# 猫狗分类

**猫狗分类**

1 实验概述

2 实验环境

2.1 硬件环境

2.2 软件环境

2.3 Conda环境

3 实验思路

3.1 数据来源

3.2 数据预处理

3.3 模型选择与参数设置

4 实验代码

4.1 数据载入部分

4.2 模型部分

4.3 训练部分

5 实验结果

5.1 训练部分输出展示

5.2 Train Loss图像

5.3 Test Acc图像

6 总结

# 1 实验概述

本实验原为一次Kaggle竞赛题目，目的是训练一个模型来对猫狗图片进行分类。

# 2 实验环境

## 2.1 硬件环境

- CPU：14 vCPU Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz
- GPU：RTX 3090(24GB) * 1

## 2.2 软件环境

- Ubuntu：20.04
- Miniconda

## 2.3 Conda环境

- Python：3.8
- PyTorch：2.0.0
- Cuda：11.8

# 3 实验思路

## 3.1 数据来源

本次实验使用的数据集为Kaggle猫狗竞赛的原始数据集，带有标签的图片共25000张，其中猫狗照片各12500 张。为了方便训练与测试，将该数据集划分为训练集与测试集，其中训练集共20000张，猫狗各10000张；测试集5000张，猫狗各2500张。

```
# 整理dataset
dataset_path = '../autodl-tmp/kaggle_datasets/train/'
train_dataset_path = '../autodl-tmp/kaggle_datasets/train/'
test_dataset_path = '../autodl-tmp/kaggle_datasets/test/'
train_cat_path = train_dataset_path + 'cat/'
train_dog_path = train_dataset_path + 'dog/'
test_cat_path = test_dataset_path + 'cat/'
test_dog_path = test_dataset_path + 'cat/'

```

```
origin_files = os.listdir(dataset_path)
for file in origin_files:
    if os.path.isdir(file):
        continue
    oldfile_path = dataset_path + file
    (file_name, file_ext) = os.path.splitext(file)
    columns = file_name.split('.')
    if len(columns) != 2:
        continue
    label = columns[0]
    index = columns[1]
    index = int(index)
    new_filename = str(index) + file_ext
    if index < 10000:
        if label == 'cat':
            newfile_path = train_cat_path + new_filename
        else:
            newfile_path = train_dog_path + new_filename
    else:
        if label == 'cat':
            newfile_path = test_cat_path + new_filename
        else:
            newfile_path = test_dog_path + new_filename
    os.replace(oldfile_path, newfile_path)

print('Preprocess datasets Done!')
```

## 3.2 数据预处理

该数据集由于宽高均不统一，且没有被torchvison封装完善，所以需要一些步骤来进行预处理。

1. 图片裁剪与图片标签向量化

```
from torchvision import transforms, datasets
#定义transforms
transforms = transforms.Compose(
[
transforms.RandomResizedCrop(150),
transforms.ToTensor(),
transforms.Normalize(mean=[0.485, 0.456, 0.406],
                     std=[0.229, 0.224, 0.225])
]
)
train_data = datasets.ImageFolder(train_dataset_path, transforms)
test_data=datasets.ImageFolder(test_dataset_path, transforms)
```

2. 划分训练集与验证集

```
def dataset_split(full_dateset, train_rate): # full_ds为train_ds, train_rate=0.8
    train_size = int(len(full_dateset) * train_rate)
    validate_size = len(full_dateset) - train_size
    train_dataset, validate_dataset = torch.utils.data.random_split(full_dateset, [train_size, validate_size])
    return train_dataset, validate_dataset
```

3. 载入DataLoader

```
def dataloader(dataset, batch_size, num_workers):
    data_loader = torch.utils.data.DataLoader(dataset, batch_size, shuffle=True, pin_memory=True, num_workers=num_workers)
    return data_loader
```

### 3.3 模型选择与参数设置

模型网络选择 `ResNet18`

参数设置如下：

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
EPOCH = 30
LR = 0.001
TRAIN_RATE = 0.8
NUM_WORKERS = 14 # cpu cores num
BATCH_SIZE = 1500
```

## 4 实验代码

### 4.1 数据载入部分

```
train_data, valid_data = dataset_split(train_data, TRAIN_RATE)

train_dataloader = dataloader(train_data, BATCH_SIZE, NUM_WORKERS)
valid_dataloader = dataloader(valid_data, BATCH_SIZE, NUM_WORKERS)
test_dataloader = dataloader(test_data, BATCH_SIZE, NUM_WORKERS)
```

## 4.2 模型部分

```
resnet18 = models.resnet18(pretrained=True)
resnet18.fc = nn.Linear(512, 2) # 将最后一层全连接的输出调整到2维
net = resnet18.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=LR, momentum=0.9, weight_decay=5e-4)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.5)
```

## 4.3 训练部分

```
train_losses = []
train_counter = []
test_accuracy = [0.0]
test_counter = [i*len(train_dataloader.dataset) for i in range(EPOCH + 1)]

for epoch in range(0, EPOCH):
    net.train()
    sum_loss = 0.0
    correct = 0.0
    total = 0.0
    for i, data in enumerate(train_dataloader, 0):
        length = len(train_dataloader)
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        sum_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += predicted.eq(labels.data).cpu().sum()

        train_losses.append(sum_loss / (i + 1))
        train_counter.append((i*BATCH_SIZE) + ((epoch-1)*len(train_dataloader.dataset)))
    scheduler.step()
    print('[epoch:%d] <Train> Loss: %.03f, Acc: %.3f%% ' % (epoch + 1, sum_loss / (i + 1), 100. * correct / total), end='\t')


    with torch.no_grad():
        correct = 0
        total = 0
```

```
        for data in valid_dataloader:
            net.eval()
            images, labels = data
            images, labels = images.to(device), labels.to(device)
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum()
        print('<Test> Acc: %.3f%%' % (100 * correct / total))
        test_accuracy.append(correct.item() / total)

print('Train has finished, total epoch is %d' % EPOCH)
torch.save(net.state_dict(), './' + str(EPOCH) +'epoch-model.pth')
torch.save(optimizer.state_dict(), './' + str(EPOCH) +'epoch-optimizer.pth')
```

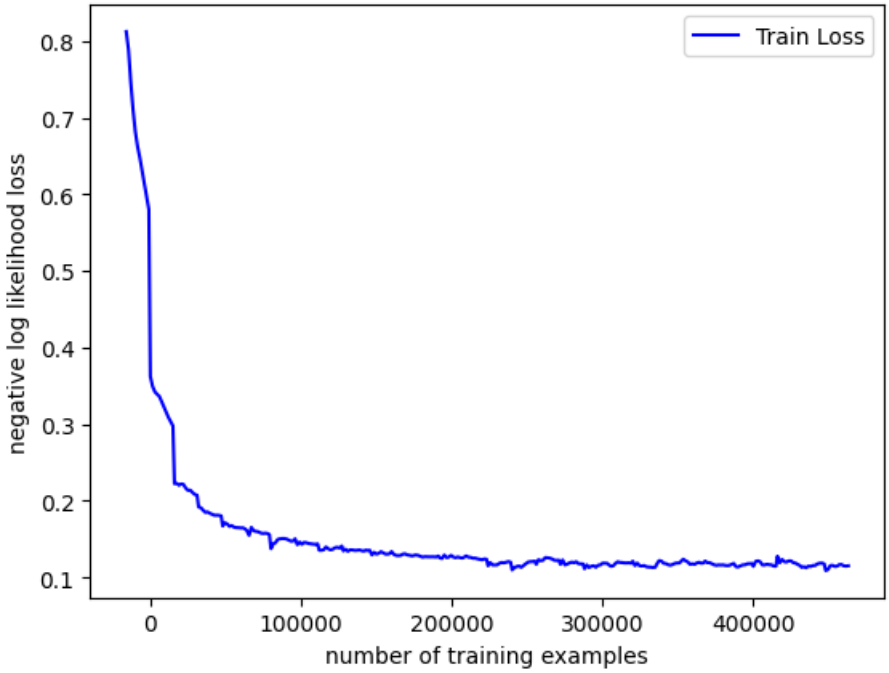
# 5 实验结果

## • 5.1 训练部分输出展示

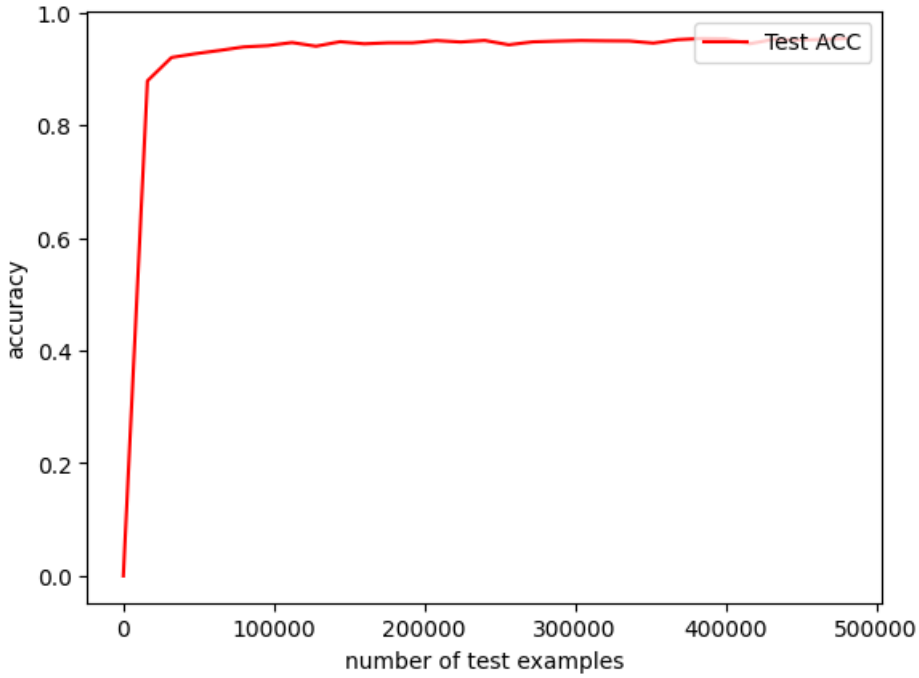
```
[epoch:1] <Train> Loss: 0.580, Acc: 67.600%     <Test> Acc: 87.850%
[epoch:2] <Train> Loss: 0.298, Acc: 88.144%     <Test> Acc: 92.000%
[epoch:3] <Train> Loss: 0.207, Acc: 92.325%     <Test> Acc: 92.675%
[epoch:4] <Train> Loss: 0.180, Acc: 92.762%     <Test> Acc: 93.250%

...
[epoch:28] <Train> Loss: 0.115, Acc: 95.269%    <Test> Acc: 95.075%
[epoch:29] <Train> Loss: 0.118, Acc: 95.200%    <Test> Acc: 95.200%
[epoch:30] <Train> Loss: 0.115, Acc: 95.200%    <Test> Acc: 95.350%
Train has finished, total epoch is 30
```

## • 5.2 Train Loss图像




- **5.3 Test Acc图像**




# 6 总结

第一次实验使用的样本集在 `torch` 里内置，所以在本次实验中才相对完整的进行了一遍整个深度学习的实验过程。从数据预处理到训练再到测试。通过本次实验，学习了如何去预处理图像数据并载入 `dataloader` 中，继续练习了使用 `CNN` 模型，并尝试使用 `ResNet` 架构来提升训练效果，最终也确实将训练效果提升到了95%以上。