Deep Hallucination Classification

Gherghescu Andreea 243

Introducere

‘Deep Hallucination Classification’ este un concurs din etapa de Machine Learning a cursului de Inteligență Artificială în cadrul Facultății de Matematică și Informatică. Studenții au avut ca scop principal să găsească metode cât mai eficiente pentru a clasifica imagini în 7 categorii diferite, acestea fiind catalogate de la 0 până la 6. Imaginile erau colore (RGB) și de mărime 16x16 pixeli. Antrenarea modelelor s-a realizat pe setul de antrenare, unde am avut la dispoziție 8000 de imagini, validarea pe setul de 1173 de imagini, iar, pentru testare, 2819 de imagini.

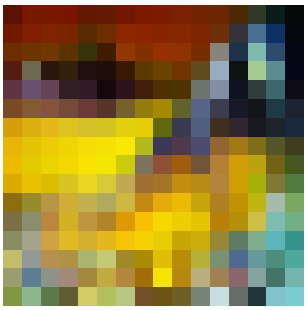
Clasificarea imaginilor

Procesul de clasificare al imaginilor acoperă o sferă foarte vastă a domeniului numit ‘Artificial Intelligence’, reprezentând unul dintre cele mai importante concepte ce modelează lumea în care trăim. Din acest motiv, trebuie să fim atenți la metodele pe care încercăm să le implementăm în acest domeniu.

Datele dispuse

Imaginile dispuse acestui concurs sunt de tipul RGB și de mărime 16x16px.

Imagine de tip 0



Imagine de tip 6

Imagine de tip 4

Citirea Imaginilor

Citirea imaginilor a fost realizată utilizând modulul ‘open-cv’ astfel încât aceasta să fie realizată cât mai rapid și mai eficient. De asemenea am folosit modulul ‘pickle’ pentru a salva imaginile, odată citite, într-un format mai rapid pentru a evita citirea lor de la 0 pentru fiecare iterație a programului.

```
# Read train images and labels
with open("./data/train.txt", 'r') as file:
    file.readline()
    for line in file.readlines():
        img = cv.imread(TRAIN_IMAGES_PATH + line.split(',')[0])
        img = (img - np.mean(img)) / np.std(img)
        train_load_images.append(img)
        train_load_labels.append(int(line.split(',')[1]))

# Read validation images and labels
with open("./data/validation.txt", 'r') as file:
    file.readline()
    for line in file.readlines():
        img = cv.imread(VALIDATION_IMAGES_PATH + line.split(',')[0])
        img = (img - np.mean(img)) / np.std(img)
        validation_load_images.append(img)
        validation_load_labels.append(int(line.split(',')[1]))
```

Citirea imaginilor (plus preprocesare)

```
# Save loaded data to pickle
if SAVE_PICKLE:
    with open("./data/pickle/train_images", 'wb') as pickle_save_file:
        pickle.dump(train_load_images, pickle_save_file)

    with open("./data/pickle/validation_images", 'wb') as pickle_save_file:
        pickle.dump(validation_load_images, pickle_save_file)

    with open("./data/pickle/train_labels", 'wb') as pickle_save_file:
        pickle.dump(train_load_labels, pickle_save_file)

    with open("./data/pickle/validation_labels", 'wb') as pickle_save_file:
        pickle.dump(validation_load_labels, pickle_save_file)
```

Utilizarea modulului pickle

Preprocesarea Imaginilor

De asemenea, pentru a avea o standardizare și o evidențiere cât mai bună a trăsăturilor specifice imaginilor am folosit (după cum se poate vedea în a 2-a imagine de mai sus) una dintre cele mai clasice metode de standardizare - scăderea mediei și împărțirea la deviația standard.

Metode abordate pentru rezolvarea problemei supuse

Cele două metode folosite pentru a încerca să obțin un scor cât mai bun au fost **Support Vector Clustering (SVC)** și **Convolutional Neural Network (CNN)**.

Support Vector Clustering

Support Vector Clustering (SVC) este un algoritm de clasificare ce are ca scop principal împărțirea un set de date în grupuri conform unui anumit criteriu. Pentru problema noastra, criteriile folosite vor fi pixelii individuali și valorile acestora. Clusterizarea reprezintă procesul în sine de formare a diferitelor grupuri și de modelare a barierelor de separare dintre ele, urmând ca, clasificarea să reprezinte doar procesul de a 'observa' în ce grup pică imaginea în funcție de valoarea tuturor pixelilor.

```
classifier = svm.SVC()
classifier.fit(train_images, train_images_labels)
```

Metoda aceasta nu este optimă pentru clasificarea imaginilor întrucât nu percepe atât de bine conceptul de 'pattern match' între imaginile de același fel. (Acuratețea pentru modelul acesta a fost de 47.6%.

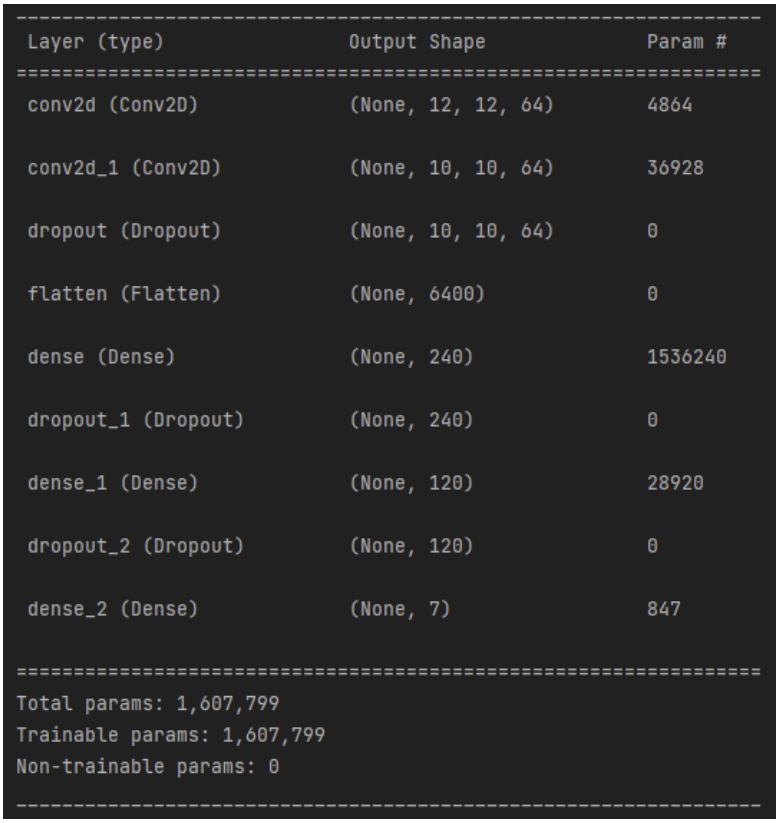
Convolutional Neural Network

Convolutional Neural Network este un algoritm de clasificare din cadrul conceptului de Deep Learning ce are ca input o imagine asupra căreia calculează diverse pattern-uri pe care le consideră 'importante', astfel, problema clasificării reprezentând doar evidențierea pe cât mai posibil a acestor pattern-uri.

<u>Arhitectura modelului</u>

```
classifier = keras.Sequential([
    keras.layers.Conv2D(64, (5, 5), input_shape=(16, 16, 3), activation='relu'),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.Dropout(0.4),
    keras.layers.Flatten(),
    keras.layers.Dense(240, activation='relu'),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(120, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(7, activation='softmax')
])
```

După cum se poate observa, arhitectura modelului este una relativ simplă, având numai 2 layer-uri convoluționale și 3 layere dense. Desigur, pentru a evita overfitting-ul, am adăugat layere de dropout.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 12, 12, 64)	4864
conv2d_1 (Conv2D)	(None, 10, 10, 64)	36928
dropout (Dropout)	(None, 10, 10, 64)	0
flatten (Flatten)	(None, 6400)	0
dense (Dense)	(None, 240)	1536240
dropout_1 (Dropout)	(None, 240)	0
dense_1 (Dense)	(None, 120)	28920
dropout_2 (Dropout)	(None, 120)	0
dense_2 (Dense)	(None, 7)	847

Total params: 1,607,799
Trainable params: 1,607,799
Non-trainable params: 0

Summay-ul rețelei

Layer-ul Conv2D

Din cauza marimilor scăzute ale imaginilor de antrenare și validare, modelul realizat mai sus conține numai două layere convoluționale. Aceste layere au ca scop principal detectarea de pattern-uri realizate în urma aplicării mai multor filtre. Primul layer are 64 de filtre, un kernel de 5x5, un input shape de 16x16x3 și o funcție de activare Relu, al doilea layer având doar kernelul mai mic (3x3).

- Filtrele

  Filtrele au ca scop principal detecatea de pattern-uri prin calcularea (împreună cu kernel-ul) a mai multor valori realizate în urma aplicării valorilor din cadrul filtrelor. Aceste filtre pot fi analoage edge detection-ului, emboss-ului sau bevel-ului.

- Kernel

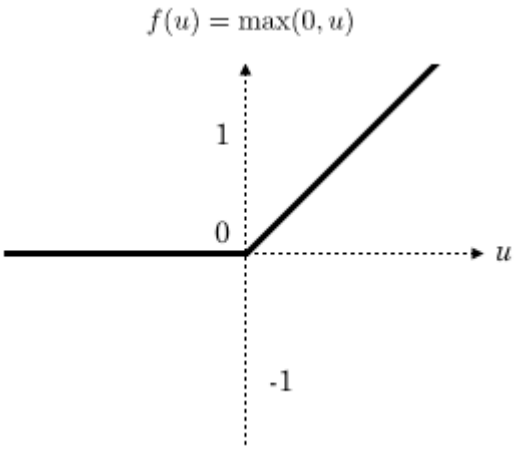
  Kernel-ul nu reprezinta nimic mai mult decât un ‘window’ în care sunt calculate valorile filtrelor. Acest kernel se plimba pe fiecare pixel și conține valorile din filtre, iar, în urma calculelor, pixelului îi este atribuită valoarea mediei filtrului. Un kernel de 3x3 reprezintă ca dimensiunea window-ului are mărime de 3 pixeli pe 3 pixeli.

- Input Shape

  Deoarece imaginile sunt destul de mici, am considerat că o caracteristică foarte importantă poate fi și culoarea imaginii (de obicei, image classification-ul se realizează pe imagini de tip ‘grayscale’). De aceea, am luat toate cele 3 canale de culoare, rezultând într-un input shape de 16x16x3.

- Funcția de activare

  Pentru modelul actual, am folosit cea mai utilizată funcție de activare în rețelele neuronale convoluționare - Relu.



Layer-ul Dropout

Deseori, rețelele neuronale convoluționare antrenate pe seturi de date mici pot avea parte de conceptul de overfit pe datele de antrenament. Pentu a rezolva această problemă, layer-ul dropout elimină temporar legăturile dintre diverși neuroni pentru a încerca să vadă ‘altfel’ imaginea. Parametrul din paranteză reprezintă procentul de legături la care renunță.

Layer-ul Dense

A doua parte importantă a modelului ales este reprezentată de către cele trei layere Dense. Aceste layere sunt cele mai utilizate atunci când vine vorba de rețele neuronale întrucât își îndeplinesc scopul principal cât se poate de eficient. Layer-ul dense, are ca input produsul scalar al tuturor neuronilor și greutăților din stratul de dinainte, iar ca output suma valorilor trecută prin funcția de activare.

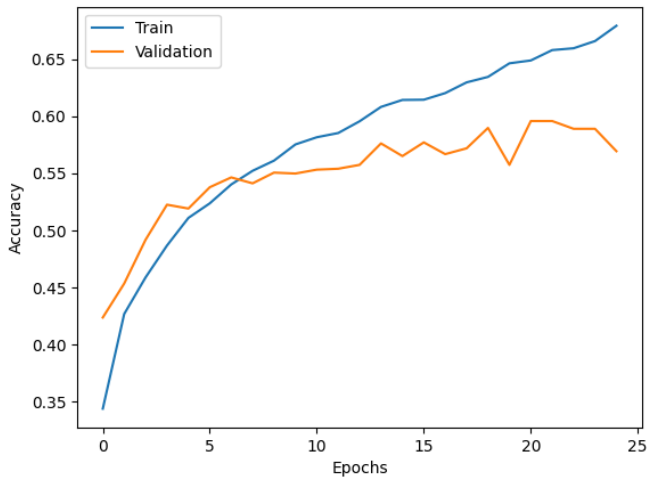
Primul layer are 240 de neuroni, al doilea 120, iar, ultimul, 7. Al treilea layer dense are doar 7 neuroni întrucât acesta este layer-ul de output din rețea pentru cele 7 clase pe care trebuie sa facem clasificări. Primele două layere au ca funcție de activare 'relu', pe când ultimul 'softmax' întrucât rămânem, după toate ecuațiile realizate, cu o listă de probabilități, iar clasificatorul va alege valoarea cu cea mai mare probabilitate ca fiind clasa identificată.

Augmentare pe date

Dat fiind setul mic de date de antrenare și validare, am fost nevoit să mă folosesc de augmentare, astfel încât să pot realiza cât mai multe din cât mai puțin. Augmentarea nu reprezintă nimic mai mult decât aducerea a mici modificări asupra datelor de antrenare (flip, rotation, zoom etc.).

```
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    horizontal_flip=True,
    zoom_range=0.1,
)
```

Astfel, modelul prezentat mai sus are o acuratețe de 59.8% pe datele de testare. Evoluția în timp a modelului și matricea de confuzie fiind:



923	4	21	8	4	1	5
5	972	2				
26	2	892	30	13	8	17
12	4	32	826	24	48	30
5	1	28	24	898	13	14
7	2	28	111	18	801	13
5		16	27	3	4	943