Brain Tumor Classification - Pt1

Manea Lidia-Elena - 331CA

16. November 2024

```python
def load_images(path, labels):
    image_files = []
    labels_return = []
    for label in labels:
        for file in os.listdir(os.path.join(path, label)):
            image_files.append(os.path.join(path) + "/" + label + "/" + file)
            labels_return.append(label)
    return image_files, labels_return

class Dataset(torch.utils.data.Dataset):
        def __init__(self, image_files, labels, transforms):
            self.image_files = image_files
            self.labels = labels
            self.transforms = transforms
        def __len__(self):
            return len(self.image_files)
        def __getitem__(self, index):
            return self.transforms(self.image_files[index]), self.labels[index]
```

Abbildung 1: Dataset + functie de load

0.1 Introducere

Tema propune analiza si antrenarea unei retele neurale create from scratch pentru setul de date Brain Tumor Classification de pe kaggle. Datele vin in 2 foldere: unul de train si unul de test, pentru lucrul mai usor cu acestea. Datele de train se vor imparti intr-un set de antrenare si unul de validare, iar datele de test vor fi folosite pentru masurarea f1-score, acuratete, precizie.

0.2 Cerinta 1

Pentru extragerea si manipularea datelor, am creat o clasa care mosteneste Dataset-ul din torch. Am facut functii de incarcare a datasetului, de accesare a itemelor si de a numara imaginile din dataset.
Suplimentar, dupa ce am incarcat dataset-urile si label-urile, am transformat listele de labels in liste de numere, pentru a fi convertite dupa aceea in tensori la antrenare.

0.3 Cerinta 2

Am implementat o functie care imparte setul de date in train si val, pentru a fi folosite in antrenarea modelului. Aceasta functie ia ca argumente labelurile si imaginile, respectiv procentele in care se impart pentru train si pentru val.

0.4 Cerinta 3

Am creat barplots pentru fiecare folder: set de train si set de test.
Pentru train, se observa ca exista un dezechilibru intre clase, cele mai putine imagini fiind cele fara tumori. Acest lucru poate pune probleme in cazul in care modelul nu stie sa recunoasca imagini cu persoane care nu sunt afectate de tumori.

```python
def split_train_val(images_train, label_train, val, train):
    val_frac = val #0.2
    train_frac = train #0.8
    length = len(images_train)
    train_length = int(train_frac * length)
    indices = np.arange(length)
    np.random.shuffle(indices)

    val_split = int(val_frac * length)
    val_indices = indices[:val_split]
    train_indices = indices[val_split:train_length]
    train_x = [images_train[i] for i in train_indices]
    val_x = [images_train[i] for i in val_indices]

    train_y = [label_train[i] for i in train_indices]
    val_y = [label_train[i] for i in val_indices]

    return train_x, val_x, train_y, val_y
```

Abbildung 2: Functie de split

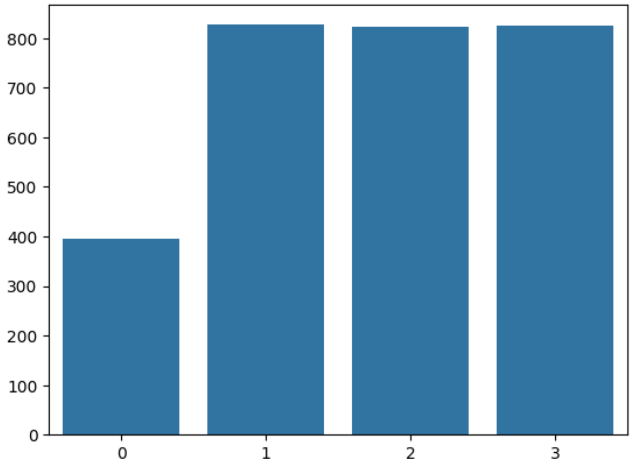

Abbildung 3: Folder train

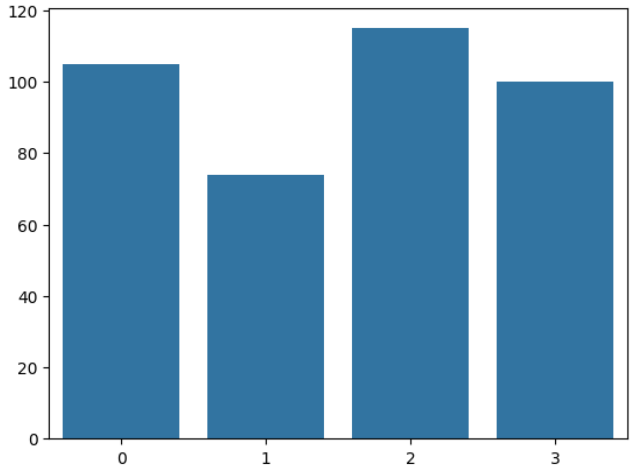

Abbildung 4: Folder test

Pentru test, clasele sunt in echilibru, existand diferente mici intre numarul de imagini per clasa.

0.5 Cerinta 4

Am extras cate 10 imagini din fiecare tip de tumora. Tumorile hipofizare sunt situate la baza craniului, iar acestea sunt omogene, dense si bine conturate. Tumorile meningiomale sunt situate la suprafata creierului, mai spre exterior, aproape de marginea craniului. Acestea sunt dense si sunt usor de identificat pe MRI din cauza contrastului de culoare. Tumorile glioma sunt mai rasfirate si nu au marginile atat de bine delimitate. Se afla in creier, cam oriunde, si nu sunt la fel de dense.

0.6 Cerinta 5

Pentru studierea caracteristicilor, am creat 3 set-uri: unul pentru dimensiuni, unul pentru numarul de canale si unul pentru distributia pixelilor. Pentru fiecare imagine, am scos informatiile relevante si le-am introdus in cele 3 set-uri. Se observa ca toate imaginile au formatul RGB (dar de fapt ele sunt grayscale, deci le fac conversie), ca au dimensiuni diferite si ca distributia pixelilor este diferita. Prin urmare, in seria de transformari care trebuie aplicate imaginilor, se introduc si transformari de resize la 256x256 si transformari din rgb in grayscale si de normalizarea a distributiei pixelilor, cu media 0.485, respectiv deviatia standard 0.229.

0.7 Cerinta 6

Am aplicat diverse transformari pentru prelucrarea si transformarea imaginilor. Am creat un pipeline Compose cu functii care sunt aplicate: prima oara se deschide imaginea in format PIL, apoi se redimensioneaza la 256 pe 256. Apoi se aplica cateva filtre: blur gaussian cu intensitatea 0.5, autocontrast, sharpeness. Dupa aplicarea transformarilor, imaginea se converteste din PIL in cv2,, in format grayscale. Apoi se reconverteste imaginea in format PIL, tot in format grayscale, apoi se roteste imaginea cu 10 grade; rezultatul transforma in tensor, si se normalizeaza imaginea cu media, respectiv deviatia standard mentionate mai sus. Gaussian blur estompeaza imaginea; daca rata de aplicare este foarte mare, detaliile nu se mai disting. Autocontrast evidentiaza detaliile din imagine, pe baza unor parametri dati in functie. Sharpness creste claritatea imaginii. Clahe imbunatateste contrastul unei imagini pe un anumit grid. Transformarile de resize si normalizare se aplica atat pe imaginile de validare, cat si pe cele de test. Rotatia roteste imaginea.

	precision	recall	f1-score	support
glioma_tumor	0.2613	0.4952	0.3421	105
meningioma_tumor	0.5882	0.2703	0.3704	74
no_tumor	0.4419	0.3304	0.3781	115
pituitary_tumor	0.2533	0.1900	0.2171	100
accuracy			0.3274	394
macro avg	0.3862	0.3215	0.3269	394
weighted avg	0.3734	0.3274	0.3262	394

Abbildung 5: Rezultate la test

```python
class LeNet(nn.Module):
    def __init__(self, in_size, out_size):
        super().__init__()
        self.conv1 = nn.Conv2d(in_size, 4, kernel_size=4, stride=2, padding=1, dilation=1)
        self.relu1 = nn.ReLU()
        self.avg = nn.MaxPool2d(kernel_size=4, stride=3)
        self.conv2 = nn.Conv2d(4, 8, kernel_size=4, stride=2, padding=1, dilation=1)
        self.drop = nn.Dropout(p=0.2)
        self.relu2 = nn.ReLU()
        self.flatten = nn.Flatten()
        self.linear = nn.Linear(3528, 4)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.avg(x)
        x = self.conv2(x)
        x = self.drop(x)
        x = self.relu2(x)
        x = self.flatten(x)
        x = self.linear(x)
        return x

    def name(self):
        return "LeNet"
```

Abbildung 6: Retea neurala

0.8 Cerinta 7

Am creat o retea neurala care este alcatuita din urmatoarele layere: 2 de conv2d, unul de maxpool2d, 2 de relu, unul de dropout, unul de flatten si unul de linear. Am folosit optimizatorul Adam cu lr = 0.001 si crossentropyloss la masurarea lossului.

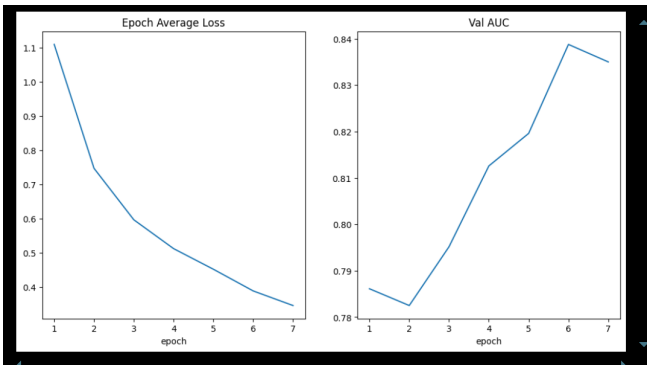

Abbildung 7: Performantele modelului