오픈소스 연구논문 분석

소프트웨어학과_2022315965_문준원

사전 학습된 VGG-16 모델 가중치 활용

본 연구에서는 OxfordNet (VGG-16)네트워크를 R-CNN모델에 적용하여 성능을 향상시키고자 하였습니다. 이를 위해 Caffe의 Model Zoo플랫폼에서 제공하는 VGG_ILSVRC_16_layers 모델의 **사전 학습된 가중치**를 사용하였습니다. 연구진은 이 가중치를 R-CNN 모델에 불러와, 특정 작업에 맞춰 미세 조정(fine-tuning)을 수행했습니다. 이를 통해 모델을 처음부터 학습시키는 비용을 줄이고, 효율적으로 높은 성능을 낼 수 있었습니다.

구글코랩을 사용하여 실습해 보았습니다.

1. Google Drive 연결 및 라이브러리 설치 Google Colab을 사용할 경우, 아래와 같이 Google Drive를 마운트하고, 필요한 라이브러리를 설치 및 불러옵니다.

```
[3] from google.colab import drive

    # Attempting to mount the drive with a longer timeout
    drive.mount('/content/vggnet', force_remount=True, timeout_ms=300000)  # Increased timeout to 5 minutes (300000 milliseconds)

    # If it still fails, try this alternative:
    # from google.colab import auth
    # auth.authenticate_user()

    # drive.mount('/content/vggnet')
```

```
Mounted at /content/vggnet
```

2. 라이브러리 불러오기 이미지 전처리, 데이터 로드, 학습에 필요한 라이브러리를 불러옵니다.

```
[4] # Import required packages
    import torch
    import torch.nn as nn
    import torch.nn.functional as F
    from torchsummary import summary
    from torch import optim
    from torchvision import datasets, transforms
    from torch.utils.data import DataLoader
    import numpy as np
    import os
    import matplotlib.pyplot as plt

    # Enable inline plotting
    %matplotlib inline
```

3. STL-10 데이터셋 불러오기 및 정규화 STL-10 데이터셋을 다운로드하고, **정규화(mean, std)**를 수행한 후 FiveCrop 증강을 적용합니다.

```
# Set the data path
path2data = '/content/vggnet/MyDrive/data'

# Create directory if not exists
if not os.path.exists(path2data):
    os.mkdir(path2data)

# Load STL10 dataset
train_ds = datasets.STL10(path2data, split='train', download=True, transform=transforms.ToTensor())
val_ds = datasets.STL10(path2data, split='test', download=True, transform=transforms.ToTensor())

# Calculate mean and std for normalization
train_meanRGB = [np.mean(x.numpy(), axis=(1, 2)) for x, _ in train_ds]
train_stdRGB = [np.std(x.numpy(), axis=(1, 2)) for x, _ in train_ds]
train_mean = [np.mean([m[i] for m in train_meanRGB]) for i in range(3)]
train_std = [np.mean([s[i] for s in train_stdRGB]) for i in range(3)]

# Define transformations with FiveCrop and normalization
train_transformer = transforms.Compose([
    transforms.Resize(256),
    transforms.FiveCrop(224),
    transforms.Lambda(lambda crops: torch.stack([transforms.ToTensor()(crop) for crop in crops])),
    transforms.Normalize(train_mean, train_std),
])
val_transformer = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(train_mean, train_std),
])

# Apply transformation
train_ds.transform = train_transformer
val_ds.transform = val_transformer
```

```
Downloading http://ai.stanford.edu/~acoates/stl10/stl10_binary.tar.gz to /content/vggnet/MyDrive/data/stl10_binary.tar.gz
100%|██████████| 2.64G/2.64G [1:58<00:00, 3.67MB/s]
Extracting /content/vggnet/MyDrive/data/stl10_binary.tar.gz to /content/vggnet/MyDrive/data
Files already downloaded and verified
```

4. DataLoader 설정 데이터를 학습에 사용할 수 있도록 DataLoader로 변환합니다.

```
[6]  # Create DataLoader
     train_dl = DataLoader(train_ds, batch_size=4, shuffle=True)
     val_dl = DataLoader(val_ds, batch_size=4, shuffle=False)
```

5. VGG 모델 정의 VGG16 모델을 정의하고, 학습을 위한 가중치를 초기화합니다.

```
# Define VGG network structure
VGG_types = {
    'VGG16': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512, 'M']
}

class VGG_net(nn.Module):
    def __init__(self, model, in_channels=3, num_classes=10, init_weights=True):
        super(VGG_net, self).__init__()
        self.in_channels = in_channels
        self.conv_layers = self.create_conv_layers(VGG_types[model])

        self.fc_layers = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(),
            nn.Dropout(p=0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(),
            nn.Dropout(p=0.5),
            nn.Linear(4096, num_classes),
        )

        if init_weights:
            self._initialize_weights()

    def forward(self, x):
        x = self.conv_layers(x)
        x = x.view(x.size(0), -1)
        x = self.fc_layers(x)
        return x

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
                if m.bias is not None:
                    nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, 0, 0.01)
                nn.init.constant_(m.bias, 0)
```

```
    def create_conv_layers(self, architecture):
        layers = []
        in_channels = self.in_channels
        for x in architecture:
            if type(x) == int:
                out_channels = x
                layers += [nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1),
                           nn.BatchNorm2d(out_channels),
                           nn.ReLU(inplace=True)]
                in_channels = out_channels
            elif x == 'M':
                layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        return nn.Sequential(*layers)

# Initialize model
device = 'cuda' if torch.cuda.is_available() else 'cpu'
model = VGG_net('VGG16', in_channels=3, num_classes=10, init_weights=True).to(device)
summary(model, (3, 224, 224), device=device) # Changed: removed .type
```

출력결과:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
       BatchNorm2d-2         [-1, 64, 224, 224]             128
              ReLU-3         [-1, 64, 224, 224]               0
            Conv2d-4         [-1, 64, 224, 224]          36,928
       BatchNorm2d-5         [-1, 64, 224, 224]             128
              ReLU-6         [-1, 64, 224, 224]               0
         MaxPool2d-7         [-1, 64, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]          73,856
       BatchNorm2d-9        [-1, 128, 112, 112]             256
             ReLU-10        [-1, 128, 112, 112]               0
           Conv2d-11        [-1, 128, 112, 112]         147,584
      BatchNorm2d-12        [-1, 128, 112, 112]             256
             ReLU-13        [-1, 128, 112, 112]               0
        MaxPool2d-14          [-1, 128, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         295,168
      BatchNorm2d-16          [-1, 256, 56, 56]             512
             ReLU-17          [-1, 256, 56, 56]               0
           Conv2d-18          [-1, 256, 56, 56]         590,080
      BatchNorm2d-19          [-1, 256, 56, 56]             512
             ReLU-20          [-1, 256, 56, 56]               0
           Conv2d-21          [-1, 256, 56, 56]         590,080
      BatchNorm2d-22          [-1, 256, 56, 56]             512
             ReLU-23          [-1, 256, 56, 56]               0
        MaxPool2d-24          [-1, 256, 28, 28]               0
           Conv2d-25          [-1, 512, 28, 28]       1,180,160
      BatchNorm2d-26          [-1, 512, 28, 28]           1,024
             ReLU-27          [-1, 512, 28, 28]               0
           Conv2d-28          [-1, 512, 28, 28]       2,359,808
      BatchNorm2d-29          [-1, 512, 28, 28]           1,024
             ReLU-30          [-1, 512, 28, 28]               0
           Conv2d-31          [-1, 512, 28, 28]       2,359,808
      BatchNorm2d-32          [-1, 512, 28, 28]           1,024
             ReLU-33          [-1, 512, 28, 28]               0
        MaxPool2d-34          [-1, 512, 14, 14]               0
           Conv2d-35          [-1, 512, 14, 14]       2,359,808
      BatchNorm2d-36          [-1, 512, 14, 14]           1,024
             ReLU-37          [-1, 512, 14, 14]               0
           Conv2d-38          [-1, 512, 14, 14]       2,359,808
      BatchNorm2d-39          [-1, 512, 14, 14]           1,024
             ReLU-40          [-1, 512, 14, 14]               0
           Conv2d-41          [-1, 512, 14, 14]       2,359,808
      BatchNorm2d-42          [-1, 512, 14, 14]           1,024
             ReLU-43          [-1, 512, 14, 14]               0
        MaxPool2d-44            [-1, 512, 7, 7]               0
           Linear-45                 [-1, 4096]     102,764,544
             ReLU-46                 [-1, 4096]               0
          Dropout-47                 [-1, 4096]               0
           Linear-48                 [-1, 4096]      16,781,312
             ReLU-49                 [-1, 4096]               0
          Dropout-50                 [-1, 4096]               0
           Linear-51                   [-1, 10]          40,970
================================================================
Total params: 134,309,962
Trainable params: 134,309,962
Non-trainable params: 0
```

6. 모델 학습 및 평가 학습, 평가 루프는 성능을 측정하는 중요한 단계입니다. 아래는 간단한 학습 루프 예시입니다.

```
from tqdm import tqdm  # 진행 바 표시를 위해 tqdm 라이브러리 사용

# Define optimizer and loss function
import time
from torch import repeat_interleave
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
# Define number of epochs
num_epochs = 10  # 원하는 에포크 수 설정

for epoch in range(num_epochs):
    model.train()
    train_loss = 0.0
    train_correct = 0
    total_samples = 0
    start_time = time.time()

    # tqdm을 사용하여 진행 바 표시
    with tqdm(total=len(train_dl), desc=f"Epoch {epoch+1}/{num_epochs}", unit="batch") as pbar:
        for images, labels in train_dl:
            images = images.to(device)
            labels = labels.to(device)

            # Forward pass
            outputs = model(images)
            loss = criterion(outputs, labels)
            train_loss += loss.item()

            # Calculate accuracy
            _, preds = torch.max(outputs, 1)
            train_correct += (preds == labels).sum().item()
            total_samples += labels.size(0)

            # Backward pass and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # tqdm 진행 바 업데이트
            pbar.update(1)

    # Calculate and print results for each epoch
    train_loss /= len(train_dl)
    train_accuracy = 100 * train_correct / total_samples
    train_time = time.time() - start_time
    print(f"Epoch {epoch+1}/{num_epochs}: "
          f"train loss: {train_loss:.4f}, accuracy: {train_accuracy:.2f}%, "
          f"time: {train_time:.4f} s")
    print("--" * 10)
```

출력값: 본 연구에서는 **VGG16 모델**을 사용하여 학습을 진행했습니다. 하지만 VGG16 모델은 파라미터 수가 매우 많고, 연산량이 커서 Google Colab 환경의 자원으로는 제한된 에포크까지만 학습을 수행할 수 있었습니다. 특히, Colab의 GPU 메모리 한계와 학습 시간이 길어지는 문제로 인해 **에포크 설정 중 10 에포크**까지만 실행이 가능했습니다. 실제 학습 결과, 에포크가 진행됨에 따라 손실값이 점차 줄어들고 정확도가 서서히 증가하는 경향을 보였으나, 자원 제약으로 인해 모델이 충분히 수렴하지 못한 상태에서 학습을 중단하게 되었습니다.

```
Epoch 1/10
train loss: 2.305832, accuracy: 9.88, time: 151.3181 s
----------
Epoch 2/10
train loss: 2.306310, accuracy: 9.34, time: 151.5804 s
----------
Epoch 3/10
train loss: 2.307419, accuracy: 9.02, time: 151.5708 s
----------
Epoch 4/10
train loss: 2.304202, accuracy: 9.96, time: 151.6122 s
----------
Epoch 5/10
train loss: 2.303574, accuracy: 9.50, time: 151.5158 s
----------
Epoch 6/10
train loss: 2.303992, accuracy: 8.82, time: 151.5466 s
----------
Epoch 7/10
train loss: 2.303563, accuracy: 9.62, time: 151.5048 s
----------
Epoch 8/10
train loss: 2.303516, accuracy: 9.26, time: 151.2012 s
----------
Epoch 9/10
train loss: 2.303282, accuracy: 9.66, time: 151.1494 s
----------
Epoch 10/10
train loss: 2.303453, accuracy: 9.36, time: 151.3132 s
----------
```