resnet50_mstar

July 2, 2023

[ ]:
```
import os
from PIL import Image
from torchvision.transforms import ToTensor
import torchvision.models as models
import torch
from torch.utils.data import Dataset, DataLoader
from torch.optim import Adam
from torch.nn import CrossEntropyLoss
from tqdm import tqdm
```

[ ]:
```
class CustomDataset(Dataset):
    def __init__(self, root_dir, transform=ToTensor()):
        self.root_dir = root_dir
        self.transform = transform
        self.classes = ['BMP2', 'BTR70', 'T72']
        self.images = []
        self.labels = []

        for label, class_name in enumerate(self.classes):
            class_dir = os.path.join(self.root_dir, class_name+'/')
            for filename in os.listdir(class_dir):
                if filename.endswith('.jpg'):
                    self.images.append(os.path.join(class_dir, filename))
                    self.labels.append(label)

    def __getitem__(self, index):
        image = Image.open(self.images[index])
        if self.transform:
            image = self.transform(image)
        label = self.labels[index]
        return image, label

    def __len__(self):
        return len(self.images)
```

[ ]:
```
project_path = 'C:/Users/tnblt/Desktop/project/data/converted_data_set/'
```

```
train_dataset = CustomDataset(root_dir= os.path.join(project_path, 'TRAIN/
 ↪17_DEG/'))
test_dataset = CustomDataset(root_dir= os.path.join(project_path, 'TEST/15_DEG/
 ↪'))
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=True)
```

[ ]:
```
from torchvision.models.resnet import ResNet50_Weights

model = models.resnet50(weights = ResNet50_Weights.DEFAULT)
# mstar
model.conv1 = torch.nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2),␣
 ↪padding=(3, 3), bias=False)

num_classes = 3
model.fc = torch.nn.Linear(model.fc.in_features, num_classes)

# GPU
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
model = model.to(device)

#
criterion = CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0001, weight_decay=0.01)
```

[ ]:
```
#
best_loss = float('inf')
counter = 0
patience = 10  #

#
num_epochs = 100
for epoch in tqdm(range(num_epochs)):
    for inputs, labels in train_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)

        #
        optimizer.zero_grad()

        #
```

```
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        #print(f"loss: {loss.item()}")

        #
        if loss <= best_loss:
            best_loss = loss
            counter = 0
        else: #
            counter += 1
            if counter >= patience:
                #print("Early stopping, ", end="")
                break

        #
        loss.backward()
        optimizer.step()

    #
    best_loss = float('inf')
    counter = 0
```

```
100%|      | 100/100 [01:53<00:00,  1.14s/it]
```

[ ]:
```
#
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for inputs, labels in test_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        batch_correct = (predicted == labels).sum().item()
        batch_total = labels.size(0)
        correct += batch_correct
        total += batch_total
        #print(f'Batch Accuracy: {100 * batch_correct / batch_total}')

    print(f'Total Accuracy: {100 * correct / total}')
```

```
Total Accuracy: 97.28937728937728
```