Great! Based on your preferences, I will generate a Python code pipeline using PyTorch that includes data augmentation, utilizes a pre-trained model, saves evaluation metrics to a file, and includes GPU-specific configurations. Below is the code:

```python

import os

import numpy as np

import pandas as pd

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision.transforms as transforms

from torchvision import datasets, models

from torch.utils.data import DataLoader, random\_split

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, mean\_squared\_error

from PIL import Image

import pydicom

# Define constants

DATA\_DIR = 'path/to/your/dicom/images' # Update this path

BATCH\_SIZE = 32

EPOCHS = 10

LEARNING\_RATE = 0.001

DEVICE = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# Data augmentation and preprocessing

transform = transforms.Compose([

transforms.Resize((224, 224)), # Resize to match the input size of the pre-trained model

transforms.RandomHorizontalFlip(), # Data augmentation

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) # Normalize for pre-trained models

])

# Custom dataset class for DICOM images

class DICOMDataset(torch.utils.data.Dataset):

def \_\_init\_\_(self, data\_dir, transform=None):

self.data\_dir = data\_dir

self.transform = transform

self.images = []

self.labels = []

# Load images and labels

for label in ['tumor\_present', 'tumor\_not\_present']:

label\_dir = os.path.join(data\_dir, label)

for filename in os.listdir(label\_dir):

if filename.endswith('.dcm'):

self.images.append(os.path.join(label\_dir, filename))

self.labels.append(1 if label == 'tumor\_present' else 0)

def \_\_len\_\_(self):

return len(self.images)

def \_\_getitem\_\_(self, idx):

img\_path = self.images[idx]

img = pydicom.dcmread(img\_path).pixel\_array

img = Image.fromarray(img).convert('RGB') # Convert to RGB

if self.transform:

img = self.transform(img)

label = self.labels[idx]

return img, label

# Load dataset

dataset = DICOMDataset(DATA\_DIR, transform)

train\_size = int(0.8 \* len(dataset))

val\_size = int(0.1 \* len(dataset))

test\_size = len(dataset) - train\_size - val\_size

train\_dataset, val\_dataset, test\_dataset = random\_split(dataset, [train\_size, val\_size, test\_size])

train\_loader = DataLoader(train\_dataset, batch\_size=BATCH\_SIZE, shuffle=True)

val\_loader = DataLoader(val\_dataset, batch\_size=BATCH\_SIZE, shuffle=False)

test\_loader = DataLoader(test\_dataset, batch\_size=BATCH\_SIZE, shuffle=False)

# Load pre-trained model

model = models.resnet18(pretrained=True)

model.fc = nn.Linear(model.fc.in\_features, 2) # Adjust the final layer for binary classification

model = model.to(DEVICE)

# Define loss function and optimizer

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=LEARNING\_RATE)

# Training loop

for epoch in range(EPOCHS):

model.train()

running\_loss = 0.0

for images, labels in train\_loader:

images, labels = images.to(DEVICE), labels.to(DEVICE)

optimizer.zero\_grad()

outputs = model(images)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

print(f'Epoch [{epoch+1}/{EPOCHS}], Loss: {running\_loss/len(train\_loader):.4f}')

# Evaluation function

def evaluate(model, data\_loader):

model.eval()

all\_preds = []

all\_labels = []

with torch.no\_grad():

for images, labels in data\_loader:

images, labels = images.to(DEVICE), labels.to(DEVICE)

outputs = model(images)

\_, preds = torch.max(outputs, 1)

all\_preds.extend(preds.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

return np.array(all\_labels), np.array(all\_preds)

# Evaluate on validation set

val\_labels, val\_preds = evaluate(model, val\_loader)

# Calculate metrics

accuracy = accuracy\_score(val\_labels, val\_preds)

precision = precision\_score(val\_labels, val\_preds)

recall = recall\_score(val\_labels, val\_preds)

f1 = f1\_score(val\_labels, val\_preds)

rmse = np.sqrt(mean\_squared\_error(val\_labels, val\_preds))

# Save metrics to a file

metrics = {

'Accuracy': accuracy,

'Precision': precision,

'Recall': recall,

'F1 Score': f1,

'RMSE': rmse

}

metrics\_df = pd.DataFrame(metrics, index=[0])

metrics\_df.to\_csv('evaluation\_metrics.csv', index=False)

print("Evaluation metrics saved to 'evaluation\_metrics.csv'")

```

### Explanation of the Code:

1. \*\*Imports\*\*: Necessary libraries are imported, including PyTorch and torchvision for model building and data handling.

2. \*\*Constants\*\*: Constants for data directory, batch size, epochs, learning rate, and device configuration are defined.

3. \*\*Data Augmentation and Preprocessing\*\*: A series of transformations are applied to the images, including resizing, random horizontal flipping, and normalization.

4. \*\*Custom Dataset Class\*\*: A custom dataset class is created to load DICOM images and their corresponding labels.

5. \*\*Data Loading\*\*: The dataset is split into training, validation, and test sets, and DataLoader objects are created for each.

6. \*\*Model Setup\*\*: A pre-trained ResNet-18 model is loaded, and the final layer is adjusted for binary classification.

7. \*\*Training Loop\*\*: The model is trained over the specified number of epochs, and the loss is printed for each epoch.

8. \*\*Evaluation Function\*\*: A function is defined to evaluate the model on a given dataset and calculate predictions.

9. \*\*Metrics Calculation\*\*: Evaluation metrics (accuracy, precision, recall, F1 score, RMSE) are calculated on the validation set.

10. \*\*Metrics Saving\*\*: The metrics are saved to a CSV file for later analysis.

### Note:

- Make sure to update the `DATA\_DIR` variable with the correct path to your DICOM images.

- Ensure that you have the required libraries installed (`torch`, `torchvision`, `pydicom`, `PIL`, `pandas`, `scikit-learn`).

- You may need to adjust the number of epochs and learning rate based on your dataset and training results.

Feel free to ask if you have any questions or need further modifications!