Implementación de una CNN para Clasificación de Imágenes

Una CNN simple utilizando el dataset CIFAR-10, que contiene 60,000 imágenes de 32x32 píxeles en 10 clases diferentes.

In [1]:
```
# Importar las librerías necesarias
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np

# Definir las transformaciones para normalizar los datos
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# Cargar el dataset CIFAR-10
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

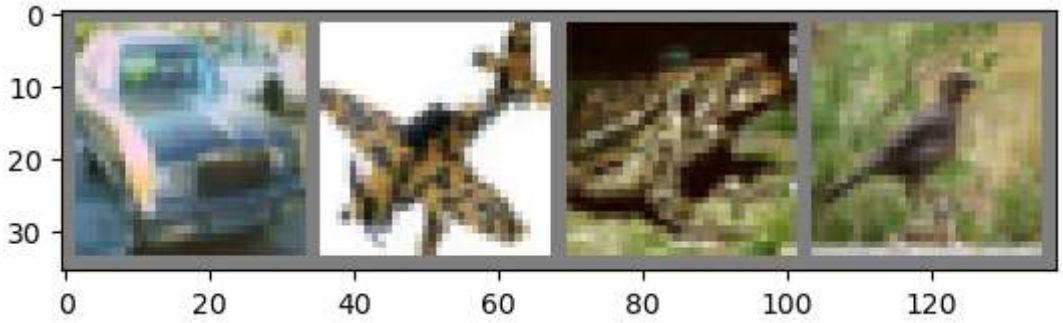
```
Files already downloaded and verified
Files already downloaded and verified
```

Mostrar algunas imágenes del dataset

In [3]:
```
# Función para mostrar una imagen
def imshow(img):
    img = img / 2 + 0.5  # desnormalizar
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

# Mostrar algunas imágenes del dataset
dataiter = iter(trainloader)
images, labels = next(dataiter)

imshow(torchvision.utils.make_grid(images))
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```




```
  car plane  frog  bird
```

Construcción del Modelo CNN

Vamos a construir una CNN con la siguiente arquitectura:

- Una capa convolucional con 32 filtros, tamaño de kernel 3x3 y activación ReLU
- Una capa de max pooling con tamaño de pool 2x2
- Una segunda capa convolucional con 64 filtros, tamaño de kernel 3x3 y activación ReLU
- Una segunda capa de max pooling con tamaño de pool 2x2
- Una tercera capa convolucional con 64 filtros, tamaño de kernel 3x3 y activación ReLU
- Una capa completamente conectada con 64 unidades y activación ReLU

- Una capa de salida con 10 unidades y activación softmax

In [12]:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.conv3 = nn.Conv2d(64, 64, 3, 1)
        self.pool = nn.MaxPool2d(2, 2)
        self.flatten = nn.Flatten()

        # Calculamos la dimensión de entrada de la capa completamente conectada
        self._to_linear = None
        self.convs = nn.Sequential(
            self.conv1,
            nn.ReLU(),
            self.pool,
            self.conv2,
            nn.ReLU(),
            self.pool,
            self.conv3,
            nn.ReLU(),
            self.pool
        )
        self._get_conv_output_shape()
        self.fc1 = nn.Linear(self._to_linear, 64)
        self.fc2 = nn.Linear(64, 10)

    def _get_conv_output_shape(self):
        with torch.no_grad():
            x = torch.zeros(1, 3, 32, 32)
            x = self.convs(x)
            self._to_linear = x.numel()

    def forward(self, x):
        x = self.convs(x)
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
net = Net()
```

In [13]:
```
net
```

Out[13]:
```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (convs): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Linear(in_features=256, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=10, bias=True)
)
```

Compilación y Entrenamiento del Modelo

Compilamos el modelo con el optimizador Adam, función de pérdida CrossEntropyLoss y métricas de precisión. Luego, entrenamos el modelo con el conjunto de entrenamiento.

In [14]:

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)

for epoch in range(10):  # loop de entrenamiento

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # obtener las entradas; data es una lista de [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimizar
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # imprimir estadísticas
        running_loss += loss.item()
        if i % 2000 == 1999:    # imprimir cada 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 1.867
[1,  4000] loss: 1.498
[1,  6000] loss: 1.391
[1,  8000] loss: 1.331
[1, 10000] loss: 1.271
[1, 12000] loss: 1.234
[2,  2000] loss: 1.144
[2,  4000] loss: 1.127
[2,  6000] loss: 1.092
[2,  8000] loss: 1.071
[2, 10000] loss: 1.068
[2, 12000] loss: 1.060
[3,  2000] loss: 0.986
[3,  4000] loss: 0.977
[3,  6000] loss: 0.958
[3,  8000] loss: 0.968
[3, 10000] loss: 0.963
[3, 12000] loss: 0.973
[4,  2000] loss: 0.879
[4,  4000] loss: 0.881
[4,  6000] loss: 0.901
[4,  8000] loss: 0.888
[4, 10000] loss: 0.904
[4, 12000] loss: 0.907
[5,  2000] loss: 0.832
[5,  4000] loss: 0.849
[5,  6000] loss: 0.858
[5,  8000] loss: 0.844
[5, 10000] loss: 0.856
[5, 12000] loss: 0.856
[6,  2000] loss: 0.784
[6,  4000] loss: 0.775
[6,  6000] loss: 0.813
[6,  8000] loss: 0.803
[6, 10000] loss: 0.844
[6, 12000] loss: 0.826
[7,  2000] loss: 0.735
[7,  4000] loss: 0.768
[7,  6000] loss: 0.778
[7,  8000] loss: 0.793
[7, 10000] loss: 0.790
[7, 12000] loss: 0.798
[8,  2000] loss: 0.720
```

```
[8,  4000] loss: 0.762
[8,  6000] loss: 0.731
[8,  8000] loss: 0.749
[8, 10000] loss: 0.791
[8, 12000] loss: 0.775
[9,  2000] loss: 0.692
[9,  4000] loss: 0.718
[9,  6000] loss: 0.714
[9,  8000] loss: 0.732
[9, 10000] loss: 0.742
[9, 12000] loss: 0.748
[10,  2000] loss: 0.678
[10,  4000] loss: 0.687
[10,  6000] loss: 0.708
[10,  8000] loss: 0.717
[10, 10000] loss: 0.722
[10, 12000] loss: 0.728
Finished Training
```

Evaluación del Modelo

Evaluamos el modelo entrenado en el conjunto de prueba y visualizamos los resultados.

In [15]:
```python
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Precisión en el conjunto de prueba: {100 * correct / total} %')
```

```
Precisión en el conjunto de prueba: 69.09 %
```

Visualización de Resultados

In [ ]:
```
# Debido a la estructura de PyTorch, las estadísticas del entrenamiento ya fueron impresas durante el proceso
# Si se desea guardar y graficar las estadísticas, se puede modificar el loop de entrenamiento para almacenar
```

```
gantt
    title Plan de Acción (18 meses máximo)
    dateFormat  MM
    axisFormat %m
    section Investigación de Mercado
    Fase 1          :active, a1, 1, 3
    section Magíster por Facultad
    Fase 2          :active, a2, 4, 6
    section Diplomados Online
    Fase 3          :active, a3, 4, 6
    section Escuela de Postgrado
    Fase 4          :active, a4, 7, 12
    section Modernización Administrativa
    Fase 5          :active, a5, 7, 12
    section Vinculación con Industria
    Fase 6          :active, a6, 1, 18
    section Articulación Postgrado-Pregrado
    Fase 7          :active, a7, 4, 6
    section Liderazgo Regional
    Fase 8          :active, a8, 1, 18
    section Educación Continua y Alumni
    Fase 9          :active, a9, 1, 18
```