DOCUMENTATION-3

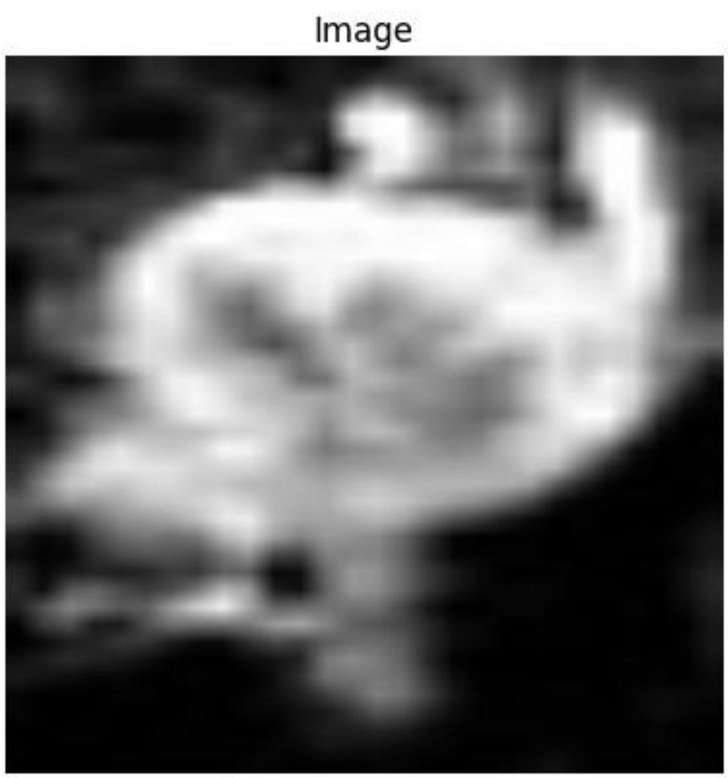
Name: Jaya Madhav
Mail: madhavjaya4@gmail.com

GLCM(Horizontal)

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

def calculate_glcm(matrix, dx=1, dy=0, intensity_levels=[0, 1, 2, 3]):
    num_levels = len(intensity_levels)
    glcm = np.zeros((num_levels, num_levels), dtype=int)

    rows, cols = matrix.shape

    for i in range(rows):
        for j in range(cols):
            if 0 <= i + dy < rows and 0 <= j + dx < cols:
                current_value = matrix[i, j]
                right_value = matrix[i + dy, j + dx]
                glcm[current_value, right_value] += 1

    return glcm

# List of full image paths
image_paths = [
        r"/kaggle/input/breast-cancer-mri/train/Malignant/BreaDM-Ma-
1926/SUB1/p-040.jpg",
        r"/kaggle/input/breast-cancer-mri/train/Malignant/BreaDM-Ma-
2024/SUB4/p-074.jpg",
        r"/kaggle/input/breast-cancer-mri/train/Malignant/BreaDM-Ma-
1803/SUB4/p-046.jpg",
        r"/kaggle/input/breast-cancer-mri/train/Malignant/BreaDM-Ma-
1802/SUB3/p-044.jpg",
        r"/kaggle/input/breast-cancer-mri/train/Malignant/BreaDM-Ma-
1818/SUB8/p-029.jpg"
]

# Parameters for GLCM calculation
dx, dy = 1, 0  # Horizontal offset
num_levels = 4 # Number of intensity levels
intensity_levels = list(range(num_levels))

# Loop through each image path
for image_path in image_paths:
    # Load the image in grayscale
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        print(f"Could not load image: {image_path}")
```

DOCUMENTATION-3

```
        continue

    # Resize the original image to 224x224
    resized_image = cv2.resize(image, (224, 224))

    # Display the resized image
    plt.imshow(resized_image, cmap='gray')
    plt.title("Image")
    plt.axis('off')
    plt.show()

    # Normalize pixel values for GLCM calculation
    normalized_image = (image / (256 / num_levels)).astype(int)

    # Calculate the GLCM
    glcm = calculate_glcm(normalized_image, dx=dx, dy=dy,
intensity_levels=intensity_levels)

    # Print the GLCM matrix
    print("GLCM Matrix (horizontal):")
    print(glcm)
    print("\n" + "="*50 + "\n")
```



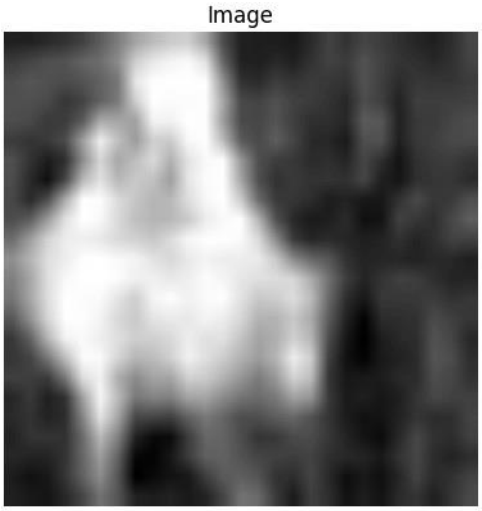

```
GLCM Matrix (horizontal):
[[434  48   7   5]
```

DOCUMENTATION-3

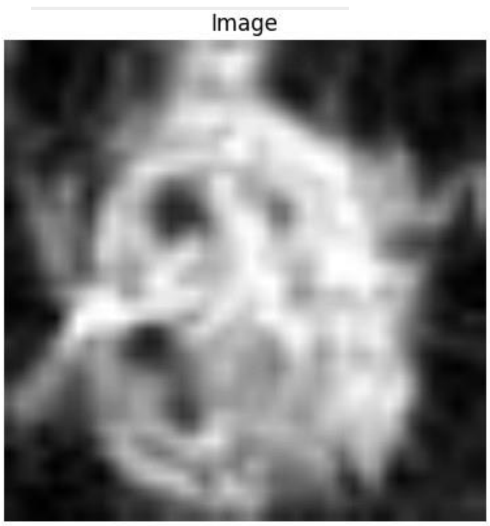
```
 [ 56  76  43   4]
 [  4  46 111  44]
 [  0   9  44 176]]
==================================================
```

```
GLCM Matrix (horizontal):

 [ 80 400  43   0]
 [  0  48  56  36]
 [  0   6  30 114]]

==================================================
```

DOCUMENTATION-3




```
GLCM Matrix (horizontal):
[[172  57   3   0]
 [ 55  87  18   3]
 [  5  18  23  30]
 [  0   3  30 120]]

==================================================
```




```
GLCM Matrix (horizontal):
[[480  60   5   0]
 [ 65 140  69   3]
 [  1  73 274  67]
 [  0   1  69 292]]

==================================================
```

DOCUMENTATION-3


```
GLCM Matrix (horizontal):
[[222  53   1   0]
 [ 55 101  26   3]
 [  0  26  38  23]
 [  0   4  22 151]]

==================================================
```

CustomVGG16

```
import torch
import torch.nn as nn
import torchvision.models as models

# Custom VGG16 class that inherits from nn.Module
class CustomVGG16(nn.Module):
    def___init__(self, num_classes=2):
        super(CustomVGG16, self).__init__()

        vgg16 = models.vgg16()

vgg16.load_state_dict(torch.load('/kaggle/input/customvgg16/pytorch/
default/1/vgg16-397923af.pth', weights_only=True))
```

DOCUMENTATION-3

```
        # Extract the features and avgpool layers from the pretrained
model
        self.features = vgg16.features
        self.avgpool = vgg16.avgpool

        # Define a new classifier using nn.Sequential
        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, num_classes)
        )

    def forward(self, x):
        # Pass the input through the feature layers
        x = self.features(x)
        # Apply the average pooling layer
        x = self.avgpool(x)
        # Flatten the output to a 2D tensor
        x = torch.flatten(x, 1)
        # Pass the reshaped output through the custom classifier
        x = self.classifier(x)
        return x

# Printing the model
if___name___== "___main__":
    model = CustomVGG16(num_classes=2)
    print(model)

CustomVGG16(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (8): ReLU(inplace=True)
```

DOCUMENTATION-3

```
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=2, bias=True)
  )
)
```

```
import torch
```

DOCUMENTATION-3

```
# Ensure that the CustomVGG16 class is already defined above or
imported
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Initialize the model
model = CustomVGG16(num_classes=2)

# Move the model to the appropriate device (CPU or GPU)
model = model.to(device)

# Print the model to confirm
print(model)
```

```
CustomVGG16(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
```

DOCUMENTATION-3

```
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=2, bias=True)
  )
)
```

EarlyStopping

```
import numpy as np
import torch

class EarlyStopping:
    """Early stops the training if validation loss doesn't improve
after a given patience."""
    def___init__(self, patience=7, verbose=False, delta=0,
path='checkpoint.pt', trace_func=print):

        self.patience = patience
        self.verbose = verbose
        self.counter = 0
        self.best_score = None
        self.early_stop = False
        self.val_loss_min = np.inf
        self.delta  =  delta
        self.path = path
        self.trace_func = trace_func
```

DOCUMENTATION-3

```
    def___call__(self, val_loss, model):
        score = -val_loss

        if self.best_score is None:
            self.best_score = score
            self.save_checkpoint(val_loss, model)
        elif score < self.best_score + self.delta:
            self.counter += 1
            if self.verbose:
                self.trace_func(f'EarlyStopping counter: 
{self.counter} out of {self.patience}')
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_score = score
            self.save_checkpoint(val_loss, model)
            self.counter = 0

    def save_checkpoint(self, val_loss, model):
        if self.verbose:
            self.trace_func(f'Validation loss decreased 
({self.val_loss_min:.6f} --> {val_loss:.6f}).  Saving model ...')
        torch.save(model.state_dict(), self.path)
        self.val_loss_min = val_loss

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

Train Function

```
import torch

from tqdm import tqdm

import torch.nn.functional as F

def train(epoch, model, num_epochs, loader, criterion, l2_decay):
    learning_rate = max(lr * (0.1**(epoch / 10)), 1e-5)
    optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, 
momentum=0.9, weight_decay=l2_decay)

    model.train()
    correct = 0

    for data, label in tqdm(loader, desc=f'Epoch 
{epoch+1}/{num_epochs}', unit='batch'):
        data = data.float().cuda()
        label = label.long().cuda()
```

```
        output = model(data)
        optimizer.zero_grad()
        loss = F.nll_loss(F.log_softmax(output, dim=1), label)
        loss.backward()
        optimizer.step()

        pred = output.data.max(1)[1]
        correct += pred.eq(label.data.view_as(pred)).cpu().sum()

    print(f'train accuracy: {100. * correct / len(loader.dataset)}%')
```

Validation Function

```
from sklearn import metrics
from sklearn.metrics import roc_auc_score, roc_curve
import numpy as np

def val(model):
    name = 'validation'
    len_valloader = len(valloader.dataset)
    model.eval()
    val_loss = 0
    correct = 0
    all_predictions = []
    all_targets = [] # To store all ground truth labels across
batches
    possibilities = None

    for data, target in valloader: # Use valloader as the validation
data loader
        if torch.cuda.is_available():
            data, target = data.cuda(), target.cuda()

        val_output = model(data)
        val_loss += F.nll_loss(F.log_softmax(val_output, dim=1),
target, reduction='sum').item()
        pred = val_output.data.max(1)[1]

        # Collect predictions and targets
        all_predictions.extend(pred.cpu().numpy())
        all_targets.extend(target.cpu().numpy())

        # Convert the output to probabilities using softmax
        possibility = F.softmax(val_output, dim=1).cpu().data.numpy()
        if possibilities is None:
            possibilities = possibility
        else:
```

DOCUMENTATION-3

```
            possibilities = np.concatenate((possibilities,
possibility), axis=0)

        correct += pred.eq(target.data.view_as(pred)).cpu().sum()

    # Compute the confusion matrix using all_targets and
all_predictions
    cm = metrics.confusion_matrix(all_targets, all_predictions)

    # One-hot encode the labels for AUC computation
    num_classes  =  val_output.shape[1]
    target_onehot = np.eye(num_classes)
[np.array(all_targets).astype(int).tolist()]

    val_loss /= len_valloader
    fpr, tpr, _ = roc_curve(all_targets, possibilities[:, 1])  #
Assuming binary classification
    auc_value = roc_auc_score(all_targets, possibilities[:, 1])

    print('Specificity: {:.4f}, Sensitivity: {:.4f}, AUC:
{:.4f}'.format(1 - fpr[0], tpr[0], auc_value))
    print('\n{} set: Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%)\
n'.format(
        name, val_loss, correct, len_valloader,
        100. * correct / len_valloader))

    return 100. * correct / len_valloader, val_loss, auc_value
```

Model Training

```
total_epochs = 50
lr = 0.01
best_accuracy = 0
momentum = 0.9
no_cuda = False
num_classes = 2
log_interval = 10
l2_decay = 0.01

trainloader =
torch.utils.data.DataLoader(train_ds,batch_size=batch_size,shuffle=Tru
e,num_workers=2)
valloader =
torch.utils.data.DataLoader(val_ds,batch_size=batch_size,shuffle=True,
num_workers=2)

criterion = torch.nn.CrossEntropyLoss()
```

DOCUMENTATION-3

```
model = CustomVGG16(num_classes=num_classes)

model = model.to(device)
# model.to(device)  # here cuda device
num_classes = 2
# model = CustomVGG16(num_classes=num_classes)



import os
# Model training
model.to(device)  # here device is cuda
highest_auc = 0
early_stopping = EarlyStopping(patience=20, verbose=True)

project_directory = 'tumor_classification'
model_type = 'vgg16'

# Directory to save model checkpoints
save_directory = os.path.join('model', project_directory, model_type)
if not os.path.exists(save_directory):
    os.makedirs(save_directory)

# Training loop for a specified number of epochs
for current_epoch in range(1, total_epochs + 1):
    # Now pass all required arguments to the train function
    train(current_epoch, model, total_epochs, trainloader, criterion,
l2_decay)

    # Evaluation with no gradient calculations
    with torch.no_grad():
        accuracy, val_loss, auc_score = val(model)

        # Handle multi-GPU setups if applicable
        model_state_dict = (
            model.module.state_dict()
            if isinstance(model, nn.parallel.DistributedDataParallel)
            else model.state_dict()
        )

        # Save the model if AUC score improves
        if auc_score > highest_auc:
            highest_auc = auc_score  # Update highest AUC score
            checkpoint_path = os.path.join(save_directory,
f'{model_type}_{current_epoch}.pth')
            torch.save(model_state_dict, checkpoint_path,
_use_new_zipfile_serialization=False)
            print(f"Model checkpoint saved at epoch {current_epoch}
with AUC: {highest_auc:.4f}")
```

DOCUMENTATION-3

```
        # Early stopping based on validation loss
        early_stopping(val_loss, model) # test_loss should be updated 
in each validation
        if early_stopping.early_stop:
            print("Early stopping triggered")
            break
```

```
Epoch 2/50: 100%|██████████| 639/639 [02:40<00:00, 3.97batch/s]

train accuracy: 72.79534149169922%


Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.8605

validation set: Average loss: 0.5209, Accuracy: 1581/1989 (79.49%)

Model checkpoint saved at epoch 1 with AUC: 0.8605
Validation loss decreased (inf --> 0.520908). Saving model ...

Epoch 3/50: 100%|██████████| 639/639 [02:40<00:00, 3.97batch/s]

train accuracy: 72.79534149169922%


Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.7676

validation set: Average loss: 0.5126, Accuracy: 1581/1989 (79.49%)

Validation loss decreased (0.520908 --> 0.512619). Saving model ...

Epoch 4/50: 100%|██████████| 639/639 [02:40<00:00,  3.97batch/s]

train accuracy: 72.79534149169922%


Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000

validation set: Average loss: 0.5189, Accuracy: 1581/1989 (79.49%)

EarlyStopping counter: 1 out of 20

Epoch 5/50: 100%|██████████| 639/639 [02:40<00:00,  3.97batch/s]

train accuracy: 72.79534149169922%


Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000

validation set: Average loss: 0.5177, Accuracy: 1581/1989 (79.49%)
```

DOCUMENTATION-3

```
EarlyStopping counter: 2 out of 20
Epoch 6/50: 100%|██████████| 639/639 [02:40<00:00, 3.97batch/s]
train accuracy: 72.79534149169922%

Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000
validation set: Average loss: 0.5226, Accuracy: 1581/1989 (79.49%)
EarlyStopping counter: 3 out of 20
Epoch 7/50: 100%|██████████| 639/639 [02:41<00:00,  3.97batch/s]
train accuracy: 72.79534149169922%

Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000
validation set: Average loss: 0.5226, Accuracy: 1581/1989 (79.49%)
EarlyStopping counter: 4 out of 20
Epoch 8/50: 100%|██████████| 639/639 [02:41<00:00,  3.96batch/s]
train accuracy: 72.79534149169922%

Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000
validation set: Average loss: 0.5198, Accuracy: 1581/1989 (79.49%)
EarlyStopping counter: 5 out of 20
Epoch 9/50: 100%|██████████| 639/639 [02:41<00:00,  3.96batch/s]
train accuracy: 72.79534149169922%

Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000
validation set: Average loss: 0.5201, Accuracy: 1581/1989 (79.49%)
EarlyStopping counter: 6 out of 20
Epoch 10/50: 100%|██████████| 639/639 [02:41<00:00,  3.96batch/s]
train accuracy: 72.79534149169922%
```

DOCUMENTATION-3

```
Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000

validation set: Average loss: 0.5213, Accuracy: 1581/1989 (79.49%)

EarlyStopping counter: 7 out of 20

Epoch 11/50: 100%|██████████| 639/639 [02:40<00:00, 3.97batch/s]

train accuracy: 72.79534149169922%


Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000

validation set: Average loss: 0.5224, Accuracy: 1581/1989 (79.49%)

EarlyStopping counter: 8 out of 20

Epoch 12/50: 100%|██████████| 639/639 [02:41<00:00, 3.97batch/s]

train accuracy: 72.79534149169922%


Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000

validation set: Average loss: 0.5214, Accuracy: 1581/1989 (79.49%)

EarlyStopping counter: 9 out of 20

Epoch 13/50: 100%|██████████| 639/639 [02:40<00:00, 3.97batch/s]

train accuracy: 72.79534149169922%


Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000

validation set: Average loss: 0.5208, Accuracy: 1581/1989 (79.49%)

EarlyStopping counter: 10 out of 20

Epoch 14/50: 100%|██████████| 639/639 [02:40<00:00, 3.97batch/s]

train accuracy: 72.79534149169922%


Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000

validation set: Average loss: 0.5211, Accuracy: 1581/1989 (79.49%)
```

DOCUMENTATION-3

```
EarlyStopping counter: 11 out of 20
Epoch 15/50: 100%|██████████| 639/639 [02:40<00:00,  3.97batch/s]
train accuracy: 72.79534149169922%

Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000
validation set: Average loss: 0.5207, Accuracy: 1581/1989 (79.49%)
EarlyStopping counter: 12 out of 20
Epoch 16/50: 100%|██████████| 639/639 [02:41<00:00,  3.97batch/s]
train accuracy: 72.79534149169922%

Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000
validation set: Average loss: 0.5212, Accuracy: 1581/1989 (79.49%)
EarlyStopping counter: 13 out of 20
Epoch 17/50: 100%|██████████| 639/639 [02:40<00:00,  3.97batch/s]
train accuracy: 72.79534149169922%

Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000
validation set: Average loss: 0.5215, Accuracy: 1581/1989 (79.49%)
EarlyStopping counter: 14 out of 20
Epoch 18/50: 100%|██████████| 639/639 [02:40<00:00,  3.97batch/s]
train accuracy: 72.79534149169922%

Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000
validation set: Average loss: 0.5213, Accuracy: 1581/1989 (79.49%)
EarlyStopping counter: 15 out of 20
Epoch 19/50: 100%|██████████| 639/639 [02:40<00:00,  3.97batch/s]
train accuracy: 72.79534149169922%
```

DOCUMENTATION-3

```
Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000
validation set: Average loss: 0.5212, Accuracy: 1581/1989 (79.49%)
EarlyStopping counter: 16 out of 20
Epoch 20/50: 100%|██████████| 639/639 [02:41<00:00, 3.97batch/s]
train accuracy: 72.79534149169922%

Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000
validation set: Average loss: 0.5211, Accuracy: 1581/1989 (79.49%)
EarlyStopping counter: 17 out of 20
Epoch 21/50: 100%|██████████| 639/639 [02:41<00:00, 3.97batch/s]
train accuracy: 72.79534149169922%

Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000
validation set: Average loss: 0.5211, Accuracy: 1581/1989 (79.49%)
EarlyStopping counter: 18 out of 20
Epoch 22/50: 100%|██████████| 639/639 [02:40<00:00, 3.97batch/s]
train accuracy: 72.79534149169922%

Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000
validation set: Average loss: 0.5211, Accuracy: 1581/1989 (79.49%)
EarlyStopping counter: 19 out of 20
Epoch 23/50: 100%|██████████| 639/639 [02:40<00:00, 3.97batch/s]
train accuracy: 72.79534149169922%

Specificity: 1.0000, Sensitivity: 0.0000, AUC: 0.5000
validation set: Average loss: 0.5211, Accuracy: 1581/1989 (79.49%)
```

DOCUMENTATION-3

```
EarlyStopping counter: 20 out of 20
Early stopping triggered
```

Test Function

```
import torch
import torch.nn.functional as F
from sklearn import metrics
from sklearn.metrics import roc_auc_score, roc_curve, auc
import numpy as np

def test(model, testloader):  # Changed test_dataloader to testloader
    name = 'test'
    len_testloader = len(testloader.dataset) # Corrected reference to
testloader
    model.eval()
    test_loss = 0
    correct = 0
    possibilities = None  # Corrected variable name
    all_predictions = []
    label_names = ['benign', 'malignant']  # Labels for classification

    # Iterate over test data
    for data, target in testloader: # Changed test_dataloader to
testloader
        if torch.cuda.is_available():
            data, target = data.cuda(), target.cuda()

        # Forward pass
        test_output = model(data)  # Corrected variable name
        test_loss += F.nll_loss(F.log_softmax(test_output, dim=1),
target, reduction='sum').item()  # Corrected test_output

        # Get predictions
        _, pred = test_output.data.max(1)  # Getting class predictions
        all_predictions.append(pred.cpu().numpy())

        # Softmax probabilities for AUC
        possibility = F.softmax(test_output, dim=1).cpu().data.numpy()
# Corrected test_output
        if possibilities is None:
            possibilities = possibility
        else:
            possibilities = np.concatenate((possibilities,
possibility), axis=0)

        correct +=
pred.eq(target.data.view_as(pred)).cpu().sum().item()  # Accumulate
```

DOCUMENTATION-3

```
correct predictions

    # Flatten the list of predictions
    all_predictions = [i for item in all_predictions for i in item]

    # Classification metrics -> accuracy, f1 score
    print(metrics.classification_report(target.cpu().numpy(),
all_predictions, labels=range(2), target_names=label_names, digits=4))
# Corrected target variable

    # Confusion matrix
    cm = metrics.confusion_matrix(target.cpu().numpy(),
all_predictions, labels=range(2))  # Corrected target variable
    print('Confusion Matrix:')
    print(cm)

    # AUC calculation
    fpr, tpr, thresholds = roc_curve(target.cpu().numpy(),
possibilities.ravel())  # Corrected target variable
    auc_value = auc(fpr, tpr)

    # Output Specificity, Sensitivity, and AUC
    print(f'Specificity: {1 - fpr[0]:.4f}, Sensitivity: {tpr[0]:.4f},
AUC: {auc_value:.4f}')

    # Average loss and accuracy
    print(f'\n{name} set: Average loss: {test_loss /
len_testloader:.4f}, Accuracy: {correct}/{len_testloader} ({100. *
correct / len_testloader:.2f}%)\n')

    return 100. * correct / len_testloader, test_loss /
len_testloader, auc_value
```