```
In [ ]: #!/usr/bin/env python
        # coding: utf-8
```

```
In [ ]: # 3D MRI classification pipeline using SimpleITK and MONAI to classify patient age using volumetric T1-weighted brain images.
        # data loading, metadata extraction, spatial normalization (center crop 70×70×70), intensity scaling, 3D augmentations, trained on a DenseNet-3D model
        # Evaluated with accuracy, confusion matrix, and ROC metrics.

        import monai
        from monai.networks.nets import DenseNet , resnet50
        from monai.transforms import Compose, ToTensor, AddChannel, RandRotate90, RandFlip, RandZoom , SpatialPad,CenterSpatialCrop
        from monai.transforms import (
            Activations,
            EnsureChannelFirst,
            AsDiscrete,
            Compose,
            LoadImage,
            RandFlip,
            RandRotate,
            RandZoom,
            ScaleIntensity,
        )

        from monai.data import ArrayDataset, DataLoader
        from torch.optim import Adam
        from torch.nn import CrossEntropyLoss
        import torch
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
        import numpy as np
        import os
        import SimpleITK as sitk
        from monai.data import decollate_batch, DataLoader
        from monai.metrics import ROCAUCMetric

        import pandas as pd
```

```
In [ ]: # label 0 and label 1?
        # see if we can identify whether age of person is above or below certain threshold

        groups = [0, 1]

        images = []
        labels = []

        paths = []
        images.append([]) #img array
        images.append([]) # label
        images.append([]) # subject id
        images.append([]) # img name + adr

        df = pd.read_csv('/home/cgelli/code/imagedata.csv')
        df.head()

        for j in groups:

            path = '/data/mengjin/Longi_T1_2GO_QC/zahra_dataset/dataset/'+str(j)
            lst = os.listdir(path)
            number_files = len(lst)
            paths.append(lst)
            for i in os.listdir(path):
                x = os.path.join(path, i)
                img = sitk.ReadImage( x , sitk.sitkFloat32)
                patient_id = i[0:10]
                #print(patient_id)
                date = i[22:32]
                #print(date)
                # find patient and date match for image
                filtered_df = df[(df['PTID'] == patient_id) & (df['EXAMDATE_MERGE'] == date)]
                age = filtered_df['AGE'].values
                if not filtered_df.empty:
                    age_dec = age[0]
                    over_75 = 0
                    if age_dec > 75:
                        over_75 = 1

                    # label 0 if under 75, else 1
                    img = sitk.GetArrayFromImage(img)
                    images[0].append(img)
                    images[1].append(over_75)
                    images[2].append(i[0:10])
                    images[3].append(x)
```

```
In [ ]: import torch
        import numpy as np
        from sklearn.model_selection import train_test_split
        from skimage.transform import resize
        from monai.transforms import EnsureChannelFirst
```

```
In [ ]: target_size = (70, 70, 70)

        # def resize_img(img, target_shape):
        #     return resize(img, target_shape, mode='constant', preserve_range=True)

        # resized_images = [resize_img(img, target_size) for img in images[0]]

        images_array = images[0]
        labels_array = images[1]
```

```
In [ ]: images_tensor = images_array #torch.tensor(images_array, dtype=torch.float32)
        labels_tensor = labels_array #torch.tensor(labels_array, dtype=torch.float32)

        train_images, test_images, train_labels, test_labels = train_test_split(images_tensor, labels_tensor, test_size=0.2, random_state=42)

        train_transforms = Compose([
            EnsureChannelFirst(channel_dim='no_channel'),
            ScaleIntensity(),
            CenterSpatialCrop((70, 70, 70)),
            RandRotate90(prob=0.5),
            RandFlip(spatial_axis=0, prob=0.5),
            RandZoom(prob=0.5, min_zoom=0.9, max_zoom=1.1),
            ToTensor()
        ])

        test_transforms = Compose([
            EnsureChannelFirst(channel_dim='no_channel'),
            ScaleIntensity(),
            CenterSpatialCrop((70, 70, 70)),
            ToTensor()
        ])
```

```
In [ ]: from monai.data import ArrayDataset, DataLoader

        train_dataset = ArrayDataset(train_images, train_transforms, train_labels)
        test_dataset = ArrayDataset(train_images, test_transforms, test_labels)

        train_loader = DataLoader(train_dataset, batch_size=2, shuffle=True)
        test_loader = DataLoader(test_dataset, batch_size=2, shuffle=False)
```

```
In [ ]: from monai.networks.nets import DenseNet , resnet50
        from monai.transforms import Compose, ToTensor, AddChannel, RandRotate90, RandFlip, RandZoom , SpatialPad,CenterSpatialCrop
        from monai.transforms import (
            Activations,
            EnsureChannelFirst,
            AsDiscrete,
            Compose,
            Resize,
            LoadImage,
            RandFlip,
            RandRotate,
            RandZoom,
            ScaleIntensity,
        )
        from monai.data import ArrayDataset, DataLoader
        from torch.optim import Adam
        from torch.nn import CrossEntropyLoss
        import torch
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
        import numpy as np
        import os
        import SimpleITK as sitk
        from monai.data import decollate_batch, DataLoader
        from monai.metrics import ROCAUCMetric
```

```
In [ ]: print(train_dataset)
```

```
In [ ]: model = DenseNet(spatial_dims=3, in_channels=1, out_channels=2)
        loss_function = CrossEntropyLoss()
        optimizer = Adam(model.parameters(), lr=0.0001)

        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        model = model.to(device)


        val_interval = 1
        metric_values = []


        num_epochs = 10


        for epoch in range(num_epochs):
            model.train()
            step = 0
            epoch_loss = 0
            train_epoch_loss = 0
            for batch_data, batch_labels in train_loader:
                batch_data, batch_labels = batch_data.to(device), batch_labels.to(device)

                optimizer.zero_grad()
                outputs = model(batch_data)
                loss = loss_function(outputs, batch_labels)
                train_epoch_loss += loss.item()
                loss.backward()
                optimizer.step()

            train_loss = train_epoch_loss / len(train_loader)
            print ('train_loss: ' + str(train_loss))
```

```
In [ ]: model.eval()
        test_labels = []
        test_preds = []

        with torch.no_grad():
            for batch_data, batch_labels in test_loader:
                batch_data, batch_labels = batch_data.to(device), batch_labels.to(device)

                outputs = model(batch_data)
                _, preds = torch.max(outputs, 1)

                test_labels.extend(batch_labels.cpu().numpy())
                test_preds.extend(preds.cpu().numpy())

        accuracy = accuracy_score(test_labels, test_preds)
        cm = confusion_matrix(test_labels, test_preds)
        report = classification_report(test_labels, test_preds)

        print(f'Test Accuracy: {accuracy}')
        print('Confusion Matrix:')
        print(cm)
        print('Classification Report:')
        print(report)

        # return (accuracy, cm)
```

```
In [ ]:
```