

# PROIECT INTELIGENȚA ARTIFICIALĂ **DEEP HALLUCINATION CLASSIFICATION**

LECTOR RADU TUDOR IONESCU

STUDENTĂ: NEAGU MARA TEODORA

SERIA 23, GRUPA 231

ANUL II, SEMESTRUL II

2023

## MODELE

Pentru a clasifica setul de date primit, care constă din tupluri de forma imagine, clasa corespunzătoare din care aceasta face parte, am utilizat două tipuri de modele pentru a obține o clasificare cât mai precisă a datelor de test: CNN (Convolutional Neural Network) și k-NN (k-Nearest Neighbours).

În continuare, voi prezenta o descriere detaliată a fiecărui model în parte, împreună cu codurile individuale în care acestea au fost utilizate.

### Convolutional Neural Network (CNN) – 80.6%

Modelul CNN este un tip de rețea neurală. Acesta este specializat în prelucrarea imaginilor, folosind următoarele elemente principale în prelucrarea acestora: straturi de convoluție pentru extragerea anumitor caracteristici relevante în datele de intrare, straturi de pooling, care reduc dimensiunea datelor prin selectarea celor mai semnificative informații dintr-o anumită fereastră de pooling indicate și straturile complet conectate (fully – connected), în care fiecare neuron este conectat la neuronii din stratul anterior; sunt utilizate în scopul combinării caracteristicilor extrase anterior prin convoluție și pooling, generând rezultatele generale de clasificare.

```
import pandas as pd
from PIL import Image
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

# --- DATA ---

# LOADING THE DATA FOR OUR MODEL
trainingDataFrame = pd.read_csv('/kaggle/input/unibuc-m12/train.csv')
validationDataFrame = pd.read_csv('/kaggle/input/unibuc-m12/val.csv')
testingDataFrame = pd.read_csv('/kaggle/input/unibuc-m12/test.csv')

# DEFINING THE IMAGES DIRECTORIES FOR OUR THREE STAGES
trainingImagesDirectory = '/kaggle/input/unibuc-m12/train_images'
validationImagesDirectory = '/kaggle/input/unibuc-m12/val_images'
testingImagesDirectory = '/kaggle/input/unibuc-m12/test_images'

# DEFINING THE GENERAL IMAGE SIZE
imageSize = (64, 64)
```

În această parte, se încarcă datele de antrenament, validare și testare. Aceste date sunt compuse din clasificările predefinite pentru fiecare set și imaginile propriu-zise asociate acestor clasificări. La final, se definește dimensiunea imaginilor modelul nostru.

```
# --- DATA PROCESSING ---

# FUNCTION TO PROCESS THE DATA THAT WE GET FROM THE DATAFRAME AND THE DIRECTORIES
def processing(dataFrame, imagesDirectory, isTestingImagesDirectory=False):
    images = []
    labels = []

    # ITERATING THROUGH EACH LINE OF THE DATAFRAME, EXTRACTING THE "IMAGE" AND THE "CLASS", IF IT EXISTS
    for index, row in dataFrame.iterrows():

        # CREATING THE FULL PATH TO THE IMAGE FILE
        imagePath = imagesDirectory + '/' + row['Image']

        # CONVERTS THE IMAGE TO THE RGB COLOR MODE, ENSURING THAT THE IMAGE HAS THREE COLOR CHANNELS (RED, GREEN, BLUE), REQUIRED BY THE CNN MODEL
        image = Image.open(imagePath).convert('RGB')

        # RESIZING THE IMAGE TO THE REQUIRED SIZE, THUS ENSURING THAT ALL THE IMAGES HAVE THE SAME DIMENSIONS
        image = image.resize(imageSize)

        # CREATING A NUMERICAL REPRESENTATION OF THE IMAGE THAT CAN BE PROCESSED BY THE CNN MODEL
        image = np.array(image)

        # ADDING THE NEW IMAGE TO OUR ARRAY OF ALREADY PROCESSED IMAGES
        images.append(image)

        # IF WE ARE PROCESSING THE TRAINING IMAGES OR THE VALIDATION IMAGES, WE HAVE THE "CLASS" COLUMN AND WE ARE ADDING IT TO THE EXISTING CLASSES
        if not isTestingImagesDirectory:
            label = row['Class']
            # ADDING THE NEW LABEL TO OUR ARRAY OF ALREADY PROCESSED LABELS
            labels.append(label)

    # WE ARE CONVERTING THE IMAGES AND THE LABELS PROCESSED TO AN ARRAY
    images = np.array(images)
    if not isTestingImagesDirectory:
        labels = np.array(labels)
    return images, labels
else:
    return images

# EXTRACTING THE DATA FOR OUR MODEL FROM THE DATAFRAME
trainingImages, trainingLabels = processing(trainingDataFrame, trainingImagesDirectory)
validationImages, validationLabels = processing(validationDataFrame, validationImagesDirectory)
testingImages = processing(testingDataFrame, testingImagesDirectory, isTestingImagesDirectory=True)
```

În această parte, am definit funcția *processing* pentru a procesa datele din dataframe și directoarele de imagini asociate cu seturile de antrenament, validare și testare.

Pentru fiecare tuplu din dataframe, se extrage numele fișierului imaginii și, dacă există, clasa corespunzătoare. Imaginea este convertită în modul de culoare RGB pentru a se asigura că are cele trei canale de culoare necesare modelului CNN: roșu, verde și albastru. Apoi, imaginea este redimensionată la o dimensiune constantă de 64 x 64 pixeli pentru a asigura uniformitatea dimensiunilor datelor de intrare. Ulterior, se creează o reprezentare numerică a imaginii sub forma unui tablou numpy, care poate fi procesată de modelul CNN, și aceasta este adăugată într-un tablou de imagini deja procesate.

Dacă se procesează datele de antrenament sau de validare, se extrage și clasa asociată fiecărei imagini și aceasta este adăugată într-un tablou de etichete deja procesate.

```
# NORMALIZING THE IMAGES FROM OUR DATA TO THE RANGE OF [0,1], ENSURING A MORE EFFICIENT TRAINING, PREVENTING A SPECIFIC FEATURE FROM DOMINATING
trainingImages = trainingImages / 255.0
validationImages = validationImages / 255.0
testingImages = testingImages / 255.0

# CONVERT LABELS TO THE CATEGORICAL FORMAT, WE TRANSFORM THE CLASSES INTO A BINARY MATRIX REPRESENTATION, EACH CLASS REPRESENTED BY A BINARY VECTOR WITH A SIM
labelEncoder = LabelEncoder()

# WE ARE USING "fit_transform" BECAUSE IT INVOLVES BOTH LEARNING THE MAPPING OF CLASSES TO INTEGERS AND THEN APPLYING THE TRANSFORMATION TO THE LABELS
trainingLabels = labelEncoder.fit_transform(trainingLabels)

# WE ARE USING "transform" BECAUSE IT APPLIES THE LEARNED MAPPING FROM THE trainingLabels AND THUS IT NEEDS ONLY TO APPLY THE TRANSFORMATION TO THE LABELS
validationLabels = labelEncoder.transform(validationLabels)
```

În această parte, imaginile din seturile de antrenament, validare și testare sunt normalizate în intervalul  $[0, 1]$  pentru a asigura un antrenament mai eficient al modelului. În acest mod este prevăzută dominarea unei caracteristici specifice în timpul antrenamentului.

Pentru a transforma clasele într-un format categoric adecvat pentru antrenament, se utilizează obiectul *LabelEncoder*. Prin utilizarea metodei *fit\_transform*, se învață maparea claselor în valori întregi și se aplică transformarea asupra etichetelor din setul de antrenament. Pentru setul de validare, se folosește metoda *transform* pentru a aplica maparea învățată anterior în cadrul etichetelor de antrenament.

```
# --- THE CNN MODEL ---

model = Sequential()
# LAYER WITH 64 FILTERS, EACH WITH THE SIZE OF 3X3
model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(imageSize[0], imageSize[1], 3)))
# LAYER THAT REDUCES THE DIMENSIONALITY AND EXTRACTS THE MOST IMPORTANT FEATURES
model.add(MaxPooling2D((2, 2)))
# WE REPEATED THE PROCESS FOR 128 AND 256 FILTERS, AND THEN APPLIED MaxPooling2D ON BOTH
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
# FLATTENS THE 3D OUTPUT FROM THE PREVIOUS LAYERS INTO A 1D VECTOR, PREPARING THE DATA FOR THE FULLY CONNECTED LAYERS
model.add(Flatten())
# FULLY CONNECTED LAYER WITH 256 UNITS THAT LEARNS COMPLEX PATTERNS FROM FLATTENED FEATURES
model.add(Dense(256, activation='relu'))
# DROPOUT TO PREVENT OVERFITTING OVER THE TRAINING ACCURACY
model.add(Dropout(0.4))
# PRODUCES THE OUTPUT PROBABILITIES FOR EACH CLASS, INDICATING THE LIKELIHOOD OF AN IMAGE BELONGING TO EACH CLASS
model.add(Dense(96, activation='softmax'))
```

În această parte, este definit modelul CNN, utilizându-se obiectul *Sequential*. Apoi sunt definite straturile de convoluție, pooling și conectare completă. Le voi descrie pe fiecare în parte.

- Primul strat este un strat de convoluție *Conv2D* cu 64 de filtre de dimensiune  $3 \times 3$  și funcția de activare *ReLU*. Prin aplicarea operației de convoluție și utilizarea mai multor filtre, stratul de convoluție extrage caracteristici din imaginea de intrare, capturând detalii relevante pentru clasificare.
- Urmează un strat de reducere a dimensionalității *MaxPooling2D* cu fereastră de  $2 \times 2$ , care reduce dimensiunea datelor și extrage caracteristici semnificative.
- Urmează două straturi de convoluție *Conv2D* cu 128 și, respectiv, 256 de filtre de dimensiune  $3 \times 3$ . Aceste straturi extrag și mai multe caracteristici din imagini. După fiecare strat de convoluție, se aplică un strat de reducere a dimensionalității *MaxPooling2D*.
- Urmează un strat de aplanare *Flatten* care transformă ieșirea tridimensională într-un vector unidimensional.
- Urmează un strat complet conectat *Dense* cu 256 de unități și funcția de activare *ReLU*. Acest strat învață modele complexe din caracteristicile aplatizate la pasul anterior.
- Pentru a preveni fenomenul de overfitting (acuratețea datelor de antrenament este exponențial mai mare față de datele de validare), se adaugă un strat de eliminare *Dropout* cu o rată de 0,4.

- Ultimul strat este unul complet conectat *Dense* cu 96 de unități și funcția de activare *softmax*. Acest strat produce probabilitățile de ieșire pentru fiecare clasă, indicând probabilitatea de apartenență a unei imagini la fiecare clasă.

```
# DATA AUGMENTATION FOR THE TRAINING OF OUR MODEL
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.1,
    horizontal_flip=True,
    vertical_flip=True
)
```

Pentru a diversifica setul nostru de date de antrenament, folosim tehnici de augmentare a datelor cu clasa *ImageDataGenerator*. Această abordare ne permite să generăm noi exemple de antrenament prin transformarea imaginilor. Astfel, obținem un set de date mai variat, ceea ce poate îmbunătăți capacitatea modelului nostru de a se generaliza mai bine.

```
# COMPILING THE MODEL
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# TRAINING THE MODEL
model.fit(datagen.flow(trainingImages, trainingLabels, batch_size=32),
          steps_per_epoch=len(trainingImages) // 32,
          epochs=54,
          validation_data=(validationImages, validationLabels))

# THE TRAINING ACCURACY FOR OUR MODEL
_, trainingAccuracy = model.evaluate(trainingImages, trainingLabels, verbose=0)
print("Training Accuracy:", trainingAccuracy)

# THE VALIDATION ACCURACY FOR OUR MODEL
_, validationAccuracy = model.evaluate(validationImages, validationLabels, verbose=0)
print("Validation Accuracy:", validationAccuracy)

# TESTING THE MODEL
predictions = model.predict(testingImages)
# FINDING THE INDEX OF THE CLASS WITH THE HIGHEST PROBABILITY FOR EACH IMAGE, FOUND ALONG THE ROW AXIS
# THEY ARE THEN MAPPED FROM THEIR NUMERICAL LABELS BACK TO THEIR ORIGINAL CLASS LABELS
predictedLabels = labelEncoder.inverse_transform(np.argmax(predictions, axis=1))
```

În această parte, modelul este compilat utilizând optimizatorul *adam*, funcția de loss *sparse\_categorical\_crossentropy* și metrica *accuracy*.

Modelul este antrenat utilizând datele de antrenament prin apelul funcției *fit*. Datele de antrenament au fost furnizate în loturi de dimensiune 32. Am specificat numărul de pași pe epocă utilizând *steps\_per\_epoch* și am setat numărul total de epoci la 54. Pentru validare, am folosit setul de date de validare *validationImages* și *validationLabels*.

După antrenare, evaluăm performanța modelului pe datele de antrenament și de validare. Utilizăm funcția *evaluate* pentru a obține acuratețea atât pentru datele de antrenament, cât și pentru cele de validare.

Pentru a testa modelul pe datele de testare, utilizăm funcția *predict* pentru a obține predicțiile. Apoi, folosim *inverse\_transform* pentru a mapa etichetele numerice înapoi la etichetele de clasă originale.

```
# --- THE SUBMISSION FILE ---

# CREATING THE SUBMISSION FILE
submissionDataFrame = pd.DataFrame({'Image': testingDataFrame['Image'], 'Class': predictedLabels})
submissionDataFrame.to_csv('/kaggle/working/submission.csv', index=False)
```

În această parte, este creat fișierul de submission utilizând un *DataFrame* nou numit *submissionDataFrame*. Acesta conține numele fișierelor de testare și etichetele prezise pentru aceste imagini.



## **k – Nearest Neighbours (k – NN) – 11.9%**

Modelul k-NN (k-nearest neighbors) este un algoritm de învățare utilizat pentru clasificarea datelor. Acesta se bazează pe ideea că imaginile similare tind să apară în aceeași clasă. Pentru a clasifica o nouă imagine, algoritmul identifică cei mai apropiați vecini din setul de date de antrenament și atribuie clasa majoritară dintre acești vecini. Astfel, k-NN compară caracteristicile imaginii noi cu cele ale vecinilor săi și decide clasa bazându-se pe votul majorității. Parametrul k determină câți vecini sunt considerați în procesul de votare.

Încărcarea și procesarea datelor de antrenament, validare și testare este similară cu cea folosită în cazul modelului CNN. Cu toate acestea, modelul k-NN nu necesită straturi de convoluție, pooling sau conectare completă, iar datele de augmentare nu sunt aplicabile direct asupra imaginilor. Prin urmare, putem să trecem direct la crearea modelului k-NN.

```
# --- THE KNN MODEL ---

trainingImages = trainingImages.reshape(len(trainingImages), -1)
validationImages = validationImages.reshape(len(validationImages), -1)
testingImages = testingImages.reshape(len(testingImages), -1)

# CREATE AND TRAIN THE KNN MODEL
model = KNeighborsClassifier(n_neighbors=5)
model.fit(trainingImages, trainingLabels)

# THE TRAINING ACCURACY FOR OUR MODEL
trainingAccuracy = model.score(trainingImages, trainingLabels)
print("Training Accuracy:", trainingAccuracy)


# THE VALIDATION ACCURACY FOR OUR MODEL
validationAccuracy = model.score(validationImages, validationLabels)
print("Validation Accuracy:", validationAccuracy)

# PREDICT LABELS FOR TESTING IMAGES
predictedLabels = model.predict(testingImages)
```


În acest segment de cod, sunt pregătite datele de antrenament, validare și testare prin redimensionarea lor pentru a se potrivi cu formatul necesar modelului k-NN. Apoi, creăm și antrenăm modelul k-NN, specificând că acesta va utiliza cei 5 vecini cei mai apropiați în procesul de clasificare. Acuratețea modelului este evaluată pe datele de antrenament și validare. În final, prezicem etichetele claselor pentru imaginile de testare folosind modelul antrenat.

## SUBMISII

### Convulational Neural Network (CNN)

	submission (8).csv Complete · 3d ago	0.56542	0.582
---	---	---------	-------

*fără augmentarea datelor, 30 de epoci, fără Dropout(), overfitting*

	submission (10).csv Complete · 3d ago	0.75742	0.77333
---	--	---------	---------

*augmentarea datelor, 30 de epoci*

	submission (11).csv Complete · 3d ago	0.79228	0.806
---	--	---------	-------

*augmentarea datelor, 54 de epoci*

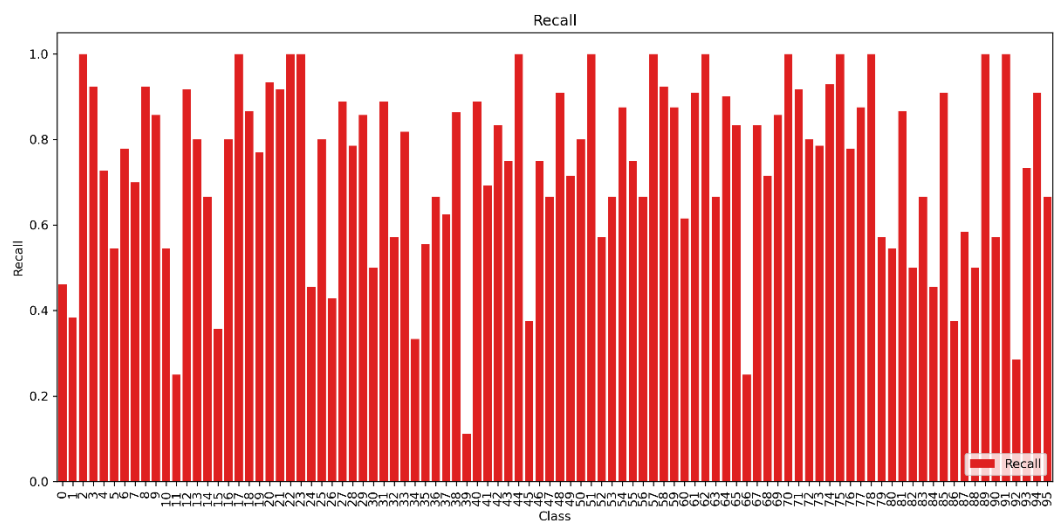
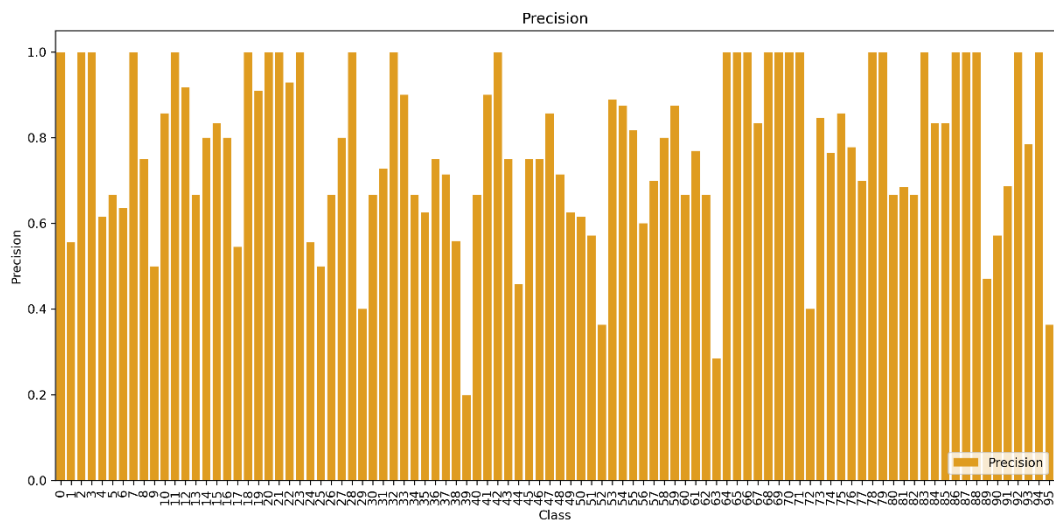
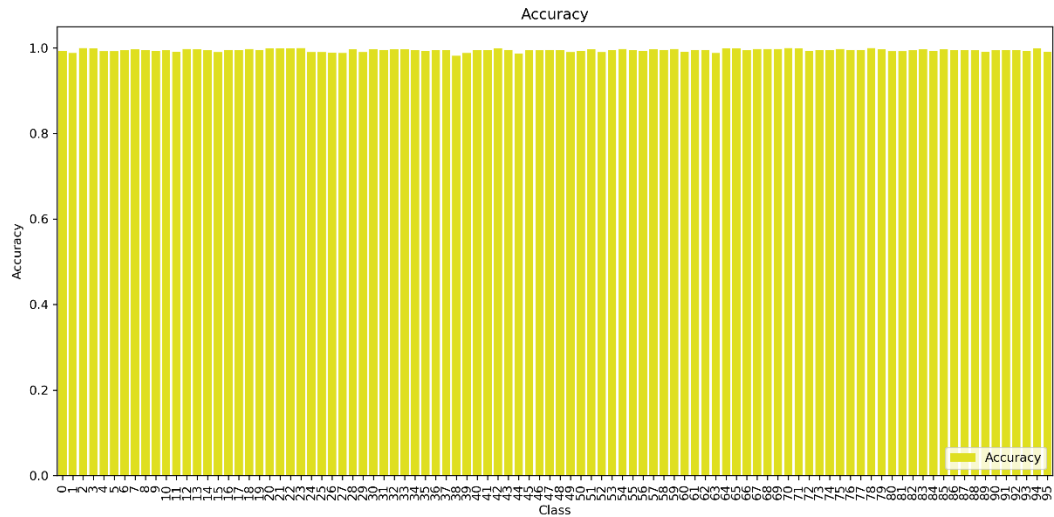
### k – Nearest Neighbours (k – NN)

	submission.csv Complete · 3d ago	0.12257	0.11933
---	-------------------------------------	---------	---------

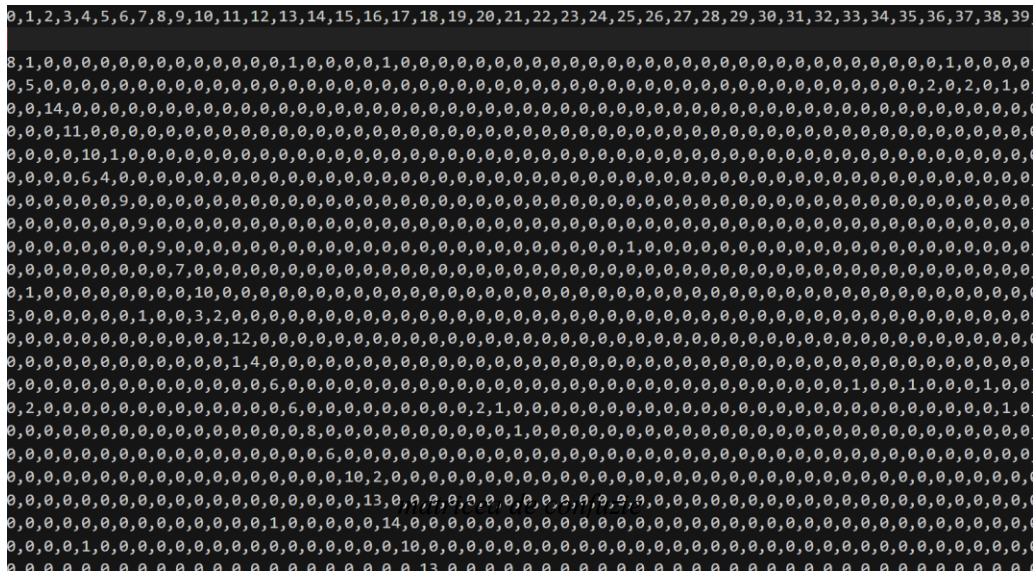
*5 vecini*

## GRAFICE

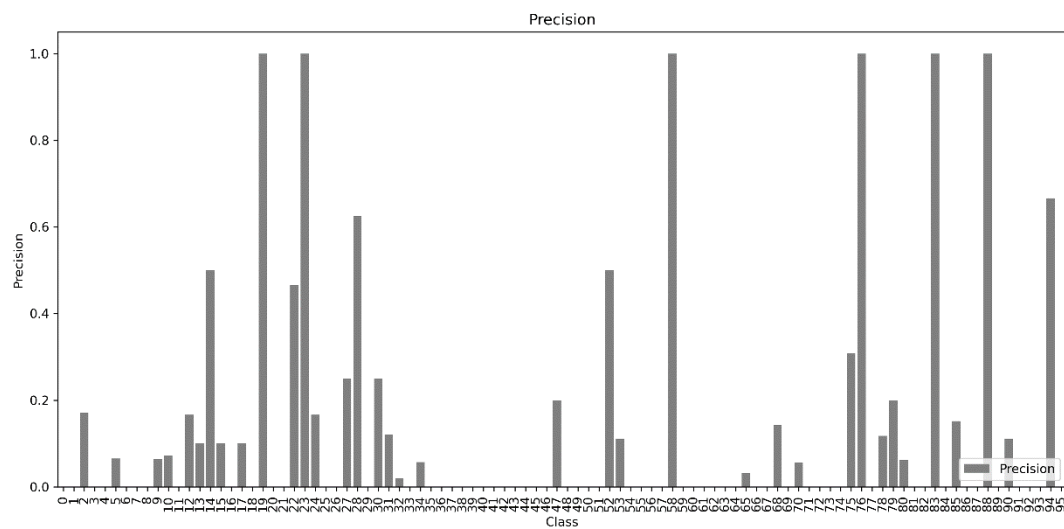
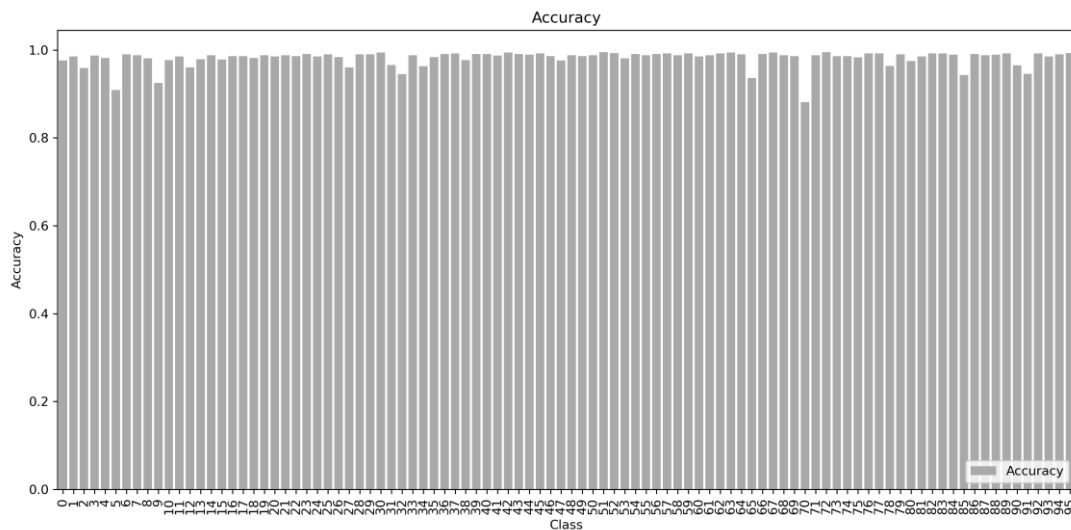
### Convulational Neural Network (CNN) – 80.6%

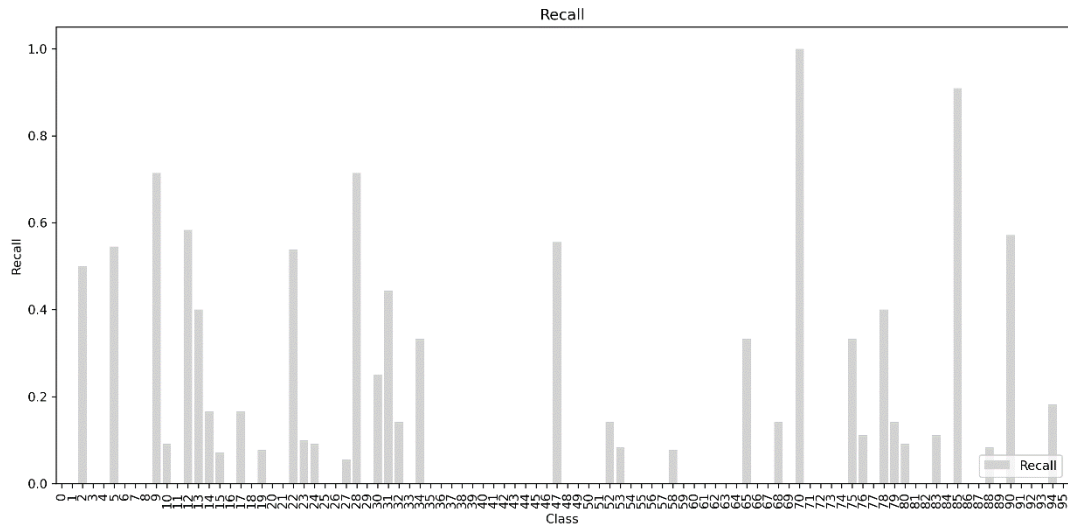






 **k – Nearest Neighbours (k – NN) – 11.9%**



[illegible]

*matricea de confuzie*