In [45]:
```
import torch
from torchvision.datasets import CIFAR10
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torchsummary import summary
from torchvision.utils import make_grid
import numpy as np
import matplotlib.pyplot as plt
```

In [46]:
```
tfms=transforms.Compose([transforms.ToTensor()])
train_data=CIFAR10(root='./data',train=True,transform=tfms,download=True)
test_data=CIFAR10(root='./data',train=False,download=True,transform=tfms)
```

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

In [47]:
```
batch_size=32
lr=.01
karnal_size=3
device=torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cpu
```

In [48]:
```
train_loader=DataLoader(train_data,batch_size=batch_size,shuffle=False)
test_loader=DataLoader(test_data,batch_size=batch_size,shuffle=False)
```

In [49]:
```
trainiter=iter(train_loader)
images,labels=trainiter.next()
images.shape
```

Out[49]:
```
torch.Size([32, 3, 32, 32])
```

In [50]:
```
plt.imshow((make_grid(images).numpy().transpose((1,2,0))))
```

Out[50]:
```
<matplotlib.image.AxesImage at 0x7fad6a73b250>
```

In [51]:
```
def conv_layer(in_channels,out_channels,kernel_size):

    return nn.Sequential(
        nn.Conv2d(in_channels=in_channels,out_channels=out_channels,kernel_size=kernel_s
ize),
        nn.MaxPool2d(kernel_size=(2,2)),
        nn.ReLU()

        )
```

In [52]:
```
class Conv_net(nn.Module):
    def __init__(self):
        super(Conv_net, self).__init__()
        self.layer1=conv_layer(3,16,3)
        self.layer2=conv_layer(16,32,3)
        self.layer3=conv_layer(32,64,3)
        self.flatten=nn.Flatten()
        self.fc1=nn.Linear(64*2*2,50)
        self.fc2=nn.Linear(50,10)
    def forward(self,x):
        x=self.layer1(x)
        x=self.layer2(x)
        x=self.layer3(x)
        x=self.flatten(x)
        x=self.fc1(x)
        x=self.fc2(x)
        return x
```

In [53]:
```
model=Conv_net()
model
```

Out[53]:
```
Conv_net(
  (layer1): Sequential(
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
    (1): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=
False)
    (2): ReLU()
  )
  (layer2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=
False)
    (2): ReLU()
  )
  (layer3): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=
False)
    (2): ReLU()
  )
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=256, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=10, bias=True)
)
```

In [54]:
```
summary(model=model,input_size=(3,32,32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 30, 30]             448
         MaxPool2d-2          [-1, 16, 15, 15]               0
              ReLU-3          [-1, 16, 15, 15]               0
            Conv2d-4          [-1, 32, 13, 13]           4,640
         MaxPool2d-5            [-1, 32, 6, 6]               0
              ReLU-6            [-1, 32, 6, 6]               0
            Conv2d-7            [-1, 64, 4, 4]          18,496
         MaxPool2d-8            [-1, 64, 2, 2]               0
              ReLU-9            [-1, 64, 2, 2]               0
           Flatten-10                 [-1, 256]               0
            Linear-11                  [-1, 50]          12,850
            Linear-12                  [-1, 10]             510
================================================================
Total params: 36,944
Trainable params: 36,944
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.24
Params size (MB): 0.14
Estimated Total Size (MB): 0.39
----------------------------------------------------------------
```

In [55]:
```
loss_fn=nn.CrossEntropyLoss()
optimizer=torch.optim.Adam(model.parameters(),lr=lr)
```

In [67]:
```
def train(mode,trainloader,loss_fn=None,optimizer=None,seed=32,EPOCHS=100):
    torch.manual_seed(seed)
    cost=[]

    model.to(device)
    for epoch in range(EPOCHS):
        train_corr=[]
        for idx,(img,lbl) in enumerate(trainloader):
            ##1 Compute the Output
            img,lbl=img.to(device),lbl.to(device)
            yhat=model(img)

            ##2 Compute the Loss
            loss=loss_fn(yhat,lbl)

            ##3 Compute the gradients
            optimizer.zero_grad()
            loss.backward()

            ##4 update the model parameters
            optimizer.step()
            with torch.no_grad():
                yhat=model(img)
                loss=loss_fn(yhat,lbl)
                core_lbl=(torch.argmax(F.softmax(yhat,dim=1),dim=1)==lbl).sum().item()
                ##print('Model accuracy on this batch %.2f', core_lbl)
                train_corr.append(core_lbl)
        with torch.no_grad():
            acc=sum(train_corr)/len(train_data)*100

            print(f'After {epoch+1} Model Training Accuracy {acc}')
```

In [68]:
```
len(train_data)
```

Out[68]:
```
50000
```

In [69]:
```
model=Conv_net()
cost=train(mode=model,trainloader=train_loader,loss_fn=loss_fn,optimizer=optimizer,EPOCHS=10)
```

```
After 1 Model Training Accuracy 10.0

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/var/folders/_j/jdf44bpj0w5cfxqs6v379p_80000gn/T/ipykernel_43172/1302054821.py in <module>
      1 model=Conv_net()
----> 2 cost=train(mode=model,trainloader=train_loader,loss_fn=loss_fn,optimizer=optimizer,EPOCHS=10)

/var/folders/_j/jdf44bpj0w5cfxqs6v379p_80000gn/T/ipykernel_43172/3500034063.py in train(mode, trainloader, loss_fn, optimizer, seed, EPOCHS)
     19
     20             ##4 update the model parameters
---> 21             optimizer.step()
     22             with torch.no_grad():
     23                 yhat=model(img)

~/opt/miniconda3/envs/DL/lib/python3.7/site-packages/torch/optim/optimizer.py in wrapper(*args, **kwargs)
     86                 profile_name = "Optimizer.step#{}.step".format(obj.__class__.__name__)
     87                 with torch.autograd.profiler.record_function(profile_name):
---> 88                     return func(*args, **kwargs)
     89             return wrapper
     90

~/opt/miniconda3/envs/DL/lib/python3.7/site-packages/torch/autograd/grad_mode.py in decorate_context(*args, **kwargs)
     26         def decorate_context(*args, **kwargs):
     27             with self.__class__():
---> 28                 return func(*args, **kwargs)
     29         return cast(F, decorate_context)
     30

~/opt/miniconda3/envs/DL/lib/python3.7/site-packages/torch/optim/adam.py in step(self, closure)
    142                     lr=group['lr'],
    143                     weight_decay=group['weight_decay'],
--> 144                     eps=group['eps'])
    145         return loss

~/opt/miniconda3/envs/DL/lib/python3.7/site-packages/torch/optim/_functional.py in adam(params, grads, exp_avgs, exp_avg_sqs, max_exp_avg_sqs, state_steps, amsgrad, beta1, beta2, lr, weight_decay, eps)
     84
     85         # Decay the first and second moment running average coefficient
---> 86         exp_avg.mul_(beta1).add_(grad, alpha=1 - beta1)
     87         exp_avg_sq.mul_(beta2).addcmul_(grad, grad.conj(), value=1 - beta2)
     88         if amsgrad:

KeyboardInterrupt:
```