In [1]:
```python
from torchvision import transforms
from PIL import Image

# Image Preprocessing: Resize, Crop, Normalize
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Load Image
image_path = "cat.jpg"  # Replace with your image file path
image = Image.open(image_path).convert("RGB")
input_tensor = transform(image).unsqueeze(0)  # Add batch dimension
```

In [2]:
```python
import torch
from torchvision.models import alexnet

# Load Pretrained AlexNet Model
model = alexnet(pretrained=True)
model.eval()  # Set to evaluation mode

# Forward Pass
with torch.no_grad():
    output = model(input_tensor)
    predicted_class = torch.argmax(output, 1)
    print(f"Predicted class: {predicted_class.item()}")
```

```
c:\Users\jerin\AppData\Local\Programs\Python\Python311\Lib\site-packages\torchvision\models\_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
c:\Users\jerin\AppData\Local\Programs\Python\Python311\Lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing `weights=AlexN
et_Weights.IMAGENET1K_V1`. You can also use `weights=AlexNet_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Predicted class: 285
```

In [3]:
```python
from torchvision.datasets import CIFAR10
from torch.utils.data import DataLoader

# Data Loading and Augmentation
transform_train = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
])

train_dataset = CIFAR10(root="./data", train=True, download=True, transform=transform_train)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

test_dataset = CIFAR10(root="./data", train=False, download=True, transform=transform_train)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [4]:
```python
import torch.nn as nn
import torch.optim as optim

# Modify AlexNet for CIFAR-10 (10 classes)
model.classifier[6] = nn.Linear(4096, 10)  # Output layer for 10 classes
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Loss Function and Optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training Loop
epochs = 5
for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

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

        running_loss += loss.item()
    print(f"Epoch {epoch+1}/{epochs}, Loss: {running_loss/len(train_loader)}")
```

```
Epoch 1/5, Loss: 2.0272153523665395
Epoch 2/5, Loss: 1.5890526312021438
Epoch 3/5, Loss: 1.3956197938397383
Epoch 4/5, Loss: 1.273427989874905
Epoch 5/5, Loss: 1.192788677343709
```

In [5]:
```python
# Validation Loop
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
```

```
print(f"Accuracy: {100 * correct / total:.2f}%")
```

```
Accuracy: 63.76%
```