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)
```

```python
    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!