Clasificare binara al Alzheimerului folosing ResNet50

Bursuc Serban-Mihai, Avramescu Andrei-Cosmin

May 11, 2022

Abstract

ResNet50, binary Alzheimer classification

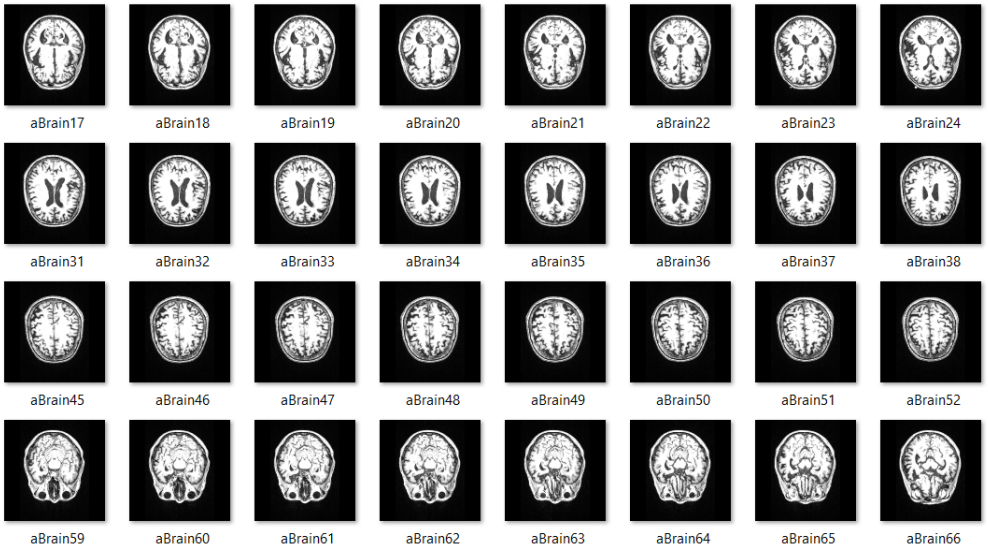
1 Introducere

Boala Alzheimer (AD) este o boală neurodegenerativă progresivă și incurabilă care distruge celulele creierului și provoacă pierderea memoriei pacientului. O depistare timpurie poate preveni deteriorarea în continuare a celulelor cerebrale și, prin urmare, poate evita pierderea permanentă a memoriei. În ultimii câțiva ani, au fost propuse diverse instrumente și tehnici automate pentru diagnosticarea bolii Alzheimer. Mai multe metode se concentrează pe detectarea rapidă, precisă și timpurie a bolii pentru a minimiza pierderile de sănătate mintală a pacienților.

Propunem un model de retea neuronala care foloseste ca baza arhitectura ResNet.

2 Dataset

Ca dataset s-au folosit o serie de scanari RMN care au fost feliate in poze, despre care se stie diagnosticul. Pentru ambele clase exista aproximativ 4000 de poze.



Antrenarea a avut loc folosind aceste poze. Clasele de clasificare sunt cognitiv normal si dementie Alzheimer.

3 Arhitectura retelei

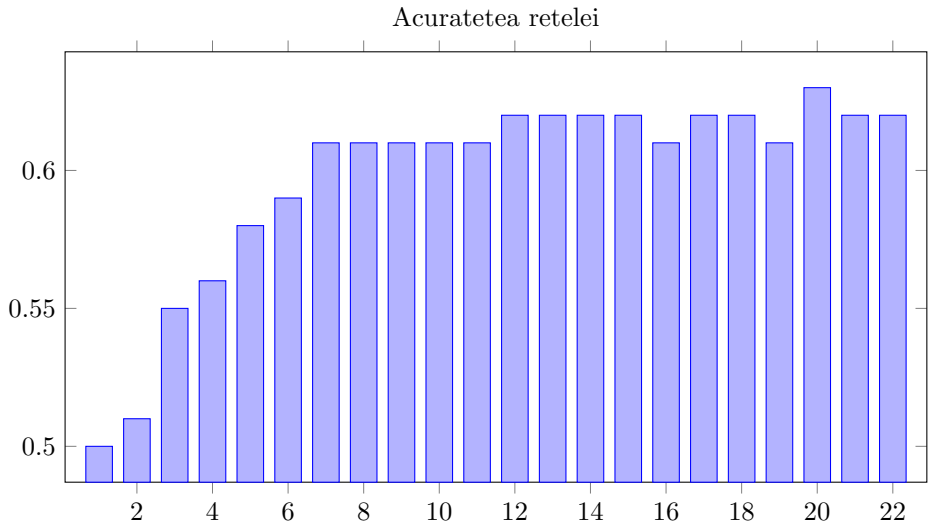
Pe langa arhitectura de baza ResNet au mai fost adaugate niste layere pentru optimizarea acuratetei retelei.

```
base_model = ResNet50(input_shape=(224,224,3),
                      include_top=False,
                      weights="imagenet")

for layer in base_model.layers:
    layer.trainable=False


headModel = base_model.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(256, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)

model = Model(inputs=base_model.input, outputs=headModel)
```

4 Rezultate

Reteaua a fost antrenata folosind 22 de epoci. Rezultatele sunt urmatoarele:

```
Total params: 30,424,225
Trainable params: 6,635,297
Non-trainable params: 23,788,928
_________________________________________________________________
Epoch 1/50
202/202 [==============================] - 1219s 6s/step - loss: 0.7106 - accuracy: 0.5003 - val_loss: 0.6906 - val_accuracy: 0.5469 - lr: 0.0100
Epoch 2/50
202/202 [==============================] - 1149s 6s/step - loss: 0.6941 - accuracy: 0.5149 - val_loss: 0.6898 - val_accuracy: 0.5781 - lr: 0.0063
Epoch 3/50
202/202 [==============================] - 1145s 6s/step - loss: 0.6849 - accuracy: 0.5548 - val_loss: 0.6622 - val_accuracy: 0.5469 - lr: 0.0040
Epoch 4/50
202/202 [==============================] - 1145s 6s/step - loss: 0.6820 - accuracy: 0.5647 - val_loss: 0.6191 - val_accuracy: 0.7344 - lr: 0.0025
Epoch 5/50
202/202 [==============================] - 1128s 6s/step - loss: 0.6753 - accuracy: 0.5870 - val_loss: 0.7318 - val_accuracy: 0.5469 - lr: 0.0016
Epoch 6/50
202/202 [==============================] - 1143s 6s/step - loss: 0.6685 - accuracy: 0.5937 - val_loss: 0.7314 - val_accuracy: 0.5469 - lr: 0.0010
Epoch 7/50
202/202 [==============================] - 1148s 6s/step - loss: 0.6618 - accuracy: 0.6140 - val_loss: 0.7212 - val_accuracy: 0.5625 - lr: 6.3096e-04
Epoch 8/50
202/202 [==============================] - 1102s 5s/step - loss: 0.6586 - accuracy: 0.6164 - val_loss: 0.7592 - val_accuracy: 0.5781 - lr: 3.9811e-04
Epoch 9/50
202/202 [==============================] - 1084s 5s/step - loss: 0.6583 - accuracy: 0.6179 - val_loss: 0.6701 - val_accuracy: 0.6250 - lr: 2.5119e-04
Epoch 10/50
202/202 [==============================] - 1125s 6s/step - loss: 0.6559 - accuracy: 0.6195 - val_loss: 0.8420 - val_accuracy: 0.4062 - lr: 1.5849e-04
Epoch 11/50
202/202 [==============================] - 1129s 6s/step - loss: 0.6584 - accuracy: 0.6184 - val_loss: 0.7359 - val_accuracy: 0.5312 - lr: 1.0000e-04
Epoch 12/50
202/202 [==============================] - 1143s 6s/step - loss: 0.6534 - accuracy: 0.6216 - val_loss: 0.7818 - val_accuracy: 0.4219 - lr: 6.3096e-05
Epoch 13/50
202/202 [==============================] - 1140s 6s/step - loss: 0.6528 - accuracy: 0.6253 - val_loss: 0.7427 - val_accuracy: 0.5469 - lr: 3.9811e-05
Epoch 14/50
202/202 [==============================] - 1126s 6s/step - loss: 0.6528 - accuracy: 0.6240 - val_loss: 0.7223 - val_accuracy: 0.5781 - lr: 2.5119e-05
Epoch 15/50
202/202 [==============================] - 1109s 5s/step - loss: 0.6522 - accuracy: 0.6277 - val_loss: 0.7593 - val_accuracy: 0.5156 - lr: 1.5849e-05
Epoch 16/50
202/202 [==============================] - 1104s 5s/step - loss: 0.6564 - accuracy: 0.6195 - val_loss: 0.7053 - val_accuracy: 0.5781 - lr: 1.0000e-05
Epoch 17/50
202/202 [==============================] - 1182s 6s/step - loss: 0.6547 - accuracy: 0.6213 - val_loss: 0.7713 - val_accuracy: 0.5000 - lr: 6.3096e-06
Epoch 18/50
202/202 [==============================] - 1085s 5s/step - loss: 0.6494 - accuracy: 0.6278 - val_loss: 0.7587 - val_accuracy: 0.4219 - lr: 3.9811e-06
Epoch 19/50
202/202 [==============================] - 1131s 6s/step - loss: 0.6555 - accuracy: 0.6188 - val_loss: 0.7386 - val_accuracy: 0.5312 - lr: 2.5119e-06
Epoch 20/50
202/202 [==============================] - 1139s 6s/step - loss: 0.6537 - accuracy: 0.6314 - val_loss: 0.7573 - val_accuracy: 0.4688 - lr: 1.5849e-06
Epoch 21/50
202/202 [==============================] - 1108s 5s/step - loss: 0.6500 - accuracy: 0.6294 - val_loss: 0.7022 - val_accuracy: 0.5625 - lr: 1.0000e-06
Epoch 22/50
 26/202 [==>...........................] - ETA: 15:32 - loss: 0.6627 - accuracy: 0.5950
```



Cel mai important parametru este **accuracy**. Primele 5 epoci ofera o crestere substantiala a acuratetii, insa dupa a 6-a epoca acuratetea se stabilizeaza in jur de 62%. Rularea mai indelungata a antrenarii ar fi oferit rezultate asemanatoare.

In practica reteaua are sansa de 38% de a oferi un diagnostic fals. Acest lucru se poate observa usor in urmatorul test:

```
def test1():
  predictions = nn.model.predict(cnn.CNN.predictSetup("./binaryClassificationDataset/test/yes/slice1.png"))
  if(predictions[0][0]>predictions[0][1]):
    test="no"
  elif(predictions[0][1]>predictions[0][0]):
    test="yes"
  assert test != "no", "ERROR: slice1.png from YES is considered NO or the prediction was equal"
  print("Test 1 was successfully passed")
test1()

# testing for a single NonDementia image
def test2():
  predictions = nn.model.predict(cnn.CNN.predictSetup("./binaryClassificationDataset/test/no/slice99.png"))
  if(predictions[0][0]>predictions[0][1]):
    test="no"
  elif(predictions[0][1]>predictions[0][0]):
    test="yes"
  assert test != "yes", "ERROR: slice99.png from NO is considered YES or the prediction was equal"
  print("Test 2 was successfully passed")
test2()
```

```
Test 1 was successfully passed
Traceback (most recent call last):
  File "D:\IP\binary classification using resnet50\testing.py", line 35, in <module>
    test2()
  File "D:\IP\binary classification using resnet50\testing.py", line 33, in test2
    assert test != "yes", "ERROR: slice99.png from NO is considered YES or the prediction was equal"
AssertionError: ERROR: slice99.png from NO is considered YES or the prediction was equal
```

In primul test punem reteaua sa prezica diagnosticul bazat pe o poza despre care se stie ca se clasifica ca dementie Alzheimer. Reteaua prezice cu succes acest lucru, insa pentru a doua poza despre care se stie ca se clasifica ca cognitiv normal, reteaua ofera diagnosticul de dementie Alzheimer. Sansele de esec sunt similare cu cele de succes.

5 Imbunatatiri

O prima imbunatatire ar fi o arhitectura mai profesional definita. O astfel de arhitectura se poate obtine prin FastAI, care incarca prin libraria sa o retea deja optimizata. O alta imbunatatire ar fi cresterea numarului de poze, precum si tipul de felii ale scanarilor (la un unghi, incetosate etc.)