Realistic Image Classification

-documentatie-

Daria Maria Pirvulescu

June 2024

1 Analiza si prelucrarea datelor

Am inceput acest proiect prin analiza datelor, astfel am implementat o clasa (*CustomImageDataset*) pentru a citi datele de antrenare si validare (idee inspirata din Laboratorul 6 de Machine Learning). Datele contineau atat imagini alb-negru, cat si imagini color, de aceea pentru a asigura consistenta inputului, imaginile alb-negru sunt convertite in imagini cu trei canale. De asemenea, pentru a mentine consistenta si compatibilitatea inputului am eliminat profilul ICC al imaginilor.

Analiza clasei *CustomImageDataset*:

Clasa primeste folderul cu imaginile si cel cu etichete lor si, optional, o functie de transformare pentru normalizarea imaginilor. Am ales sa normalizez imaginile prin folosirea :

```
transforms.Normalize((0.4912, 0.4820, 0.4464), (0.2472, 0.2436, 0.2617))
```

Această transformare normalizeaza imaginile prin scaderea mediei si impartirea la deviatia standard pentru fiecare canal de culoare.

- (0.4912, 0.4820, 0.4464): Acestea sunt valorile medii pentru fiecare canal (R, G, B) al imaginilor.
- (0.2472, 0.2436, 0.2617): Acestea sunt valorile deviatiei standard pentru fiecare canal (R, G, B) al imaginilor.

Dupa citirea datelor din fisierele aferente se convertesc valorile pixelilor in tipul $float$ si se normalizeaza la intervalul $[0, 1]$ prin impartirea la 255.0.

```
  imagine = imagine.float() / 255.0
```

Asemanator clasei *CustomImageDataset*, am implementat o alta clasa *CustomTrainClass* care imi prelucreaza datele de test intr-un mod asemanator.

2 Modele testate

2.1 Retele neuronale convolutionale

2.1.1 Descriere

Prin operatia de convolutie se intelege o operatie matematica ce genereaza un set de greutati, formand o reprezentare a anumitor detalii ale imaginii. Setul de greutati se mai numeste si filtru si este mai mic decat intreaga imagine de intrare, in consecinta acoperind doar o portiune a acesteia. Valorile din filtru sunt multiplicate cu valorile corespunzatoare din imagine(adica cele doua matrici se inmultesc intre ele element cu element). Filtrul se va muta in mod repetata, pana cand intreaga imagine este acoperita si prelucrata.

De fiecare data cand acest filtru este mutat, se va crea un pixel in imaginea de iesire. Pixelul este suma elementelor din matricea rezultata dupa acea inmultire despre care am vorbit anterior. Aceasta operatie de convolutie permite extragerea de caracteristici din imagine, cum ar fi margini, colturi sau alte detalii relevante ce ajuta la clasificarea lor.

Un avantaj major al operatiei de convolutie este minimizarea numarului de parametri si a complexitatii modelului, datorita distribuirii greutatilor (aceleasi filtre sunt utilizate pentru toate bucatile din imagine). Retelele neuronale convolutionale extrag pe rand parametrii din regiuni locale ale imaginii, in loc sa prelucreze imaginea cu totul. De asemenea, operatia de convolutia captureaza invarianta la translatie, ceea ce inseamna ca caracteristicile sau detaliile recunoscute sunt independente de pozitia lor in imagine. Aceasta face convolutia extrem de eficienta pentru prelucrarea imaginilor de dimensiuni mari si pentru aplicatii in care pozitia exacta a caracteristicilor delimitatorii nu este fixa.

Mai mult de atat, in retelele clasice odata cu avansarea in retea, pozitiile precise ale unor trsasaturi devin mai putin importante, in anumite cazuri cauzand dificultati procesului de antrenare. Din acest motiv, majoritatea arhitecturilor retelelor convolutionale utilizează operatii de subesantionare, adica mai concret: straturilor de pooling. Acestea parcurg diversele vecinatati din stratul anterior si inlocuiesc valorile din fiecare vecinatate cu una singura, iar cel mai intalnit astfel de strat este $max - pooling$, care presupune pastrarea valorii maxime. Mai exista si $avg - pooling$, care pastreaza valorea medie.

O alta componenta importanta o reprezinta *padding*-ul. Uneori dimensiunea imaginilor si a filtrelor nu sunt compatibile, aici intra in prim-plan notiune de *padding*. Mai concret, imaginea o sa fie bordata de unul sau mai multe linii de pixeli 0. In general, acest padding este corelat cu marimea filtrului aplicat pe imagine astfel:

- kernel=1x1 $\rightarrow$ padding de 0 (nu este necesar)
- kernle=3x3 $\rightarrow$ padding de 1
- kernel=5x5 $\rightarrow$ padding de 2
- s.a.m.d

După cum se observă, padding-ul este de obicei $\left\lfloor \frac{\text{kernel\_size}}{2} \right\rfloor$.

O alta componenta importanta o reprezinta *stride*-ul, care este distanta, sau numarul de pixeli, pe care kernelul ii deplaseaza peste matricea de intrare pentru a face acea operatie de inmultire a filtrului cu o portiune din imaginea originala. Un stride mai mare genereaza o iesire mai mica, iar in modelul meu *stride*-ul ramane constant de 1.

Dupa fiecare operatie de convolutie, o retea neuronala convolutiva (CNN) aplica o transformare Rectified Linear Unit (ReLU) introducand non-linearitate in model.

Valorile pixelilor din imaginea de intrare originala nu sunt conectate direct la stratul de iesire in straturile partial conectate, asadar pentru a putea clasifica imaginile in categoriile prezise de model ne trebuie un $fully - connected - layer$. In stratul complet conectat, fiecare nod din stratul de iesire se conecteaza direct la un nod din stratul anterior.

Acest strat indeplineste clasificarea pe baza caracteristicilor extrase prin straturile anterioare si filtrele lor diferite. In timp ce straturile de convolutie si pooling tind sa foloseasca functii ReLU, straturile complet conectate utilizeaza de obicei o functie de activare $softmax$ pentru a clasifica intrarile in mod corespunzator, producand o probabilitate intre 0 si 1.

Clasa *ReteaConvolutionala* defineste arhitectura retelei neuronale convolutionale astfel: mai multe $convolutional - layers$ (se aplica filtre care extrag caracteristici esentiale din imagini, cum ar fi margini, texturi si alte detalii relevante), urmate de $batch - normalization$ (sunt aplicate după fiecare strat pentru a stabiliza antrenarea retelei, asigurand ca iesirile fiecarui strat au distributii similare), urmate de functia de activare *Relu* (este utilizata pentru a introduce non-linearitati in retea, permitand modelului sa invete relatii mai complexe intre date). Metoda $forward$ este folosita pentru a defini trecerea imaginilor prin retea:

- Dupa repetarea acestui triplet (strat convolutional, batch-normalization si ReLu) de numarul dorit de ori, se aplica stratul de $Adaptive - Average - Pooling$ pentru reduce dimensiunile, in acest caz, pooling-ul este efectuat pentru a obtine o dimensiune fixa de 1x1.
- Ieșirea este aplatizata (flattened) intr-un vector de dimensiune 1D.
- Aplicare unui strat linear imi va clasifica imaginile in cele 3 clase posibile

```
self.linear = nn.Linear(128, 3)
```

Functia de **train** explicata:

Primeste ca parametrii *optimizer*-ul si *loss function*-ul care in majoritatea cazurilor vor fi *SGD* (calculează valoarea gradientului pentru fiecare exemplu din setul de date de antrenare si actualizeaza *weight*-urile pentru a creste acuratetea) respectiv *CrossEntropyLoss* (masoara piererea dintre etichetele prezise de model si etichetele reale). Dupa ce se optin predictiile modelului, cu ajutorul aceste functii se va calcula *loss*-ul, apoi se aplica *optimizer*-ul care imbunatateste reteau, optional se mai poate calcula si acuratetea si pierderea pentru fiecare epoca.

Functia **validate** explicata:

Modelul este setata in modul de evaluare si se fac iar predictii pe date de validare, de data aceasta, fara a mai actualiza gradientul pentru fiecare imagine. Acesata functie este folosita pentru a evita *overfitting*-ul si pentru a putea ajusta hiperparametrii pe parcursul epocilor.

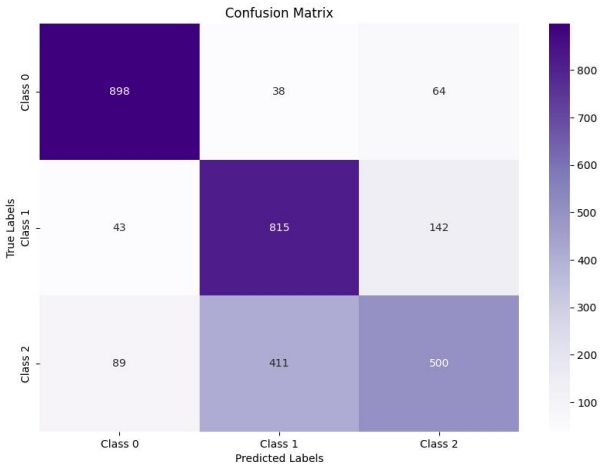
Urmeaza functia pentru a calcula predictiile pe imaginile de test, ce este asemanatoare functiei de validare, insa de data acesata etichetele reale ale imaginilor nu sunt cunoscute. Rezultatul este salvat intr-un fisier *csv*.

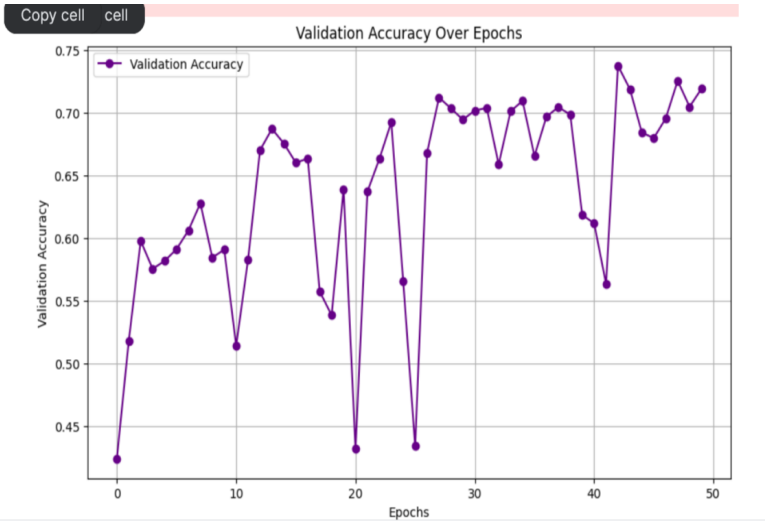
Pentru a arata mai bine evolutia modelului, am decis sa ilustrez matricea de confuzie si evolutia acuratetii pe datele de validare folosind libraria *matplotlib.pyplot*.

Am folosit o abordare de antrenare a modelui bazata pe epoci (in general intre 50 si 100), ce ajuta la imbunatatirea performantei prin ajustarea ponderilor.

2.1.2 Hiperparametrii testati

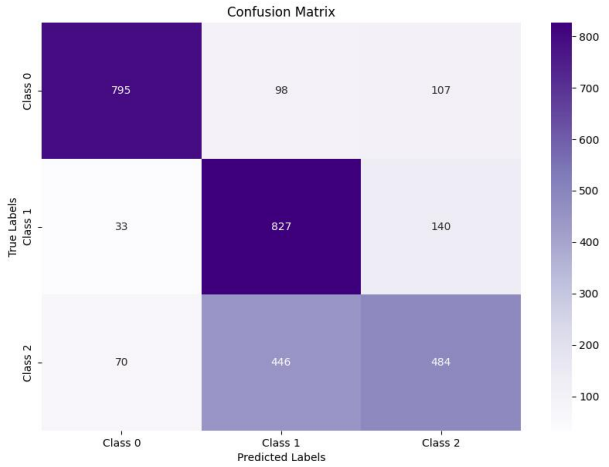
1.) Numarul de straturi = 4 Kernel= 3x3 Numar filter pentru fiecare strat = 64, 64, 128, 128
Batch-size = 128 Learning rate = 0.01 Momentum = 0.9 Optimizer = SGD Numar de epoci = 50

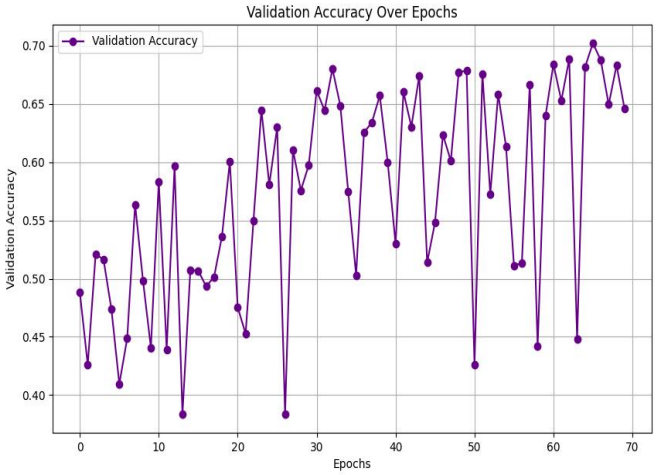




ACURATETE MAXIMA=0.7376666666666667

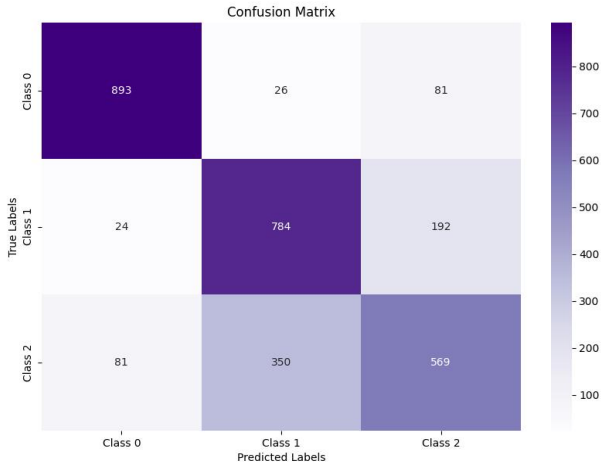
2.) Numarul de straturi = 5 Kernel= 3x3 Numar filter pentru fiecare strat = 64, 64, 128, 128, 128 Batch-size = 256 Learning rate = 0.005 Momentum = 0.9 Optimizer = SGD Numar de epoci = 70

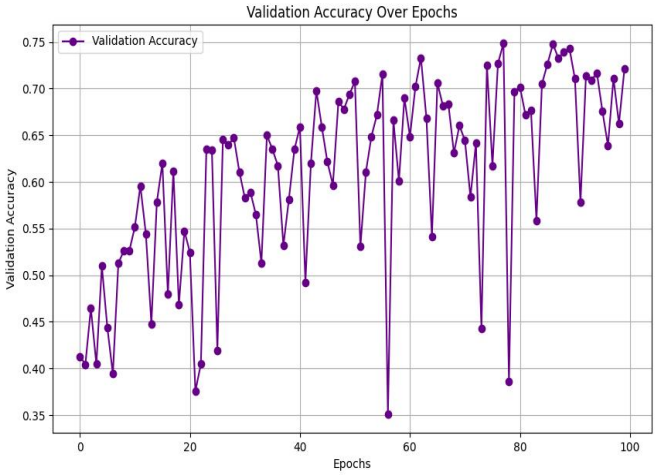




ACURATETE MAXIMA=0.702

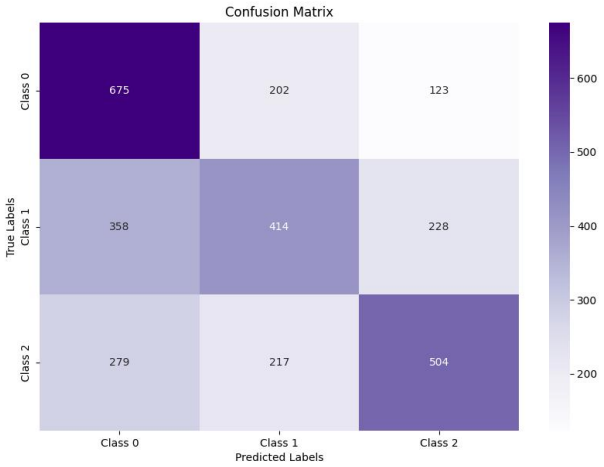
3.) Numarul de straturi = 6 Kernel= 3x3 Numar filter pentru fiecare strat = 64, 64, 128, 128, 128,128 Batch-size = 256 Learning rate = 0.1 Momentum = 0.9 Optimizer = SGD Numar de epoci = 100

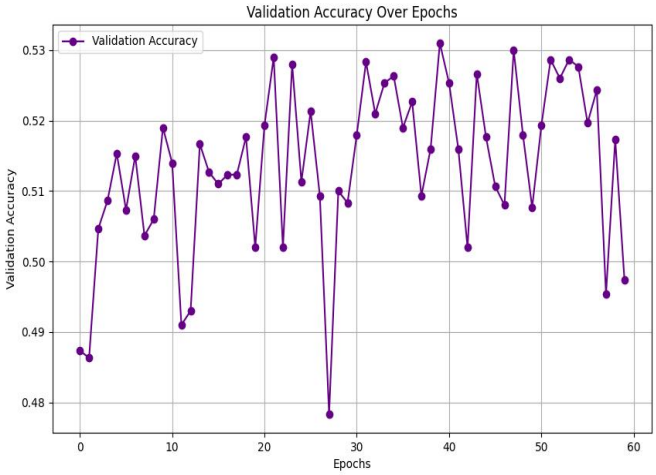




ACURATETE MAXIMA=0.7486666666666667

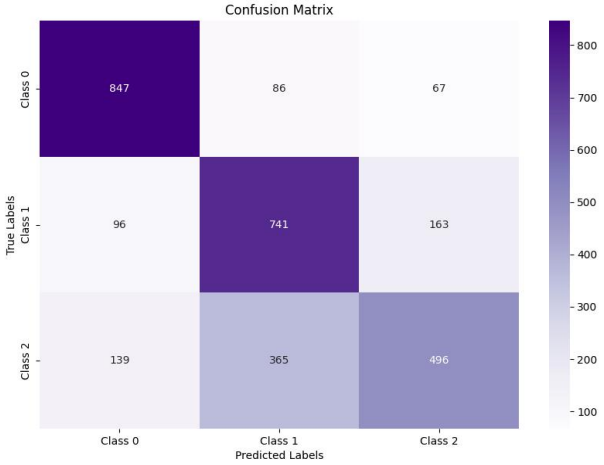
4.) Numarul de straturi = 6 Kernel= 1x1 (padding=0) Numar filter pentru fiecare strat = 64, 64, 128, 128, 128,128 Batch-size = 256 Learning rate = 0.01 Momentum = 0.9 Optimizer = SGD Numar de epoci = 60

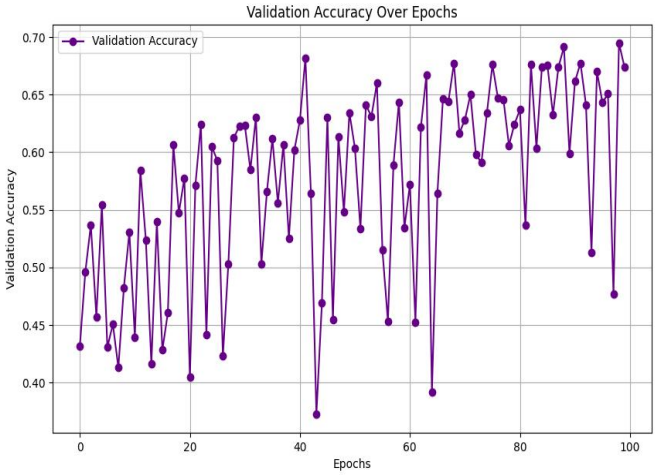




ACURATETE MAXIMA=0.531

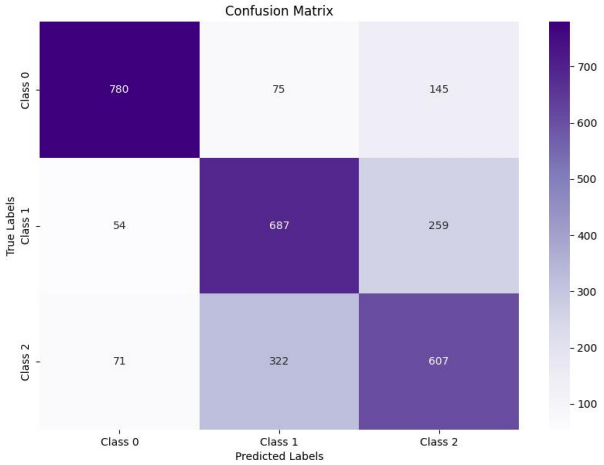
4.) Numarul de straturi = 6 Kernel= 5x5 (padding=2) Numar filter pentru fiecare strat = 64, 64, 128, 128, 128,128 Batch-size = 256 Learning rate = 0.01 Momentum = 0.9 Optimizer = SGD Numar de epoci = 100

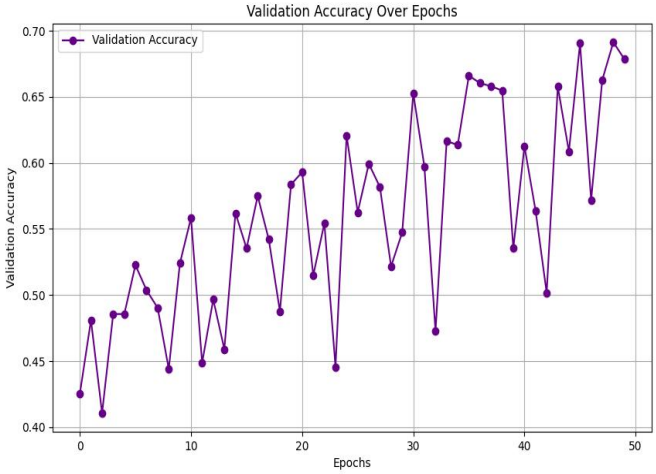




ACURATETE MAXIMA=0.689

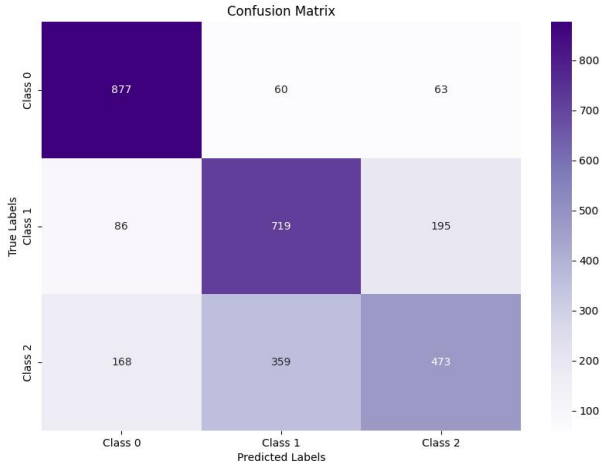
5.) Numarul de straturi = 7 Kernel= 3x3 (padding=1) Numar filter pentru fiecare strat = 64, 64, 128, 128, 128,128, 128 Batch-size = 256 Learning rate = 0.01 Momentum = 0.9 Optimizer = ADAM Numar de epoci = 50

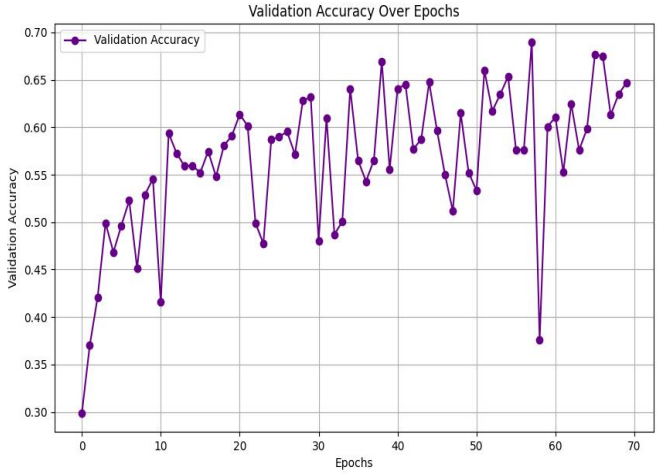




ACURATETE MAXIMA=0.6913333333333334

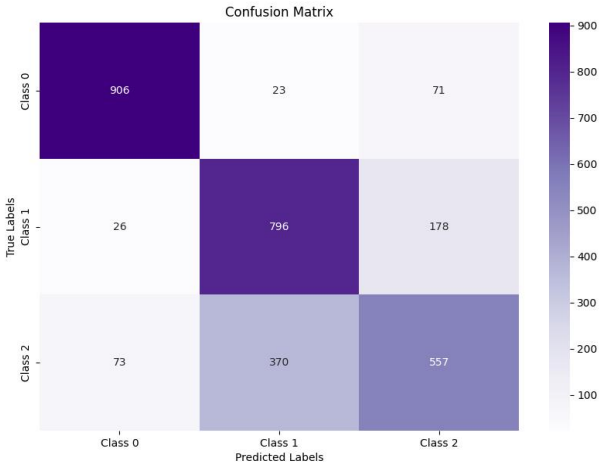
6.) Numarul de straturi = 8 Kernel= 3x3 (padding=1) Numar filter pentru fiecare strat = 64, 64, 128, 128, 128,128, 128 Batch-size = 256 Learning rate = 0.05 Momentum = 0.9 Optimizer = SGD Numar de epoci = 70

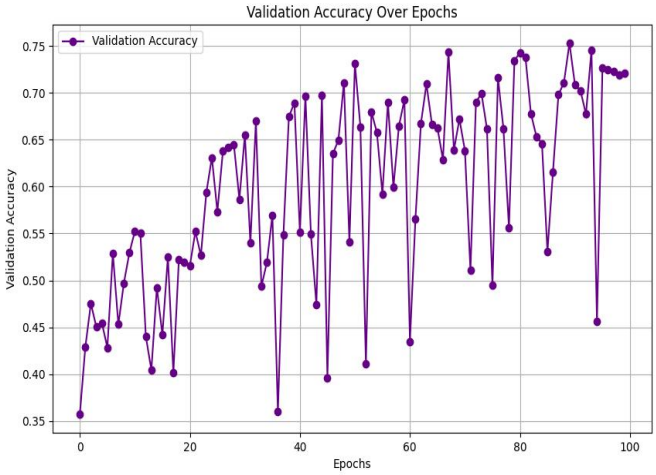




ACURATETE MAXIMA=0.6896666666666667

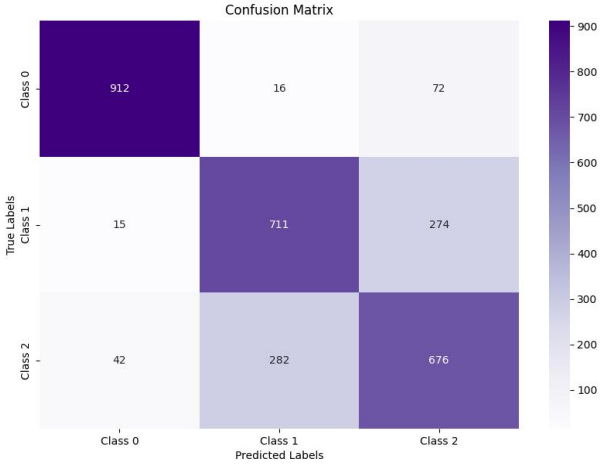
7.) Numarul de straturi = 7 Kernel= 3x3 5x5 (padding=1 si 2) Numar filter pentru fiecare strat = 32,32,64,64,128,128,256 Batch-size = 128 Learning rate = 0.1 Momentum = 0.9 Optimizer = SGD Numar de epoci = 100

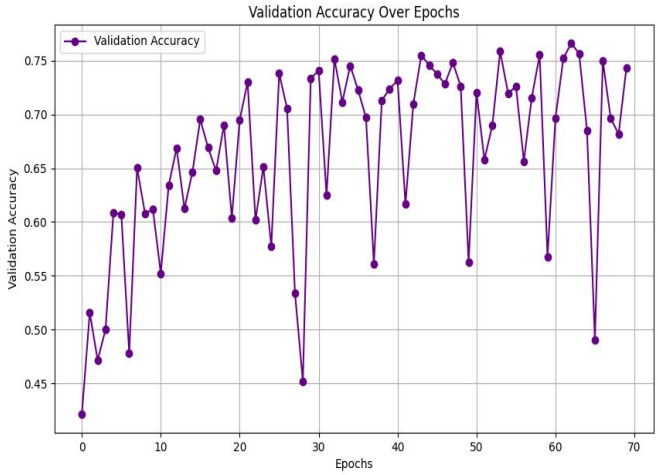




ACURATETE MAXIMA=0.753

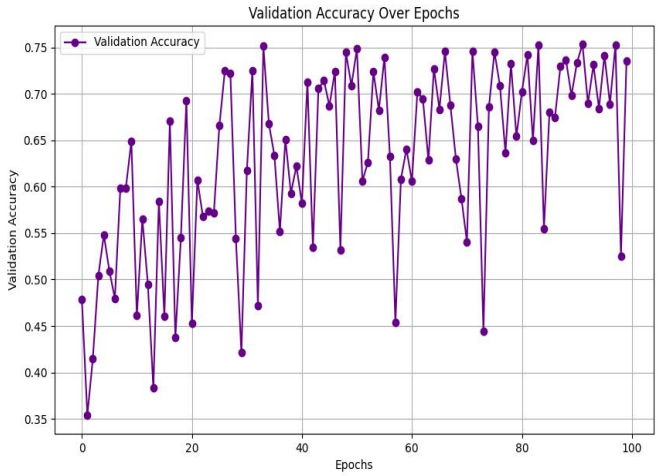
8.) Numarul de straturi = 6 Kernel= 3x3 5x5 (padding=1 si 2) Numar filter pentru fiecare strat = 32,64,64,128,128,256 Batch-size = 128 Learning rate = 0.01 Momentum = 0.9 Optimizer = SGD Numar de epoci = 70





ACURATETE MAXIMA= 0.7663333333333333

9.) Numarul de straturi = 6 Kernel= 3x3 5x5 (padding=1 si 2) Numar filter pentru fiecare strat = 32,64,64,128,128,256 Batch-size = 64 Learning rate = 0.05 Momentum = 0.9 Optimizer = SGD Numar de epoci = 100



ACURATETE MAXIMA= 0.752

2.2 MLPClassifier

2.2.1 Descriere

La baza, un clasificator MLP (clasificator supervizat) consta din mai multe straturi de neuroni artificiali interconectati, cunoscuti si sub denumirea de perceptroni. Fiecare conexiune are o greutate asociată care este ajustata în timpul procesului de antrenare. Acest clasificator include in mod obisnuit un strat de intrare, unul sau mai multe straturi ascunse si un strat de iesire. Fiecare neuron din retea primeste semnale de intrare, aplica o functie de activare non-liniara si transmite iesirea transformata catre stratul urmator. Acest proces continua pana la stratul final, care produce iesirea de clasificare.

A doua retea testata este o $Multi-layer\ Perceptron-classifier$ implementat cu ajutorul bibliotecii *sklearn*. Pentru citirea si prelucrarea datelor se folosesc aceleasi doua clase de la modelul incercat anterior. Se defineneste **MLPClassifier**-ul cu toti parametrii sai, apoi se aplica functiile pentru train, validare si test pe imaginile prelucrate anterior.

```
mlp_classifier_model = MLPClassifier(hidden_layer_sizes=(150),
                       activation='relu',solver='adam',alpha=0.01,batch_size=64,learning_rate='
    adaptive',
                             learning_rate_init=0.05,power_t=0.5,max_iter=200,shuffle=True,
                             random_state=None, tol=0.0001, momentum=0.9, early_stopping=False,
                             validation_fraction=0.1, n_iter_no_change=10)
```

Explicare parametrii

1. **hidden_layer_sizes**:

   Acest parametru specifica arhitectura retelei neuronale artificiale (MLP) prin specificarea numarului de neuroni in fiecare strat ascuns.

2. **activation**:

   Functia de activare folosita pentru straturile ascunse, deoarce vrem sa introducem *neliniariatate*

3. **solver**:

   Algoritmul folosit pentru optimizarea ponderilor retelei in procesul de *training*. In exemplul dat, se folosește algoritmul 'adam', dar mai sunt si 'lbfgs' si 'sgd'.

4. **alpha**:

   Parametrul de regularizare L2 (termenul de penalizare a ponderilor), care ajuta la prevenirea overfitting-ului prin adaugarea unei penalizari proportionale cu norma L2 a vectorului de ponderi.

5. **batch_size**:

   Numarul de exemple de antrenare folosite intr-o iteratie a algoritmului.

6. **learning_rate** și **learning_rate_init**:

   `learning_rate` specifică modul in care rata de invatare poate sa varieze ('adaptive' inseamna ca rata de invatare se adapteaza pe masura ce antrenarea progreseaza). `learning_rate_init` specifică rata initiala de invatare.

7. **power_t**:

   Exponentul pentru calcularea ratei de invatare in algoritmul 'sgd' (stochastic gradient descent). In acest caz, este setat la 0.5, ceea ce inseamna ca rata de invatare se reduce in mod liniear pe parcursul invatarii.

8. **max_iter**:

   Numarul maxim de iteratii (epoci) pana la oprirea training-ului retelei. Ajuta in asigurarea ca reteau se va oprii din invatare si nu va merge un numar infinit de ori.

9. **shuffle**:

   Specifica daca setul de date trebuie amestecat la fiecare epoca de antrenare pentru a imbunatati generalizarea modelului.

10. **random_state**:

    Starea initiala a generatorului de numere aleatoare. Daca este setat la None, fiecare rulare a modelului va genera rezultate diferite.

11. **tol**:

    Toleranaa pentru oprirea algoritmului de optimizare. Antrenarea se va opri atunci cand diferenta dintre două iteratii succesive este mai mica sau egala cu această valoare.

12. **momentum**:

    Viteza de miscare a gradientului in directia anterioara (*backwords*). Aceasta ajuta la accelerarea convergentei si la evitarea minimelor locale.

13. **early_stopping**:

    Daca este setat la True, antrenarea va opri automat atunci cand performanta pe setul de validare nu se imbunatateste pentru un numar specificat de iteratii .(`n_iter_no_change`).

14. **validation_fraction**:

    Fractiunea de date de antrenare folosite pentru validarea modelului in timpul antrenarii.

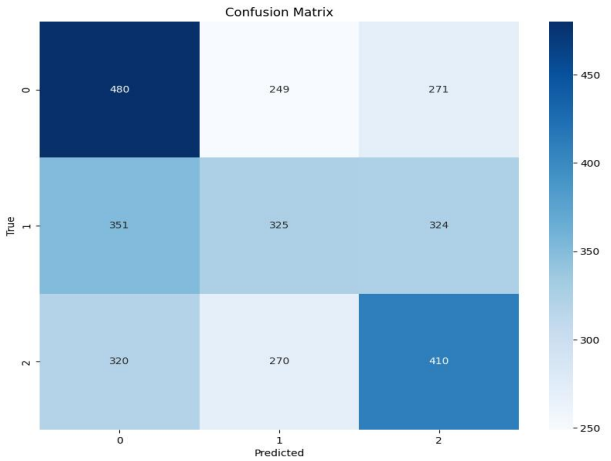
15. **n_iter_no_change**:

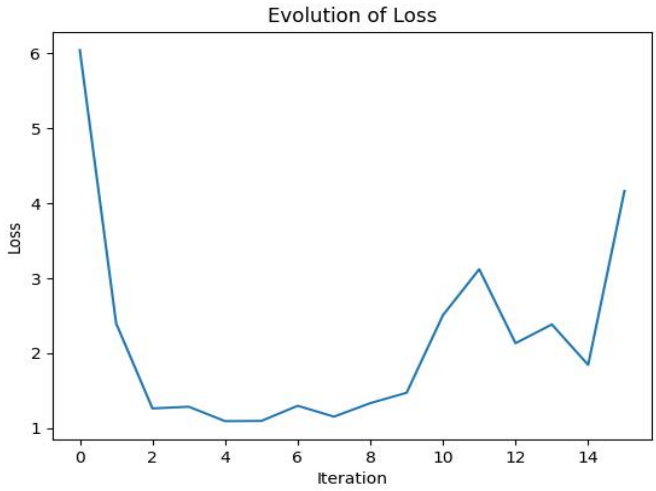
    Numarul maxim de iteratii fara imbunatatiri ale acuratetii pe setul de validare inainte ca antrenarea sa se opreasca (doar daca `early_stopping=True`).

Pentru ilustrarea progresului se foloseste un sistem bazat pe epoci, construindu-se mai apoi cu ajutorul bibliotecii *matplotlib* matricea de confuzie si evolutia *loss*-ului.

2.2.2 Hiperparametrii testati

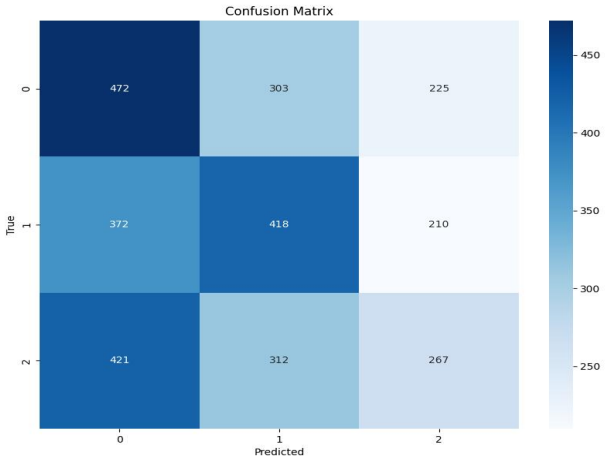
1.) hidden layer sizes=(100,) activation='relu' solver='adam' alpha=0.0001 batch size=128 learning rate='adaptive' learning rate init=0.01 power t=0.5 max iter=300 shuffle=True random state=None tol=0.0001 momentum=0.9 early stopping=False validation fraction=0.1 n iter no change=10

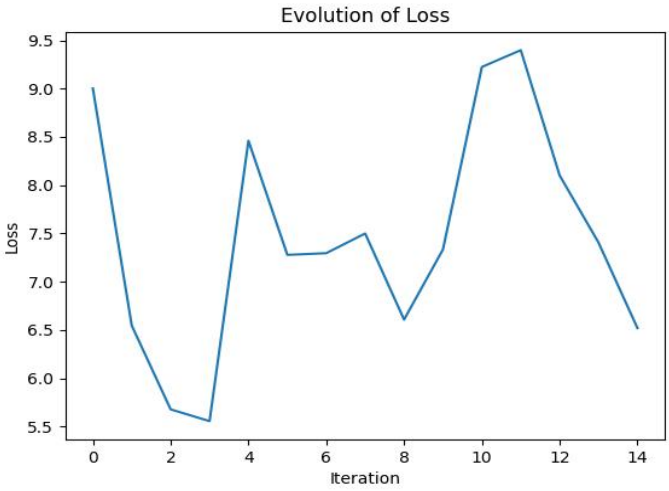




ACURATETE MAXIMA= 0.4327

2.) hidden layer sizes=(100,) activation='relu' solver='adam' alpha=0.0001 batch size=256 learning rate='adaptive' learning rate init=0.05 power t=0.5 max iter=500 shuffle=True random state=None tol=0.0001 momentum=0.9 early stopping=False validation fraction=0.1 n iter no change=10





ACURATETE MAXIMA= 0.39

Am oprit testarea pentru acest model, deoarce rezultatele erau semnificativ de mici si m-am concentrat pe CNN pentru a obtine o acuratete cat mai buna.

In concluzie, se poate observa ca Retelele Neuronale Convolutionale au o acuratete mai buna cand vine vorba de clasificarea acestui set specific de date, diferenta fiind aproape dubla (0.76 fata de 0.4).