Lung CT Scan Classification

using Machine Learning techniques

Ciauşu Nicoleta
Grupa 234

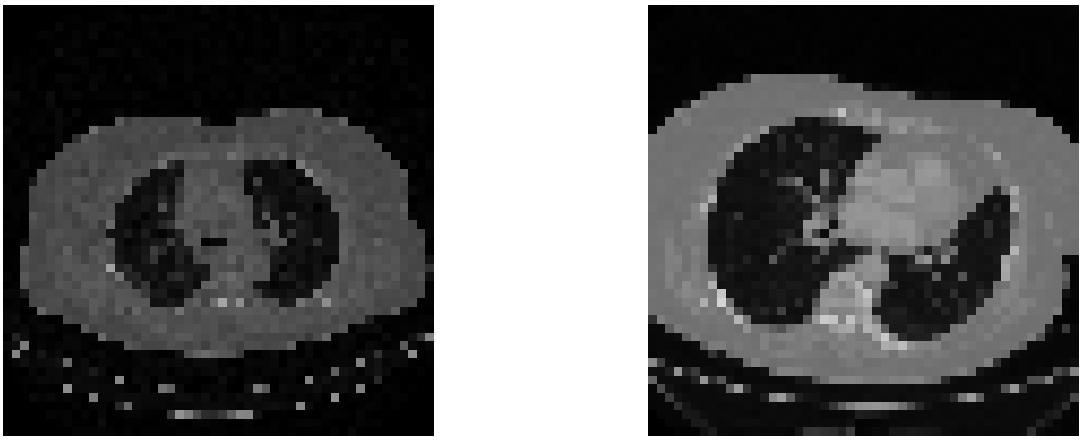
1 Introducere

Tema competitiei presupunea crearea unui model prin Machine Learning care sa clasifice cat mai bine posibil o imagine obtinuta prin scanarea CT a plamanilor (Computer Tomograf) intr-una dintre trei categorii: `native, arterial, venous`, pentru aceasta avand la dispozitie un dataset compus din 15000 imagini de antrenament, 4500 de validare si 3900 de testare.

2 Explorarea si vizualizarea datelor

Uitandu-ma prin dataset, am constatat ca am de a face cu imagini grayscale de **dimensiune fixa** `(50x50px)` ceea ce insemna ca nu a fost nevoie de pasul de preprocesare in care as fi adus toate imaginile la aceeasi dimensiune, cum ma asteptam la inceput. Totodata, am mai sesizat si faptul ca imaginile au fost downsampled mult (o imagine CT are in mod normal rezolutia `512x512px`). Am facut si urmatoarele observatii ce m-au ajutat pe partea de augumentare a datelor:

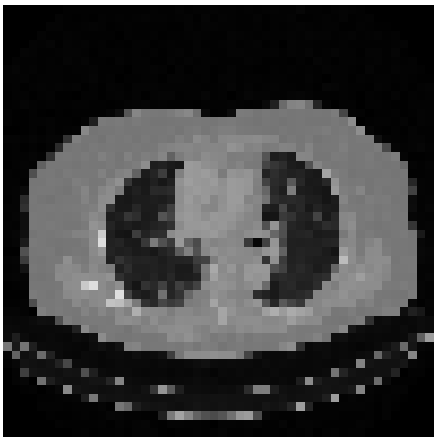
- exista **diferente de contrast** intre imagini, probabil deoarece au fost folosite diferite aparate tip CT folosite pentru a obtine imaginile.



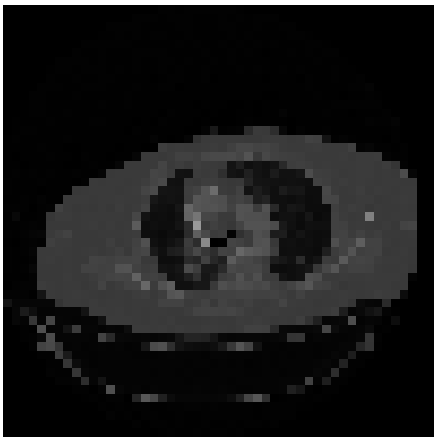
(a) Contrast slab (b) Contrast puternic

Figure 1: Imagini din dataset cu contrast diferit

- Imaginile CT au **aceeasi orientare**, deci operatii de tipul flip orizontal/vertical nu au sens.
- Exista o oarecare **diferenta in centrarea** plamanilor in poze (acest lucru fiind dat de faptul ca persoanele au greutate diferita si se centreaza altfel in aparat)



(a) Centrare obisnuita



(b) Subiect deplasat la dreapta

Figure 2: Imagini din dataset in care subiectul este centrat diferit

3 Modele folosite

Initial am incercat sa folosesc un clasificator tip **Naive Bayes** pentru a rezolva aceasta problema de clasificare, desi nu ma asteptam sa am foarte mult succes - voiam mai mult sa ma acomodez cu lucrul cu NumPy si laboratoarele erau foarte clare.

Mai apoi, facand research, m-am intalnit cu modelele tip **Convolutional Neural Network - CNN**, care conform mai multor surse ar obtine performante foarte bune pe problemele tip clasificare de imagini. Asadar, am decis sa creez si eu un astfel de model, ca sa vad cum se comporta.

3.1 MLP vs. CNN

Si totusi, de ce n-am incercat intai **MLP** (multi-layer perceptron)? Din putinul pe care il cunosc despre anatomie, intuitia mi-a spus ca pozitionarea feature-urilor conteaza foarte mult in determinarea clasei corecte asociate unei imagini (Categoriile fiind legate de sistemul circulator). MLP-urile au marele **dezavantaj** ca accepta input **flattened** (practic, in loc sa procesez o matrice 50x50 de valori intre [0,1] as fi procesat un array de 2500x1) asa ca **pierd informatii vitale** despre pozitionarea valorilor, respectiv a imprejurimilor unui punct.

Aceasta slabiciune este **depasita** de Convolutional Neural Networks prin introducerea conceptului de **convolutie**.

3.1.1 Ce este o convolutie?

In operatia de convolutie, un filtru este aplicat in mod repetat asupra unui input pentru a obtine ceea ce se numeste un **"feature map".**

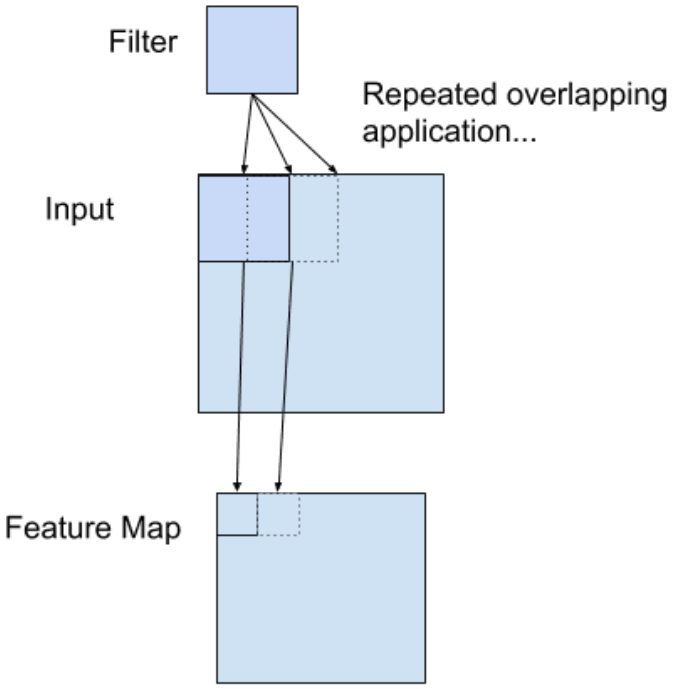


Figure 3: Operatia de convolutie genereaza un feature map

Diferite filtre aplicate unei imagini pot obtine **diferite efecte.**

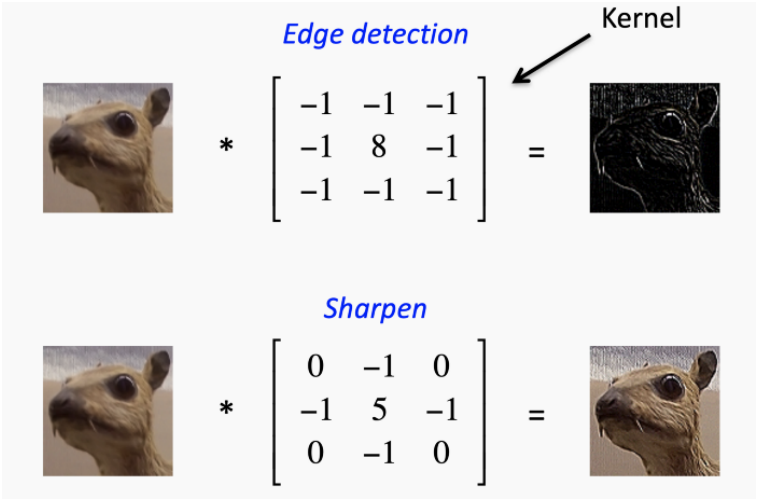


Figure 4: Exemple de filtre care obtin efecte inteligibil umane. In CNN-uri, valorile din filtru sunt alese **aleator**, pentru a evidentia anumite trasaturi ale input-ului.

Aplicand in etapa de convolutie un numar de filtre asupra input-ului obtin feature maps -

versiuni modificate ale input-ului in care sunt evidentiate anumite trasaturi. Asupra acestor feature maps este aplicata apoi o **transformare non-liniara** (cea mai des folosita este ReLU), deoarece operatia de convolutie este liniara si imi doresc sa introduc non-linearitate.

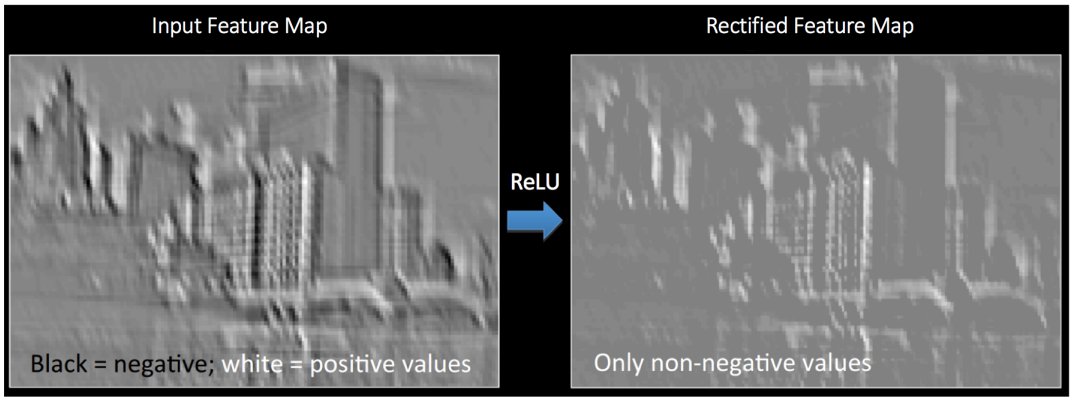


Figure 5: Efectul functiei de activare ReLU asupra unui feature map.

3.1.2 Operatia de pooling

Scopul operatiei de pooling este acela de a **reduce dimensiunile** feature map-urilor pentru a scadea capacitatea de procesare necesara a calculatorului pentru a efectua operatiile urmatoare.

In majoritatea CNN-urilor este nevoie de mai mult de o operatie de convolutie pentru a extrage trasaturile utile in clasificarea input-ului. Pentru acest lucru se foloseste pooling-ul, care este o operatie ce are efectul de a **micsora dimensiunea** unui feature map. Cele mai comune metode de pooling sunt MinPooling, **MaxPooling** si AveragePooling. Pentru min/max, pentru un anumit pool size(lungimea laturii submatricii pe care sa fie efectuata operatia), se alege valoarea minima/maxima. Pentru average, se face media tuturor valorilor din patrat.

In general este preferat max-pooling deoarece obtine rezultate mai evidente pentru model.

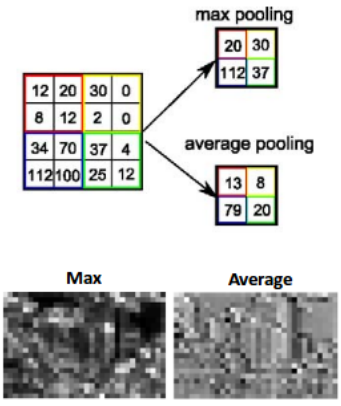


Figure 6: Comparatie dintre max-pooling si average-pooling.

Pe scurt, in crearea unui model CNN au loc urmatoarele **etape obligatorii**:

1. Normalizarea input-ului, daca e cazul: aici, pentru imagini, am transformat spatiul [0,255] in [0,1]
2. **Etapa de extragere a feature-urilor** relevante: Una sau mai multe serii de operatiile urmatoare:
   (a) Convolutia
   (b) Aplicarea functiei de activare non-liniara
   (c) Pooling
3. **Etapa de clasificare**
   (a) Flattening
   (b) Conectarea la unul sau mai multe layere dense
   (c) Conectarea la layer-ul final, de output, unde se face clasificarea in functie de numarul de categorii dorit.

In urma research-ului meu, eu vizualizez CNN-urile ca fiind un fel de extensie a MLP-urilor, deoarece ceea ce am numit "etapa de clasificare" mai sus seamana foarte mult cu ceea ce se face in MLP (flatten pe input, 1/mai multe hidden layers, layer de output), la care adaug o etapa de extragere a detaliilor relevante ce se preteaza mai bine pe input-uri 2d precum imaginile.

3.2 Alegerea arhitecturii CNN-ului

Avand o idee mai clara despre ceea ce trebuie sa fac in modelul meu, acum trebuie sa ma decid asupra **arhitecturii** si a **hiperparametrilor** care dau rezultate cat mai bune pentru problema data.

3.2.1 Cate layere de convolutie?

Am incercat mai multe variante si notat ce acuratete am obtinut. Pentru aceste teste, am folosit:

- **C32** = Layer de convolutie cu 32 filtre si kernel size 3
- **MP** = Layer de MaxPooling cu stride 2 si dimensiune (2,2)
- **D128** = Layer de clasificare cu 128 noduri
- **ReLU** = Functia de activare folosita peste tot in model
- **softmax** = functia cu care am scos distributia de probabilitate a imaginii de a se afla in una din cele 3 categorii

Arhitectura folosita - feature extraction	Validation Accuracy	Validation Loss
C24 - MP	0.58	>1
C24 - MP - C48 - MP	0.71	>1
C24 - MP - C48 - MP - C64 - MP	0.73 0.72	1.03 0.70
C24 - MP - C48 - MP - C64 - MP - C96 - MP	0.74	0.78
C32 - MP - C64 - MP - C128 - MP - C256 - MP	0.76 0.75	0.93 0.67
C64 - MP - C128 - MP - C256 - MP - C512 - MP	0.74	0.66

(+ layer D128 pentru clasificare)

Am incercat mai multe arhitecturi si obtinut urmatoarele valori care imi sugerau ca **mai multe layere = performanta mai buna**, si ca **valori mai mari = acuratete mai mare**, dar si **timp mult mai mare de antrenament**. Pana la urma am decis sa maresc numarul de filtre si sa pastrez **doar 4 straturi**, asa am obtinut performanta buna si timp de antrenament decent pe arhitectura **C64 - MP - C128 - MP - C128 - MP - C256**

3.2.2 Cate layere dense?

Pe arhitectura **C64 - MP - C128 - MP - C128 - MP - C256**, am testat:

Arhitectura folosita - etapa de clasificare	Validation Accuracy	Validation Loss
D0 (adica fara, din flatten in softmax direct)	0.77 0.74	1.2 0.66
D32	0.75	0.73
D128	0.75 0.74 0.77	0.67 0.66 0.60
D256	-	>1.6
D128 - D128	0.75	0.87
D128 - D64	-	>1

D = dense layer

In urma acestei primei serii de incercari, am ales sa raman cu un singur layer dense de dimensiune 128. Mai tarziu, cand am implementat dropout, am modificat acest layer, deoarece am testat si valori mai mari.

3.2.3 Cat dropout?

Si de ce dropout, pana la urma?

Cand am facut tabelele de mai sus, m-am oprit in momentul in care training accuracy incepea sa creasca, in timp ce validation accuracy stagna sau chiar incepea sa coboare (semne clare de **over-fitting**), motiv pentru care am considerat ca aici este momentul sa introduc **tehnici de regularizare** pentru a face modelul sa invete mai lent. Pe scurt, **dropout** este tehnica prin care ii asociez fiecarui nod o sansa de a fi dezactivat per epoca, incercand astfel sa fac modelul sa antreneze si alte zone pe unde a trecut mai putin.

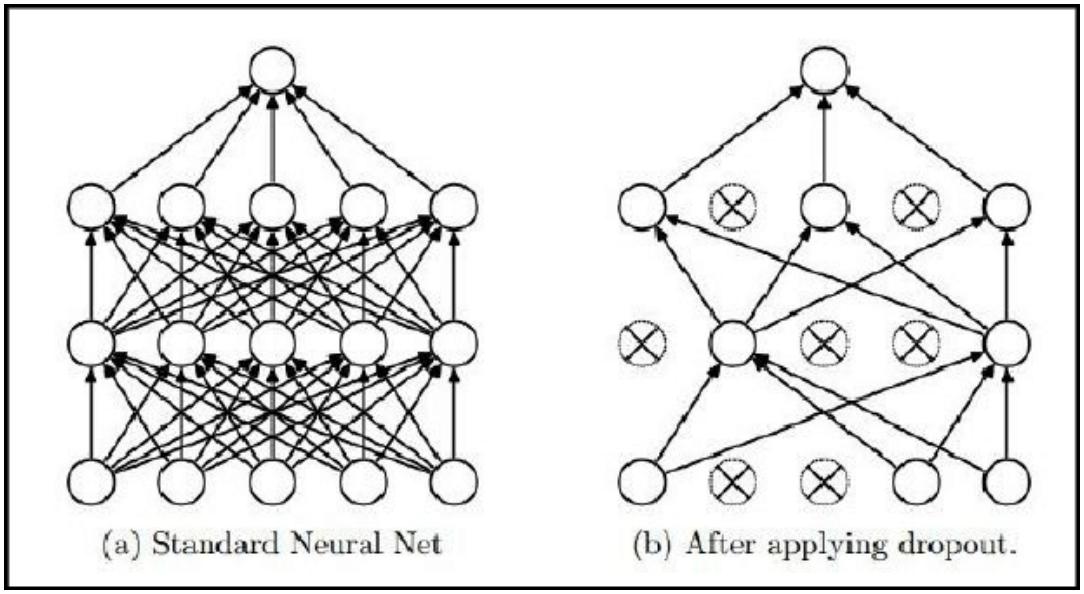


Figure 7: Efectul dropout-ului asupra modelului

In aceasta etapa am incercat urmatoarele noi lucruri peste arhitectura deja stabilita de clasificare **C64 - MP - C128 - MP - C128 - MP - C256 - D128**:

Dropout	**Validation Accuracy**	**Validation Loss**	**Avg. Val. Acc**	**Avg. Val. Loss**
dropout 0.1	0.71	0.69	0.71	0.69
dropout 0.2	0.70 0.74 0.74 0.74	0.60 0.64 0.58 0.62	0.73	0.61
dropout 0.3	0.74 0.72 0.74 0.74	0.64 0.60 0.56 0.55	**0.735**	**0.587**
dropout 0.4	0.70	0.64	0.70	0.64

In urma **average-ului** (caci fiind drop cu anumita sansa, au fost nevoie de mai multe rulari pentru a-mi face o parere) am concluzionat ca un **dropout de 0.3** este potrivit pentru modelul meu.

Intr-adevar, acum ajungeam la accuracy-ul de 0.72 dupa un timp de rulare mai lung. Am considerat, asadar, ca ar fi o idee buna sa incerc sa maresc dense layer-ul pentru o clasificare si mai buna, si pentru a testa mai bine diferenta intre dropout 0.2/0.3 (pentru ca am obtinut valori apropiate):

Dense + Dropout	Validation Accuracy	Validation Loss	Avg. Val Acc	Avg. Val Loss
D256 + dropout 0.2	0.71 0.73	0.68 0.62	0.72	0.65
D512 + dropout 0.2	0.73 0.75	0.65 0.71	0.74	0.68
D512 + dropout 0.3	0.73 0.75 0.75 0.70 0.71	0.60 0.59 0.57 0.66 0.65	0.72	0.61
D1024 + dropout 0.3	0.74 0.72 0.70	0.62 0.59 0.59	**0.72**	**0.60**

In urma acestui experiment, am concluzionat ca un **dense layer de 1024** ofera performanta mai buna modelului meu.

3.2.4 Augumentarea datelor

In urma studiilor de mai sus, am creat modelul **C64 - MP - C128 - MP - C128 - MP - C256 - Flatten - D1024** care, alaturi de dropout 0.3, a obtinut acuratetea 0.79 pe Kaggle dupa un antrenament de 70 epoci. De cele mai multe ori, **un dataset mai mare te duce spre acuratete mai buna**, asa ca am incercat sa maresc artificial dataset-ul meu folosind tehnici de **data augumentation.**

Din fericire, facand analiza datelor la inceput nu mi-a fost greu sa identific operatiile pe care sa le folosesc. Am ales sa folosesc:

1. **RandomTranslation**
2. **RandomContrast**

pentru a genera noi imagini plauzibile. Dupa mai multe incercari, am ajuns la hiperparametrii 0.15 si 0.2 pentru translatie, respectiv contrast. Toate celelalte augumentari posibile nu au adus la vreo imbunatatire a acuratetii.

3.2.5 Alte incercari

- Am mai incercat sa modific kernel size-ul convolutiilor fara vreun succes.
- Am testat alte functii de activare (LeakyReLU, PReLU) dar nu am observat vreo imbunatatire a acuratetii, ci din contra.
- Am incercat 2 sau chiar 3 layere Dense, dar pe langa faptul ca incetinea clasificarea, nici nu se imbunatatea acuratetea.
- Am incercat sa adaug regularizare L2 pentru a penaliza weight-urile foarte mari pe unele muchii, fara succes.
- Am adaugat BatchNormalization cu succes! Efectul a fost scaderea numarului de epoci necesare pentru a ajunge la rezultatul final.

- Initial am implementat EarlyStopping, dar pana la urma m-am imprietenit mult mai bine cu sistemul de **checkpoints** din Keras - dupa fiecare epoca rulata, daca obtineam un model cu validation accuracy mai mare decat cel anterior, salvam modelul intr-un fisier cu nume de forma valAcc-valLoss. Astfel, dupa o tura de antrenament cu multe epoci, ramaneam cu mai multe modele dintre care imi alegeam ce checkpoint salvat mi se parea cel mai promitator. Asa am obtinut scorul meu maxim, de 0.81.

3.2.6 Arhitectura finala

In final, cu modelul meu cu arhitectura compusa din:

- Augumentarea de date descrisa mai sus (translatie + contrast random)
- 4 straturi convolutionale cu 64, 128, 128, 256 filtre, cu dimensiunea kernel-ului de 3, pentru a invata detaliile relevante din imagini, si functia de activare ReLU pentru a introduce nonlinearitate
- MaxPooling intre straturile convolutionale pentru a reduce dimensiunea feature map-urilor rezultate din convolutii
- Dropout de 0.3 pentru a preveni overfitting-ul
- Urmat de stratul fully connected de 1024 neuroni cu functia de activare ReLU pentru a face clasificarea
- Iar ultimul strat, de output, cu 3 neuroni si functia de activare Softmax, pentru a obtine probabilitatea cu care modelul meu crede ca imaginea data apartine fiecarei categorii. Indicele valorii maxime reprezinta label-ul asociat de model imaginii testate.

am obtinut o acuratete de 0.81 pe Kaggle.

In final, ar trebui sa mentionez putin si despre optimizer, eu am folosit Adam, care este un optimizer mai nou, o imbunatatire a Stochastic Gradient Descent, care pentru mine a avut marele avantaj de a-mi scadea unul dintre hiperparametrii ce trebuiau determinati (Adam isi adapteaza singur learning rate-ul pe parcursul antrenarii). Loss function-ul pe care l-am folosit este sparse categorical crossentropy, cu acesta determinand cat de buna este estimarea mea.

3.2.7 Matricea de confuzie

```
[[1435.   25.   40.]
 [  77. 1096.  327.]
 [  42.  109. 1349.]]
```

3.3 Naive Bayes

Primul clasificator pe care l-am facut a fost de tipul **Naive Bayes.** Desi nu ma asteptam sa obtin o performanta foarte buna cu el, am ales sa creez acest tip de clasificator pentru a ma **obisnui** cu lucrul cu librariile NumPy, Pandas, pe care aveam de gand sa le folosesc la modelul urmator. Asadar, desi **nu am obtinut o acuratete prea buna (0.52),** exercitiul acesta mi-a fost foarte util in familiarizarea cu mediul de lucru.

Acest tip de clasificator are marele dezavantaj ca **presupune ca nu exista nicio corelatie intre caracteristicile cautate** - adica evenimentele sunt independente. Acesta este lucrul care il face sa fie "naiv". In cazul acestei probleme, fiecare **pixel este considerat un atribut independent** in apartenenta imaginii X la clasa c. Folosind teorema lui Bayes pe dataset-ul de antrenament stabilim probabilitatile pentru fiecare feature in functie de apartenenta la clasa respectiva.

Pentru fiecare imagine, probabilitatea ca ea sa se incadreze intr-una dintre clase se calculeaza prin **inmultirea probabilitatilor fiecarui atribut** (adica al fiecarui pixel din cei 2500) conditionat de clasa c. Raspunsul ales va fi clasa careia ii corespunde probabilitatea maxima.

Am testat mai multe valori posibile pentru intervale (trebuie transformat spatiul continuu in valori discrete folosind o histograma) si am ajuns la concluzia ca **2 intervale furnizeaza rezultatul optim:**

```
2 intervale, acuratete: 0.5265614581018004
4 intervale, acuratete: 0.3903089575461214
6 intervale, acuratete: 0.37430540120026673
10 intervale, acuratete: 0.3665258946432541
12 intervale, acuratete: 0.3578573016225828
16 intervale, acuratete: 0.3494109802178262
18 intervale, acuratete: 0.3511891531451434
22 intervale, acuratete: 0.35519004223160705
142 intervale, acuratete: 0.3620804623249611
252 intervale, acuratete: 0.3620804623249611
Acuratete maxima obtinuta: 0.5265614581018004 (cu nr de intervale = 2)
```

3.3.1 Matricea de confuzie

```
[[1406.   65.   29.]
 [ 688.  572.  239.]
 [ 583.  526.  391.]]
```

Putem observa faptul ca modelul clasifica bine categoria 0, in timp ce categoriile 1 si 2 contin numere aleatorii.

3.4 References

1. Laboratoarele si cursurile de Machine Learning, Universitatea din Bucuresti
2. Convolutional Neural Networks - Basics to Implementation, Navendu Pottekkat
   https://medium.com/geeythree/convolutional-neural-networks-basics-to-implementation-d9dd96d1
3. https://www.doc.ic.ac.uk/~bkainz/teaching/DL/examinable.pdf
4. Convolutional Neural Network(CNN) Simplified, Renu Khandelwal,
   https://www.doc.ic.ac.uk/~bkainz/teaching/DL/examinable.pdf

5. Max pooling versus average pooling,
   https://www.quora.com/What-is-the-benefit-of-using-average-pooling-rather-than-max-pooling
6. https://www.researchgate.net/figure/Illustration-of-Dropout-Regularization2_fig1_316615383