Experiment - Implementation of Transfer Learning

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
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
```

In [3]:

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

In [4]:

```
def show(inp) :
    inp = inp.numpy().transpose((1,2,0))
    plt.imshow(inp)
```

In [6]:

```
transform = transforms.Compose([transforms.Resize((224,224)),
                               transforms.ToTensor()
])
```

In [7]:

```
train = ImageFolder('/kaggle/input/animals141/dataset/dataset',transform)
```

In [8]:

```
train_data_loader = torch.utils.data.DataLoader(train,batch_size=32,num_workers=3,shuffle=True)
```

In [9]:

```
train[0]
```

In [10]:

```
train.classes
```

In [11]:

```
print(train.class_to_idx)
```

In [12]:

```
show(train[50][0])
```

In [14]:

```
vgg = models.vgg16(pretrained = True)
vgg = vgg.cuda()
```

In [15]:

```
vgg
```

In [16]:

```
vgg.classifier[6].out_features = 151
for param in vgg.features.parameters():
    param.requires_grad = False
```

In [17]:

```
optimizer = optim.SGD(vgg.classifier.parameters(),lr=0.01)
```

In [18]:

```
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}/{len(data_loader.dataset)}{accuracy:{10}.{4}}')
    return loss,accuracy
```

In [19]:

```
train_losses , train_accuracy = [],[]
val_losses , val_accuracy = [],[]
for epoch in range(1,15):
    epoch_loss, epoch_accuracy = fit(epoch,vgg,train_data_loader,phase='training')
    train_losses.append(epoch_loss)
    train_accuracy.append(epoch_accuracy)
```