

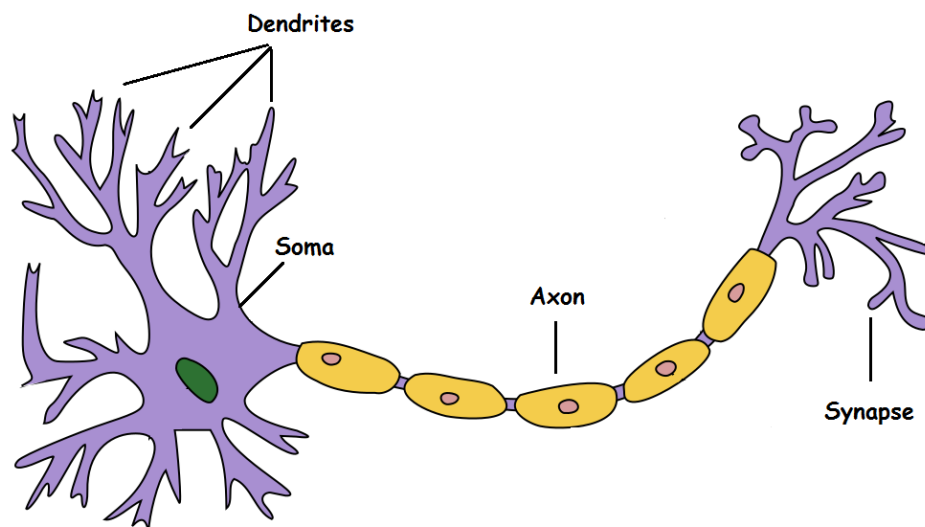
Образна диагностика с използването на невронни мрежи

I. Кратка теоретична обосновка

I.1. Структура на невронната мрежа

Изкуствените невронни мрежи са проектирани по модел на естествените невронни мрежи и изпълнението на функции на човешкия мозък.

Невронната мрежа (НН) е изградена от краен брой неврони, свързани помежду си по определен модел с цел изпълнението на конкретна задача. На фиг. 1 е илюстриран модел на неврон, който се състои от тяло, входни връзки (дендриди), изход (аксон), изходящи връзки (синапси).



Фиг. 1 Модел на неврон

Процедурата по обработка на входния сигнал за изпълнението на дадена функционалност започва с подаването на сигнали от входните връзки. Те се предават към тялото на неврона, където в зависимост от сумарната им стойност неврона преминава или не във възбудено състояние. Така генерирания сигнал към аксона на неврона се предава към други неврони от мрежата чрез синапсите им.

За постигането на висока ефективност при работата с невронни мрежи информацията се обработва на базата на обучение като еквивалент на натрупания опит в естествените невронни мрежи. В резултат на обучението

се запомня топологията на невронната мрежа и нейните количествени характеристики.

Определянето характеристиките на невронните мрежи зависи от параметри като: характеристики на неврона, HE_i , топология на мрежата, състояние на активация на мрежата, модели на невронните мрежи, обработващ алгоритъм, организация на обработката в мрежата.

На базата на стойност на активация $a_i(t)$ за всеки отделен неврон се формира изходна стойност чрез функцията $f_i(a_i(t))$:

$$o_i(t) = f_i(a_i(t)), \quad (1)$$

Получената стойност $o_i(t)$ се предава към невроните от мрежата чрез връзките между тях, които се характеризират с тегло w_{ji} . Теглото е реално число, при умножението на което със стойността на изходния сигнал определят въздействието на неврона HE_j върху неврона HE_i . Всички въздействия, постъпили от входовете на даден неврон се сумират:

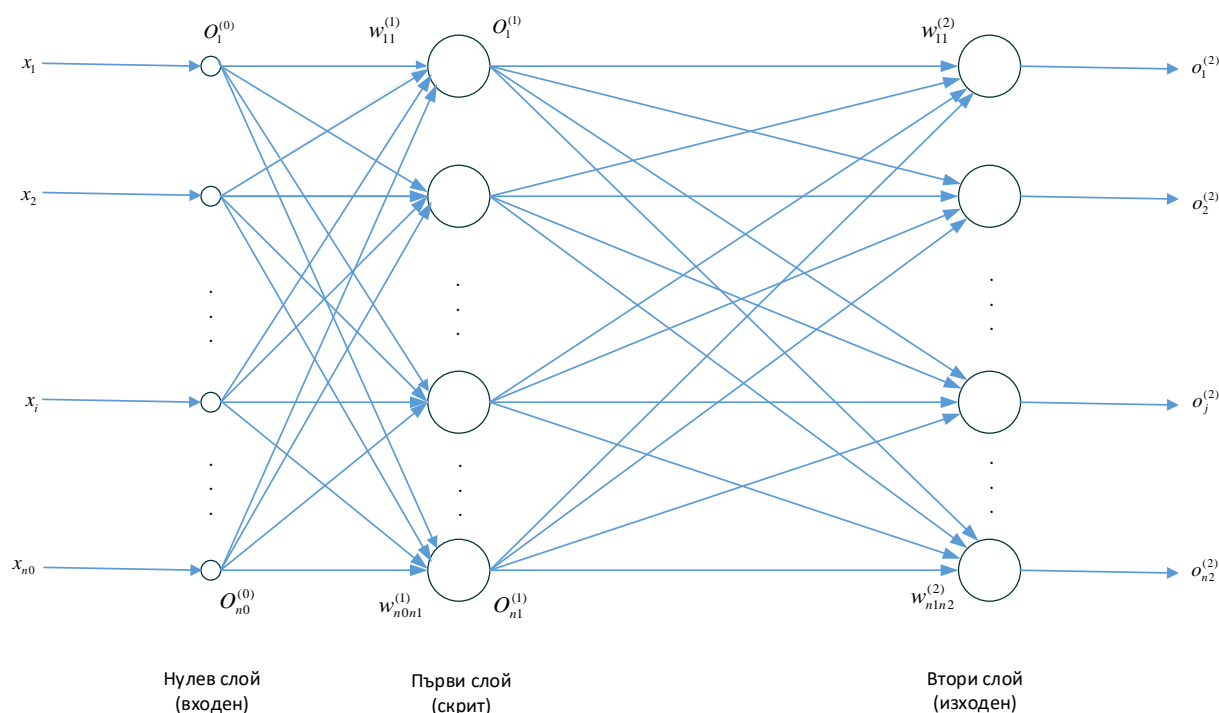
$$net_i = \sum_j w_{ji} o_j, \quad (2)$$

На базата на net_i се формира новата стойност на активация за даден неврон:

$$a_i(t+1) = F(net_i(t), a_i(t)), \quad (3)$$

1.2. Топология на невронна мрежа

Невронната мрежа има йерархична структура, която се определя от няколко слоя. Налице са входни неврони, вътрешни (скрити) и външни (фиг. 2). Входните сигнали, постъпващи чрез входните неврони се приемат от системи, интергирани в невронната мрежа (сензори). След обработка на сигнала от мрежата, полученият резултат се извежда от нея чрез изходните неврони. Организацията по обработка на информацията се изпълнява по модела: отдолу-нагоре, отгоре-надолу или интерактивно. При първия модел невроните от дадено ниво l не могат да въздействат на неврони от по-ниско ниво от ниво l . При втория модел информацията се разпространява от горните невронни нива към по-ниските, а при интерактивните нива се комбинират и двата описани модела, което се прилага за организирането на обратна връзка.



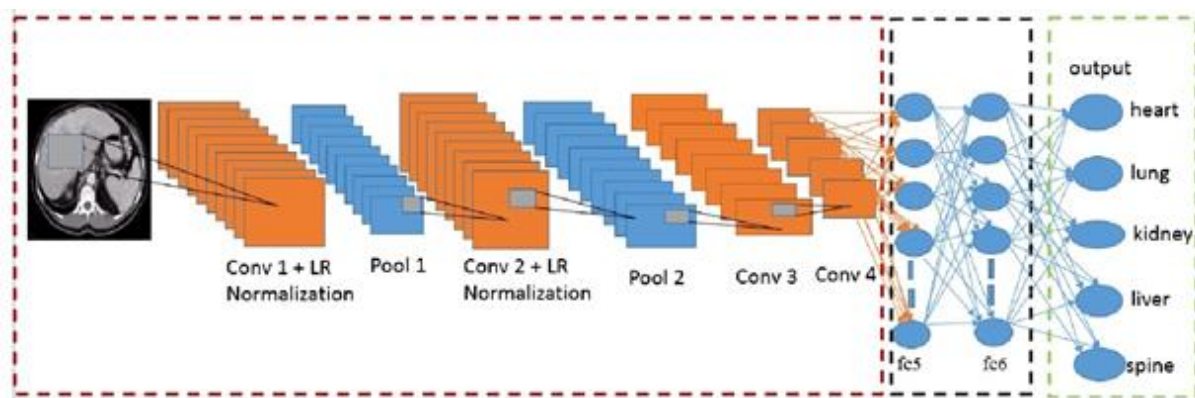
Фиг. 2 Топология на невронна мрежа

Обработвания алгоритъм преминава през последователен процес на инициализация, установяване, обучение и същностна обработка на входните данни. В етапа на инициализация се подават входни данни към входния слой неврони, след което те се предават по невронната мрежа и се изчислява стойността на активация на всеки неврон до достигането до изходния слой. За постигането на висока ефективност на мрежата и търсеното решение на поставената задача се извършва процес на обучение. То може да протече с учител или без учител [1].

1.3. Конволюционни невронни мрежи

За работа с изображения се използват конволюционни невронни мрежи (КНМ), тъй като без да се стига до пренапрежение те могат да обработват двумерни сигнали с по-голяма размерност от $32 \times 32 \times 3$. Посочените стойности отразяват броя пиксели по параметри ширина, височина и дълбочина съответно.

В КНМ невроните от един слой се съединяват само с един малък участък от предходния слой вместо с всички неврони по един напълно свързан начин. Освен това, крайният изходен слой има размерности $1 \times 1 \times 10$, тъй като в края на своята работа архитектурата на КНМ ще намали пълното изображение до един единствен вектор от оценки на класове, подреден по размерност „дълбочина“. На фиг. 3 [2] е изобразена примерна архитектура на КНМ.



Фиг. 3 Архитектура на конволюционна невронна мрежа

Конволюционните невронни (КНМ) мрежи се състоят от входен, конволюционен, пулингов и напълно свързан слой. За изображение с параметри $[32 \times 32 \times 3]$ входния слой поема пикселните стойности на изображението с ширина 32, височина 32 и с три цветови канала R, G, B. Конволюционният слой изчислява изхода на неврони, които са свързани с част от входове (локална област) от входния слой. Всеки неврон от конволюционния слой изчислява скалярно произведение между неговите тегла и входните стойности от локалната област, с която той е свързан. Пулонговият слой изпълнява операция по намаляване на размера по пространствените размерности (ширина и височина), което води до изходния обем от типа на $[16 \times 16 \times 3]$. Напълно свързаният слой изчислява оценки на класове, което води до изходния обем с размер $[1 \times 1 \times 10]$, където всяко от 10-те числа съответства на оценката на някой клас от 10 категории. Както и при обикновени НМ всеки неврон от този слой е свързан с всички неврони от предишния обем (слой).

I.4. Tensorflow

TensorFlow е платформа („двигател“) за реализация на алгоритми за машинно обучение и имплементирането за изпълнение на такива алгоритми. Изчисленията, извършени чрез TensorFlow могат да се изпълняват на хетерогенни системи вариращи от мобилни устройства (телефони, таблети) до разпределени системи от стотици машини и хиляди компютърни устройства като GPU карти. Системата се отличава с гъвкавост и може да се използва за възпроизвеждането на разнообразни алгоритми включително и обучение на алгоритми за модели на дълбоки невронни мрежи. Намира приложение в провеждането на научни експерименти, в производството в сферата на компютърните технологии за целите на разпознаването на реч, компютърно зрение, роботика, извличане на информация, превод от един език на друг, извличане на информация с географска насоченост и др.

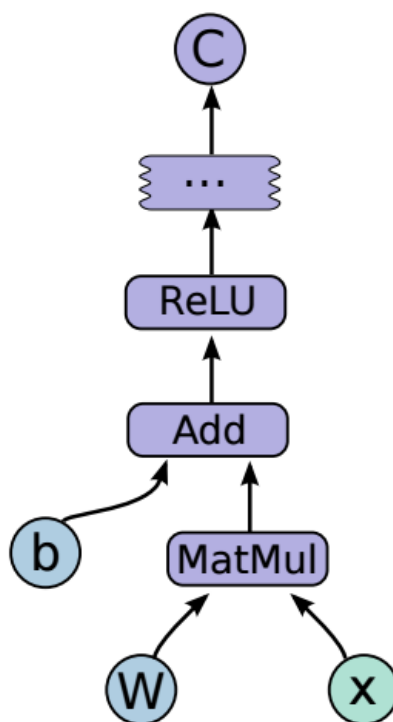
Изчисленията на TensorFlow се описват чрез насочен граф, съставен от набор възли. Графът представя изчислението на потока от данни с разширения, позволяващи на някои видове възли да поддържат и актуализират устойчиво състояние и за разклоняване и цикли за управление в граф. Обикновено клиентите конструират изчислителен граф, използващ един C++ или Python.). На фиг. 4 е представена част от програмна реализация, изпълнена на TensorFlow, а на фиг. 5 е представен графа, изобразяващ резултата, получен от извършените изчисления.

```
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))          # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784,100],-1,1))  = # 784x100 matrix w/rnd vals
x = tf.placeholder(name="x")              # Placeholder for input
relu = tf.nn.relu(tf.matmul(W, x) + b)    # Relu(Wx+b)
C = [..]                                  # Cost computed as a function of Relu

s = tf.Session()
for step in xrange(0, 10):
    input = ...construct 100-D input array ... # Create 100-d vector for input
    result = s.run(C, feed_dict={x: input})    # Fetch cost, feeding x=input
    print step, result
```

Фиг. 4: Програмна реализация, изпълнена чрез TensorFlow



Фиг. 5 Граф изобразяващ резултата от програмната реализация на фиг. 4

В графа на TensorFlow всеки възел има нула или повече входове и нула или повече изходи и представлява инстанция на операция. Стойностите, които протичат по нормалните ръбове на графа (от изходи към входове), са тензори, масиви с произволна размерност, при които типът на основния елемент е определен или изведен по време на изграждане на граф. В графа могат да съществуват и специални ръбове, наречени контролни зависимости: по тези ръбове не се движат данни, но те показват, че изходният възел за контролната зависимост трябва да приключи изпълнението, преди да започне да се изпълнява целеви възел за контролната зависимост.

Операции и ядра

Всяка операция има име и представлява абстрактно изчисление (напр. „Умножение на матрицата“ или „добавяне“). Операцията може да има атрибути и всички атрибути трябва да бъдат предоставени или изведени по време на генериране на графа, за да се създаде възел за извършване на операцията. Една обща употреба на атрибути е да се правят операции полиморфично да се извършват различни типове тензорни елемент. Ядрото (kernel) е конкретна реализация на операция, която може да се изпълнява на

определен тип устройство (напр. CPU or GPU). Двоичният TensorFlow дефинира наборите от операции и ядра, достъпни чрез механизъм за регистрация, и този набор може да бъде разширен чрез свързване в допълнителни операции и / или дефиниции / регистрации на ядрото.

Сесии

Програмите `SessionsClients` взаимодействат със системата TensorFlow чрез създаване на сесии. За да се създаде изчислителен граф, интерфейсът на сесията поддържа метод на разширяване за увеличаване на текущия граф, управляван от сесията, с допълнителни възли и ръбове (първоначалния граф, когато се създава сесия, е празен). Другата основна операция, поддържана от интерфейса на сесията, е `Run`, която приема набор от изходни имена, които трябва да бъдат изчислени, както и незадължителен набор от тензори, които ще бъдат въведени в графа вместо определени изходи на възли. Използвайки аргументите за изпълнение, реализацията на TensorFlow може да изчисли преходното затваряне на всички възли, които трябва да бъдат изпълнени, за да се изчислят изходните данни, които са били поискани, и след това може да организира изпълнението на съответните възли в ред, който зачита техните зависимости. Повечето от приложенията на TensorFlow създават сесия с графа веднъж и след това изпълняват пълния граф или няколко отделни подграфа хиляди или милиони пъти чрез стартиращи обаждания.

Променливи

В повечето изчисления графът се изпълнява многократно. Повечето тензори не оцеляват след еднократно изпълнение на граф. Въпреки това, `Variable` е специален вид операция, която връща управлението към тензор, който се запазва при изпълнение на графа. За приложения за машинно обучение на TensorFlow, параметрите на модела обикновено се съхраняват в тензори, които се актуализират като част от графа за обучение на модела.

I.5. Keras

Keras е приложение за невронни мрежи от високо ниво, създадено на програмния език Python. То може да се изпълнява в комбинация с платформите за дълбоко обучение TensorFlow или Theano като по подразбиране работи с първата от двете платформи. Чрез него се програмно се реализира модел на конволюционна или рекурентна невронна мрежа. Той е подходящ за работа с тях поотделно или в комбинация.

За успешното протичане на Keras симулациите е необходима предварителната инсталацията на TensorFlow за превода на Keras инструкциите на ниско ниво. Основната поддръжка на Keras е Google. В TensorFlow приложението Keras се представя като `tf.keras`.

Модела Keras се характеризира с два режима: режим на обучение и режим на тестване. По време на тестването всички процесите по регуларизация се изключват. Загубите при обучение представляват средноаритметичното от загубите на обучаемите данни. Поради това, че модела се променя във времето, загубите върху първите обучаеми бройки данни са по-високи отколкото при последните.

II. Експериментална постановка

II.1. Архитектура на използваната CNN

Целта на настоящето упражнение е чрез конволюционна невронна мрежа да се извърши класификация и оценка на серия от изображения, от тестова база данни от изображения за целите на медицината. За изпълнението на целта са предвидени задачи за симулиране и оценка на ефективността на невронната мрежа с използването на две различни мерки.

Използваният модел на конволюционна невронна мрежа за класифициране на изображения използва архитектурата на конволюционна невронна мрежа, изобразена на фиг. 3. Тя се характеризира с конволюционни нива, пулингovi нива и напълно свързани нива.

Първото ниво е конволюционно и съдържа 32 филтъра. Второто е пулинг ниво, което се използва за намаляване размерността. Третото ниво от архитектурата също е конволюционно с 64 филтъра с размер 3×3 , последвано от пулинг ниво с прозорец 2×2 . Първите нива с по-нисък брой филтри се обучава на по простите характеристики на изображенията, а по-дълбоките нива на мрежата се обучават на по-сложните характеристики.

Следващите два слоя от архитектурния модел са от конволюционен тип със същия размер филтър, но с нарастващ брой на филтрите: 128 и 256. За преобразуването на двумерен сигнал в едномерен вектор се използва „flatten“ слой, което се изпълнява преди добавянето на напълно свързани нива. На следващия етап от обработката на информацията архитектурата използва слой при който 50% от невроните на случаен принцип се изключват с цел предотвратяването на пренапрежение. Тези слоеве се добавят след всеки от напълно свързаните слоеве преди извеждането на резултата и намаляват времето за обучение.

Следващия слой е напълно свързан и се състои от 128 неврона. След него се намира слой от същия тип като предходния, при който 50% от избрани на случаен принцип неврони се изключват. Той се състои от 128 неврона. Предпоследният слой в архитектурата е отново е напълно свързан и има 128 неврона. Последният от всички извежда резултата и представлява напълно свързан слой, чиито брой неврони е равен на броя класове.

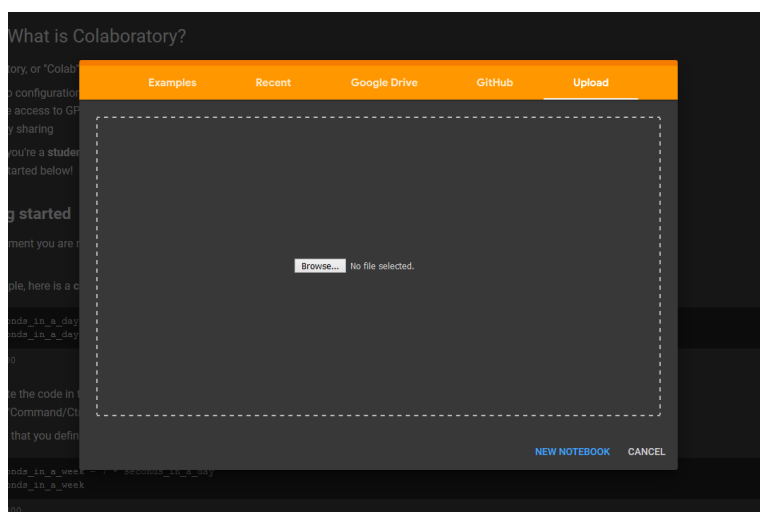
II.2. Указания за работа с експерименталната постановка

1. Стартира се посочената хипервръзка:

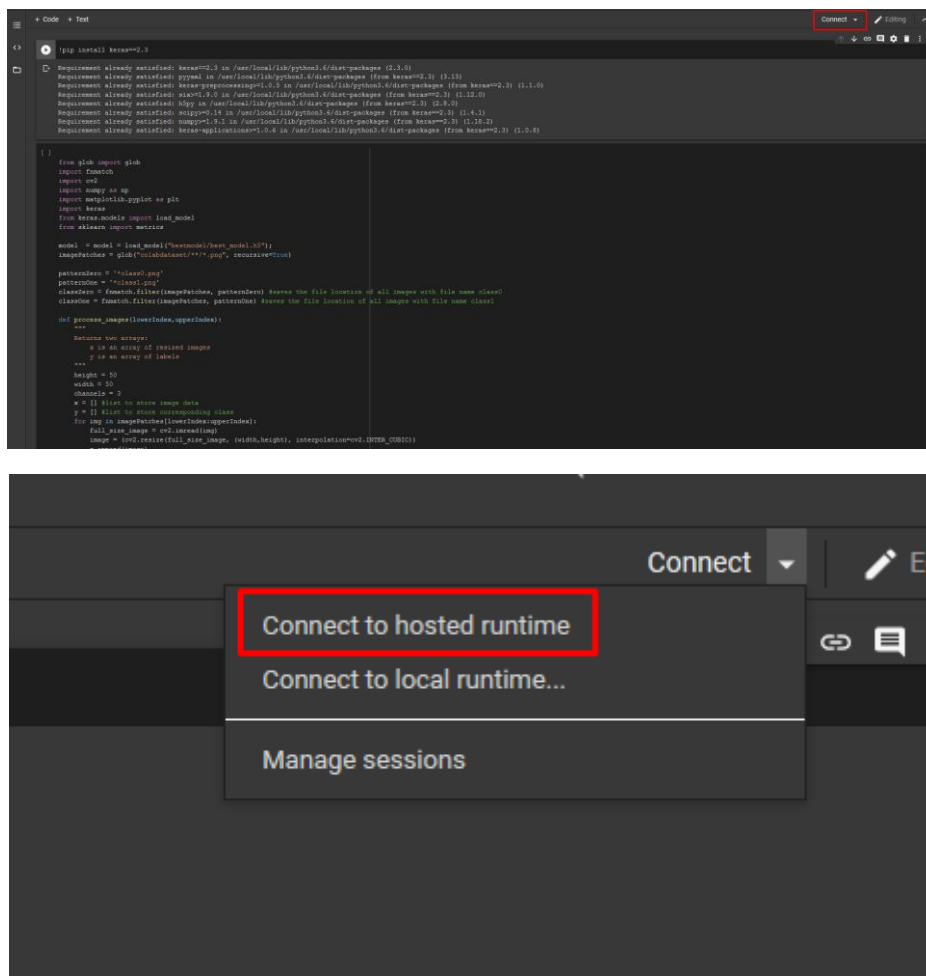
<https://colab.research.google.com/notebooks/intro.ipynb#recent=true/>

2. Зареждане програмната реализация на използвания класификационен модел:

А) в изведения прозорец се избира раздела Upload и чрез Browse се избира и зарежда файла „BioInformaticsNotebook.ipynb“ с реализация на класификационния модел:

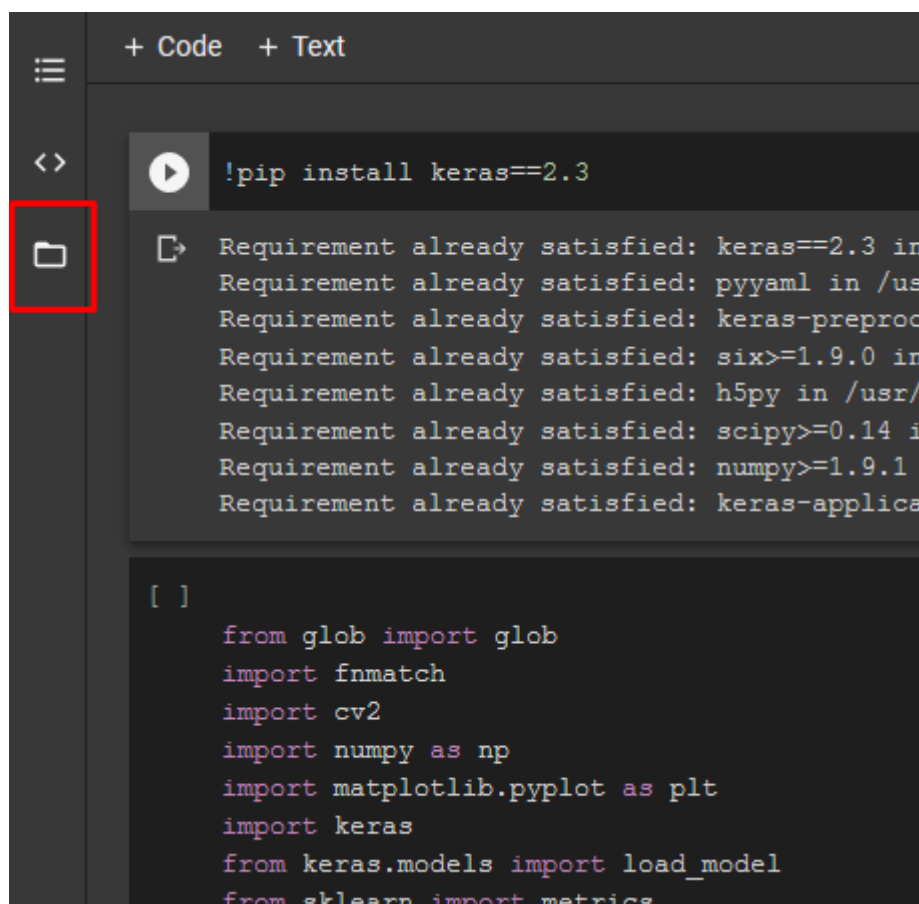


В) в заредения прозорец от менюто Connect, се избира командата Connect to hosted runtime:

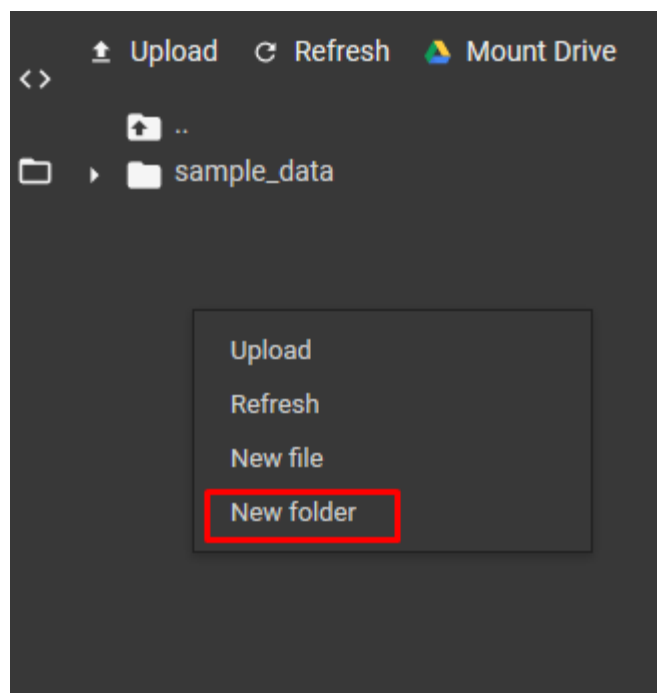


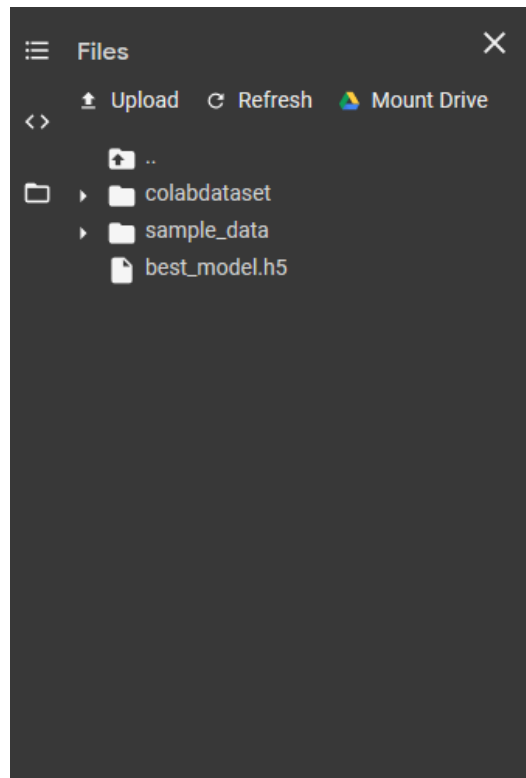
3. Зареждане на тестовата база данни от изображения:

- А) в заредения прозорец се избира иконата за папка, която се намира в менюто, разположено в ляво от изведената програма:

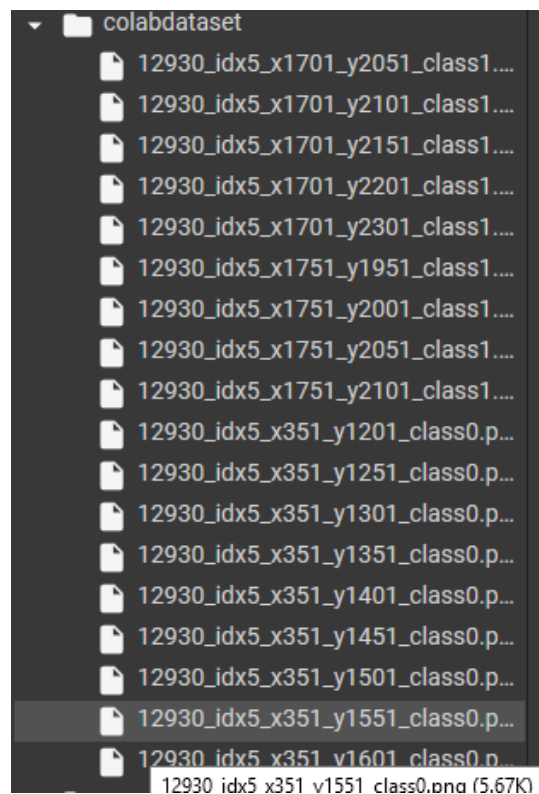


В) след като се активира папката в нейния раздел чрез изскачащото меню от десен бутон на мишката се избира New Folder, на която се присвоява името „colabdataset“



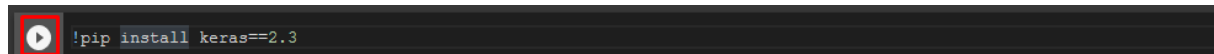


С) чрез Upload изскачащото меню на десен бутон на мишката се зарежда базата данни от изображения „colabdataset“. В основната директория се прикачва и файла „best_model.h5“:



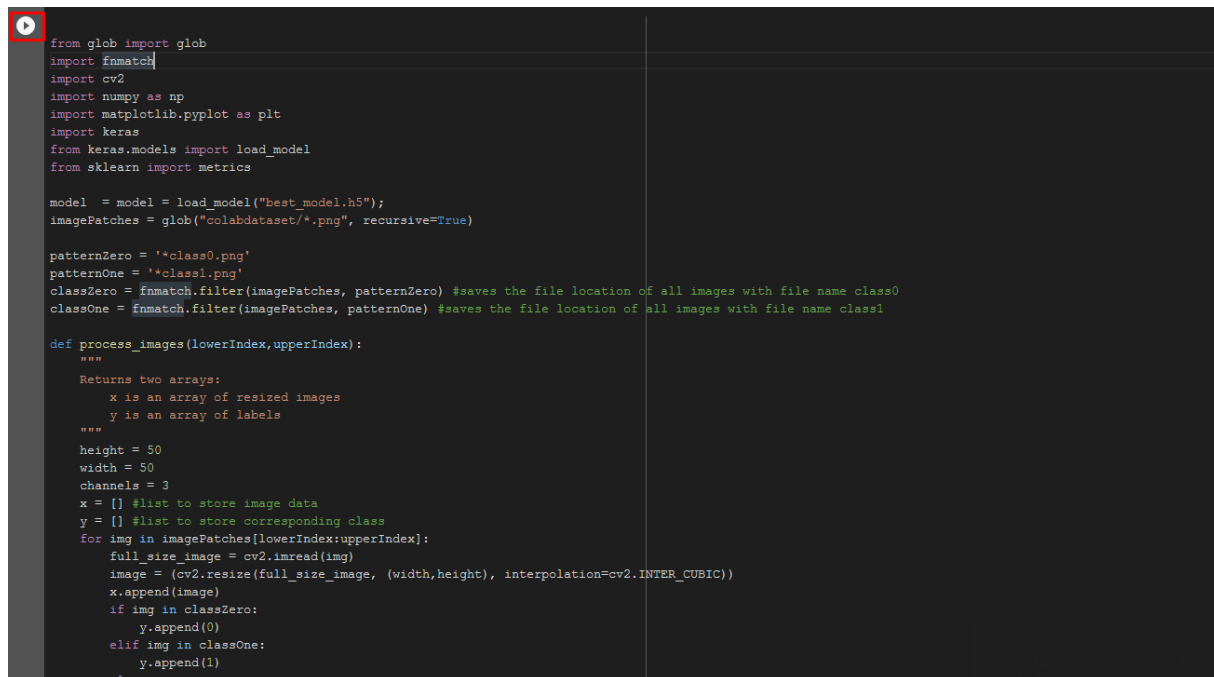
4. Активиране процеса на класификация:

А) процедурата по класифициране на изображения се стартира чрез бутона “Play” за инсталиране на keras:



```
!pip install keras==2.3
```

В) изпълнението на програмния модел на класификатор се стартира чрез бутона Play, разположен непосредствено преди началото на модела:



```
from glob import glob
import fnmatch
import cv2
import numpy as np
import matplotlib.pyplot as plt
import keras
from keras.models import load_model
from sklearn import metrics

model = load_model("best_model.h5");
imagePatches = glob("colabdataset/*.png", recursive=True)

patternZero = '*class0.png'
patternOne = '*class1.png'
classZero = fnmatch.filter(imagePatches, patternZero) #saves the file location of all images with file name class0
classOne = fnmatch.filter(imagePatches, patternOne) #saves the file location of all images with file name class1

def process_images(lowerIndex,upperIndex):
    """
    Returns two arrays:
        x is an array of resized images
        y is an array of labels
    """
    height = 50
    width = 50
    channels = 3
    x = [] #list to store image data
    y = [] #list to store corresponding class
    for img in imagePatches[lowerIndex:upperIndex]:
        full_size_image = cv2.imread(img)
        image = (cv2.resize(full_size_image, (width,height), interpolation=cv2.INTER_CUBIC))
        x.append(image)
        if img in classZero:
            y.append(0)
        elif img in classOne:
            y.append(1)
        else:
```

С) на екрана се извежда резултата от класификационния модел, съдържащ вероятността за принадлежност на всяко изображение към категория 0 и 1 от тестовата БДИ. Тези вероятностни стойности, които са по-големи от прага 0.5 спадат към група 1, показваща наличието на заболяване. В резултата е показана и матрица на класификация за съответния опит.

IV. Определяне ефективността на класификационен модел

IV.1. Матрица на класификация:

Ефективността на класификационен модел (класификатор) се определя чрез матрица на класификация (confusion matrix). Тя представлява таблично обобщение на резултатите от прогнозирането на класификационния проблем. Матрицата включва броят на коректните и некоректните прогнози в числови стойности поотделно за всеки клас. Матрицата на класификация има за задача да покаже грешките и техните видове, които класификационния модел допуска при предсказване на резултата. По същество тя е бинарен модел, състоящ се от четири квадранта, с четири стойности false positives (FP), false negatives (FN), true positives (TP) and true negatives (TN).

Таблица 1. Матрица на класификация

True Negative – TN (предсказана е ниска степен на заболяване и действителната е ниска степен на заболяване)	False Positive – FP (предсказана е висока степен на заболяване, а действителната е ниска степен на заболяване)
False Negative - FN (предсказана е ниска степен на заболяване, а действителната е висока степен на заболяване)	True Positive – TP (предсказана е висока степен на заболяване и действителната е висока степен на заболяване)

Пример за матрица на класификация

2256	749
369	1992

Физическо значение на получените резултати:

В посочения пример са предсказани по-малък брой FN стойности от броя на FP, което е по-благоприятно от обратния вариант, поради това че предсказването на тумор в напреднал стадий (malignant) като такъв в начален (benign) е много по-малко рисковано, отколкото предсказването на

тумор в начален стадий, докато в действителност той е в напреднал. Получената грешка в първия случай не представлява опасност за даден пациент, но изисква допълнителни изследвания за потвърждение на резултата. Във втория случай такава грешка е рискова и води до неправилна диагноза и лечение на заболяването.

IV. Крива на работната характеристика (ROC-AUC)

Крива на работната характеристика (Receiver Operating Characteristics-Area Under Curve, съкр. ROC-AUC) е мярка за определяне на ефективност на класификационен модел с използването на прагови критерии. Освен ROC кривата в ROC пространството се оценява и параметъра „област под кривата (Area Under Curve - AUC). ROC е вероятностна крива, докато AUC показва степента на различаване на класовете. По-високите стойности на AUC показват способност за по-висока точност на предсказване от страна на класификационния модел и обратно. ROC-AUC кривата се изобразява с използването на TPR (True Positive Rate) и FPR (False Positive Rate) характеристики, които се изчисляват съответно по формули (4) и (5):

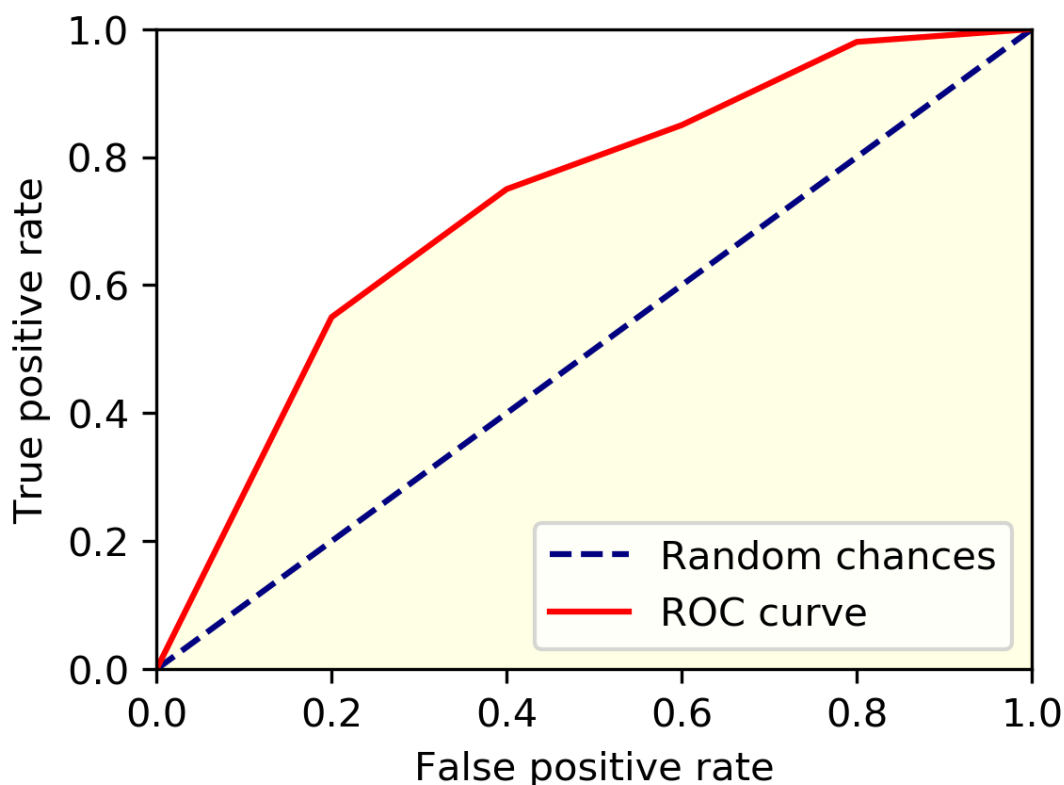
$$TPR = \frac{TP}{TP + FN}, \quad (4)$$

$$FPR = \frac{FP}{FP + TN}, \quad (5)$$

За физическото тълкуване на получените резултати при AUC са валидни следните прагови коефициенти:

$$AUC = \begin{cases} 0.5 & \text{няма различаване} \\ 0.6 - 0.7 & \text{слабо различаване} \\ 0.7 - 0.8 & \text{приемливо различаване} \\ 0.8 - 0.9 & \text{добро различаване} \\ > 0.9 & \text{висока степен на различаване} \end{cases}$$

На фиг. 7 е представена примерна ROC-AUC крива.



Фиг. 7 ROC-AUC крива

V. Задачи за изпълнение

Да се изпълни описаната в т. II. („Експериментална постановка“) последователност от стъпки по класификация на изображения с конволюционна невронна мрежа за три тестови БДИ: „colabdatasetA“, „colabdatasetB“, „colabdatasetC“ от папката „Medical Imaging“.

Като се използват получените резултати за проведените опити за тестовите групи от изображения „colabdatasetA“, „colabdatasetB“, „colabdatasetC“ да се изпълнят следните задачи и се състави лабораторен протокол според указания модел:

1. Да се определи ефективността на използвания класификационен модел чрез матрица на класификация (confusion matrix) за всяка тестова БДИ: „colabdatasetA“, „colabdatasetB“, „colabdatasetC“.
2. На база получените числови резултати за матрица на класификация да се изчислят стойностите на TPR и FPR за трите тестови БДИ.
3. Получените резултати от т. 2 да се представят графично чрез обща ROC крива за трите тестови БДИ.
4. Да се направи анализ на база построената ROC крива от т. 3 на заданието.

VI. Библиография

- [1] Г. Гочев „Компютърно зрение и невронни мрежи,“ Технически Университет, София, 1998.
- [2] Sameer Khan and Suet-Peng Yong, “A Deep Learning Architecture for Classifying Medical Images of Anatomy Object,” Proceedings of APSIPA Annual Summit and Conference, 2017.

VII. Допълнителни източници

V.1. Невронни мрежи

Christopher M. Bishop, “Pattern recognition and Machine learning,” Springer, 2006.

Sergios Theodoridis, Konstantinos Koutroumbas “Pattern recognition,” Elsevier, 2003.

V.2. TensorFlow

<https://www.tensorflow.org/>

<https://www.tensorflow.org/tutorials/keras/classification>

<https://www.tensorflow.org/about/bib>

V.3. Keras документация

<https://keras.io/>

<https://github.com/JGuillaumin/keras-doc-pdf/releases>

Контакти

vetova.bas@gmail.com

ПРОТОКОЛ

Тема: Образна диагностика с използването на невронни мрежи

Име:

Факултет:

Специалност:

Фак. №:

Група:

Дата:

I. Цел на упражнението

II. Задачи за изпълнение

1.
2.
3.
-

III. Получени резултати от проведените експериментални изследвания

1. Определяне ефективността на класификационен модел чрез матрица на класификация:

А) за тестова БДИ „colabdatasetA“:

В) за тестова БДИ „colabdatasetB“:

С) за тестова БДИ „colabdatasetC“:

2. Определяне на TPR и FPR:

А) за тестова БДИ „colabdatasetA“:

В) за тестова БДИ „colabdatasetB“:

С) за тестова БДИ „colabdatasetC“:

- 3. Получените резултати от т. 2 да се представят графично чрез обща ROC крива за трите тестови БДИ.**
- 4. Да се направи анализ на база построената ROC крива от т. 3 на заданието.**

IV. Изводи