

Surgical Mask Detection

DOCUMENTATIE

Chirculete Vlad Mihai




24.05.2020

Unelte utilizate

In realizarea acestui proiect am utilizat framework-ul [Keras](#) cu backend-ul [Tensorflow](#) pentru a antrena o retea neurala ce primeste ca input niste sunete si trebuie sa distinga intre cele cu masca chirurgicala si cele fara.

Structura proiectului

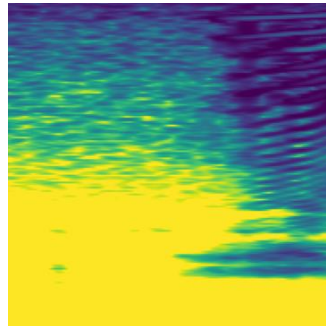
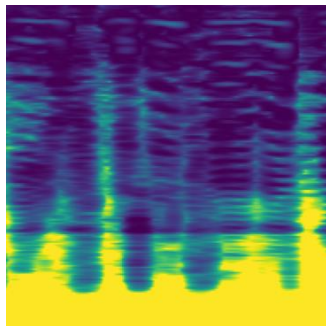
Datasetul initial era alcatuit din 12.000 de sunete impartite astfel:

-  **test**
-  **train**
-  **validation**

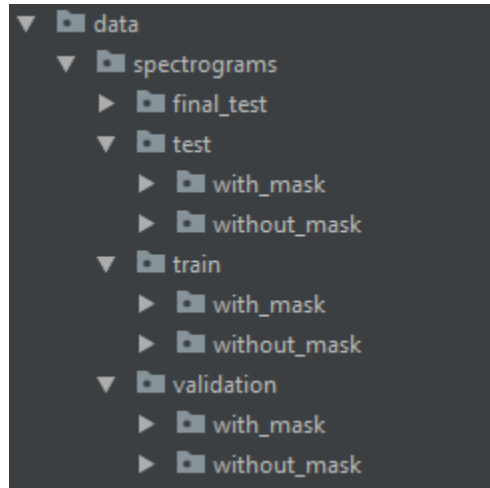
- Folderul **test** continea 3.000 de sunete destinate evaluarii finale.
- Folderul **train** continea 8.000 de sunete destinate antrenarii.
- Folderul **validation** continea 1.000 de sunete destinate validarii.

Fisierele din datasetul initial erau in format ***.wav** iar modelul retelei pe care urma sa o construiesc necesita un dataset alcatuit din imagini. Din acest motiv am utilizat pachetul python [LibROSA](#) pentru a genera spectrogramele fiecarui sunet din dataset si a le salva in format ***.png** pentru a antrena ulterior reseaua pe acestea.

Exemplu de spectrograme (acestea sunt pentru sunete cu masca din setul de antrenare):



Asadar a fost necesara construirea unui alt dataset din cel initial. Datasetul obtinut in urma generarii spectrogramelor pentru fiecare sunet a fost ulterior impartit in felul urmator:



- Folderul **final_test** contine spectrogramele corespunzatoare celor 3.000 de sunete din datasetul initial de testare.
- Folderul **test** contine 1.000 de spectrograme luate din cele 8.000 destinate antrenarii.
- Folderul **train** contine 7.000 de spectrograme din cele 8.000 initial destinate antrenarii.
- Folderul **validation** contine 1.000 de spectrograme corespunzatoare celor 1.000 de sunete din datasetul initial de validare.

Codul responsabil construirii acestui dataset secundar se regaseste in scripturile **build_datasets.py** si **main.py** care apeleaza functiile din primul script in vederea construirii datasetului. Codul responsabil de generarea spectrogramei pe baza unui fisier *.wav se regaseste in **utils.py** si este apelat in functiile din **build_datasets.py** iar outputul este prelucrat si salvat in fisiere *.png.

Abordare

Inspiratia pentru acest proiect a provenit dintr-un proiect asemanator. [Proiectul](#) in cauza propunea identificarea celulelor parazitare cu malarie cu ajutorul unui dispozitiv portabil pe care era pusa o retea antrenata pentru a distinge intre imagini cu cellule parazitare si imagini cu cellule neparazitare.

Proiectul respectiv era ca si proiectul meu o problema de clasificare binara: in cazul proiectului meu trebuia sa disting intre sunete cu masca si sunete fara, iar in cel de mai sus trebuia facuta diferenta intre celule parazitare si celule neparazitate.

Prin urmare am construit un model de retea neurala reziduala ([ResNet](#)) pe care am antrenat-o pe datasetul de spectrograme obtinute din procesarea sunetelor din datasetul initial. Arhitectura modelului utilizat este definita in scriptul **resnet.py** din cadrul folderului **pyimagesearch**.

Despre model

Scriptul care se ocupa de antrenarea retelei, compilarea si salvarea modelului este **save_model.py**. Acesta primeste ca parametrii numele fisierului in care va fi salvat modelul si numarul de epoci pentru care se va antrena acesta.

Utilizam **ImageDataGenerator** pentru a augmenta dataseturile:

```
44 # initializam obiectul de augmentare al datasetului de spectrograme de antrenare
45 trainAug = ImageDataGenerator(
46     rescale=1 / 255.0,
47     rotation_range=20,
48     zoom_range=0.05,
49     width_shift_range=0.05,
50     height_shift_range=0.05,
51     shear_range=0.05,
52     horizontal_flip=True,
53     fill_mode="nearest")
```

```
55 # initializam obiectul de augmentare al datasetului de validare si testare
56 valAug = ImageDataGenerator(rescale=1 / 255.0)
```

```
58 # initializam generatorul de antrenare
59 trainGen = trainAug.flow_from_directory(
60     TRAIN_PATH,
61     class_mode="categorical",
62     target_size=(64, 64),
63     color_mode="rgb",
64     shuffle=True,
65     batch_size=BS)
```

```

67 # initializam generatorul de validare
68 valGen = valAug.flow_from_directory(
69     VAL_PATH,
70     class_mode="categorical",
71     target_size=(64, 64),
72     color_mode="rgb",
73     shuffle=False,
74     batch_size=BS)

```

```

76 # initializam generatorul de testare
77 testGen = valAug.flow_from_directory(
78     TEST_PATH,
79     class_mode="categorical",
80     target_size=(64, 64),
81     color_mode="rgb",
82     shuffle=False,
83     batch_size=BS)

```

Parametrii utilizati:

- class_mode = “**categorical**” (deoarece etichetele sunt de tipul ‘categorical vector’)
- target_size reprezinta dimensiunea la care vom redimensiona imaginile
- color_mode este RGB deoarece imaginile sunt in format RGB
- batch_size reprezinta cate imagini se trec simultan prin model intr-o iteratie
- pentru generatorul de antrenare se vor amesteca imaginile (shuffle=**true**)

Urmatorul pas este sa ne initializam si sa compilam modelul pentru retea.

```

85 # initializam implementarea Keras a modelului ResNet si il compilam
86 model = ResNet.build(64, 64, 3, 2, (2, 2, 3),
87                     (32, 64, 128, 256), reg=0.0005)

```

- Imaginile vor fi de forma 64 x 64 x 3 (canale deoarece folosim RGB)
- Avem un total de 2 clase (cu sau fara masca)
- Primul layer CONV din ResNet-ul inainte de reducerea dimensiunilor va avea un total de 64 de filtre
- Urmand sa suprapunem 3 seturi de module reziduale. Cele trei straturi CONV din fiecare modul rezidual va invata 32, 32 si 128 de filtre CONV. Dupa care reducem dimensiunile.
- Apoi suprapunem 4 seturi de module reziduale, unde fiecare strat CONV va invata 64, 64 si 256 de filtre. Reducem din nou dimensiunile.
- In final suprapunem 6 seturi de module reziduale, unde fiecare strat va invata 128, 128 si 512 filtre. Se reduc inca o data dimensiunile inainte de a aplica un clasificator **softmax**.

```
88 opt = SGD(lr=1e-1, momentum=0.9, decay=1e-1 / NUM_EPOCHS)
```

Initializam un optimizator SGD cu o rata de invatare initiala de **1e-1**, momentum **0.9**.

In cele din urma ne compilam modelul:

```
89 model.compile(loss="binary_crossentropy", optimizer=opt,  
90               metrics=["accuracy"])
```

A fost utilizata functia de loss “**binary_crossentropy**” deoarece problema noastra este una de clasificare binara (cu sau fara masca).

Dupa ce am parcurs toti acesti pasi putem incepe antrenarea retelei:

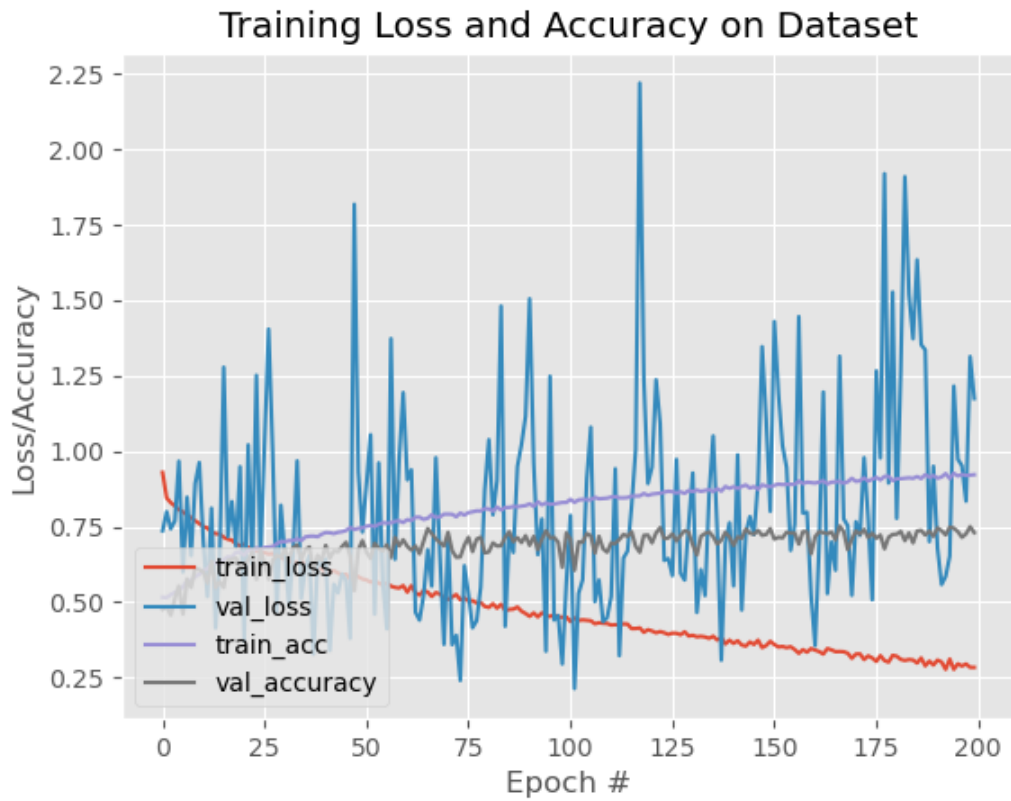
```
92 # antrenam modelul Keras  
93 H = model.fit_generator(  
94     trainGen,  
95     steps_per_epoch=totalTrain // BS,  
96     validation_data=valGen,  
97     validation_steps=totalVal // BS,  
98     epochs=NUM_EPOCHS)
```

Generatorul nostru de antrenare (trainGen) se va ocupa de incarcarea imaginilor si parsarea etichetelor (claselor) in functie de numele folderului in care se afla (with_mask sau without_mask). La fel se va intampla si pentru imaginile de validare prin intermediul generatorului respectiv (valGen).

Finalul acestui script se ocupa de afisarea rezultatelor antrenarii cat si de salvarea unui grafic cu aceste date.

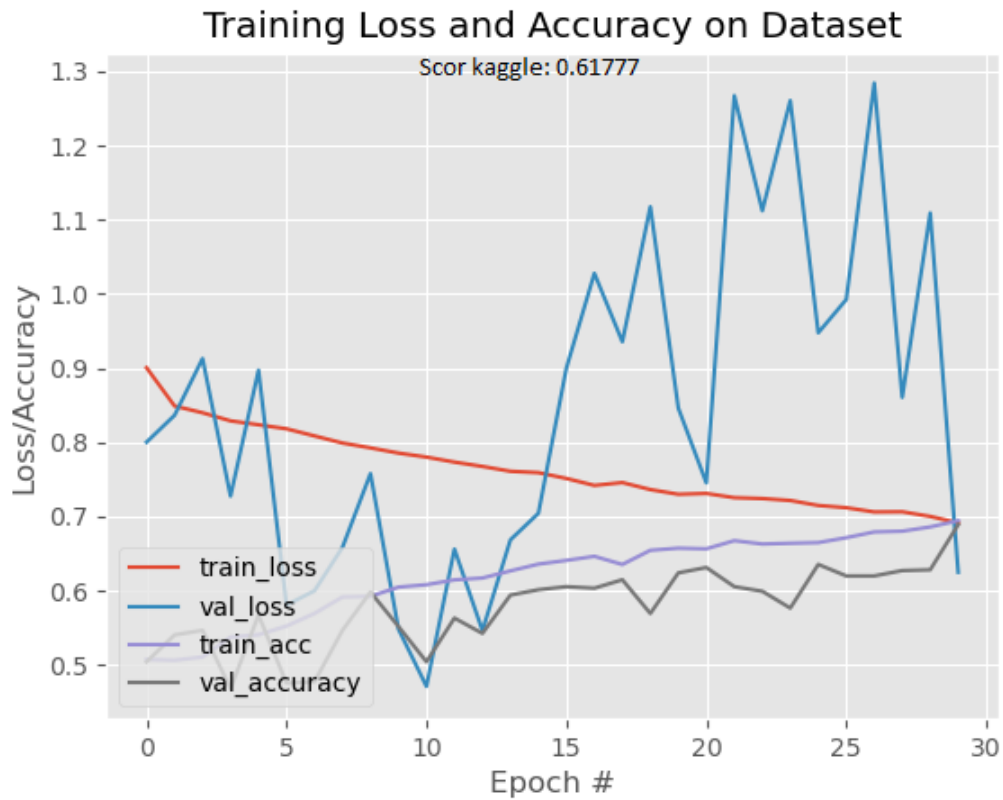
Rezultatele celor 2 modele utilizate in submisiile finale de pe Kaggle sunt urmatoarele:

Acest model a obtinut o acuratete de 0.62333 pe 30% din date:



	precision	recall	f1-score	support
with_mask	0.72	0.77	0.74	500
without_mask	0.75	0.70	0.72	500
accuracy			0.73	1000
macro avg	0.73	0.73	0.73	1000
weighted avg	0.73	0.73	0.73	1000

Acest model a obtinut o acuratete de 0.61777



	precision	recall	f1-score	support
with_mask	0.51	0.60	0.55	1500
without_mask	0.51	0.41	0.46	1500
accuracy			0.51	3000
macro avg	0.51	0.51	0.50	3000
weighted avg	0.51	0.51	0.50	3000

Evaluarea sunetelor din setul final de testare

Acest proces se realizeaza prin intermediul scriptului `load_model.py` care primeste ca parametrii calea modelului serializat si calea catre setul de testare.

Se colecteaza toate caile catre spectrogramele de testat dupa care se incarca modelul serializat, ca mai apoi sa parcurgem toate imaginile si pe fiecare sa o trecem prin model si acesta sa o clasifice.


```
# facem predictii pe imagine  
pred = model.predict(image)  
pred = pred.argmax(axis=1)[0]
```

Luam valoarea maxima prezisa pentru fiecare din imagini si o salvam in fisierul cu submisii.