Documentatie

Dragomir David-Alin 232

1)Clasificatorul:

Clasificatorul pe care l-am ales pentru modelul prezentat este Convolutional Neural Network pe care am obtinut si cel mai bun scor(**0.60444** pe 20% din date cat si o crestere pe toate datele **0.61523**). Un alt clasificatorul pe care am mai incercat a fost SVM insa acuratetea cu parametrii alesi de C=20 si Gamma de 10e-5 cu un scor de doar (**0.52777** pe 20% din date cat si un **0.50857**).Ca si observatie toate rezultate pe care le-am avut cu CNN pe 20% din date au avut cresteri mai mari in testarea pe datele finale in schimb de SVM au avut numai scaderi

CNN :
20% test | final result :
0.57666 → 0.59476
0.57444 → 0.60809
0.60444 → 0.61523

SVM :
20% test | final result :
0.52777 → 0.50857
0.56111 → 0.54476

Citirea datelor am facut-o folosind functia “open” pentru a citi din .txt [train.txt, test.txt cat si validation.txt] numele fisierelor cat si labelul in cazul fisierelor de antrenare si validare iar mai apoi am importat wavfile din modulul scipy.io pentru a citi datele si a pune informatia in vectori(fig 1.0).

```
with open("train.txt") as f:
  for line in f:
    filename = [elt.strip() for elt in line.split(',')]
    files.append(filename)
    wav_filePath = 'train/train/' + filename[0]
    fs, data = wavfile.read(wav_filePath)
    np_arr.append(data)
    train_label.append(int(filename[1]))
```

Fig 1.0

Dupa ce am realizat acest pas am transformat datele pe care le-am citit in numpy arrays si cu .shape am afisat forma array-urilor.(fig 1.1)

```
train_array_np = np.array(np_arr)
print("Train shape : ",train_array_np.shape)
test_array_np = np.array(np_arr_test)
print("Test shape : ",test_array_np.shape)
train_label_array_np = np.array(train_label)
validation_array_np = np.array(valdiation)
validation_label_np = np.array(validation_label)
print("Validation shape : ",validation_array_np.shape)
```

Fig 1.1

Dupa care am realizat split-ul datelor de antrenare in 0.2 de test pentru construirea modelului.(Fig 1.2)

```
X_train, X_test, y_train, y_test = train_test_split(train_array_np,train_label_array_np,test_size=0.2)
```

Fig 1.2

2)Construirea Modelului [Alegerea parametrilor]

```
class_count = 2
model = Sequential()
model.add(Conv1D(64, 3, activation='relu', input_shape=(16000, 1)))
model.add(Conv1D(64, 3, activation='relu'))
model.add(MaxPooling1D(3))
model.add(Conv1D(128, 3, activation='relu'))
model.add(Conv1D(128, 3, activation='relu'))
model.add(GlobalAveragePooling1D())
model.add(Dropout(0.5))
model.add(Dense(class_count, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])
```

Fig 1.3

Deoarece nu am transformat datele .wav in imagini pentru a folosi Conv2D am decis sa folosesc modelul in plan. input_shape`ul este de forma (len(data),1) [len(data) din fig 1.0].

Ultima parte a proiectului consta in folosirea functiei de expandare train_x cat si transformarea din array in matrice a lui y_train pentru a putea fi folosite in functia de model.fit

```
y_train = keras.utils.to_categorical(y_train, num_classes=class_count)
X_train = np.expand_dims(X_train, axis=2)
X_Validation = np.expand_dims(validation_array_np, axis=2)
y_Validation = keras.utils.to_categorical(validation_label_np, num_classes=class_count)
model.fit(X_train, y_train, batch_size=16, epochs=40)
X_predict = np.expand_dims(test_array_np, axis=2)
pred = model.predict_classes(X_predict)
```

(Fig 1.4)

Antrenarea datelor este realizata in 40 de stagii cu 16 sample-uri[batch_size].

```
Epoch 1/40
6400/6400 [==============================] - 630s 98ms/step - loss: 4.4432 - accuracy: 0.5250
Epoch 2/40
6400/6400 [==============================] - 615s 96ms/step - loss: 0.6920 - accuracy: 0.5350
Epoch 3/40
6400/6400 [==============================] - 614s 96ms/step - loss: 0.6905 - accuracy: 0.5423
Epoch 4/40
6400/6400 [==============================] - 615s 96ms/step - loss: 0.6894 - accuracy: 0.5373
Epoch 5/40
6400/6400 [==============================] - 613s 96ms/step - loss: 0.6923 - accuracy: 0.5356
Epoch 6/40
6400/6400 [==============================] - 616s 96ms/step - loss: 0.6903 - accuracy: 0.5325
Epoch 7/40
6400/6400 [==============================] - 614s 96ms/step - loss: 0.6918 - accuracy: 0.5314
Epoch 8/40
6400/6400 [==============================] - 614s 96ms/step - loss: 0.6894 - accuracy: 0.5436
Epoch 9/40
6400/6400 [==============================] - 614s 96ms/step - loss: 0.6917 - accuracy: 0.5459
Epoch 10/40
6400/6400 [==============================] - 614s 96ms/step - loss: 0.6881 - accuracy: 0.5425
Epoch 11/40
6400/6400 [==============================] - 614s 96ms/step - loss: 0.6859 - accuracy: 0.5487
Epoch 12/40
6400/6400 [==============================] - 615s 96ms/step - loss: 0.6835 - accuracy: 0.5586
Epoch 13/40
6400/6400 [==============================] - 613s 96ms/step - loss: 0.6838 - accuracy: 0.5558
Epoch 14/40
6400/6400 [==============================] - 615s 96ms/step - loss: 0.6795 - accuracy: 0.5648
Epoch 15/40
6400/6400 [==============================] - 614s 96ms/step - loss: 0.6750 - accuracy: 0.5723
Epoch 16/40
6400/6400 [==============================] - 613s 96ms/step - loss: 0.6729 - accuracy: 0.5750
Epoch 17/40
6400/6400 [==============================] - 614s 96ms/step - loss: 0.6696 - accuracy: 0.5781
Epoch 18/40
6400/6400 [==============================] - 615s 96ms/step - loss: 0.6735 - accuracy: 0.5733
Epoch 19/40
6400/6400 [==============================] - 613s 96ms/step - loss: 0.6702 - accuracy: 0.5800
Epoch 20/40
6400/6400 [==============================] - 614s 96ms/step - loss: 0.6678 - accuracy: 0.5789
Epoch 21/40
6400/6400 [==============================] - 613s 96ms/step - loss: 0.6841 - accuracy: 0.5664
Epoch 22/40
6400/6400 [==============================] - 613s 96ms/step - loss: 0.6729 - accuracy: 0.5688
Epoch 23/40
6400/6400 [==============================] - 615s 96ms/step - loss: 0.6721 - accuracy: 0.5727
Epoch 24/40
```

(Fig 1.5.1)

```
Epoch 24/40
6400/6400 [==============================] - 615s 96ms/step - loss: 0.6735 - accuracy: 0.5767
Epoch 25/40
6400/6400 [==============================] - 618s 97ms/step - loss: 0.6761 - accuracy: 0.5805
Epoch 26/40
6400/6400 [==============================] - 614s 96ms/step - loss: 0.6675 - accuracy: 0.5788
Epoch 27/40
6400/6400 [==============================] - 613s 96ms/step - loss: 0.7139 - accuracy: 0.5689
Epoch 28/40
6400/6400 [==============================] - 618s 97ms/step - loss: 0.6761 - accuracy: 0.5794
Epoch 29/40
6400/6400 [==============================] - 615s 96ms/step - loss: 0.6719 - accuracy: 0.5736
Epoch 30/40
6400/6400 [==============================] - 614s 96ms/step - loss: 0.6684 - accuracy: 0.5802
Epoch 31/40
6400/6400 [==============================] - 614s 96ms/step - loss: 0.6667 - accuracy: 0.5769
Epoch 32/40
6400/6400 [==============================] - 614s 96ms/step - loss: 0.6666 - accuracy: 0.5858
Epoch 33/40
6400/6400 [==============================] - 639s 100ms/step - loss: 0.6658 - accuracy: 0.5847
Epoch 34/40
6400/6400 [==============================] - 751s 117ms/step - loss: 0.6606 - accuracy: 0.5970
Epoch 35/40
6400/6400 [==============================] - 644s 101ms/step - loss: 0.6640 - accuracy: 0.5877
Epoch 36/40
6400/6400 [==============================] - 911s 142ms/step - loss: 0.6603 - accuracy: 0.5948
Epoch 37/40
6400/6400 [==============================] - 730s 114ms/step - loss: 0.6591 - accuracy: 0.5914
Epoch 38/40
6400/6400 [==============================] - 632s 99ms/step - loss: 0.6602 - accuracy: 0.5920
Epoch 39/40
6400/6400 [==============================] - 649s 101ms/step - loss: 0.6739 - accuracy: 0.5869
Epoch 40/40
6400/6400 [==============================] - 644s 101ms/step - loss: 0.6574 - accuracy: 0.5942
```

(Fig 1.5.2)

Dupa ce am reusit sa finalizam antrenarea datelor, folosind functia de predict_classes pe modelul construit putem afla care sunt vocile din folderul “test/test” care poarta sau nu poarta masti in functie de datele de intrare iar mai apoi sa le scriem intr-un fisier.

```
file = open("submission.txt", "w")
for i in range (len(file_test)):
  s = file_test[i] + "," + str(pred[i]) + "\n"
  file.write(s)
file.close()
```

(Fig 1.6)

Folosind functia evaluate pe datele de validare care asemeni datelor de train_x si train_y trebuie sa expandam X_Validation si sa transformam intr-o matrice binara y_Validation(Fig 1.4) putem afla si care este acuratetea si scorul obtinut de modelul nostru :

```
score, acc = model.evaluate(X_Validation, y_Validation, batch_size=16)
print("Score = ",score)
print("Acc = ", acc)
1000/1000 [==============================] - 23s 23ms/step
Score =  0.6534314885139465
Acc =  0.6179999709129333
```

(Fig 1.7)

Acesta este modelul de antrenare pe care l-am folosit pentru a prezice din fisierele de intrare, vocile care s-ar putea sa aiba masca si care nu, desi acuratetea putea fi imbunatatita probabil prin schimbarea datelor in imagini, folosirea unui alt optimizator sau modificarea dimensiunilor layerelor acesta este modelul final la care am ajuns si dupa care am trimis submissionul meu.