Postolache Andreea Miruna

Grupa 242

Proiect inteligență artificială
Deep Hallucination Classification

Universitatea București, Facultatea de Matematică și Informatică, Anul 2, Grupa 242

Cuprins documentație

1. Introducerea

În cadrul acestui proiect am ales să clasific imaginile apelând la mai mulți clasificatori, precum: Naive Bayes, K-NN, SVM și CNN.

Primul pas pentru a ajunge la aplicarea acestor clasificatori este citirea și salvarea datelor. Se poate observa faptul că fișierul „train.txt" și „validation.txt" au pe prima linie un text „id,label" care este urmat de linii de forma: id-ul pozei (în format .png), eticheta corespunzătoare. Spre deosebire de acestea, fișierul „test.txt" are pe prima linie textul „id", fiind urmat de linii care conțin doar id-ul pozei (format .png). Pentru a citi informația din aceste fișiere s-a făcut o funcție numită „citire" care primește ca parametru numele fișierului .txt din care se dorește preluarea datelor.

În cazul fișierelor de antrenare și validare se vor inițializa două liste:

- „datele" care va conține imaginile din setul de antrenare/validare (în funcție de ce fișier este dat ca parametru); imaginile vor fi încărcate din fișierul „train+validation" prin utilizarea funcției mping.imread care le va transforma în matrici.
- „etichetele" care va fi o listă cu etichetele pozelor (numere întregi – eliminarea caracterului '\n' se face prin selectarea doar a primului caracter).

În cazul fișierului de testare se vor inițializa tot două liste:

- „datele" care va conține imaginile din setul de test; pentru a prelua corect id-ul pozei se va face un split după '.', iar pentru a putea încărca poza cu funcția mping.imread se va adăuga la finalul id-ului extensia '.png' (deoarece aceasta a fost înlăturată la split).
- „id_uri_test" care va conține id-ul tuturor pozelor din fișierul „test.txt".

În urma apelării funcției citire de către fișierele de antrenare, validare și testare se vor obține 6 liste:

- datele_antrenare, etichetele_antrenare
- datele_validare, etichetele_validare
- datele_test, id_uri_test

Primele 5 vor fi transformate în np.array, pentru a putea da reshape la: datele_antrenare, datele_validare, datele_test care din 4Dimensional (numărul de date, înălțimea, lățimea, canalul de

culori(=3)) ajung 2Dimensional (numărul de date, înălțimea*lățimea*canalul de culori(=3)).

Această transformare a fost făcută în două etape: cu funcția shape am aflat parametrii descriși mai sus, iar cu funcția reshape s-a schimbat în 2Dimensional.

Mai mult, transformarea a fost necesară pentru că funcția de antrenare a modelului (fit) și cea de prezicere a etichetelor (predict) pot lucra pe date cel mult 2Dimensionale.

În acest mod datele au ajuns la o formă care poate fi folosită pentru antrenarea modelelor obținute cu clasificatorii doriți și prezicerea etichetelor pe datele de test.

2. Clasificatorul Naive Bayes

2.1. Descriere model

Acest tip de clasificator are la bază teorema lui Bayes. În cadrul antrenării se consideră că fiecare pixel al imaginii este un atribut independent în ceea ce privește calculul probabilității apartenenței imaginii I (pixelii: i1,i2,...) la o clasa c1.

$$P(c1|I) = \frac{P(i|c1) * P(c1)}{P(I)}$$

Unde P(i|c1) este probabilitatea de a avea atributul i în clasa c1 și P(c1) este probabilitatea ca un atribut să se afle în clasa c1.

Prezicerea etichetelor: probabilitatea ca imaginea I să fie în clasa c1 este dată de formula de mai jos, iar această probabilitatea va fi aplicată pentru toate cele 7 clase, eticheta finală fiind dată de clasa care are probabilitatea cea mai mare.

$$\mathrm{P(c1|I)} = \mathrm{p(c1)} \prod_{j=1}^{768} P(i_j | c1)$$

În cadrul proiectului am folosit acest clasificator așa cum urmează:

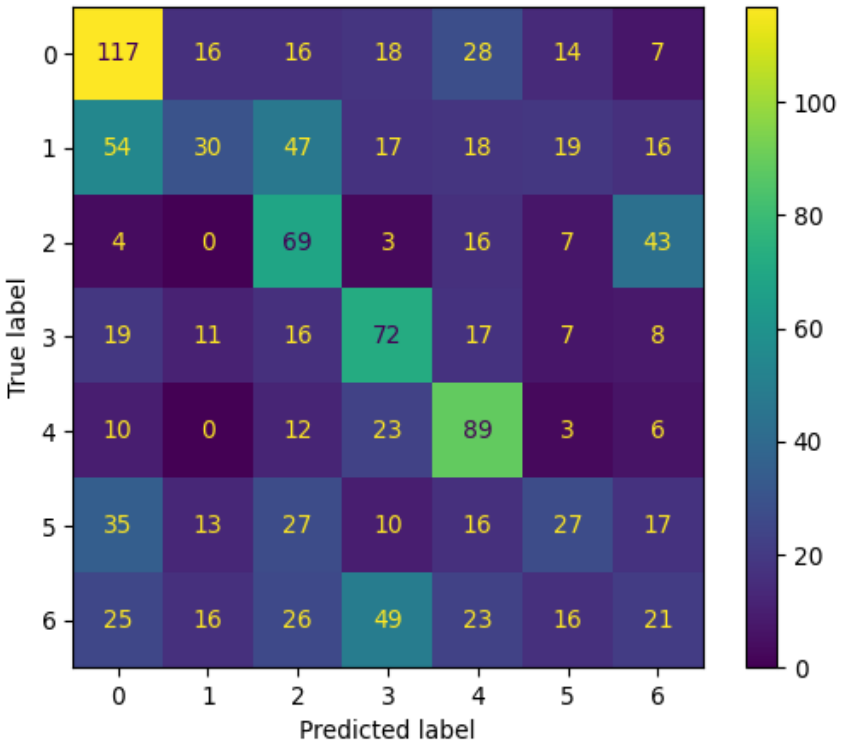
după ce s-a aplicat funcția de reshape pe datele de antrenare, validare și testare, s-a creat un model de tipul MultinomialNB care a fost antrenat (folosindu-se funcția fit) pe datele de antrenare cu etichetele de antrenare. Aplicând funcția score pe datele de validare se va afla acuratețea modelului. În continuare se va aplica funcția predict care va prezice etichetele pe setul de test.

2.2. Acuratețe

După aplicarea funcției score, pe setul de validare acuratețea este de 0.**36231884057971014**, iar pe 25% din datele de testare acuratețea este de 0.**34659**.

2.3. Matrice de confuzie

Pentru realizarea acestei matrici de confuzie a fost nevoie de lista cu etichetele din validare și prezicerile etichetelor pe datele de validare făcute de model. Pentru reprezentarea sub această forma a matricei s-a folosit funcția ConfusionMatrixDisplay care primește ca parametru matricea de confuzie calculată cu ajutorul funcției confusion_matrix.



3. Clasificatorul celor mai apropiați K vecini (KNN)

3.1. Descriere model

Acest clasificator funcționează astfel: pentru fiecare imagine din setul de testare se găsesc cei mai apropiați K vecini, iar eticheta atribuită imaginii va fi cea care este întâlnită majoritar la acei K vecini. Pentru a afla cei mai apropiați K vecini se utilizează funcții de distanță, precum: Euclidiană (L2), Manhattan (L1), Minkowski (Lp) (este o generalizare a primelor două) etc.

Distanta Minkowski:

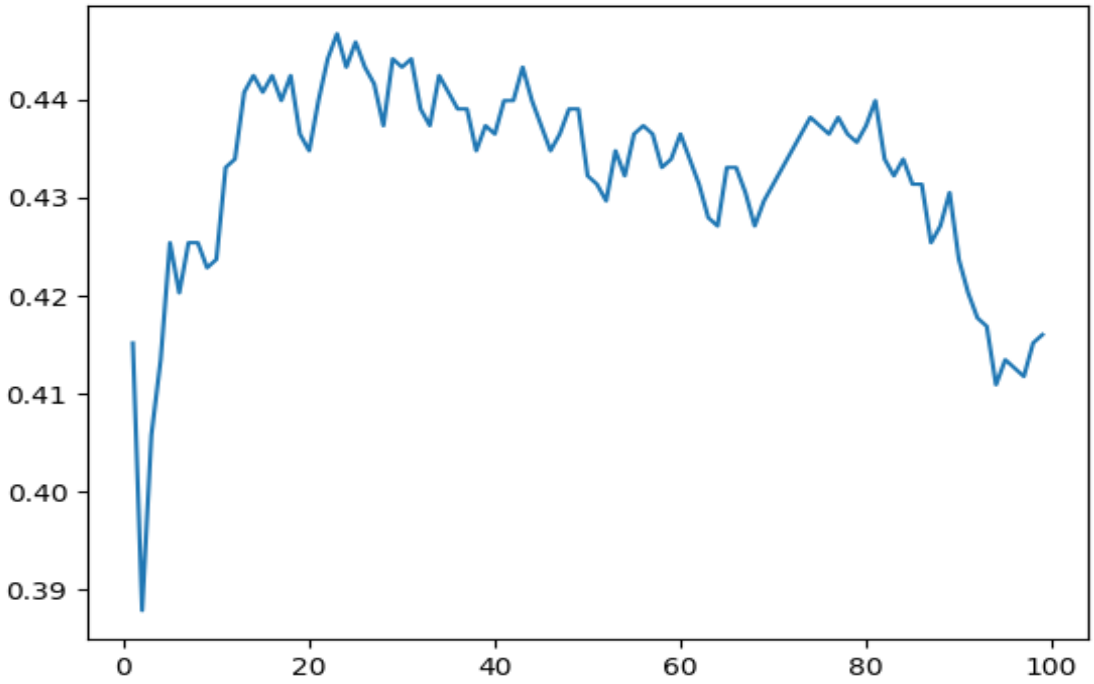
$$d_{L_p}(x, y) = \sqrt[p]{|x_1 - y_1|^p + \cdots + |x_n - y_n|^p}$$

În cadrul proiectului am importat clasificatorul KneighborsClassifier din biblioteca sklearn.neighbors. Acesta are default ca tip de distanță utilizată, distanța Euclidiană (L2) și i-am dat ca parametru numărul de vecini. L-am antrenat prin aplicarea funcției fit pe datele de testare, am afișat acuratețea obținută pe datele de validare, urmând să aplic funcția de predict pe datele de testare pentru a obține etichetele prezise.

3.2. Alegerea hiperparametrului potrivit

Numărul de vecini ales influențează acuratețea modelului. Dacă numărul de vecini este foarte mic atunci pe setul de antrenare eroarea de clasificare este mai mică decât cea pe setul de testare. Cu cât crește numărul de vecini cu atât predicțiile pe datele de testare devin mai bune până la o anumită valoarea a lui K.

În ceea ce privește alegerea acestui hiperparametru K pentru modelul din competiție am făcut un grafic care pentru K din intervalul [1,99] calculează acuratețea pe datele de validare (se construiește modelul, se antrenează și se reține acuratețea obținută și numărul de vecini).



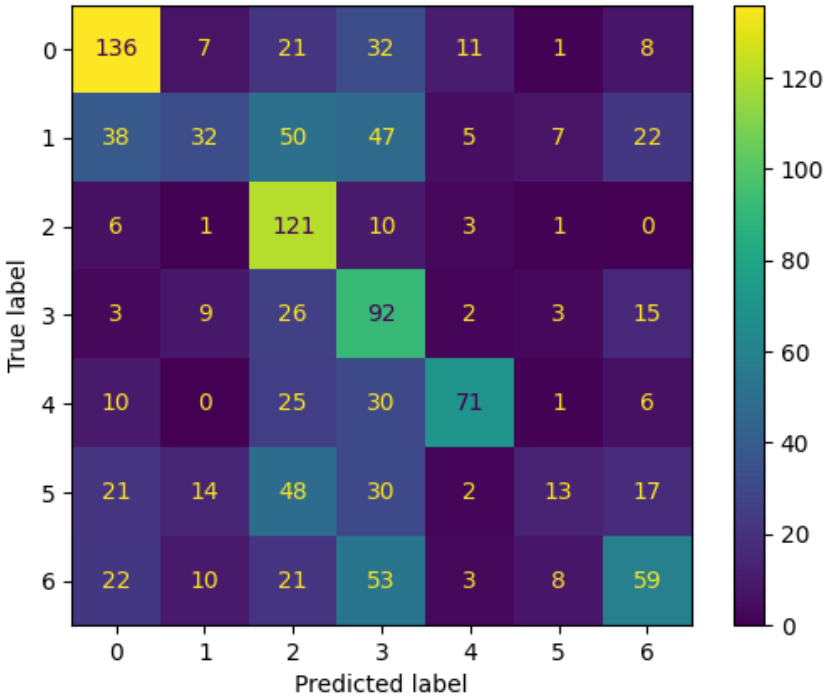
Se poate observa faptul că în intervalul [22, 22+puțin] este cea mai mare acuratețe (>0.44) dintre cele ale modelelor cu K cuprins între 1 și 99.

Astfel, alegem să facem un model cu numărul de vecini egal cu 23 și altul cu 29.

3.3. Acurateţe şi Matrice de confuzie

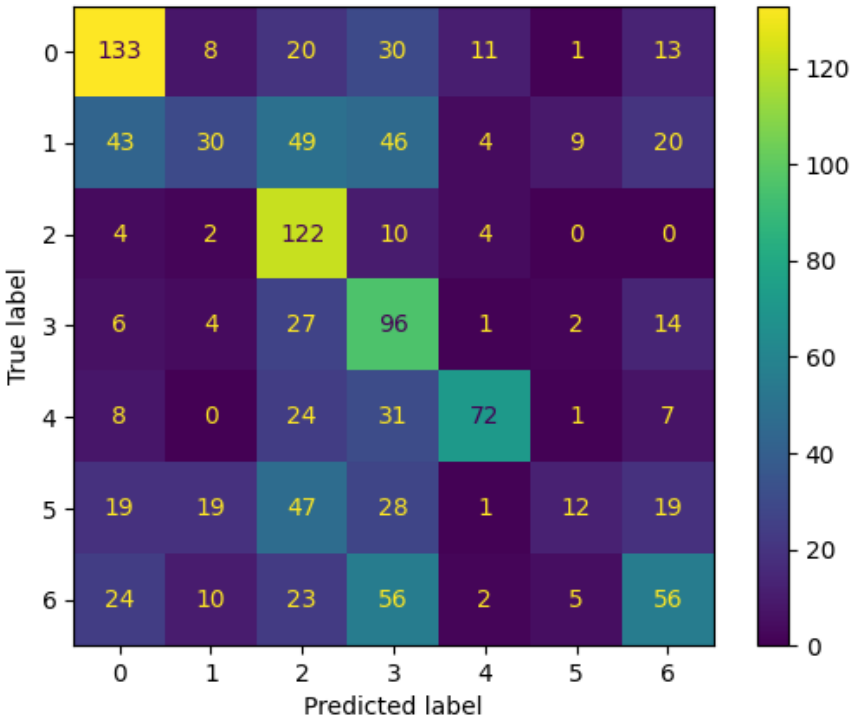
Pentru K = 23:

- acuratețea pe datele de validare : 0.44671781756180734, iar pe 25% din datele de testare: 0.42045



Pentru K = 29:

- acuratețea pe datele de validare : 0.4441602728047741, iar pe 25% din datele de testare: 0.42329



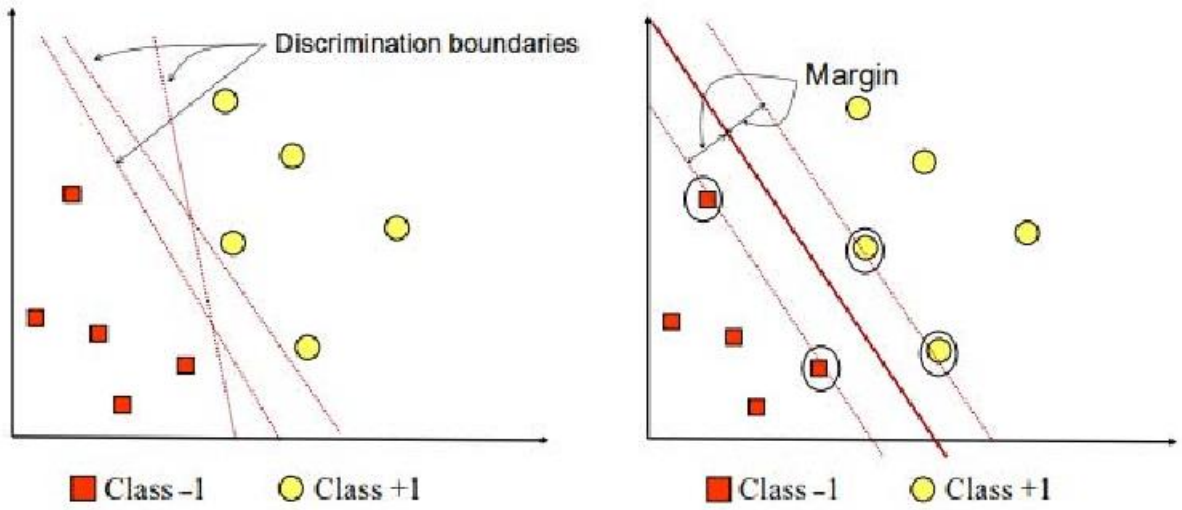
Astfel, se remarcă faptul că deși pentru K = 23 acuratețea pe setul de validare este mai mare decât cea pentru K = 29, pe setul de testare este invers.

Se poate observa că spre deosebire de clasificatorul Naive Bayes, clasificatorul KNN este mai bun, acesta realizând o creștere a acurateții pe datele de testare de la 0.34659 la 0.42329.

4. Clasificatorul cu vectori suport (SVM)

4.1. Descriere model

Clasificatorul SVM are la bază conceptul maximizării marginii care delimitează entitățile din două clase diferite (sunt căutate hiperplanele care maximizează marginea dintre cele două clase).



În exemplul de mai sus hiperplanul care maximizează marginea este reprezentat în figura din dreapta.

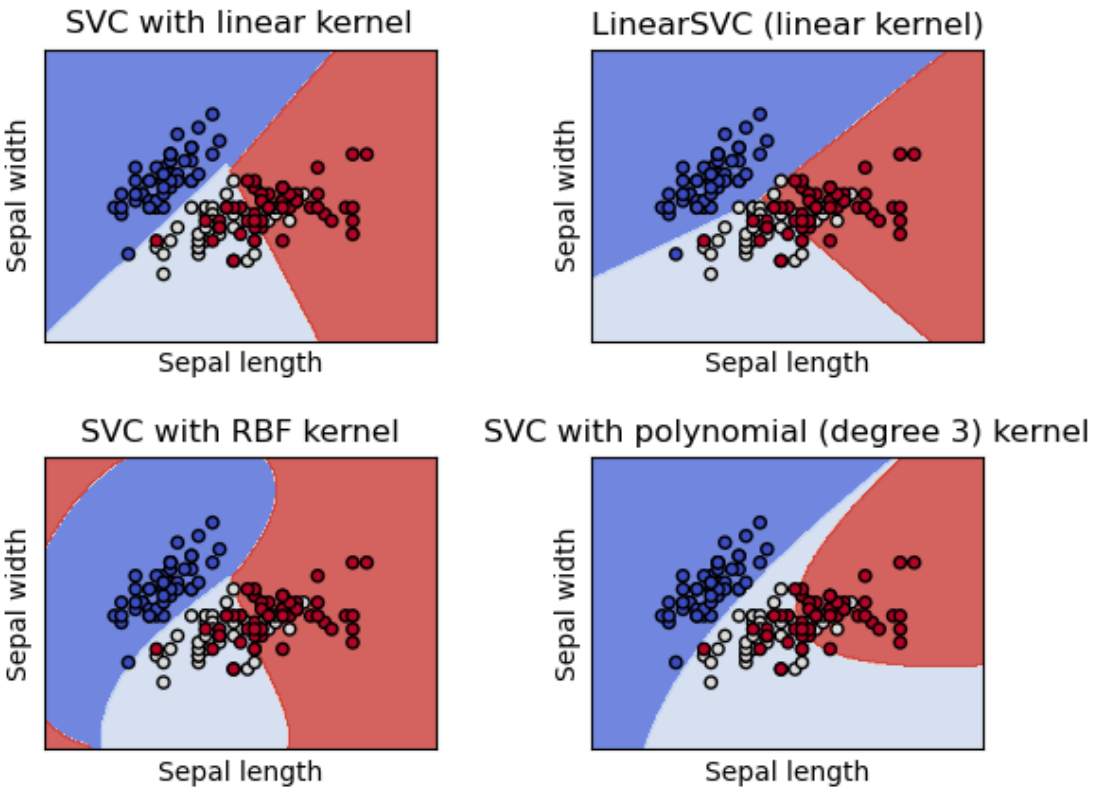
În cazul în care sunt prezente mai multe clase, ceea ce presupune o clasificare multiplă rezultă două tipuri de clasificări: One-versus-one, One-versus-all.

Metoda One-versus-all presupune antrenarea a 7 clasificatori (în cazul nostru), unul pentru fiecare clasă care asigură diferențierea unei clase de restul claselor. Eticheta prezisă pentru o imagine din setul de testare este dată de clasificatorul cu scorul maxim.

Metoda One-versus-one presupune clasificarea claselor două câte două, iar eticheta prezisă pentru o imagine din setul de testare este aceea care o să aibă voturi maxime în urma antrenării clasificatorilor.

În cadrul proiectului am importat modelul SVC din svm (care la rândul ei este importată din biblioteca sklearn), acesta apelând la metoda de clasificare One-versus-one. Acest clasificator are drept parametrii:

- funcția kernel (aceasta scufundă datele într-un spațiu cu mai multe dimensiuni – atâtea câte clase sunt; transformă relațiile neliniare din spațiul inițial în relații liniare) care poate fi: linear, rbf, sigmoid, poly.



- C, parametru de penalitate pentru eroare, care în funcție de valoarea primită poate să evite să clasifice greșit imaginile din setul de antrenare. Dacă valoarea lui C este mare, atunci se va alege un hiperplan cu marginea mai mică, iar în cazul în care valoarea este una mică se va alege un hiperplan cu marginea mai mare. Nu este bine

dacă C este ales fie prea mare, fie prea mic, deoarece se obține supraînvățare, respectiv subînvățare.

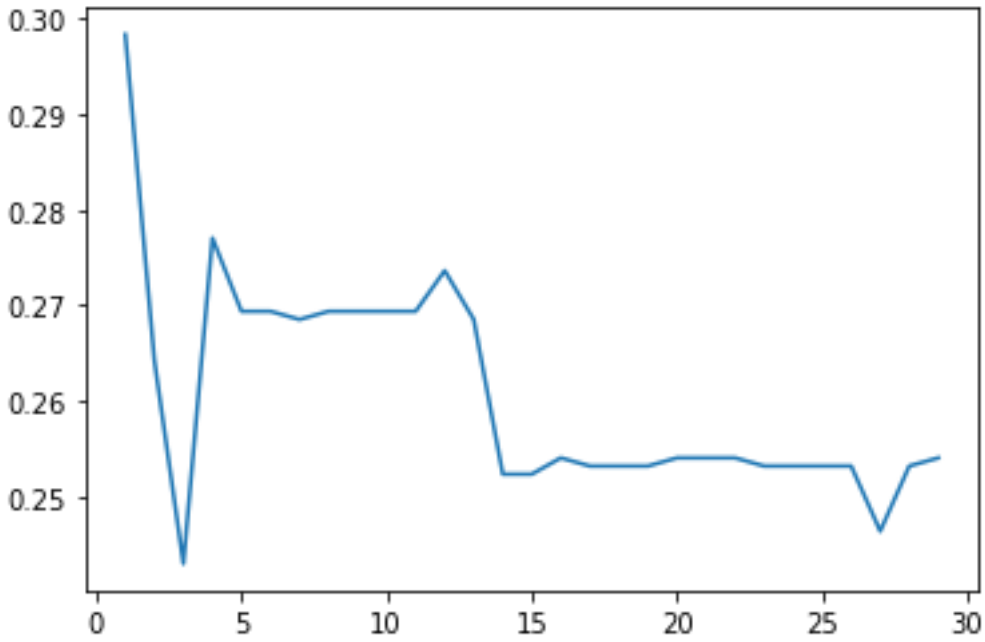
4.2. Preprocesare

După ce s-a dat reshape la datele de atrenare, validare și testare s-a folosit funcția StandardScaler importată din librăria sklearn.preprocessing, care presupune standardizarea datelor prin scăderea imaginii medie și împărțirea rezultatului obținut la deviația standard. Se aplică acest tip de scalare pe datele de antrenare, validare și testare.

4.3. Alegerea funcției Kernel și a parametrului C potrivit

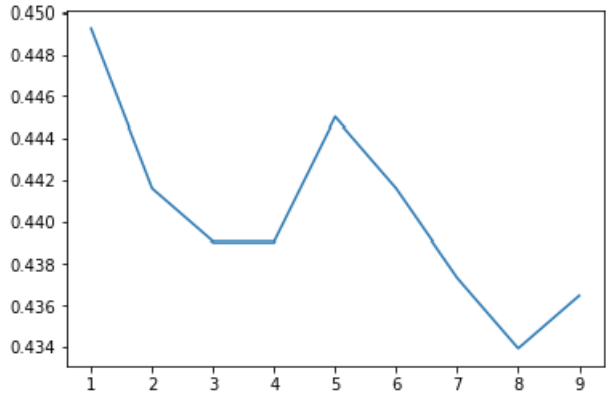
Pentru a vizualiza mai bine care este cea mai bună funcție Kernel pentru setul de date am antrenat pentru fiecare tip câte 29 de modele, care luau valoarea lui C în intervalul [1,29] și am făcut un grafic în funcție de K și valoarea obținută pentru acuratețe (prin aplicarea funcției score) pe setul de validare.

Kernel = sigmoid



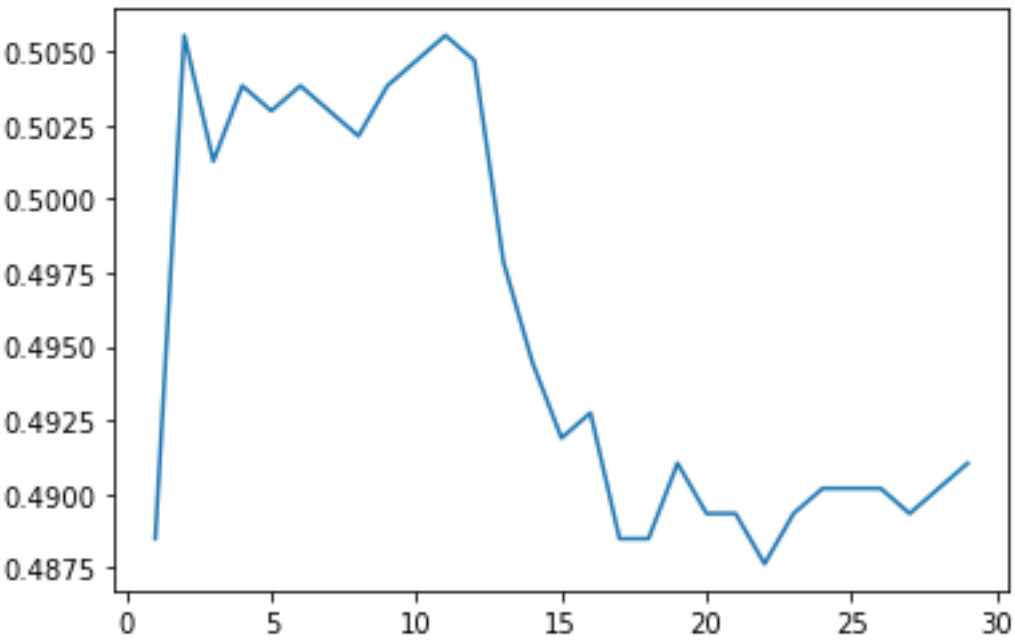
Kernel = linear (acestuia îi ia foarte mult timp să se antreneze)

```
1 0.4492753623188406
2 0.4416027280477408
3 0.4390451832907076
4 0.4390451832907076
5 0.44501278772378516
6 0.4416027280477408
7 0.4373401534526854
8 0.4339300937766411
9 0.43648763853367434
```

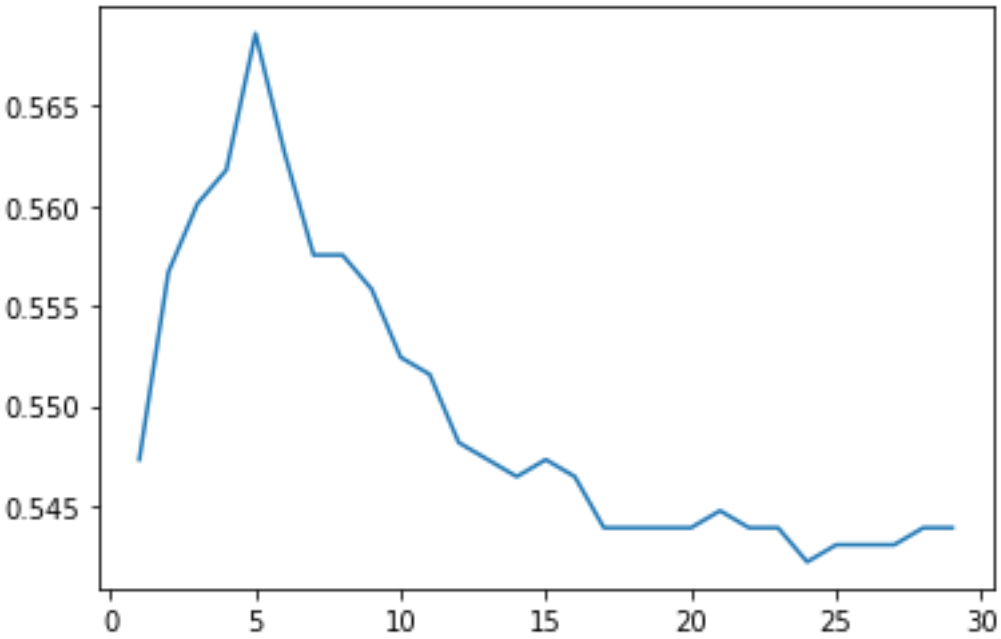


Pentru C = 30 valoarea acurateții pe datele de validare este 0.42028985507246375.

Kernel = poly



Kernel = rbf



Astfel, se poate observa că funcția Kernel de tip rbf obține cele mai bune rezultate.

4.4. Matricea de confuzie și acuratețea

Valorile obținute fără scalare

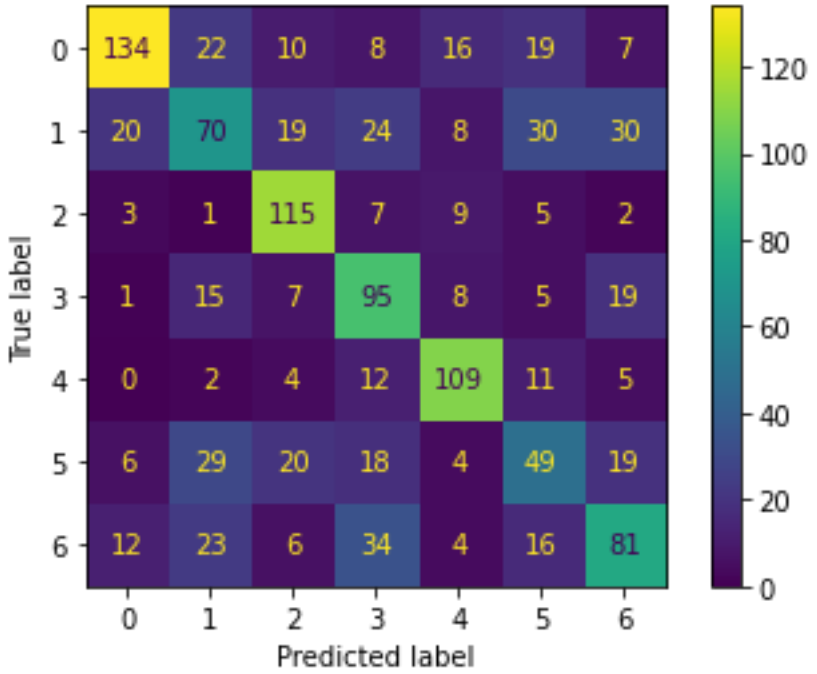
C	Kernel	Acuratețe validare
1	linear	0.4799658994032396
1	rbf	0.5413469735720375

Valori obținute după ce datele au fost și scalate

C	Kernel	acuratețe validare	acuratețe test
1	rbf	0.5473145780051151	0.56392
5	rbf	0.5686274509803921	0.56250
2	rbf	0.556692242114237	0.57528
7	rbf	0.5575447570332481	0.55823
9	rbf	0.5558397271952259	0.55397

Astfel, se poate observa că pentru modelul care are C=2 și Kernel de tip rbf se obține cea mai mare acuratețe.

Matricea de confuzie pentru modelul mai sus menționat se obține prin aplicarea funcției confusion_matrix care are ca parametrii lista cu etichete de validare și lista cu etichetele prezise de model pentru setul de validare.



Se poate remarca faptul că modelul SVM este mai bun decât KNN și a reușit să ridice valoarea acurateței pe datele de testare de la 0.42329 la 0.57528.

5. Rețea neuronală convoluțională (CNN)

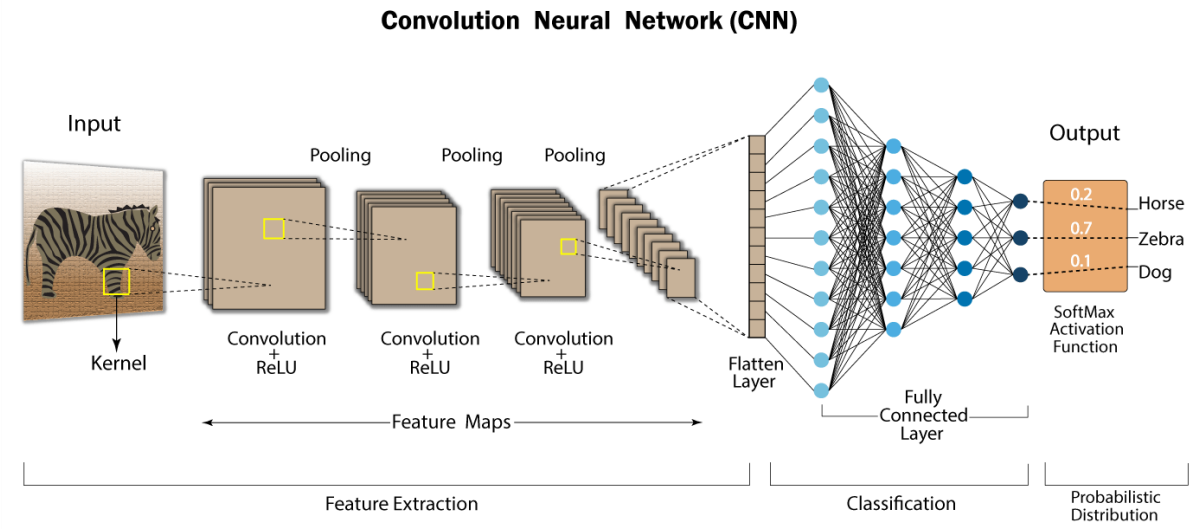
5.1. **Descriere model**

Rețeaua neuronală convoluțională este alcătuită dintr-un strat de input, straturi ascunse și din stratul de ieșire. Această rețea poate fi văzută ca fiind formată din două părți: cea care conține o succesiune de straturi convoluționale, de pooling, de dropout și cea care conține straturi fully connected.

Stratul de input primește o poză asupra căreia vor surveni modificări specifice straturilor convoluționale și de pooling, urmând ca rezultatul să treacă prin straturi fully connected, iar ultimul strat

al rețelei are drept scop încadrarea pozei într-o clasă (acest ultim strat este reprezentat printr-o listă în care sunt specificate probabilitățile ca imaginea intrată pe stratul de input să facă parte din clasa_i unde clasa_i ia valori în intervalul [0,6]).

În continuare se vor prezenta tipurile de straturi prezente în CNN:



1. Stratul de input primește o imagine de dimensiunea 16x16x3.
2. Stratul de convoluție presupune extragerea trăsăturilor din imagine.
   - În cadrul acestor straturi se utilizează o matrice mai mică (nucleul/kernel) care va parcurge line cu linie imaginea din input (dacă este primul strat) sau matricea stratului obținută anterior – harta de trăsături și va învăța anumite trăsături ale imaginii.
   - Un rol important îl au și numărul de filtre utilizate.
   - Dimensiunea utilizată pentru kernel este stabilită în funcție de posibilitatea recunoașterii imaginii: kernel de dimensiune mare se folosește dacă se consideră că pentru recunoașterea pozei este nevoie de un număr mai mare de pixeli, iar kernelul de dimensiune mai mică se folosește dacă se observă că imaginile se diferențiază între ele prin trăsături mici.
   - Funcția de activare folosită este „relu”, deoarece aceasta este foarte eficientă computațional și converge mult mai rapid.
3. Stratul de pooling reduce dimensiunea hărților de trăsături obținute în stratul anterior. În cadrul proiectului am folosit dimensiune pentru pooling de 2x2 și stratul de pooling de tipul MaxPooling2D. Astfel, matricea este împărțită în pătrate de 4 pixeli și din fiecare grup de 4 se alege pixelul cu valoarea cea mai mare. În acest mod dimensiunea pozei ajunge la jumătate.

4. Stratul de droput are scopul de a tăia din legăturile cu neuronii din stratul anterior pentru a combate supraînvățarea. Valoarea primită de droput nu trebuie să fie foarte mare, pentru că astfel modelul nu mai poate învăța corect.
5. Stratul de flatten are rolul de a transforma hărțile de trăsături într-un vector pentru a putea fi input pentru partea cu straturi fully connected.
6. Stratul dens ajută la clasificarea trăsăturilor imaginii și decide probabilitatea ca imaginea să aparțină clasei X (unde X aparține intervalului [0,6]). Aceste tipuri de straturi au și ele funcții de activare, iar în competiție s-a folosit „tanh" și „softmax"(pentru ultimul strat).

5.2. Preprocesare

După ce datele au fost transformate în np.array, s-au împărțit datele de antrenare, validare și testare la 255 (valoarea maximă care putea fi luată de un pixel) pentru a le aduce în intervalul [0,1]. Mai mult, etichetele vor fi transformate în vectori prin aplicarea funcției to_categorical importată din tensorflow.keras.utils; dimensiunea vectorului este egală cu numărul de clase (7) și este populat peste tot cu 0, doar pe poziția = valoarea etichetei are valoarea 1.

5.3. Alegerea distribuției de layere potrivite

Cum această rețea nu are un tipar universal valabil am încercat foarte multe variante de distribuții.

Am importat straturile din biblioteca keras.layers: Dense, Dropout, Activation,Conv2D,MaxPooling2D,Flatten. Rețeaua este una de tip secvențială (conține o stivă liniară de straturi).

Prima încercare – cod1 – (datele au respectat preprocesarea descrisă mai sus) în găsirea celei mai bune distribuții a fost de următorul tip:

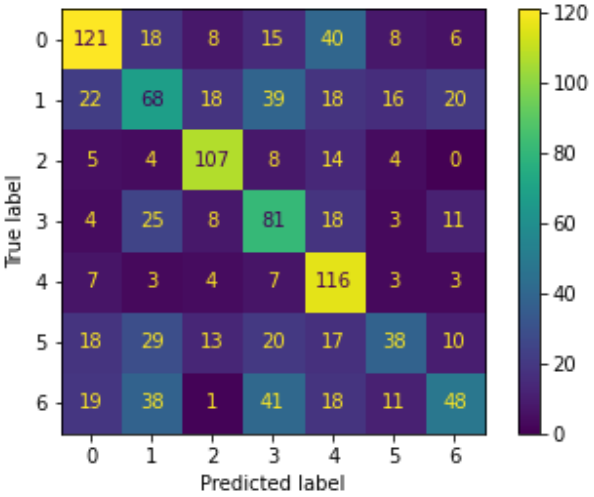
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 14, 14, 16)        448

max_pooling2d_6 (MaxPooling  (None, 7, 7, 16)          0
2D)

conv2d_7 (Conv2D)            (None, 5, 5, 32)          4640

max_pooling2d_7 (MaxPooling  (None, 2, 2, 32)          0
2D)
```

```
 flatten_3 (Flatten)         (None, 128)               0

 dense_6 (Dense)             (None, 101)               13029

 dense_7 (Dense)             (None, 7)                 714

=================================================================
Total params: 18,831
Trainable params: 18,831
Non-trainable params: 0
_________________________________________________________________
```

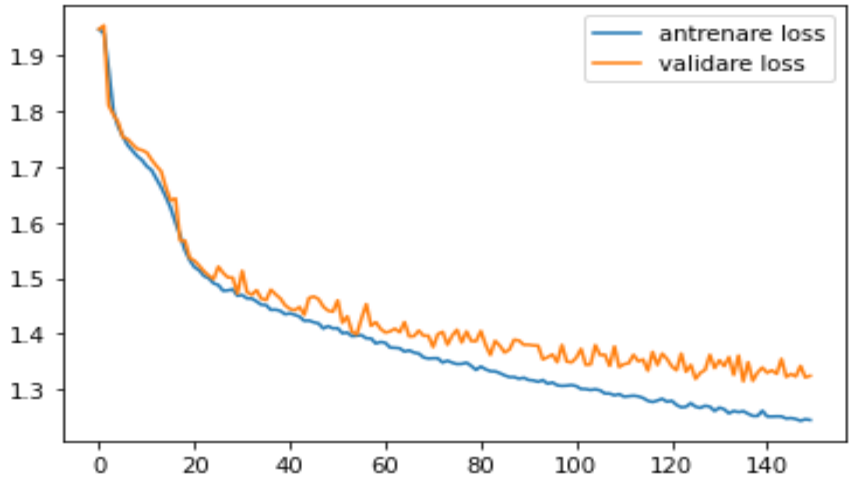
150 epoci : [1.3244644403457642, 0.4936061501502991]
(primul este loss-ul, a doua este curatețea)

Matricea de confuzie

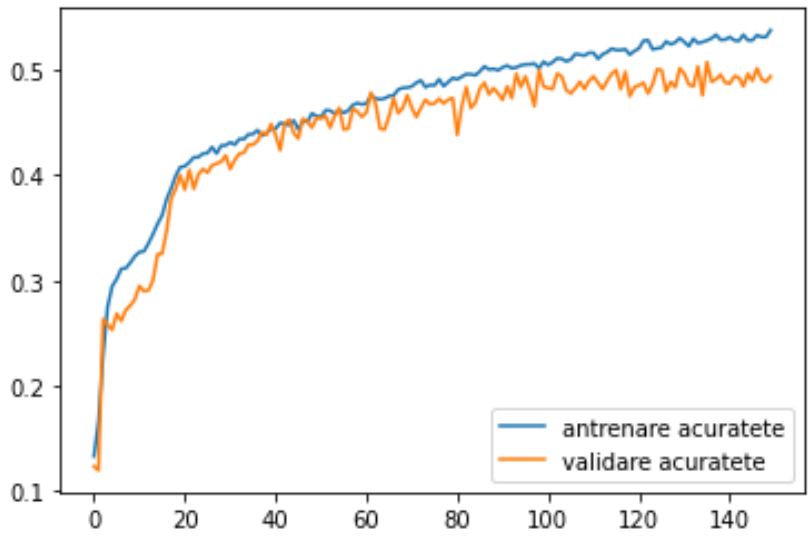
Pentru a putea face această matrice trebuie să se salveze etichete de validare inițiale într-un vector, deoarece acestea își vor schimba forma (fiecare etichetă este transformată în vector) și, totodată, trebuie să existe un vector care să conțină predicțiile etichetelor de validare (se va folosi np.argmax pentru a selecta eticheta cu predicția maximă). După ce sunt obținute aceste date se poate aplica funcția confusion_matrix și ulterior ConfusionMatrixDisplay.



Variația lossului pe datele de antrenare și validare



Variația acurateței pe datele de antrenare și validare



Remarcăm că acuratețea este destul de mică și pentru a o crește ne gândim să creștem numărul de straturi pentru a-i permite modelului să învețe mai bine.

Modelul al doilea (cnn35) are următoarea distribuție:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 16, 16, 32)        896

 conv2d_1 (Conv2D)            (None, 16, 16, 64)        18496

 max_pooling2d (MaxPooling2D  (None, 8, 8, 64)          0
 )

 dropout (Dropout)            (None, 8, 8, 64)          0

 conv2d_2 (Conv2D)            (None, 8, 8, 128)         73856
```

```
 conv2d_3 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 dropout_1 (Dropout)         (None, 4, 4, 128)         0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 32)                65568

 dropout_2 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 7)                 231

=================================================================
Total params: 306,631
Trainable params: 306,631
Non-trainable params: 0
```

106 epoci [loss=1.1275025606155396, acuratețe=0.6138107180595398], iar pe 25% din datele de testare am obținut 0.61505

Față de primul model, acesta are două perechi de două straturi convoluționale urmate de un strat de MaxPooling și de unul de Dropout.

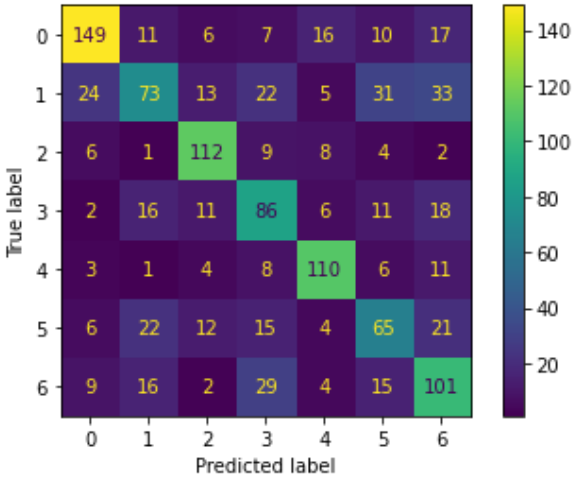
Având mai multe straturi convoluționale, modelul poate învăța mai multe trăsături ale imaginii (în primul strat se învață linii, colțuri, ajungând ca în cel de-al patrulea strat să distingă mai multe elemente, forme).

De asemenea, s-a ales să se folosească nucleu de tip (3,3) pentru straturile convoluționale, deoarece este nevoie ca modelul să învețe trăsături locale mai mici pentru că imaginile se diferențiază prin detalii mici. În ceea ce privește funcția de activare pentru straturile convoluționale s-a ales tot „relu”, iar pentru instanțierea greutăților s-a ales kernel_initializer='he_uniform'.

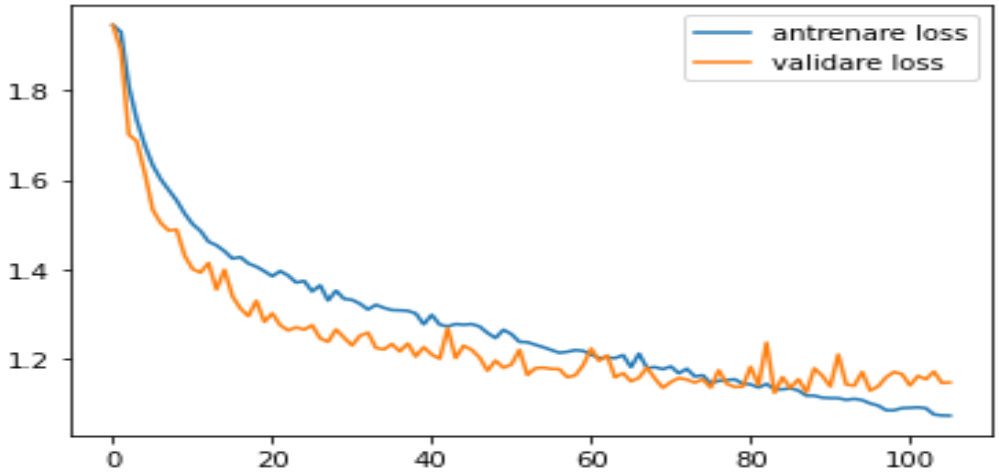
S-a ales să se facă MaxPooling după două straturi convoluționale pentru a lăsa modelul să învețe mai mult, iar după acest strat se apelează la Dropout pentru a preveni supraînvățarea.

În ceea ce privește partea fully connected a rețelei s-a ales să se scadă numărul de filtre din primul strat dens, tot pentru a preveni supraînvățarea, iar funcția de activare s-a schimbat în „tanh” care converge mai lent decât „relu”. Ultimul strat dens conține 7 filtre corespunzătoare celor 7 clase, iar prin funcția de activare „softmax” rezultă un vector de 7 probabilități (suma lor fiind 1).

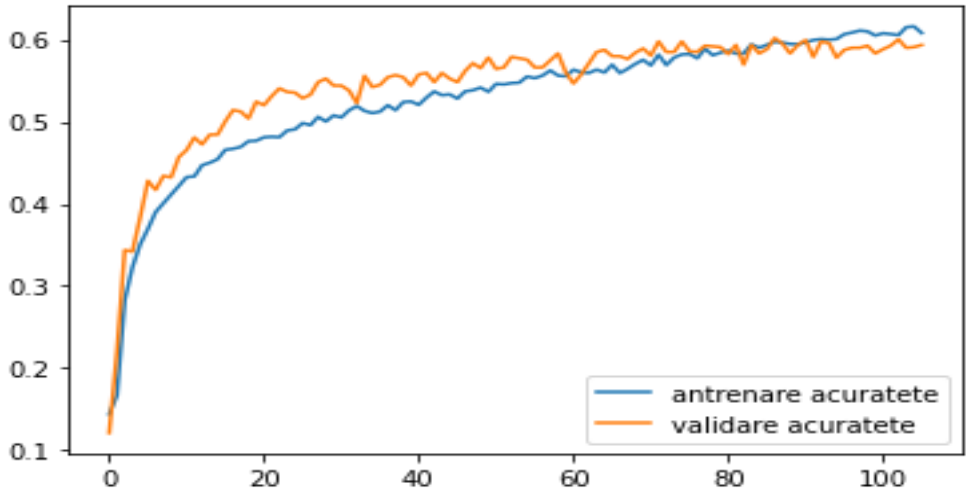
Matricea de confuzie



Variația lossului pe datele de antrenare și validare



Variația acurateței pe datele de antrenare și validare



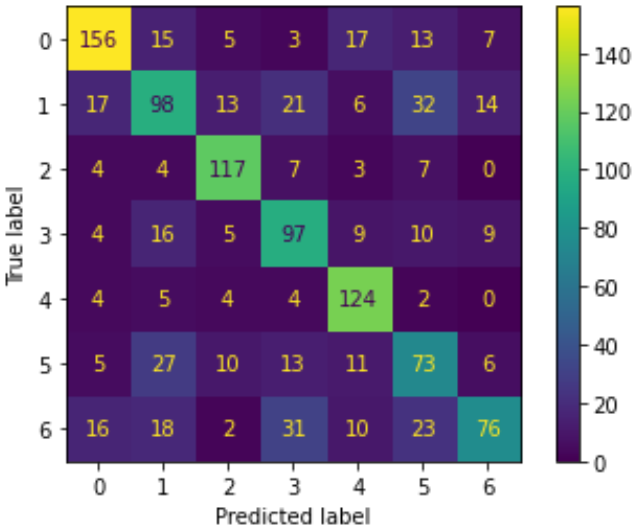
Gândindu-ne cum să îmbunătățim modelul de mai sus alegem să **nu mai aducem pixelii în intervalul [0,1] (nu se mai împart datele de antrenare, validare și testare la 255)**, creștem numărul de filtre de la ultimul strat convoluțional de la 128 la 256 pentru a putea lăsa modelul să învețe mai bine și creștem și numărul de unități ale primului strat dens la 64, pentru că avem mai multe informații de procesat. Se mai adaugă un strat de Dropout pentru a reduce supraînvățarea.

Modelul al treilea (cnn52) are următoarea distribuție (acest model a fost ales pentru submisia finală):

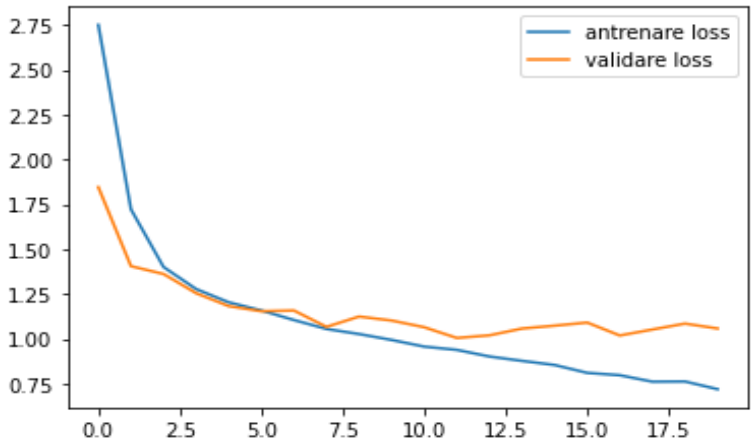
```
Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 16, 16, 32)        896

 conv2d_5 (Conv2D)           (None, 16, 16, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 dropout_3 (Dropout)         (None, 8, 8, 64)          0

 conv2d_6 (Conv2D)           (None, 8, 8, 128)         73856

 dropout_4 (Dropout)         (None, 8, 8, 128)         0

 conv2d_7 (Conv2D)           (None, 8, 8, 256)         295168

 max_pooling2d_3 (MaxPooling  (None, 4, 4, 256)        0
 2D)

 dropout_5 (Dropout)         (None, 4, 4, 256)         0

 flatten_1 (Flatten)         (None, 4096)              0

 dense_2 (Dense)             (None, 64)                262208

 dropout_6 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 7)                 455

=================================================================
Total params: 651,079
Trainable params: 651,079
Non-trainable params: 0
```

În 20 de epoci:[loss=1.0002490282058716,acuratete= 0.6342710852622986], iar acuratețea pe 25% din datele de test = 0.64062, iar pe 75% acuratețea este 0.62978.

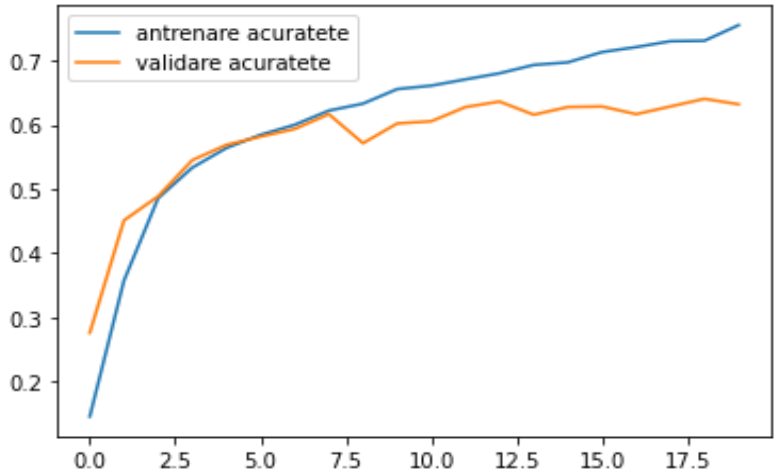
Matricea de confuzie



Variația lossului pe datele de antrenare și validare

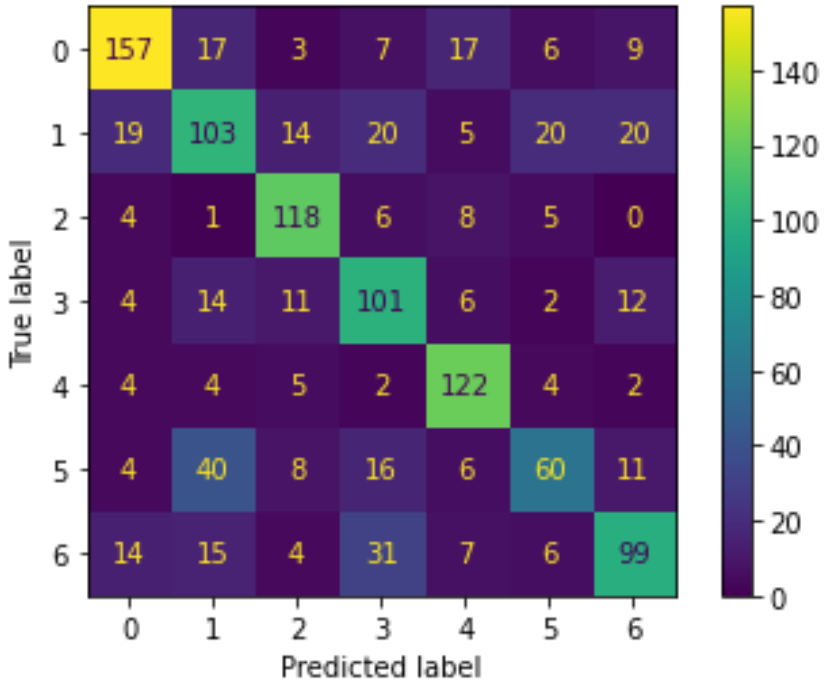


Variația acurateței pe datele de antrenare și validare

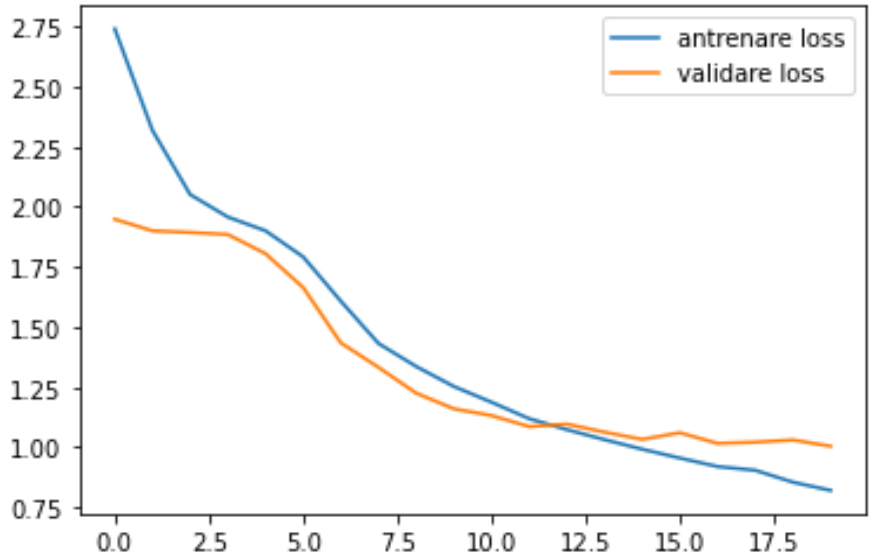


Al patrulea model (cnn55) are schimbat față de al treilea model doar numărul de epoci în care este rulat, anume 19 (acesta este al doilea model trimis spre validarea finală)

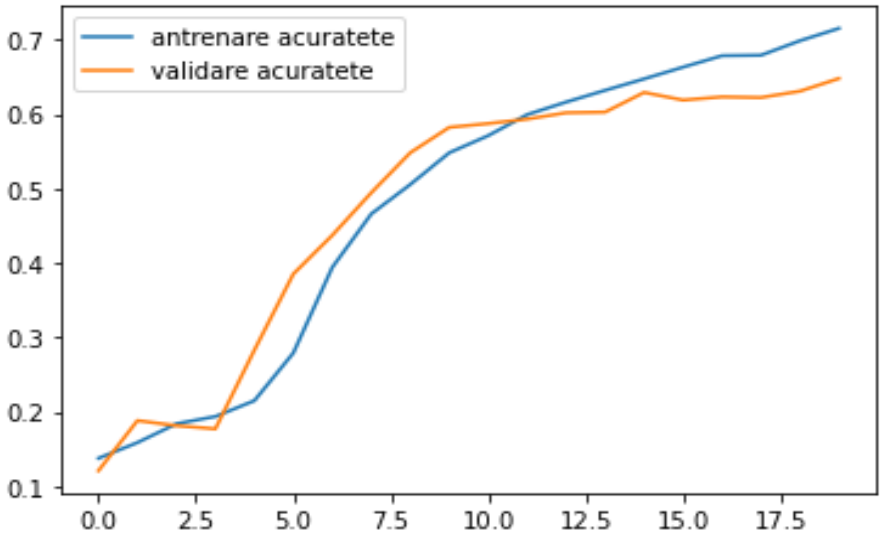
Matricea de confuzie



Variația lossului pe datele de antrenare și validare



Variația acurateței pe datele de antrenare și validare



În 19 epoci: [loss = 0.9927256107330322, acuratete = 0.6496163606643677], iar acuratețea pe 25% din datele de test = 0.63068, iar pe 75% acurateț̦e este 0.63026.

Astfel, se poate observa faptul că se obțin rezultate mai bune dacă nu se aduc pixelii în intervalul [0,1], dacă sunt mai multe straturi convoluționale care conțin un număr de filtre care crește gradual (multiplu de 2), dacă se aplică mai multe straturi de Dropout.

Se poate observa o îmbunătățire foarte bună a acurateței cu acest tip de clasificator, CNN, care a reușit să o ridice de la 0.57528 la 0.64062 pe 25% din datele de test.

6. Concluzii

Clasificator	Acurateț̦e validare	Acurateț̦e test 25%	Acurateț̦e test 75%
Naive Bayes	0.36231884057971014	0.34659	0.35839
KNN	0.4441602728047741	0.42329	0.44964
SVM	0.556692242114237	0.57528	0.55602
CNN	0.6496163606643677	0.63068	0.63026

Astfel, cel mai bun clasificator este CNN.