Experiment - Implementation of Transfer Learning

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import shutil
from torchvision import transforms
from torchvision import models
import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import lr_scheduler
from torch import optim
from torchvision.datasets import ImageFolder
from torchvision.utils import make_grid
from torch.utils.data import Dataset,DataLoader
import time
%matplotlib inline
```

In [ ]:
```
def imshow(inp,cmap=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,cmap)
```

In [ ]:
```
is_cuda = False
if torch.cuda.is_available():
    is_cuda = True
```

In [ ]:
```
simple_transform = transforms.Compose([transforms.Resize((224,224))
                                       ,transforms.ToTensor()
                                       ,transforms.Normalize([0.485, 0.456, 0.406], [0.2
29, 0.224, 0.225])
                                      ])
train = ImageFolder('/kaggle/input/cat-and-dog/training_set/training_set/',simple_transf
orm)
valid = ImageFolder('/kaggle/input/cat-and-dog/test_set/test_set/',simple_transform)
```

In [ ]:
```
print(train.class_to_idx)
print(train.classes)
```

In [ ]:
```
imshow(valid[770][0])
```

In [ ]:
```
train_data_loader = torch.utils.data.DataLoader(train,batch_size=32,num_workers=3,shuffl
e=True)
valid_data_loader = torch.utils.data.DataLoader(valid,batch_size=32,num_workers=3,shuffl
e=True)
```

In [ ]:
```
vgg = models.vgg16(pretrained=True)
vgg = vgg.cuda()
```

In [ ]:
```
vgg
```

In [ ]:
```
vgg.classifier[6].out_features = 2
for param in vgg.features.parameters(): param.requires_grad = False
```

In [ ]:
```
optimizer = optim.SGD(vgg.classifier.parameters(),lr=0.0001,momentum=0.5)
```

In [ ]:
```
def fit(epoch,model,data_loader,phase='training',volatile=False):
    if phase == 'training':
        model.train()
    if phase == 'validation':
        model.eval()
        volatile=True
    running_loss = 0.0
    running_correct = 0
    for batch_idx , (data,target) in enumerate(data_loader):
        if is_cuda:
            data,target = data.cuda(),target.cuda()
        data , target = Variable(data,volatile),Variable(target)
        if phase == 'training':
            optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output,target)

        running_loss += F.cross_entropy(output,target,size_average=False).item()
        preds = output.data.max(dim=1,keepdim=True)[1]
        running_correct += preds.eq(target.data.view_as(preds)).cpu().sum()
        if phase == 'training':
            loss.backward()
            optimizer.step()

    loss = running_loss/len(data_loader.dataset)
    accuracy = 100. * running_correct/len(data_loader.dataset)

    print(f'{phase} loss is {loss:{5}.{2}} and {phase} accuracy is {running_correct}/{le
n(data_loader.dataset)}{accuracy:{10}.{4}}')
    return loss,accuracy
```

In [ ]:
```
train_losses , train_accuracy = [],[]
val_losses , val_accuracy = [],[]
for epoch in range(1,10):
    epoch_loss, epoch_accuracy = fit(epoch,vgg,train_data_loader,phase='training')
    val_epoch_loss , val_epoch_accuracy = fit(epoch,vgg,valid_data_loader,phase='validat
ion')
    train_losses.append(epoch_loss)
    train_accuracy.append(epoch_accuracy)
    val_losses.append(val_epoch_loss)
    val_accuracy.append(val_epoch_accuracy)
```