Documentație Proiect IA

Nume: **Buhai Darius**

Grupă: **234.2**

Descrierea abordărilor:

Pentru dataset-ul primit, am folosit 2 tipuri de modele:

- un model clasic k-NN (k-nearest neighbors) cu o acuratețe maximă de ***0.51***.
- un model mai avansat de tip CNN (Convolutional Neuronal Network) cu o acuratețe de ***0.82***.

Încărcarea datelor

Datele de antrenare sunt încărcate în clasa Data ce are 2 metode principale (**load** și **dump**). Imaginile sunt încarcate în 2 formate, (50 x 50 x 1) pentru modelul **CNN**, respectiv (2500 x 1) pentru modelul **k-NN**

```
@staticmethod
def load(input_file='train'):
    images_arr, images_mat, labels = [], [], []
    with open(f"data/{input_file}.txt", "r") as f:
         line = f.readline()
         while line:
             line_data = line.replace("\n", "").split(",")
             line = f.readline()

             image_path = f"data/{input_file}/{line_data[0]}"
             if not path.exists(image_path):
                continue

             img_arr = imageio.imread(image_path)
             img_arr = np.asarray(img_arr).reshape(-1)

             img_mat = image.load_img(image_path, target_size=(50, 50, 1), color_mode='grayscale')
             img_mat = image.img_to_array(img_mat)
             img_mat = img_mat / 255

             images_arr.append(img_arr)
             images_mat.append(img_mat)

             if len(line_data) > 1:
                labels.append(int(line_data[1]))
     return images_mat, images_arr, labels
```

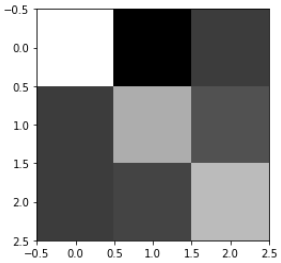
Abordarea k-NN

Modelul k-nearest neighbors caută cei mai apropiați k vecini pentru exemplul de test în train data, returnând eticheta majoritară.

Pentru modelul nostru am setat k=3 si metric='l2' și am obtinut o acuratețe maximă de **0.52**.

Matrice de confuzie

914	208	378
378	686	436
376	398	726



Abordarea CNN

Explicație (Ce este un CNN?)

Convolutional Neural Network este un algoritm din ML ce poate categoriza imagini primite ca date de intrare. Arhitecura acestui algoritm a fost inspirată din rețeaua neuronală umană.

Mai precis, algoritmul **CNN** reprezintă o rețea de perceptroni multilayer, în care neutronii dintr-un layer sunt conecțati complet cu neutronii din următorul layer. Pentru a se evita overfitting-ul, modelului **CNN** îi sunt aplicate anumite regularizări specifice, precum dropout-ul (*ce modifică valorile nenule cu 1/(1 - rate), fără a modifica suma finală a input-urilor*).

Implementare

În implementarea algoritmului **CNN** am folosit librăria **keras** din tenserflow și urmatoarele module:

```
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, Flatten, Conv2D, BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import LearningRateScheduler, ModelCheckpoint
```

Modelul l-am structurat după cum urmează:

- 6 layere 2D (3 x 32, 3 x 64) cu o funcție de activare de tip relu, din care primul layer primește forma inițială a imaginilor (50 x 50 x 1).
- Dupa fiecare layer 2D, am adăugat un layer de normalizare (Batch normalization) ce menține valoarea medie a output-ului spre 0.
- Pentru a evita overfitting-ul, am adăugat 2 Dropout-uri
- Un layer Flatten ce transformă forma input-ului primit din (10, 10, 64) în (6400)
- Un layer Dense de forma (128), urmat de un layer de normalizare
- Un dropout urmat de un ultim layer Dense (10).

```
self.model = Sequential()

self.model.add(Conv2D(32, kernel_size=3, activation='relu', input_shape=(50, 50, 1)))
self.model.add(BatchNormalization())
self.model.add(Conv2D(32, kernel_size=3, activation='relu'))
self.model.add(BatchNormalization())
self.model.add(Conv2D(32, kernel_size=5, padding='same', strides=2, activation='relu'))
self.model.add(BatchNormalization())
self.model.add(Dropout(dropout))

self.model.add(Conv2D(64, kernel_size=3, activation='relu'))
self.model.add(BatchNormalization())
self.model.add(Conv2D(64, kernel_size=3, activation='relu'))
self.model.add(BatchNormalization())
self.model.add(Conv2D(64, kernel_size=5, padding='same', strides=2, activation='relu'))
self.model.add(BatchNormalization())
self.model.add(Dropout(dropout))

self.model.add(Flatten())
self.model.add(Dense(128, activation='relu'))
self.model.add(BatchNormalization())
self.model.add(Dropout(dropout))
self.model.add(Dense(10, activation='softmax'))
```

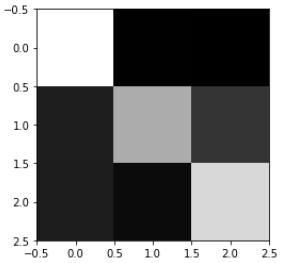
Preprocesare

Pe partea de preprocesare, am expandad datele de antrenare folosind ImageDataGenerator din keras.preprocessing, cu batch_size=64.

```
self.datagen = ImageDataGenerator(rotation_range=10, zoom_range=0.10, width_shift_range=0.1, height_shift_range=0.1)
[...]
self.datagen.flow(train_images, train_labels, batch_size=64)
```

Matrice de confuzie

1453	27	20
192	986	322
184	83	1233

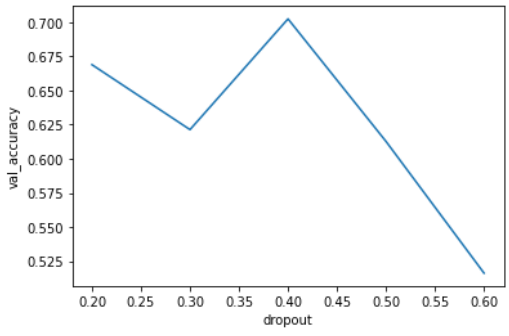


Dropout

Pentru a alege cel mai bun dropout pentru model, am testat acuratețea algoritmului pe câte 10 epoci, iar în tabelul de mai jos putem observa cum se comporta CNN-ul nostru în funcție de dropout-ul setat:

Dropout	val_loss	val_acccuracy
0.2	1.1538	0.6689
0.3	1.2546	0.62133
0.4	0.7208	0.70244
0.5	1.1618	0.61267
0.6	2.2282	0.51622

Grafic:



Îmbunătățiri

Pentru început, după ce am stabilit modelul de antrenare, am pornit cu 20 de epoci pe care am obtinut un scor maxim de **0.71**. În continuare, am antrenat modelul pe 100 de epoci, iar mai apoi pe 240, obținând un scor de **0.778**.

Dupa ce am rulat pe mai mult de 240 de epoci am observat că acuratețea testelor de validare scade (cu toate că acuratețea algoritmului crește), așa că am decis să salvez epocile cele mai bune, folosind funcția de callback **model_checkpoint** din keras.callbacks.

```
self.model_checkpoint = ModelCheckpoint(f'data/[...]', monitor='val_accuracy', mode='max', save_best_only=True)
[...]
self.model.fit([...], callbacks=[self.model_checkpoint])
```

Astfel rulând algoritmul pe câte >16 epoci, mi-am creat un model deja antrenat cu cea mai buna epoca din fiecare iterație. Pentru scorul de**0.81743** am rulat peste 20 de iterații de antrenare a modelului plecând de la ultimul model antrenat (din cea mai buna epoca).

```
for i in range(20):
    # Incarcam ultimul model antrenat
    cnn_classifier.load_best()
    # Continuam antrenarea
    cnn_classifier.train(train_images_c, train_labels_c, validation_images_c, validation_labels_c, epochs=(16 + i))

    # Testam acuratetea
    acc = test_accuracy(cnn_classifier, validation_images_c, validation_labels_c)

    # Salvam doar modelele cu acurateti mai bune
    if acc > best_accuracy:
       best_accuracy = acc
       predicted_labels = cnn_classifier.classify_images(test_images_c)
       Data.dump(predicted_labels, input_file="test", output_file="cnn")
```

Astfel, vom urmări doar epocile ce cresc acuratețea testelor de validare, fără a face overfitting.