Proeict 1 Invatare Supervizata

Pintoiu Alina-Diana, grupa 342CC5

Cerinta 1:

Pentru cerinta 1 am avut de implementat o clasa, numita BrainTumorDataset, care incarca datele din arhiva cu imagini cu brain tumors si le categorisea dupa anumite label-uri.

Am creat o lista numita labels, care are cele 4 tipuri de brain tumors ('glioma_tumor', 'meningioma_tumor', 'no_tumor', 'pituitary_tumor') si 2 stringuti care erau path-urile catre imaginile de testare si antrenare.

Clasa **BrainTumorDataset** mosteneste Dataset din torch.utils.data.

Exista 3 metode in aceasta clasa:

- __init__ -> “Constructorul”, care populeaza variabilele necesare in acest fel:
    - self.data_paths -> aceasta este path-ul datelor, poate fi path-ul catre antrenare sau testare
    - self.image_paths -> este o lista in care se appenduiesc toate imaginile din acel path. Se appenduieste full path-ul.
        - Exemplu: Data/Testing/glioma_tumor/image(1).jpg
    - self.labels -> este o lista care pentru fiecare imagine appenduita in self.image_paths la index-ul i, este scris in labels din ce clasa face parte.
    - De asemenea, pentru training la clasa no_tumor se vor uploada de 2 ori imaginile (mai multe la subpunctele ulterioare)
- __len__ -> returneaza length-ul tuturor imaginiilor din acel set din dataset (antrenare/validare)
- __getitem__ -> returneaza imaginea si labelul la index-ul i dat din functie. De asemenea, daca exista transformare, aceasta transforma imaginea apoi o trimite inapoi.

Am folosit acest link ca ajutor: https://github.com/adisve/brain-tumor-classifier/tree/main

Cerinta 2:

Pentru cerinta 2 a trebuit sa impart setul de antrenare(training path) in 2 parti 80% antrenare si 20% validare.

Functia split_data_by_class(dataset) creeaza aceasta impartire:

- Se va memora impartirea in aceste variabile
  - train_image_paths = []
  - train_labels = []
  - val_image_paths = []
  - val_labels = []
- Pentru o buna impartire a setului de antrenare vor trebui
  - prima data impartite egal toate label-urile, adica sa fie 80% / 20% antrenare pt 'glioma_tumor', 'meningioma_tumor', 'no_tumor', 'pituitary_tumor'.
  - Pentru asta se vor prelucra imaginile specifice pentru fiecare clasa specific,
  - se va face un random la imaginile din acea clasa
  - se vor spiltui cu functia aceasta din lab 5:
    - **X_train, X_val = train_test_split(class_image_paths, test_size=0.2, random_state=42)**
  - Dupa ce sunt spilt-uite se vor appendui in variabilele de main sus (train_image_paths/val_image_paths, train_labels/val_labels)
  - Acest lucru se face pt fiecare clasa, iar la final se vor returna aceste liste

```
Number of training images: 2610
Number of validation images: 655
['Data/Training/glioma_tumor/gg (228).jpg', 'Data/Training/glioma_tumor/gg (208).jpg', 'Data/Training/glioma_tumor/gg (267).jpg', 'Data/Training
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
['Data/Training/glioma_tumor/gg (214).jpg', 'Data/Training/glioma_tumor/gg (215).jpg', 'Data/Training/glioma_tumor/gg (597).jpg', 'Data/Training
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Cerinta 3:

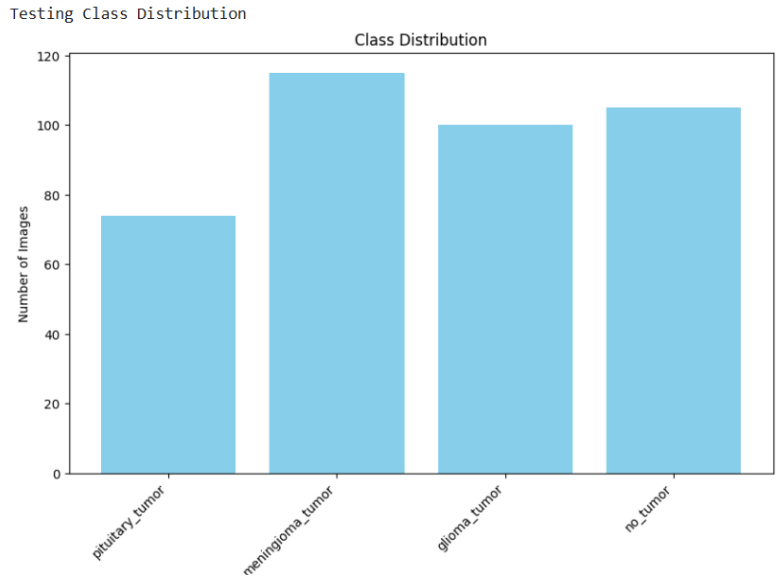
Pentru cerinta 3 am realizat o vizualizare a distributiei claselor pentru cele 3 seturi de date (testare, antrenare, validare)

Pentru vizualizare se va folosi aceasta functie: **plot_class_distribution(dataset)**

Pentru antrenare/validare se va folosi functia de la cerinta 2 pentru a splitui dataset-ul.

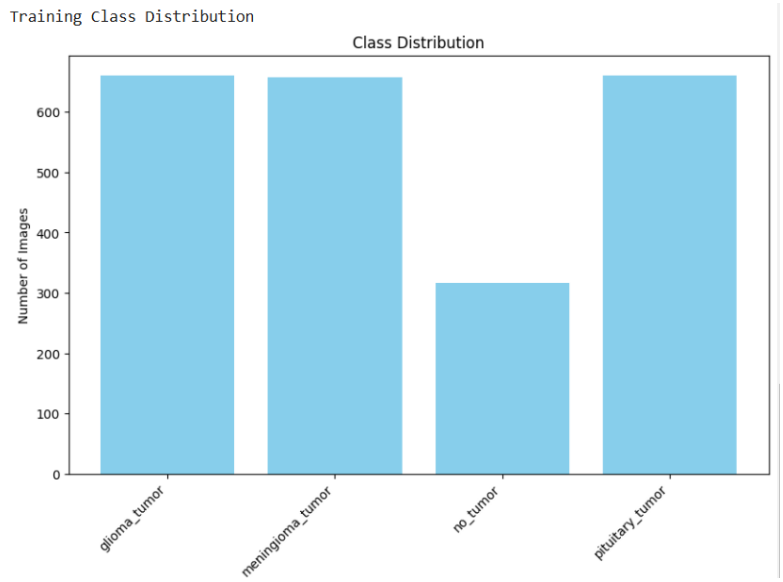
Acestea sunt histogramele date:

Testing class distribution:



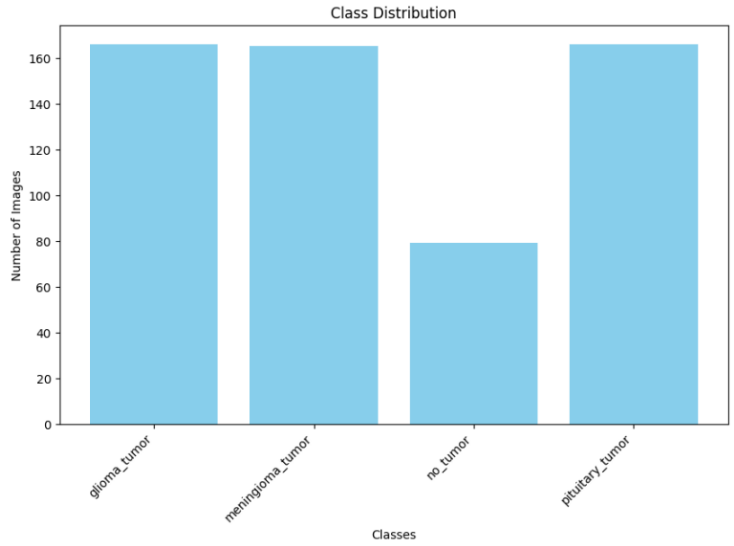
Se poate observa ca la setul de date de testing sunt ok reprezentate clasele, nu exista diferente mari intre numarul de imagini pentru fiecare clasa.

Training class distribution:



Validation class distribution:

Validation Class Distribution



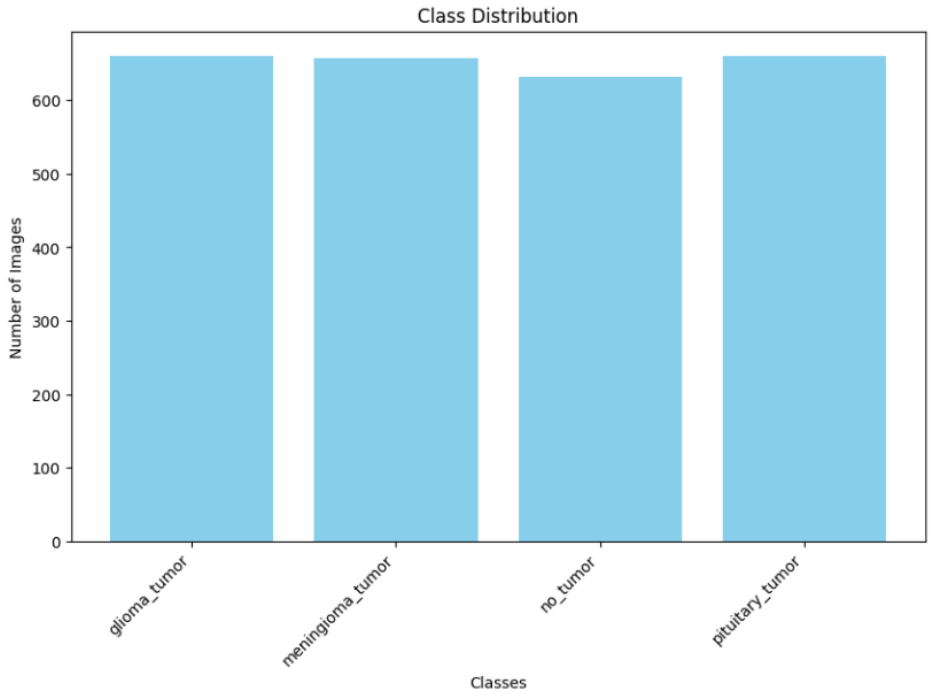
Se poate observa ca pt seturile de date de validation si training, imaginile de la no_tumor sunt jumate cat celelalte, adic sunt sub-reprezentate.

Pentru a putea remedia aceasta situatie si a echilibra clasele, o solutie ar fi augumentarea setului de date pentru clasa de no_tumor, prin dublarea acestuia. Acest lucru se realizeaza la initializare dataset-ului in clasa **BrainTumorDataset,** unde, daca suntem in clasa de 'no_tumor' se vor augumenta imaginile dublandu-le:

```
        elif class_name == 'no_tumor':
          label = labels[2]
          # if we are on the training set, we'll double the no_tumor images
          if 'Training' in data_paths:
            for image_name in os.listdir(class_type_path):
              whole_image_path = os.path.join(class_type_path, image_name)
              self.image_paths.append(whole_image_path)
              self.labels.append(label)
        elif class_name == 'pituitary_tumor':
```

Acum setul de date arata asa:

Training Class Distribution

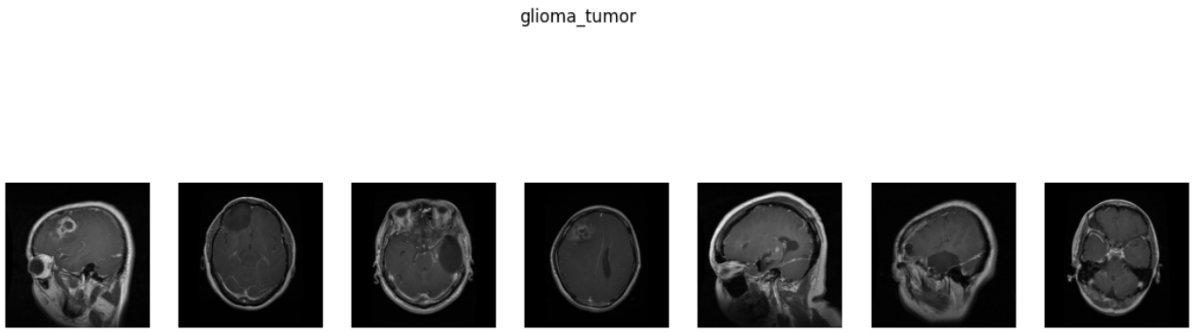


Validation Class Distribution

Cerinta 4:

Pentru cerinta 4 am afisat din dataset-ul de training cate 7 imagini pentru fiecare clasa.

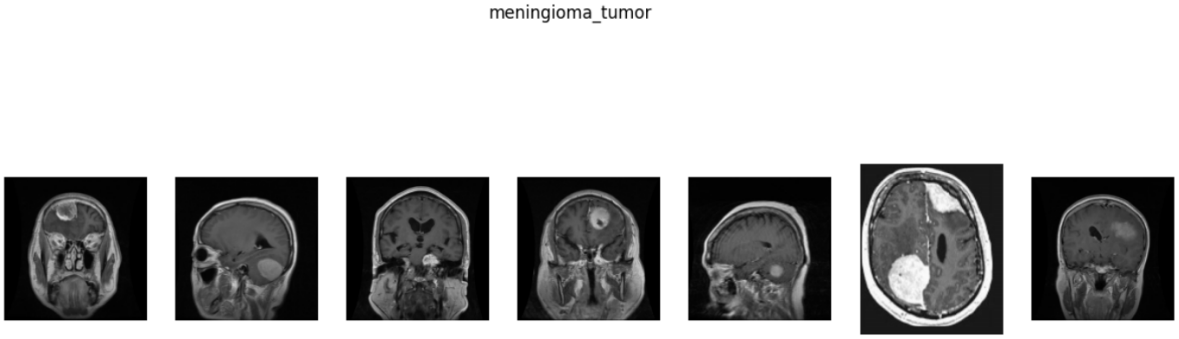
Glioma_tumor:



Se poate observa ca imaginile cu glioma tumor sunt vazute in celulele gliale in creier, acestea fiind mai mici, adica partile albe sunt vazute mai putin decat la alte tipuri de tumori. Se găsesc mai ales la nivelul substanței cenușii, în vecinătatea vaselor de sânge.

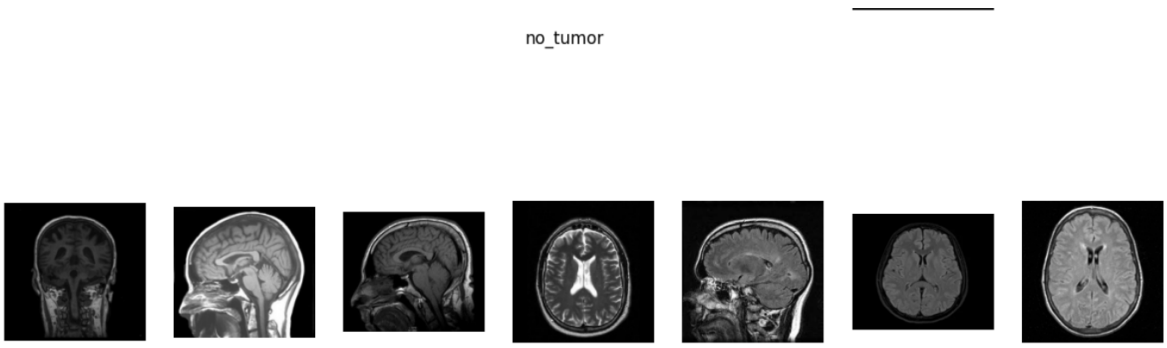
De aceea, ar fi greu sa fie confundate cu alte tipuri de tumori ale creierului.

Meningioma_tumor:



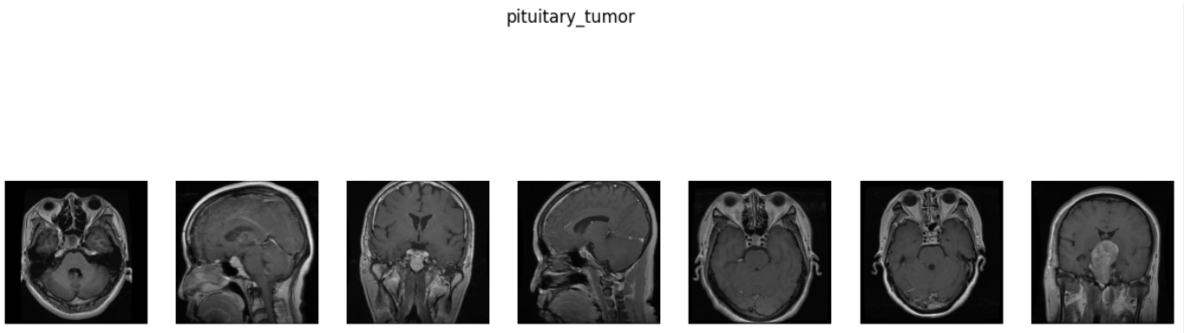
Se poate observa ca imaginile cu meningioma_tumor sunt vazute mai mari. Meningele este învelișul encefalului și a măduvei spinării si format din trei membrane conjunctive. De aceea, dupa cum se poate remarca, tumorile de tip meningioma sunt vizualizate la marginea craniului. Aceasta este o diferenta majora fata de alte tipuri de tumori si este mai usor pentru un algoritm de predictie sa isi dea seama de acest tip de tumoare fata de alte tumori. Tumoare este vazuta destul de usor, fiind mai luminoasa fata de alte tipuri de tumori.

No_tumor:



In aceste imagini se poate observa faptul ca este un creier sanatos, nu are nicio pata mai alba care ar putea da ideea de o anumita tumoare. Ajuta aceasta clasa pentru a face diferenta intre un creier sanatos si unul cu tumori, fiind un baseline foarte bun.

Pituitary_tumor:



Glanda pituitară este un organ de dimensiuni mici (aproximativ cât un bob de mazăre), situat la baza creierului, imediat sub hipotalamus (in spatele ochilor). Deoarece este foarte mica ca dimensiuni, la anumite unghiuri poate sa nu fie remarcata si sa dea impresia de un creier sanatos fara niciun tip de tumoare. Dar, se poate observa ca tumoarea apare mai luminoasa.

Cerinta 5:

Pentru cerinta 5 am verificat consistenta si integritatea setului de date dat, avand 3 criterii:

- Daca imaginiile sunt greyscale sau RGB
- Daca au dimesiuni uniforme sau nu (256x256 pixeli)
- Daca valorile pixelilor sunt exprimate pe aceleasi scale sau unitati

Am realizat aceste verificari pentru ambele seturi de date training/testing.

- **RGB/Greyscale**: Pentru aceasta verificare am parcurs fiecare imagine din setul de date si am verificat ce mod au. Daca aveau mode-ul 'L' erau greyscale si adunam la variable nrGreyScale, altfel era adunata la nrRGB.

Pentru training set la final erau:

There are 3265 images with RGB

There are 0 images with GreyScale

Pentru testing set la final erau:

There are 394 images with RGB

There are 0 images with GreyScale

- **Dimensiuni uniforme:** Pentru a vedea daca sunt dimensiuni uniforme am creat un dictionar care avea ca si key dimensiuniile imaginii, iar ca valori cate din acestea au acea dimensiune.

Pentru training set la final erau extrem de multe chei, adica multe imagini cu dimensiuni diferite:

{(512, 512): 2314, (474, 474): 2, (1365, 1365): 1, (256, 256): 16, (940, 900): 1, (721, 903): 1, (470, 432): 1, .. }

➔ Asadar am aplicat o transformare de resize la 256x256 pe toate imaginile.

Pentru testing set la final erau extrem de multe chei, adica multe imagini cu dimensiuni diferite:

{(892, 826): 3, (913, 875): 1, (512, 512): 32, (502, 502): 3, (1335, 1302): 3, (488, 512): 3, (442, 442): 7, (441, 429): 2, …}

➔ Asadar am aplicat o transformare de resize la 256x256 pe toate imaginile.

- Valorile pixelilor: Pentru valorile pixelilor, am transformat imaginea intr-un array de pixeli si am facut media tuturor pixelilor din imaginea respective. Aceasta valoare a fost appenduita la o lista de pixeli. Dupa prelucrarea tuturor imaginilor am luat lista ‘pixels’ si am luat minimul si maximul din acea lista si le am afisat.

Pentru training set:

Cea mai mare media a pixelilor a unei imagini este 137.76430335097

Cea mai mica media a pixelilor a unei imagini este 9.768336756651017

Pentru testing set la final:

Cea mai mare media a pixelilor a unei imagini este 137.76430335097

Cea mai mica media a pixelilor a unei imagini este 17.430407928756104

Acestea sunt ok ca medii, deci nu am realizat nicio transformare.

Cerinta 6:

Pentru cerinta 6 am implementat 5 tipuri de preprocesare si normalizare a imaginilor mri din acest set de date.

Tehniciile folosite au fost:

- Filtrul gaussian:

```
def filtru_Gaussian(image):
  blurrer = v2.GaussianBlur(kernel_size=(5, 5), sigma=(0.1, 3.))
  new_image = blurrer(image)

  new_image_np = new_image.permute(1, 2, 0).numpy()

  return new_image_np
```

Filtrul Gaussian este utilizat pentru a efectua **netezirea** sau **blurarea** imaginii. Este folosit pentru a elimina zgomotul dintr-o imagine, lasand componente mari in prim plan.

- Normalizarea intensitatii:

```
def norm_intensity(image):
  norm = v2.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])

  new_image = norm(image)
  new_image_np = new_image.permute(1, 2, 0).numpy()

  return new_image_np
```

Normalizarea intensitatii utilizata pentru a ajusta valorile pixelilor dintr-o imagine sau pentru a echilibra distributia intensitatilor. Folosita pentru imbunatatirea performantei algoritmilor care operează asupra imaginilor.

- Ajustarea constrastului(CLAHE):

```
def clahe(image):
  clh = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))

  new_image_np = image.permute(1, 2, 0).numpy()

  if new_image_np.ndim == 3:
    new_image_np = cv2.cvtColor(new_image_np, cv2.COLOR_RGB2GRAY)
```

```
  new_image_np = (new_image_np * 255).astype('uint8')
  new_image = clh.apply(new_image_np)

  return new_image
```

Clahe este folosit pentru imbunatatirea a contrastului imaginii, care rezolva problemele metodei clasice de egalizare a histogramei. Este folosita pentru a imbunatati detaliile dintr-o imagine fara a amplifica excesiv zgomotul sau artefactele.

- Flitru de detectie a marginilor (Sobel):

```
def sobel(image):
  image_np = image.permute(1, 2, 0).numpy()
  image_gray = cv2.cvtColor(image_np, cv2.COLOR_RGB2GRAY)
  sobel_x = cv2.Sobel(image_gray, cv2.CV_64F, 1, 0, ksize=5)
  sobel_y = cv2.Sobel(image_gray, cv2.CV_64F, 0, 1, ksize=5)

  new_image = np.sqrt(sobel_x**2 + sobel_y**2)

  return new_image
```

Dupa nume, este folosit pentru detectarea marginilor. Ajuta la identificarea formelor, delimitarea obiectelor.

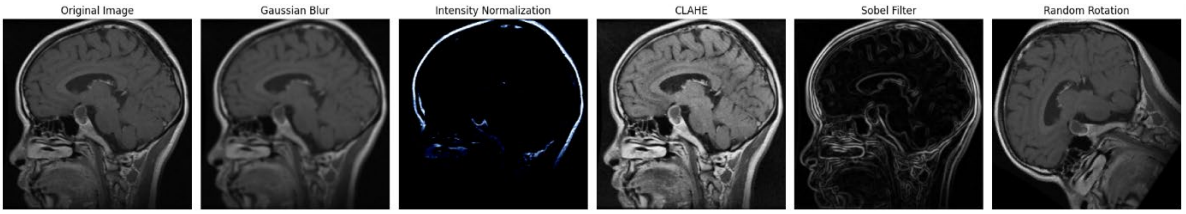
- Random Rotate:

```
def randomRot(image):
  rotate = transforms.RandomRotation(degrees=(0, 180))
  new_image = rotate(image)

  new_image_np = new_image.permute(1, 2, 0).numpy()

  return new_image_np
```

Folosit in principal la preprocesarea datelor pentru invatarea automata. Scopul principal este augumentarea datelor si imbunatatirea robustetii modelelor.


Original Image
Gaussian Blur
Intensity Normalization
CLAHE
Sobel Filter
Random Rotation