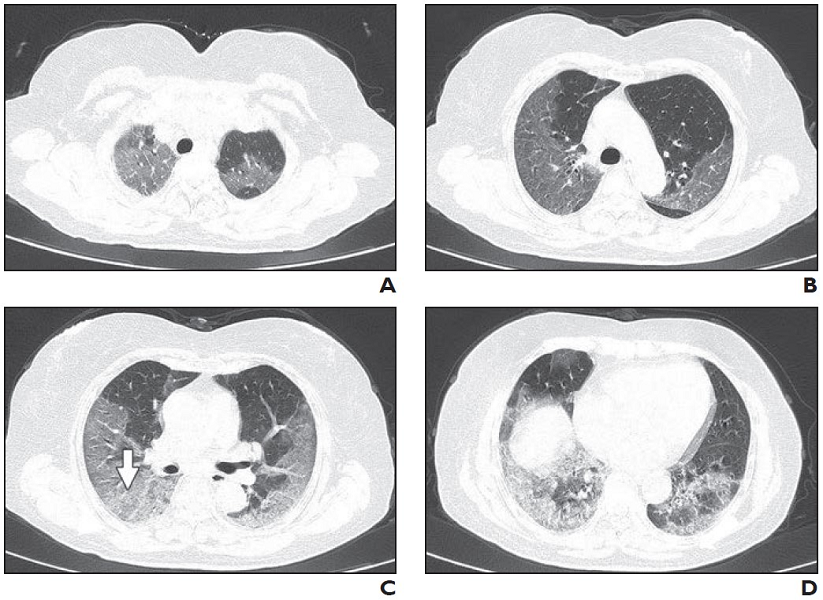
CT-SCAN CLASSIFICATION

DOCUMENTATIE

Proiect ML

Naiboiu Teodor

Grupa 312

# Introducere

Scopul acestui proiect este de a clasifica un set de date ce consta intr-un numar imagini CT realizate asupra plamanilor. Categoriile de clasificare sunt nativ, arterial si venos.

# Datele de intrare

Datele de intrare sunt impartite fiecare in foldere cu nume suggestive pentru implementarea retelei neronale, spre exemplu datele de train pot fi citite initial din fisierul “train.txt” pentru a corela numele imagini cu eticheta corespunzatoare ei: 1-nativ, 2-arterial, 3-venos si apoi putem naviga in directorul cu acelasi nume pentru a accesa imaginea de forma “nume\_imagine.png”.

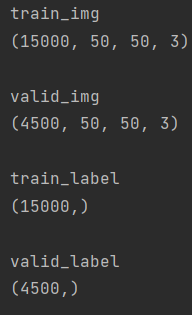
# Incarcarea datelor

In cazul datelor de antrenare si datelor de validare am parcurs fisierul .txt cu numele scopului datelor pe care vrem sa le incarcam si am citit numele imaginii impreuna cu eticheta corespunzatoare ei.

Am incarcat apoi imaginea citita intr-o lista, image = cv2.imread(file\_path)   
pe care am transformat-o intr-un sir de date numpy, images = np.array(images).

Incarcarea etichetelor s-a realizat in aceeasi modalitate, fiind doar citita valoarea de dupa “,” in fisierul .txt.

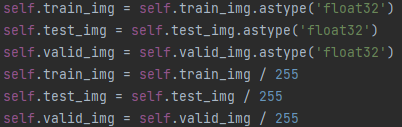
Dupa incarcarea datelor, lista numpy era de forma:



Prima valorea reprezinta numarul de imagini din setul de date corespunzator cu numele variabilei. Pe pozitia 1 si 2 se afla dimensiunile imaginii ( latime, inaltime ), iar pe ultima pozitie se afla canalul pentru culoare, in cazul nostru am incarcat imaginile sub format RGB, asadar canalul este 3.

# Normalizarea datelor

Inainte sa incepem sa construim modelul de antrenare este nevoie sa aducem datele la o forma normalizata, cat mai optimizata pentru ca calculele realizatet de sistemul de antrenare sa fie cat mai precise si rapide.



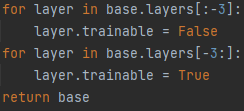
Am impartit numarul de pixeli la 255 pentru a aduce datele in intervalul [0, 1].

# Arhitectura modelului si abordare

Am considerat ca o abordare eficienta ar fi sa ma folosesc de un set pre-antrenat.



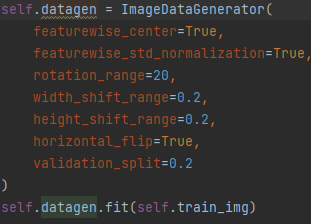
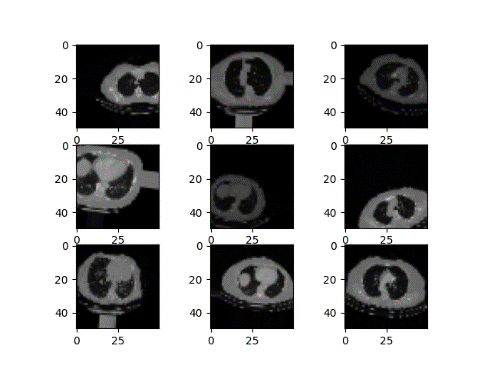
Am folosit Densely Connected Convolutional Networks pre-antrenata pe setul “imagenet” unde am “inghetat” toate layerele inafara de ultimele 3, cele folosite pentru output.



Am facut asta pentru a evita sa modific imaginile deja antrenate, ramanand sa reantrenez ultimele 3 straturi pe setul meu de antrenare pentru ca modelul sa aiba o acuratete mai buna.

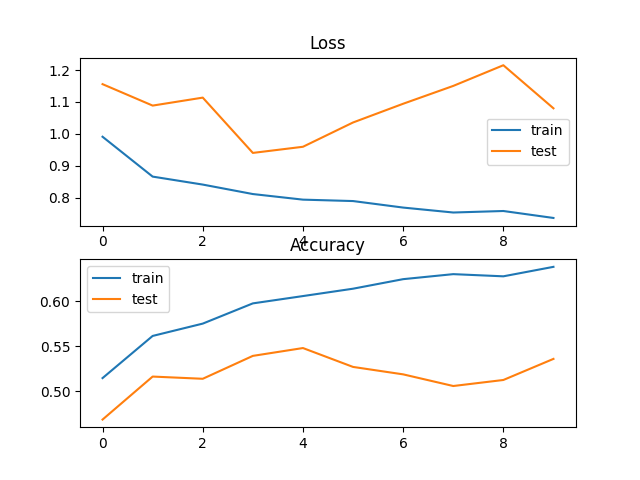
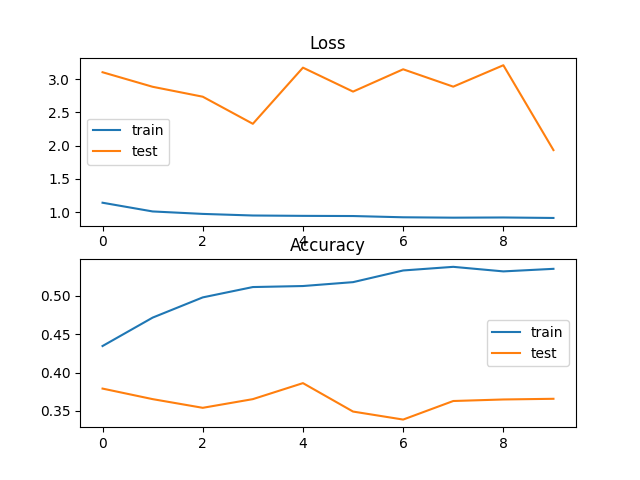
Am augumentat apoi datele pentru a evita posibilitatea ca modelul sa-si creeze “obiceiuri” (pattern-uri) de antrenare din datele de antrenare.

Augumentarea creeaza un dataset mai mare care ar trebui sa inbunatateasca rezultatul



Imaginile din dreapta reprezinta o singura imagine augumentata ( transformata ).

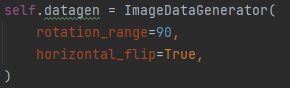
Din cauza ca imaginile sunt foarte mici, am reusit doar sa scad drastic acuratetea, augumentarea transformand foarte puternic datele. Asadar am incercat sa modific datele cat mai putin dar cat mai eficient prin diferite incercari. Argumentele care modificau width, height sau care faceau zoom pe poza, reuseau doar sa invete gresit modelul.



In imaginea din stanga se afla graficul pentru prima incercare cu augumentarea cat mai mare. In acest caz acuratetea este foarte slaba.

In imaginea din dreapta observam o mica imbunatatire, ceea ce ne sugereaza ca am putea incerca sa augumentam datele chiar si mai putin.

Augumentarea finala este aceasta:



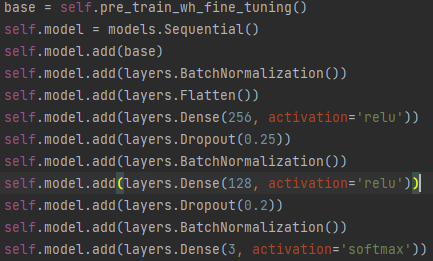
## Arhitectura modelului

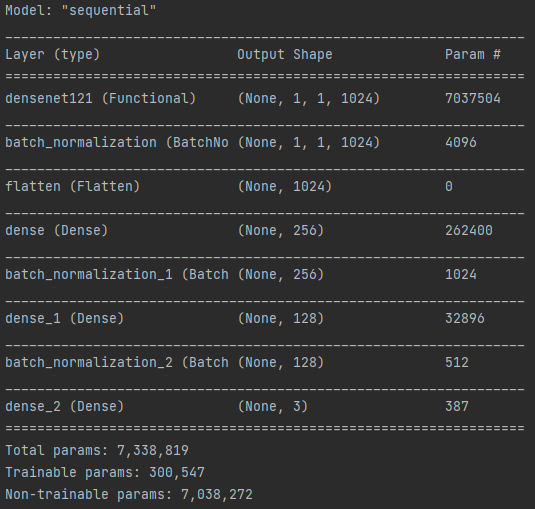
Functia self.pre\_train\_wh\_fine\_tuning() returneaza „baza” modelului, adica modelul cu layere deja antrenate. Retin in variabila „base” aceasta baza si imi declar modelul de antrenare.

Modelul final este reprezentat de modelul de baza, cu datele preantrenate plus straturi de Dense

pentru conectare.

Stratul final de Dense este reprezentat de 3 straturi, deoarece numarul claselor in care vrem sa clasificam este 3 cu functia de activare „softmax”



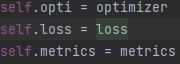


Detaliile despre model ne arata ca avem 7,038,272 parametrii neantrenabili, acestia sunt parametrii deja antrenati din DenseNet212.

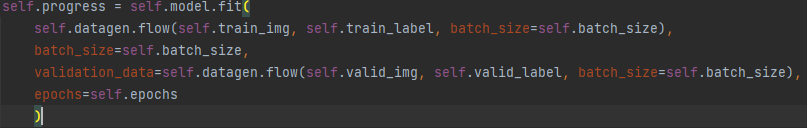
# Compilarea modelului si antrenarea datelor.

Initial pentru compilare am folosit optimizerul „adam” cu functia de loss „sparse\_categorical\_crossentropy”. Deoarece nu avem o clasificare binara, functia de loss ar trebui sa fie optima pentru acest model.

Aceste argumente sunt preluate de clasa programului in initializarea sa.



Am inceput antrenarea modelului folosindu-ma de datagenul generat mai sus, in augumentarea datelor



In aceeasi faza initiala am modificat steps\_per\_epoch pentru o mai buna antrenare. Am antrenat modelul in 10 epoci. Acuratetea obtinuta nu era una foarte buna, ~0.50.

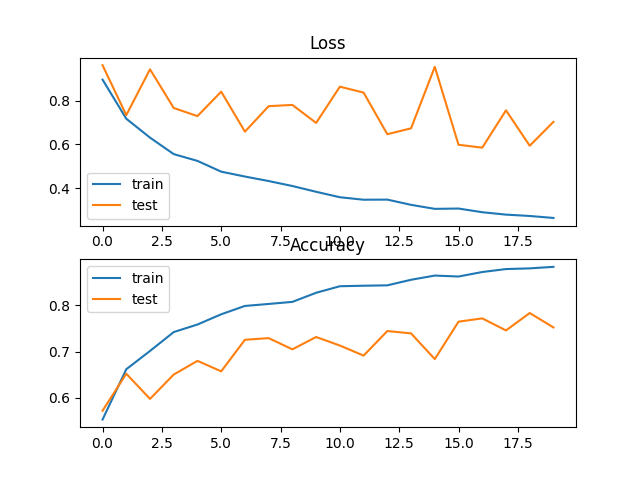
Am incercat apoi sa modific diversi parametrii in faza de compilare, dar am obtinut un scor asemanator de slab.

Asadar am considerat ca ar fi mai bine sa renunt la implementare unei retele preantrenate si am implementat o retea convolutionara.

# A doua implementare

In a doua implemenare am introdus o retea standard de straturi convolutionare.

Am implementat un strat convolutional format din 16 neurori, urmat de un Pool de ( 2,2 ).

Apoi am repetat procesul dubland numarul de neuroni din fiecare strat convolutional pana la 128.

Am observat ca rezultatul obtinut este mult mai bun decat rezultatul precedent, asa ca am hotarat sa pastrez aceasta implementare.

Acuratete: 0.7300

Kaggle: 0.71965

Matricea de confuzie pentru acest caz:

# 

# Configurarea finala si scor kaggle

In configurarea finala am ales sa pastrez a doua arhitectura.