

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
import torch
from torch import nn
import torchvision
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder
import torchvision.models as models
from torch.utils.data import TensorDataset, ConcatDataset, DataLoader, Sub
from sklearn.model_selection import KFold
import os
from torchinfo import summary
```

```
In [ ]: device = "cuda" if torch.cuda.is_available() else "cpu"
```

```
In [ ]: # ResNet 10 Approach
```

```
In [6]: import timm
class Resnet10(nn.Module):
    def __init__(self, num_classes=3):
        super(Resnet10, self).__init__()
        self.resnet10 = timm.create_model("resnet10t", pretrained=True)

        # Replace the fully connected layer for classification
        in_features = self.resnet10.get_classifier().in_features
        self.resnet10.fc = nn.Linear(in_features, num_classes)

    def forward(self, x):
        return self.resnet10(x)
```

```
In [8]: torch.cuda.empty_cache()
```

```
In [9]: encoder = Resnet10(num_classes=3).to(device)
```

```
In [13]: transformer=transforms.Compose([
    transforms.Resize((150,150)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(degrees=10),
    transforms.ToTensor(), #0-255 to 0-1, numpy to tensors
    transforms.Normalize([0.5,0.5,0.5], # 0-1 to [-1,1] , formula (x-mean
        [0.5,0.5,0.5])
])
```

```
In [14]: # Mentioning training path
train_path = '/kaggle/input/gsoc123/new_dataset/train'
```

```
In [15]: class_folders = torchvision.datasets.ImageFolder(train_path,transform=tra
```

```
In [16]: from torch.optim.lr_scheduler import StepLR
# Define the scheduler
```

```
In [17]: def CNN_train(epochs,class_folders, num_folds, lr):
    loss_fn = nn.CrossEntropyLoss()
    train_acc_values = []
    best_accuracy = 0.0
```

```

test_acc_values = []
num_epochs = epochs
epoch_count = []
iteration_details = []
optimizer = torch.optim.Adam(encoder.parameters(), lr=lr, weight_decay=weight_decay)
scheduler = StepLR(optimizer, step_size=10, gamma=0.1, verbose=True)

# Initialize KFold object
kf = KFold(n_splits=num_folds, shuffle=True)

# Loop through each fold
for fold, (train_index, val_index) in enumerate(kf.split(class_folders)):
    print(f"Fold {fold + 1}/{num_folds}")

    dataset_train = Subset(class_folders, train_index)
    dataset_valid = Subset(class_folders, val_index)

    train_loader = torch.utils.data.DataLoader(
        dataset_train, batch_size=64, shuffle=True
    )
    val_loader = torch.utils.data.DataLoader(
        dataset_valid, batch_size=32, shuffle=True
    )

    # Training loop
    for epoch in range(num_epochs):
        # Set model to training mode
        encoder.train()

        train_accuracy = 0.0
        train_loss = 0.0

        for i, (images, labels) in enumerate(train_loader):
            if torch.cuda.is_available():
                images = images.cuda()
                labels = labels.cuda()

            optimizer.zero_grad()
            outputs = encoder(images)
            loss = loss_fn(outputs, labels)
            loss.backward()
            optimizer.step()

            train_loss += loss.item() * images.size(0)
            _, prediction = torch.max(outputs.data, 1)
            train_accuracy += int(torch.sum(prediction == labels.data).item())

        train_accuracy = train_accuracy / len(train_index)
        train_loss = train_loss / len(train_index)

    # Validation loop
    encoder.eval()

    val_accuracy = 0.0
    val_loss = 0.0

    for i, (images, labels) in enumerate(val_loader):
        if torch.cuda.is_available():
            images = images.cuda()
            labels = labels.cuda()

```

```

        outputs = encoder(images)
        loss = loss_fn(outputs, labels)
        val_loss += loss.item()* images.size(0) # Accumulate the
        _, prediction = torch.max(outputs.data, 1)
        val_accuracy += int(torch.sum(prediction == labels.data))

    # Compute average loss and accuracy
    val_loss /= len(val_index)
    val_accuracy = val_accuracy / len(val_index)

    # Step the scheduler
    scheduler.step()

    print(f"Epoch [{epoch + 1}/{num_epochs}], Train Loss: {train_

    # Save the best model
    if val_accuracy > best_accuracy:
        torch.save(encoder.state_dict(), 'best_model.pth')
        best_accuracy = val_accuracy

    print(f"Best Validation Accuracy: {best_accuracy}")

```

In [18]: `results =CNN_train(3,class_folders,10,0.001)`

```

/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:62: Us
erWarning: The verbose parameter is deprecated. Please use get_last_lr() t
o access the learning rate.
  warnings.warn(

```

Fold 1/10

Epoch [1/3], Train Loss: 1.094, Train Accuracy: 0.361, Val Loss: 1.136, Validation Accuracy: 0.355

Epoch [2/3], Train Loss: 0.966, Train Accuracy: 0.496, Val Loss: 1.014, Validation Accuracy: 0.507

Epoch [3/3], Train Loss: 0.721, Train Accuracy: 0.674, Val Loss: 0.586, Validation Accuracy: 0.758

Fold 2/10

Epoch [1/3], Train Loss: 0.526, Train Accuracy: 0.783, Val Loss: 0.492, Validation Accuracy: 0.802

Epoch [2/3], Train Loss: 0.445, Train Accuracy: 0.823, Val Loss: 0.479, Validation Accuracy: 0.829

Epoch [3/3], Train Loss: 0.403, Train Accuracy: 0.843, Val Loss: 0.521, Validation Accuracy: 0.782

Fold 3/10

Epoch [1/3], Train Loss: 0.366, Train Accuracy: 0.857, Val Loss: 0.473, Validation Accuracy: 0.837

Epoch [2/3], Train Loss: 0.350, Train Accuracy: 0.866, Val Loss: 0.365, Validation Accuracy: 0.862

Epoch [3/3], Train Loss: 0.327, Train Accuracy: 0.875, Val Loss: 0.454, Validation Accuracy: 0.814

Fold 4/10

Epoch [1/3], Train Loss: 0.319, Train Accuracy: 0.879, Val Loss: 0.391, Validation Accuracy: 0.867

Epoch [2/3], Train Loss: 0.244, Train Accuracy: 0.909, Val Loss: 0.217, Validation Accuracy: 0.921

Epoch [3/3], Train Loss: 0.228, Train Accuracy: 0.917, Val Loss: 0.203, Validation Accuracy: 0.926

Fold 5/10

Epoch [1/3], Train Loss: 0.221, Train Accuracy: 0.918, Val Loss: 0.213, Validation Accuracy: 0.922

Epoch [2/3], Train Loss: 0.212, Train Accuracy: 0.922, Val Loss: 0.198, Validation Accuracy: 0.923

Epoch [3/3], Train Loss: 0.209, Train Accuracy: 0.924, Val Loss: 0.209, Validation Accuracy: 0.921

Fold 6/10

Epoch [1/3], Train Loss: 0.206, Train Accuracy: 0.923, Val Loss: 0.176, Validation Accuracy: 0.933

Epoch [2/3], Train Loss: 0.196, Train Accuracy: 0.927, Val Loss: 0.180, Validation Accuracy: 0.935

Epoch [3/3], Train Loss: 0.195, Train Accuracy: 0.928, Val Loss: 0.177, Validation Accuracy: 0.939

Fold 7/10

Epoch [1/3], Train Loss: 0.188, Train Accuracy: 0.931, Val Loss: 0.177, Validation Accuracy: 0.935

Epoch [2/3], Train Loss: 0.183, Train Accuracy: 0.934, Val Loss: 0.170, Validation Accuracy: 0.934

Epoch [3/3], Train Loss: 0.175, Train Accuracy: 0.935, Val Loss: 0.167, Validation Accuracy: 0.938

Fold 8/10

Epoch [1/3], Train Loss: 0.170, Train Accuracy: 0.938, Val Loss: 0.160, Validation Accuracy: 0.944

Epoch [2/3], Train Loss: 0.169, Train Accuracy: 0.938, Val Loss: 0.166, Validation Accuracy: 0.938

Epoch [3/3], Train Loss: 0.172, Train Accuracy: 0.937, Val Loss: 0.162, Validation Accuracy: 0.939

Fold 9/10

Epoch [1/3], Train Loss: 0.169, Train Accuracy: 0.939, Val Loss: 0.161, Validation Accuracy: 0.945

Epoch [2/3], Train Loss: 0.165, Train Accuracy: 0.939, Val Loss: 0.164, Va

Validation Accuracy: 0.939  
Epoch [3/3], Train Loss: 0.168, Train Accuracy: 0.938, Val Loss: 0.163, Validation Accuracy: 0.936  
Fold 10/10  
Epoch [1/3], Train Loss: 0.163, Train Accuracy: 0.941, Val Loss: 0.160, Validation Accuracy: 0.939  
Epoch [2/3], Train Loss: 0.163, Train Accuracy: 0.941, Val Loss: 0.154, Validation Accuracy: 0.940  
Epoch [3/3], Train Loss: 0.161, Train Accuracy: 0.941, Val Loss: 0.167, Validation Accuracy: 0.936  
Best Validation Accuracy: 0.9446666666666667

```
In [20]: test_path='/kaggle/input/gsoc123/new_dataset/val'
test_loader=DataLoader(
    torchvision.datasets.ImageFolder(test_path,transform=transformer),
    batch_size=32, shuffle=True
)
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

# Assuming model is already defined and moved to GPU if available

# Assuming transformer is defined

# Assuming test_loader is defined

y_score_list = []
y_true_list = []

# Evaluate model
encoder.eval()
for images, labels in test_loader:
    if torch.cuda.is_available():
        images = images.cuda()
        labels = labels.cuda()

    with torch.no_grad():
        y_score_batch = encoder(images)

    y_score_list.append(y_score_batch.cpu().numpy())
    y_true_list.append(labels.cpu().numpy())

y_score = np.vstack(y_score_list)
y_true = np.hstack(y_true_list)

# Binarize the ground truth labels
y_true_bin = label_binarize(y_true, classes=np.unique(y_true))

# Compute ROC curve and ROC area for each class
n_classes = y_score.shape[1]
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

```
# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_true_bin.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot ROC curve
plt.figure()
plt.plot(fpr["micro"], tpr["micro"], color='deeppink', lw=2, label=f'ROC curve (micro-AUC = {roc_auc["micro"]:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve (Multiclass)')
plt.legend(loc='lower right')
plt.show()
```

