$28 \times 28 \xrightarrow{\text{Flatten}} 900$ | W |

300×300
$\underbrace{}$
$90K$

$\underset{5 \times 900}{W} \quad \underset{900 \times 1}{x}$
$\overline{90K} \quad \overline{90K}$

Prob with

Linear layer

| Number of weights required $\propto$ input dimension |

Convolution layers $\longrightarrow$ When can we use

$\rightarrow$ shares the weights

Conv layers

Image-based problems
shift invariant

or
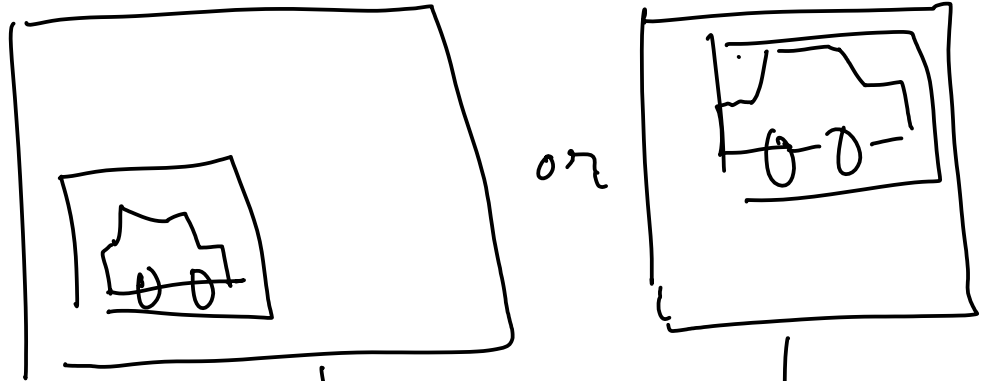
$\longrightarrow$ label $\longleftarrow$
$= car$

Image - classification

Convolution operation

$$(k \otimes f)(y) = \int_t k(t) f(y-t) dt \quad k(x)$$

Gaussian kernel

$$(k \otimes f)(y) = \sum_t k(t) f(y-t) \quad f(x)$$

weight

Conv 2D

$K=3$ Weight kernel $\leftarrow$ learn in NN

$L=3$
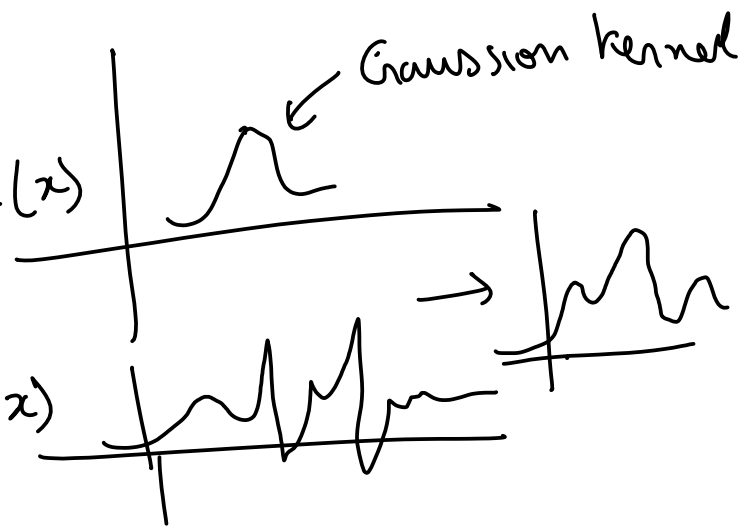
$$Y[i,j] = \sum_{k=0}^{K-1} \sum_{l=0}^{L-1} W[k,l] \underbrace{X[i-k,j-l]}$$
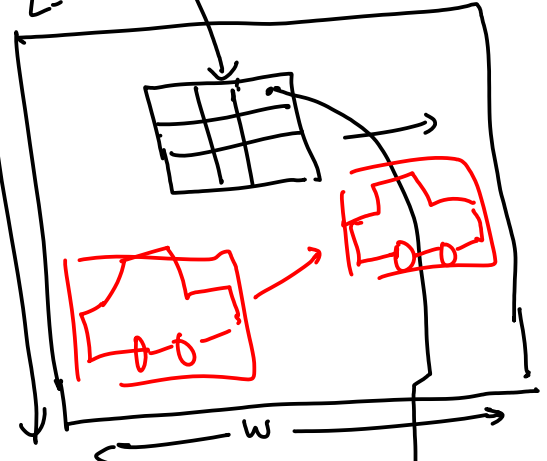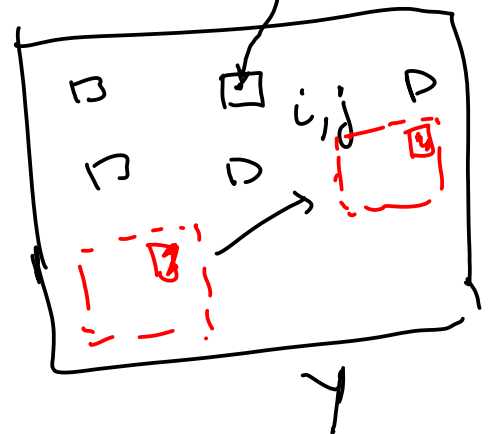
Observations

① Conv is Linear

$$f(\alpha \underline{x} + \beta \underline{y}) = \alpha f(\underline{x}) + \beta f(\underline{y}) \qquad X$$

② Conv is shift invariant

$$\text{flatten}(Y) = W_{\text{Linear}} \text{ flatten}(X)$$

You can write Conv layers as a Linear Layer

Conv $\subseteq$ Linear layer

Conv 1D

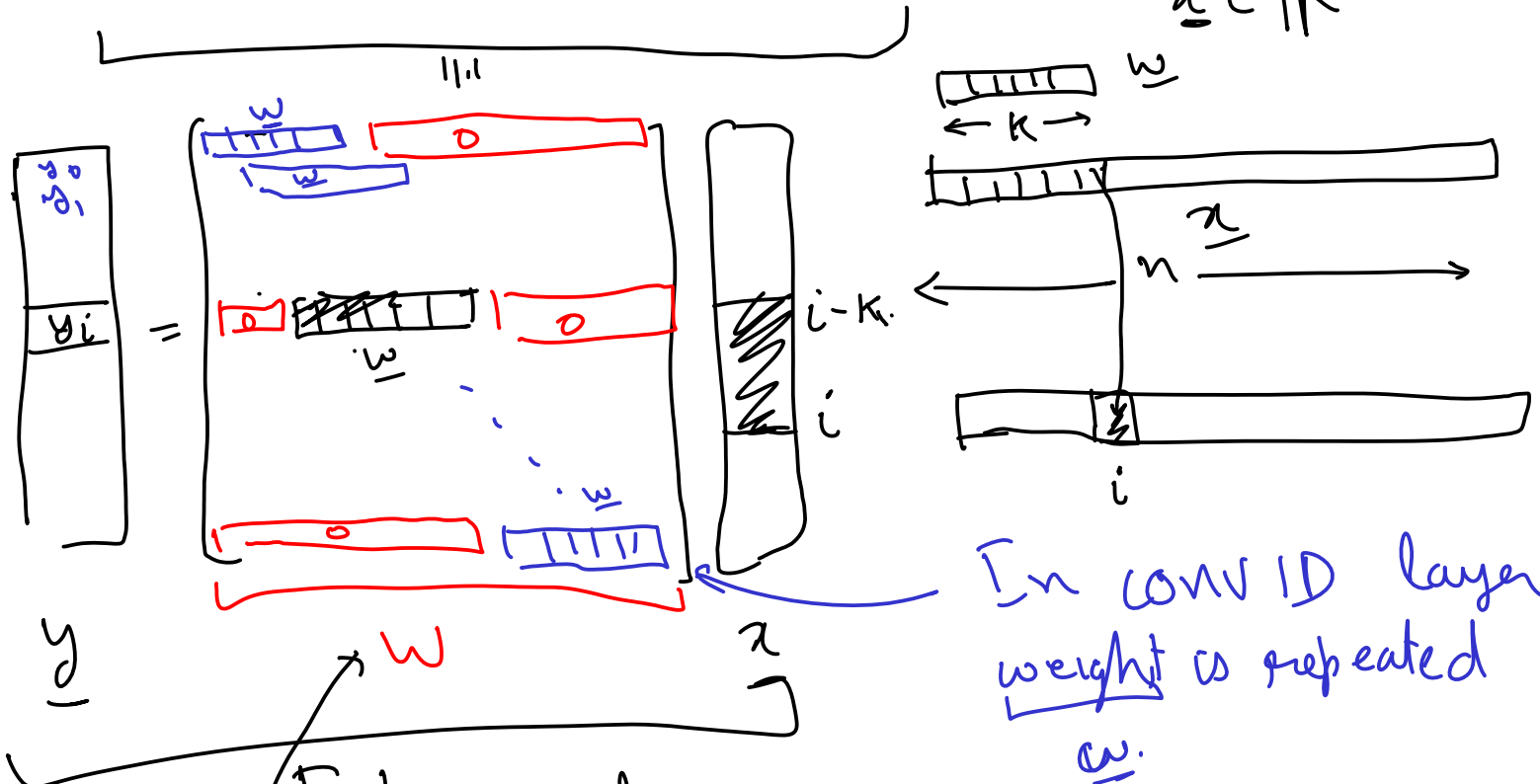$$y[i] = \sum_{k=0}^{K-1} w[k]\, x[i-k]$$

$K < n$

$y \in \mathbb{R}^{n+k-1}$

$x \in \mathbb{R}^n$

y

$= W \cdot x$

In Linear layer

Entire W is learnable with no repitition

In conv1D layer weight is repeated w.

In conv1D layer weight is repeated w.

```
In [1]: # Adapted from: Chapter 7 and 8 of Deep Learning with Pytorch by Eli Stevens
        try:
            import torch as t
            import torch.nn as tnn
        except ImportError:
            print("Colab users: pytorch comes preinstalled. Select Change Ru")
            print("Local users: Please install pytorch for your hardware using instr")
            print("ACG users: Please follow instructions here: https://vikasdhiman.i")

            raise

        if t.cuda.is_available():
            DEVICE="cuda"
        elif t.mps.is_available():
            DEVICE="mps"
        else:
            DEVICE="cpu"

        DTYPE = t.get_default_dtype()
```

```
In [2]: ## Doing it the Pytorch way without using our custom feature extraction

        import torch
        import torch.nn
        import torch.optim
        import torchvision
        from torchvision.transforms import ToTensor, Compose, Normalize
        from torch.utils.data import DataLoader

        torch.manual_seed(17)
        DATASET_MEAN = [0.4914, 0.4822, 0.4465]
        DATASET_STD = [0.2470, 0.2435, 0.2616]
        # Getting the dataset, the Pytorch way
        all_training_data = torchvision.datasets.CIFAR10(
            root="data",
            train=True,
            download=True,
            transform=Compose([ToTensor(),
                               Normalize(DATASET_MEAN, # dataset mean
                                         DATASET_STD)]) # dataset std
        )

        test_data = torchvision.datasets.CIFAR10(
            root="data",
            train=False,
            download=True,
            transform=Compose([ToTensor(),
                               Normalize(DATASET_MEAN, # dataset mean
                                         DATASET_STD)]) # dataset std
        )

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

```
In [3]:  training_data, validation_data = torch.utils.data.random_split(all_training_
```
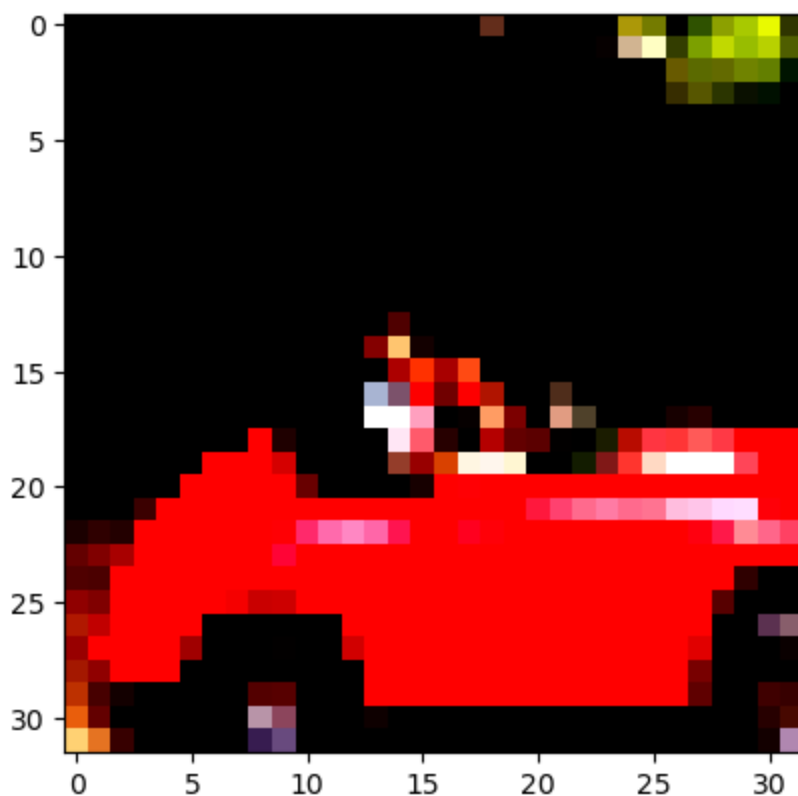
```
In [4]:  img, label = all_training_data[99]
         img.shape, label
```

```
Out[4]:  (torch.Size([3, 32, 32]), 1)
```

```
In [5]:  import matplotlib.pyplot as plt
         plt.imshow(img.permute(1, 2, 0))
```
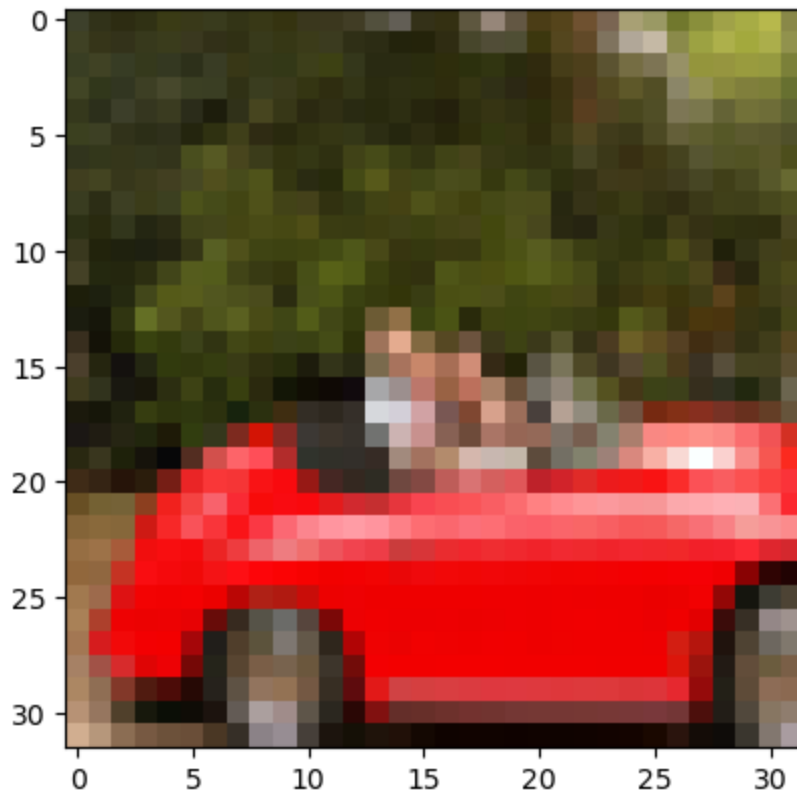
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).

```
Out[5]:  <matplotlib.image.AxesImage at 0x7efdecbcb5b0>
```



```
In [6]:  plt.imshow((img.permute(1, 2, 0) *  torch.Tensor(DATASET_STD)
                    +  torch.Tensor(DATASET_MEAN)))
```

```
Out[6]:  <matplotlib.image.AxesImage at 0x7efdecb04940>
```

```
In [7]: imgs = torch.stack([img_t for img_t, _ in all_training_data], dim=3)
        imgs.reshape(3, -1).mean(dim=-1), imgs.reshape(3, -1).std(dim=-1)
```

```
Out[7]: (tensor([-1.2762e-06, -1.7074e-04,  1.1819e-04]),
          tensor([1.0001, 0.9999, 1.0000]))
```

```
In [8]: import pickle
        cifar_meta = pickle.load(open("data/cifar-10-batches-py/batches.meta", "rb")
        class_names = [c.decode('utf-8') for c in cifar_meta[b'label_names']]
        class_names
```

```
Out[8]: ['airplane',
          'automobile',
          'bird',
          'cat',
          'deer',
          'dog',
          'frog',
          'horse',
          'ship',
          'truck']
```

```
In [9]: # Hyper parameters
        learning_rate = 1e-3 # controls how fast the gradient descent goes
        batch_size = 64
        epochs = 5
        momentum = 0.9

        training_dataloader = DataLoader(training_data, shuffle=True, batch_size=bat
        validation_dataloader = DataLoader(validation_data,  batch_size=batch_size)
        test_dataloader = DataLoader(test_data,  batch_size=batch_size)
```

```python
X, y = next(iter(training_dataloader))
X.shape
```

Out[9]: torch.Size([64, 3, 32, 32])

In [10]:
```python
!pip install tensorboard
```

Requirement already satisfied: tensorboard in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (2.12.0)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from tensorboard) (0.4.6)
Requirement already satisfied: grpcio>=1.48.2 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from tensorboard) (1.53.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from tensorboard) (0.7.0)
Requirement already satisfied: requests<3,>=2.21.0 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from tensorboard) (2.28.1)
Requirement already satisfied: markdown>=2.6.8 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from tensorboard) (3.4.3)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from tensorboard) (1.8.1)
Requirement already satisfied: google-auth<3,>=1.6.3 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from tensorboard) (2.17.0)
Requirement already satisfied: wheel>=0.26 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from tensorboard) (0.37.1)
Requirement already satisfied: setuptools>=41.0.0 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from tensorboard) (65.6.3)
Requirement already satisfied: werkzeug>=1.0.1 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from tensorboard) (2.2.3)
Requirement already satisfied: numpy>=1.12.0 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from tensorboard) (1.23.5)
Requirement already satisfied: protobuf>=3.19.6 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from tensorboard) (4.22.1)
Requirement already satisfied: absl-py>=0.4 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from tensorboard) (1.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from google-auth<3,>=1.6.3->tensorboard) (4.9)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from google-auth<3,>=1.6.3->tensorboard) (5.3.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from google-auth<3,>=1.6.3->tensorboard) (0.2.8)
Requirement already satisfied: six>=1.9.0 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from google-auth<3,>=1.6.3->tensorboard) (1.16.0)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard) (1.3.1)
Requirement already satisfied: certifi>=2017.4.17 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorboard) (2022.12.7)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/vdhiman/.local/miniconda3/envs/ece490/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorboard) (1.26.14)

```
Requirement already satisfied: charset-normalizer<3,>=2 in /home/vdhiman/.l
ocal/miniconda3/envs/ece490/lib/python3.10/site-packages (from requests<3,>
=2.21.0->tensorboard) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /home/vdhiman/.local/minicon
da3/envs/ece490/lib/python3.10/site-packages (from requests<3,>=2.21.0->ten
sorboard) (3.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in /home/vdhiman/.local/mi
niconda3/envs/ece490/lib/python3.10/site-packages (from werkzeug>=1.0.1->te
nsorboard) (2.1.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /home/vdhiman/.loca
l/miniconda3/envs/ece490/lib/python3.10/site-packages (from pyasn1-modules>
=0.2.1->google-auth<3,>=1.6.3->tensorboard) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in /home/vdhiman/.local/mini
conda3/envs/ece490/lib/python3.10/site-packages (from requests-oauthlib>=0.
7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard) (3.2.2)
```

In [17]: `%tensorboard --logdir=runs`

```
Reusing TensorBoard on port 6006 (pid 543069), started 0:05:49 ago. (Use '!
kill 543069' to kill it.)
```

```
In [22]:  from torch.utils.tensorboard import SummaryWriter
          import os
          writer = SummaryWriter()

          loss = torch.nn.CrossEntropyLoss()
          # TODO:
          # Define model = ?
```

```python
model = tnn.Sequential(
    torch.nn.Flatten(),
    tnn.Linear(3*32*32, 100),
    tnn.ReLU(),
    tnn.Linear(100, 10))


# Define optimizer
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=m

def loss_and_accuracy(model, loss, validation_dataloader, device=DEVICE):
        # Validation loop
        validation_size = len(validation_dataloader.dataset)
        num_batches = len(validation_dataloader)
        test_loss, correct = 0, 0

        with torch.no_grad():
            model.eval() # Put model in eval mode, affects layers like dropo
            for X, y in validation_dataloader:
                X = X.to(device)
                y = y.to(device)
                pred = model(X)
                test_loss += loss(pred, y)
                correct += (pred.argmax(dim=-1) == y).type(DTYPE).sum()

        test_loss /= num_batches
        correct /= validation_size
        return test_loss, correct

def train(model, loss, training_dataloader, validation_dataloader, device=DE
    model.to(device)
    t0 = 0
    if os.path.exists("runs/model_ckpt.pt"):
        checkpoint = torch.load("runs/model_ckpt.pt")
        model.load_state_dict(checkpoint['model_state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
        t0 = checkpoint['epoch']

    for t in range(t0, epochs):
        # Train loop
        training_size = len(training_dataloader.dataset)
        nbatches = len(training_dataloader)
        model.train() # Put model in train mode, affects layers like dropout
        for batch, (X, y) in enumerate(training_dataloader):
            X = X.to(device)
            y = y.to(device)
            # Compute prediction and loss
            pred = model(X)
            loss_t = loss(pred, y)

            # Backpropagation
            optimizer.zero_grad()
            loss_t.backward()
            optimizer.step()

            if batch % 100 == 0:
```

```
                writer.add_scalar("Train/loss_batch", loss_t,  t*nbatches +
                loss_t, current = loss_t.item(), (batch + 1) * len(X)
                print(f"loss: {loss_t:>7f}  [{current:>5d}/{training_size:>5

        writer.add_scalar("Train/loss", loss_t, t)
        valid_loss, correct = loss_and_accuracy(model, loss, validation_data
        writer.add_scalar("Valid/loss", valid_loss, t)
        writer.add_scalar("Valid/accuracy", correct, t)
        print(f"Validation Error: \n Accuracy: {(100*correct):>0.1f}%, Avg l
        if t % 3 == 0:
            torch.save({
                'epoch': t,
                'model_state_dict': model.state_dict(),
                'optimizer_state_dict': optimizer.state_dict()
                }, "runs/model_ckpt.pt")
    return model

trained_model = train(model, loss, training_dataloader, validation_dataloade

test_loss, correct = loss_and_accuracy(model, loss, test_dataloader)
print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_los
```

```
Validation Error: 4864/45000]
 Accuracy: 48.4%, Avg loss: 1.453683

Validation Error: 4864/45000]
 Accuracy: 49.2%, Avg loss: 1.440753

Test Error:
 Accuracy: 50.3%, Avg loss: 1.427571
```

Limits of fully connected layers

Convolutions bring translation invariance

In [32]:
```python
conv = torch.nn.Conv2d(3, 1, kernel_size=3)
conv
```

Out[32]: Conv2d(3, 1, kernel_size=(3, 3), stride=(1, 1))
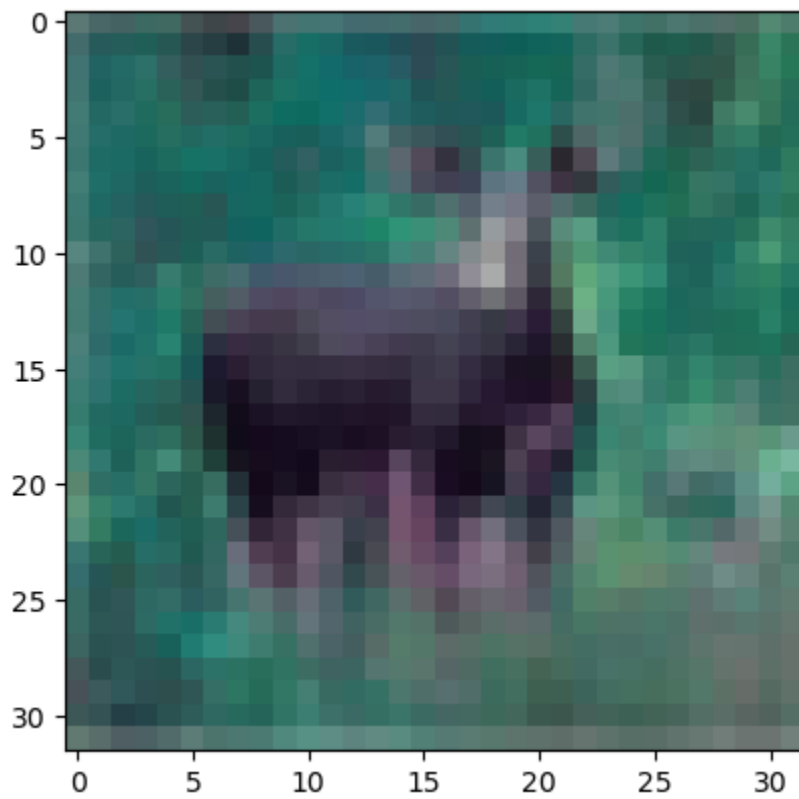
In [33]:
```python
img, label = training_data[99]
plt.imshow(img.permute(1, 2, 0) * t.Tensor(DATASET_STD) + t.Tensor(DATASET_M
```

Out[33]: <matplotlib.image.AxesImage at 0x7efd51214ee0>

Learning with Convolutions

Maxpooling