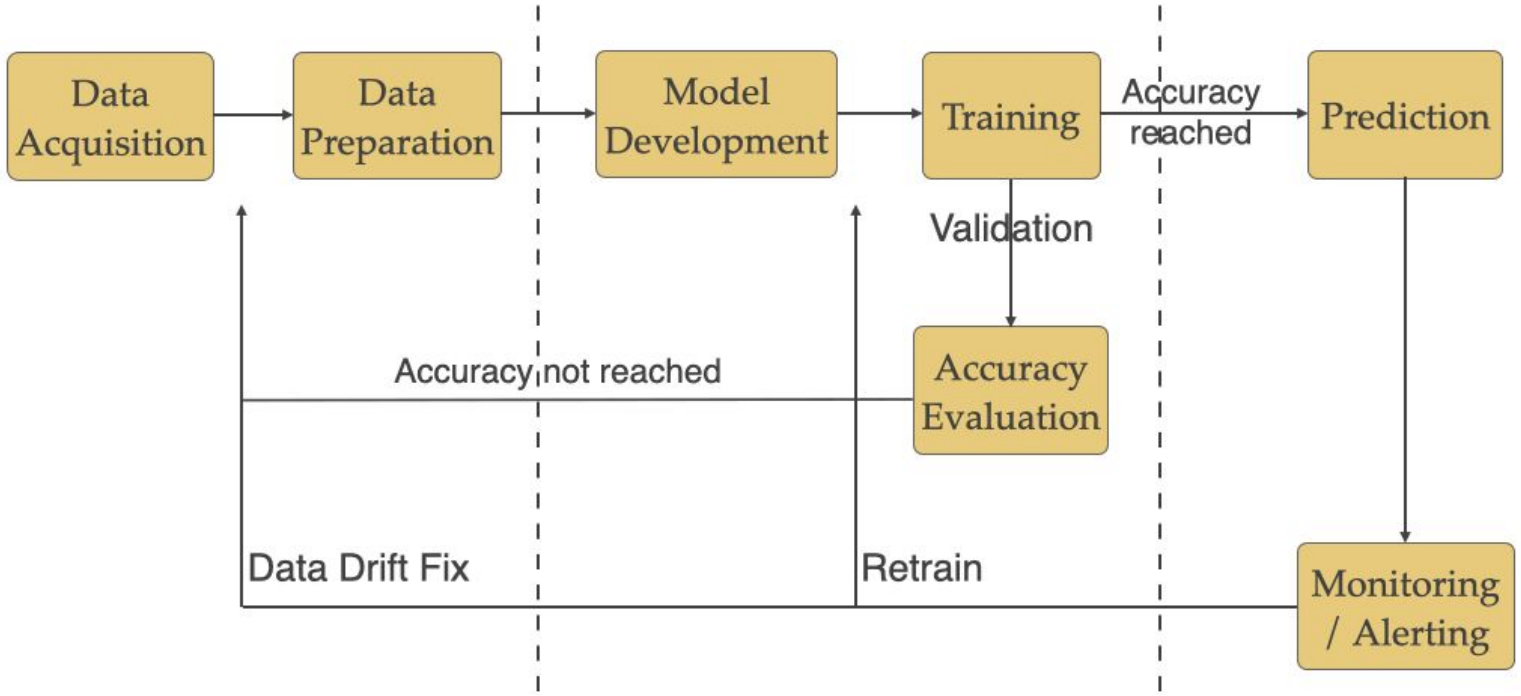
- Segmentace obrazů
  - oddělení tumoru od fyziologické tkáně
- Klasifikace pacientů s podobnými symptomy
- Identifikace vzorců v genomických datech
- Clustering (K-means)
- Vizualizace
- Komprese (PCA)

Aplikace Semi-supervised Machine Learning

- Analýza hlasu
- Rozšíření diagnostických modelů
- Automatická anotace datasetu
- Klasifikace proteinové sekvence
- analýza elektronických zdravotnických záznamů

Jak funguje Machine Learning

1. Rozhodovací proces - odhad predikce / klasifikace
2. Chybová funkce - vyhodnocení predikce / klasifikace
3. Proces optimalizace modelu - upravení váh, aktualizace váh

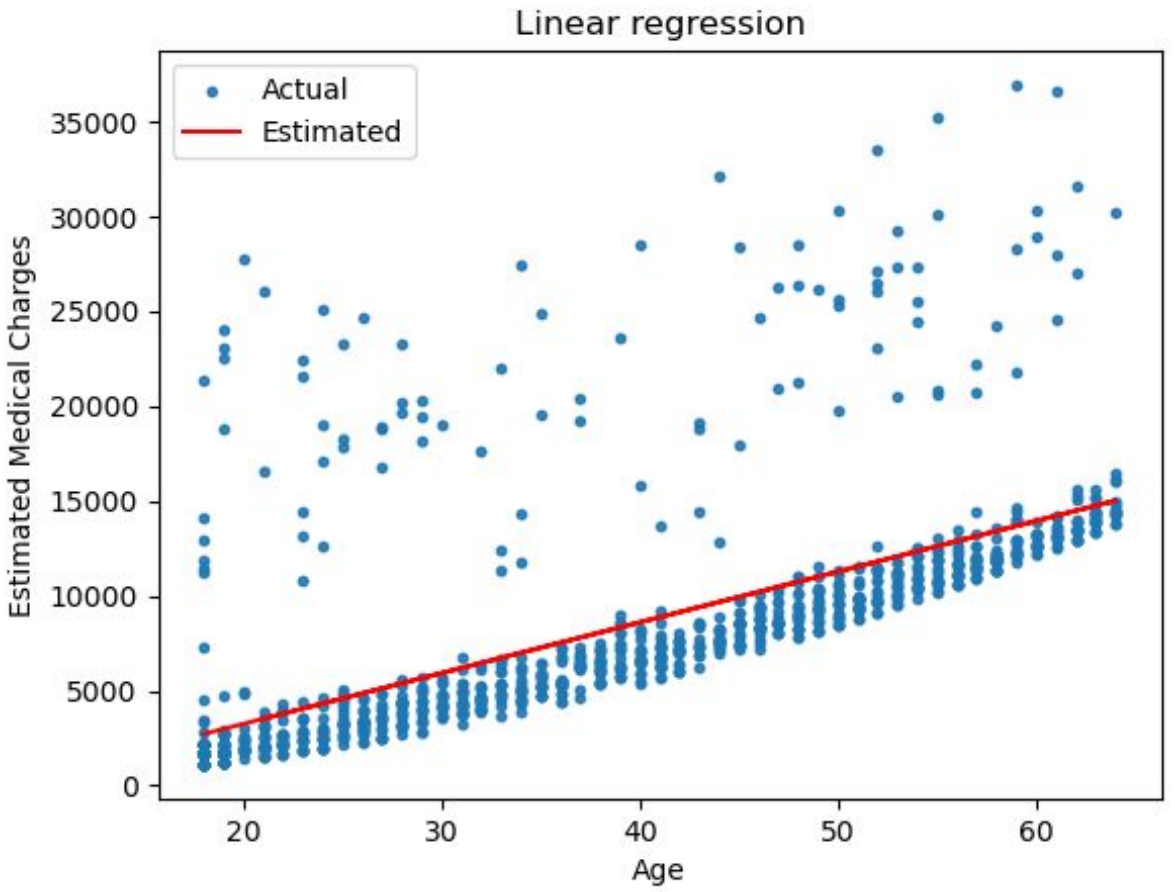


Algoritmy Strojového Učení

- Lineární regrese
- Logistická regrese
- Metoda nejbližšího souseda - Nearest neighbour algorithm
- Metoda podpůrných vektorů - Support Vector Machine

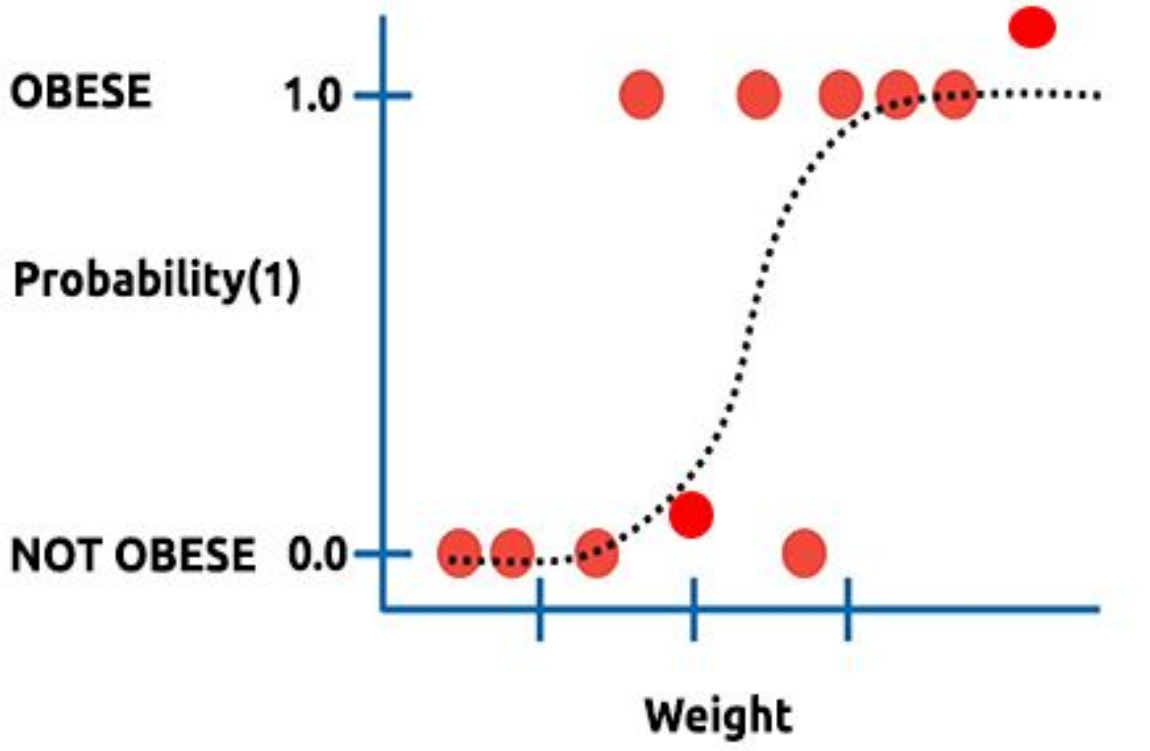
Regrese - Lineární regrese

- Dávkování léků na základě biologických informací
- Predikce růstu dětí
- Analýza krevního tlaku



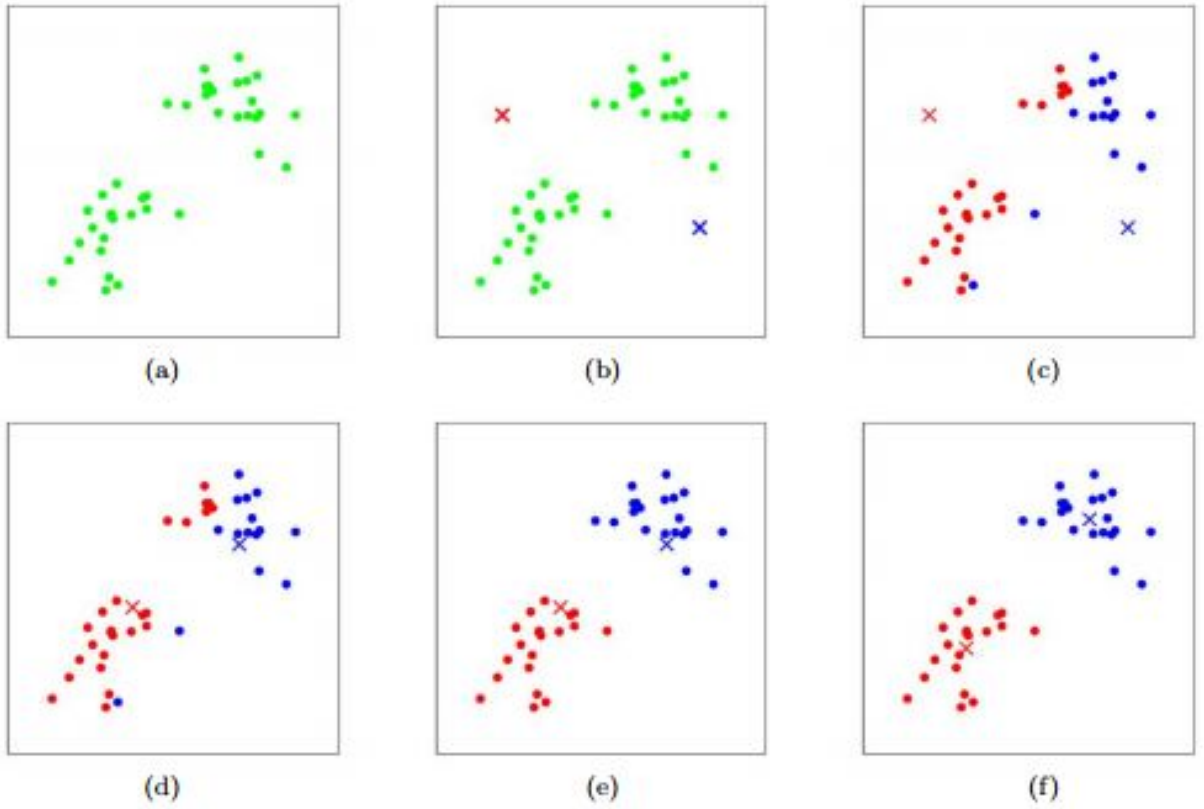
Regrese - Logistická regrese

- Predikce výskytu nemocí (diabetes) na základě klinických parametrů pacienta (BMI, věk,...)
- Rozhodování o nutnosti hospitalizace pacientů na základě symptomů a historie onemocnění.

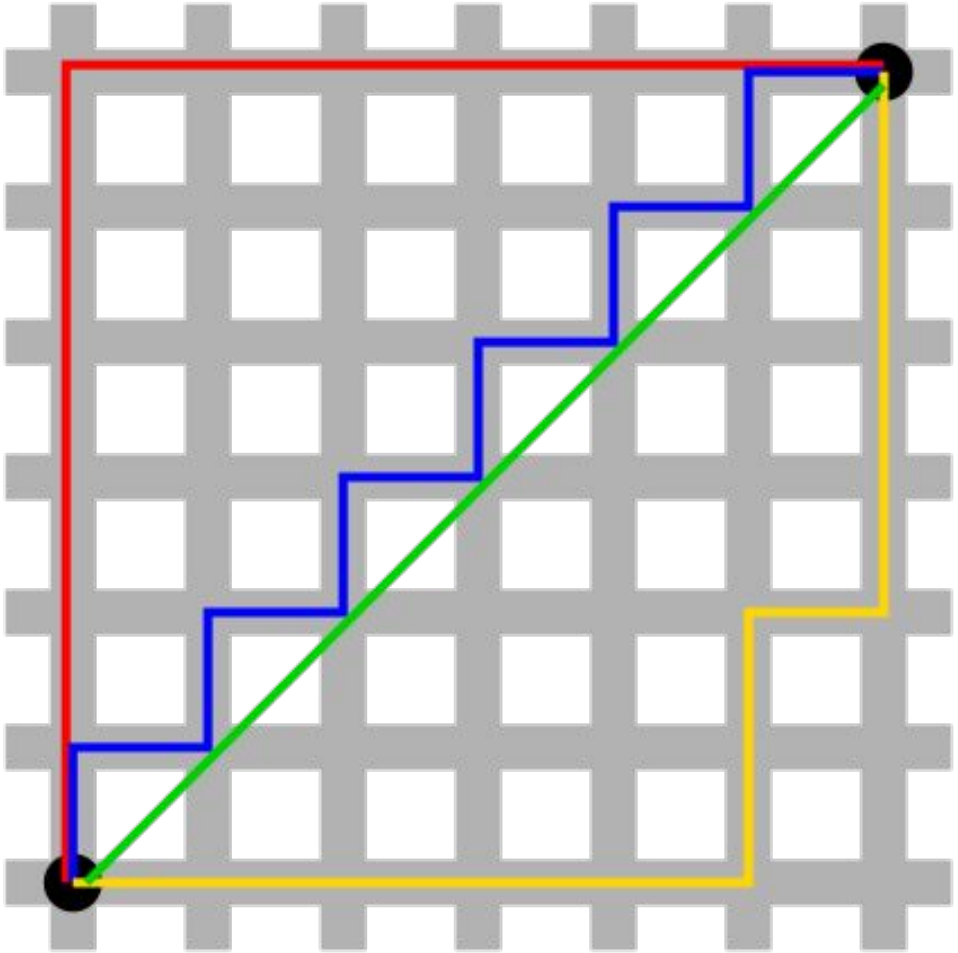


Klasifikace – Metoda Nejbližšího Souseda (Nearest neighbour algorithm)

- Klasifikace různých typů rakoviny
- Predikce srdečních onemocnění na základě EKG záznamů, úroveň cholesterolu
- Detekce diabetu na základě hladiny glukózy, krevního tlaku, BMI,...

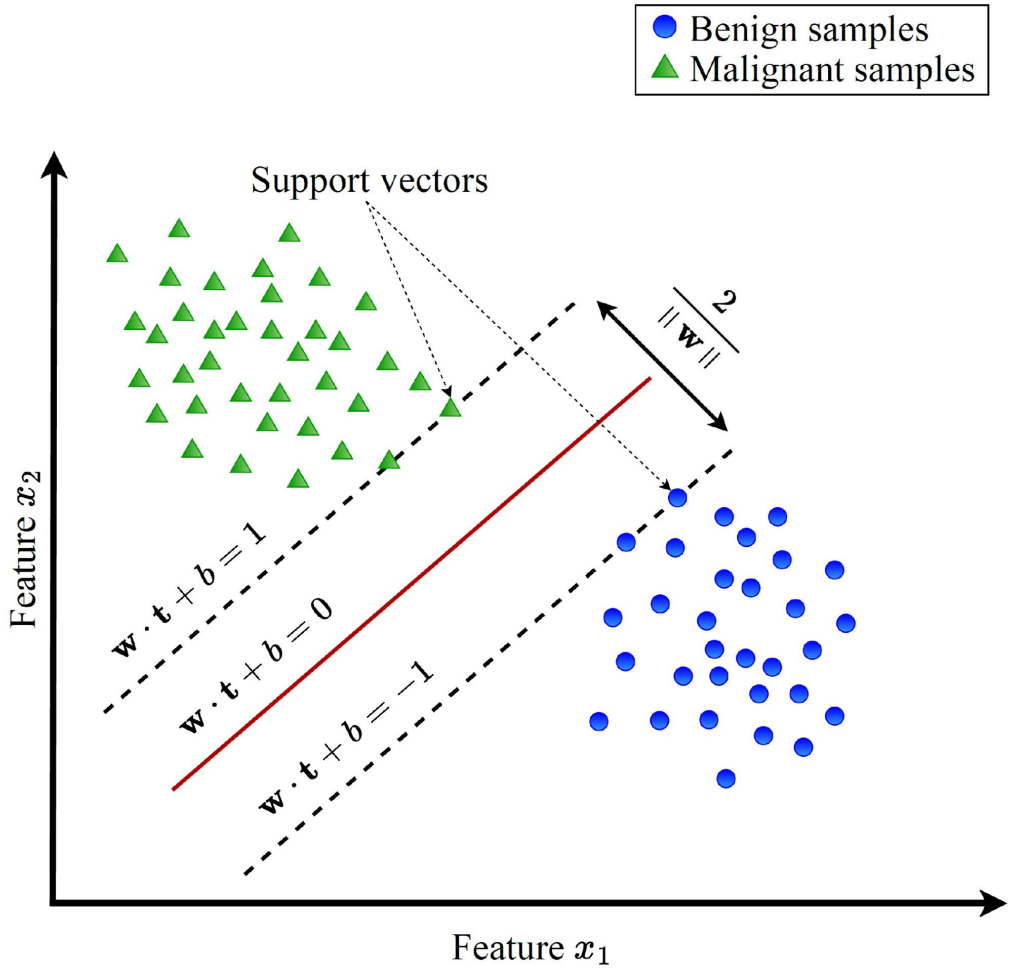


Vzdálenost

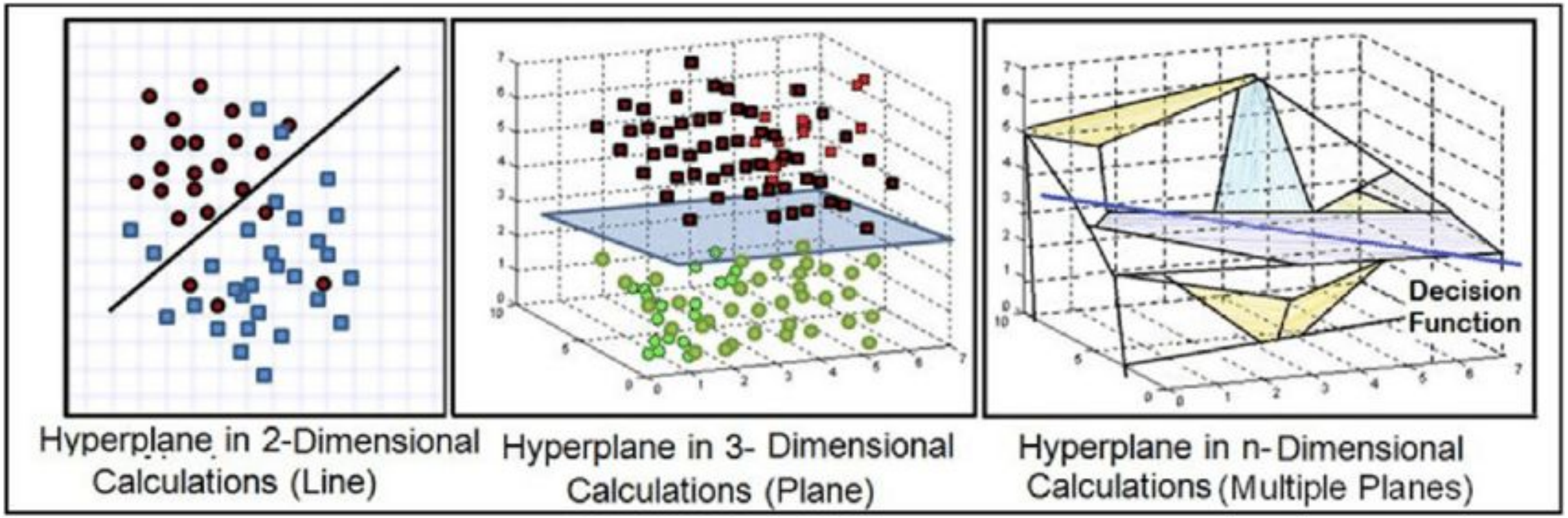


Klasifikace – Metoda podpůrných vektorů (Support Vector Machine)

- Klasifikace biomarkerů do patologické nebo fyziologické třídy (epilepsie)
- Detekce abnormalit v srdečním rytmu - systoly, arytmie



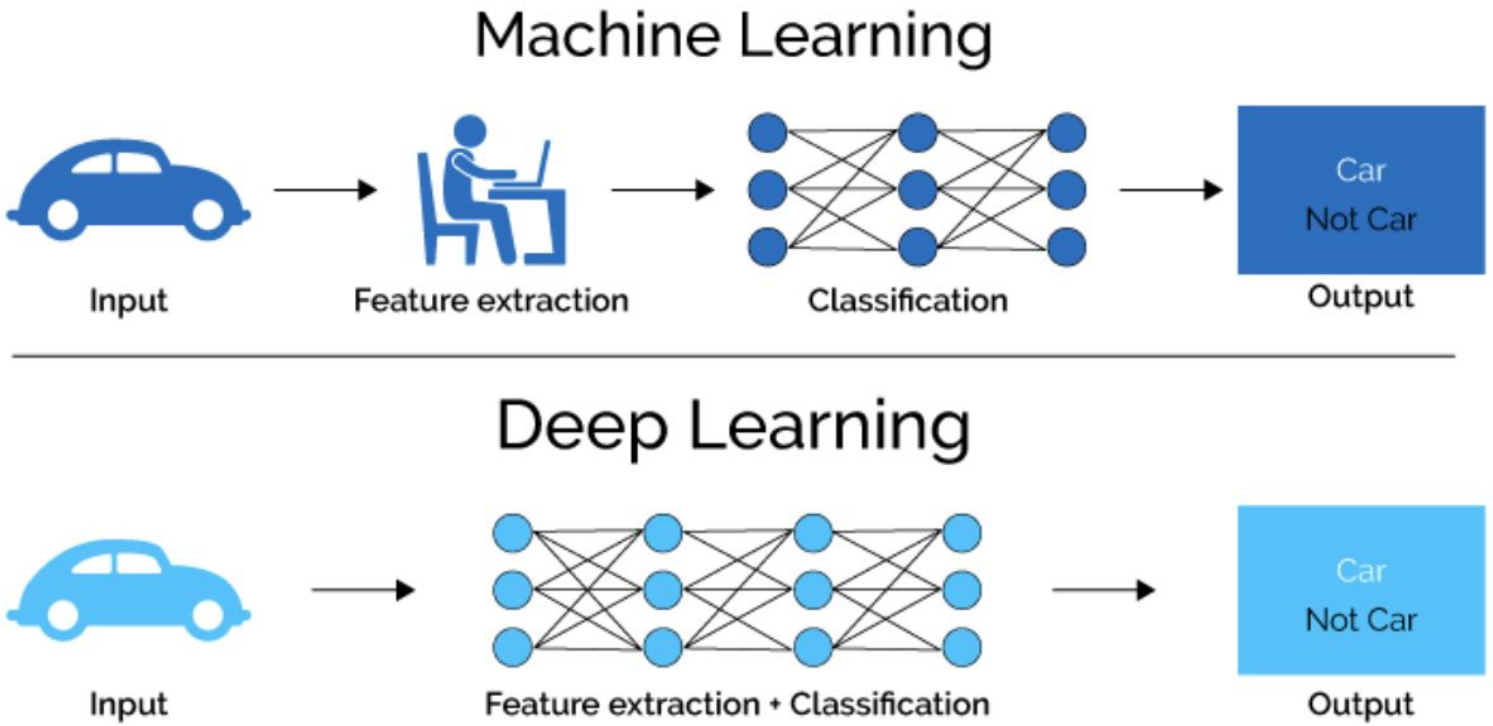
Klasifikace – Metoda podpůrných vektorů (Support Vector Machine)



Deep Learning (Hloubkové učení)

Deep Learning (Hloubkové učení)

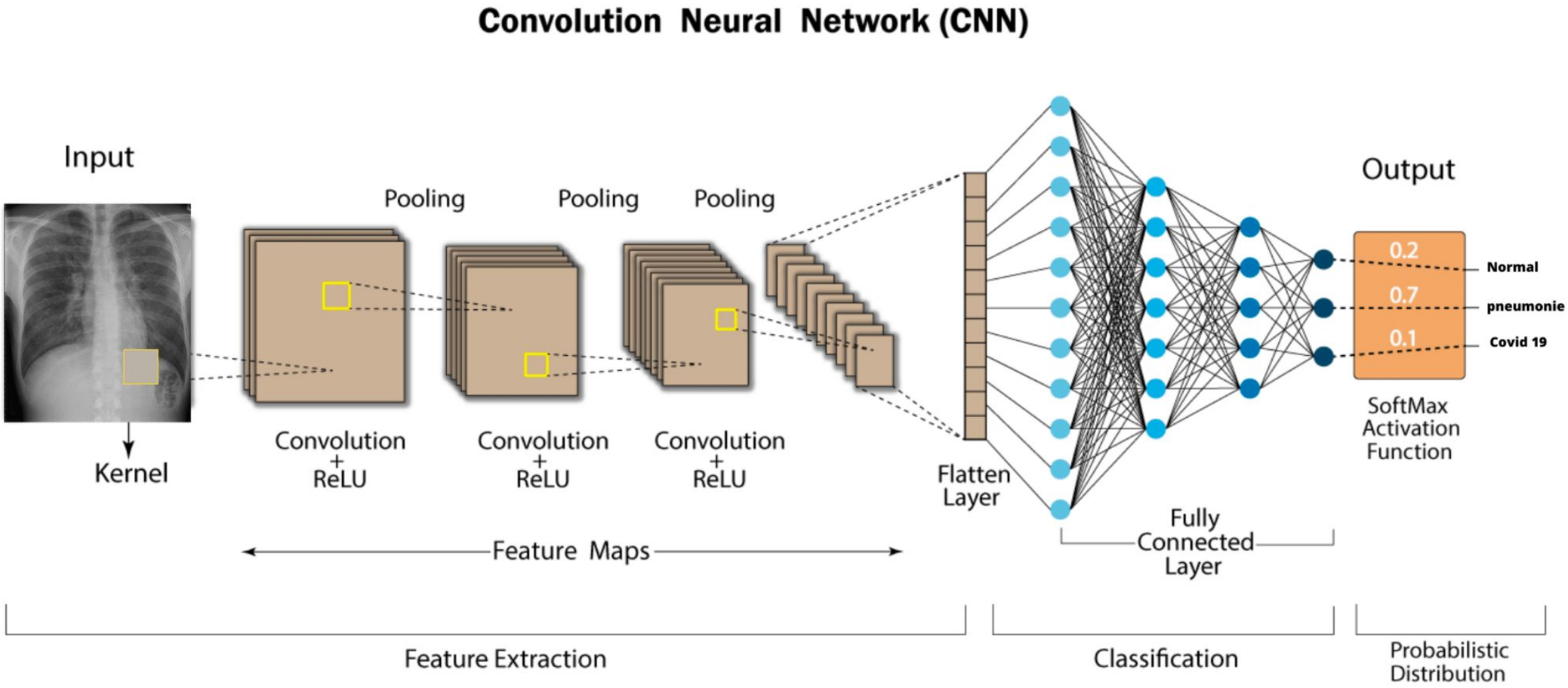
- Podsekce strojového učení
- Automatizuje extrakci příznaků



Konvoluční neuronové sítě - CNN

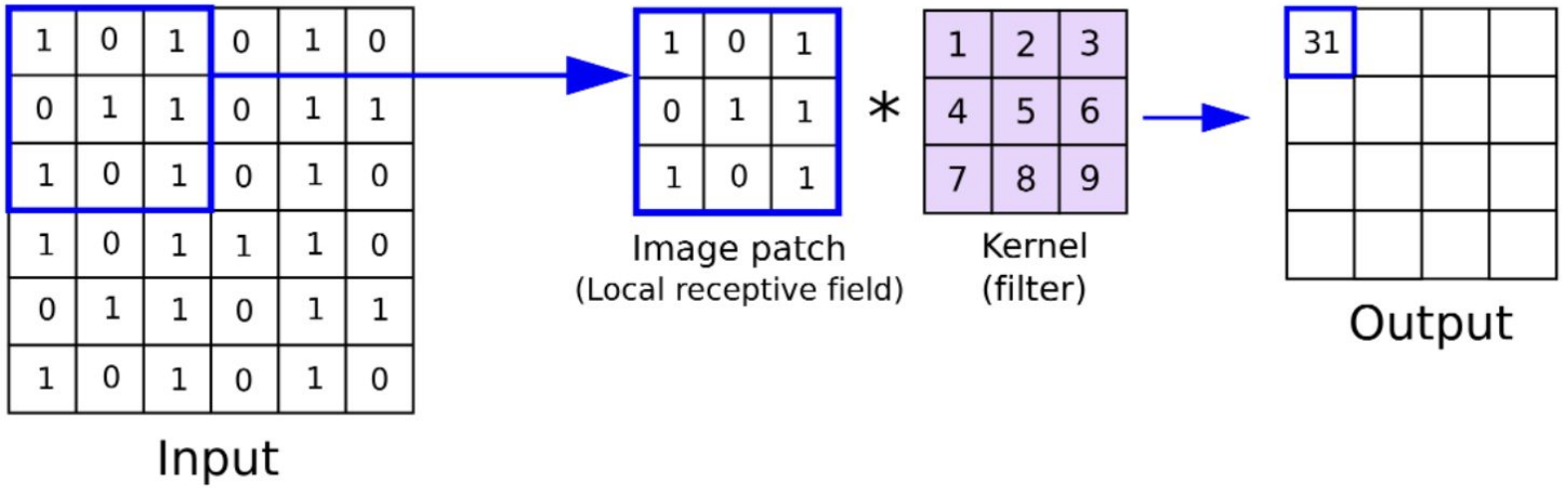
- Konvoluční vrstvy,
- Vrstvy s podvzorkováním = mean – max pooling
- Fully connected Layers
- Primárním účelem je extrahovat vlastnosti obrázku a na základě těchto vlastností zařadit obrázek do správných skupiny

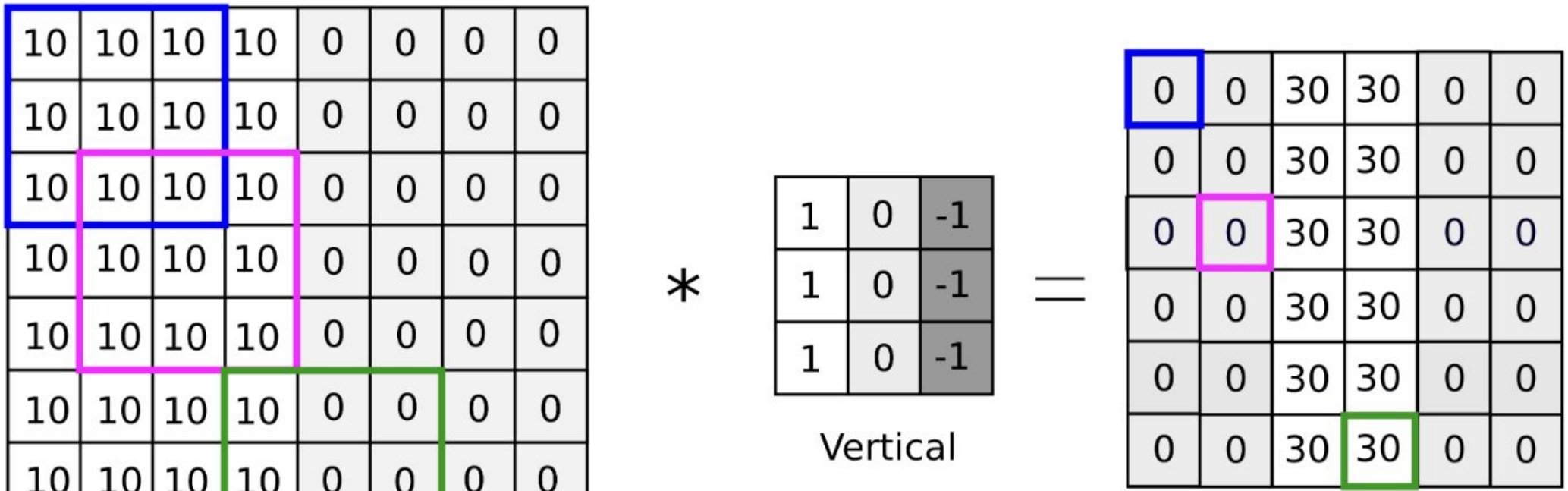
Konvoluční neuronové sítě - CNN



Konvoluční vrstva

- Extrakce prvků se provádí pomocí konvolučních vrstev.
- Různé konvoluční filtry




10 10 10 10 0 0 0 0
10 10 10 10 0 0 0 0
10 10 10 10 0 0 0 0
10 10 10 10 0 0 0 0
10 10 10 10 0 0 0 0
10 10 10 10 0 0 0 0
10 10 10 10 0 0 0 0
10 10 10 10 0 0 0 0
*
1 0 -1
1 0 -1
1 0 -1
Vertical
=
0 0 30 30 0 0
0 0 30 30 0 0
0 0 30 30 0 0
0 0 30 30 0 0
0 0 30 30 0 0
0 0 30 30 0 0

Konvoluce

Obrázek 14.11. Zaostření

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

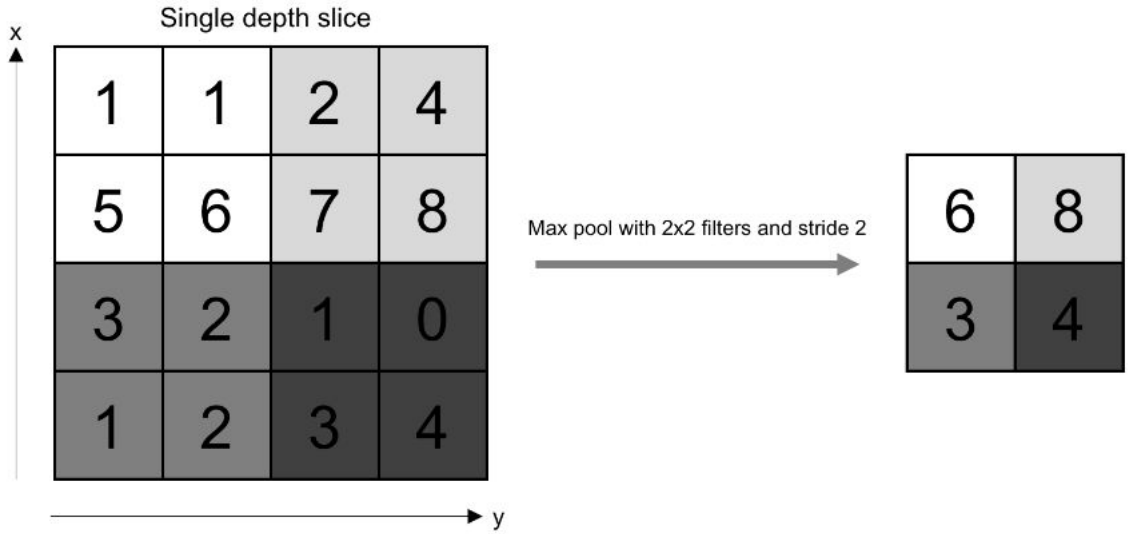
Konvoluce

Obrázek 14.12. Rozmazání

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

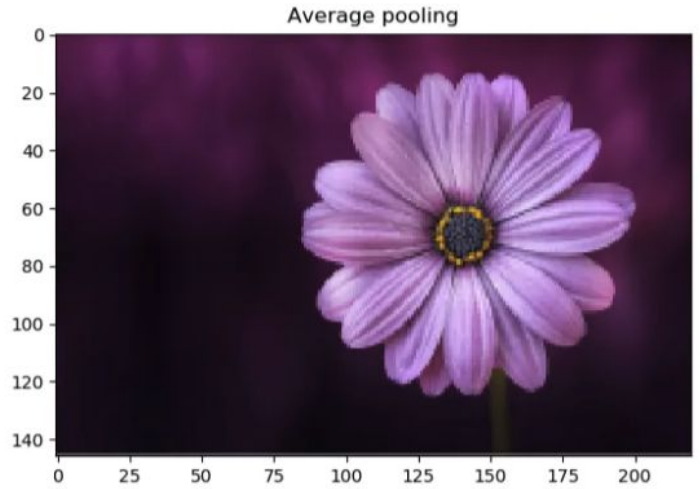
Pooling

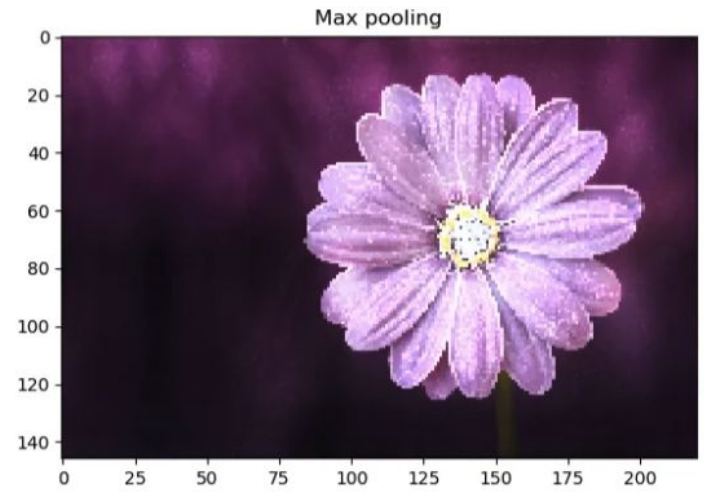
- Používá se k podvzorkování a lepší reprezentaci informací
- Max Pooling, Min Pooling, Average Pooling

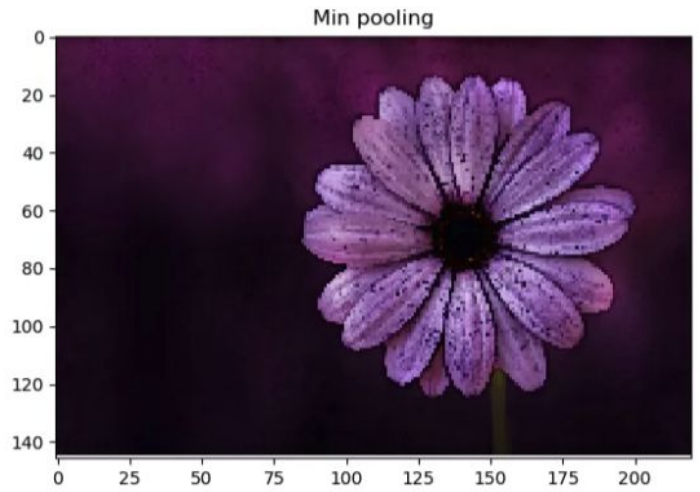


Pooling


Original Image
0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750

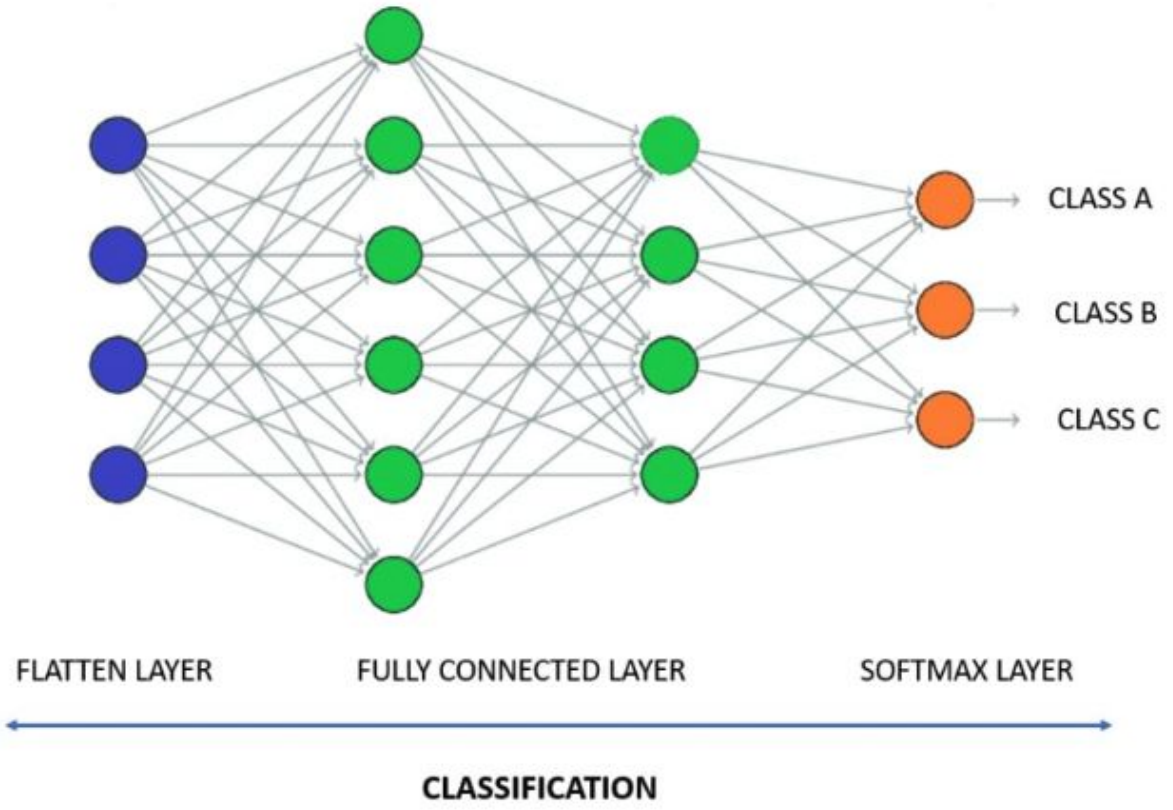

Average pooling
0
20
40
60
80
100
120
140
0
25
50
75
100
125
150
175
200


Max pooling
0
20
40
60
80
100
120
140
0
25
50
75
100
125
150
175
200

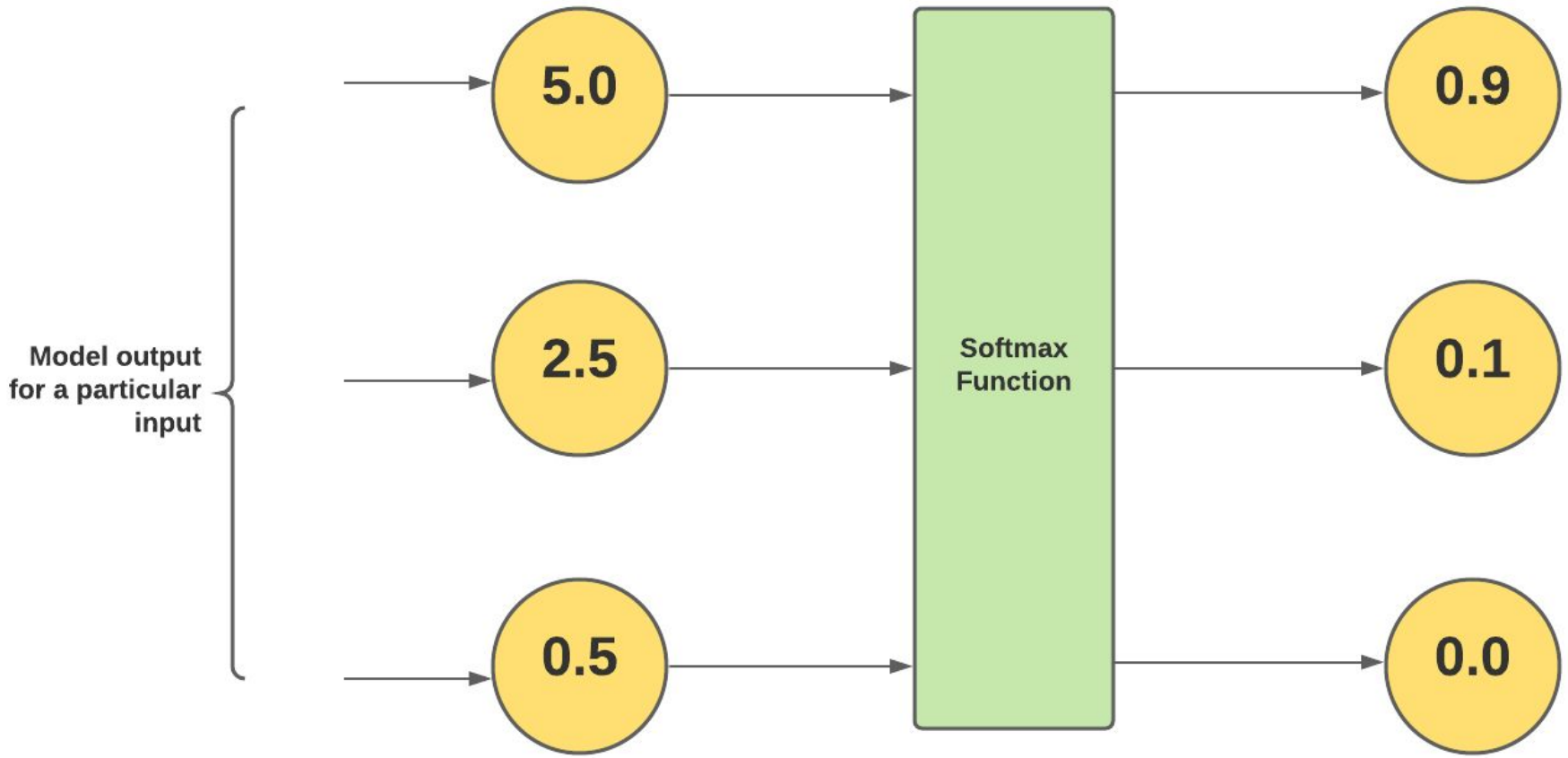

Min pooling
0
20
40
60
80
100
120
140
0
25
50
75
100
125
150
175
200

Fully Connected Layer

- Mapa vlastností je spojená do poslední vrstvy



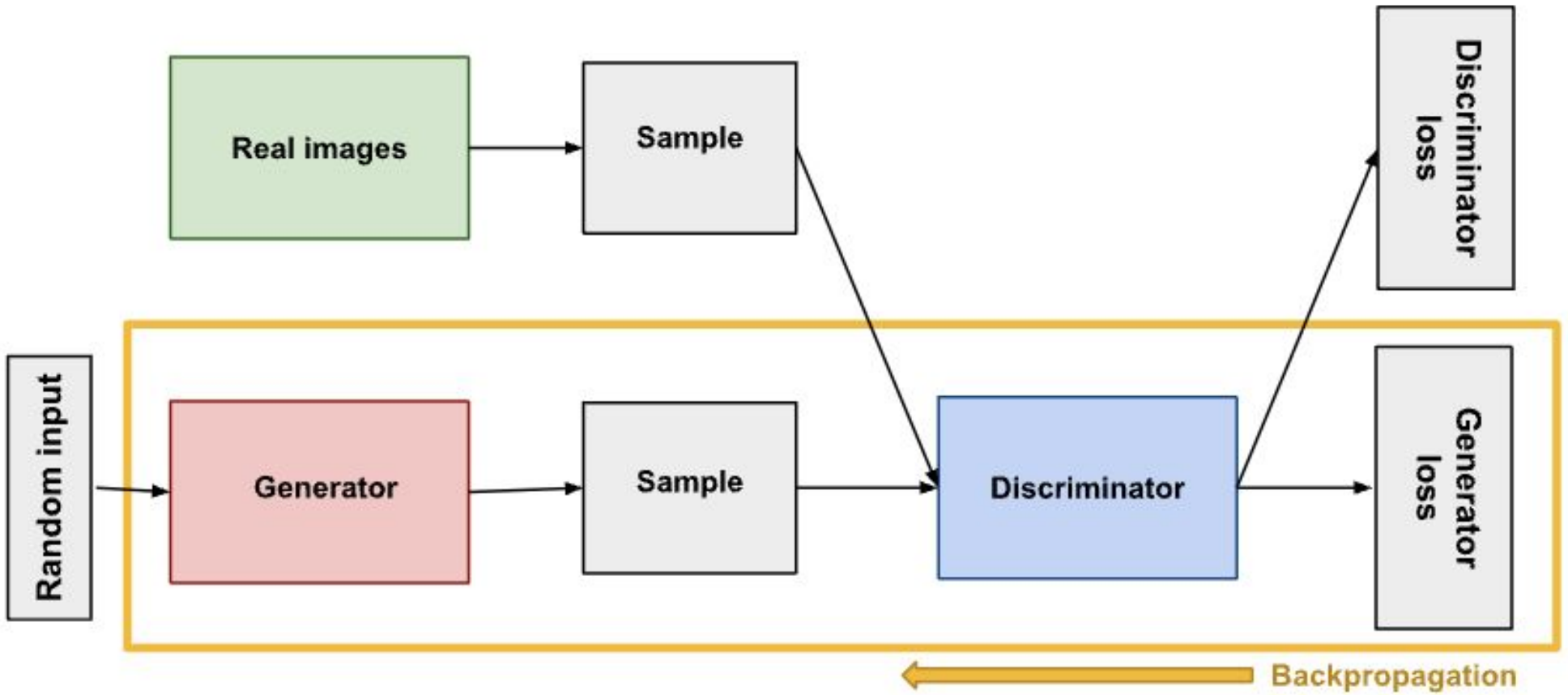
Softmax Layer



Generativní Adversariální sítě - GANs

- GANs se skládají ze dvou neuronových sítí, které jsou trénovány současně v rámci tzv. adversariálního procesu
  - generátor (generator)
  - diskriminátor (discriminator)
- **Generátor**: Snaží se vytvářet falešné vzorky, které jsou co nejvíce podobné reálným vzorkům z trénovacího datového souboru.
- **Diskriminátor:** Snaží se odlišit reálné vzorky od falešných vzorků vytvořených generátorem

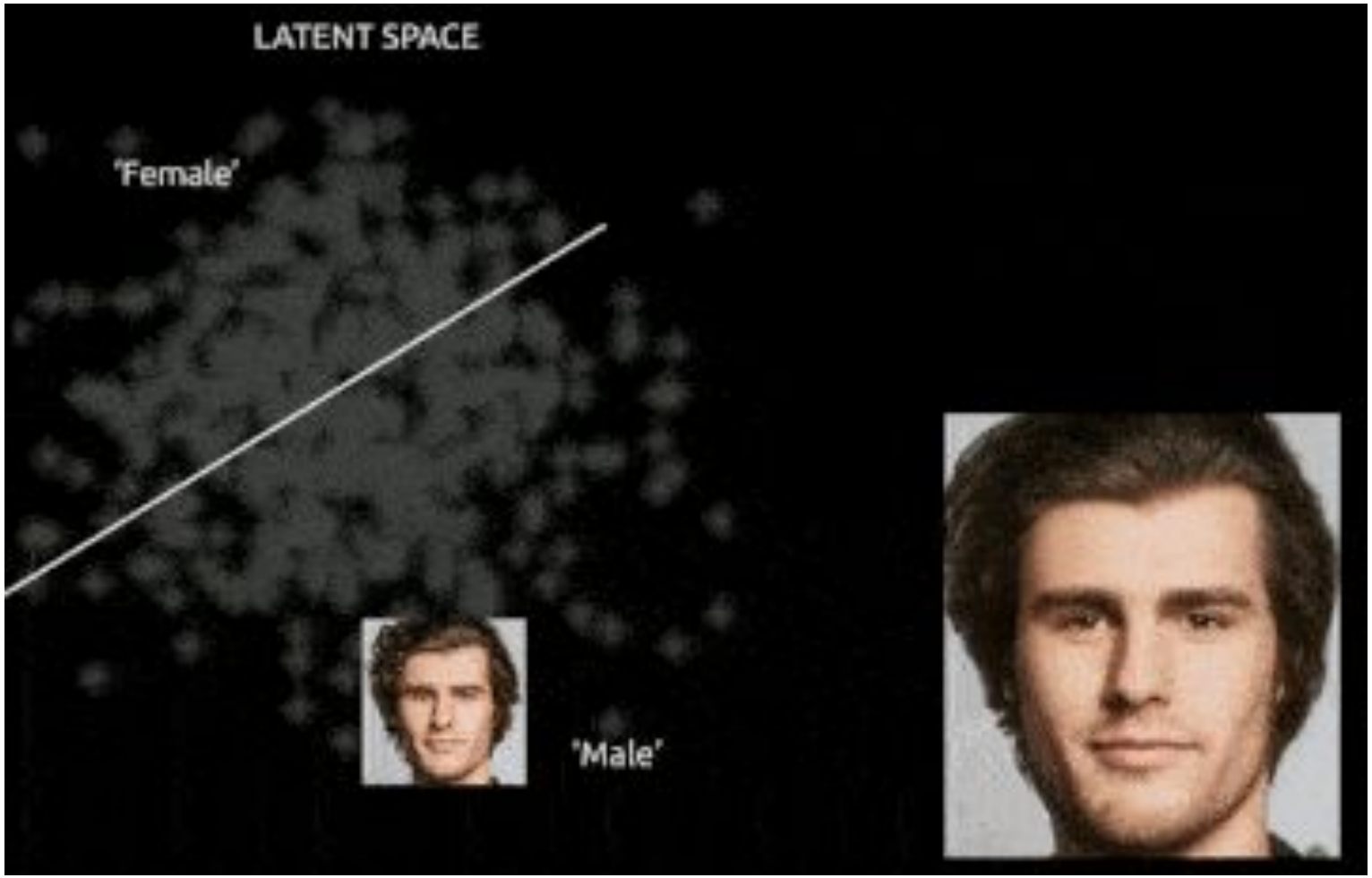
Generativní Adversariální sítě - GANs



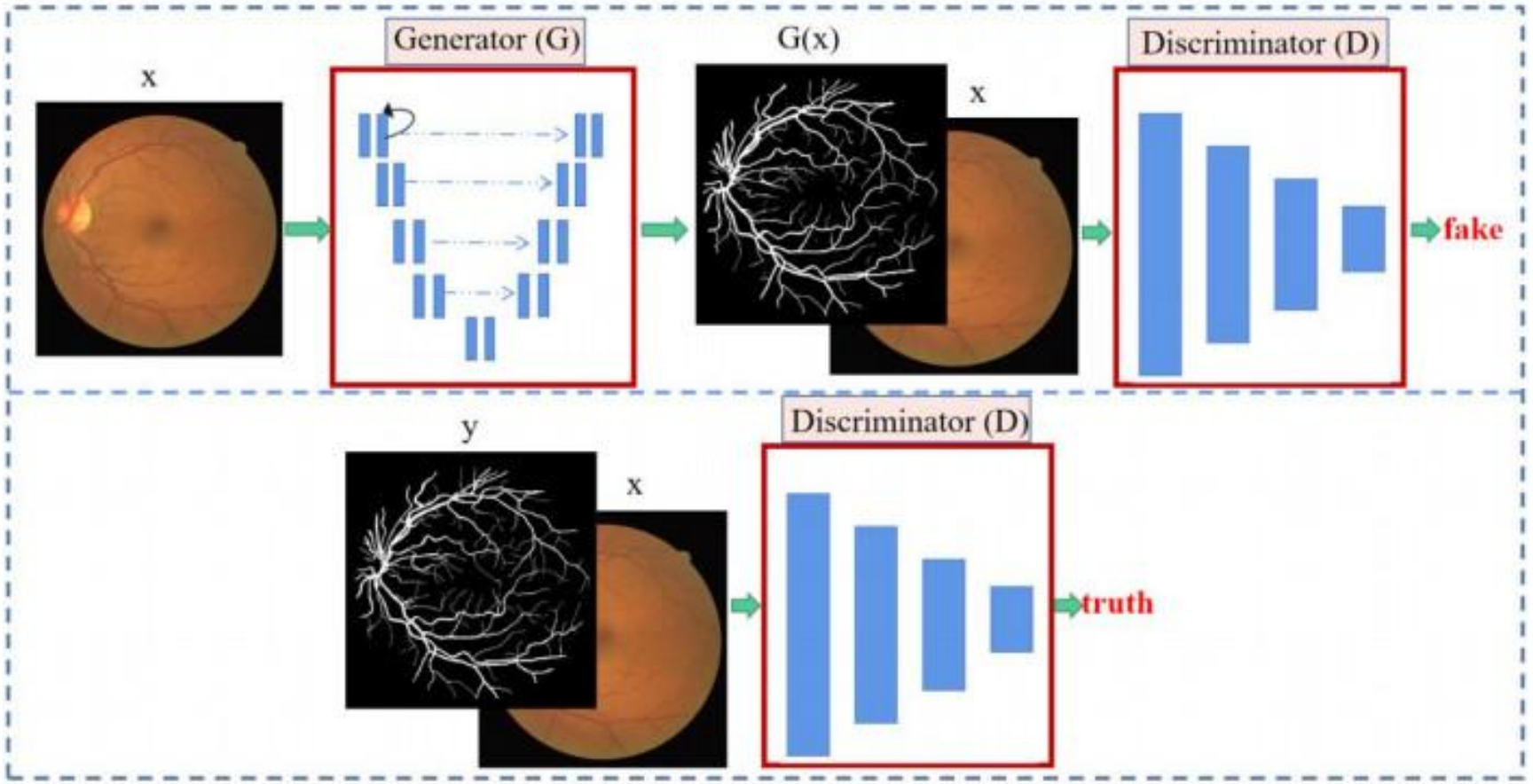
GANs - trénink

- Iterativní
- "hra s nulovým součtem" mezi generátorem a diskriminátorem:
  - Generátor vytváří nové vzorky a předává je diskriminátoru spolu s reálnými vzorky.
  - Diskriminátor se snaží co nejlépe rozpoznat, které vzorky jsou reálné a které jsou falešné.
  - Generátor se snaží vylepšit své schopnosti vytvářet falešné vzorky tak, aby oklamal diskriminátor.
  - Tento proces se opakuje, dokud generátor nevytvoří vzorky, které diskriminátor nedokáže odlišit od reálných.

GANs - trénink



GANs



[Chen Yue, Mingquan Ye, Peipei Wang, Daobin Huang, Xiaojie Lu. SRV-GAN: A generative adversarial network for segmenting retinal vessels[J]. Mathematical Biosciences and Engineering, 2022, 19(10): 9948-9965. doi: 10.3934/mbe.2022464]

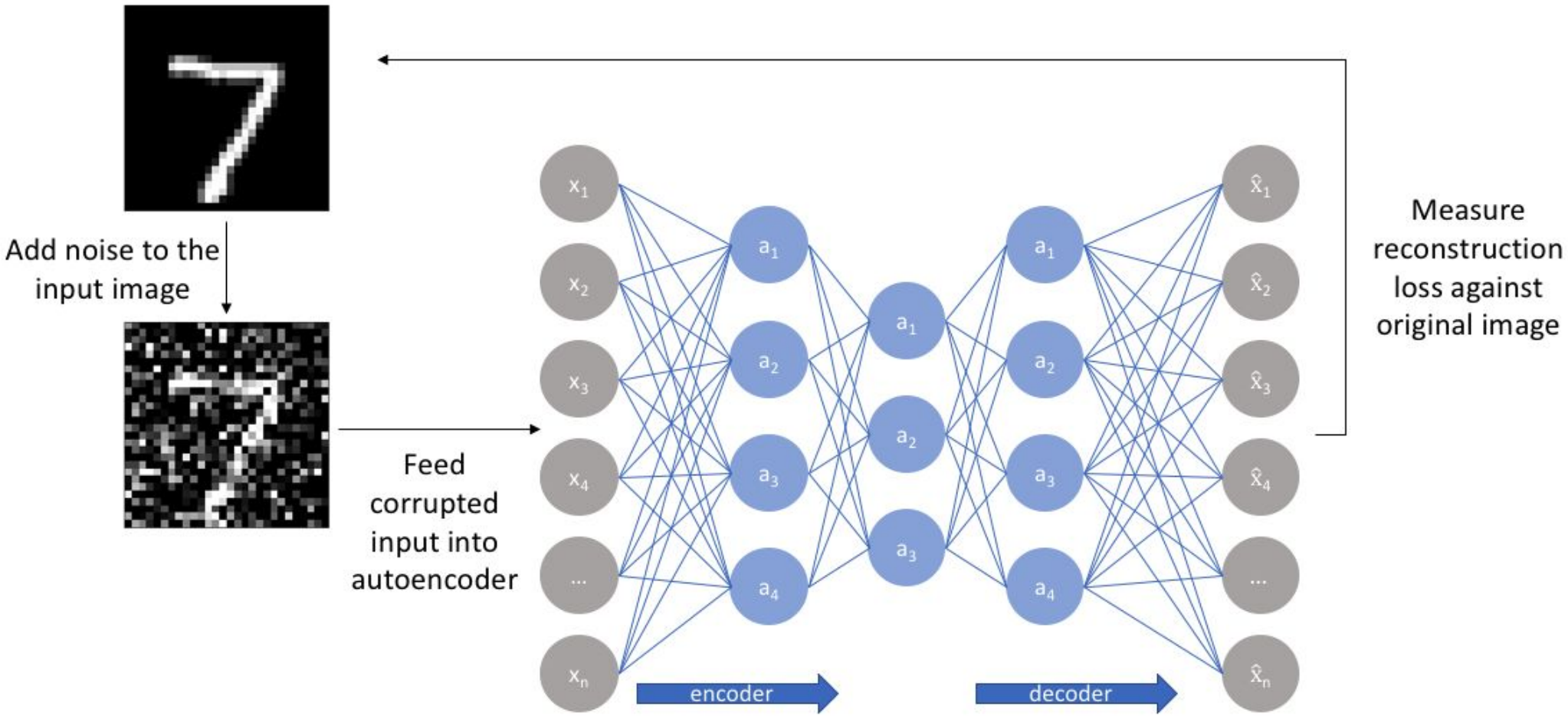
GANs - aplikace

- **Zlepšování kvality lékařských snímků**
  - Super-resolution: Zlepšování nízkokvalitních lékařských snímků
  - Odstranění šumu
- **Generování syntetických lékařských snímků**
  - Vytváření trénovacích dat: GANs mohou generovat syntetické lékařské snímky
- **Rekonstrukce lékařských snímků**
  - Rekonstrukce z podvzorkovaných dat: GANs mohou rekonstruovat plnohodnotné lékařské snímky z podvzorkovaných dat
  - Kompletace snímků: GANs mohou doplňovat chybějící části lékařských snímků

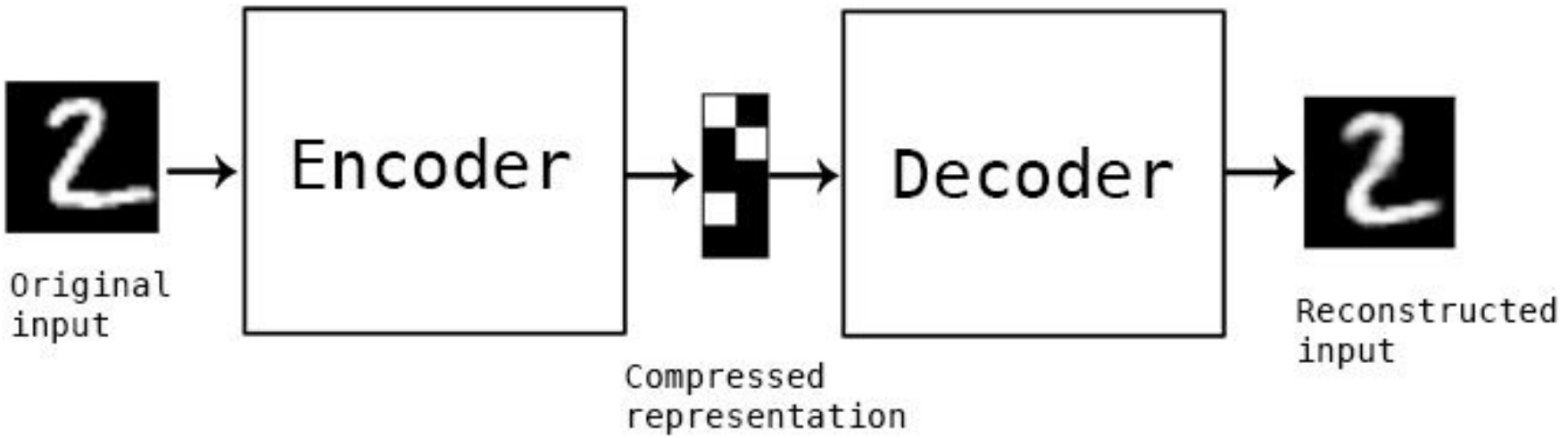
Autoenkodéry - Autoencoders

- Typ neuronových sítí používaných pro neřízené učení
- Hlavním cílem je naučit se efektivní reprezentace dat, často pro účely redukce dimenzionality nebo odstranění šumu.
- Skládají se ze dvou hlavních částí:
  - Enkodéru
  - Dekodéru

Autoenkodéry



Autoenkodéry - Autoencoders

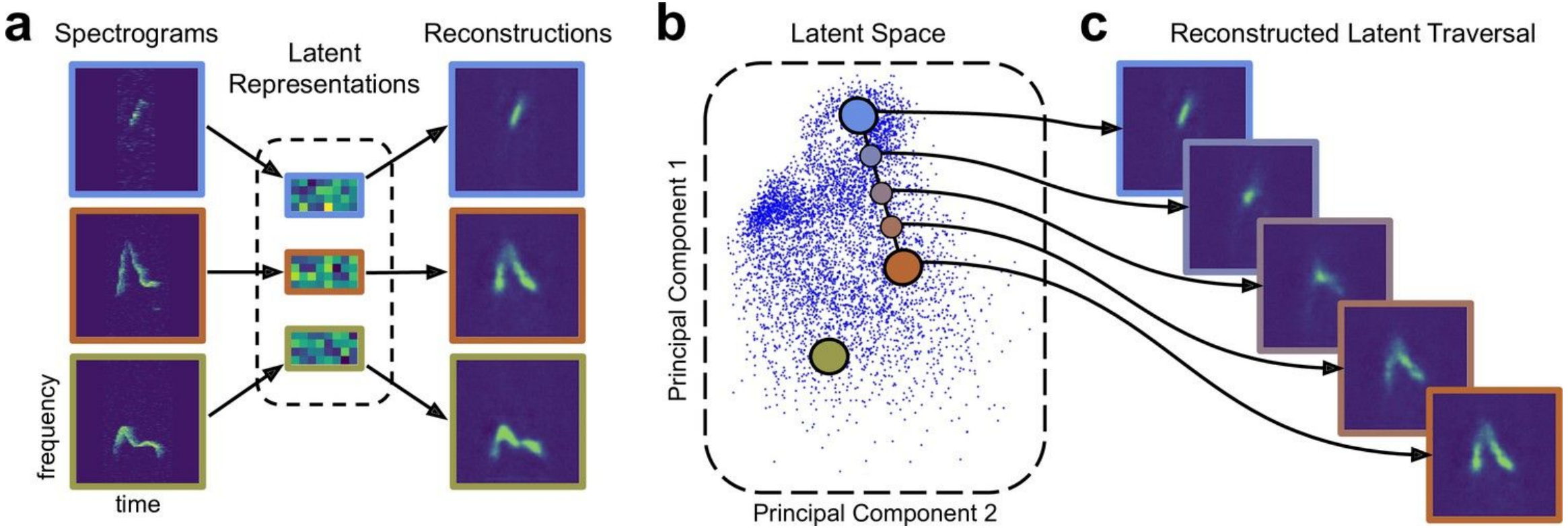


Autoenkodéry - Klíčové koncepty

- **Enkodér**
  - Převádí vstupní data na komprimovanou reprezentaci (latentní prostor). Enkodér je neuronová síť, která zmenšuje dimenzi vstupních dat, čímž vytváří zhuštěnou reprezentaci, která zachovává podstatné informace.

- **Latentní prostor (Latent Space)**
  - Reprezentace s nižší dimenzí, která obsahuje klíčové rysy vstupních dat. Tato reprezentace se nazývá latentní prostor a je výstupem enkodéru.

Autoenkodéry - Klíčové koncepty



[Jack Goffinet, Samuel Brudner, Richard Mooney, John Pearson (2021) Low-dimensional learned feature spaces quantify individual and group differences in vocal repertoires eLife 10:e67855]

Autoenkodéry - Klíčové koncepty

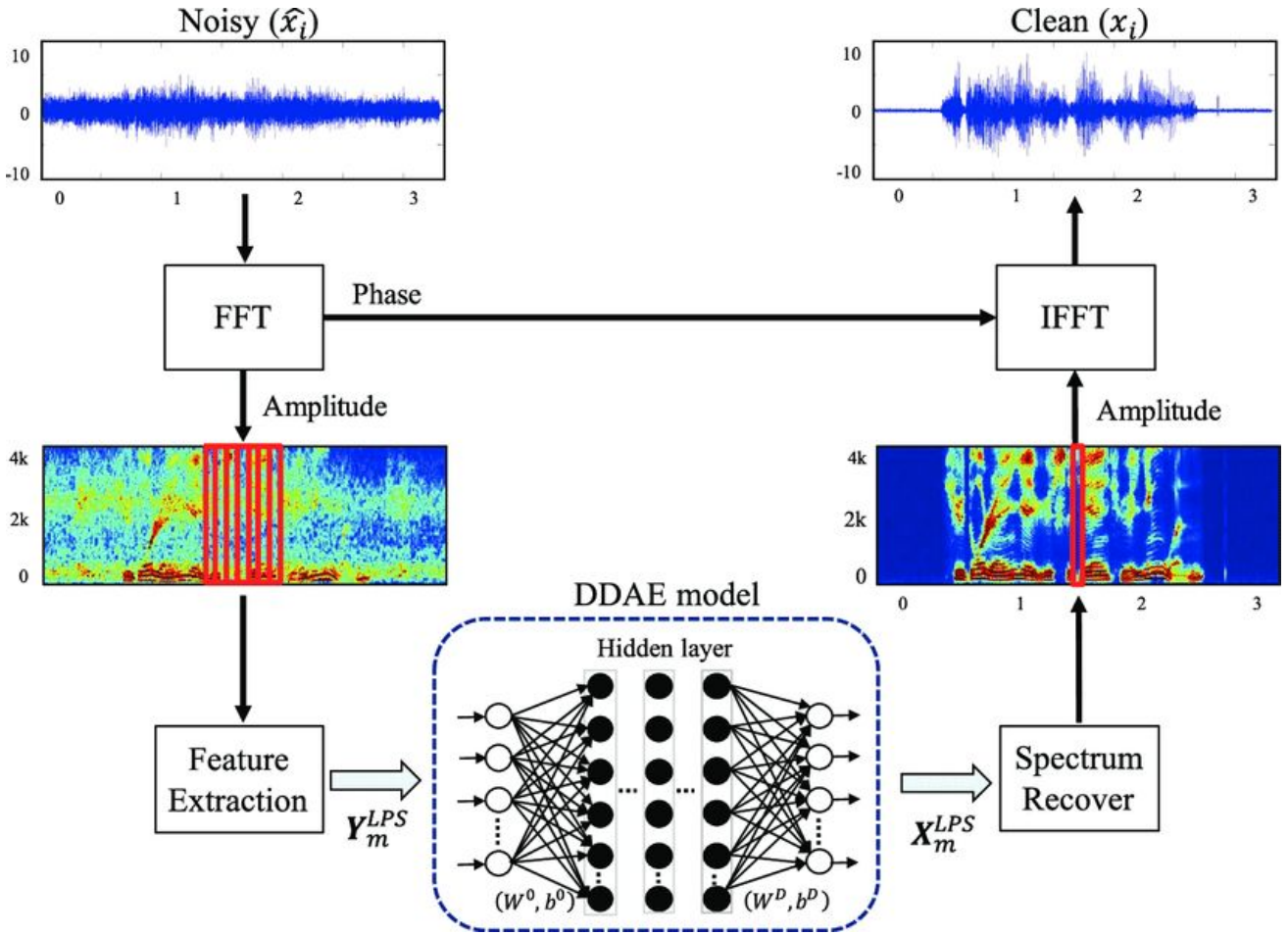
- **Dekodér**
  - Rekonstruuje data z latentní reprezentace zpět do původního formátu. Dekodér je neuronová síť, která přebírá latentní prostor a snaží se rekonstruovat původní vstupní data.

- **Ztrátová funkce (Loss Function)**
  - Měří rozdíl mezi původními vstupními daty a rekonstruovanými daty. Typicky se používá metrika jako Mean Squared Error (MSE).

Autoenkodéry - aplikace

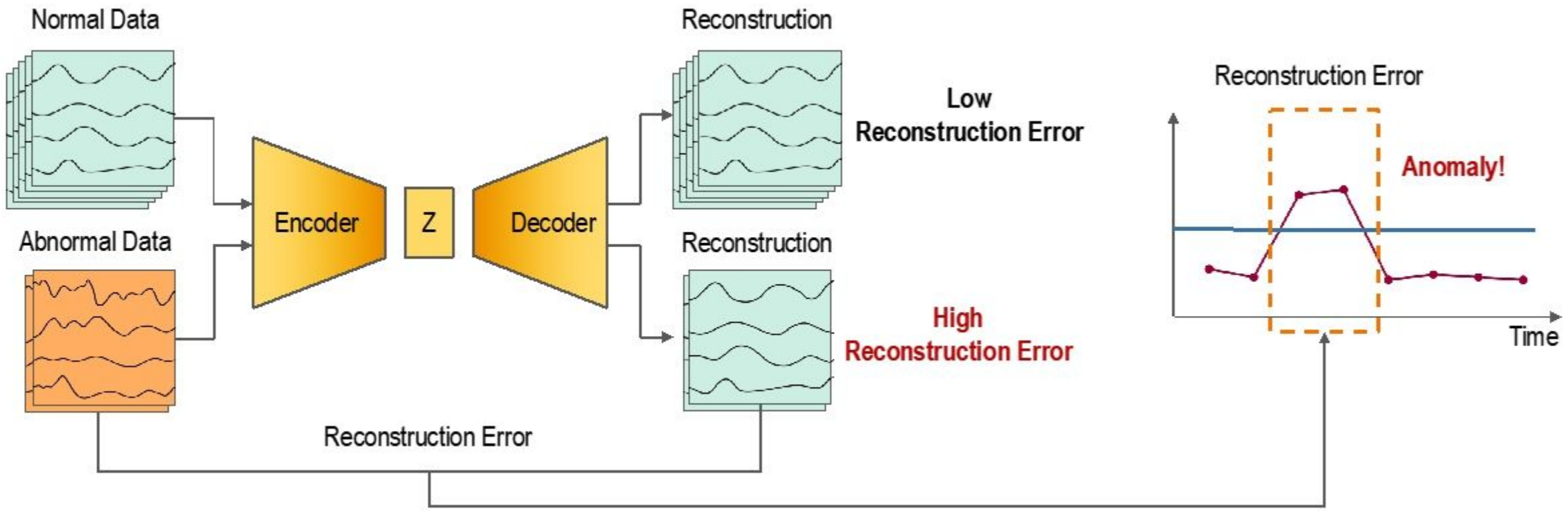
- Redukce dimenze
- Odstranění šumu
- Generování dat
- Anomaly Detection
- Předzpracování dat
- Vizualizace dat

Autoenkodéry - Odstranění šumu



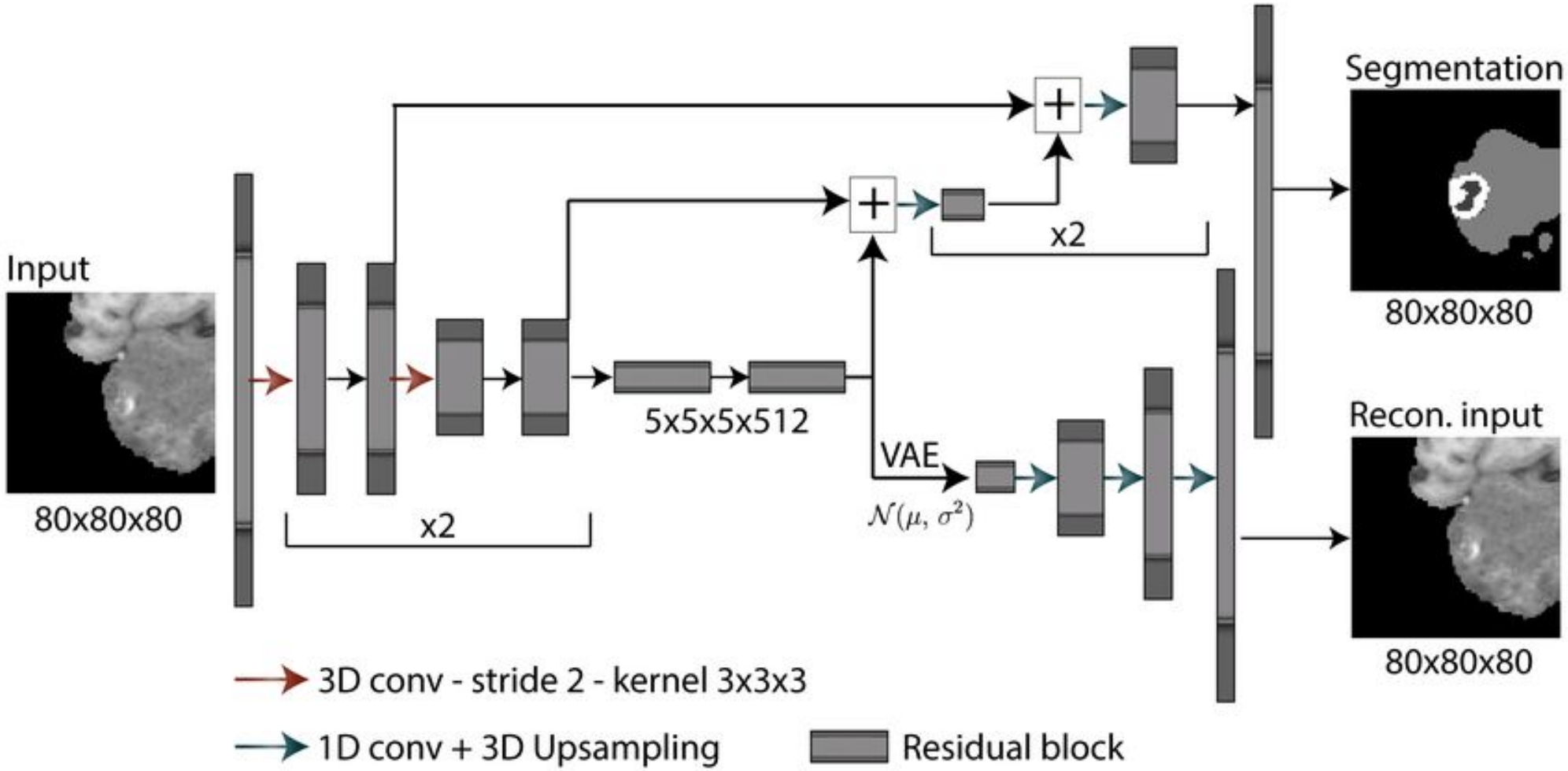
[Lai, Ying-Hui & Tsao, Yu & lu, Xugang & Chen, Fei & Su, Yu-Ting & Chen, Kuang-Chao & Chen, Yu-Hsuan & Chen, Li-Ching & Li, Lieber & Lee, Chin-Hui. (2018). Deep Learning–Based Noise Reduction Approach to Improve Speech Intelligibility for Cochlear Implant Recipients. Ear and Hearing. 39. 1. 10.1097/AUD.0000000000000537.]

Autoenkodéry - Anomaly Detection



[https://www.lgresearch.ai/blog/view/?seq=231]

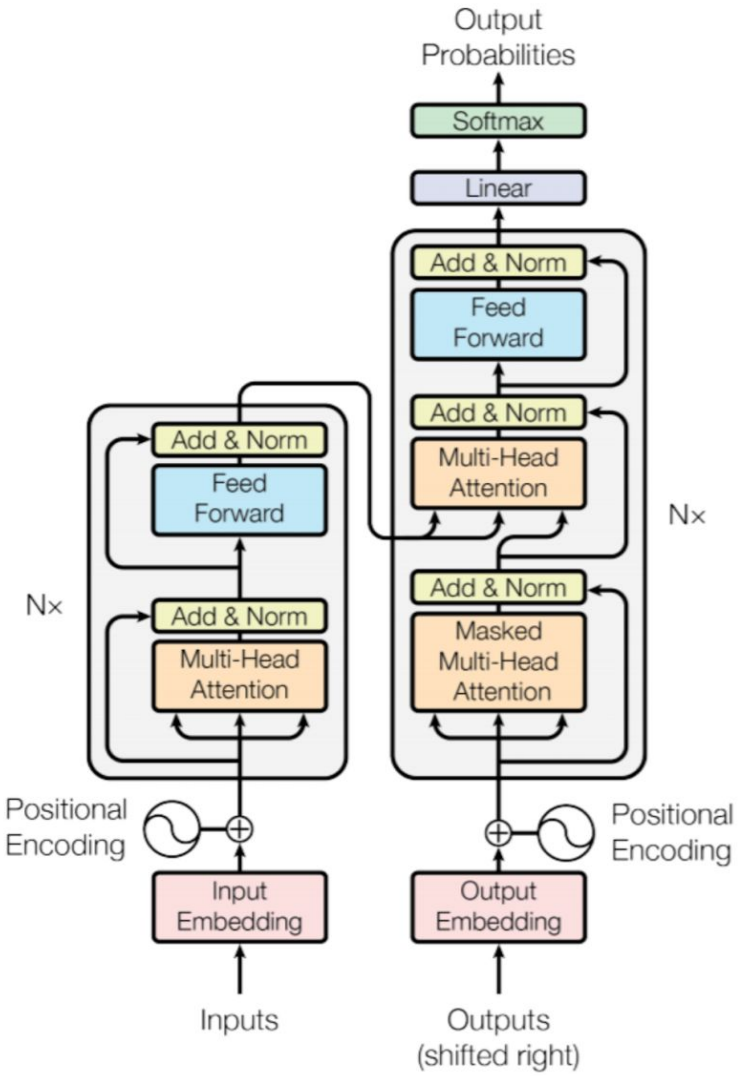
Autoenkodéry- Segmentace

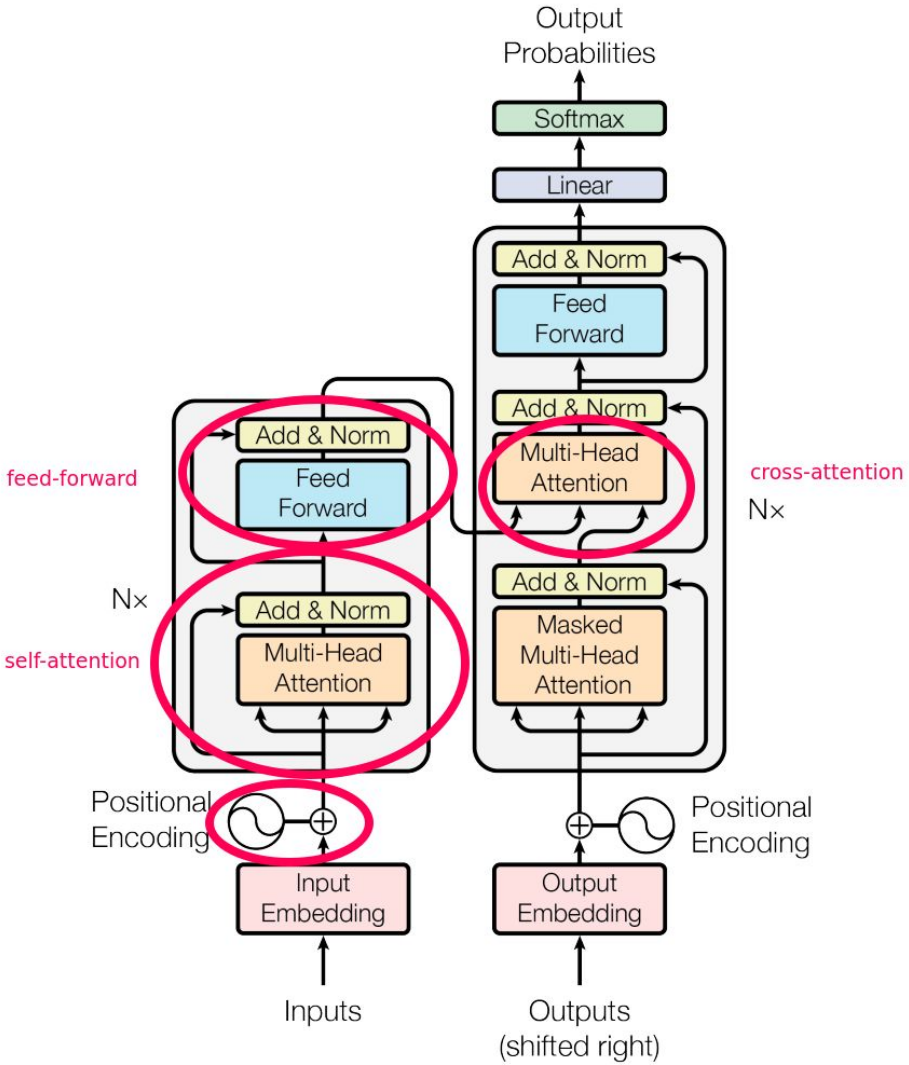


Transformer Networks

- Typ architektury hlubokých neuronových sítí
- Attention is All You Need - 2017, Vaswani
- Staly se základem pro mnoho pokročilých modelu v oblasti zpracování přirozeného jazyka (NLP) a dalších sekvenčních úloh
- Efektivní při zpracování sekvenčních dat
- Modelují dlouhé závislosti v datech

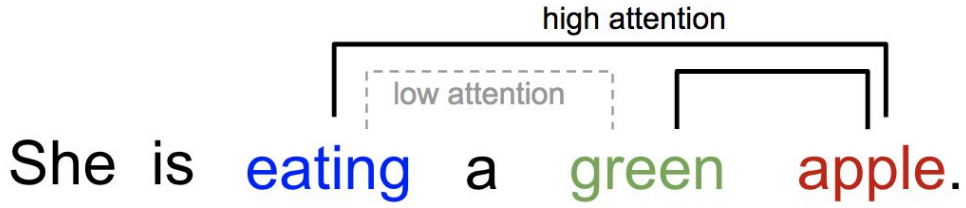
Transformer Networks





Transformer Networks - klíčové koncepty

- **Attention Mechanism**
  - **Self-Attention**: Každé slovo (token) v sekvenci se váže (attend) na každé jiné slovo v sekvenci, včetně sebe sama. To umožňuje modelu porozumět kontextu každého slova v rámci celé sekvence.
- **Positional Encoding**
  - Protože transformery neobsahují žádné rekurentní nebo konvoluční vrstvy, používají positional encoding k zakódování informace o pořadí slov v sekvenci.



Transformer Networks - klíčové koncepty

- **Feed-Forward Neural Networks**
  - Každý výstup z multi-head attention vrstvy je procházen samostatnou feed-forward neuronovou sítí, což poskytuje dodatečnou nelinearitu.
- **Layer Normalization and Residual Connections**
  - Každá sub-vrstva v transformeru je doplněna o residual connection (přeskokové spojení) a následně normalizována, což pomáhá při trénování hlubokých sítí.

Transformer Networks - klíčové koncepty

- **Encoder**: Skládá se z několika identických vrstev (např. 6 vrstev), přičemž každá vrstva obsahuje dvě sub-vrstvy: **multi-head self-attention mechanism a feed-forward neuronovou síť.**

- **Decoder**: Také se skládá z několika identických vrstev, ale každá vrstva obsahuje tři sub-vrstvy: **multi-head self-attention mechanism, multi-head attention mechanism** (který se váže na výstupy z encoderu) a **feed-forward neuronovou síť.**

Transformer Networks - aplikace

- **BERT /Bidirectional Encoder Representation from Transformers**
  - Používá dvousměrný (bidirectional) trénink k porozumění kontextu slova z obou stran jeho okolí.
  - **Aplikace**: Textová klasifikace, otázky a odpovědi, rozpoznávání pojmenovaných entit (NER).

- **GPT (Generative Pre-trained Transformer):**
  - Používá unidirectional (jednosměrný) trénink k generování textu.
  - **Aplikace:** Generování textu, překlad, shrnování textu.

ChatGPT

- ChatGPT je model strojového učení založený na architektuře transformerů
  - GPT (Generative Pre-trained Transformer)
- Jsou trénovány na obrovských množstvích textových dat
- Používají se k generování textu na základě poskytnutých vstupů

ChatGPT - klíčové koncepty

- **Transformers**
  - Transformerová architektura: efektivní zpracování sekvenčních dat
  - Pouze dekodér generují text sekvenčně, predikují další slovo na základě předchozích slov v sekvenci.
- **Self-attention**
  - V rámci každé vrstvy modelu se pro každý token (slovo) počítá vážený průměr všech ostatních tokenů v sekvenci. Váhy jsou určeny na základě podobnosti mezi tokeny, což umožňuje modelu zachytit kontextuální závislosti

ChatGPT

N can you recognize this:

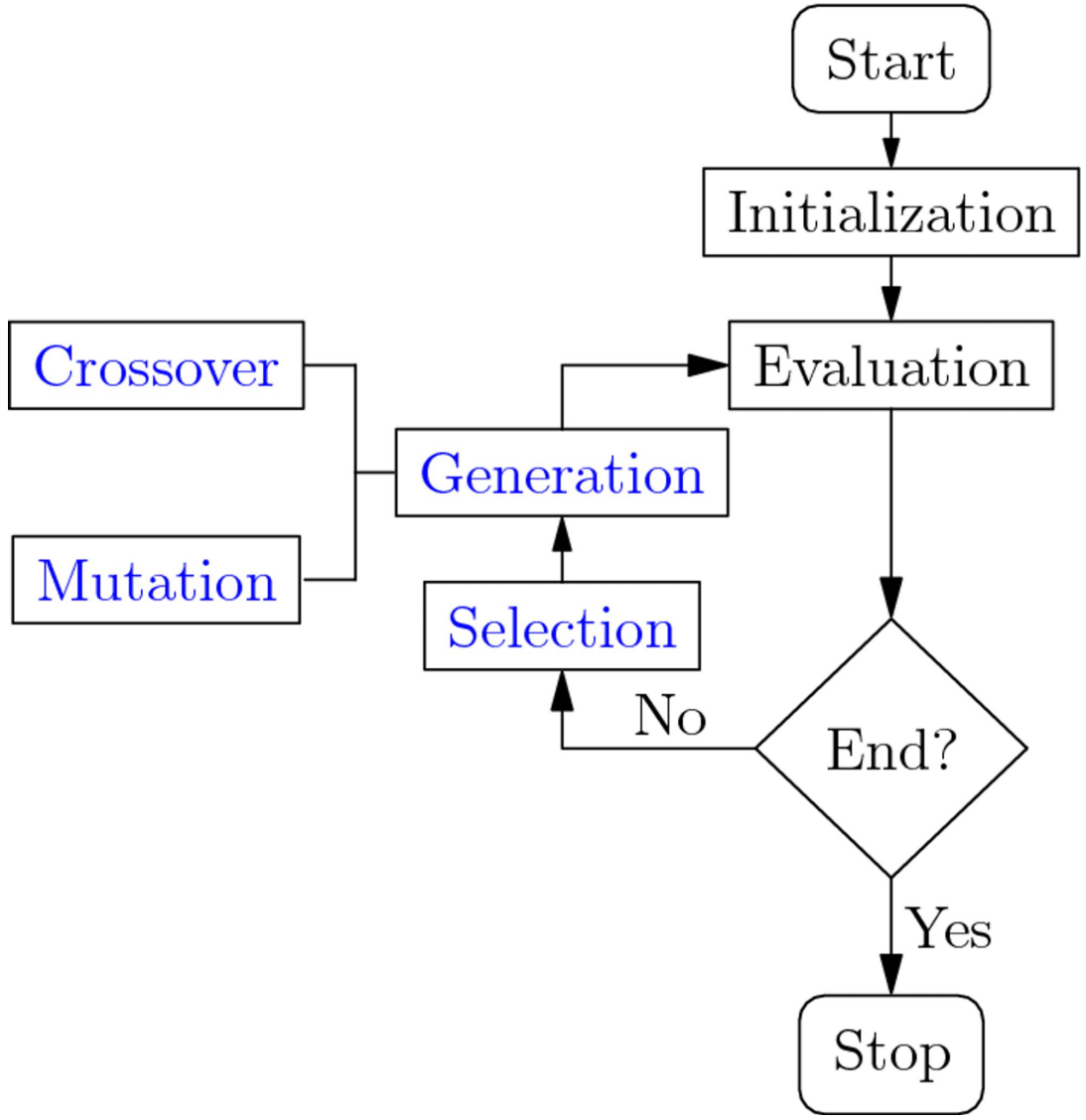
Yes, that is the famous ASCII art representation of the Mona Lisa painting by Leonardo da Vinci.

Genetický algoritmus

- Evoluční optimalizační technika inspirovaná procesy přírodní evoluce
- Používány k řešení složitých optimalizačních a hledacích problémů

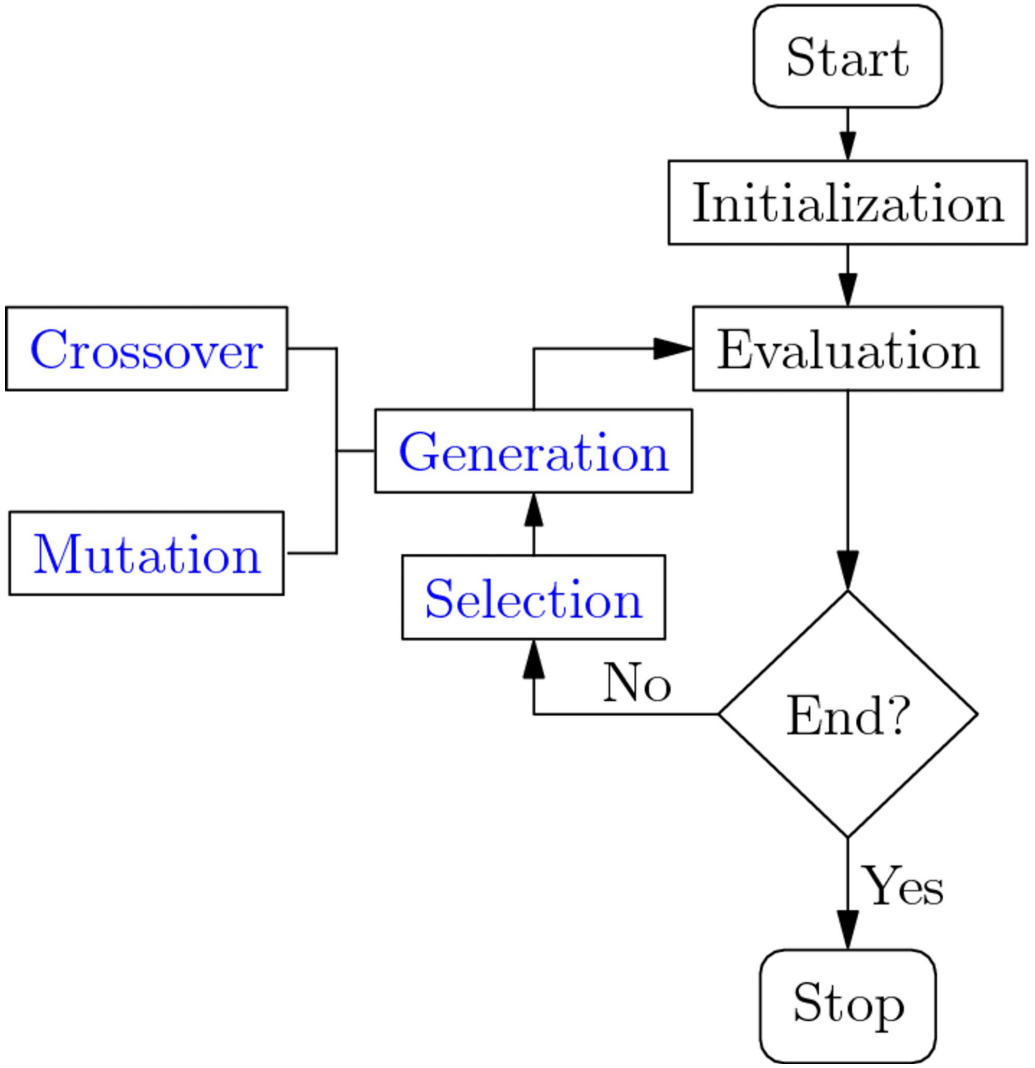
Genetický algoritmus - klíčové koncepty

- Populace
- Chromozomy
- Fitness funkce
- Selekce
- Křížení
- Mutace



Genetický algoritmus - trénovací proces

- Inicializace populace
- Vyhodnocení fitness všech jedinců
- Výběr rodičů na základě jejich fitness
- Vytvoření nové generace - mutace
- Nahrazení staré populace novou
- Opakování



Genetický algoritmus

Výhody:

- Schopnost najít řešení v rozsáhlém a složitém prostoru řešení.
- Flexibilita při definování fitness funkce a genetických operátorů.
- Vhodnost pro problémy, kde tradiční optimalizační metody selhávají.

Nevýhody:

- Může být výpočetně náročný, zejména pro velké populace a mnoho generací.
- Nezaručuje nalezení globálního optima.
- Výkon závisí na správné volbě parametrů (velikost populace, pravděpodobnosti křížení a mutace).

AI v neurologii

Automatic classification of MR scans in Alzheimer’s disease

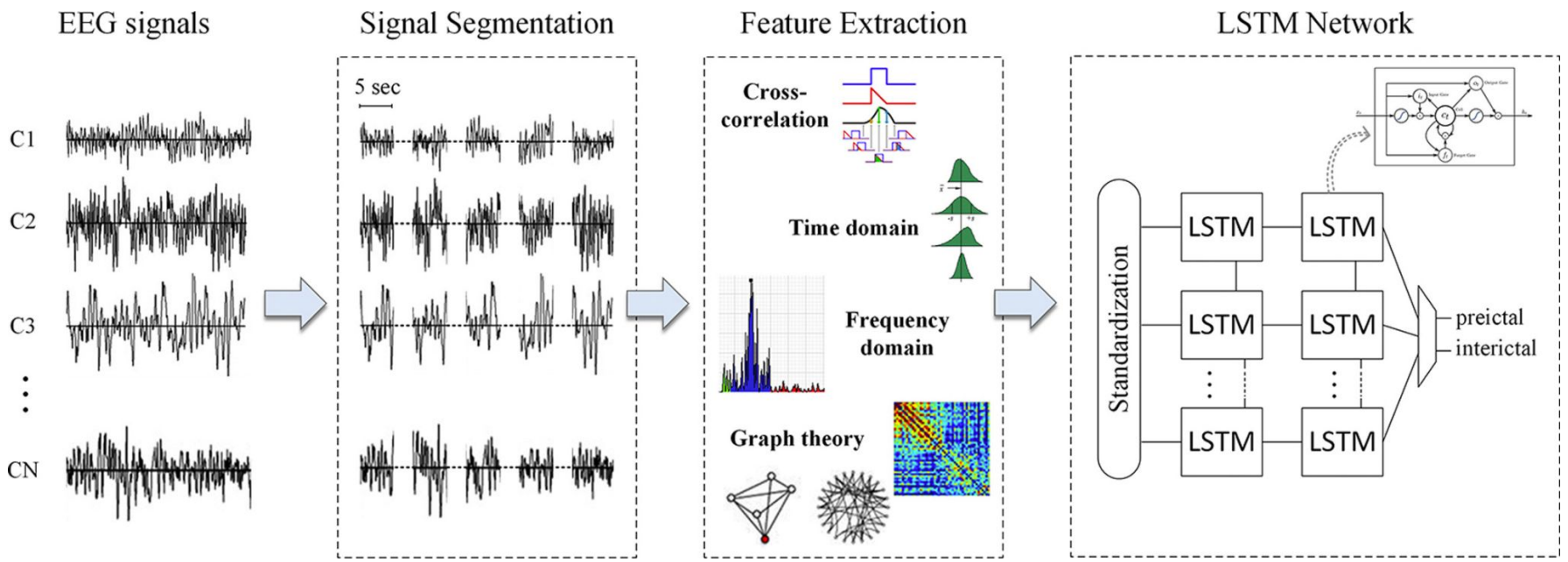
- **Support Vector Machine Algorithm**

- 96 % patologicky ověřených pacientů s AD bylo správně klasifikováno pomocí MRI snímků mozku.
- Pacienti s mírnou, klinicky pravděpodobnou AD a kontroly byli správně rozděleni v 89 %.

[Klöppel S, Stonnington CM, Chu C, et al. Automatic classification of MR scans in Alzheimer's disease. *Brain*. 2008;131(Pt 3):681-689. doi:10.1093/brain/awm319]

A long short-term memory deep learning network for the prediction of epileptic seizures using EEG signals

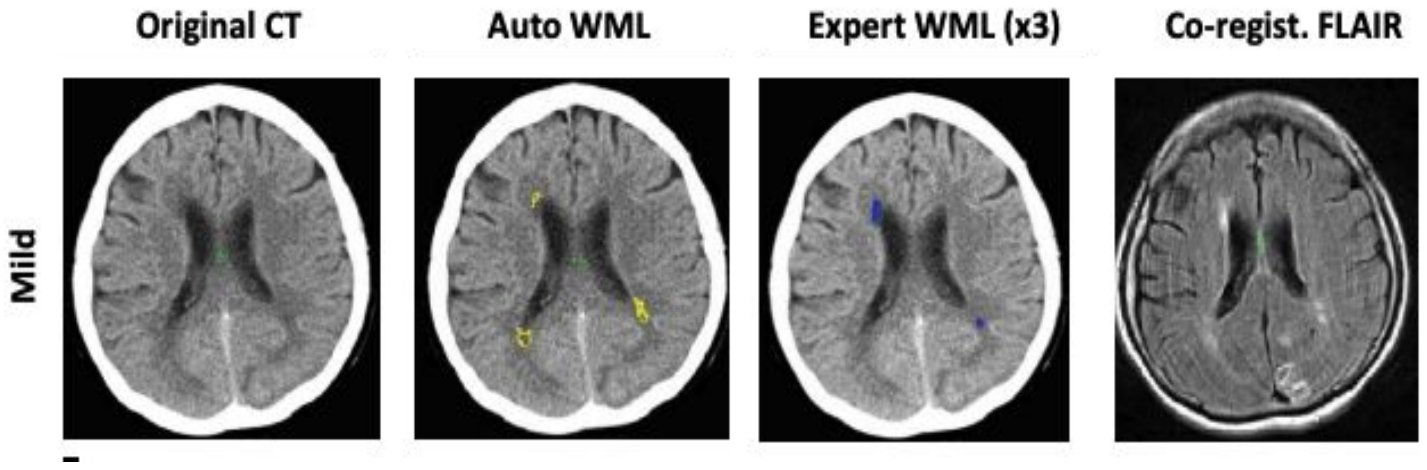
No seizures were missed with zero false predictions in up to 17 of 24 cases across four preictal windows up to 2 h.



[Kostas M. Tsiouris, et al, A Long Short-Term Memory deep learning network for the prediction of epileptic seizures using EEG signals, Computers in Biology and Medicine, Vol 99, 2018, 24-37, ISSN 0010-4825, https://doi.org/10.1016/j.compbiomed.2018.05.019.]

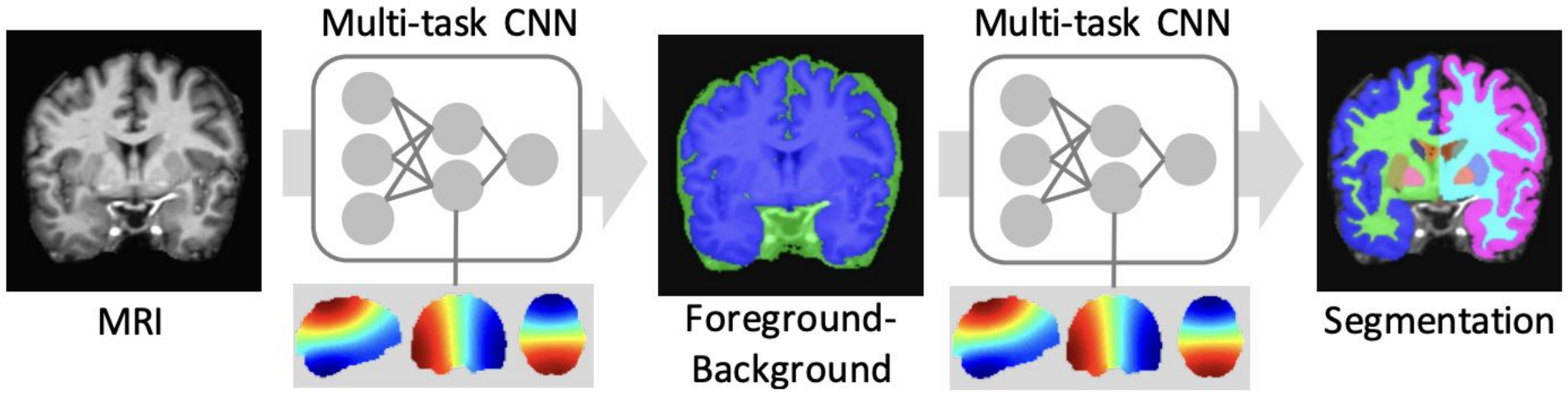
Rapid automated quantification of cerebral leukoaraiosis on CT images: a multicenter validation study

- **Random Forest Algorithm**
- Automatická detekce lézí bílé hmoty byla aplikována na CT snímky od subjektů s akutní ischemickou cévní mozkovou příhodou.
- **Metoda měla podobné výsledky jako experti (CT: R2 = 0,71, MRI: R2 = 0,85).**
- Algoritmus měl chybovost 4 % a střední dobu zpracování 32 s (7,9 min).

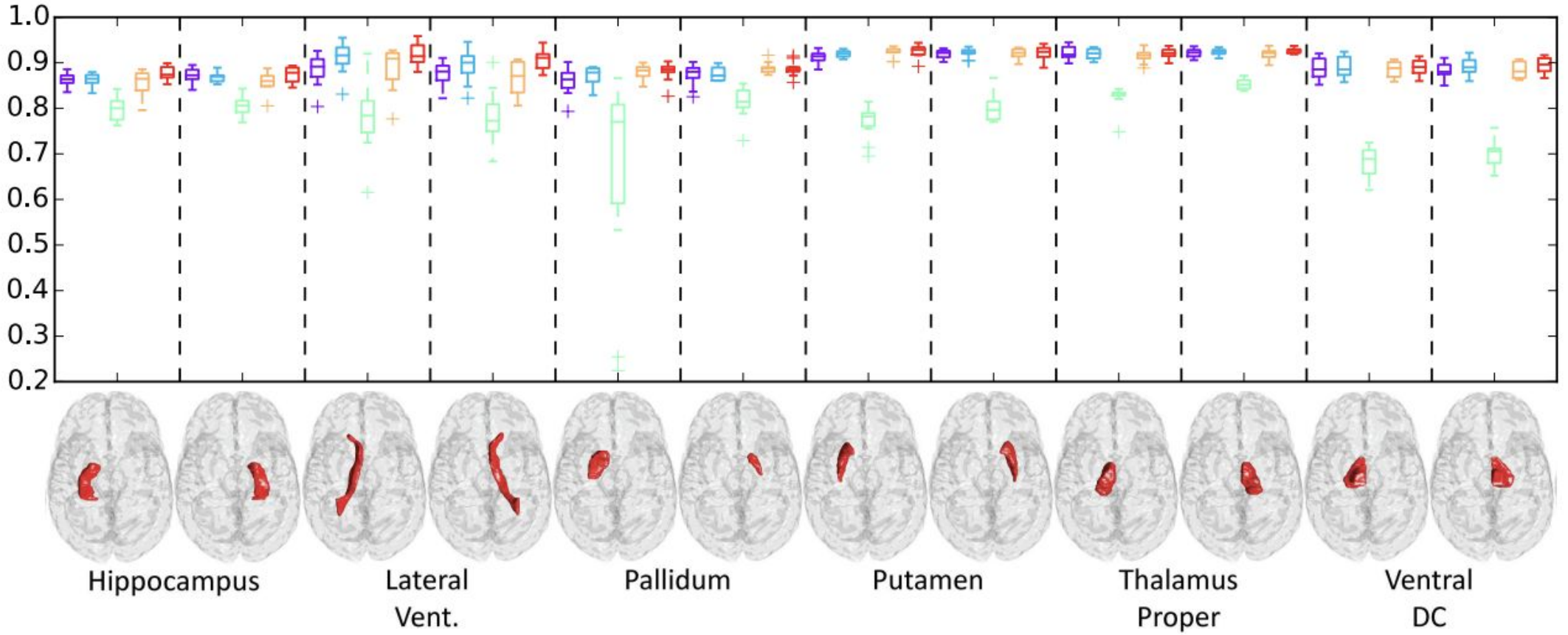


[Rapid Automated Quantification of Cerebral Leukoaraiosis on CT Images: A Multicenter Validation Study Liang Chen, et al. Radiology 2018 288:2, 573-581]

DeepNAT: Deep Convolutional Neural Network for Segmenting Neuroanatomy

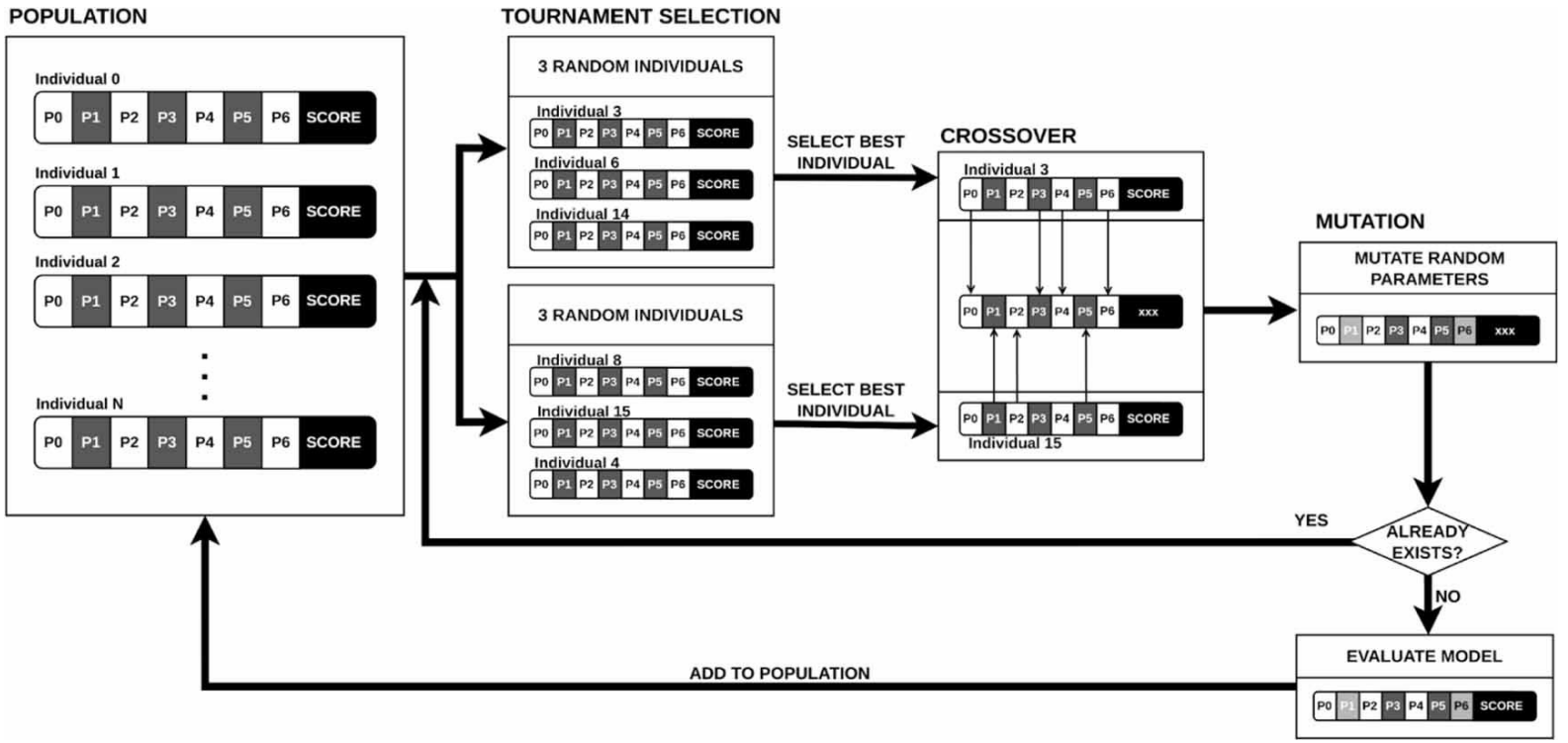


[Christian Wachinger, Martin Reuter, Tassilo Klein, DeepNAT: Deep convolutional neural network for segmenting neuroanatomy,, NeuroImage, Vol 170, 2018, 434-445,ISSN 1053-8119, https://doi.org/10.1016/j.neuroimage.2017.02.035.]


1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
Hippocampus
Lateral Vent.
Pallidum
Putamen
Thalamus Proper
Ventral DC
DeepNAT
DeepNATcrf
FreeSurfer
Spatial Staple
PICSL

Genetic algorithm designed for optimization of neural network architectures for intracranial EEG recordings analysis

Metoda zlepšila makro F1 skóre state-of-the-art modelu na dvou nezávislých datasetech (FNUSA, Mayo Clinic) z 0,91 na 0,97 a z 0,92 na 0,94.



[Pijackova K, Nejedly P, Kremen V, Plesinger F, Mivalt F, Lepkova K, Pail M, Jurak P, Worrell G, Brazdil M, Klimes P. Genetic algorithm designed for optimization of neural network architectures for intracranial EEG recordings analysis. J Neural Eng. 2023 Jun 16;20(3). doi: 10.1088/1741-2552/acdc54. PMID: 37285840]

AI v medicíně ČR

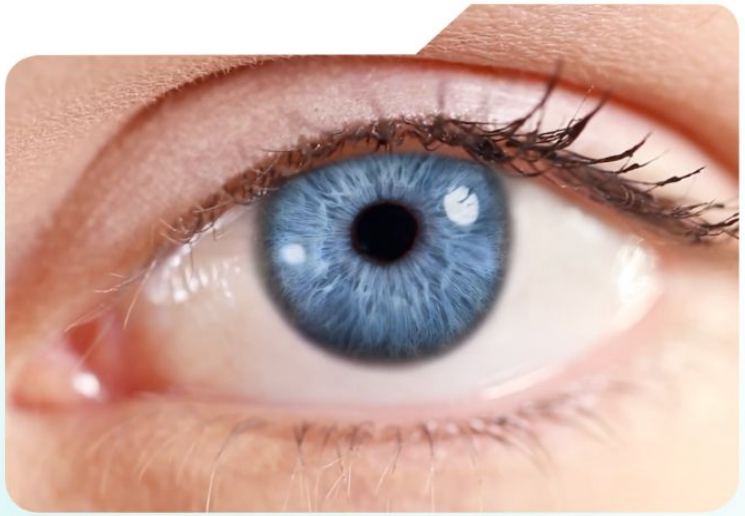
Aireen

Aireen

Jak to funguje Diagnostika Media O nás Pro pacienty Kontakt Eng

Diagnostika diabetické retinopatie umělou inteligencí

- Hrazeno zdravotními pojišťovnami
- Intuitivní ovládání pro sestry i lékaře
- Výsledky diagnostiky do 60 vteřin

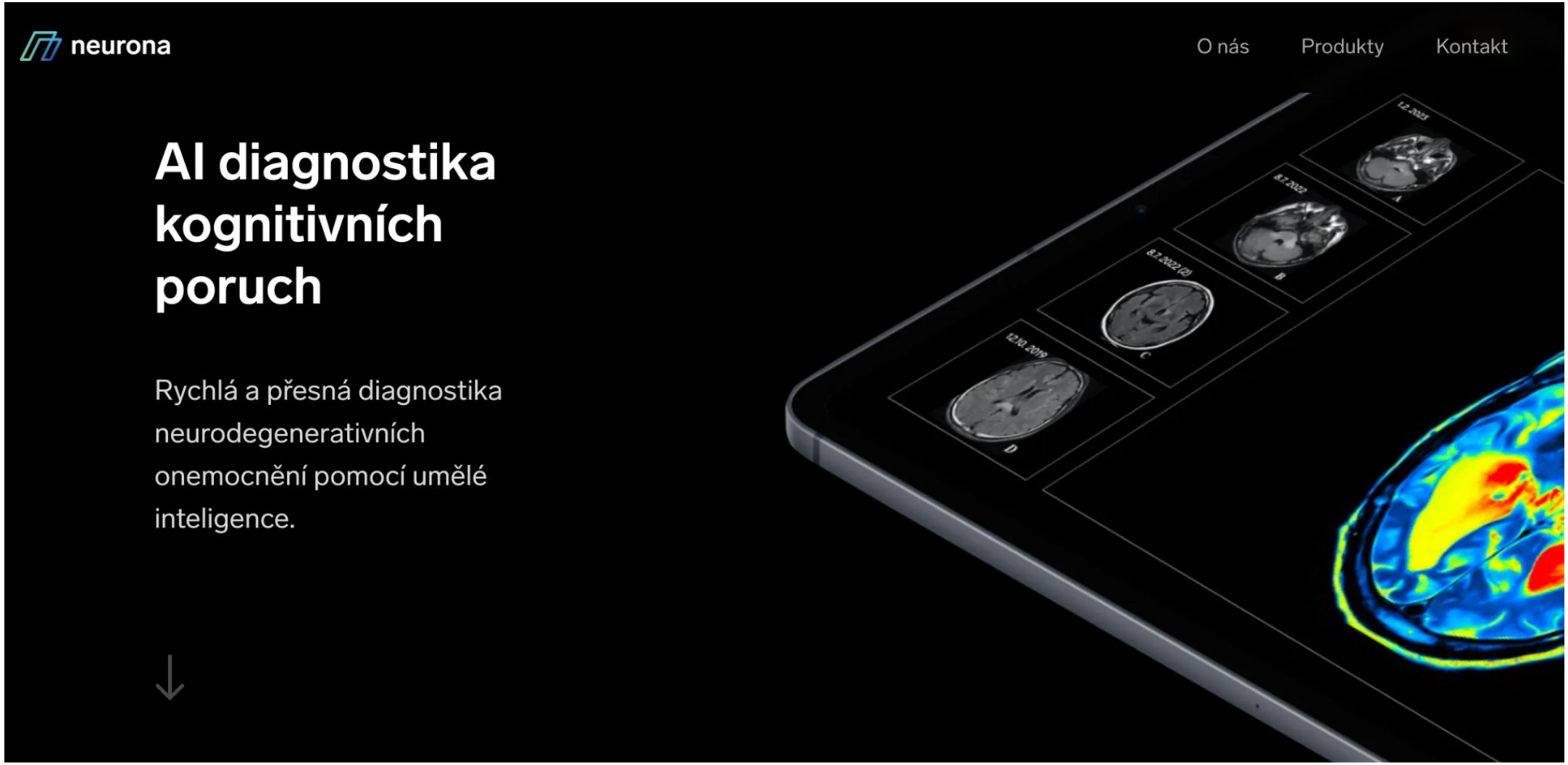


Vyzkoušet Aireen®

ISO 13485 CERTIFIED CE 2265

Aireen je první **CE-MDR IIb** certifikovaný zdravotnický prostředek, založený na umělé inteligenci, vyvinutý v ČR.

Neurona Lab


neurona
O nás
Produkty
Kontakt
AI diagnostika kognitivních poruch
Rychlá a přesná diagnostika neurodegenerativních onemocnění pomocí umělé inteligence.

Kardi AI

Kardi Ai

Pro uživatele Pro lékaře Pro partnery Klinické studie O nás

Kontrola rytmu Vašeho srdce

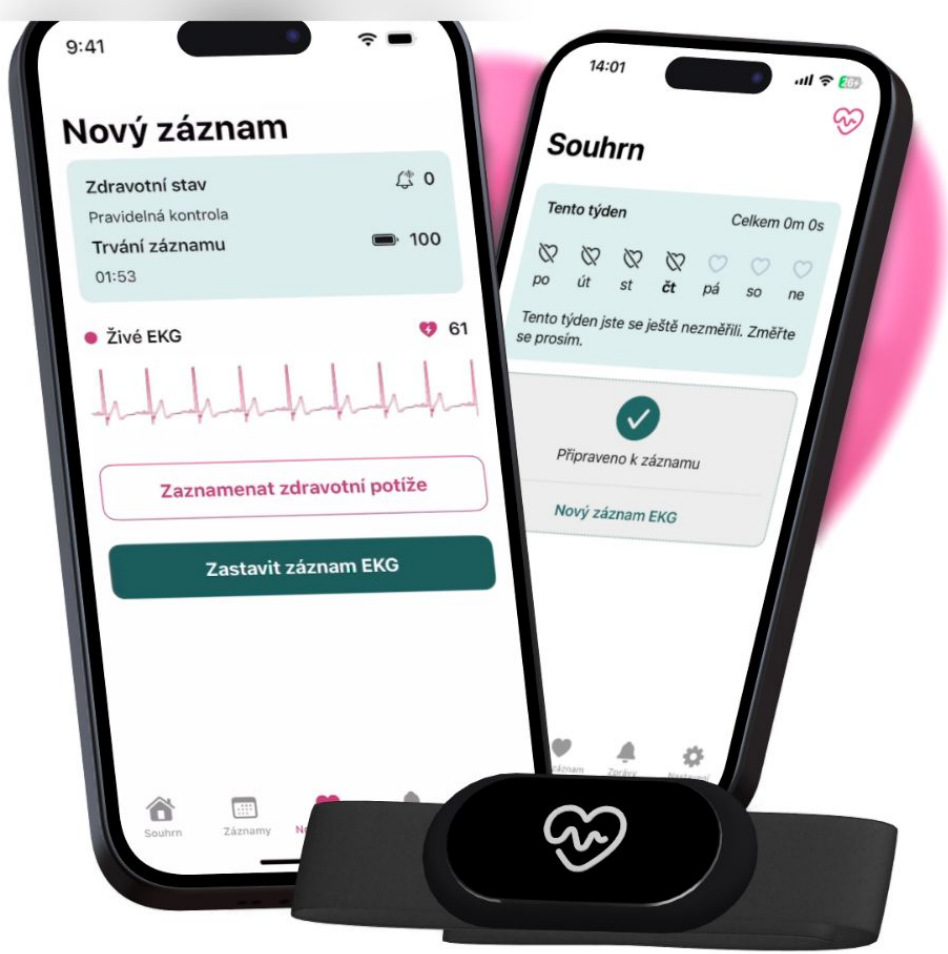
Kdykoliv, kdekoliv, neomezeně
a v naprostém pohodlí

Objednat MyKARDI

84097+ měření EKG

6196 dnů měření

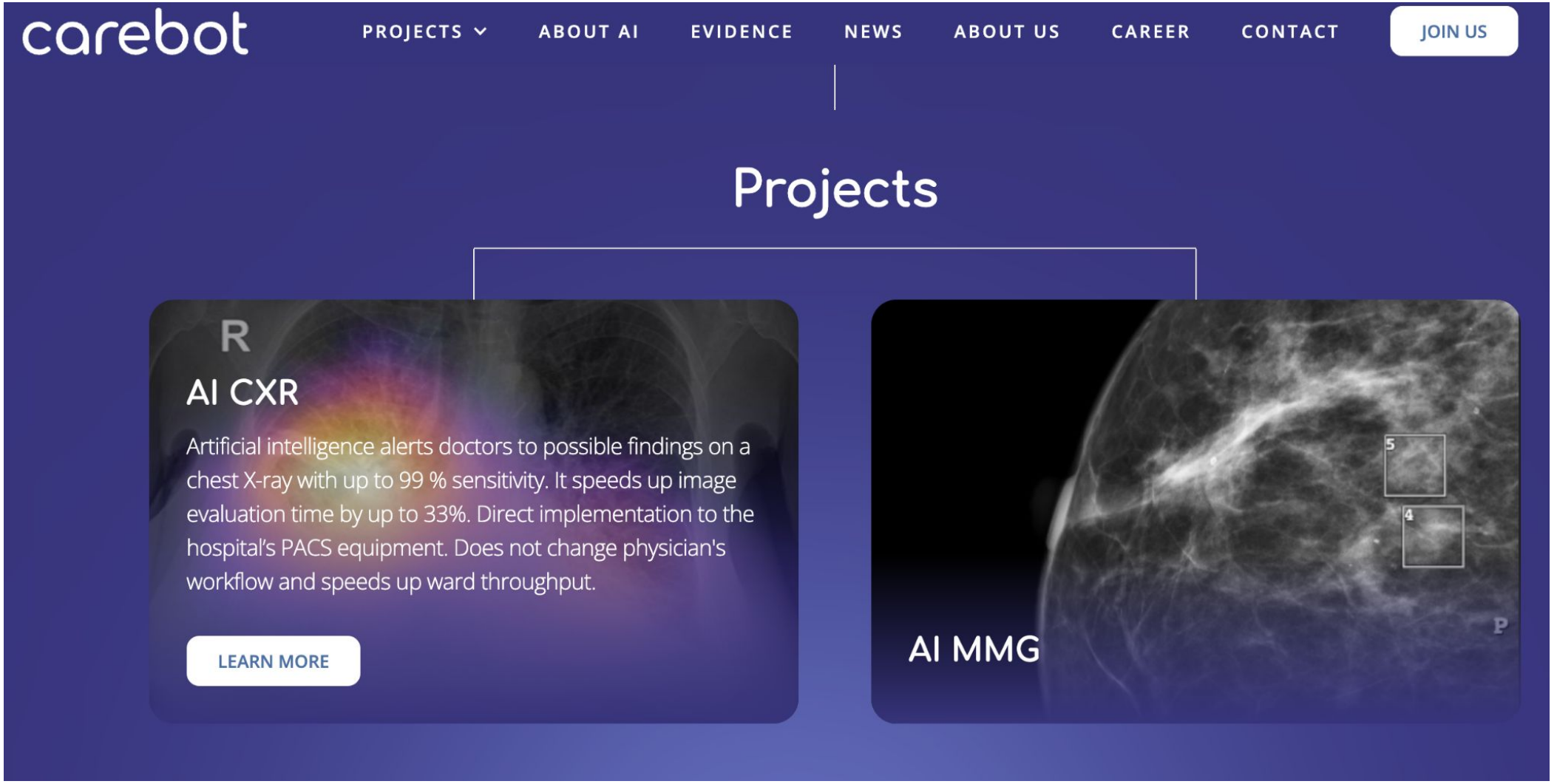
190+ zachráněných životů



MAIA Labs


MAIA
We help doctors to see everything!
MAIA is an intelligent endoscopy module, based on state of the art AI technology.
SEE MORE

Carebot


carebot
PROJECTS
ABOUT AI
EVIDENCE
NEWS
ABOUT US
CAREER
CONTACT
JOIN US
Projects
R
AI CXR
Artificial intelligence alerts doctors to possible findings on a chest X-ray with up to 99 % sensitivity. It speeds up image evaluation time by up to 33%. Direct implementation to the hospital's PACS equipment. Does not change physician's workflow and speeds up ward throughput.
LEARN MORE
AI MMG

Proč používat AI v medicíně?

Analýza velkého datasetu	**Rychlá a přesná diagnostika**	**Dostupnější screening**
Automatická anotace datasetu	**Automatické upozornění na abnormality**	**Personalizovaná léčba**
Efektivita a úspora nákladů	**24/7 Dostupnost**	**Telemedicína na dálku**

AI výzvy

- Data a Soukromí
- Etika a Odpovědnost
- Přesnost a Spolehlivost
- Integrace do Klinické Praxe
- Regulace a Standardizace

Budování neuronových sítí (Python)

- Klasifikace do dvou tříd
  - Jedna skrytá vrstva
  - Nelineární aktivační funkce
  - Cross-entropy chybová funkce
  - Forward and Back Propagation

Knihovny

- Numpy (matematické funkce)
- Sklearn (data mining + data analysis)
- Matplotlib (vykreslování)
- testCases (příklady)

```
# Package imports
import numpy as np
import matplotlib.pyplot as plt
from testCases_v2 import *
import sklearn
import sklearn.datasets
import sklearn.linear_model
from planar_utils import plot_decision_boundary, sigmoid, load_planar_dataset
```

Data

- X = features (x1, x2)
- Y = labels (red = 0, blue = 1)

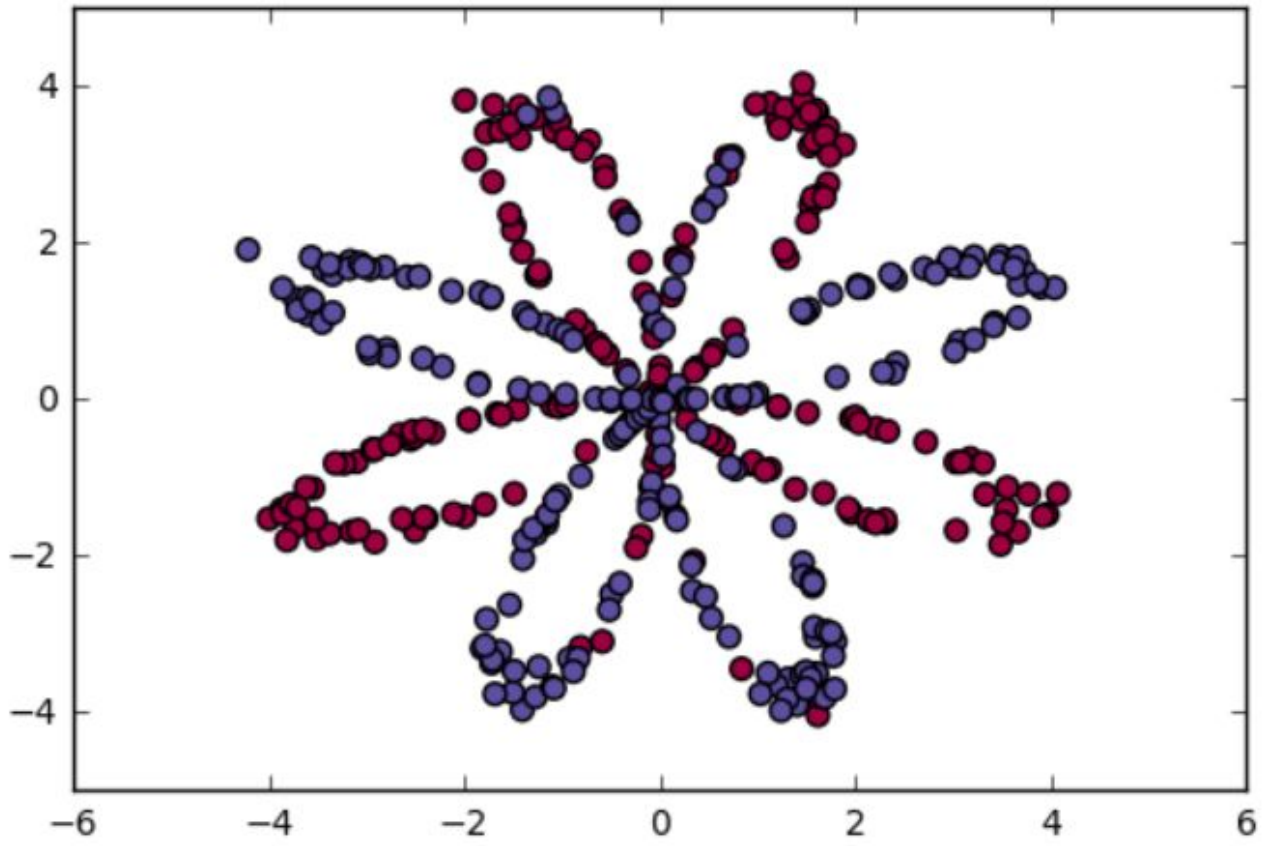
In [3]:

```
X, Y = load_planar_dataset()
print("X.shape:",X.shape,";Y.shape:",Y.shape)
```

```
X.shape: (2, 400) ;Y.shape: (1, 400)
```

In [4]:

```
# Visualize the data:
plt.scatter(X[0, :], X[1, :], c=Y, s=40, cmap=plt.cm.Spectral);
```



In [5]:

```
shape_X = X.shape
shape_Y = Y.shape
m = shape_X[1]  # training set size


print ('The shape of X is: ' + str(shape_X))
print ('The shape of Y is: ' + str(shape_Y))
print ('I have m = %d training examples!' % (m))
```

```
The shape of X is: (2, 400)
The shape of Y is: (1, 400)
I have m = 400 training examples!
```

Expected Output:

shape of X	(2, 400)
shape of Y	(1, 400)
m	400

Logistická regrese

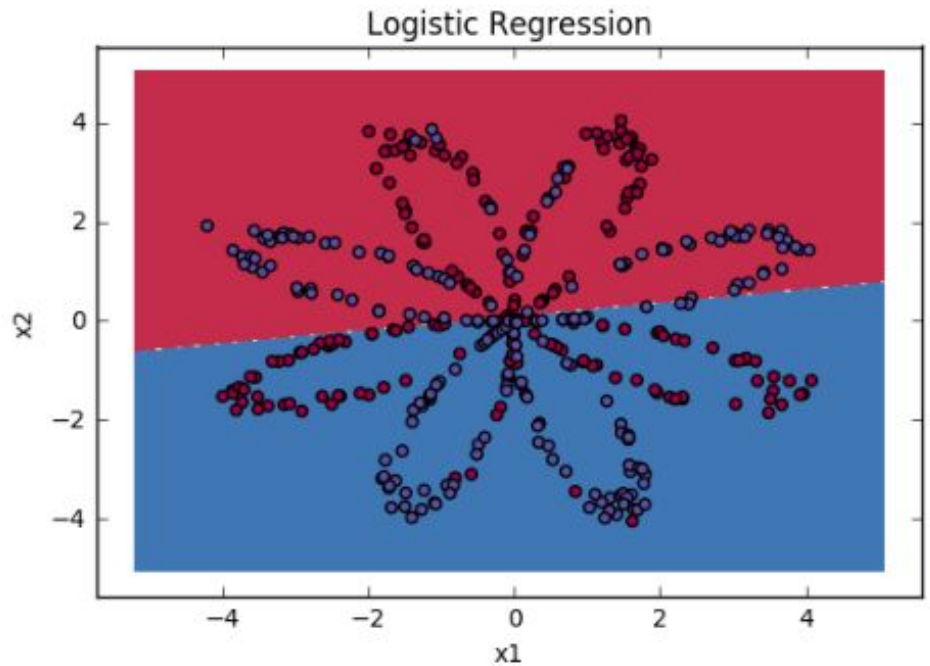
In [6]:

```
# Train the logistic regression classifier
clf = sklearn.linear_model.LogisticRegressionCV();
clf.fit(X.T, Y.T);
```

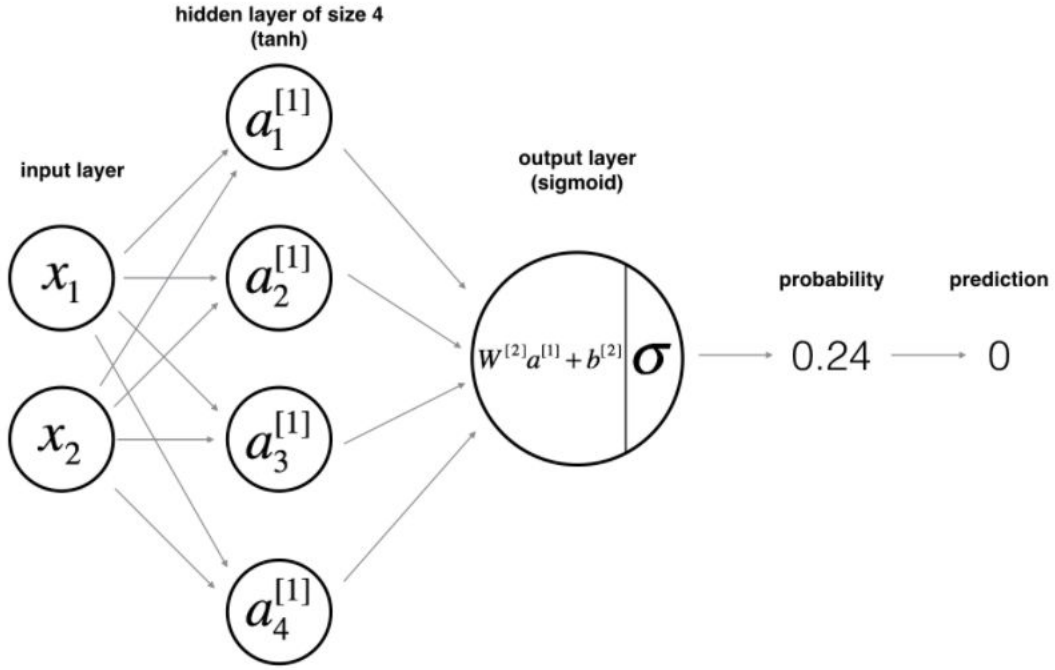
In [7]:

```
# Plot the decision boundary for logistic regression
plot_decision_boundary(lambda x: clf.predict(x), X, Y)
plt.title("Logistic Regression")

# Print accuracy
LR_predictions = clf.predict(X.T)
print ('Accuracy of logistic regression: %d ' % float((np.dot(Y,LR_predictions)
       '% ' + "(percentage of correctly labelled datapoints)")

print("Y.shape:",Y.shape,";LR_predictions.shape:",LR_predictions.shape)
correction_prediction=Y[0,:]==LR_predictions
print("correction_prediction.shape:",correction_prediction.shape)
acc=np.sum(correction_prediction)/m
print("acc:",acc)
```

```
Accuracy of logistic regression: 47 % (percentage of correctly label
led datapoints)
Y.shape: (1, 400) ;LR_predictions.shape: (400,)
correction_prediction.shape: (400,)
acc: 0.47
```



Neuronová sítě - model



$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1](i)}$$

$$a^{[1](i)} = \tanh(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2](i)}$$

$$\hat{y}^{(i)} = a^{[2](i)} = \sigma(z^{[2](i)})$$

$$y^{(i)}_{prediction} = \begin{cases} 1 & a^{[2](i)}>0.5 \\ 0 & \end{cases}$$

1. Definovat strukturu NN (input, skryté vrstvy)
2. Inicializovat parametry
3. Smyčka: forward propagation, vypočítat chybu, back propagation, upravení parametrů)

```
def layer_sizes(X, Y):
    """
    Arguments:
    X -- input dataset of shape (input size, number of examples)
    Y -- labels of shape (output size, number of examples)

    Returns:
    n_x -- the size of the input layer
    n_h -- the size of the hidden layer
    n_y -- the size of the output layer
    """
```

```
    n_x = X.shape[0] # size of input layer
    n_h = 4
    n_y = Y.shape[0] # size of output layer
```

```
    return (n_x, n_h, n_y)
```

```
X_assess, Y_assess = layer_sizes_test_case()
(n_x, n_h, n_y) = layer_sizes(X_assess, Y_assess)
print("The size of the input layer is: n_x = " + str(n_x))
print("The size of the hidden layer is: n_h = " + str(n_h))
print("The size of the output layer is: n_y = " + str(n_y))
```

```
The size of the input layer is: n_x = 5
The size of the hidden layer is: n_h = 4
The size of the output layer is: n_y = 2
```

n_x	5
n_h	4
n_y	2

```python
def initialize_parameters(n_x, n_h, n_y):
    """
    Argument:
    n_x -- size of the input layer
    n_h -- size of the hidden layer
    n_y -- size of the output layer

    Returns:
    params -- python dictionary containing your parameters:
                    W1 -- weight matrix of shape (n_h, n_x)
                    b1 -- bias vector of shape (n_h, 1)
                    W2 -- weight matrix of shape (n_y, n_h)
                    b2 -- bias vector of shape (n_y, 1)
    """

    W1 = np.random.randn(n_h, n_x) * 0.01
    b1 = np.zeros(shape=(n_h, 1))
    W2 = np.random.randn(n_y, n_h) * 0.01
    b2 = np.zeros(shape=(n_y, 1))

    assert (W1.shape == (n_h, n_x))
    assert (b1.shape == (n_h, 1))
    assert (W2.shape == (n_y, n_h))
    assert (b2.shape == (n_y, 1))

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}

    return parameters
```

```
n_x, n_h, n_y = initialize_parameters_test_case()

parameters = initialize_parameters(n_x, n_h, n_y)
print("W1 = " + str(parameters["W1"]))
print("b1 = " + str(parameters["b1"]))
print("W2 = " + str(parameters["W2"]))
print("b2 = " + str(parameters["b2"]))
```

```
W1 = [[-0.00416758 -0.00056267]
 [-0.02136196  0.01640271]
 [-0.01793436 -0.00841747]
 [ 0.00502881 -0.01245288]]
b1 = [[ 0.]
 [ 0.]
 [ 0.]
 [ 0.]]
W2 = [[-0.01057952 -0.00909008  0.00551454  0.02292208]]
b2 = [[ 0.]]
```

```python
def forward_propagation(X, parameters):
    """
    Argument:
    X -- input data of size (n_x, m)
    parameters -- python dictionary containing your parameters (output of initialization function)

    Returns:
    A2 -- The sigmoid output of the second activation
    cache -- a dictionary containing "Z1", "A1", "Z2" and "A2"
    """
    W1 = parameters['W1']
    b1 = parameters['b1']
    W2 = parameters['W2']
    b2 = parameters['b2']
    Z1 = np.dot(W1, X) + b1
    A1 = np.tanh(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)

    assert(A2.shape == (1, X.shape[1]))

    cache = {"Z1": Z1,
             "A1": A1,
             "Z2": Z2,
             "A2": A2}

    return A2, cache
```

```
X_assess, parameters = forward_propagation_test_case()

A2, cache = forward_propagation(X_assess, parameters)

# Note: we use the mean here just to make sure that your output matches ours.
print(np.mean(cache['Z1']), np.mean(cache['A1']), np.mean(cache['Z2']), np.mean(cache['A2']))
```

```
-0.000499755777742 -0.000496963353232 0.000438187450959 0.500109546852
```

```python
def compute_cost(A2, Y, parameters):
    """
    Computes the cross-entropy cost given in equation (13)

    Arguments:
    A2 -- The sigmoid output of the second activation, of shape (1, number of examples)
    Y -- "true" labels vector of shape (1, number of examples)
    parameters -- python dictionary containing your parameters W1, b1, W2 and b2

    Returns:
    cost -- cross-entropy cost given equation (13)
    """

    m = Y.shape[1] # number of example

    # Retrieve W1 and W2 from parameters
    W1 = parameters['W1']
    W2 = parameters['W2']

    # Compute the cross-entropy cost
    logprobs = np.multiply(np.log(A2), Y) + np.multiply((1 - Y), np.log(1 - A2))
    cost = - np.sum(logprobs) / m
    cost = np.squeeze(cost)     # makes sure cost is the dimension we expect.
                                # E.g., turns [[17]] into 17
    assert(isinstance(cost, float))

    return cost
```

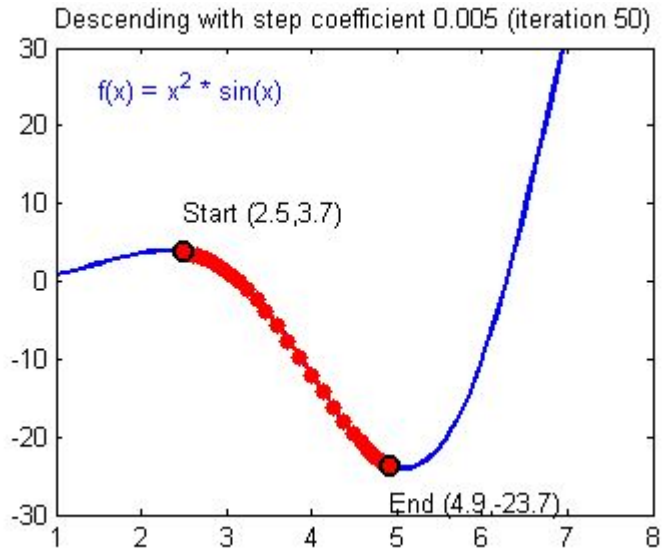
```python
A2, Y_assess, parameters = compute_cost_test_case()

print("cost = " + str(compute_cost(A2, Y_assess, parameters)))
```

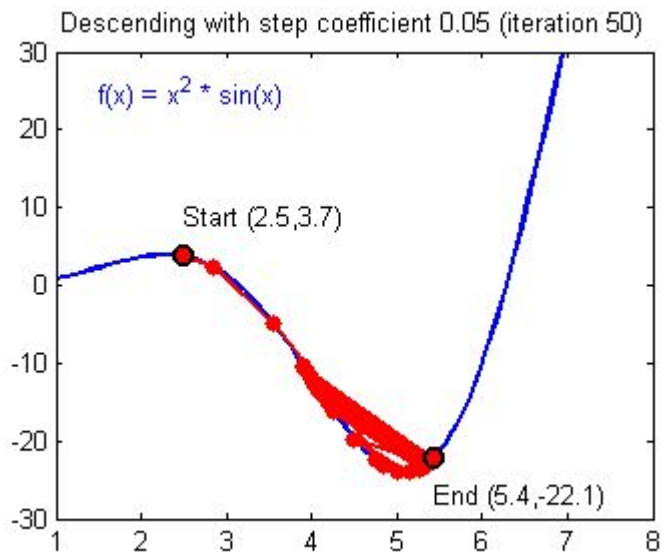
```
cost = 0.692919893776
```

```python
def backward_propagation(parameters, cache, X, Y):
    """
    Implement the backward propagation using the instructions above.

    Arguments:
    parameters -- python dictionary containing our parameters
    cache -- a dictionary containing "Z1", "A1", "Z2" and "A2".
    X -- input data of shape (2, number of examples)
    Y -- "true" labels vector of shape (1, number of examples)

    Returns:
    grads -- python dictionary containing your gradients with respect to different parameters
    """
    m = X.shape[1]

    # First, retrieve W1 and W2 from the dictionary "parameters".
    W1 = parameters['W1']
    W2 = parameters['W2']

    # Retrieve also A1 and A2 from dictionary "cache".
    A1 = cache['A1']
    A2 = cache['A2']

    # Backward propagation: calculate dW1, db1, dW2, db2.
    dZ2= A2 - Y
    dW2 = (1 / m) * np.dot(dZ2, A1.T)
    db2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True)
    dZ1 = np.multiply(np.dot(W2.T, dZ2), 1 - np.power(A1, 2))
    dW1 = (1 / m) * np.dot(dZ1, X.T)
    db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)

    grads = {"dW1": dW1,
             "db1": db1,
             "dW2": dW2,
             "db2": db2}

    return grads
```

```
parameters, cache, X_assess, Y_assess = backward_propagation_test_case()

grads = backward_propagation(parameters, cache, X_assess, Y_assess)
print ("dW1 = "+ str(grads["dW1"]))
print ("db1 = "+ str(grads["db1"]))
print ("dW2 = "+ str(grads["dW2"]))
print ("db2 = "+ str(grads["db2"]))
```

```
dW1 = [[ 0.01018708 -0.00708701]
 [ 0.00873447 -0.0060768 ]
 [-0.00530847  0.00369379]
 [-0.02206365  0.01535126]]
db1 = [[-0.00069728]
 [-0.00060606]
 [ 0.000364  ]
 [ 0.00151207]]
dW2 = [[ 0.00363613  0.03153604  0.01162914 -0.01318316]]
db2 = [[ 0.06589489]]
```





```python
def update_parameters(parameters, grads, learning_rate=1.2):
    """
    Updates parameters using the gradient descent update rule given above

    Arguments:
    parameters -- python dictionary containing your parameters
    grads -- python dictionary containing your gradients

    Returns:
    parameters -- python dictionary containing your updated parameters
    """
    # Retrieve each parameter from the dictionary "parameters"
    W1 = parameters['W1']
    b1 = parameters['b1']
    W2 = parameters['W2']
    b2 = parameters['b2']

    # Retrieve each gradient from the dictionary "grads"
    dW1 = grads['dW1']
    db1 = grads['db1']
    dW2 = grads['dW2']
    db2 = grads['db2']

    # Update rule for each parameter
    W1 = W1 - learning_rate * dW1
    b1 = b1 - learning_rate * db1
    W2 = W2 - learning_rate * dW2
    b2 = b2 - learning_rate * db2

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}

    return parameters
```

```
parameters, grads = update_parameters_test_case()
parameters = update_parameters(parameters, grads)

print("W1 = " + str(parameters["W1"]))
print("b1 = " + str(parameters["b1"]))
print("W2 = " + str(parameters["W2"]))
print("b2 = " + str(parameters["b2"]))
```

```
W1 = [[-0.00643025  0.01936718]
 [-0.02410458  0.03978052]
 [-0.01653973 -0.02096177]
 [ 0.01046864 -0.05990141]]
b1 = [[ -1.02420756e-06]
 [  1.27373948e-05]
 [  8.32996807e-07]
 [ -3.20136836e-06]]
W2 = [[-0.01041081 -0.04463285  0.01758031  0.04747113]]
b2 = [[ 0.00010457]]
```

```python
def nn_model(X, Y, n_h, num_iterations=10000, print_cost=False):
    """
    Arguments:
    X -- dataset of shape (2, number of examples)
    Y -- labels of shape (1, number of examples)
    n_h -- size of the hidden layer
    num_iterations -- Number of iterations in gradient descent loop
    print_cost -- if True, print the cost every 1000 iterations

    Returns:
    parameters -- parameters learnt by the model. They can then be used to predict.
    """

    np.random.seed(3)
    n_x = layer_sizes(X, Y)[0]
    n_y = layer_sizes(X, Y)[2]

    # Initialize parameters, then retrieve W1, b1, W2, b2. Inputs: "n_x, n_h, n_y". Outputs = "W1, b1, W2, b2, parameters".

    parameters = initialize_parameters(n_x, n_h, n_y)
    W1 = parameters['W1']
    b1 = parameters['b1']
    W2 = parameters['W2']
    b2 = parameters['b2']

    # Loop (gradient descent)

    for i in range(0, num_iterations):

        # Forward propagation. Inputs: "X, parameters". Outputs: "A2, cache".
        A2, cache = forward_propagation(X, parameters)

        # Cost function. Inputs: "A2, Y, parameters". Outputs: "cost".
        cost = compute_cost(A2, Y, parameters)

        # Backpropagation. Inputs: "parameters, cache, X, Y". Outputs: "grads".
        grads = backward_propagation(parameters, cache, X, Y)

        # Gradient descent parameter update. Inputs: "parameters, grads". Outputs: "parameters".
        parameters = update_parameters(parameters, grads)

        # Print the cost every 1000 iterations
        if print_cost and i % 1000 == 0:
            print ("Cost after iteration %i: %f" % (i, cost))

    return parameters
```

```
def nn_model(X, Y, n_h, num_iterations=10000, print_cost=False):
    """
X_assess, Y_assess = nn_model_test_case()

parameters = nn_model(X_assess, Y_assess, 4, num_iterations=10000, print_cost=False)
print("W1 = " + str(parameters["W1"]))
print("b1 = " + str(parameters["b1"]))
print("W2 = " + str(parameters["W2"]))
print("b2 = " + str(parameters["b2"]))
```

```
W1 = [[-4.18494056  5.33220609]
 [-7.52989382  1.24306181]
 [-4.1929459   5.32632331]
 [ 7.52983719 -1.24309422]]
b1 = [[ 2.32926819]
 [ 3.79458998]
 [ 2.33002577]
 [-3.79468846]]
W2 = [[-6033.83672146 -6008.12980822 -6033.10095287  6008.06637269]]
b2 = [[-52.66607724]]
```

```python
def predict(parameters, X):
    """
    Using the learned parameters, predicts a class for each example in X

    Arguments:
    parameters -- python dictionary containing your parameters
    X -- input data of size (n_x, m)

    Returns
    predictions -- vector of predictions of our model (red: 0 / blue: 1)
    """

    # Computes probabilities using forward propagation, and classifies to 0/1 using 0.5 as the threshold.
    A2, cache = forward_propagation(X, parameters)
    predictions = np.round(A2)

    return predictions
```

```python
parameters, X_assess = predict_test_case()

predictions = predict(parameters, X_assess)
print("predictions mean = " + str(np.mean(predictions)))
```

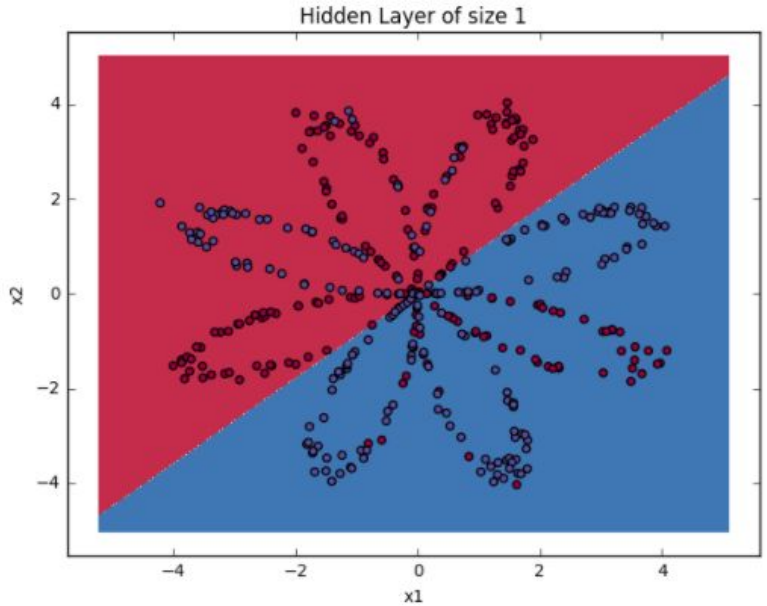
```
predictions mean = 0.666666666667
```

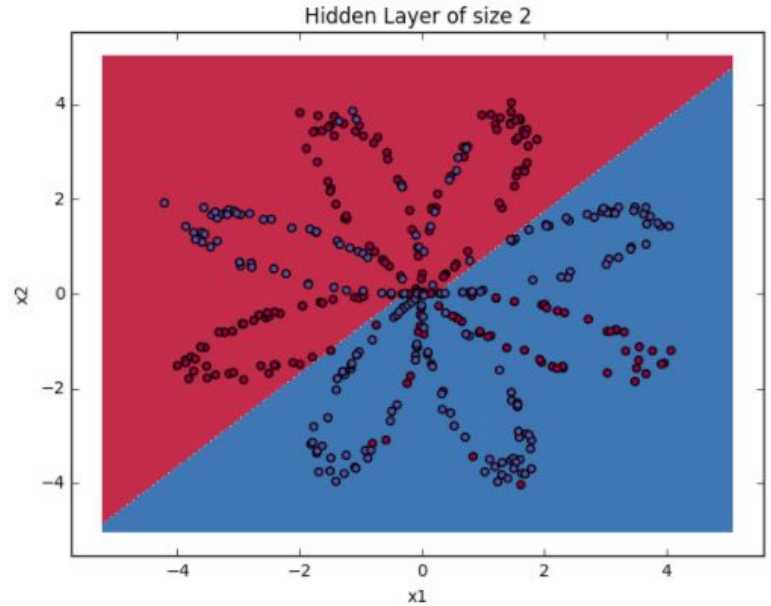
```
# Build a model with a n_h-dimensional hidden layer
parameters = nn_model(X, Y, n_h = 4, num_iterations=10000, print_cost=True)

# Plot the decision boundary
plot_decision_boundary(lambda x: predict(parameters, x.T), X, Y)
plt.title("Decision Boundary for hidden layer size " + str(4))
```

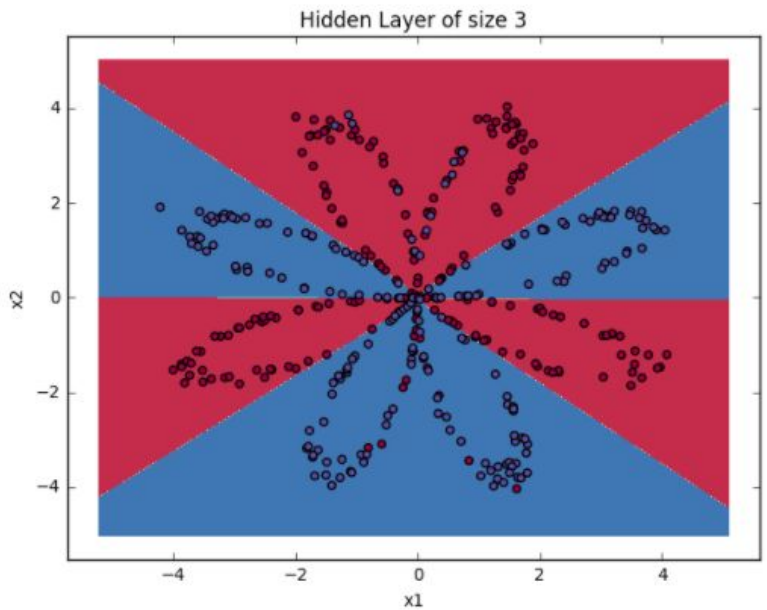
```
Cost after iteration 0: 0.693048
Cost after iteration 1000: 0.288083
Cost after iteration 2000: 0.254385
Cost after iteration 3000: 0.233864
Cost after iteration 4000: 0.226792
Cost after iteration 5000: 0.222644
Cost after iteration 6000: 0.219731
Cost after iteration 7000: 0.217504
Cost after iteration 8000: 0.219454
Cost after iteration 9000: 0.218607
```

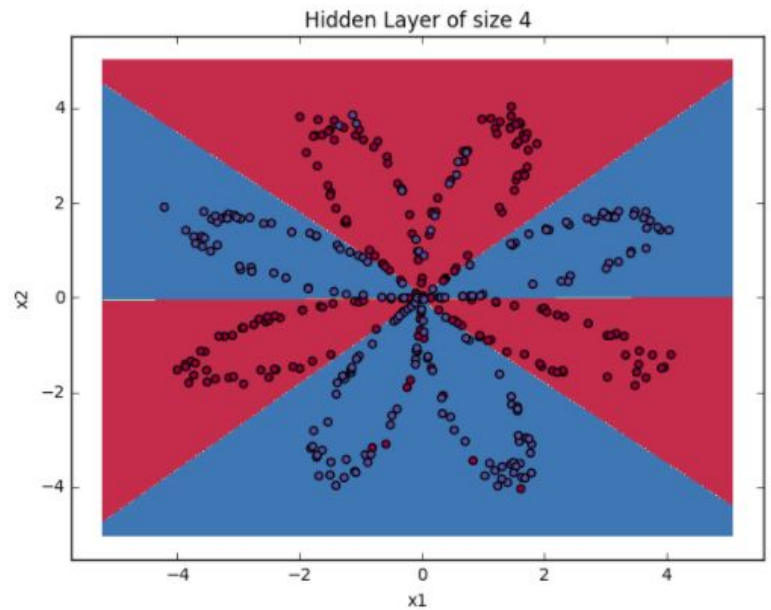
```
# Print accuracy
predictions = predict(parameters, X)
print ('Accuracy: %d' % float((np.dot(Y, predictions.T) + np.dot(1 - Y, 1 - predictions.T)) / float(Y.size) * 100) + '%')
```

```
Accuracy: 90%
```


Hidden Layer of size 1
x1
x2


Hidden Layer of size 2
x1
x2


Hidden Layer of size 3
x1
x2


Hidden Layer of size 4
x1
x2