Învățare Automată: Tema 3

Clasificare imagini medicale

Vlad Bogolin, Mihai Trăscău

Facultatea de Automatică și Calculatoare, UPB

Cerință

Implementați și antrenați o rețea convoluțională pentru clasificarea unor radiografii în două clase: normal și anormal.

Setul de date folosit este MURA v1.1* si poate fi descărcat de aici:
`https://tinyurl.com/y2z3eqrn`

* Rajpurkar, Pranav, et al. ”Mura: Large dataset for abnormality detection in musculoskeletal radiographs.” arXiv preprint arXiv:1712.06957 (2017).

Setul de date

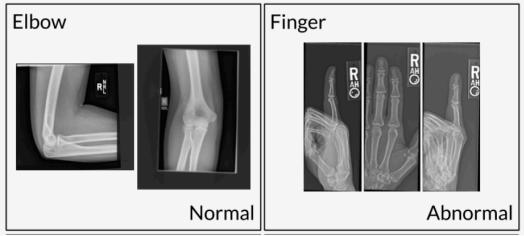
Setul de date este compus din N perechi $(\mathbf{x}_n, \mathbf{t}_n)$ unde:

- $\mathbf{x}_n \in \mathbb{R}^{1\times H\times W}$ este o imagine alb-negru de dimensiuni $H \times W$ reprezentată printr-un tensor;
- $\mathbf{t}_n \in \{0, 1\}$ reprezintă clasa (1 pentru normal și 0 pentru anormal).

Vom nota cu X și T mini-batch-uri ce conțin B intrări, respectiv B ieșiri:

$$X \in \mathbb{R}^{B\times 1\times H\times W}, T \in \{0, 1\}^B$$

.



Cerință detaliată (I)

- Pentru această temă va trebui să implementați/folosiți un framework care să permită antrenarea unei rețele neurale:
  - calculul predicției rețelei pentru un mini-batch de exemple date la intrare (forward);

$$Y = f\left(X, \theta^{(t)}\right) = f_L\left(\ldots f_2\left(f_1\left(X, \theta_1^{(t)}\right), \theta_2^{(t)}\right)\ldots, \theta_L^{(t)}\right)$$

    unde $f_1, f_2, \ldots f_L$ reprezintă straturile (sau blocurile) rețelei
  - calculul unei funcții de cost (pentru segmentare: binary cross entropy);

$$\mathcal{L} \overset{not.}{=} BCE(Y, T) = -\frac{1}{B}\sum_{b=1}^{B} T_b \log\left(Y_b\right) + \left(1 - T_b\right)\log\left(1 - Y_b\right)$$

  - calculul gradientului erorii în raport cu parametrii curenți folosind backpropagation;

$$g^{(t)} = \frac{\partial BCE\left(f\left(X, \theta^{(t)}\right), T\right)}{\partial \theta}$$

  - ajustarea parametrilor pe baza gradientului folosind stochastic gradient descent

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \alpha g^{(t)}$$

  - adaugare strat Dropout care dezactiveaza un procent p din neuroni

Cerința 1 (Definire model) - 3p

- Implementați într-un framework la alegere o rețea neurală care are următoarele layere:
  - Input: 227x227x1
  - Conv1: 7x7 filter, 64 feature maps, stride 2, ReLU activation
  - Max Pooling: 2x2 filter, stride 2
  - Conv2: 5x5 filter, 128 feature maps, stride 2, ReLU activation
  - Max Pooling: 2x2 filter, stride 2
  - Conv3: 3x3 filter, 256 feature maps, stride 1, ReLU activation
  - Conv4: 3x3 filter, 384 feature maps, stride 1, ReLU activation
  - Conv5: 3x3 filter, 256 feature maps, stride 1, ReLU activation
  - Conv6: 3x3 filter, 256 feature maps, stride 1, ReLU activation
  - Max Pooling: 2x2 filter, stride 2
  - Dropout 0.5
  - Fully Connected: 2048, ReLU activation
  - Dropout 0.5
  - Fully Connected: 1, Logistic activation

Cerința 2 (Antrenare) - 2p

- Antrenați rețeaua definită la punctul 1. Utilizați funcția BCE descrisă mai sus ca funcție de loss.
- Afișați grafic valoarea loss-ului pe durata antrenării.
- Salvați modelele antrenate astfel încât să poată fi folosit pentru a clasifica noi imagini.

Cerința 3 (Evaluare) - 2p

Pentru evaluarea performanței rețelei se vor folosi două metrici. Metricile se vor calcula atât pentru setul de date de antrenare, cât și pentru setul de date de validare:

- acuratețe per imagine: $acc = \frac{nr\_imagini\_clasificate\_corect}{nr\_total\_de\_imagini}$.
- acuratețe per pacient: $acc = \frac{nr\_pacienti\_clasificati\_corect}{nr\_total\_de\_pacienti}$ unde clasa per pacient este data de clasa majoritară a tuturor imaginilor pentru un anumit pacient.

Cerința 4 (Grafice) - 3p

2p Variați numărul de layere și analizați impactul lor asupra performanței. Afișați într-un grafic cum se modifică performanța atât pentru setul de date de antenare cât și pentru cel de testare. Faceți comparațiile la același număr de pași de antrenare. Pentru a vedea mai bine diferențele afișați totul pe același grafic.

- Adăugați minim încă două straturi convoluționale modelului de la Cerința 1. Alegeți voi unde, folosiți o dimensiune adecvată (de exemplu, dimensiunea filtrului să fie de 3x3, 5x5). Este indicat ca filtrele mai mari să fie la începutul rețelei, iar cele mai mici spre final.
- Mai adăugați un strat Fully Connected după ultima convoluție.

1p Variați numărul de feature map-uri și analizați impactul lor asupra performanței.

- Adăugați mai multe feature map-uri (modelului de la Cerința 1) la fiecare nivel și comparați efectul avut față de adăugarea mai multor layere (experimentele descrise mai sus).

Cerința BONUS - 2p

- Adăugați conexiuni reziduale modelului definit la Cerința 1 și analizați impactul lor asupra performanței (pentru detalii legate de conexiuni reziduale vezi Detalii implementare).

Detalii implementare (1)

- În cazul în care doriți utilizarea framework-ului de la laborator, implementați următoarele straturi (este necesară reimplementarea convoluțiilor folosind operații tensoriale):
  - strat de convoluție care primește un volum $X \in \mathbb{R}^{B \times D_{in} \times H \times W}$ și produce un volum de ieșire $Y \in \mathbb{R}^{B \times D_{out} \times H \times W}$ pe baza unui filtru pătratic (de dimensiune K) ce are parametrii $\theta \in \mathbb{R}^{D_{out} \times D_{in} \times K \times K}$
    - folosiți *padding* în volumul original pentru a păstra dimensiunea hărților
    - o convoluție aplicată intrărilor ar avea $D_{in} = 1$ (alb-negru)
  - bloc rezidual aplicat asupra unui strat sau a unei secvențe de straturi : $Y = f(X, \theta) + X$
  - activare ReLU $y = \max(x, 0)$ (se aplică element cu element)
  - activare logistică $y = \frac{1}{1+e^{-x}}$ (pentru a produce probabilități pentru clasă la ieșire)
  - strat Dropout care dezactiveaza un procent p din neuroni atât în partea de *forward* cât și în partea de *backpropagation*

- Pentru fiecare strat veți implementa:
  - $Y = f(X, \theta)$ pentru pasul forward și
  - $\frac{\partial \mathcal{L}}{\partial X}$, $\frac{\partial \mathcal{L}}{\partial \theta}$ pe baza $\frac{\partial \mathcal{L}}{\partial Y}$ pentru backpropagation.

Detalii implementare (2)

- Pentru implementarea eficientă a convoluțiilor puteți folosi funcțiile `im2col` și `col2im`.
  - O implementare a acestor funcții găsiți aici: `http://cs231n.github.io/assignments2018/assignment2/`
- Pentru a păstra dimensiunea hărților (*feature maps*) de la intrare până la ieșire, adăugați la fiecare pas bordură de zero (*padding*).
- Un bloc rezidual poate conține unul sau mai multe straturi de convoluție (și ReLU). E.g.

$$ResBlock(X, \theta^{(t)}) = ReLU\left(conv\left(ReLU\left(conv\left(X, \theta_1^{(t)}\right)\right), \theta_2^{(t)}\right) + X\right)$$

Observații

- Imaginile au dimensiuni diferite. Puteți rezolva această problemă în mai multe moduri: prin scalare sau prin tăiere de petice de dimensiunea intrării în rețea.
- În cazul în care implementați straturile, pentru fiecare strat este necesar să implementați atât calculul forward cât și cel backward folosind operații tensoriale din `numpy` (sau altă bibliotecă pentru aritmetică tensorială). În caz contar, veți avea probleme legate de timpul de antrenare și nu o să reușiți să terminați tema.
- Evaluați modelul după fiecare epocă și construiți grafice cu performanța atât pe setul de date de antrenare, cât și pe cel de validare.
- Discutați efectul indus de: numărul de blocuri (adâncime), numărul de hărți din straturile intermediare, dimensiunea mini-batch-ului, rata de învățare, momentum.

Resurse

- Mini-batch SGD https://www.youtube.com/watch?v=4qJaSmvhxi8
- Convoluții http://cs231n.github.io/convolutional-networks/
- ResNets https://www.youtube.com/watch?v=K0uoBKBQ1gA pentru a înțelege blocurile reziduale
- Momentum https://www.youtube.com/watch?v=k8fTYJPd3_I&t=4s

Atenție!

Sfat!

Nu lăsați tema pentru ulimele zile. Antrenarea unui model poate **dura mult**! Nu veți putea realiza toate experimentele dacă lăsați tema pentru ultimele zile.