Cifar10-CNN ipynb version

Image Classification using Convolutional Neural Networks(CNN) in PyTorch

1. Exploring the Cifar10 Dataset

In [2]:
```python
import os
import torch
import torchvision
import tarfile
from torchvision.datasets.utils import download_url
from torch.utils.data import random_split
```

```
c:\Users\Infornet\anaconda3\envs\kingdu\lib\site-packages\tqdm\auto.py:22: TqdmWarning: IProgress not foun
d. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
c:\Users\Infornet\anaconda3\envs\kingdu\lib\site-packages\torchvision\io\image.py:13: UserWarning: Failed
to load image Python extension:
  warn(f"Failed to load image Python extension: {e}")
```

In [3]:
```python
# Dowload the dataset
dataset_url = "https://s3.amazonaws.com/fast-ai-imageclas/cifar10.tgz"
download_url(dataset_url, '.')
```

```
Using downloaded and verified file: .\cifar10.tgz
```

In [4]:
```python
# Extract from archive
with tarfile.open('./cifar10.tgz', 'r:gz') as tar:
    tar.extractall(path='./data')
```

In [5]:
```python
data_dir = './data/cifar10'

print(os.listdir(data_dir))
classes = os.listdir(data_dir + "/train")
print(classes)
```

```
['test', 'train']
['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

두개의 폴더 내부를 살펴보면, 하나는 train, test이고 각 클래스에 동일한 수의 이미지가 있는지 확인. train은 5000개, test는 1000개

In [6]:
```python
airplane_files = os.listdir(data_dir + "/train/airplane")
print('No. of training examples for airplanes:', len(airplane_files))
print(airplane_files[:5])
```

```
No. of training examples for airplanes: 5000
['0001.png', '0002.png', '0003.png', '0004.png', '0005.png']
```

In [7]:
```python
ship_test_files = os.listdir(data_dir + "/test/ship")
print("No. of test examples for ship:", len(ship_test_files))
print(ship_test_files[:5])
```

```
No. of test examples for ship: 1000
['0001.png', '0002.png', '0003.png', '0004.png', '0005.png']
```

이미지 폴더의 class를 torchvision을 통해 pytorch tensor로 로드해본다.

In [8]:
```python
from torchvision.datasets import ImageFolder
from torchvision.transforms import ToTensor
```

In [9]:
```python
dataset = ImageFolder(data_dir+'/train', transform = ToTensor())
```

각 원소는 튜플이고 이미지 텐서와 라벨을 갖고있다.
cf. 튜플 : 순서가 있는 객체의 집합으로 리스트와 유사하다. 하지만 값 변경 X

- 이미지 텐서는 32*32 pixel 에 channel은 3(RGB)라서 각 이미지 shape = (3,32,32)

In [10]:
```python
img, label = dataset[0]
print(img.shape, label)
img
```

Out[10]:
```
torch.Size([3, 32, 32]) 0
tensor([[[0.7922, 0.7922, 0.8000,  ..., 0.8118, 0.8039, 0.7961],
         [0.8078, 0.8078, 0.8118,  ..., 0.8235, 0.8157, 0.8078],
         [0.8235, 0.8275, 0.8314,  ..., 0.8392, 0.8314, 0.8235],
         ...,
         [0.8549, 0.8235, 0.7608,  ..., 0.9529, 0.9569, 0.9529],
         [0.8588, 0.8510, 0.8471,  ..., 0.9451, 0.9451, 0.9451],
         [0.8510, 0.8471, 0.8510,  ..., 0.9373, 0.9373, 0.9412]],

        [[0.8000, 0.8000, 0.8078,  ..., 0.8157, 0.8078, 0.8000],
         [0.8157, 0.8157, 0.8196,  ..., 0.8275, 0.8196, 0.8118],
         [0.8314, 0.8353, 0.8392,  ..., 0.8392, 0.8353, 0.8275],
         ...,
         [0.8510, 0.8196, 0.7608,  ..., 0.9490, 0.9490, 0.9529],
         [0.8549, 0.8471, 0.8471,  ..., 0.9412, 0.9412, 0.9412],
         [0.8471, 0.8431, 0.8471,  ..., 0.9333, 0.9333, 0.9333]],

        [[0.7804, 0.7804, 0.7882,  ..., 0.7843, 0.7804, 0.7765],
         [0.7961, 0.7961, 0.8000,  ..., 0.8039, 0.7961, 0.7882],
         [0.8118, 0.8157, 0.8235,  ..., 0.8235, 0.8157, 0.8078],
         ...,
         [0.8706, 0.8392, 0.7765,  ..., 0.9686, 0.9686, 0.9686],
         [0.8745, 0.8667, 0.8627,  ..., 0.9608, 0.9608, 0.9608],
         [0.8667, 0.8627, 0.8667,  ..., 0.9529, 0.9529, 0.9529]]])
```

In [11]:
```python
print(dataset.classes)
```

```
['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

여기 ipynb 상에서 시각화를 하고싶다면 matplotlib을 써야하는데 이때는 텐서 차원을
(32x32x3)으로 바꿔야함

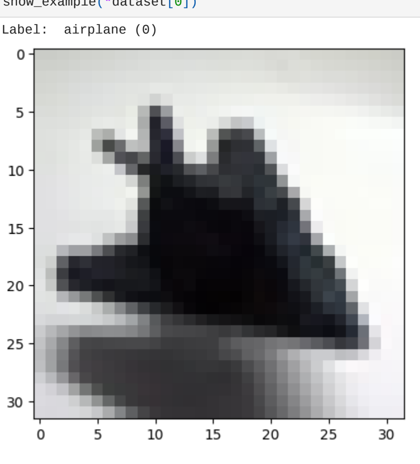
In [12]:
```python
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
          # 요건 그림 위에 수식 없이 바로 보여지게끔 하는 코딩
matplotlib.rcParams['figure.facecolor'] = '#ffffff'
```

In [13]:
```python
def show_example(img, label) :
    print('Label: ', dataset.classes[label], "(" + str(label) + ")")
    plt.imshow(img.permute(1,2,0))

# """
# # Permute() : 모든 차원들을 맞교환할 수 있음.
# ex)
# x = torch.rand(16, 32, 3)
# y = x.permute(2,1,0)    --> [3, 32, 16]
# """
```

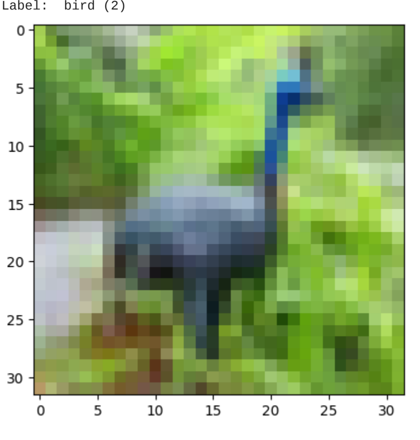
In [14]:
```python
show_example(*dataset[0])
```

```
Label:  airplane (0)
```



In [15]:
```python
show_example(*dataset[10000])
```

```
Label:  bird (2)
```



Train, Valid Split

In [16]:
```python
random_seed = 42
torch.manual_seed(random_seed);
```

In [17]:
```python
val_size = 5000
train_size = len(dataset) - val_size

train_ds, val_ds = random_split(dataset, [train_size, val_size])
len(train_ds), len(val_ds)
```

Out[17]:
```
(45000, 5000)
```

In [18]:
```python
from torch.utils.data.dataloader import DataLoader

batch_size=128
```

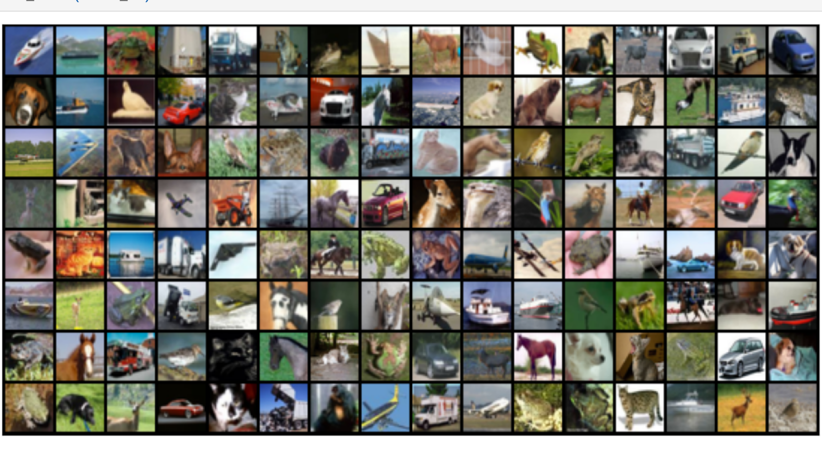
배치사이즈에 load 시켜주려면 data loader가 필요하니까

In [19]:
```python
train_dl = DataLoader(train_ds, batch_size, shuffle=True, num_workers=4, pin_memory=True)
val_dl = DataLoader(val_ds, batch_size*2, num_workers=4, pin_memory=True)
```

In [20]:
```python
from torchvision.utils import make_grid

def show_batch(dl):
    for images, labels in dl:
        fig, ax = plt.subplots(figsize=(12,6))
        ax.set_xticks([]); ax.set_yticks([])
        ax.imshow(make_grid(images, nrow=16).permute(1, 2, 0))
        break
```

In [21]:
```python
show_batch(train_dl)
```



Defining the Model(Convolutional Neural Network)

In [22]:
```python
def apply_kernel(image, kernel):
    ri, ci = image.shape        # image dimensions
    rk, ck = kernel.shape       # kernel dimensions
    ro, co = ri-rk+1, ci-ck+1   # output dimensions
    output = torch.zeros([ro, co])
    for i in range(ro):
        for j in range(co):
            output[i,j] = torch.sum(image[i:i+rk, j:j+ck] * kernel)

        return output
```

In [23]:
```python
import torch.nn as nn
import torch.nn.functional as F
```

In [24]:
```python
# CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size,
#                       stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=N

simple_model = nn.Sequential(        # input channel : 3, out_channel:
    nn.Conv2d(3, 8, kernel_size=3, stride=1, padding=1),
    nn.MaxPool2d(2, 2)
)
```

In [25]:
```python
for images, labels in train_dl:
    print('images.shape:', images.shape)
    out = simple_model(images)
    print('out.shape:', out.shape)
    break
```

```
images.shape: torch.Size([128, 3, 32, 32])
out.shape: torch.Size([128, 8, 16, 16])
```

In [26]:
```python
class ImageClassificationBase(nn.Module):
    def training_step(self, batch):
        images, labels = batch
        out = self(images)                  # Generate predictions
        loss = F.cross_entropy(out, labels) # Calculate loss
        return loss

    def validation_step(self, batch):
        images, labels = batch
        out = self(images)                    # Generate predictions
        loss = F.cross_entropy(out, labels)   # Calculate loss
        acc = accuracy(out, labels)           # Calculate accuracy
        return {'val_loss': loss.detach(), 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()   # Combine losses
        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean()      # Combine accuracies
        return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

    def epoch_end(self, epoch, result):
        print("Epoch [{}], train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.4f}".format(
            epoch, result['train_loss'], result['val_loss'], result['val_acc']))

def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))
```

nn.sequential 을 사용하여 층과 활성화함수를 하나의 network architecture로 연결시킬것이다.

In [27]:
```python
# torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1,
#                 padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None

class Cifar10CnnModel(ImageClassificationBase):
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 64 x 16 x 16

            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 128 x 8 x 8

            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 256 x 4 x 4

            nn.Flatten(),
            nn.Linear(256*4*4, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 10))

    def forward(self, xb):
        return self.network(xb)
```

In [28]:
```python
model = Cifar10CnnModel()
model
```

Out[28]:
```
Cifar10CnnModel(
  (network): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU()
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU()
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU()
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU()
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (15): Flatten(start_dim=1, end_dim=-1)
    (16): Linear(in_features=4096, out_features=1024, bias=True)
    (17): ReLU()
    (18): Linear(in_features=1024, out_features=512, bias=True)
    (19): ReLU()
    (20): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

In [29]:
```python
for images, labels in train_dl:
    print('images.shape:', images.shape)
    out = model(images)
    print('out.shape:', out.shape)
    print('out[0]:', out[0])
    break
```

```
images.shape: torch.Size([128, 3, 32, 32])
out.shape: torch.Size([128, 10])
out[0]: tensor([ 0.0239, -0.0466,  0.0067,  0.0193,  0.0044, -0.0598, -0.0188, -0.0242,
         0.0431, -0.0164], grad_fn=<SelectBackward0>)
```

GPU를 원활하게 사용하기 위해 사용 가능한 경우 몇 가지 도우미 함수(get_default_device 및 to_device)와 도우미 클래스 DeviceDataLoader를 정의하여 필요에 따라 모델 및 데이터를 GPU로 이동시킨다!

In [30]:
```python
def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

def to_device(data, device):
    """Move tensor(s) to chosen device"""
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():
    """Wrap a dataloader to move data to a device"""
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        """Yield a batch of data after moving it to device"""
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        """Number of batches"""
        return len(self.dl)
```

In [31]:
```python
device = get_default_device()
device
```

Out[31]:
```
device(type='cuda')
```

이제 DeviceDataLoader를 사용하여 데이터 배치를 GPU로 자동 전송(사용 가능한 경우)하고 to_device를 사용하여 모델을 GPU(사용 가능한 경우)로 이동하기 위해 교육 및 검증 데이터 로더를 래핑할 수 있습니다.

In [32]:
```python
train_dl = DeviceDataLoader(train_dl, device)
val_dl = DeviceDataLoader(val_dl, device)
to_device(model, device);
```

Training the Model

In [33]:
```python
@torch.no_grad()
def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.SGD):
    history = []
    optimizer = opt_func(model.parameters(), lr)
    for epoch in range(epochs):
        # Training Phase
        model.train()
        train_losses = []
        for batch in train_loader:
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
        # Validation phase
        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        model.epoch_end(epoch, result)
        history.append(result)
    return history
```

In [34]:
```python
model = to_device(Cifar10CnnModel(), device)
```

In [35]:
```python
evaluate(model, val_dl)
```

Out[35]:
```
{'val_loss': 2.302245855331421, 'val_acc': 0.10039062798023224}
```

무작위에서 뽑았을때(초기정확도)는 10% 이며, 우리는 모델을 훈련하기 위해 하이퍼파라미터(학습속도, epoch수, batch_size 등)를 사용할 것이다. -> 높은 정확도를 위하여

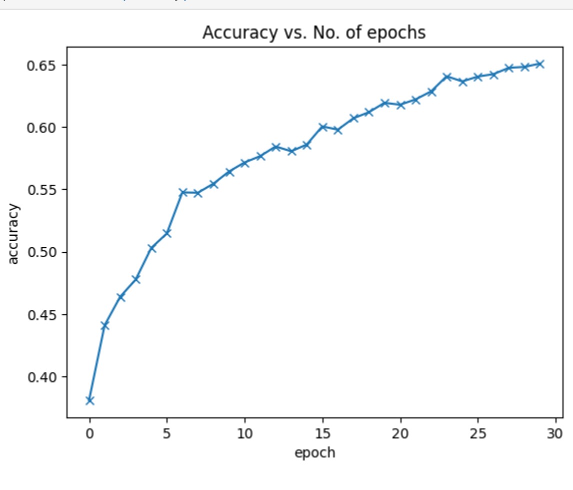
In [36]:
```python
num_epochs = 30
opt_func = torch.optim.Adam
lr = 0.001
```

In [37]:
```python
history = fit(num_epochs, lr, model, train_dl, val_dl, opt_func)
```

```
Epoch [0], train_loss: 1.8620, val_loss: 1.6588, val_acc: 0.3808
Epoch [1], train_loss: 1.5650, val_loss: 1.5005, val_acc: 0.4409
Epoch [2], train_loss: 1.4717, val_loss: 1.4334, val_acc: 0.4638
Epoch [3], train_loss: 1.4029, val_loss: 1.4123, val_acc: 0.4777
Epoch [4], train_loss: 1.3518, val_loss: 1.3767, val_acc: 0.5030
Epoch [5], train_loss: 1.3078, val_loss: 1.3254, val_acc: 0.5146
Epoch [6], train_loss: 1.2707, val_loss: 1.2587, val_acc: 0.5475
Epoch [7], train_loss: 1.2387, val_loss: 1.2567, val_acc: 0.5473
Epoch [8], train_loss: 1.2103, val_loss: 1.2291, val_acc: 0.5545
Epoch [9], train_loss: 1.1853, val_loss: 1.2028, val_acc: 0.5643
Epoch [10], train_loss: 1.1628, val_loss: 1.1866, val_acc: 0.5715
Epoch [11], train_loss: 1.1422, val_loss: 1.1849, val_acc: 0.5766
Epoch [12], train_loss: 1.1229, val_loss: 1.1442, val_acc: 0.5844
Epoch [13], train_loss: 1.1048, val_loss: 1.1740, val_acc: 0.5805
Epoch [14], train_loss: 1.0883, val_loss: 1.1482, val_acc: 0.5859
Epoch [15], train_loss: 1.0716, val_loss: 1.1028, val_acc: 0.6005
Epoch [16], train_loss: 1.0598, val_loss: 1.1180, val_acc: 0.5978
Epoch [17], train_loss: 1.0434, val_loss: 1.0900, val_acc: 0.6070
Epoch [18], train_loss: 1.0304, val_loss: 1.0842, val_acc: 0.6119
Epoch [19], train_loss: 1.0169, val_loss: 1.0683, val_acc: 0.6194
Epoch [20], train_loss: 1.0055, val_loss: 1.0588, val_acc: 0.6178
Epoch [21], train_loss: 0.9903, val_loss: 1.0579, val_acc: 0.6221
Epoch [22], train_loss: 0.9800, val_loss: 1.0426, val_acc: 0.6286
Epoch [23], train_loss: 0.9674, val_loss: 1.0266, val_acc: 0.6407
Epoch [24], train_loss: 0.9570, val_loss: 1.0226, val_acc: 0.6365
Epoch [25], train_loss: 0.9472, val_loss: 1.0106, val_acc: 0.6404
Epoch [26], train_loss: 0.9380, val_loss: 1.0077, val_acc: 0.6424
Epoch [27], train_loss: 0.9272, val_loss: 1.0063, val_acc: 0.6474
Epoch [28], train_loss: 0.9178, val_loss: 1.0028, val_acc: 0.6482
Epoch [29], train_loss: 0.9101, val_loss: 0.9973, val_acc: 0.6509
```

In [38]:
```python
def plot_accuracies(history):
    accuracies = [x['val_acc'] for x in history]
    plt.plot(accuracies, '-x')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.title('Accuracy vs. No. of epochs');
```

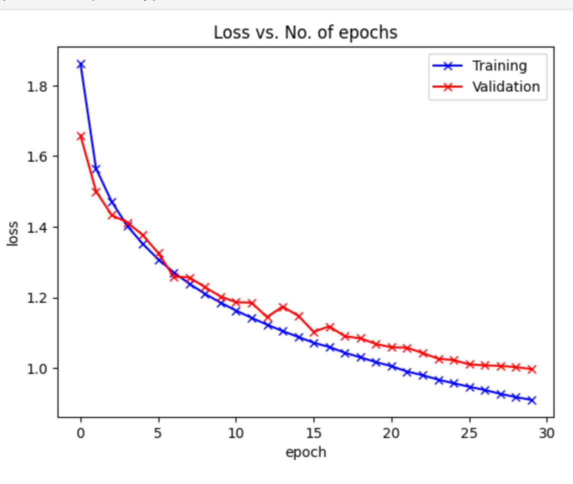
In [39]:
```python
plot_accuracies(history)
```



이 모델은 약 65% 이상 정도의 정확도에 도달하였고, epoch을 더 돌린다 하더라도 정확도가 올라갈 것 같지 않다.

In [40]:
```python
def plot_losses(history):
    train_losses = [x.get('train_loss') for x in history]
    val_losses = [x['val_loss'] for x in history]
    plt.plot(train_losses, '-bx')
    plt.plot(val_losses, '-rx')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend(['Training', 'Validation'])
    plt.title('Loss vs. No. of epochs');
```

In [41]:
```python
plot_losses(history)
```



처음에는 훈련과 검증 모두 손실이 낮아지다가 epoch 3부터는 훈련손실만 더 낮아지고 검증set 손실은 더 높아지는것을 확인할 수 있다. -> 과적합!

이 과적합을 피하기 위해 "노이즈"를 추가한다. 배치 정규화 및 드롭아웃과 같은 정규화 기술 사용을 통해.

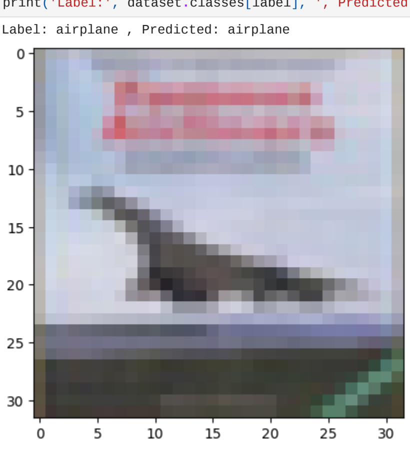
Testiong with individual images

In [42]:
```python
test_dataset = ImageFolder(data_dir+'/test', transform=ToTensor())
```

In [43]:
```python
def predict_image(img, model):
    # Convert to a batch of 1
    xb = to_device(img.unsqueeze(0), device)
    # Get predictions from model
    yb = model(xb)
    # Pick index with highest probability
    _, preds  = torch.max(yb, dim=1)
    # Retrieve the class label
    return dataset.classes[preds[0].item()]
```

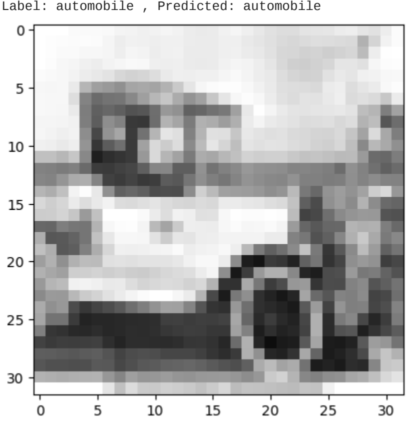
In [44]:
```python
img, label = test_dataset[0]
plt.imshow(img.permute(1, 2, 0))
print('Label:', dataset.classes[label], ', Predicted:', predict_image(img, model))
```

```
Label: airplane , Predicted: airplane
```



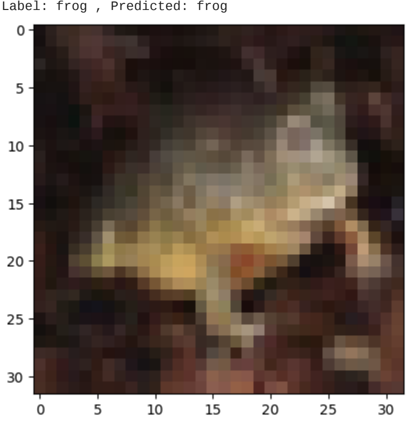
In [45]:
```python
img, label = test_dataset[1002]
plt.imshow(img.permute(1, 2, 0))
print('Label:', dataset.classes[label], ', Predicted:', predict_image(img, model))
```

```
Label: automobile , Predicted: automobile
```



In [46]:
```python
img, label = test_dataset[6153]
plt.imshow(img.permute(1, 2, 0))
print('Label:', dataset.classes[label], ', Predicted:', predict_image(img, model))
```

```
Label: frog , Predicted: frog
```



In [47]:
```python
test_loader = DeviceDataLoader(DataLoader(test_dataset, batch_size*2), device)
result = evaluate(model, test_loader)
result
```

Out[47]:
```
{'val_loss': 0.9809231162071228, 'val_acc': 0.6494140625}
```

Saving and loading the model

In [48]:
```python
torch.save(model.state_dict(), 'cifar10-cnn-acc0.7637-epoch200-lr0.001-adam-bs256.pth')
```

In [ ]: