Brain anomaly detection – binary image classification

Proiectul constă în detectarea de anomalii pe radiografii ale creierului. Scopul este de a diferenția imaginile încadrându-le in doua clase: anomalii cu label 1 si normale cu label 0.

Setul de date conține un număr de 22149 de imagini cu dimensiunea 224x224 pixeli in format grayscale:

- 15000 imagini pentru antrenarea modelului
- 2000 imagini pentru validarea modelului si testarea acestuia local
- 5149 imagini pentru testarea modelului

Pentru a crea acest model de învățare automată am parcurs următorii pași:

1. Înțelegerea cerinței si a datelor
2. Colectarea datelor, împărțirea acestora in seturi de date(train/ validation/ test) si pregătirea acestora (citire si preprocesare)
3. Selectarea algoritmului
4. Antrenarea modelului pe setul de date train
5. Evaluarea modelului pe setul de date validation
6. Utilizarea modelului pentru a face predicții noi pe setul de date test

După ce au fost parcurși acești pași, în funcție de randamentul obținut de modelul de învățare, acesta trece printr-o perioada de testare si experimentare pentru a găsi valorile optime sau mai eficiente pentru setul nostru de date.

Pentru crearea unui astfel de model de învățare am implementat 2 algoritmi de machine learning, Random Forrest Classifier (RFC) si Convolutional Neural Network (CNN).

Random Forrest Classifier

Random Forrest Classifier sau RFC este un algoritm de învățare automata care se folosește in problemele de clasificare si presupune construirea unor arbori de decizie care fac predicții. In final, rezultatele se combina si obținem o predicție finală.

1. Se colectează datele de antrenare sub forma de matrice
2. Se selectează submulțimi din setul de antrenare pentru a evita overfitting-ul
3. Se construiește un număr mare de arbori de decizie, antrenându-se fiecare pe o submulțime de date
4. Se randomizează caracteristicile ca fiecare arbore de decizie sa fie diferit pentru a găsi cele mai importante atribute si pentru a evita overfitting-ul
5. Se combina predicțiile prin votare sau prin calcularea mediei predicțiilor ale arborilor de votare pentru a obține o predicție finala

Acesta este primul algoritm pe care l-am încercat pentru rezolvarea cerinței si prin urmare nu exista multe variante ale acestuia. M-am concentrat mai mult pe partea de pregătire a datelor decât pe testarea modelului si modificarea parametrilor.

Prima varianta a modelului este una clasica. Am început prin citirea datelor din fișier si împărțirea acestora in seturi de date, pe care apoi le-am memorat in fișiere de tipul {set_name}_images.npy, unde set_name este train/validation/test in funcție de setul de date al cărui aparțin imaginile. Am făcut acest lucru pentru a eficientiza citirea imaginilor atunci când vom avea nevoie de ele, realizându-se mult mai rapid decât citirea din fișier. După ce am încărcat in variabile imaginile si label-urile corespunzătoare pentru fiecare set de date am creat modelul de antrenare cu ajutorul librăriei sklearn:

rf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)

- n_estimators - numărul de arbori de decizie ce vor fi construiți
- max_depth – înălțimea/ adâncimea arborilor de decizie
- random_state – o valoare întreagă care este utilizată ca ”sămânță” pentru generatorul de numere aleatoare; asigura reproductibilitatea experimentului

După ce am creat si antrenat modelul, am făcut predicții pe setul de validare pentru a obține un f1_score aproximativ pentru setul de testare, în final salvând într-un fișier *.csv predicțiile pentru acesta. Pentru acest model am obținut un f1_score = 0.46 local.

Am încercat îmbunătățirea modelului concentrându-mă pe pregătirea datelor. Prima modificarea a fost adăugarea redimensionării imaginilor si convertirea acestora la grayscale. După mai multe teste asupra dimensiunii imaginilor, am ajuns la concluzia ca nu pot depăși scorul de 0.46 doar cu aceste modificări.

- 32x32 – 0.40
- 64x64 – 0.45
- 128x128 – 0.46
- 224x224 – 0.46
- 240x240 – 0.46
- 256x256 – 0.43
- 300x300 – 0.44

Având in vedere setul de date de antrenare nu este echilibrat am decis sa fac undersampling. Am adăugat in setul de antrenare toate imaginile cu label 1, iar din cele cu label 0 am ales un numar de imagini pe care l-am testat astfel încât sa obțin scorul cel mai bun. După ce am creat cele 2 np array-uri cu imaginile clasei 1 si imaginile clasei 0 le-am concatenat si le-am dat shuffle pentru a nu apărea într-o anumita ordine ca modelul sa se antreneze mai bine si mai eficient. Am memorat array-ul de imagini in același tip de fișier train_images.npy si label-urile sale în fișierul train_labels.npy.

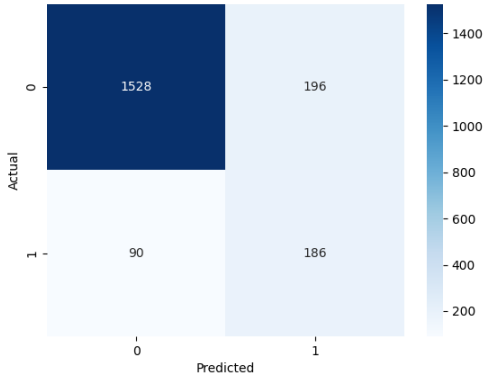
- 224x224 2000 imagini cu label 0 – 0.54
- 240x240 2000 imagini cu label 0 – 0.54

- 240x240 2500 imagini cu label 0 – 0.56
- 224x224 2500 imagini cu label 0 – 0.55

- 224x224 3000 imagini cu label 0 – 0.55
- 240x240 3000 imagini cu label 0 – 0.56

- 240x240 3500 imagini cu label 0 – 0.56
- 224x224 3500 imagini cu label 0 – 0.57

Am testat pentru 2 cele mai mari valori in funcție de dimensiunea imaginii mai multe variante de undersampling ca setul de date de antrenare sa fie echilibrat. In medie cel mai bun scor l-am obținut pe imaginile cu dimensiune 224x224 si 3500 de imagini cu label 0, însă cea mai buna predicție a fost data de setul de imagini de dimensiune 240x240 ce conține 3500 de imagini ale clasei 0 (aprox. 0.57 pe platforma Kaggle).



Confusion Matrix

Classification Report

	precision	recall	F1-score	Support
0	0.94	0.89	0.91	1724
1	0.49	0.67	0.56	276
Accuracy			0.86	2000
Macro avg	0.72	0.78	0.74	2000
Weighted avg	0.88	0.86	0.87	2000

Convolutional Neural Network

Convolutional Neural Network sau CNN este un algoritm utilizat frecvent in problemele de clasificare si de recunoaștere a imaginilor. Rețelele neuronale convoluționale sunt un tip de rețele neuronale artificiale specializate in prelucrarea imaginilor. Ele sunt capabile sa extragă caracteristici semnificative din setul de antrenare si se auto învață sa clasifice aceste imagini in diferite categorii.

CNN-urile sunt construite de obicei trei tipuri de straturi de procesare: straturi convoluționale (Conv2D), straturi de pooling (MaxPooling2D) și straturi complet conectate (Dense).

In prima etapa, setul de imagini de antrenare este prelucrat de CNN cu ajutorul straturilor convolutionale. Acestea folosesc filtre sau kernel-uri ce reprezinta de fapt matrici, a cărei dimensiune este data de parametrul kernel_size al stratului. Filtrele recunosc caracteristici deplasându-se peste imagini si calculează produsul dintre elementul din kernel si pixelul imaginii. In

acest mod se construiește o harta de caracteristici. Aceasta harta reprezintă imaginea prelucrata de stratul convolutional.

In a doua etapa, stratul de pooling micșorează dimensiunea hărții de caracteristici luând cel mai important element al acesteia, fără a pierde informații utile.

Aceste etape se repeta in funcție de numărul de straturi care diferă de la rețea la rețea. In final, apar straturile complet conectate, formate, după cum spune si numele, din neuroni complet conectați, care transforma harta într-un vector de caracteristici. Vectorul este apoi preluat de un strat final care calculează probabilitățile apartenentei la fiecare clasa.

Hiperparametrii:

1. Conv2D
- un numar de filtre puteri ale lui 2 de la 16 la 256
- kernel_size = (3, 3) – dimensiunea matricei reprezentanta a filtrului unui strat
- activation = ‘relu’ – funcția relu este de forma f(x) = max(0,x) si este o funcție de optimizare comună, simplă și rapidă
- input_shape = (x, x, 1), unde x este dimensiunea imaginilor care sunt primite ca input, iar 1 este numărul de canale de culoare (grayscale – 1, RGB – 3)
2. MaxPool2D
- Pool_size = (2, 2) – dimensiunea matricei asupra căreia se face maximul pentru a găsi caracteristici importante
3. Dropout
- un număr subunitar care reprezintă rata de dropout; am folosit 0, 0.25 si 0.5
4. Dense
- Un număr de neuroni complet conectați ai stratului; am încercat mai multe variante (512, 256, 128, 64, 32) , dar am rămas la 64
- 1 ca număr de neuroni si activatorul „sigmoid” pentru a obține probabilitățile apartenentei la clasa 0 si 1

Compilarea modelului:

- optimizer='adam' – optimizer-ul este functia care actualizeaza ponderile in timpul antrenarii modelului. Adam este cel mai popular pentru eficienta sa si funcționalitatea sa pe o arie mare de probleme
- loss sau funcția de pierdere reprezintă capacitatea modelului de a prezice outputul corect si folosesc binary_crossentropy deoarece este o clasificare binara
- metrics = ”accuracy” – eficienta modelului se măsoară in acuratețe

Pentru problema găsirii anomaliilor din radiografii ale creierului am testat mai multe rețele neuronale convolutionale si am redimensionat imaginile la cea mai mica varianta cu scorul cel mai bun (128x128 pixeli). Am evitat undersampling-ul ca modelul să se antreneze cât mai mult si să găsească cât mai multe caracteristici.

Am început cu un CNN cu 4 straturi cu un număr de filtre crescător. Numărul de filtre creste de la strat la strat, fiind reprezentat de puteri ale lui 2, de la 16 la 128.

Stratul complet conectat al rețelei este format din 32 de neuroni.

Straturi	Parametrii
Conv2D	16, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Conv2D	128, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Flatten	Default
Dense	32, activation='relu'
Dense	1, activation='sigmoid'

F1_score = 0.55

Batch size-ul reprezintă numărul de imagini procesate in același timp. După ce este parcurs un batch, modelul își ajustează parametrii pe parcursul antrenării. După mai multe teste ale rețelei am constat ca batch-ul cu eficiența cea mai mare pentru setul de date este 64.

Antrenând CNN-ul pe 50 de epoci am descoperit ca 32 de neuroni al stratului complet conectat oferă un scor mai bun pe aceasta rețea si am obținut scorul 0.55 local si 0.58 pe platforma Kaggle, depășind scorul obținut cu algoritmul RFC.

Am încercat sa măresc numărul de neuroni la 64 pentru care am primit un rezultat mai bun local, însă doar 0.54 pe platforma, probabil din cauza overfitting-ului.

Pentru a obține rezultate mai bune, am decis sa adaug încă un strat în rețea si straturi de Dropout. Straturile Dropout setează aleator unități sau neuroni cu 0 pentru a reduce overfitting-ul.

Următoarele experimente au fost făcute pe imagini de dimensiune 128x128 pixeli, batch_size 64 si 100 de epoci.

Straturi	Parametrii
Conv2D	16, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	128, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	256, kernel_size=(3, 3), activation='relu'

MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Flatten	Default
Dense	64, activation='relu'
Dropout	0.5
Dense	1, activation='sigmoid'

F1_score = 0.39

După modificările făcute am obținut un scor si mai mic, așa că am modificat in continuare rețeaua pentru a obține un rezultat cat mai mare. Am început prin a schimba numărul de filtre al parametrilor si a-l face cat mai echilibrat.

Straturi	Parametrii
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	128, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Flatten	Default
Dense	64, activation='relu'
Dropout	0.5
Dense	1, activation='sigmoid'

F1_score = 0.41

Testul a avut succes, însă scorul este mult mai mic decât maximul obținut pe rețeaua precedenta cu doar 4 straturi. Analizând modificările, in încercarea de a rezolva overfitting-ul o problema ar putea fi underfitting-ul. Prin urmare, scăpam de unele straturi de dropout, pe rând si observam modificarea scorului.

Straturi	Parametrii
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25

Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	128, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Flatten	Default
Dense	64, activation='relu'
Dropout	0
Dense	1, activation='sigmoid'

F1_score = 0.50

Straturi	Parametrii
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	128, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Flatten	Default
Dense	64, activation='relu'
Dropout	0
Dense	1, activation='sigmoid'

F1_score = 0.55

Straturi	Parametrii
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)

Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	128, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Flatten	Default
Dense	64, activation='relu'
Dropout	0
Dense	1, activation='sigmoid'

F1_score = 0.56

Straturi	Parametrii
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	128, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Flatten	Default
Dense	64, activation='relu'
Dropout	0
Dense	1, activation='sigmoid'

F1_score = 0.50

Scorul s-a oprit din a creste, revenind la problema overfitting-ului. Prin aceste teste am obținut scor maxim pe rețeaua 32, 32 dropout 0; 64, 64, 128 dropout 0.25; dense 64 dropout 0, rețea pe care m-am decis sa o explorez mai mult. La o prima modificare a numărului de filtre pe ultimul strat am obținut deja un scor mai mare care mi-a adus un nou maxim.

Straturi	Parametrii
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)

Dropout	0
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Flatten	Default
Dense	64, activation='relu'
Dropout	0
Dense	1, activation='sigmoid'

F1_score = 0.58 (0.63 Kaggle)

Pentru a încerca sa depășesc acest scor am adăugat încă un strat pe rețeaua neuronala si am redimensionat imaginile, mărind numărul de pixeli.

Păstram informațiile obținute in testele anterioare si anume ca scorul cel mai bun a fost obținut de rețelele cu straturi care au numărul de filtre cat mai echilibrat intre 32 si 128, iar primele 2-3 straturi nu au dropout pentru a evita undersampling-ul.

In continuare testam diferite variante ale rețelei cu 6 straturi, mai complexa fata de cea precedenta, si obținem cu ușurința un scor care sa depășească rețelele anterioare. Următoarele experimente au fost făcute pe imagini de dimensiune 200x200 pixeli, batch_size 64 si 100 de epoci.

Straturi	Parametrii
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	128, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Flatten	Default
Dense	64, activation='relu'
Dropout	0

Dense	1, activation='sigmoid'

F1_score = 0.59 (0.64 Kaggle)

Verificam daca obținem un scor mai bun in cazul in care avem Dropout 0 pe jumătate din straturile rețelei.

Straturi	Parametrii
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	128, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Flatten	Default
Dense	64, activation='relu'
Dropout	0
Dense	1, activation='sigmoid'

F1_score = 0.61(0.63 Kaggle)

Testul a eșuat, deci revenim la varianta anterioara pe care o supunem la alte modificări. Creștem numărul de filtre pentru al treilea strat si obținem rețeaua cu 5 straturi care a avut cel mai mare scor, dar cu un strat in plus.

Straturi	Parametrii
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'

MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	128, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Flatten	Default
Dense	64, activation='relu'
Dropout	0
Dense	1, activation='sigmoid'

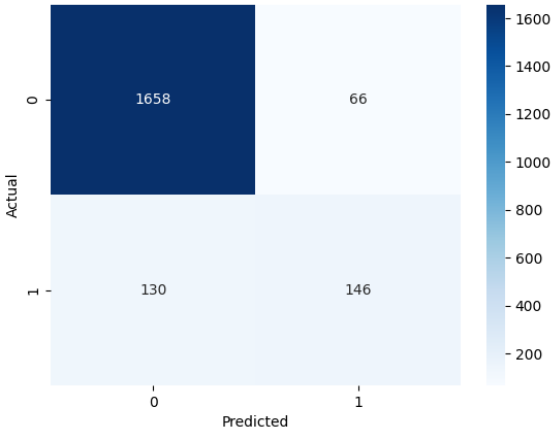
F1_score = 0.53

Rezultatul nu este cel așteptat si avem din nou un eșec pentru experiment, deci reluam rețeaua cu 5 straturi si adăugăm un altul cu 64 de filtre.

Straturi	Parametrii
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	32, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Conv2D	64, kernel_size=(3, 3), activation='relu'
MaxPooling2D	pool_size=(2, 2)
Dropout	0.25
Flatten	Default
Dense	64, activation='relu'
Dropout	0
Dense	1, activation='sigmoid'

F1_score = 0.60 (0.67 Kaggle)

Aceasta este rețeaua creata de mine care a obținut cel mai bun scor.



Confusion Matrix

Classification Report

	precision	recall	F1-score	Support
0	0.92	0.98	0.94	1724
1	0.75	0.43	0.55	276
Accuracy			0.90	2000
Macro avg	0.83	0.71	0.75	2000
Weighted avg	0.89	0.90	0.89	2000