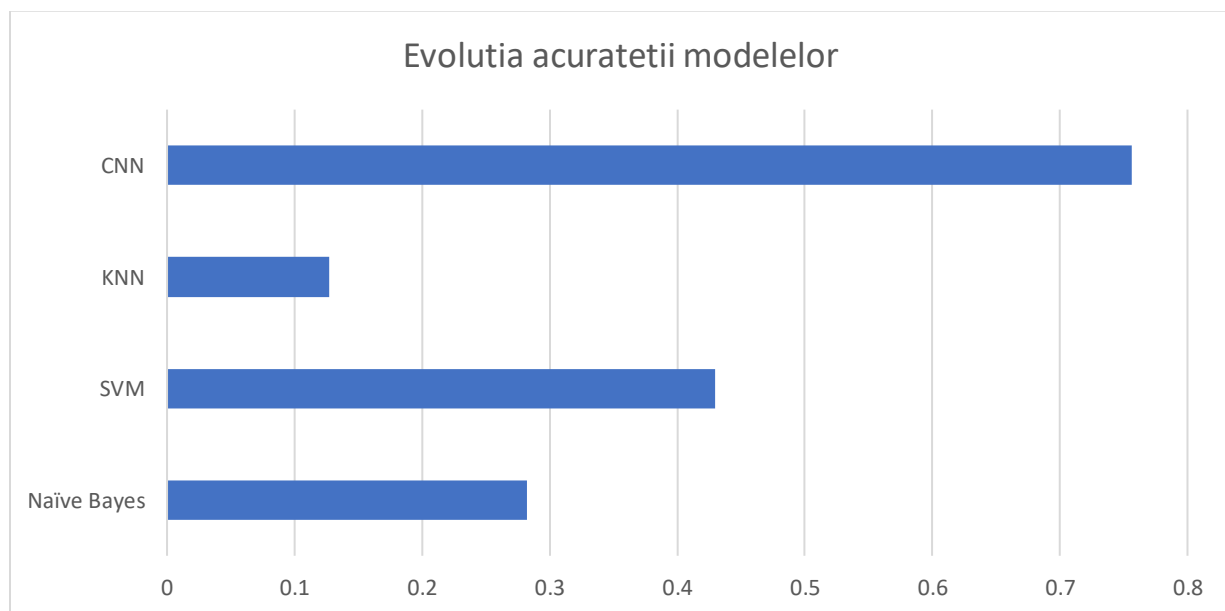


## Deep Hallucination Classification

Pentru acest proiect, am creat, in total, 4 modele pentru clasificarea deep hallucination in una din cele 96 de clase date. Modelele folosite sunt: Naïve-Bayes, SVM, KNN si CNN. Acestea au avut o acuratete din ce in ce mai buna pe datele de antrenare, alegand, in final, cele mai bune doua acurateti pentru datele de test oferite. Modele vor fi explicate pe larg si amanuntite putin mai tarziu Pentru moment, valorile finale obtinute pentru fiecare model sunt:

Model	Acuratete
Naïve-Bayes	0,28200
SVM	0,43
KNN	0,12733
CNN	0,756

Graficul acuratetilor maxime obtinute este:



Dupa cum se poate observa in grafic, cea mai buna acuratete a avut-o CNN-ul, urmat de SVM, Naïve-Bayes si KNN. Voi descrie, pe rand, fiecare model folosit si-i voi detalia modul de lucru, implementarea, cat si alte aspecte importante pentru acesta.

## 1. NAIVE-BAYES

Naïve-Bayes este un algortim de de invatare supervizata cunoscut pentru usurinta si eficienta sa pe un set de date. Am implementat acest model utilizand biblioteca scikit-learn.

### *1.1 Importarea bibliotecilor*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os
from sklearn import preprocessing
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix
```

Voi folosi bibliotecile importate pentru a manipula datele, pentru preprocesarea datelor, pentru implementarea modelului Naïve-Bayes si pentru a calcula, ulterior, scorul de acuratete si matricea de confuzie.

## *1.2 Citirea si scrierea in fisier*

```
def images(images_dir, image_names):
    images = []
    for element in image_names:
        image = plt.imread(os.path.join(images_dir, element))
        images.append(image.flatten())
    return np.array(images)

def csv(csv_path, column_names, has_labels=True):
    if has_labels:
        column_data = [pd.read_csv(csv_path)[column_name].tolist() for column_name in column_names]
        return column_data
    else:
        image_names = pd.read_csv(csv_path)[column_names[0]].tolist()
        return image_names

train_csv_path = '/kaggle/input/unibuc-dhc-2023/train.csv'
train_images_dir = '/kaggle/input/unibuc-dhc-2023/train_images'
train_image_names, train_labels = csv(train_csv_path, ['Image', 'Class'], has_labels=True)
train_images = images(train_images_dir, train_image_names)

test_csv_path = '/kaggle/input/unibuc-dhc-2023/test.csv'
test_images_dir = '/kaggle/input/unibuc-dhc-2023/test_images'
test_image_names = csv(test_csv_path, ['Image', 'Class'], has_labels=False)
test_images = images(test_images_dir, test_image_names)

test_images = scaler.transform(test_images)

test_predictions = model.predict(test_images)

test_data = {'Image': test_image_names, 'Class': test_predictions}
pd.DataFrame(test_data).to_csv('test_predictions.csv', index=False)

print("CSV - created.")
```

Pentru citire, am folosit doua functii ajutatoare(images si csv) pentru a putea citi atat dintr-un director de poze (train/val/test\_images\_dir), cat si pentru a putea citi numele si clasele, unde era cazul, pentru setul de date corespunzator. In plus, functia csv detecteaza cu ajutorul parametrului setat implicit **True** daca pentru un fisier dat ca parametru exista label-uri (clasele unde sunt repartizate pozele). Daca acestea exista, citesc din path-ul dat ca parametru coloanele fisierului, altfel citesc numai denumirile imaginilor.

Pentru scrierea in fisier, realizez un dictionar, dupa ce predictiile au fost efectuate, avand ca si chei 'Image' si 'Class', iar ca valori voi pune predictiile aferente. Ulterior, utilizeaz functia **pd.DataFrame(test\_data)** pentru a transforma dictionarul facut intr-un obiect DataFrame. Apeland metoda **to\_csv('test\_predictions.csv', index=False)** a obiectului DataFrame, rezultatul este salvat intr-un fisier CSV. Parametrul **index=False** este utilizat pentru a exclude indexul randurilor din fisierul CSV rezultat.

### *1.3 Utilizarea datelor train, val, test si preprocesarea lor*

Dupa citirea datelor de train, le-am scalat folosit **scaler = preprocessing.StandardScaler()** si **scaler.fit(train\_images)** astfel incat sa ajustez intervalul si dispersia datelor pentru a putea sa le compar, iar distributia lor sa fie cat mai aproape de cea normala. **Scaler.fit()** calculeaza media si deviatia standard a fiecarei caracteristici, urmand ca **train\_images = scaler.transform(train\_images)** sa-mi ajusteze fiecare caracteristica la o scara standard. Vom realiza acest lucru atat pentru val (**val\_images = scaler.transform(val\_images)**), cat si pentru test (**test\_images = scaler.transform(test\_images)**), deoarece imi doresc ca datele mele, in principiu train si val, sa fie comparabile, pentru ca pe train antrenez modelul, initial, si, apoi, dau predict pe val sa vad cat de bine a a performat invatarea modelul meu.

### *1.4 Model Naïve-Bayes*

Am incercat sa creez modele Naïve-Bayes cu diferite distributii: **GaussianNB()** (distributie normala gaussiana), **MultinomialNB()** (model Naïve-Bayes pentru caracteristici de tip multinomial) si **BernoulliNB()** (pentru caracteristici binare Bernoulli). Am comparat cele trei acurateti obtinute dupa cum se vede in tabelul de mai jos:

Model Naïve-Bayes	Acuratete
GaussianNB()	0,27
MultinomialNB()	0,186
BernoulliNB()	0,184

Dupa cum se poate observa, am ales sa antrez modelul cu ajutorul **GaussianNB()**, intrucat avea o acuratete vizibil mai buna decat celelalte doua.

### 1.5 Matricea de confuzie

Matricea de confuzie generata de cod:

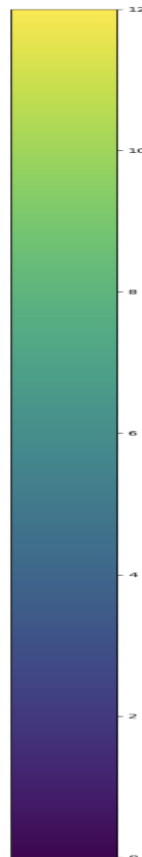
```
from sklearn.metrics import confusion_matrix

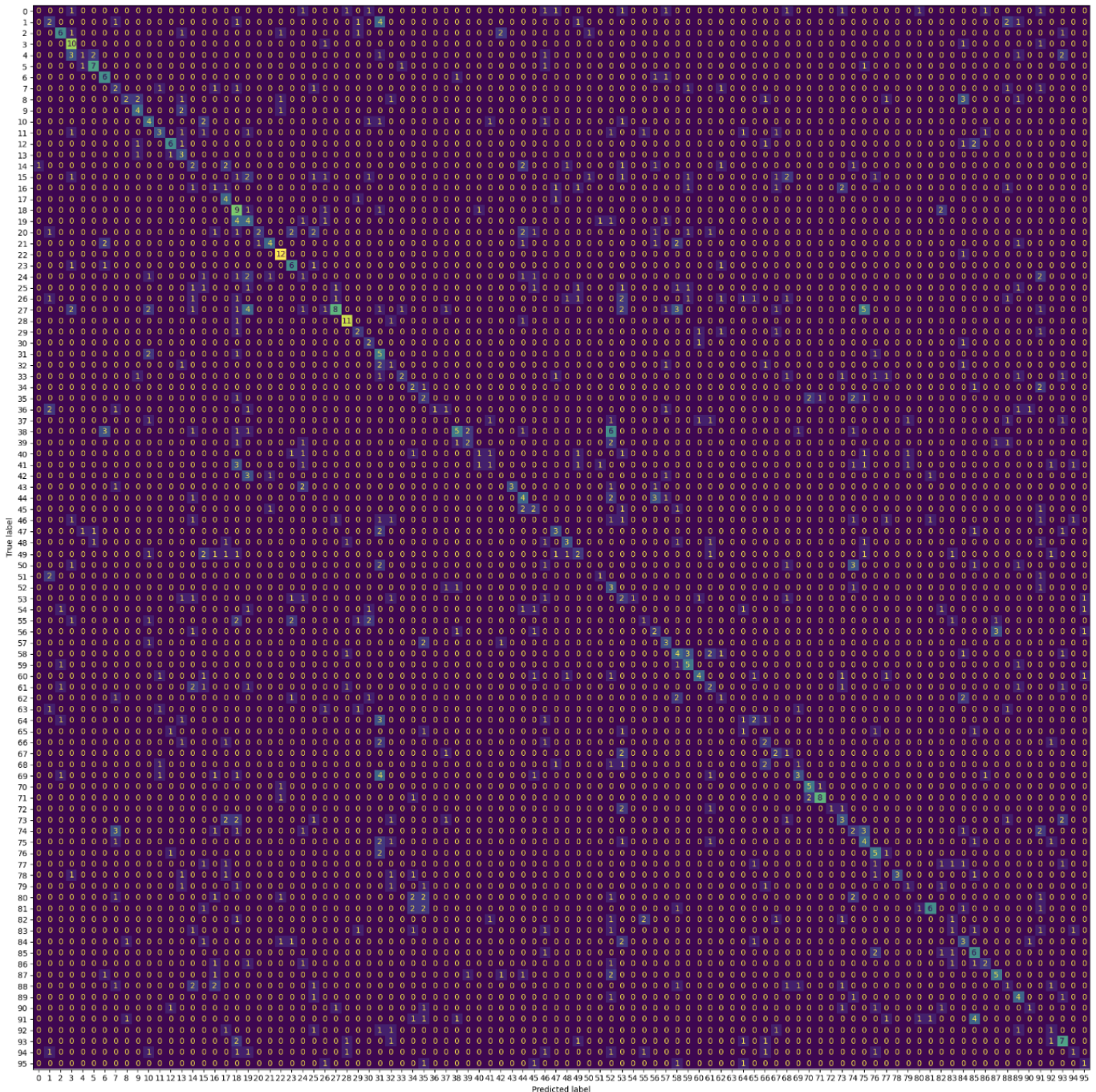
print(confusion_matrix(val_labels, predictions))
```

```
[[0 0 0 ... 0 0 0]
 [0 2 0 ... 0 0 0]
 [0 0 6 ... 1 0 0]
 ...
 [0 0 0 ... 7 0 0]
 [0 1 0 ... 0 1 0]
 [0 0 0 ... 0 0 1]]
```

+ Code   + Markdown

Reprezentare mai pe larg a matricii de confuzie:





### 1.6. Concluzii

Desi, la prima vedere, acuratetea este peste baseline, am obtinut, totusi, rezultate mai bune.

## 2. SVM

SVM (Support Vector Machines) este un algoritm de invatare supervizata utilizat in principal pentru clasificare si regresie.

### 2.1 Importarea bibliotecilor

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
```

Voi folosi bibliotecile importate pentru a manipula datele, pentru preprocesarea datelor, pentru implementarea modelului SVM si pentru a calcula, ulterior, scorul de acuratete si matricea de confuzie.

### 2.2 Citirea si scrierea in fisier

```
def images(images_dir, image_names):
    images = []
    for element in image_names:
        image = plt.imread(os.path.join(images_dir, element))
        images.append(image.flatten())
    return np.array(images)

def csv(csv_path, column_names, has_labels=True):
    if has_labels:
        column_data = [pd.read_csv(csv_path)[column_name].tolist() for column_name in column_names]
        return column_data
    else:
        image_names = pd.read_csv(csv_path)[column_names[0]].tolist()
        return image_names

train_csv_path = '/kaggle/input/unibuc-dhc-2023/train.csv'
train_images_dir = '/kaggle/input/unibuc-dhc-2023/train_images'
train_image_names, train_labels = csv(train_csv_path, ['Image', 'Class'], has_labels=True)
train_images = images(train_images_dir, train_image_names)
```

```
test_csv_path = '/kaggle/input/unibuc-dhc-2023/test.csv'
test_images_dir = '/kaggle/input/unibuc-dhc-2023/test_images'
test_image_names = csv(test_csv_path, ['Image', 'Class'], has_labels=False)
test_images = images(test_images_dir, test_image_names)

test_images = scaler.transform(test_images)

test_predictions = model.predict(test_images)

test_data = {'Image': test_image_names, 'Class': test_predictions}
pd.DataFrame(test_data).to_csv('test_predictions.csv', index=False)

print("CSV - created.")
```

Pentru citire, am folosit doua functii ajutatoare(`images` si `csv`) pentru a putea citi atat dintr-un director de poze (`train/val/test_images_dir`), cat si pentru a putea citi numele si clasele, unde era cazul, pentru setul de date corespunzator. In plus, functia `csv` detecteaza cu ajutorul parametrului setat implicit **True** daca pentru un fisier dat ca parametru exista label-uri (clasele unde sunt repartizate pozele). Daca acestea exista, citesc din path-ul dat ca parametru coloanele fisierului, altfel citesc numai denumirile imaginilor.

Pentru scrierea in fisier, realizez un dictionar, dupa ce predictiile au fost efectuate, avand ca si chei 'Image' si 'Class', iar ca valori voi pune predictiile aferente. Ulterior, utilizez functia **`pd.DataFrame(test_data)`** pentru a transforma dictionarul facut intr-un obiect `DataFrame`. Apeland metoda **`to_csv('test_predictions.csv', index=False)`** a obiectului `DataFrame`, rezultatul este salvat intr-un fisier CSV. Parametrul **`index=False`** este utilizat pentru a exclude indexul randurilor din fisierul CSV rezultat.

### 2.3 Utilizarea datelor train, val, test si preprocesarea lor

Dupa citirea datelor de train, le-am scalat folosit **`scaler = preprocessing.StandardScaler()`** si **`scaler.fit(train_images)`** astfel incat sa ajustez intervalul si dispersia datelor pentru a putea sa le compar, iar distributia lor sa fie cat mai aproape de cea normala. **`Scaler.fit()`** calculeaza media si deviatia standard a fiecarei caracteristici, urmand ca **`train_images = scaler.transform(train_images)`** sa-mi ajusteze fiecare caracteristica la o scara standard. Vom realiza acest lucru atat pentru val (**`val_images = scaler.transform(val_images)`**), cat si pentru test



(**test\_images = scaler.transform(test\_images)**), deoarece imi doresc ca datele mele, in principiu train si val, sa fie comparabile, pentru ca pe train antrenez modelul, initial, si, apoi, dau predict pe val sa vad cat de bine a a performat invatarea modelul meu.

## 2.4 Model SVM

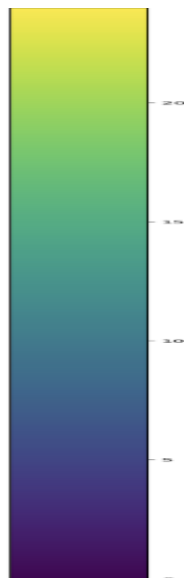
Am incercat sa creez modele SVM cu diferiti parametrii: **svm = SVC(kernel='linear', C=3)**, **svm = SVC(kernel='linear', C=5)**, **svm = SVC(kernel='rbf', C=3)**, **svm = SVC(kernel='rbf', C=5)**, dar, deoarece a rulat cel mai repede (aprox. o ora), am ales-o pe aceasta, intrucat si acuratetea era mai buna decat ceea ce obtinusem anterior! (0,43)

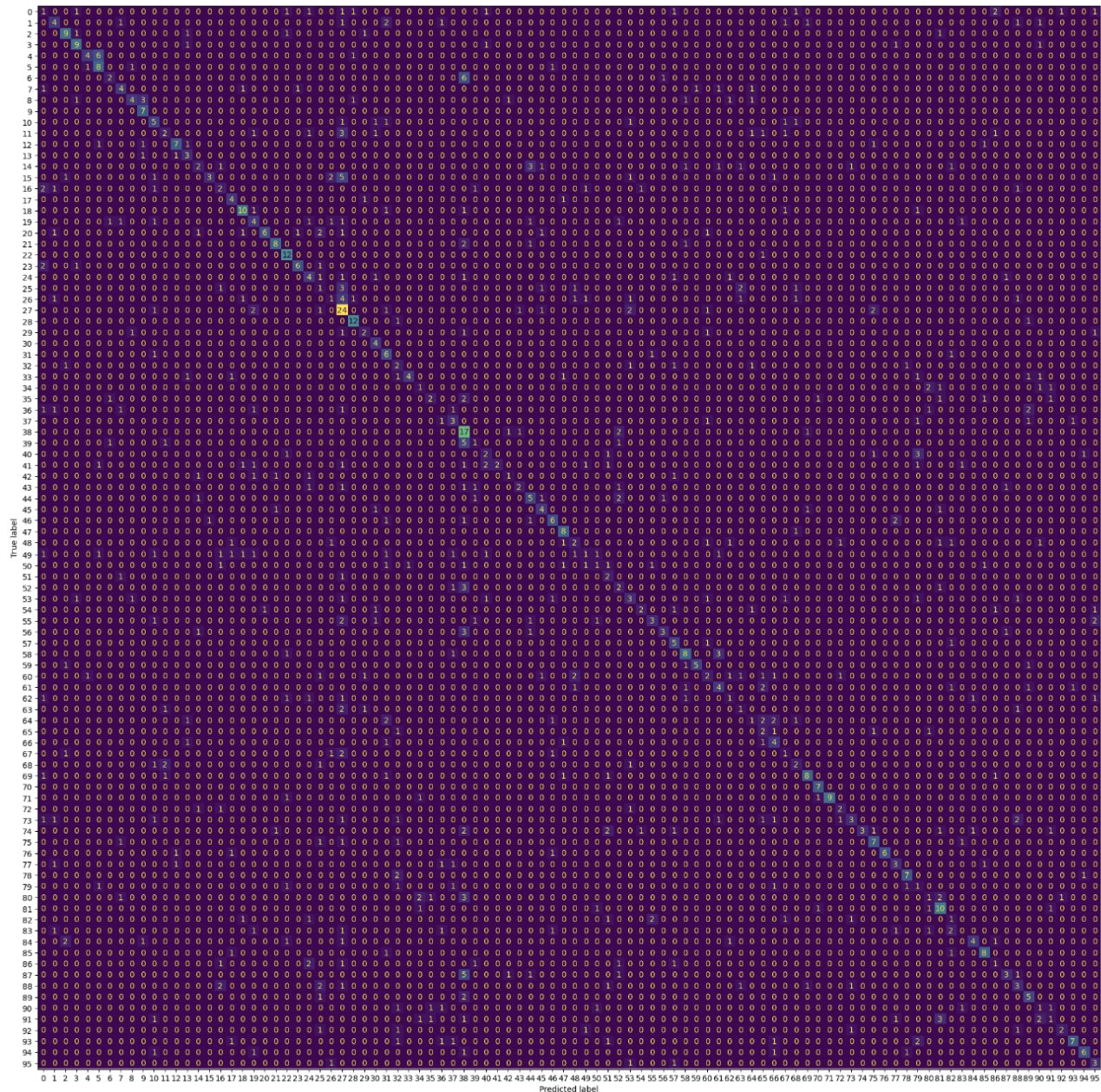
## 2.5 Matricea de confuzie

Matricea de confuzie generata de cod:

```
[[1 0 0 ... 0 0 1]
 [0 4 0 ... 0 0 0]
 [0 0 9 ... 0 0 0]
 ...
 [0 0 0 ... 7 0 0]
 [0 0 0 ... 0 6 0]
 [0 0 0 ... 0 0 3]]
```

Reprezentare mai pe larg a matricii de confuzie:





## 2.6. Concluzii

In ciuda timpului mare de rulare, SVM-ul se dovedeste a fi destul de eficient pentru datele de train, val si test date.

### 3. KNN

KNN (K-nearest neighbors) este un algoritm de invatare supervizata folosit in principal pentru clasificarea si regresia datelor.

#### 3.1 Importarea bibliotecilor

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing
from sklearn.metrics import accuracy_score, confusion_matrix
```

Voi folosi bibliotecile importate pentru a manipula datele, pentru preprocesarea datelor, pentru implementarea modelului KNN si pentru a calcula, ulterior, scorul de acuratete si matricea de confuzie.

#### 3.2 Citirea si scrierea in fisier

```
def images(images_dir, image_names):
    images = []
    for element in image_names:
        image = plt.imread(os.path.join(images_dir, element))
        images.append(image.flatten())
    return np.array(images)

def csv(csv_path, column_names, has_labels=True):
    if has_labels:
        column_data = [pd.read_csv(csv_path)[column_name].tolist() for column_name in column_names]
        return column_data
    else:
        image_names = pd.read_csv(csv_path)[column_names[0]].tolist()
        return image_names

train_csv_path = '/kaggle/input/unibuc-dhc-2023/train.csv'
train_images_dir = '/kaggle/input/unibuc-dhc-2023/train_images'
train_image_names, train_labels = csv(train_csv_path, ['Image', 'Class'], has_labels=True)
train_images = images(train_images_dir, train_image_names)
```

```
test_csv_path = '/kaggle/input/unibuc-dhc-2023/test.csv'
test_images_dir = '/kaggle/input/unibuc-dhc-2023/test_images'
test_image_names = csv(test_csv_path, ['Image', 'Class'], has_labels=False)
test_images = images(test_images_dir, test_image_names)

test_images = scaler.transform(test_images)

test_predictions = model.predict(test_images)

test_data = {'Image': test_image_names, 'Class': test_predictions}
pd.DataFrame(test_data).to_csv('test_predictions.csv', index=False)

print("CSV - created.")
```

Pentru citire, am folosit doua functii ajutatoare(`images` si `csv`) pentru a putea citi atat dintr-un director de poze (`train/val/test_images_dir`), cat si pentru a putea citi numele si clasele, unde era cazul, pentru setul de date corespunzator. In plus, functia `csv` detecteaza cu ajutorul parametrului setat implicit **True** daca pentru un fisier dat ca parametru exista label-uri (clasele unde sunt repartizate pozele). Daca acestea exista, citesc din path-ul dat ca parametru coloanele fisierului, altfel citesc numai denumirile imaginilor.

Pentru scrierea in fisier, realizez un dictionar, dupa ce predictiile au fost efectuate, avand ca si chei 'Image' si 'Class', iar ca valori voi pune predictiile aferente. Ulterior, utilizez functia **`pd.DataFrame(test_data)`** pentru a transforma dictionarul facut intr-un obiect `DataFrame`. Apeland metoda **`to_csv('test_predictions.csv', index=False)`** a obiectului `DataFrame`, rezultatul este salvat intr-un fisier CSV. Parametrul **`index=False`** este utilizat pentru a exclude indexul randurilor din fisierul CSV rezultat.

### 3.3 Utilizarea datelor train, val, test si preprocesarea lor

Dupa citirea datelor de train, le-am scalat folosit **`scaler = preprocessing.StandardScaler()`** si **`scaler.fit(train_images)`** astfel incat sa ajustez intervalul si dispersia datelor pentru a putea sa le compar, iar distributia lor sa fie cat mai aproape de cea normala. **`Scaler.fit()`** calculeaza media si deviatia standard a fiecarei caracteristici, urmand ca **`train_images = scaler.transform(train_images)`** sa-mi ajusteze fiecare caracteristica la o scara standard. Vom realiza acest lucru atat pentru val (**`val_images = scaler.transform(val_images)`**), cat si pentru test

(`test_images = scaler.transform(test_images)`)), deoarece imi doresc ca datele mele, in principiu train si val, sa fie comparabile, pentru ca pe train antrenez modelul, initial, si, apoi, dau predict pe val sa vad cat de bine a a performat invatarea modelul meu.

### 3.4. Model KNN

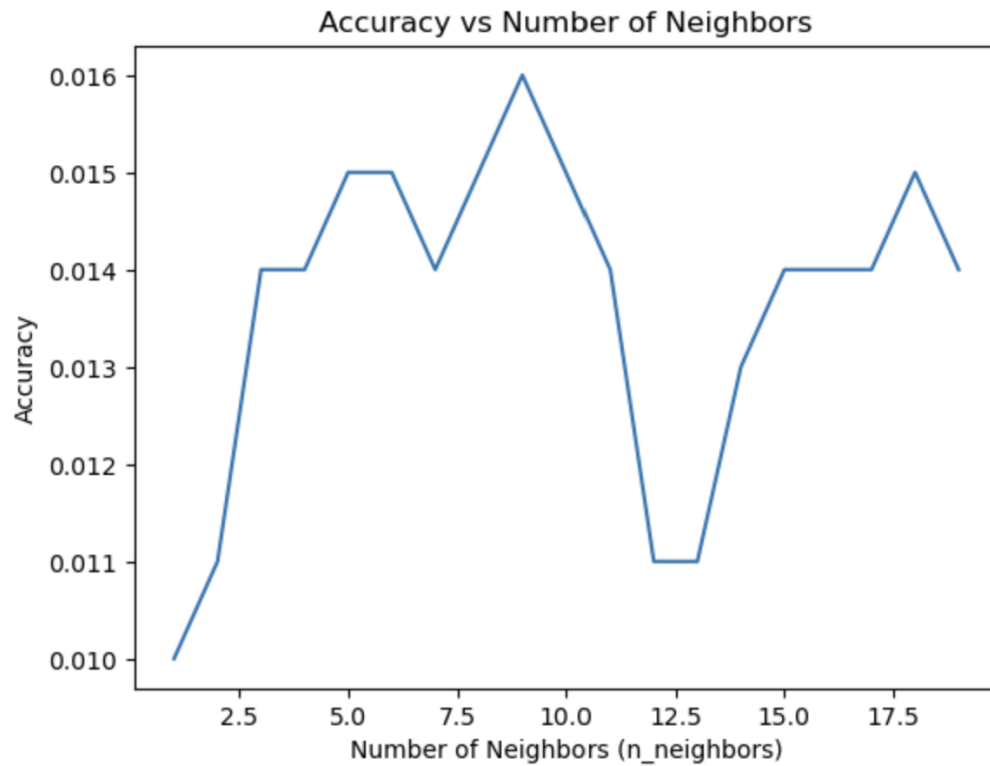
Intrucat nu stiam cu exactitate care este numarul de vecini care ofera cea mai buna acuratete, m-am gandit sa realizez un cod care imi va prelua n-ul de la 1 la 20 cu cea mai buna acuratete pe datele oferite:

```
max_acc=-1
n_max=-1
accuracies = []
for n in range(1,20):
    knn = KNeighborsClassifier(n_neighbors=n)
    knn.fit(train_images, train_labels)

    predictions = knn.predict(val_images)

    accuracy = accuracy_score(val_labels, predictions)
    accuracies.append(accuracy)
    if accuracy > max_acc:
        max_acc=accuracy
        n_max=n
    print(str(n) + ":" + str(max_acc))
```

Astfel, am realizat si un grafic care sa arate evolutia acuratetii pentru diferite valori ale lui n:



Prin urmare, cel mai bun n a fost n=9.

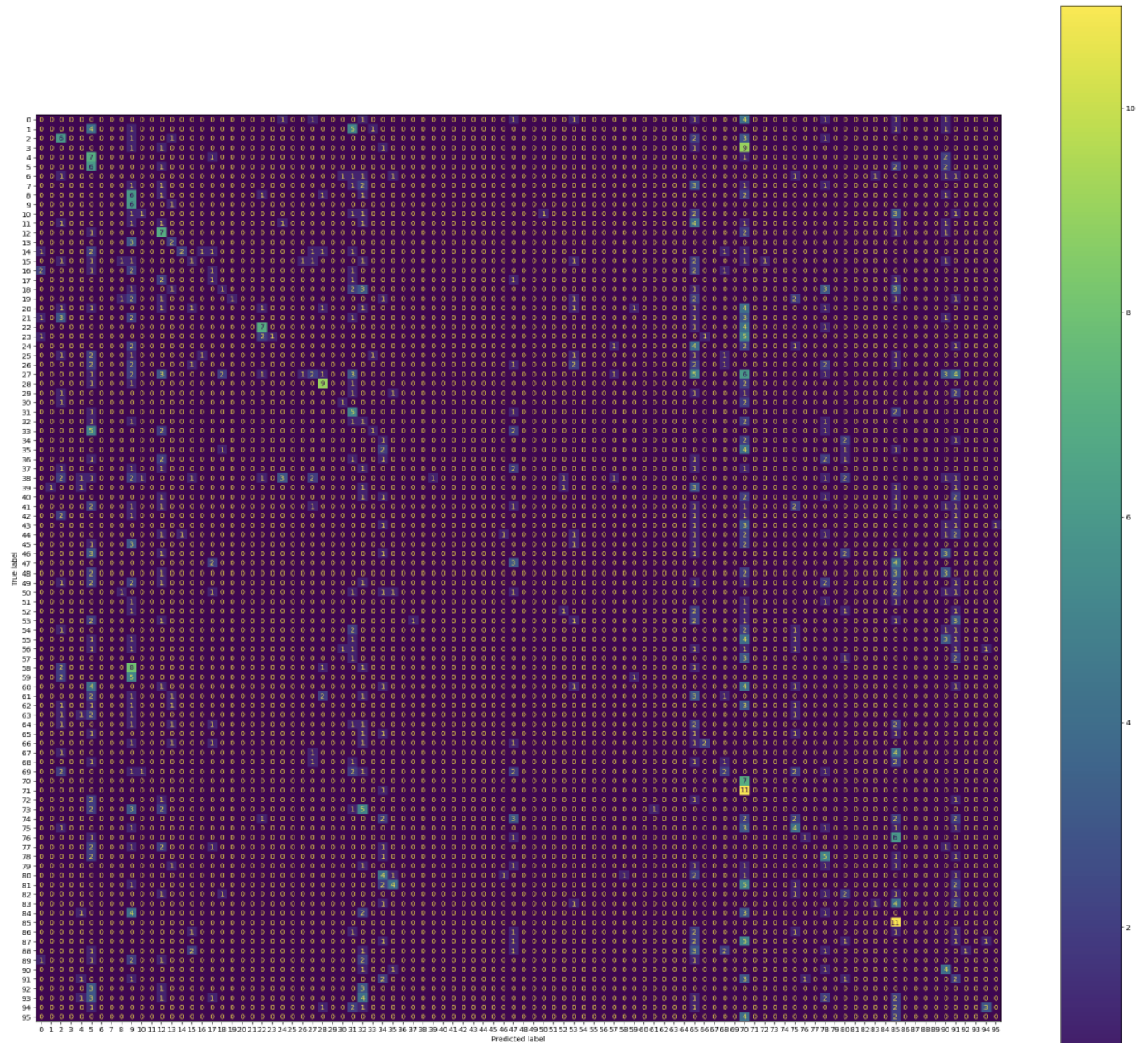
### 3.5. Matricea de confuzie

Matricea de confuzie generata de cod:

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 6 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 0]]
```

Reprezentare mai pe larg a matricii de confuzie:





### 3.6. Concluzii

În ciuda așteptărilor, KNN-ul a avut cea mai mică acuratețe din toate modelele create. (0,12733)

## 4. CNN

CNN (Convolutional Neural Network) este un tip de retea neuronală artificială, conceput pentru a procesa date având structura de grila și este folosit în domeniul viziunii artificiale și a recunoașterii de modele.

### 4.1 Importarea bibliotecilor

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import tensorflow as tf
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from PIL import Image
```

Voi folosi bibliotecile importate pentru a manipula datele, pentru preprocesarea și augumentarea datelor pentru prelucrare, pentru implementarea modelului CNN, crearea unui model secvențial de straturi și crearea straturilor pentru definirea arhitecturii modelului și pentru a calcula, ulterior, scorul de acuratețe și matricea de confuzie.

### 4.2 Scriere și citire în fișier

```
test_data = {'Image': test_image_names, 'Class': test_classes}
pd.DataFrame(test_data).to_csv('test_predictions.csv', index=False)

print("CSV - created.")
```



```
def images(images_dir, image_names, size=(32, 32)):
    images = []
    for element in image_names:
        image = Image.open(os.path.join(images_dir, element))
        images.append(np.array(image.resize(size)) / 255.0)
    return np.array(images)

def csv(csv_path, column_names, has_labels=True):
    if has_labels:
        column_data = [pd.read_csv(csv_path)[column_name].tolist() for column_name in column_names]
        return column_data
    else:
        image_names = pd.read_csv(csv_path)[column_names[0]].tolist()
        return image_names

train_csv_path = '/kaggle/input/unibuc-dhc-2023/train.csv'
train_images_dir = '/kaggle/input/unibuc-dhc-2023/train_images'
train_image_names, train_labels = csv(train_csv_path, ['Image', 'Class'], has_labels=True)
train_labels = to_categorical(train_labels)
train_images = images(train_images_dir, train_image_names)

train_images, val_images, train_labels, val_labels = train_test_split(train_images, train_labels, test_size=0.5, random_state=42)

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
```

Pentru scrierea in fisier, realizez un dictionar, dupa ce predictiile au fost efectuate, avand ca si chei 'Image' si 'Class', iar ca valori voi pune predictiile aferente. Ulterior, utilizeaz functia **pd.DataFrame(test\_data)** pentru a transforma dictionarul facut intr-un obiect DataFrame. Apeland metoda **to\_csv('test\_predictions.csv', index=False)** a obiectului DataFrame, rezultatul este salvat intr-un fisier CSV. Parametrul **index=False** este utilizat pentru a exclude indexul randurilor din fisierul CSV rezultat.

Pentru citire, am folosit doua functii ajutatoare(images si csv) pentru a putea citi atat dintr-un director de poze (train/test\_images\_dir), cat si pentru a putea citi numele si clasele, unde era cazul, pentru setul de date corespunzator. In plus, functia csv detecteaza cu ajutorul parametrului setat implicit **True** daca pentru un fisier dat ca parametru exista label-uri (clasele unde sunt repartizate pozele). Daca acestea exista, citesc din path-ul dat ca parametru coloanele fisierului, altfel citesc numai denumirile imaginilor. In plus, fata de celelalte citiri, am transformat train\_labels in **train\_labels = to\_categorical(train\_labels)**, deoarece, avand un model deep de inavatare, este necesara transformarea etichetelor intr-un format "one-hot encoding" (transforma fiecare categorie intr-un vector binar in care doar o pozitie are 1, restul 0). Ulterior, am impartit datele de antrenament intr-un set de antrenament si unul de validare, impartind random setul de train (atat images, cat si labels). **test\_size=0.5** asigura faptul ca jumatate din date vor fi folosite pentru validare, iar **random\_state=42** asigura impartirea datelor la fiecare executie a programului. In







plus, crearea unui obiect de tipul **ImageDataGenerator** imi asigura augmentarea datelor, astfel incat, prin aplicarea acestor transformari, sa pot imbunatati performanta modelului. Parametrii folositi ajuta la diversificarea imaginilor si, de asemenea, pentru obtinerea unui set de antrenament mai variat.

### 4.3 Model CNN

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    BatchNormalization(),
    Flatten(),
    Dense(256, activation='relu'),
    Dense(96, activation='softmax')
])

model.summary()
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
datagen.fit(train_images)
```

Voi explica fiecare strat al modelului dupa cum urmeaza:

-  **Conv2D( \_, (3, 3), activation='relu', input\_shape=(32, 32, 3) ) =>** avem un prim strat de 32/64/128 de filtre cu activarea 'relu'. Aceasta functie este necesara pentru a se extrage astfel caracteristici importante dintr-o imagine
-  **BatchNormalozation() =>** normalizez activarile stratului anterior pentru accelerarea antrenarii modelului
-  **MaxPooling2D((2, 2)) =>** reduce dimensiunea imaginilor, deoarece selecteaza valoarea maxima de dimensiunea data
-  **Flatten() =>** transforma datele adunate dintr-o matrice intr-un vector unidimensional si le pregateste, astfel, pentru conectarea la straturile fully connected
-  **Dense(256, activation='relu') =>** strat complet conectat care are 256 neuroni
-  **Dense(96, activation='softmax') =>** strat complet conectat care are 256 neuroni, avand functia de activare softmax pentru distribuirea probabilitatilor pentru fiecare clasa

Dupa crearea modelului, am realiza un sumar al arhitecturii sale:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
batch_normalization (Batch Normalization)	(None, 30, 30, 32)	128
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
batch_normalization_1 (Batch Normalization)	(None, 13, 13, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 128)	73856
batch_normalization_2 (Batch Normalization)	(None, 4, 4, 128)	512
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 256)	524544
dense_1 (Dense)	(None, 96)	24672
Total params: 643,360		
Trainable params: 642,912		
Non-trainable params: 448		

Ulterior, compilez modelul definit anterior cu parametrii: **optimizer='adam'** – optimizator pentru gradient, **loss='categorical\_crossentropy'** – functia de pierdere care masoara diferenta dintre distributia claselor prezise de model si distributia claselor reale si **metrics=['accuracy']** – masoara proportia corecta de clasificari.

In cele din urma, dau fit pe train\_images.

#### 4.4 Rechemarea modelului

Intrucat am observat ca pe parcursul rularii unui numar mici/mari de epoci acuratetea se modifica constanta, de la valori cat mai mici pana la valori cat mai mari, am decis sa salvez modelul cu cea mai buna acuratete obtinuta din setul de epoci rulat.

```
checkpoint = tf.keras.callbacks.ModelCheckpoint('best.csv', monitor='val_accuracy', save_best_only=True, mode='max')

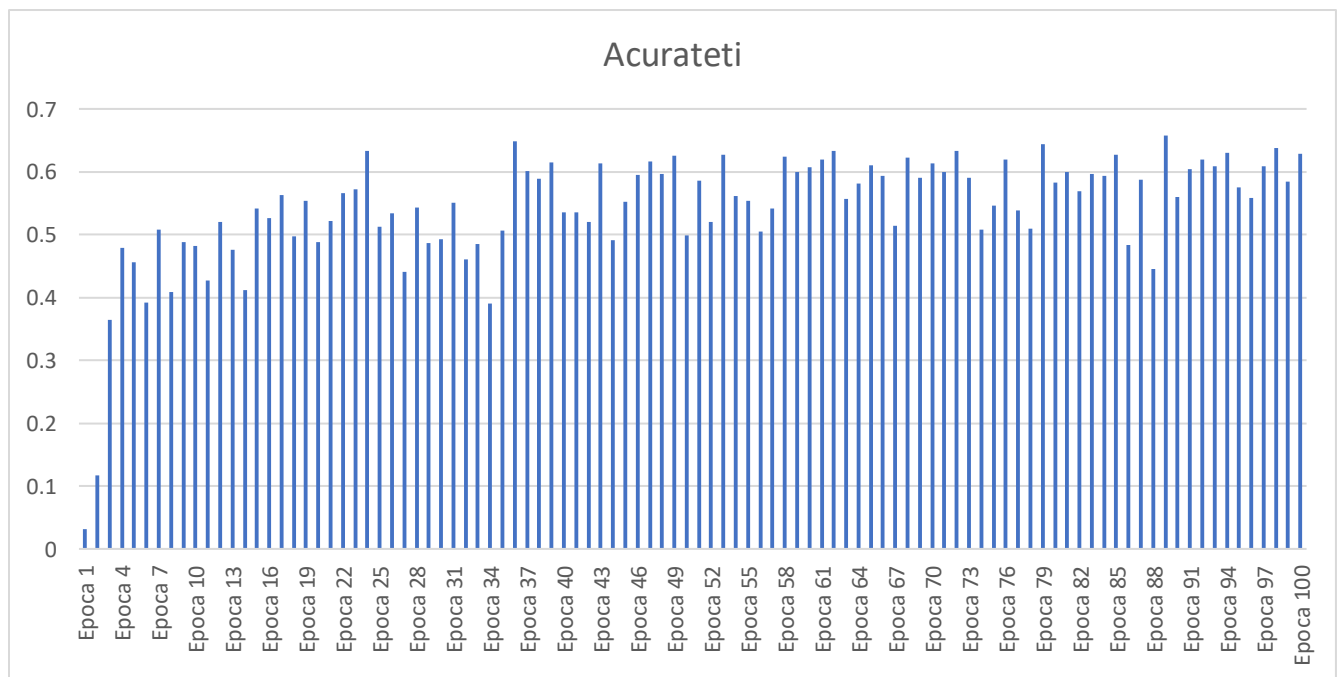
history = model.fit(
    datagen.flow(train_images, train_labels, batch_size=32),
    epochs=100,
    validation_data=(val_images, val_labels),
    callbacks=[checkpoint]
)

best = tf.keras.models.load_model('best.csv')
```

Astfel, voi salva in checkpoint cel mai bun model din punct de vedere al acuratetii. Dau fit cu parametrii: **datagen.flow(train\_images, train\_labels, batch\_size=32)** – furnizeaza catre generator setul de antrenament in calupuri de 32 de exemple / epoca, **epochs=100** – numarul epocilor (treceri complete prin train), **validation\_data = (val\_images, val\_labels)** – datele de validare utilizate de model si **callbacks=[checkpoint]** – lista de callback-uri realizate. In final, salvez cel mai bun model in fisierul ‘best.csv’.

#### 4.5. Variatii ale acuratetii de la o epoca la alta

Pentru a realiza un chart care sa ilustreze cum variaza acuratetea pentru CNN, am rulat modelul pe 100 de epoci. (acuratetea cea mai mare a fost obtinuta pe rularea a 300 de epoci).



#### 4.6. Concluzii

In concluzie, am ales aceasta acuratete ca fiind cea mai buna! (0,756)