CNN

IMPORTING LIBRARIES

```
from __future__ import print_function, division

import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import torch.backends.cudnn as cudnn
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import copy

cudnn.benchmark = True
plt.ion()


from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

DATA TRANSFORMATION

```
# Data augmentation and normalization for training
# Just normalization for validation
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
```

```
}

data_dir = '/content/gdrive/MyDrive/Machine Learning assignments/hymenoptera_data'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                              shuffle=True, num_workers=4)
               for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarnir
  cpuset_checked))
```

Data Visualisation

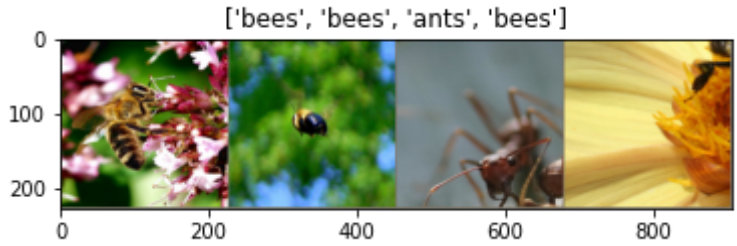
```
def imshow(inp, title=None):
    """Imshow for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)  # pause a bit so that plots are updated


# Get a batch of training data
inputs, classes = next(iter(dataloaders['train']))

# Make a grid from batch
out = torchvision.utils.make_grid(inputs)

imshow(out, title=[class_names[x] for x in classes])
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarnir
  cpuset_checked))
```



Model Training

We have defined val_loss, accuracy and epoch_count to use it in plotting "**Validation loss and accuracy plot**". We have appened all three variable when **phase = 'val'**.

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()
    val_loss=[]
    accuracy=[]
    epoch_count=[]

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
```

```
            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')
            if(phase=='val'):
              epoch_count.append(epoch)
              val_loss.append(epoch_loss)
              accuracy.append(epoch_acc)

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time() - since
    print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s')
    print(f'Best val Acc: {best_acc:4f}')

    #plot the validation loss and accuracy plot
    plt.plot(epoch_count, val_loss, color='r', label='Validation loss')
    plt.plot(epoch_count, accuracy, color='g', label='Accuracy')
    plt.xlabel("Epoch")
    plt.ylabel("Val_loss or Accuracy")
    plt.title("Validation loss and accuracy plot")
    plt.legend()
    plt.show()

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model
```

Model Visualization

```
def visualize_model(model, num_images=6):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()

    with torch.no_grad():
        for i, (inputs, labels) in enumerate(dataloaders['val']):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
```

```
            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title(f'predicted: {class_names[preds[j]]}')
                imshow(inputs.cpu().data[j])

                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
        model.train(mode=was_training)


model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
# Here the size of each output sample is set to 2.
# Alternatively, it can be generalized to nn.Linear(num_ftrs, len(class_names)).
model_ft.fc = nn.Linear(num_ftrs, 2)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.ca
100%                                  44.7M/44.7M [00:00<00:00, 127MB/s]
```

```
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,num_epochs=25)
```

```
Epoch 0/24
----------
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarnir
  cpuset_checked))
train Loss: 0.4983 Acc: 0.7828
val Loss: 0.2657 Acc: 0.8889

Epoch 1/24
----------
train Loss: 0.5046 Acc: 0.7828
val Loss: 0.2430 Acc: 0.8954

Epoch 2/24
----------
train Loss: 0.6078 Acc: 0.7787
val Loss: 0.2700 Acc: 0.9216

Epoch 3/24
----------
train Loss: 0.5183 Acc: 0.7787
val Loss: 0.4290 Acc: 0.8497

Epoch 4/24
----------
train Loss: 0.6185 Acc: 0.7746
val Loss: 0.3569 Acc: 0.8954

Epoch 5/24
----------
train Loss: 0.5155 Acc: 0.7828
val Loss: 0.3247 Acc: 0.8954

Epoch 6/24
----------
train Loss: 0.4920 Acc: 0.8279
```

```
visualize_model(model_ft)
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490: UserWarnir
  cpuset_checked))
```

predicted: bees

predicted: bees

predicted: bees

```
model_conv = torchvision.models.resnet18(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()
```

predicted: bees

```
def Conv(Ler, moe, ep):
  v_l = pd.DataFrame(columns=[_ for _ in range(ep)])
  acc = pd.DataFrame(columns=[_ for _ in range(ep)])

  for x in range(len(Ler)):
    for y in range(len(moe)):
      for z in range(len(ep)):
        optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=Ler[x], momentum=moe[y])
        exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=4, gamma=0.1)
        mod_con, L, A = train_model(model_conv, criterion, optimizer_conv,exp_lr_scheduler
        v_l.loc[len(v_l)] = L
        acc.loc[len(acc)] = A

  print (v_l)
  print (acc)
  plt.plot(v_l,acc)
  return mod_con
```

val Loss: 0.2025 Acc: 0.9412

```
#Change the learning rate, momentum, and number of epochs
lr = [0.001,0.003,0.005]
m = [0.3,0.5,0.9]
ep = [22,25,27]

Conv(lr,m,ep)
```

```
----------
```

Conclusion: CNN was performed the best epoch was found to be 24 with val_Loss: 0.1975 and val_Acc: 0.954248.

```
train Loss: 0.2875 Acc: 0.8811
```

Reference:

1. Discussed with 213300017
2. https://pytorch.org/docs/stable/optim.html
3. https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html