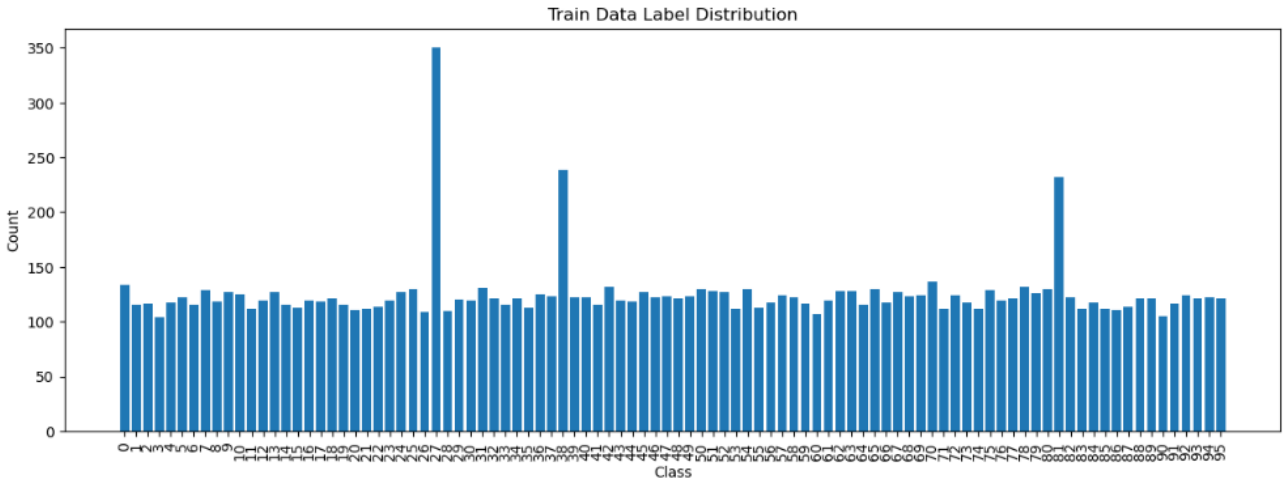
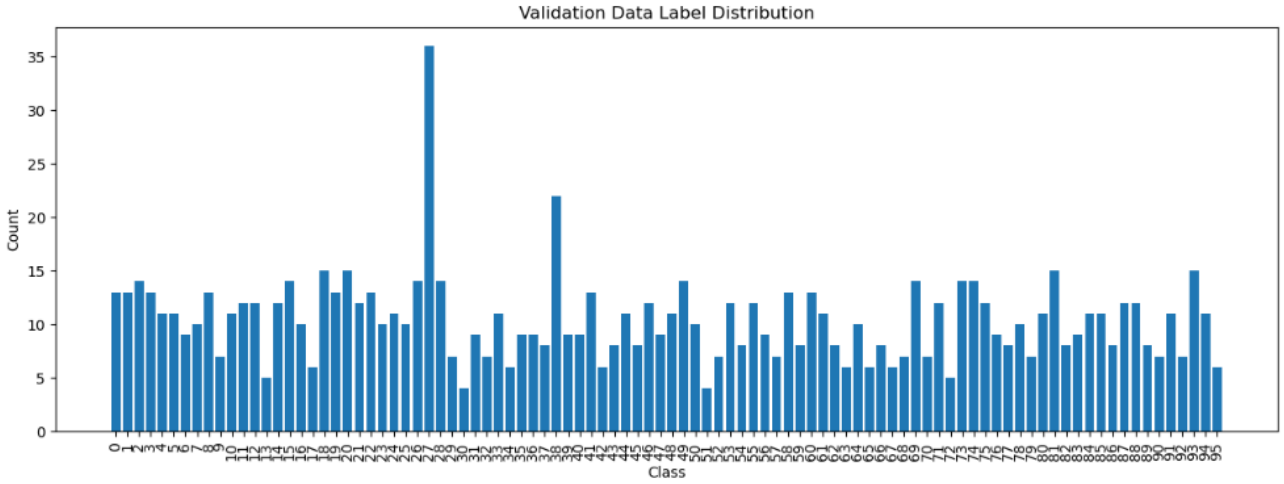
Deep Hallucination Classification

Ionescu Radu-Constantin, grupa 234

- ***Tema Lucrarii***
  - Proiectul consta in clasificarea unor imagini colorate cu design modern, posibil dintr-o colectie de arta digitala, intr-una dintre cele 96 de categorii existente
  - Imaginile sunt date in format PNG si sunt de dimensiune 64 x 64 pixeli
  - Se cere implementarea a doi algoritmi de Machine Learning pentru a rezolva aceasta cerinta, fiecare trecand de un prag minim de performanta
  - Obiectivul este proiectarea unui algoritm cat mai eficient din punct de vedere al acuratetii generale
  - Restrictii:
    - Nu se folosesc modele antrenate anterior
    - Nu se folosesc date suplimentare de input
- ***Datele***
  - Datele de input sunt impartite in 3 categorii:
    - Date de antrenare - folosite pentru antrenarea modelelor
      - Folderul train_images contine 12000 de imagini in format PNG
      - Fisierul train.csv contine etichetele corespunzatoare celor 12000 de imagini
    - Date de validare - folosite pentru verificarea acuratetii modelelor
      - Folderul val_images contine 1000 de imagini in format PNG
      - Fisierul val.csv contine etichetele corespunzatoare celor 1000 de imagini
    - Date de testare - folosite drept test final pentru model, reprezinta cerinta proiectului clasificarea lor cat mai performanta
      - Folderul test_images contine 5000 de imagini in format PNG
  - Datele de output:

- Un fisier CSV cu header-ul Image,Class care contine perechi de forma id_imagine.png - eticheta prezisa de model
- In cadrul acestui proiect, am generat un fisier dedicat outputului (output.csv) pentru a salva informatiile in formatul dorit spre a fi trimise apoi ca submisie.



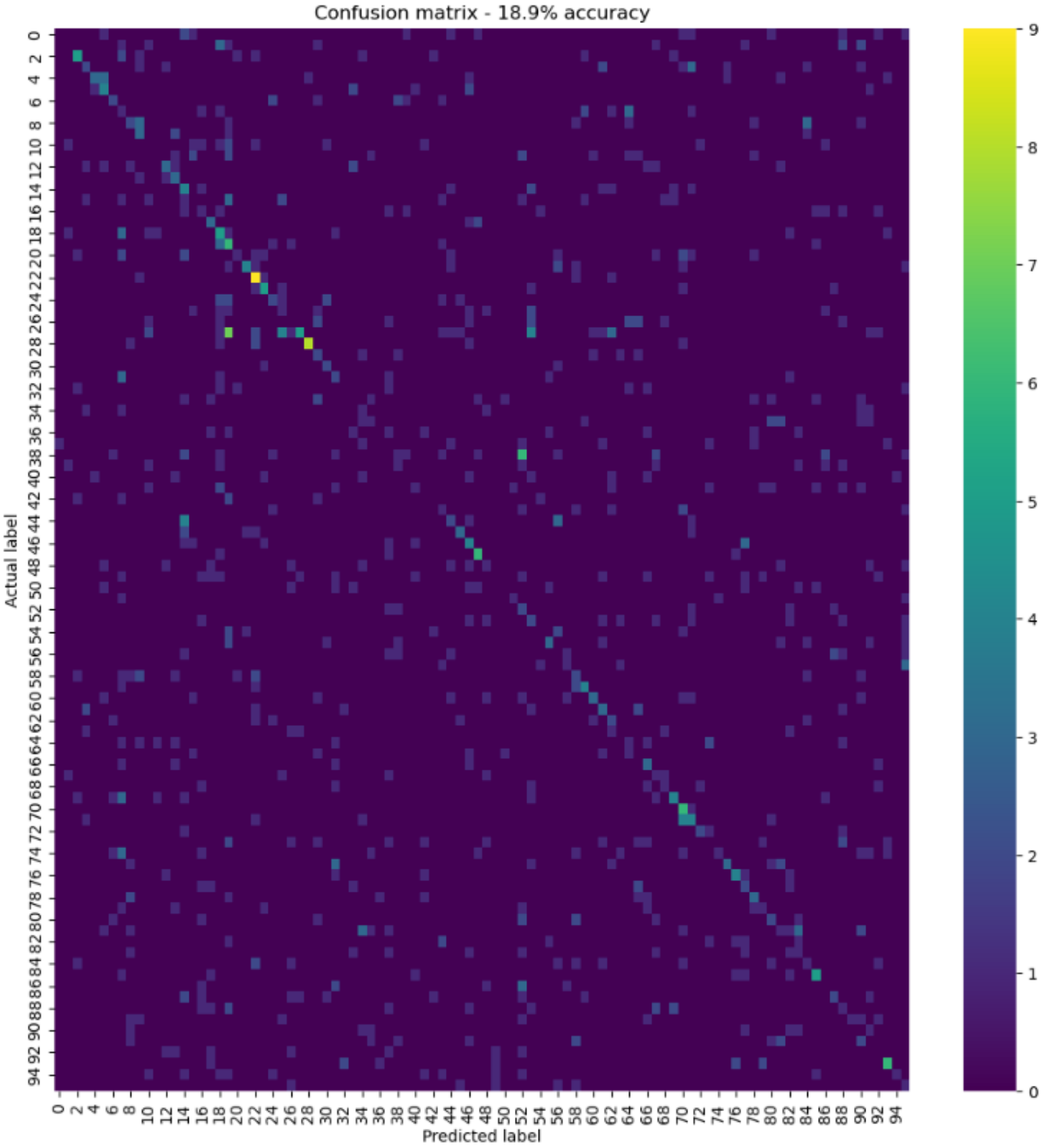


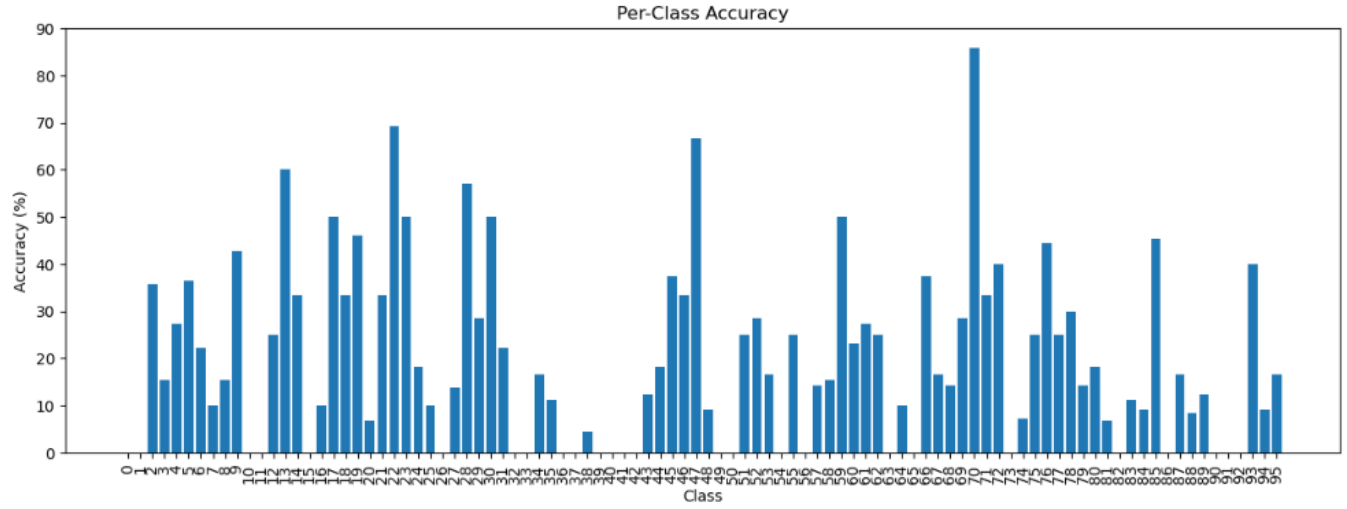
- ***Modelul Naïve Bayes (acuratete de 19-20%)***
  - ***Descriere***:
    - Modelul Naive Bayes este un tip de clasificator probabilistic bazat pe teorema Bayes, folosit adesea in machine learning. Numele de "naive" provine de la presupunerea simplista pe care o face, aceea că fiecare caracteristica de intrare este independenta de celelalte. Este util pentru probleme de clasificare cu mai multe clase.

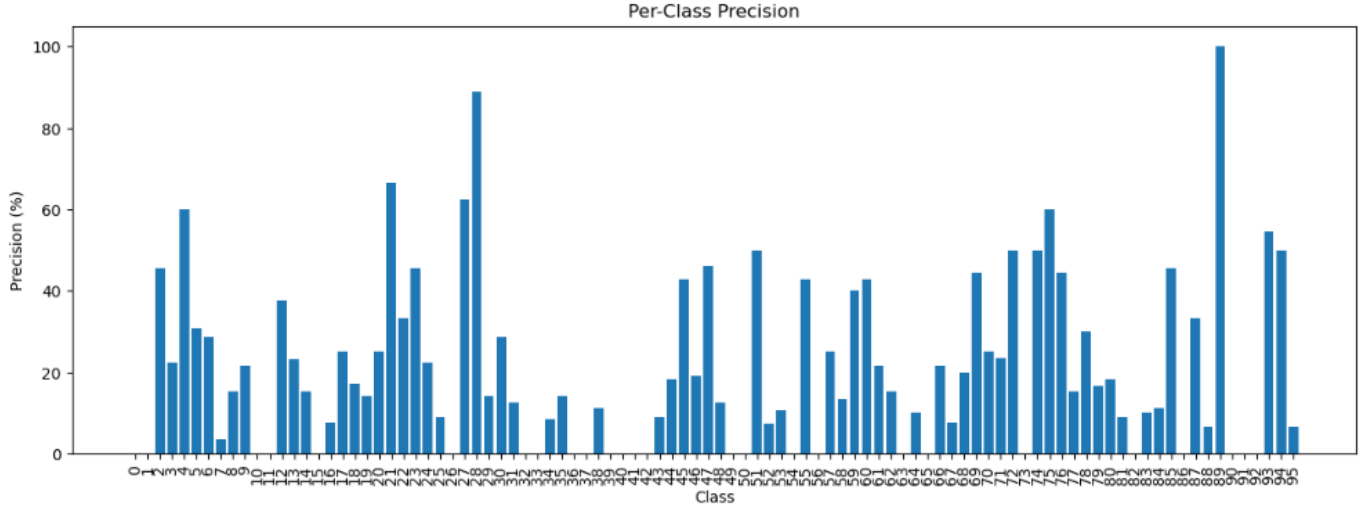
- Unul dintre avantajele modelului Naive Bayes este ca necesita mai putine date pentru a realiza predictii precise, comparativ cu alti algoritmi.
- Totusi, acest model este cu mult mai putin performant daca exist o dependenta puternic sau interactiune complexa intre caracteristici, din cauza presupunerii ca trasaturile sunt independente
- Astfel, cu cat avem mai multe caracterisitici, cu atat e mai probabil ca acestea sa aiba legatura intre ele, deci cu atat modelul devine mai neperformant

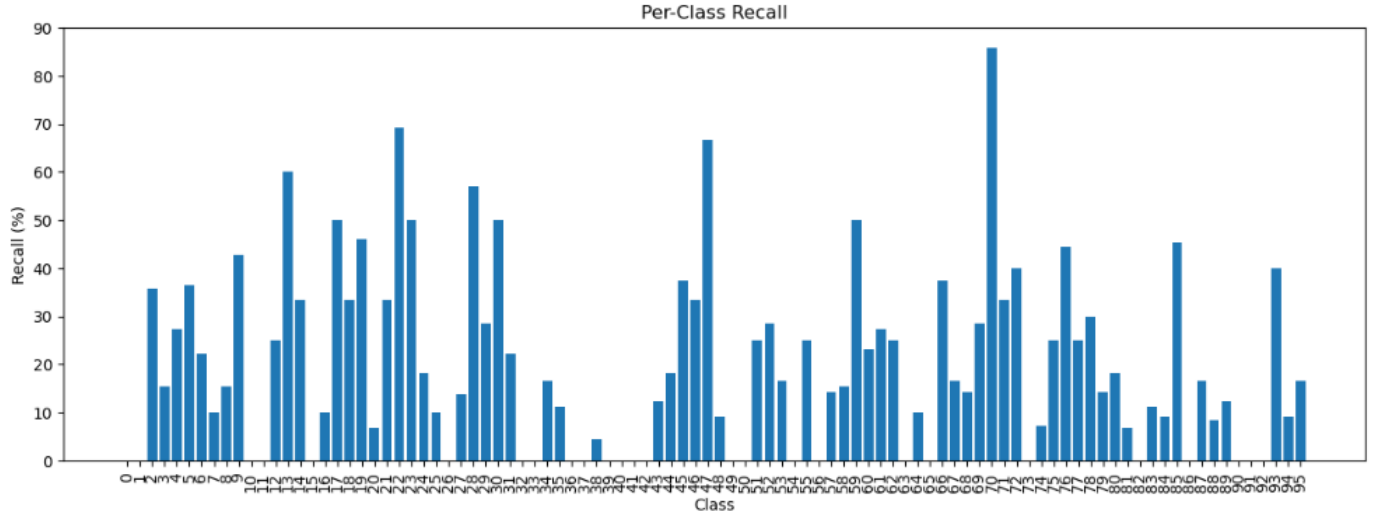
- ***Implementare:***
  - Modelul Naïve Bayes utilizat in acest proiect este cel studiat la laborator:MultinomialNB.
  - In primul rand, s-a realizat citirea datelor, adica transformarea imaginilor intr-un array-uri cvadrimensional N x 64 x 64 x 3 (N imagini cu 64 x 64 rezolutia imaginii, 3 pentru codul RGB, unde N = 1000 pentru imaginile de validare, 5000 pentru cele de test si 12000 pentru cele de antrenare). Apoi, am facut doua dictionare pentru perechi de ‘forma id_imagine.png’ - eticheta. Ulterior, pe baza acestor dictionare construim listele train_images, val_images, test_images si apoi le convertim la numpy arrays pentru a avea datele intr-un format potrivit pentru prelucrare.
  - Am definit functia *values_to_bins* pentru a impartii valorile continue ale pixelilor in containere (“bins”) discrete.
  - In continuare incercam diferite dimensiuni pentru containere: 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27 si testam pentru fiecare dintre ele acuratetea pe datele de validare si salvam acuratetea intr-un dictionar cu perechi de forma numar_containere – acuratete
  - Apoi folosim numarul optim de containere (adica valoarea maxima din dictionar) pentru a face predictii in legatura cu datele de testare
  - La final, salvam datele in fisierul de output si reprezentam vizual, prin grafice generate cu ajutorul bibliotecii *matplotlib*, matricea de confuzie si scorurile de acuratete, precizie si recall pentru fiecare clasa de imagini
- ***Rezultate:***

- In ceea ce priveste acuratetea, scopul proiectului practic, scorul este unul mic, de aproximativ 19-20%. O posibila explicatie este aceea ca imaginile folosite sunt complexe, acestea fiind opere de arta abstracta ce includ multe caracteristici pentru cele 96 de clase (o presupunere strict personala este ca imaginile reprezinta lucruri complexe, cum ar fi roboti, orase futuristice, oameni/fiinte umanoide, etc) si, dupa cum am precizat in descrierea modelului, cu cat creste numarul de caracterisitici cu atat scade acuratetea clasificatorului.


Confusion matrix - 18.9% accuracy
Actual label
Predicted label
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94
0 1 2 3 4 5 6 7 8 9


Per-Class Accuracy
Accuracy (%)
Class


Per-Class Precision
Precision (%)
Class


Per-Class Recall
Recall (%)
Class

- ***Modelul Convoluted Neural Network (acuratete de 89-90%)***
  - ***Descriere***: O Retea Neuronala consta in proiectarea unei arhitecturi de straturi, placi cu perceptroni (neuroni) spre a transmite semnale intre ei similar cu neuronii din creier. Reteaua Neuronala Convolutionala (CNN) este o subcategorie de Retea Neuronala caracterizata prin existenta straturilor convolutionale, straturi care extrag anumite caracterisitici din datele de intrare prin aplicarea unui filtru peste acestea. In cazul bidimensional, convolutia bidimensionala actioneaza ca o lupa care se plimba pe o imagine si extrage informatii doar din ce vede la un moment dat care apoi sunt trimise ca date de intrare la urmatoarele straturi din retea
  - ***Implementare***: etapele de citire si afisare sunt identice cu cele de la Naïve Bayes, codul este acelasi. Dupa citirea datelor putem spune ca incepe cu adevarat construirea CNN-ului prin urmatorii pasi:
    - ***Normalizarea datelor:*** Valorile pixelilor din imaginile de antrenare, validare și testare sunt normalizate prin impartirea la 255 deoarece array-urile au valorile din a 4-a dimensiune numere intre 0 si 255 corespunzatoare codului RGB. Astfel aducem valorile acestora în intervalul [0,1], care este mai potrivit pentru procesarea de catre modelul de rețea neuronala.
    - ***Redimensionarea datelor:*** Imaginile sunt redimensionate in formatul corespunzator pentru a fi introduse în modelul de retea neuronala prin metoda *reshape* din *numpy*.
    - ***Augmentarea datelor:*** Pentru a imbunatati performanta (acuratetea) și a evita overfitting-ul, se augmenteaza datele prin aplicarea de transformari minore si random asupra imaginilor de antrenare cum ar fi rotatii si mutari. Realizam acest lucru cu ajutorul clasei *ImageDataGenerator* din biblioteca *tensorflow.keras.preprocessing.image*
    - ***Construirea modelului:***
      - Modelul este construit folosind straturi:
        - de convolutie - Straturile de baza ale CNN-ului. Un filtru gliseaza de-a lungul imaginii, efectuand produse de matrice si aduna rezultatele pentru a genera o imagine de iesire pe baza filtrelor din prima, astfel extragand caracteristici din imagine. In acest proiect folosim straturi convolutionale de

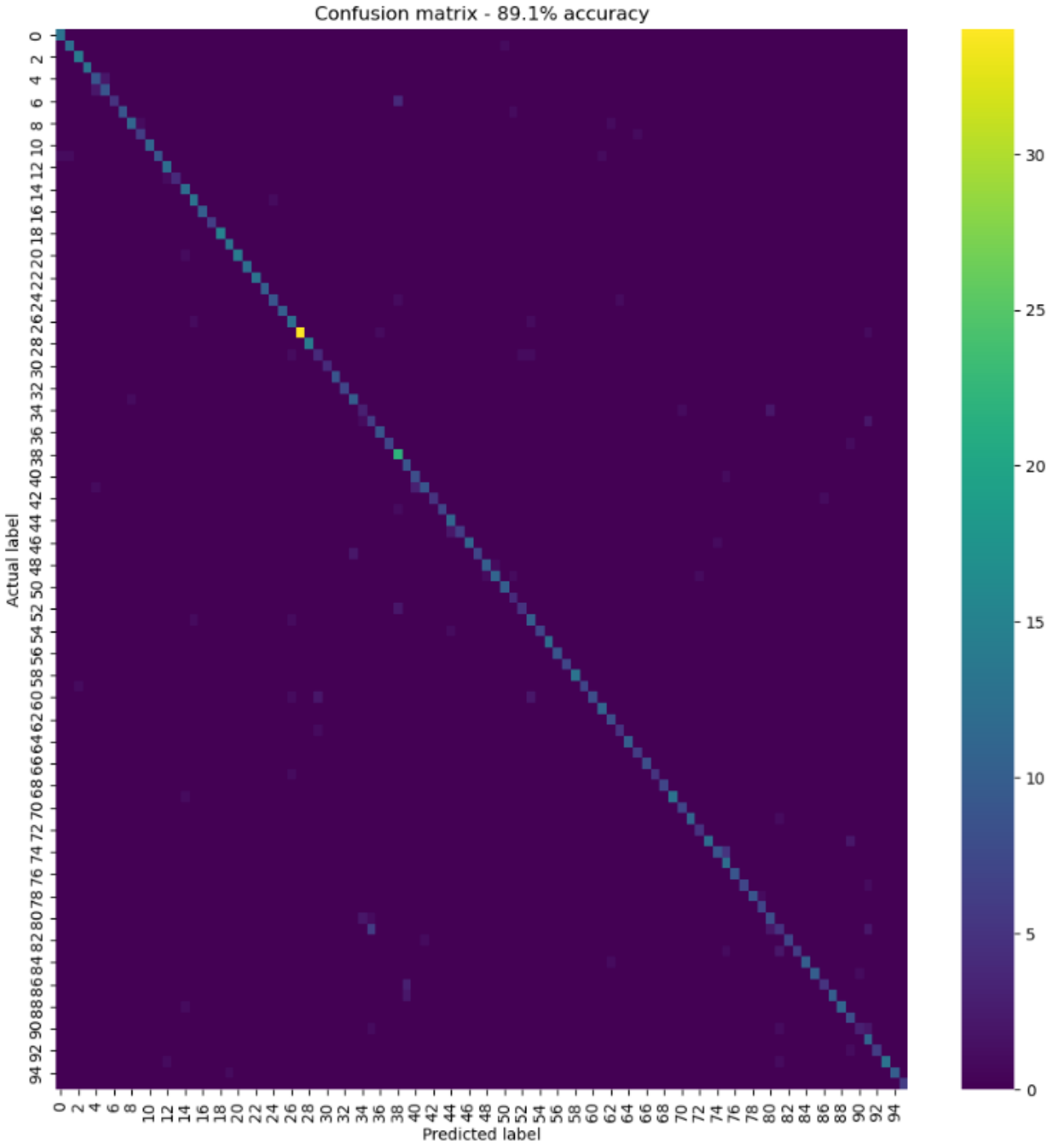
forma *Conv2D(63, (3, 3), activation='relu', padding='same')* deoarece avem ca input mereu imagini, deci obiecte bidimensionale(matrice), activam cu *ReLU* si specificam faptul ca outputul va avea aceiasi dimensiune cu inputul bordand imaginea (*padding='same'*)

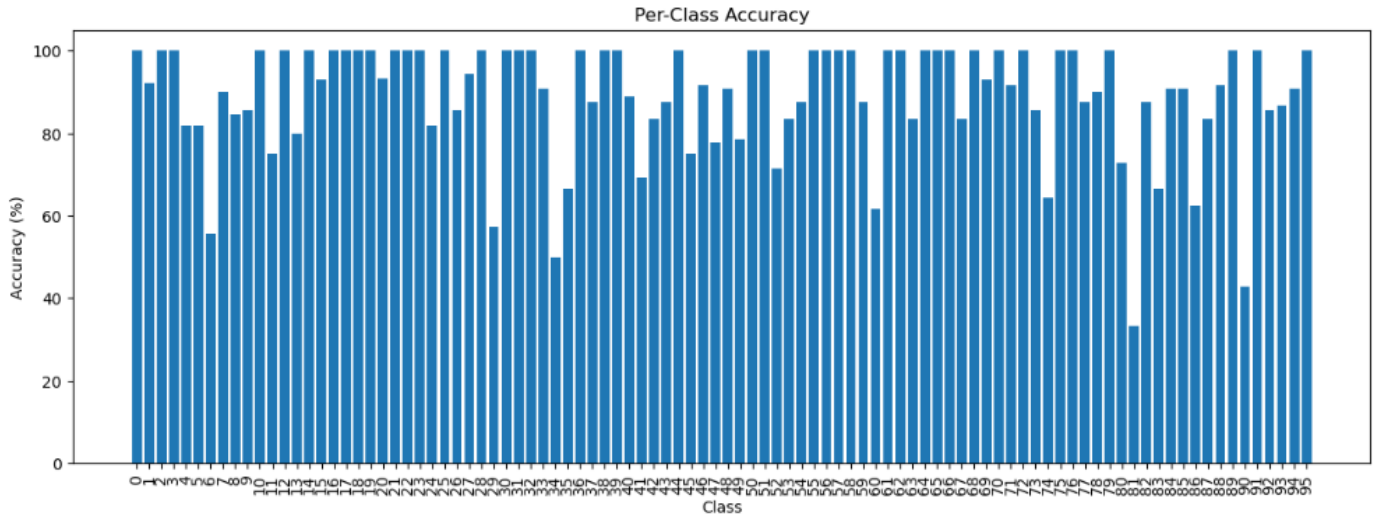
- ♦ de normalizare - Ajustam outputul stratului anterior pentru a avea o medie de zero și o deviatie standard de unu, imbunatatind performanta modelului. In acest proiect folosim normalizarea exemplificata la curs: *Batch Normalization()*.
- ♦ de pooling - Reduce dimensiunea inputului, pastrand in acelasi timp informatiile importante prin preluarea unui rezumat statistic. In proiect se foloseste *MaxPooling2D* prin care se ia valoarea maxima
- ♦ de aplatizare - Transformă datele de intrare multidimensionale intr-o forma unidimensionala. In cazul nostru, rezultatele prelucrarilor anterioare ne conduc la o matrice, un array bidimensional, pe care in etapa de aplatizare il facem unidimensional pentru a il putea conecta la straturile dense.
- ♦ dense - Straturi de neuroni complet conectate, ca intr-un graf bipartit complet, unde cele doua multimi din bipartitie reprezinta stratul precedent si stratul curent. In proioect il folosim la final cand facem clasificarea ce tine de toate caracterisiticele depistate anterior
- ♦ de regularizare – In proiectul nostru prin Dropout. Previne overfitting-ului prin dezactivarea, oprirea random a unui numar de neuroni cand se antreneaza modelul. Astfel modelul devine mai flexibil cu date noi cum ar fi cele de validare si testare

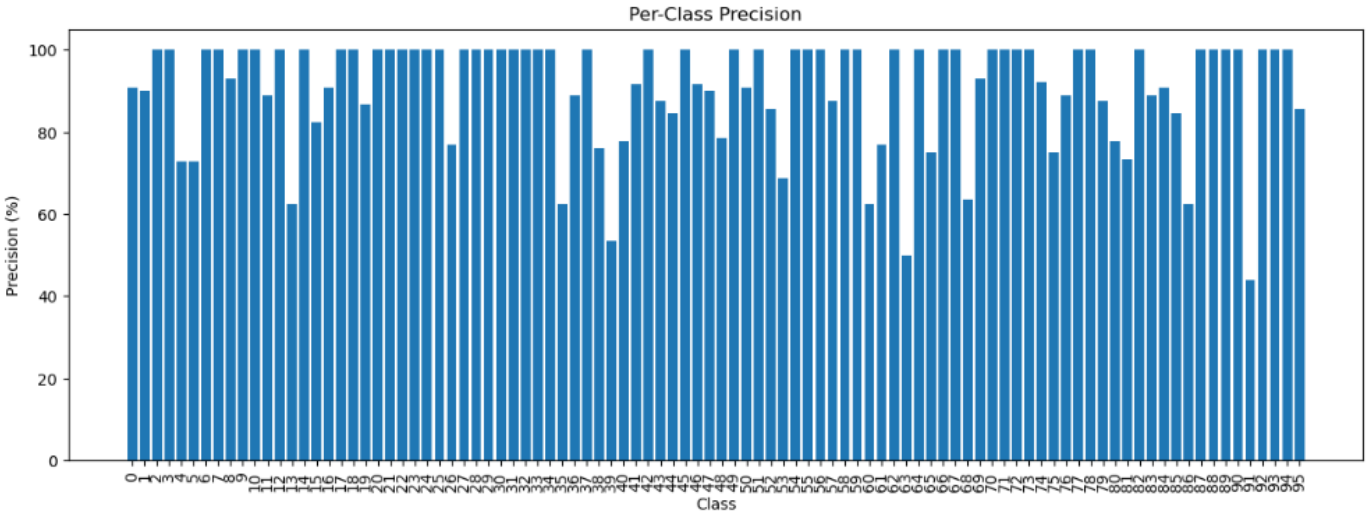
- Modelul are in total 5 straturi ascunse si un strat de iesire. Functia de activare *ReLU* este folosită pentru straturile ascunse, iar functia *softmax* pentru stratul de iesire, deoarece problema este de clasificare multipla.

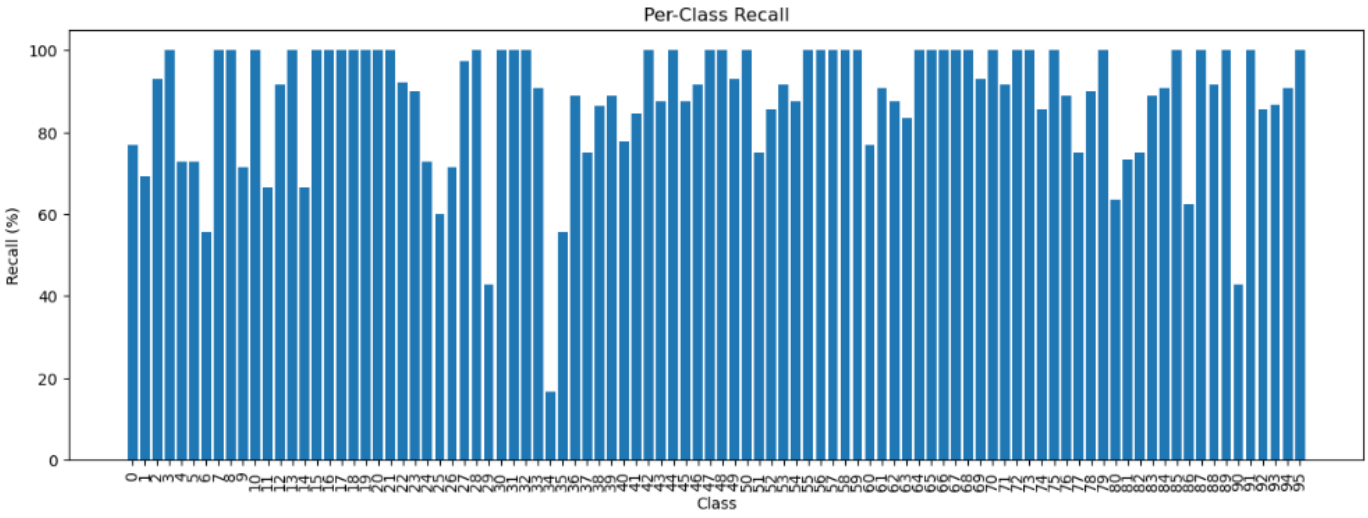
- ***Antrenarea modelului:***
  - Antrenam modelul folosind optimizatorul Adam și functia de pierdere *categorical_crossentropy*, deoarece este o problema de clasificare multipla.
  - Antrenam pe un număr de epoci specificat, pe setul de date augmentate anterior. Folosim metoda flow pe imaginile augmentate pentru a le folosi
- ***Checkpoint save:*** In timpul antrenarii, cel mai bun model este salvat pe baza acuratetii pe setul de validare iar dupa antrenare este încărcat si folosit pentru a face predictii pe setul de testare.
- ***Generarea outputului:*** Predictiile modelului optim sunt folosite pentru a genera un fisier .csv care va fi trimis ca submisie pe Kaggle

- ***Rezultate***:
  - Performanta modelului a variat pe parcursul competitiei in functie de ce modificari am facut. Prima incercare, cu 3 straturi ascunse, a rezultat intr-o rata de acuratete de aproximativ 55%. Ulterior, prin adaugarea augemntarii si normalizatii datelor, aceasta a crescut undeva la 60-70%. Apoi, printr-un lung proces de experimentare cu hiperparametrii (learning rate, optimizatori, batch size si altele descrise mai in detaliu in sectiunea ***Tunarea Hiperparametrilor***) am obtinut o acuratete de 80-85%. La final, deoarece am vazut ca pe parcursul epocilor de antrenare existau acurateti mai mari decat cele finale, am implementat strategia salvarii celui mai bun model intr-un fisier generat *best_model.h5* si am prezis cu el pe datele de testare si am obtinut o rata finala de acuratete de aproximativ 89-90%


Confusion matrix - 89.1% accuracy
Actual label
Predicted label
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
56
58
60
62
64
66
68
70
72
74
76
78
80
82
84
86
88
90
92
94
0
5
10
15
20
25
30







- ***Imbunatatirea modelului – tunarea hiperparametrilor***

- In scopul imbunatatirii performantei modelului am incercat sa modific experimental valorile anumitor hiperparametrii din cod pentru a observa daca acuratetea creste, acestia fiind:
  - Numarul de neuroni de pe fiecare strat – ajustari minore pentru a vedea daca apare o imbunatatire
  - Dimensiunea filtrului pe fiecare strat convolutional – Initial aplicam acelasi filtru 3 x 3 pe fiecare strat convolutional, insa am vazut ca daca reduc dimensiunea sa la unul 2 x 2 pe straturile mai adanci acuratetea creste
  - Rata de Dropout – Initial foloseam o regularizare cu dropout de 50% inaintea stratului de output, insa am citit in curs ca regularizarea ar trebui sa creasca treptat o data cu adancimea retelei, asa ca am adaugat dupa fiecare strat convolutional normalizat si redimensionat un dropout progresiv mai mare: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6
  - Adaugarea straturilor de normalizare dupa fiecare strat convolutional – Recomandare din curs
  - Modificarea functiei de activare – am avut un rezultat bun pe care l-am trimis printre ultimele submisii in care am inlocuit functia *ReLU* cu *Leaky ReLU* si am observat o imbunatire de 1-2%
  - Adaugarea bordarii – *padding='same'* a imbunatatit performanta cu 2-3%
  - Alegerea optimizatorului – Initial am folosit doar Adam dar am citit in curs ca exista si alti optimizatori, cum ar fi SGD(58%), Adagrad(85%), RMSProp(75%), Adadelta(0.03%), dar pana la urma tot Adam a oferit cele mai bune rezultate
  - Modificarea Learning Rate-ului – Pentru optimizatorul Adam am incercat diferite valori pentru learning rate (0.001 (default), 0.01 (a mers ok), 0.1 prea mult, a facut overfitting, a ajuns acuratetea jos de tot pe la 40-50% de la 70-80%), 0.05(nu e prea bun), 0.005(a dat 84% pe val data), earning rate 0.01 a mers cel mai bine cu un numar mare de epoci (300))
  - Modificarea numarului de epoci de antrenare – Am inceput cu numere mici, 10-15, apoi am crescut treptat la

40-50 si apoi la 200-300 sperand sa pot salva un model cat mai performant

```
Epoch 285: val_accuracy did not improve from 0.88500
188/188 [==============================] - 16s 83ms/step - loss: 0.5378 - accuracy: 0.8435 - val_loss: 0.5058 - val_accuracy: 0.8500
Epoch 286/300
188/188 [==============================] - ETA: 0s - loss: 0.5050 - accuracy: 0.8451
Epoch 286: val_accuracy did not improve from 0.88500
188/188 [==============================] - 15s 80ms/step - loss: 0.5050 - accuracy: 0.8451 - val_loss: 0.4682 - val_accuracy: 0.8800
Epoch 287/300
188/188 [==============================] - ETA: 0s - loss: 0.5162 - accuracy: 0.8491
Epoch 287: val_accuracy did not improve from 0.88500
188/188 [==============================] - 16s 84ms/step - loss: 0.5162 - accuracy: 0.8491 - val_loss: 2.2586 - val_accuracy: 0.6030
Epoch 288/300
188/188 [==============================] - ETA: 0s - loss: 0.5224 - accuracy: 0.8437
Epoch 288: val_accuracy did not improve from 0.88500
188/188 [==============================] - 15s 81ms/step - loss: 0.5224 - accuracy: 0.8437 - val_loss: 0.5803 - val_accuracy: 0.8370
Epoch 289/300
188/188 [==============================] - ETA: 0s - loss: 0.5173 - accuracy: 0.8454
Epoch 289: val_accuracy did not improve from 0.88500
188/188 [==============================] - 16s 84ms/step - loss: 0.5173 - accuracy: 0.8454 - val_loss: 0.4374 - val_accuracy: 0.8800
Epoch 290/300
188/188 [==============================] - ETA: 0s - loss: 0.5214 - accuracy: 0.8474
Epoch 290: val_accuracy did not improve from 0.88500
188/188 [==============================] - 15s 80ms/step - loss: 0.5214 - accuracy: 0.8474 - val_loss: 0.6493 - val_accuracy: 0.8380
Epoch 291/300
188/188 [==============================] - ETA: 0s - loss: 0.5066 - accuracy: 0.8508
Epoch 291: val_accuracy did not improve from 0.88500
188/188 [==============================] - 16s 84ms/step - loss: 0.5066 - accuracy: 0.8508 - val_loss: 0.5977 - val_accuracy: 0.8480
Epoch 292/300
188/188 [==============================] - ETA: 0s - loss: 0.4976 - accuracy: 0.8526
Epoch 292: val_accuracy did not improve from 0.88500
188/188 [==============================] - 15s 79ms/step - loss: 0.4976 - accuracy: 0.8526 - val_loss: 0.5786 - val_accuracy: 0.8580
Epoch 293/300
188/188 [==============================] - ETA: 0s - loss: 0.5159 - accuracy: 0.8468
Epoch 293: val_accuracy did not improve from 0.88500
188/188 [==============================] - 16s 84ms/step - loss: 0.5159 - accuracy: 0.8468 - val_loss: 0.6253 - val_accuracy: 0.8430
Epoch 294/300
188/188 [==============================] - ETA: 0s - loss: 0.5219 - accuracy: 0.8470
Epoch 294: val_accuracy did not improve from 0.88500
188/188 [==============================] - 16s 84ms/step - loss: 0.5219 - accuracy: 0.8470 - val_loss: 0.7825 - val_accuracy: 0.8290
Epoch 295/300
188/188 [==============================] - ETA: 0s - loss: 0.4962 - accuracy: 0.8480
Epoch 295: val_accuracy did not improve from 0.88500
188/188 [==============================] - 15s 80ms/step - loss: 0.4962 - accuracy: 0.8480 - val_loss: 0.6033 - val_accuracy: 0.8480
Epoch 296/300
188/188 [==============================] - ETA: 0s - loss: 0.5003 - accuracy: 0.8509
Epoch 296: val_accuracy did not improve from 0.88500
188/188 [==============================] - 16s 83ms/step - loss: 0.5003 - accuracy: 0.8509 - val_loss: 0.5628 - val_accuracy: 0.8570
Epoch 297/300
188/188 [==============================] - ETA: 0s - loss: 0.5307 - accuracy: 0.8412
Epoch 297: val_accuracy did not improve from 0.88500
188/188 [==============================] - 15s 81ms/step - loss: 0.5307 - accuracy: 0.8412 - val_loss: 0.5945 - val_accuracy: 0.8570
Epoch 298/300
188/188 [==============================] - ETA: 0s - loss: 0.4962 - accuracy: 0.8494
Epoch 298: val_accuracy improved from 0.88500 to 0.89100, saving model to best_model.hdf5
188/188 [==============================] - 15s 81ms/step - loss: 0.4962 - accuracy: 0.8494 - val_loss: 0.3818 - val_accuracy: 0.8910
Epoch 299/300
188/188 [==============================] - ETA: 0s - loss: 0.5364 - accuracy: 0.8412
Epoch 299: val_accuracy did not improve from 0.89100
188/188 [==============================] - 16s 84ms/step - loss: 0.5364 - accuracy: 0.8412 - val_loss: 0.4728 - val_accuracy: 0.8790
Epoch 300/300
188/188 [==============================] - ETA: 0s - loss: 0.5247 - accuracy: 0.8468
Epoch 300: val_accuracy did not improve from 0.89100
188/188 [==============================] - 15s 81ms/step - loss: 0.5247 - accuracy: 0.8468 - val_loss: 0.4261 - val_accuracy: 0.8640
```

- ***Observatii***

- O idee de augmentare a datelor pe care am implementat-o insa fara sa obtin rezultatele dorite a fost aplicarea unui filtru sepia peste toate imaginile de antrenare pentru a dubla volumul de date de antrenare. Principiul pe care m-am bazat este ca o imagine reprezinta acelasi lucru, indiferent de ce filtru estetic este pus pe ea. In teorie ar fi fost o idee buna, insa in practica, dupa ce am implementat functia pentru convertirea imaginii in sepia, am observat o scadere cu aproximativ 5% la rata de acuratete. O posibila explicatie ar fi ca modelul facea overfitting, fiind nevoit sa proceseze imagini aproape identice, drept care isi pierdea din capacitatea de generalizare

```python
def make_sepia(img):
    pixels = list(img.getdata())
    sepia_pixels = []
    for pixel in pixels:
        r, g, b = pixel
        new_r = min(int(r * 0.393 + g * 0.769 + b * 0.189), 255)
        new_g = min(int(r * 0.349 + g * 0.686 + b * 0.168), 255)
        new_b = min(int(r * 0.272 + g * 0.534 + b * 0.131), 255)
        sepia_pixels.append((new_r, new_g, new_b))
    sepia_img = Image.new("RGB", img.size)
    sepia_img.putdata(sepia_pixels)
    return np.array(sepia_img)


def load_images(folder, train = 0):
    images = dict()
    for filename in os.listdir(folder):
        with Image.open(os.path.join(folder, filename)) as img:
            if train == 1:
                original_img = np.array(img)
                sepia_img = make_sepia(img)
                images[filename] = [original_img, sepia_img]
            else:
                images[filename] = np.array(img)
    return images


train_data_dict = load_images('/kaggle/input/unibuc-dhc-2023/train_images', 1)
test_images_dict = load_images('/kaggle/input/unibuc-dhc-2023/test_images')
val_data_dict = load_images('/kaggle/input/unibuc-dhc-2023/val_images')

train_labels_dict = {line.split(',')[0]: int(line.split(',')[1].strip('\n')) for line in
                     open("/kaggle/input/unibuc-dhc-2023/train.csv", "r").readlines()[1:]}
test_images_names = [line.strip('\n') for line in open("/kaggle/input/unibuc-dhc-2023/test.csv", "r").readlines()[1:]]
val_labels_dict = {line.split(',')[0]: int(line.split(',')[1].strip('\n')) for line in
                   open("/kaggle/input/unibuc-dhc-2023/val.csv", "r").readlines()[1:]}

train_images = []
train_labels = []

for nume_imagine in train_data_dict.keys():
    train_images += train_data_dict[nume_imagine]
    train_labels += [train_labels_dict[nume_imagine], train_labels_dict[nume_imagine]]
```

- Codul include multe comentarii personale, e ca un fel de jurnal al procesului de dezvoltare al acestei Retele Neuronale Convolutionale

- ***Concluzii***
  - Produsul final al acestui proces de dezvoltare este un cod in python capabil sa identifice corect ce reprezinta o opera de arta contemporana in aproximativ 9/10 cazuri
  - Este o demonstratie practica a eficientei anumitor algoritmi in functie de context comparativ cu a altora
  - Mi-a facut placere sa lucrez la acest proioect, chiar daca m-am apucat mai tarziu de el, deoarece mi se pare fascinant cum reuseste cu ajutorului unui cod o masina sa inteleaga semnificatia unui tablou.
  - Am inteles mai bine anumite concepte discutate la curs si am invatat multe lucruri noi, practice, pe care mi-as dori sa le folosesc si in alte proiecte viitoare
  - Regret ca nu am mai avut timp sa imi implementez o idee, aceea de a defini un planificator pentru invatare (learning rate scheduler) deoarece am vazut din cursuri ca o rata de invatare care descreste pe masura ce inaintam in epoci conduce la rezultate mai bune
  - Acest proiect m-a ajutat sa ma decid ca vreau sa merg mai departe pe ramura de Machine Learning!