Monochrome Dreams Classification

~ Dudau Claudia Maria ~

Descrierea problemei

Antrenarea modelelor de invatare automata pentru clasificarea imaginilor alb-negru in 9 clase distincte. Datele de antrenare connstau in 300001 de imagini, cele de validare in 5000 de imagini, iar testarea se realizeaza pe alte 5000 de iamgini.

Metode abordate

1. Clasificatorul Naïve Bayes – 39.16% acuratete

A. Descrierea clasificatorului

Clasificatorul Naïve Bayes este un clasificator probabilistic simplu bazat pe aplicarea teoremei lui Bayes, cu mentiunea ca acesta considera toate trasaturile ca fiind independente intre ele.

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

Asadar, pe problema noastra, modelul clasifica imaginile dupa probabilitatea ca acestea sa apratina unei anumite clasa in fucntie de valorile pixelilor.

Pentru implementare am folosit modelul MultibinomialNB din biblioteca sklearn.

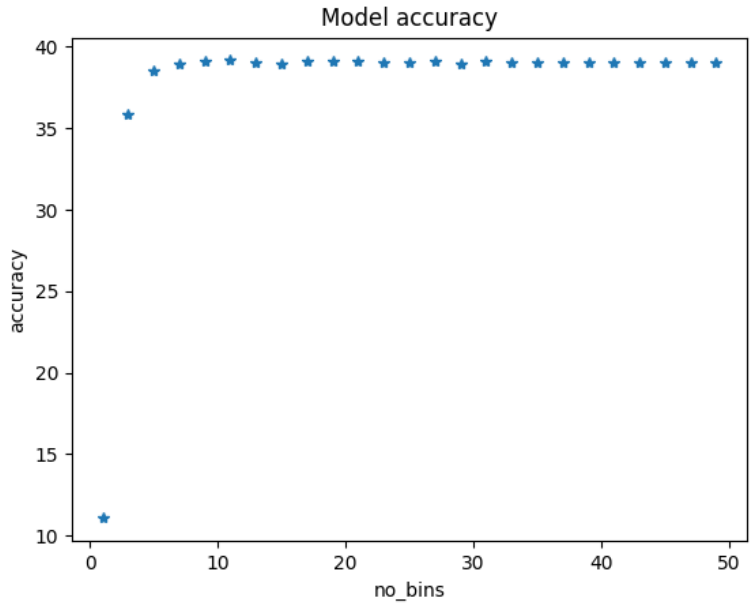
B. Preprocesarea datelor

Intru-cat valorile pixelilor sunt valori continue, este necesara o discretizare a acestor in intervale. Astfel, fiecarui pixel i se va asigna valoarea corespunazatoare intervalului din care face parte.

Pentru a realiza impartirea in intervale, am folosit functiile linspace si digitize din biblioteca numpy.

C. Parametrii utilizati

Pentru rezultatul obtinut, am incercat antrenarea modelului cu diferite valori pentru parametrului no_bins din intervalul [1, 100] (doar valorile imapre) si am constatat ca acuratetea cea mai mare se obtine pentru valoarea 11.



2. Metoda celor mai apropriati vecini

A. Descrierea clasificatorului

Algorimtul celor mai apropriat k vecini (KNN), este o metoda de clasificare neparametrizata care nu are nevoie de o antrenare in prealabil a modelului. Acesta determina eticheta unei mostre din setul de date comparand etichetele celor mai apropiati k vecini ai acesteia si alegand eticheta majoritara. Distantele intre instante se pot calcula cu ajutorul mai multor formule, cele mai poluare fiind distantele l1 si l2.

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

Dat fiind numarul mare de operatii pe care modelul trebuie sa le realizeze: calcularea distantei unei mostre fata de toate celelalte mostre din setul de date,

sortarea mostrelor in functie de distanta fata de mostra curenta si determinarea etichetei corespunzatoare mostrei, timpul de rulare al acestuia poate deveni o probelma, fiind extrem de mare.

B. Preprocesarea datelor

Pentru acest model am ales o normalizare a datelor folosind media si deviatia standard.

C. Parametrii utilizati

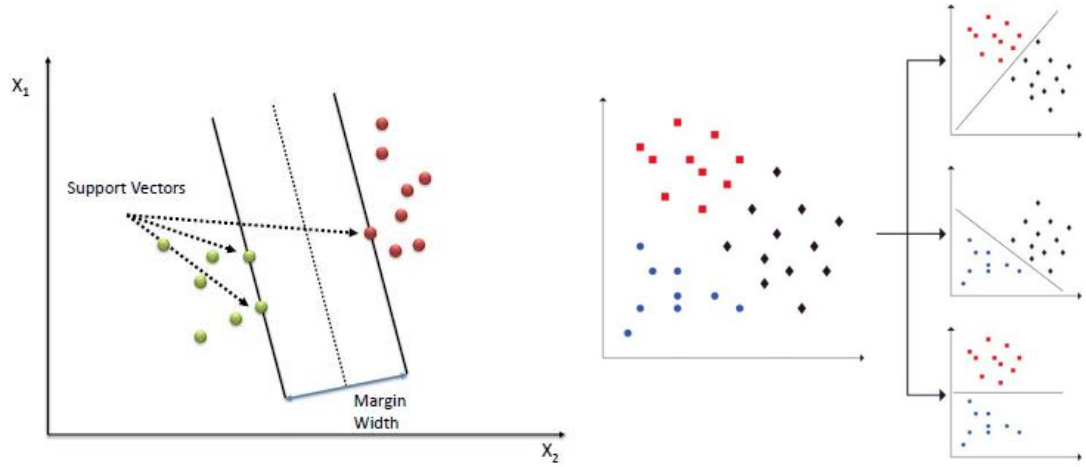
Am testat modelul pe distanta l1, iar pentru numarul de vecini am luat valori din intervalul [1, 50] (doar valorile imapre).

Din cauza numarului foarte mare de date, timpul de executie a fost foarte lung, motiv pentru care dupa ce am asteptat vreo 2 ore, iar modelul nu a terminat de rulat nici macar pentru un, am decis sa il opresc.

3. Clasificare cu suport vectorial – 63.38% acuratete

A. Descrierea clasificatorului

Clasificatorul folosind vectori suport determina o separe liniara cat mai buna intre date, alegand hiperpalnul care maximizeaza marginle dintre clase. Pentru a se asigura separabilitatea claselor, se folosesc functii kernel. Acestea presupun o scufundare a datelor dintr-un spatiu in care acestea nu sunt liniar separabile intr-un alt spatiu unde se poate determina o separare liniara a acestora.



Pentru a se realiza clasificarea datelor in mai multe clase, modelul foloseste abordarea one-vs-one care presupune antrenarea a cate unui clasificator pentru fiecare pereche de cate 2 clase. Astfel, la final, eticheta unei mostre va fi cea care obtine cele mai multe voturi de la acesti clasificatori.

De asemena, acesta ia in considerare si un prametru de penalitate pentru eroare care arata cat de dispus este modelul sa evite clasificarea gresita a datelor. Un paramatru de penalitate mare determina alegerea unui hiperpaln cu o magine mai mica (mai multi vectori suport), pe cand un parametru de penalitate mic detrmina alegerea unui hiperpaln cu o margine mai mare (mai putini vectori suport).

Pentru implementare am folosit modelul SVC din biblioteca sklearn.

B. Preprocesarea datelor

Datele au fost normalizate folosind media si deviatia stanadard.

C. Parametrii utilizati

Pentru antrenarea modelului am folosit un kernel liniar, iar pentru parametrul de penalitate am luat valori din intervalul [10^-10, 10^10]. Cel mai bun rezultata l-am observat pentru un parametru de penalitate de 10^-3.

4. Retea neuronala de perceptroni – 48.39% acuratete

A. Descrierea clasificatorului

Am ales implementarea unei retele de perceptroni de tip fit-forward ce are pe stratul de intrare si pe cele ascunse functia de activare sigmoid, iar pe stratul de iesire functia softmax pentru a putea determina pentru fiecare imagine probabilitatea sa de a apartinei celor 9 categorii in parte.

Pentru fiecare strat, matricele de ponderi isi injumatacesc numarul de linii fata de cele de pe straturile inferioare. Astfel, pe saratul de intrare matricea de ponderi o sa aiba o dimensiune de (512, 1024 – numarul de pixelei ai unei imagini), pe cel de al doilea strat dimensiunea matricei de ponderi o sa fie (256, 512 – numarul de linii ale matricei de ponderi de pe stratul anterior) si asa mai departe. In cazul in care, prin injumatatirea numarului de linii, se ajunge la o valoare mai mica decat 9 (numarul de clase), atunci matricele de ponderi o sa aiba dimensiunile de (9, 16) pentru matricea de pe stratul 7, respectiv (9, 9) pentru matricele de pe straturile 8 si mai departe. Pe stratul de

iesire matricea de ponderi o sa aiba o dimensiune de (9 – numarul de clase, numarul de linii ale matricei de ponderi de pe stratul inferior). In ceea ce priveste bias-ul, acesta o sa aiba dimeniunea (numarul de linii alea matricei de ponderi de pe stratul sau, 1). Valorile iniatiale ale ale acestora sunt generate aleator din distributia normala standard.

Actualizarea ponderilor si a bias-urilor se realizeaza folosind algoritmului de coborare pe gradient. Acesta presupune calcularea derivatelor functiilor de activare in functie de ponderi si bias pentru fiecare strat, valori care, inmultite cu rata de invatare, se scad din valorile ponderilor si bias-urilor de pe stratul corespunzator.

Functia de cost folosita este cea de cross-entropy.

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

B. Preprocesarea datelor

Datele au fost normalizare folosind normalizarea L2 care le transforma astfel incat suma patratelor valorilor sa fie cel mult 1.

$$S = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Dupa normalizare, am impartiti datele in 10 parti dintre care se iau pe rand 9 parti pentru antrenarea modelului si o parte pentru validarea acestuia. Pentru a imparti setul de date in aceste parti am folosit functia KFold din biblioteca sklearn.

C. Parametrii utilizati

Dupa mai multe incercari, schimband tipul de normalizare al datelor (standard, minmax, l1, l2), rata de invatara (de la 0.1 la 0.001), numarul de straturi (de la 1 la 3) si numarul de iteratii (de la 1000 la 3000), ceea ce a rezultat in rularea a aproximativ 20 de modele diferite, am ajuns sa folosesc o retea cu 2 straturi, o rata de invatare de 0.1 si o durata de 200 de iteratii pentru fiecare combinatie de 9 parti ale multimii de antrenare.

5. Retea neuronala convolutionala – 87.37% acuratete

A. Descrierea clasificatorului

In implementare am folosit functionalitatile oferite de bibliotecile tensorflow si keras.

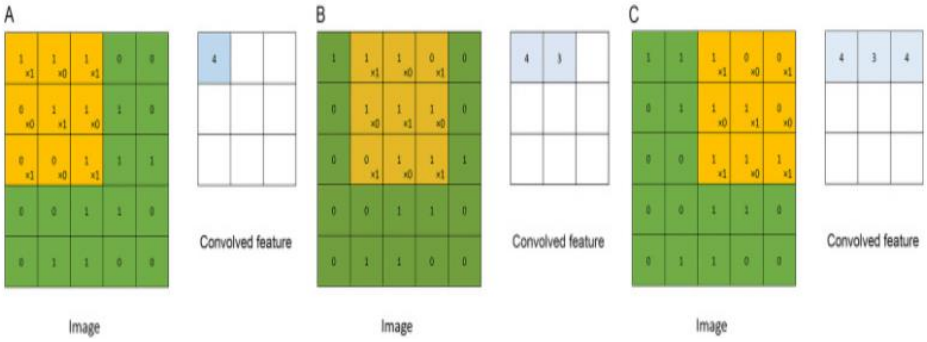
Modelul antreneaza o retea convolutionala de neuroni, pe o durata data de iteratii si o rata de invatare, prin trecerea datelor de intrare prin toate starturile retelei si reactualizand ponderile si bias-urile straturilor, prin intermediul algoritmului de coborare pe gradient, in scopul de a minimiza functia de pierdere.

Am ales implementarea unui model secvential, pentru a realiza o parcurgere liniara a straturilor de neuroni. Straturile folosite (nu neaparata in aceasta ordine) sunt urmatoarele:

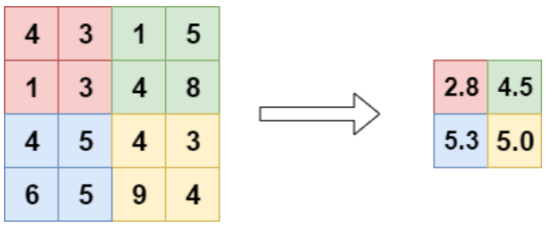
BatchNormalization: aplica o transformare asupra setului de date care mentine media aproape de 0 si deviatia standard aproape de 1.

Flatten: aplatizeaza datele, aducandu-le la o singura dimensiune.

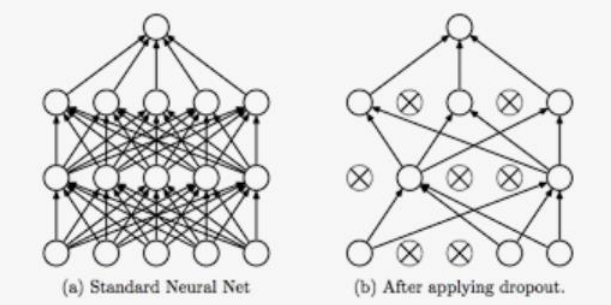
Conv2D: creaza un kernel convolutional de o dimensiune data, care aplicat pe fiecare zona a imaginii determina matricele de trasaturi, prin calcularea produsului scalar dintre fereastra convolutionala si zona imaginii, care ulterior produc un tensor, din fiecare matrice de trasaturi luandu-se suma valorilor sale.



AveragePooling2D: aplica o reducere a dimensiunii, luand din fiecare zona a imaginii de dimensiune fixa media valorilor.

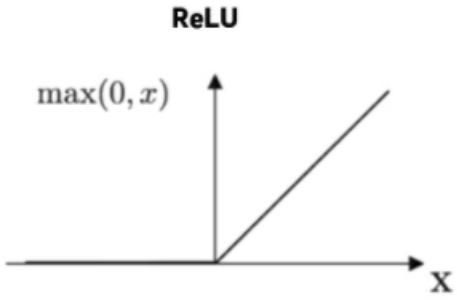


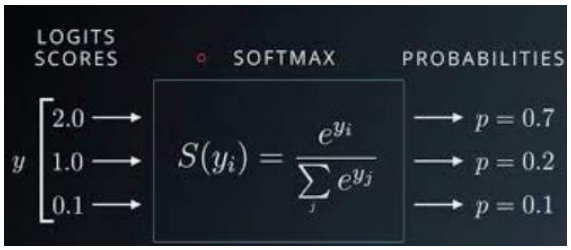
Dropout: inactiveaza in mod aleator diferiti neuroni, prin setarea la zero a anumitor unitati din input, pentru a se evita fenomenul de overfitting.



Dense: un strat de neuroni dens conectati ce implementeaza operatia: activation(dot(input, weight matrix) + bias).

Toate straturile folosesc functia de activare de rectificare liniara unitara (relu), mai putin ultimul strat (cel de output) care foloseste functia de activare softmax pentru a putea determina probabilitatile pentru fiecare imagine de a apartine unei anumite clase.





Pentru regularizare folosesc regularizarea de tip l2, astfel ponderile apropriate de 0 au un efect mai mic asupra complexitatii modelului fata de cele mai departate de 0.

$$L_2 \text{ regularization term} = ||\boldsymbol{w}||_2^2 = w_1^2 + w_2^2 + \ldots + w_n^2$$

Funnctia de pierdre folosita este cea de categorical cross entropy (functia de pierdere pentru softmax).

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output} \\ \text{size}}} y_i \cdot \log \hat{y}_i$$

De asemenea, folosesc si un optimizator in scopul de a adapta ponderile si rata de invatare a retelei pe parcursul algoritmului, pentru a ajunge la un rezultat cat

mai bun. Optimizatorul folosit este Nadam (Nesterov-accelerated Adaptive Momentum Estimation). Astfel procesul de invatare este accelerat prin calcularea unor rate de invatare adaptive individuale pentru fiecare parametru folosind estimarile primului si celului de al doilea moment atat al gradientului curent, cat si al celui anterior.

B. Preprocesarea datelor

Datele au fost normalizate folosind normalizarea minmax care transforma cea mai mica valoare in 0, cea mai mare valoare in 1, restul valorilor se vor gasi in intervalul (0, 1).

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

C. Parametrii folositi

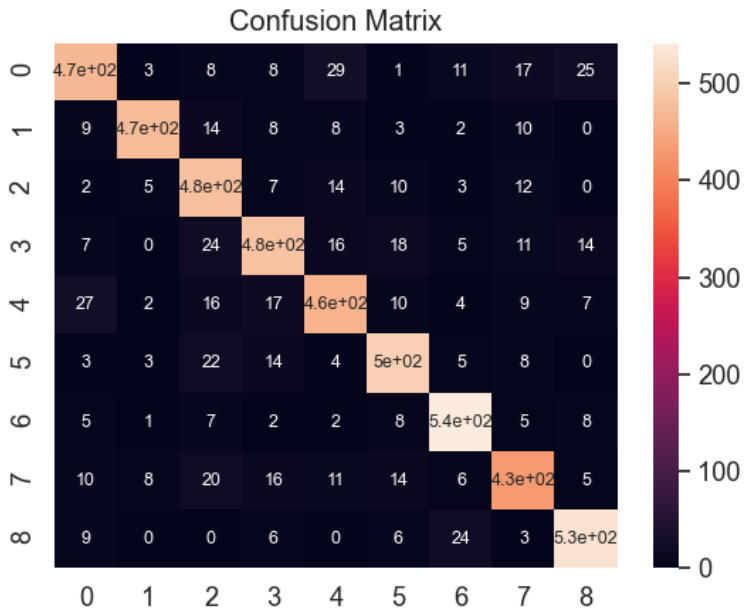
Pentru antrenarea modelului am testat mai multi parametrii: diferite modalitati de normalizare a datelor (standard, minmax), batch size-uri diferite (de la 64 la 512), numarul de epoci (de la 10 la 25), diferiti optimizatori (adam, nadam, sgd), mai multe functii de activare (relu, sigmoid, tanh), precum si diferite valori pentru hiperparametrii straturilor (numarul de filtre de la 16 la 256, strides de 1 sau 2, dimensiunea stratului de pooling intre 2 sau 3, numarul de neuroni de pe straturile dense de la 126 la 600). Astfel am obtinut acurateti intre 80 – 87% folosind aproximativ 50 de modele diferite.

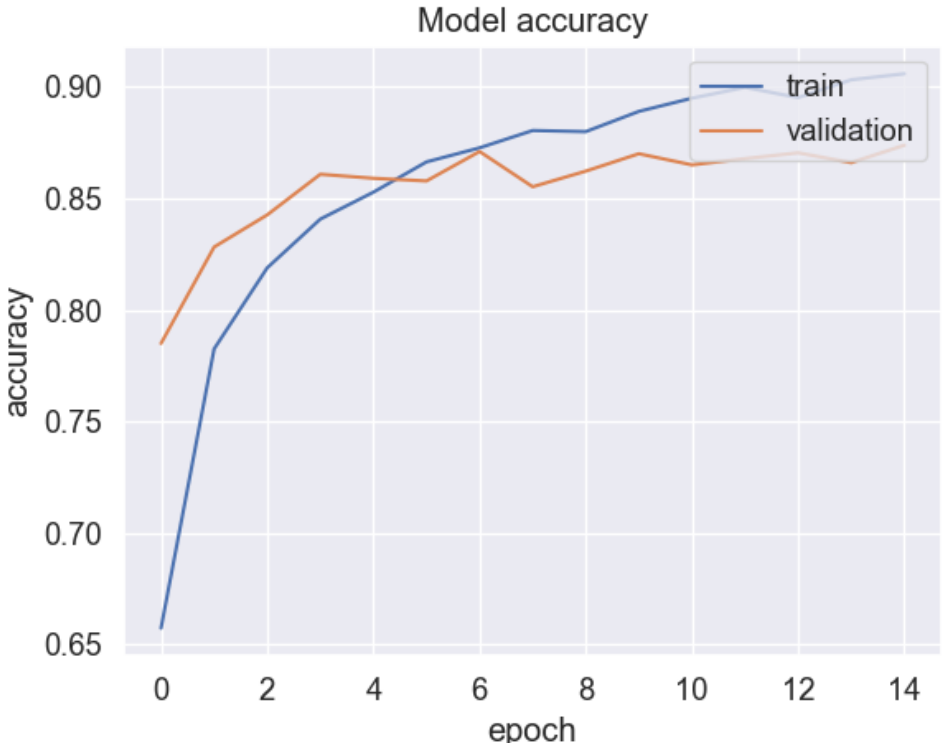
In final am ajuns sa folosesc un batch size de 64 si un numar de 15 epoci. De asemenea, am folosit 32 filtre pentru cele ambele straturi convolutionale, un pool size de (2, 2), rata de dropout de 0.4 si 0.5, iar pentru straturile dense am folosit 600 neuroni pentru stratul cu functia de activare relu, respectiv 9 neuroni pentru stratul de output ce are functia de activare softmax (fiecare neuron corespunde unei clase).

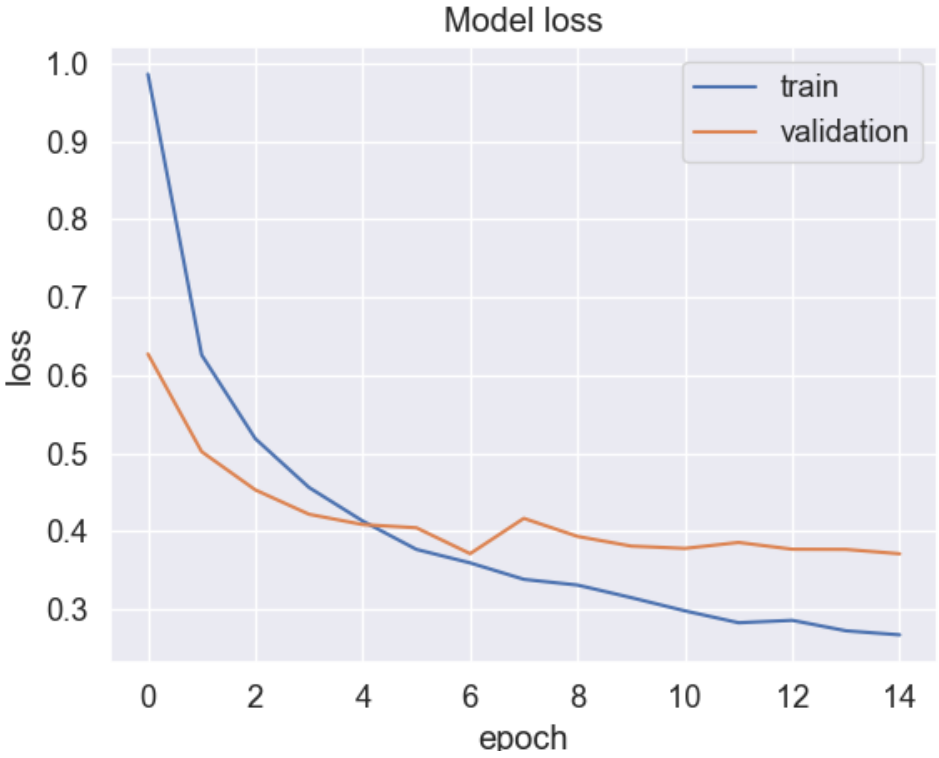
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        832
_________________________________________________________________
average_pooling2d (AveragePo (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 32)        25632
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dropout (Dropout)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 600)               4915800
_________________________________________________________________
dropout_1 (Dropout)          (None, 600)               0
_________________________________________________________________
dense_1 (Dense)              (None, 9)                 5409
=================================================================
Total params: 4,947,673
Trainable params: 4,947,673
Non-trainable params: 0
```

Concluzii

Dupa cum se observa, din modelele prezentate mai sus, modelul care a dat rezultatele cele mai bune pe setul de date este reteaua neuronala convolutionala.




Model accuracy
train
validation
accuracy
epoch
0.90
0.85
0.80
0.75
0.70
0.65
0
2
4
6
8
10
12
14


Model loss
train
validation
loss
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
2
4
6
8
10
12
14