

Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Ingineria Software și Automatică

**RAPORT**

**Lucrarea de laborator nr.5**

**la Inteligența Artificială**

*Tema: Aplicații ale rețelelor neuronale artificiale.*

Grupa academică: TI-214  
A efectuat: Reguș Ruslan

**A verificat: Mariana Rusu**

Chișinău 2024

O rețea neuronală artificială este un model matematic inspirat din sistemul nervos biologic, utilizat pentru a rezolva probleme complexe de învățare automată și inteligență artificială. Aceasta este compusă dintr-o serie de straturi de neuroni interconectați, fiecare fiind o unitate de bază de procesare a informațiilor.

**Componentele unei rețele neuronale artificiale:**

* Straturi de neuroni (input, hidden, output);
* Funcții de activare (sigmoid, relu, tanh etc.);
* Funcții de cost (pentru evaluarea performanței modelului);
* Optimizatori (gradient descent, Adam, RMSprop etc.).

import pandas as pd

import numpy as np

import os

os.environ['TF\_ENABLE\_ONEDNN\_OPTS'] = '0'

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix

import keras

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D

from keras.layers import Dense, Flatten

from keras.layers import BatchNormalization

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.utils import to\_categorical

**Importarea tuturor bibliotecilor necesare**

# Citirea seturilor de date

test = pd.read\_csv('C:/Users/schio/OneDrive/Desktop/Anul 3/sem 2/IA/lab 5/test.csv')

train = pd.read\_csv('C:/Users/schio/OneDrive/Desktop/Anul 3/sem 2/IA/lab 5/train.csv')

print(f"Training data size is {train.shape}\nTesting data size is {test.shape}")

# Setarea datelor de antrenare

X = train.drop(columns=['label']).values

y = train['label'].values

# Normalizarea datelor

X = X / 255.0

# Reshape la dimensiunile 28px/28px și 1 canal

X = X.reshape(-1, 28, 28, 1)

# Encoding la labeluri

y = to\_categorical(y)

print(f"Label size {y.shape}")

**Importarea csv-urilor și procesarea datelor**

În această secțiune de cod este prezentată importarea datelor din 3 csv-uri, setarea mulțimilor de date x și y, normalizarea datelor pentru scăderea scării acestora, reshape la dimensiunile 28\*28 px precum și encoding la label-uri în binar.

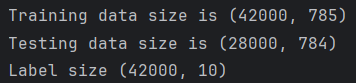


Figura 1 – Afișarea dimensiunii seturilor de date

# Afișarea a 4 imagini din setul de date

X\_train\_\_ = X\_train.reshape(X\_train.shape[0], 28, 28)

fig, axis = plt.subplots(1, 4, figsize=(20, 10))

for i, ax in enumerate(axis.flat):

ax.imshow(X\_train\_\_[i], cmap='binary')

digit = y\_train[i].argmax()

ax.set(title=f"Real Number is {digit}")

plt.show()

# Calcularea mediei și a deviației standard

mean = np.mean(X\_train)

std = np.std(X\_train)

**Afișarea unor imagini din setul de date de antrenament**

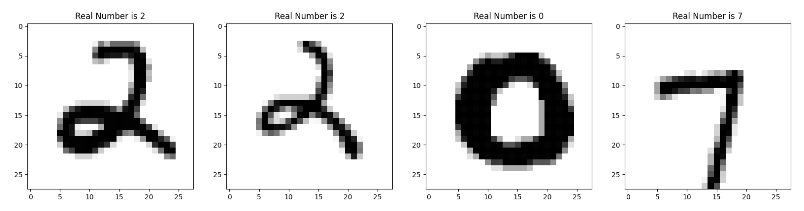
****

Figura 2 – Imaginile afișate

Observăm că inițial imaginile din setul de date conțin etichete referitor la valoarea reală a cifrei ce este reprezentată în ea.

# Crearea modelului

model = Sequential()

# Crearea straturilor convoluționale (pentru extragerea caracteristicilor)

model.add(Conv2D(filters=64, kernel\_size=(3, 3), activation="relu", input\_shape=(28, 28, 1)))

model.add(Conv2D(filters=64, kernel\_size=(3, 3), activation="relu"))

# Crearea altui strat pentru reducerea dimensiunii spațiului de ieșire

model.add(MaxPooling2D(pool\_size=(2, 2)))

# Crearea altui strat pentru normalizarea datelor și accelerarea antrenării

model.add(BatchNormalization())

model.add(Conv2D(filters=128, kernel\_size=(3, 3), activation="relu"))

model.add(Conv2D(filters=128, kernel\_size=(3, 3), activation="relu"))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(BatchNormalization())

model.add(Conv2D(filters=256, kernel\_size=(3, 3), activation="relu"))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(BatchNormalization())

# Crearea stratului de flatten pentru a transforma datele in 1D

model.add(Flatten())

# Crearea straturilor dense pentru procesarea datelor prin neuroni

model.add(Dense(512, activation="relu"))

model.add(Dense(10, activation="softmax"))

# Compilarea modelului

model.compile(loss="categorical\_crossentropy", optimizer="adam", metrics=["accuracy"])

# Afișarea arhitecturii modelului

model.summary()

**Crearea modelului**

În porțiunea dată de script are loc crearea modelului pentru rețeaua noastră neuronală. Acest model conține:

* straturi convoluționale **Conv2d**, pentru extragerea caracteristicilor din imagine;
* straturi **MaxPooling2d** pentru reducerea dimensiunea spațiului de ieșire;
* aplicarea funcției **BatchNormalization()** pentru normalizarea datelor și accelerarea antrenării;
* funcția **Flatten()** pentru transformarea datelor într-un array unidimensional de date;
* straturile dense pentru procesarea datelor cu un anumit număr de neuroni;
* apelarea pentru compilarea acestuia cu parametri specifici.

**Noțiuni:**

* **‘loss=”categorical\_crossentropy”’ –** funcție de pierdere utilizată în timpul antrenării în cazul utilizării clasificării multi-clasă;
* **‘optimizer=”adam”’ –** algoritmul de optimizare utilizat pentru minimizarea funcției de pierdere în timpul antrenării;
* **‘metrics=[”accuracy”]’-** acuratețea sau precizia, metrica ce va fi afișată în timpul antrenării modelului.

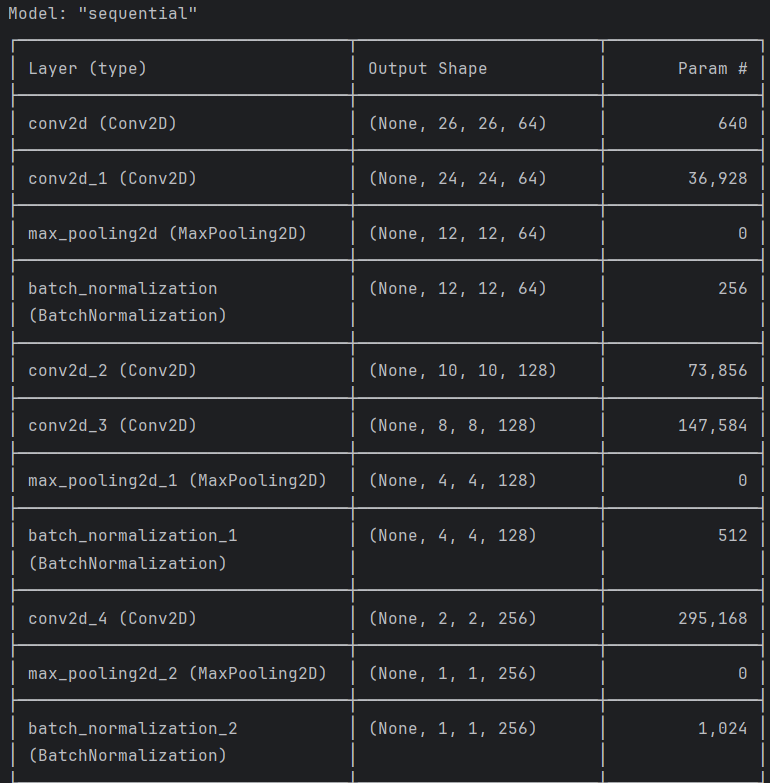


Figura 3 – Structura modelului (partea 1)

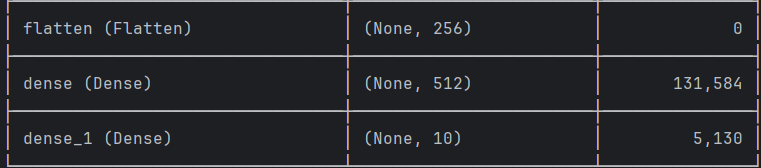


Figura 4 – Structura modelului (partea 2)

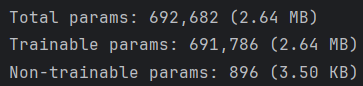


Figura 5 – Clasificarea parametrilor

# Generarea imaginilor

datagen = ImageDataGenerator(

# Nu centrează datele pe medie

featurewise\_center=False,

# Nu centrează fiecare eșantion individual pe medie

samplewise\_center=False,

# Nu normalizează datele pe deviația standard

featurewise\_std\_normalization=False,

# Nu normalizează fiecare eșantion individual pe deviația standard

samplewise\_std\_normalization=False,

# Nu aplica augmentare de date

zca\_whitening=False,

# Intervalul în grade pentru rotația aleatorie a imaginilor

rotation\_range=15,

# Intervalul de scalare aleatorie a imaginilor

zoom\_range=0.01,

# Intervalul de deplasare aleatorie a imaginilor pe orizontală si verticală

width\_shift\_range=0.1,

height\_shift\_range=0.1,

# Nu se face flip(inversarea) pe orizontală si verticală

horizontal\_flip=False,

vertical\_flip=False)

# Antrenarea modelului cu setul de date respectiv

datagen.fit(X\_train)

# Generarea de imagini pentru antrenare și testare

train\_gen = datagen.flow(X\_train, y\_train, batch\_size=128)

test\_gen = datagen.flow(X\_test, y\_test, batch\_size=128)

# Setarea parametrilor pentru antrenare

epochs = 10

batch\_size = 128

# Calcularea pașilor pentru antrenare și testare

train\_steps = X\_train.shape[0] // batch\_size

valid\_steps = X\_test.shape[0] // batch\_size

# Oprirea antrenării când val\_accuracy nu crește

es = keras.callbacks.EarlyStopping(

# Metrica de monitorizare

monitor="val\_accuracy",

# Numărul de epoci consecutive în care val\_accuracy nu crește

patience=10,

# Se va afișa un mesaj la finalul antrenării

verbose=1,

# Se oprește când metrica atinge maximum

mode="max",

# Se restaurează greutățile modelului la cea mai bună valoare

restore\_best\_weights=True)

# Reducerea ratei de învățare când val\_accuracy nu crește

rp = keras.callbacks.ReduceLROnPlateau(

# Metrica de monitorizare

monitor="val\_accuracy",

# Factorul cu care se reduce rata de învățare

factor=0.2,

# Numărul de epoci consecutive în care val\_accuracy nu crește

patience=3,

# Se va afișa un mesaj la finalul antrenării

verbose=1,

# Se oprește când metrica atinge maximum

mode="max",

# Rata de învățare minimă

min\_lr=0.00001)

# Antrenarea modelului

history = model.fit(train\_gen,

# Numărul de epoci

epochs=epochs,

# Numărul de pași per epocă

steps\_per\_epoch=train\_steps,

# Setul de validare

validation\_data=test\_gen,

# Numărul de pași per epocă pentru validare

validation\_steps=valid\_steps,

# Funcții de apelare (es-EarlyStopping, rp-ReduceLROnPlateau)

callbacks=[es, rp])

**Generarea imaginilor, completarea modelului**

**cu funcții necesare și antrenarea sa**

În această secțiune de cod observăm scriptul pentru generarea imaginilor cu toți parametrii explicați mai sus. De asemenea observăm 2 funcții adiacente ce cresc flexibilitatea codului, și anume în cazul în care timp de mai multe epoci (iterații) nu crește acuratețea, atunci antrenarea se stopează, iar dacă mai multe epoci consecutive acuratețea nu crește, rata de învățare se reduce pentru finisarea antrenării modelului cu o rată de învățare minimă.

Rata de învățare (learning rate) este o mărime scalară ce indică cât de repede se vor ajusta ponderile (parametrii) modelului în timpul procesului de antrenare. Aceasta determină mărimea pașilor pe care optimizatorul îi face în spațiul parametrilor modelului pentru a găsi minimele locale ale funcției de cost.

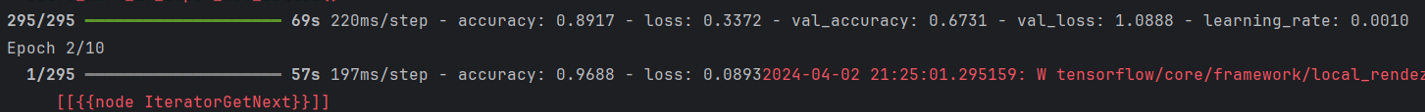


Figura 6 – Antrenarea modelului

Așa decurge antrenarea modelului pe un set de date. În cazul meu putem vizualiza multiple preîntâmpinări din cauza părăsirii intervalului de valori în cadrul unei epoce. În așa caz, RNA trece îndată la procesarea datelor din cadrul următorii epoce.



Figura 7 – Finisarea antrenării modelului

Procesul de antrenare a modelului se finisează după o perioadă de timp și i se restabilește starea la epoca cu cele mai bune caracteristici, întrucât pe parcursul a câteva epoce, acuratețea și-a atins maximul său.

# Calcularea matricii de confuzie

y\_pred = model.predict(X\_test)

y\_pred\_labels = np.argmax(y\_pred, axis=1)

conf\_mat = confusion\_matrix(np.argmax(y\_test, axis=1), y\_pred\_labels)

**Calcularea matricii de confuzie**

# Trasarea graficelor pentru loss și accuracy

fig, ax = plt.subplots(2, 1, figsize=(18, 10))

ax[0].plot(history.history['loss'], color='b', label="Training loss")

ax[0].plot(history.history['val\_loss'], color='r', label="Validation loss")

legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy")

ax[1].plot(history.history['val\_accuracy'], color='r', label="Validation accuracy")

legend = ax[1].legend(loc='best', shadow=True)

plt.show()

**Trasarea graficelor pentru pierderi și acuratețe**

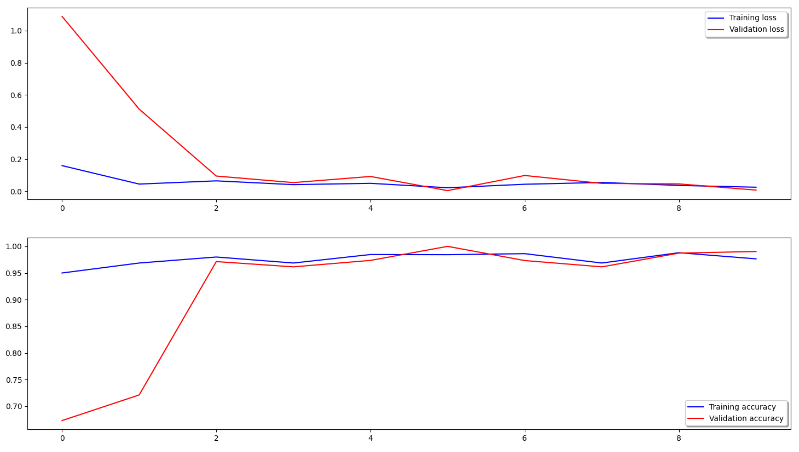
****

Figura 8 – Graficele referitor la pierderi și acuratețe

Analizând aceste grafice putem observa că pe parcursul antrenării modelului pierderile de antrenare și de validare scad constant până la valori minime, iar acuratețea de antrenament și de validare cresc până la valorile maxime ale sale.

# Trasarea matricii de confuzie

fig = plt.figure(figsize=(10, 10))

sns.heatmap(conf\_mat.T, square=True, annot=True, cbar=False, cmap=plt.cm.Blues, fmt='.0f')

plt.xlabel('Predicted Values')

plt.ylabel('True Values')

plt.show()

**Trasarea matricii de confuzie**

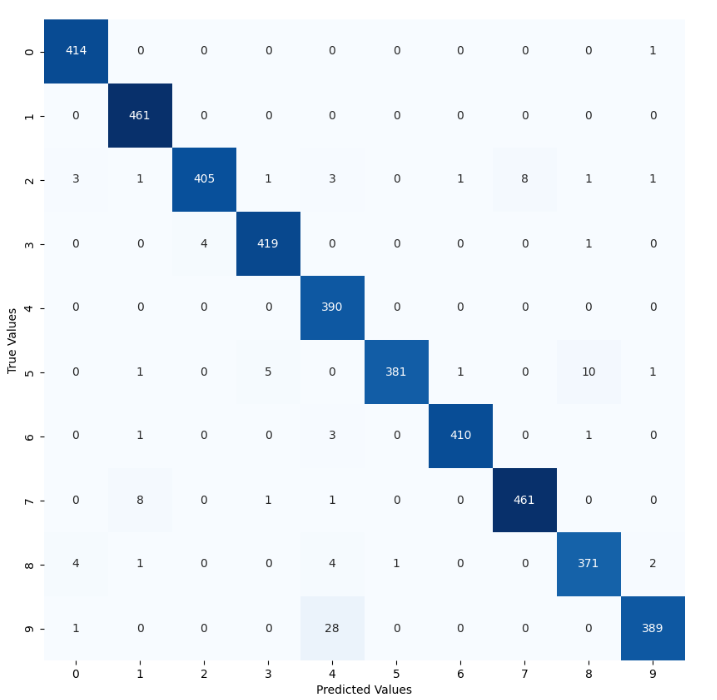
****

Figura 9 – Matricea de confuzie

# Trasarea a 16 imagini din setul de date și de testare

fig, axis = plt.subplots(4, 4, figsize=(12, 14))

for i, ax in enumerate(axis.flat):

ax.imshow(X\_test[i], cmap='binary')

ax.set(title=f"Real Number is {y\_test[i].argmax()}\nPredict Number is {y\_pred[i].argmax()}")

plt.show()

**Testarea a 16 imagini**

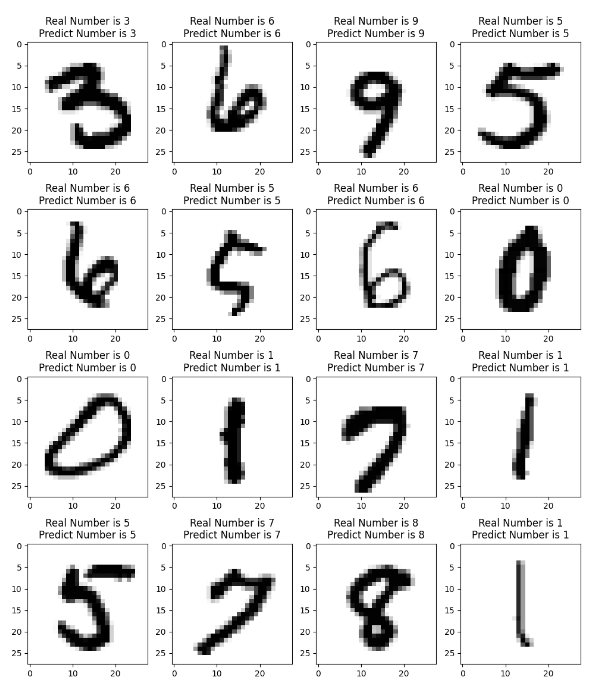
****

Figura 10 – Imaginile testate

Analizând acest grafic poate fi observat faptul că au fost preluate pentru analiză 16 imagini, cu etichetele sale reale deja existente și cele prezise de către model. Deci, datorită acurateței înalte, absolut 100% din imagini au fost procesate corect, iar etichetele au fost prezise cu succes.

**Concluzie**

În cadrul acestei lucrări de laborator, am efectuat o analiză a cifrelor scrise de mână folosind un model de rețea neurală artificială, implementat cu ajutorul framework-ului Keras și bibliotecii TensorFlow. Inițial, am prelucrat datele de antrenare și de testare din setul de date disponibil, aplicând diverse tehnici precum normalizarea datelor și augmentarea acestora pentru a îmbunătăți performanța modelului. Apoi, am definit arhitectura rețelei neurale, incluzând atât straturi convoluționale, cât și dense, pentru a captura caracteristicile semnificative ale cifrelor scrise de mână.

Următorul pas a fost antrenarea modelului utilizând datele pregătite anterior și optimizarea acestuia prin ajustarea ratei de învățare. Am monitorizat performanța modelului pe setul de validare pentru a evita overfitting-ul și pentru a asigura o generalizare cât mai bună. Pentru evaluarea performanței, am utilizat diverse metrici și am generat o matrice de confuzie pentru a analiza capacitatea modelului de a clasifica corect cifrele scrise de mână.

Am realizat o analiză vizuală a rezultatelor prin intermediul graficelor pentru loss și accuracy, urmărind evoluția antrenării și validării modelului pe parcursul epocilor. Această analiză ne-a furnizat informații importante despre performanța și generalizarea modelului în procesul de clasificare a cifrelor scrise de mână.