AML: 04 Convolutional Neural Network for CIFAR-10

Based on https://github.com/Atcold/pytorch-Deep-Learning

Data and Libraries

In [7]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
from torchvision import datasets, transforms, utils
import matplotlib.pyplot as plt
import numpy, random


# set the PseudoRandom Generator Seeds for better reproducibility
# see here for more: https://pytorch.org/docs/stable/notes/randomnes
s.html
torch.manual_seed(99)
random.seed(99)
numpy.random.seed(99)

# this 'device' will be used for training our model
device = torch.device("cuda:0" if torch.cuda.is_available() else "cp
u")
print(device)
```

```
cuda:0
```

Load the CIFAR10 dataset

Observe that we set `shuffle=True`, which means that data is randomized

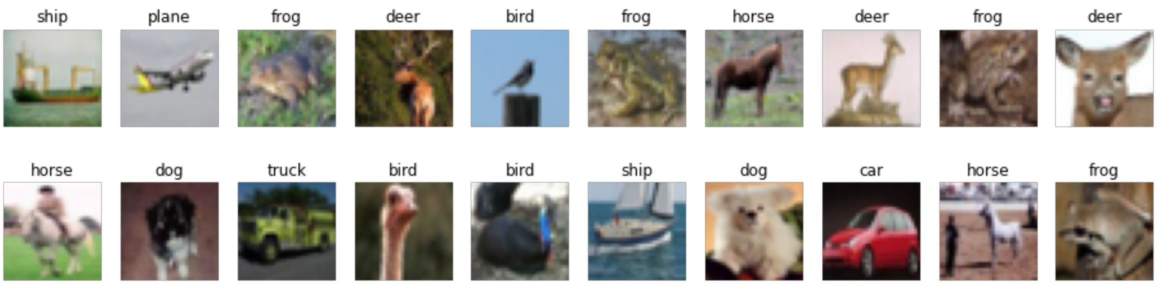
In [8]:

```
input_size  = 32*32*3   # images are 32x32 pixels with 3 channels
output_size = 10      # there are 10 classes

train_loader = torch.utils.data.DataLoader(
    datasets.CIFAR10('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.5, 0.5, 0.5), (0.5,
0.5, 0.5))
                   ])),
    batch_size=64, shuffle=True)

test_loader = torch.utils.data.DataLoader(
    datasets.CIFAR10('../data', train=False, transform=transforms.Com
pose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.5, 0.5, 0.5), (0.5,
0.5, 0.5))
                   ])),
    batch_size=1000, shuffle=True)

classNames= ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'h
orse', 'ship', 'truck']
```

```
Files already downloaded and verified
```

In [9]:

```
# show some training images
def imshow(img, plot):
    img = img / 2 + 0.5  # unnormalize
    npimg = img.numpy()   # convert from tensor
    plot.imshow(numpy.transpose(npimg, (1, 2, 0)))


plt.figure(figsize=(16,4))

# fetch a batch of train images; RANDOM
image_batch, label_batch = next(iter(train_loader))
#imshow(torchvision.utils.make_grid(image_batch))
for i in range(20):
    image = image_batch[i]
    label = classNames[label_batch[i].item()]
    plt.subplot(2, 10, i + 1)
    #image, label = train_loader.dataset.__getitem__(i)
    #plt.imshow(image.squeeze().numpy())
    imshow(image, plt)
    plt.axis('off')
    plt.title(label)
plt.show()
```



A 2-hidden layer Fully Connected Neural Network

Helper functions for training and testing

In [10]:

```
# function to count number of parameters
def get_n_params(model):
    np=0
    for p in list(model.parameters()):
        np += p.nelement()
    return np

accuracy_list = []
# we pass a model object to this trainer, and it trains this model for
one epoch
def train(epoch, model):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        # send to device
        data, target = data.to(device), target.to(device)

        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.f
ormat(
                epoch, batch_idx * len(data), len(train_loader.datase
t),
                100. * batch_idx / len(train_loader), loss.item()))

def test(model):
    model.eval()
    test_loss = 0
    correct = 0
    for data, target in test_loader:
        # send to device
        data, target = data.to(device), target.to(device)

        output = model(data)
        test_loss += F.nll_loss(output, target, reduction='sum').item
() # sum up batch loss
        pred = output.data.max(1, keepdim=True)[1] # get the index of
the max log-probability
        correct += pred.eq(target.data.view_as(pred)).cpu().sum().ite
```

```
m()

    test_loss /= len(test_loader.dataset)
    accuracy = 100. * correct / len(test_loader.dataset)
    accuracy_list.append(accuracy)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0
f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        accuracy))
```

A small Convolutional Neural Network

In [11]:

```
class CNN(nn.Module):
    def __init__(self, input_size, output_size):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=50, kernel
_size=3, padding=0)
        self.conv2 = nn.Conv2d(in_channels=50, out_channels=50, kerne
l_size=6, padding=0)
        self.conv3 = nn.Conv2d(in_channels=50, out_channels=50, kerne
l_size=6, padding=0)
        self.FLATTEN_LEN=50*3*3
        self.fc1 = nn.Linear(self.FLATTEN_LEN, 10)
        #self.fc2 = nn.Linear(200, 10)

    def forward(self, x, verbose=False):
        #print("input", x.shape)

        x = self.conv1(x)
        x = F.relu(x)
        #print("after conv1", x.shape)

        x = self.conv2(x)
        x = F.relu(x)
        #print("after conv2", x.shape)

        x = F.max_pool2d(x, kernel_size=2)
        #print("after 1st maxpool", x.shape)

        x = self.conv3(x)
        x = F.relu(x)
        #print("after conv3", x.shape)

        x = F.max_pool2d(x, kernel_size=2)
        #print("after 2nd maxpool", x.shape)

        x = x.view(-1, self.FLATTEN_LEN)
        #print("after tensor shape change", x.shape)

        x = self.fc1(x)
        #x = F.relu(x)
        #print("after fc1", x.shape)

        #x = self.fc2(x)
```

```
        ##print("after fc2", x.shape)

        x = F.log_softmax(x, dim=1)
        return x
```

Sample output of tensor shapes with above print command

```
Training on  cuda:0
Number of parameters: 290890
input torch.Size([64, 3, 32, 32])
after conv1 torch.Size([64, 20, 30, 30])
after conv2 torch.Size([64, 40, 25, 25])
after 1st maxpool torch.Size([64, 40, 12, 12])
after conv3 torch.Size([64, 80, 7, 7])
after 2nd maxpool torch.Size([64, 80, 3, 3])
after tensor shape change torch.Size([64, 720])
after fc1 torch.Size([64, 200])
after fc2 torch.Size([64, 10])
```

In [12]:

```
print("Training on ", device)
model_cnn = CNN(input_size, output_size)
model_cnn.to(device)
optimizer = optim.SGD(model_cnn.parameters(), lr=0.01, momentum=0.5)
print('Number of parameters: {}'.format(get_n_params(model_cnn)))

for epoch in range(0, 10):
    train(epoch, model_cnn)
    test(model_cnn)
```

```
Training on  cuda:0
Number of parameters: 186010
Train Epoch: 0 [0/50000 (0%)]	Loss: 2.308862
Train Epoch: 0 [6400/50000 (13%)]	Loss: 2.129453
Train Epoch: 0 [12800/50000 (26%)]	Loss: 2.006634
Train Epoch: 0 [19200/50000 (38%)]	Loss: 1.840536
Train Epoch: 0 [25600/50000 (51%)]	Loss: 1.971598
Train Epoch: 0 [32000/50000 (64%)]	Loss: 1.737201
Train Epoch: 0 [38400/50000 (77%)]	Loss: 1.844822
Train Epoch: 0 [44800/50000 (90%)]	Loss: 1.590996

Test set: Average loss: 1.6675, Accuracy: 3981/10000 (40%)

Train Epoch: 1 [0/50000 (0%)]	Loss: 1.702338
Train Epoch: 1 [6400/50000 (13%)]	Loss: 1.463339
Train Epoch: 1 [12800/50000 (26%)]	Loss: 1.544069
Train Epoch: 1 [19200/50000 (38%)]	Loss: 1.727243
Train Epoch: 1 [25600/50000 (51%)]	Loss: 1.441555
Train Epoch: 1 [32000/50000 (64%)]	Loss: 1.710955
Train Epoch: 1 [38400/50000 (77%)]	Loss: 1.348709
Train Epoch: 1 [44800/50000 (90%)]	Loss: 1.377605

Test set: Average loss: 1.5522, Accuracy: 4434/10000 (44%)

Train Epoch: 2 [0/50000 (0%)]	Loss: 1.600341
Train Epoch: 2 [6400/50000 (13%)]	Loss: 1.679207
Train Epoch: 2 [12800/50000 (26%)]	Loss: 1.204185
Train Epoch: 2 [19200/50000 (38%)]	Loss: 1.527676
Train Epoch: 2 [25600/50000 (51%)]	Loss: 1.645530
Train Epoch: 2 [32000/50000 (64%)]	Loss: 1.484010
Train Epoch: 2 [38400/50000 (77%)]	Loss: 1.532164
Train Epoch: 2 [44800/50000 (90%)]	Loss: 1.236119

Test set: Average loss: 1.3144, Accuracy: 5255/10000 (53%)

Train Epoch: 3 [0/50000 (0%)]	Loss: 1.403111
Train Epoch: 3 [6400/50000 (13%)]	Loss: 1.513695
Train Epoch: 3 [12800/50000 (26%)]	Loss: 1.356224
Train Epoch: 3 [19200/50000 (38%)]	Loss: 1.230550
Train Epoch: 3 [25600/50000 (51%)]	Loss: 1.159860
Train Epoch: 3 [32000/50000 (64%)]	Loss: 1.167630
Train Epoch: 3 [38400/50000 (77%)]	Loss: 1.128877
```

```
Train Epoch: 3 [44800/50000 (90%)]      Loss: 1.074079

Test set: Average loss: 1.2389, Accuracy: 5520/10000 (55%)

Train Epoch: 4 [0/50000 (0%)]   Loss: 1.176373
Train Epoch: 4 [6400/50000 (13%)]       Loss: 1.183324
Train Epoch: 4 [12800/50000 (26%)]      Loss: 1.160306
Train Epoch: 4 [19200/50000 (38%)]      Loss: 1.024318
Train Epoch: 4 [25600/50000 (51%)]      Loss: 1.273896
Train Epoch: 4 [32000/50000 (64%)]      Loss: 1.238139
Train Epoch: 4 [38400/50000 (77%)]      Loss: 1.358851
Train Epoch: 4 [44800/50000 (90%)]      Loss: 1.200196

Test set: Average loss: 1.2340, Accuracy: 5498/10000 (55%)

Train Epoch: 5 [0/50000 (0%)]   Loss: 1.189062
Train Epoch: 5 [6400/50000 (13%)]       Loss: 0.958405
Train Epoch: 5 [12800/50000 (26%)]      Loss: 0.763711
Train Epoch: 5 [19200/50000 (38%)]      Loss: 1.103243
Train Epoch: 5 [25600/50000 (51%)]      Loss: 0.943136
Train Epoch: 5 [32000/50000 (64%)]      Loss: 1.165479
Train Epoch: 5 [38400/50000 (77%)]      Loss: 1.128588
Train Epoch: 5 [44800/50000 (90%)]      Loss: 1.053414

Test set: Average loss: 1.1890, Accuracy: 5765/10000 (58%)

Train Epoch: 6 [0/50000 (0%)]   Loss: 1.188378
Train Epoch: 6 [6400/50000 (13%)]       Loss: 0.973491
Train Epoch: 6 [12800/50000 (26%)]      Loss: 1.202193
Train Epoch: 6 [19200/50000 (38%)]      Loss: 0.949433
Train Epoch: 6 [25600/50000 (51%)]      Loss: 0.862668
Train Epoch: 6 [32000/50000 (64%)]      Loss: 0.873058
Train Epoch: 6 [38400/50000 (77%)]      Loss: 0.884255
Train Epoch: 6 [44800/50000 (90%)]      Loss: 1.023986

Test set: Average loss: 1.0732, Accuracy: 6230/10000 (62%)

Train Epoch: 7 [0/50000 (0%)]   Loss: 1.063504
Train Epoch: 7 [6400/50000 (13%)]       Loss: 0.899900
Train Epoch: 7 [12800/50000 (26%)]      Loss: 1.133447
Train Epoch: 7 [19200/50000 (38%)]      Loss: 1.125349
Train Epoch: 7 [25600/50000 (51%)]      Loss: 0.851897
Train Epoch: 7 [32000/50000 (64%)]      Loss: 1.012613
```

```
Train Epoch: 7 [38400/50000 (77%)]      Loss: 0.993091
Train Epoch: 7 [44800/50000 (90%)]      Loss: 0.793250

Test set: Average loss: 1.0498, Accuracy: 6313/10000 (63%)

Train Epoch: 8 [0/50000 (0%)]   Loss: 0.914293
Train Epoch: 8 [6400/50000 (13%)]       Loss: 0.726605
Train Epoch: 8 [12800/50000 (26%)]      Loss: 0.814564
Train Epoch: 8 [19200/50000 (38%)]      Loss: 1.021785
Train Epoch: 8 [25600/50000 (51%)]      Loss: 0.803584
Train Epoch: 8 [32000/50000 (64%)]      Loss: 0.964835
Train Epoch: 8 [38400/50000 (77%)]      Loss: 1.235122
Train Epoch: 8 [44800/50000 (90%)]      Loss: 0.827518

Test set: Average loss: 1.0566, Accuracy: 6355/10000 (64%)

Train Epoch: 9 [0/50000 (0%)]   Loss: 1.188917
Train Epoch: 9 [6400/50000 (13%)]       Loss: 0.848569
Train Epoch: 9 [12800/50000 (26%)]      Loss: 0.876923
Train Epoch: 9 [19200/50000 (38%)]      Loss: 0.823473
Train Epoch: 9 [25600/50000 (51%)]      Loss: 0.964066
Train Epoch: 9 [32000/50000 (64%)]      Loss: 0.747255
Train Epoch: 9 [38400/50000 (77%)]      Loss: 0.868074
Train Epoch: 9 [44800/50000 (90%)]      Loss: 0.814660

Test set: Average loss: 1.0914, Accuracy: 6335/10000 (63%)
```

Myrtle5 Network

In [13]:

```python
class Flatten(nn.Module):
    def forward(self, x): return x.view(x.size(0), x.size(1))

class CNN(nn.Module):
    def __init__(self, input_size, output_size):
        super(CNN, self).__init__()
        self.input_size = input_size
        self.output_size = output_size
        self.C = 64
        self.network = nn.Sequential(
                # Layer 0
                nn.Conv2d(3, self.C, kernel_size=3, stride=1,
                          padding=1, bias=True),
                nn.BatchNorm2d(self.C),
                nn.ReLU(),

                # Layer 1
                nn.Conv2d(self.C, self.C*2, kernel_size=3,
                          stride=1, padding=1, bias=True),
                nn.BatchNorm2d(self.C*2),
                nn.ReLU(),
                nn.MaxPool2d(2),

                # Layer 2
                nn.Conv2d(self.C*2, self.C*4, kernel_size=3,
                          stride=1, padding=1, bias=True),
                nn.BatchNorm2d(self.C*4),
                nn.ReLU(),
                nn.MaxPool2d(2),

                # Layer 3
                nn.Conv2d(self.C*4, self.C*8, kernel_size=3,
                          stride=1, padding=1, bias=True),
                nn.BatchNorm2d(self.C*8),
                nn.ReLU(),
                nn.MaxPool2d(2),

                # Layer 4
                nn.MaxPool2d(4),
                Flatten(),
                nn.Linear(self.C*8, output_size, bias=True),
                nn.LogSoftmax(dim=1)
```

```
            )

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

Train the Network

In [14]:

```
print("Training on ", device)
model_cnn = CNN(input_size, output_size)
model_cnn.to(device)
optimizer = optim.SGD(model_cnn.parameters(), lr=0.01, momentum=0.5)
print('Number of parameters: {}'.format(get_n_params(model_cnn)))

for epoch in range(0, 10):
    train(epoch, model_cnn)
    test(model_cnn)
```

```
Training on  cuda:0
Number of parameters: 1558026
Train Epoch: 0 [0/50000 (0%)]	Loss: 2.617300
Train Epoch: 0 [6400/50000 (13%)]	Loss: 2.008634
Train Epoch: 0 [12800/50000 (26%)]	Loss: 1.510984
Train Epoch: 0 [19200/50000 (38%)]	Loss: 1.312724
Train Epoch: 0 [25600/50000 (51%)]	Loss: 1.187397
Train Epoch: 0 [32000/50000 (64%)]	Loss: 1.163763
Train Epoch: 0 [38400/50000 (77%)]	Loss: 1.171398
Train Epoch: 0 [44800/50000 (90%)]	Loss: 0.916907

Test set: Average loss: 1.1319, Accuracy: 6065/10000 (61%)

Train Epoch: 1 [0/50000 (0%)]	Loss: 1.093774
Train Epoch: 1 [6400/50000 (13%)]	Loss: 0.789880
Train Epoch: 1 [12800/50000 (26%)]	Loss: 0.822173
Train Epoch: 1 [19200/50000 (38%)]	Loss: 1.061568
Train Epoch: 1 [25600/50000 (51%)]	Loss: 0.752607
Train Epoch: 1 [32000/50000 (64%)]	Loss: 0.820241
Train Epoch: 1 [38400/50000 (77%)]	Loss: 0.632608
Train Epoch: 1 [44800/50000 (90%)]	Loss: 1.149157

Test set: Average loss: 1.3412, Accuracy: 5578/10000 (56%)

Train Epoch: 2 [0/50000 (0%)]	Loss: 1.111858
Train Epoch: 2 [6400/50000 (13%)]	Loss: 0.768074
Train Epoch: 2 [12800/50000 (26%)]	Loss: 0.734872
Train Epoch: 2 [19200/50000 (38%)]	Loss: 0.647956
Train Epoch: 2 [25600/50000 (51%)]	Loss: 0.560671
Train Epoch: 2 [32000/50000 (64%)]	Loss: 0.591244
Train Epoch: 2 [38400/50000 (77%)]	Loss: 0.740819
Train Epoch: 2 [44800/50000 (90%)]	Loss: 0.556424

Test set: Average loss: 1.3221, Accuracy: 5951/10000 (60%)

Train Epoch: 3 [0/50000 (0%)]	Loss: 0.899996
Train Epoch: 3 [6400/50000 (13%)]	Loss: 0.401533
Train Epoch: 3 [12800/50000 (26%)]	Loss: 0.579860
Train Epoch: 3 [19200/50000 (38%)]	Loss: 0.627890
Train Epoch: 3 [25600/50000 (51%)]	Loss: 0.428611
Train Epoch: 3 [32000/50000 (64%)]	Loss: 0.546866
Train Epoch: 3 [38400/50000 (77%)]	Loss: 0.626361
```

```
Train Epoch: 3 [44800/50000 (90%)]      Loss: 0.341851

Test set: Average loss: 0.6617, Accuracy: 7760/10000 (78%)

Train Epoch: 4 [0/50000 (0%)]   Loss: 0.545282
Train Epoch: 4 [6400/50000 (13%)]       Loss: 0.421444
Train Epoch: 4 [12800/50000 (26%)]      Loss: 0.377283
Train Epoch: 4 [19200/50000 (38%)]      Loss: 0.460052
Train Epoch: 4 [25600/50000 (51%)]      Loss: 0.335357
Train Epoch: 4 [32000/50000 (64%)]      Loss: 0.415782
Train Epoch: 4 [38400/50000 (77%)]      Loss: 0.424912
Train Epoch: 4 [44800/50000 (90%)]      Loss: 0.590783

Test set: Average loss: 0.7272, Accuracy: 7512/10000 (75%)

Train Epoch: 5 [0/50000 (0%)]   Loss: 0.394075
Train Epoch: 5 [6400/50000 (13%)]       Loss: 0.311530
Train Epoch: 5 [12800/50000 (26%)]      Loss: 0.416277
Train Epoch: 5 [19200/50000 (38%)]      Loss: 0.224971
Train Epoch: 5 [25600/50000 (51%)]      Loss: 0.518940
Train Epoch: 5 [32000/50000 (64%)]      Loss: 0.294123
Train Epoch: 5 [38400/50000 (77%)]      Loss: 0.256379
Train Epoch: 5 [44800/50000 (90%)]      Loss: 0.293397

Test set: Average loss: 0.7022, Accuracy: 7673/10000 (77%)

Train Epoch: 6 [0/50000 (0%)]   Loss: 0.208960
Train Epoch: 6 [6400/50000 (13%)]       Loss: 0.249800
Train Epoch: 6 [12800/50000 (26%)]      Loss: 0.125098
Train Epoch: 6 [19200/50000 (38%)]      Loss: 0.149408
Train Epoch: 6 [25600/50000 (51%)]      Loss: 0.165265
Train Epoch: 6 [32000/50000 (64%)]      Loss: 0.198070
Train Epoch: 6 [38400/50000 (77%)]      Loss: 0.240993
Train Epoch: 6 [44800/50000 (90%)]      Loss: 0.163804

Test set: Average loss: 0.7329, Accuracy: 7709/10000 (77%)

Train Epoch: 7 [0/50000 (0%)]   Loss: 0.122041
Train Epoch: 7 [6400/50000 (13%)]       Loss: 0.119472
Train Epoch: 7 [12800/50000 (26%)]      Loss: 0.077181
Train Epoch: 7 [19200/50000 (38%)]      Loss: 0.275790
Train Epoch: 7 [25600/50000 (51%)]      Loss: 0.060083
Train Epoch: 7 [32000/50000 (64%)]      Loss: 0.069926
```

```
Train Epoch: 7 [38400/50000 (77%)]      Loss: 0.113785
Train Epoch: 7 [44800/50000 (90%)]      Loss: 0.155599

Test set: Average loss: 0.6281, Accuracy: 8071/10000 (81%)

Train Epoch: 8 [0/50000 (0%)]   Loss: 0.060049
Train Epoch: 8 [6400/50000 (13%)]       Loss: 0.079626
Train Epoch: 8 [12800/50000 (26%)]      Loss: 0.030653
Train Epoch: 8 [19200/50000 (38%)]      Loss: 0.038837
Train Epoch: 8 [25600/50000 (51%)]      Loss: 0.054419
Train Epoch: 8 [32000/50000 (64%)]      Loss: 0.049076
Train Epoch: 8 [38400/50000 (77%)]      Loss: 0.035100
Train Epoch: 8 [44800/50000 (90%)]      Loss: 0.049201

Test set: Average loss: 0.5225, Accuracy: 8349/10000 (83%)

Train Epoch: 9 [0/50000 (0%)]   Loss: 0.024661
Train Epoch: 9 [6400/50000 (13%)]       Loss: 0.020419
Train Epoch: 9 [12800/50000 (26%)]      Loss: 0.023971
Train Epoch: 9 [19200/50000 (38%)]      Loss: 0.020140
Train Epoch: 9 [25600/50000 (51%)]      Loss: 0.016934
Train Epoch: 9 [32000/50000 (64%)]      Loss: 0.021431
Train Epoch: 9 [38400/50000 (77%)]      Loss: 0.025145
Train Epoch: 9 [44800/50000 (90%)]      Loss: 0.027827

Test set: Average loss: 0.5175, Accuracy: 8397/10000 (84%)
```

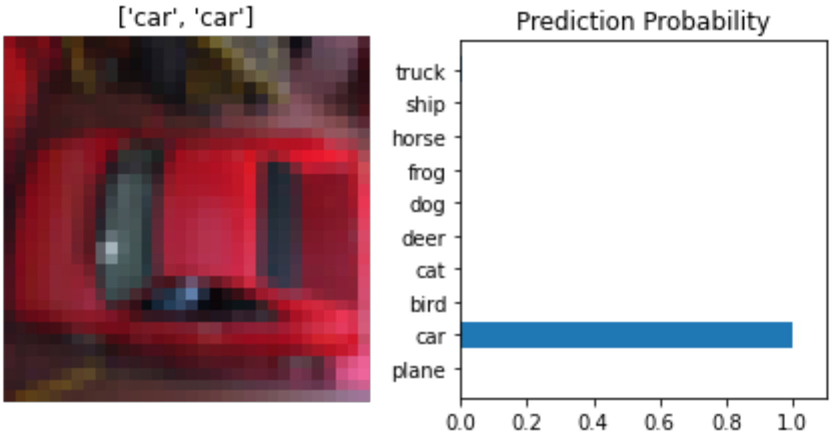
Show some predictions of the test network

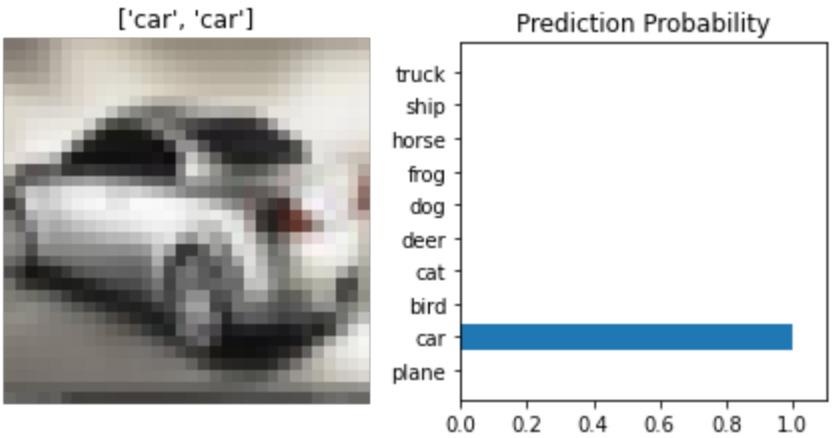
In [15]:

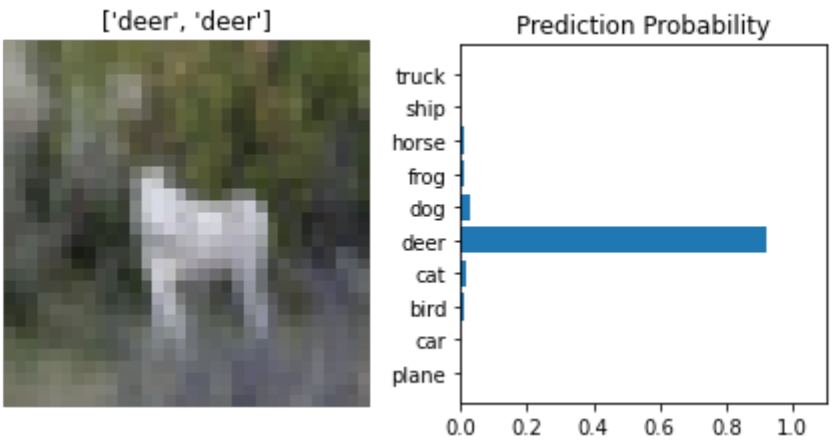
```
def visualize_pred(img, pred_prob, real_label):
    ''' Function for viewing an image and it's predicted classes.
    '''
    #pred_prob = pred_prob.data.numpy().squeeze()

    fig, (ax1, ax2) = plt.subplots(figsize=(6,9), ncols=2)
    #ax1.imshow(img.numpy().squeeze())
    imshow(img, ax1)
    ax1.axis('off')
    pred_label = numpy.argmax(pred_prob)
    ax1.set_title([classNames[real_label], classNames[pred_label]])

    ax2.barh(numpy.arange(10), pred_prob)
    ax2.set_aspect(0.1)
    ax2.set_yticks(numpy.arange(10))
    ax2.set_yticklabels(classNames)
    ax2.set_title('Prediction Probability')
    ax2.set_xlim(0, 1.1)
    plt.tight_layout()
```

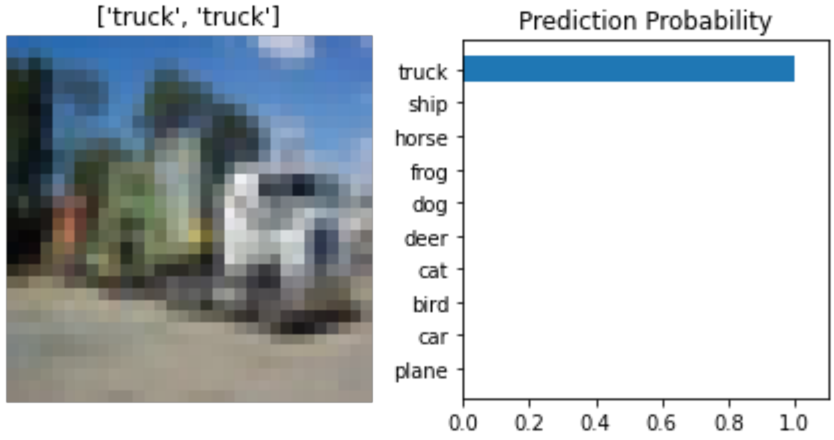
In [16]:

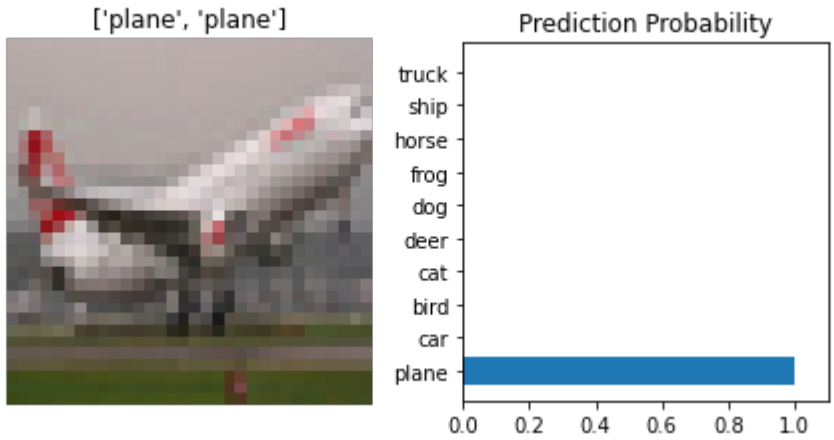
```
model_cnn.to('cpu')
Number of parameters: 186010

# fetch a batch of test images
image_batch, label_batch = next(iter(test_loader))

# Turn off gradients to speed up this part
with torch.no_grad():
    log_pred_prob_batch = model_cnn(image_batch)
for i in range(10):
    img = image_batch[i]
    real_label = label_batch[i].item()
    log_pred_prob = log_pred_prob_batch[i]
    # Output of the network are log-probabilities, need to take expone
ntial for probabilities
    pred_prob = torch.exp(log_pred_prob).data.numpy().squeeze()
    visualize_pred(img, pred_prob, real_label)
```

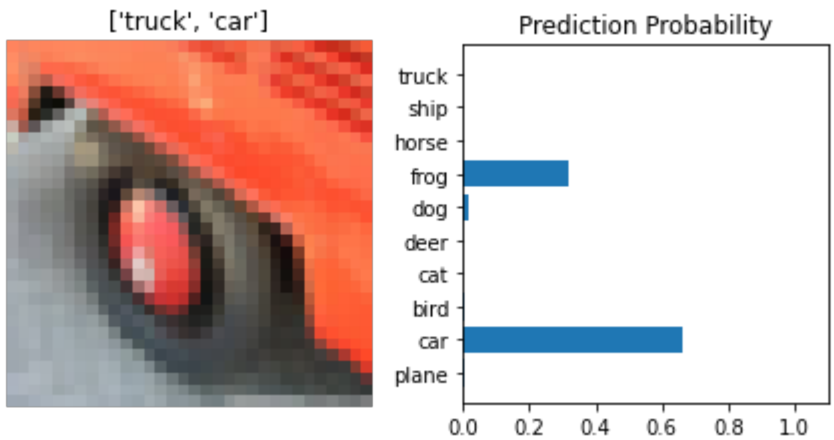

['car', 'car']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0
0.2
0.4
0.6
0.8
1.0

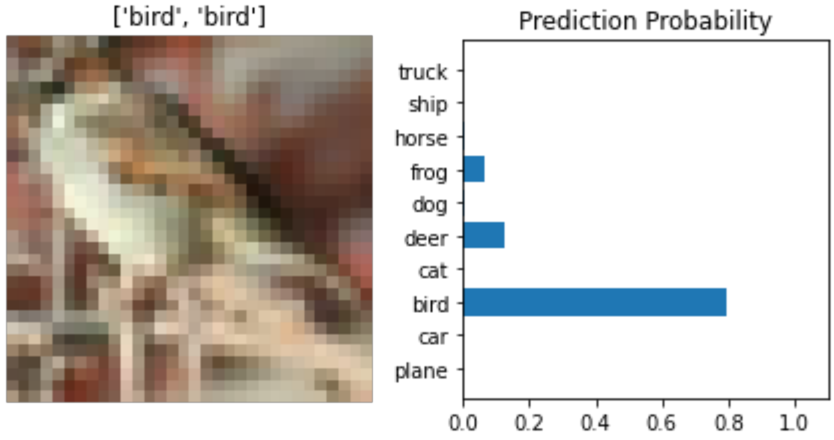

['car', 'car']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0
0.2
0.4
0.6
0.8
1.0

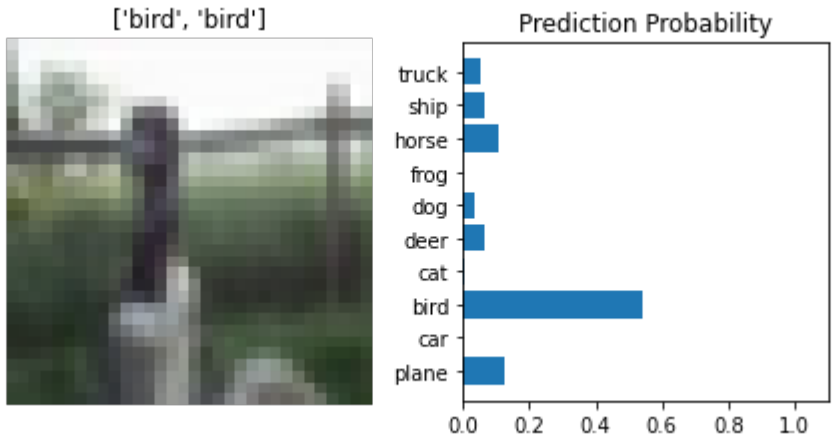

['deer', 'deer']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0
0.2
0.4
0.6
0.8
1.0

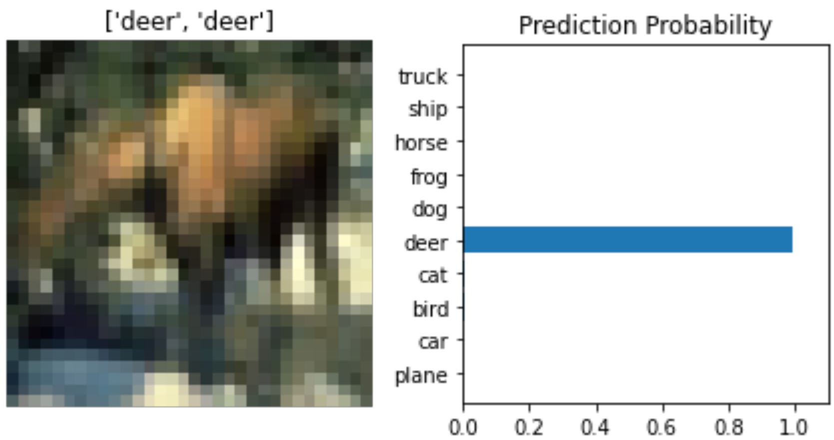

['truck', 'truck']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0
0.2
0.4
0.6
0.8
1.0

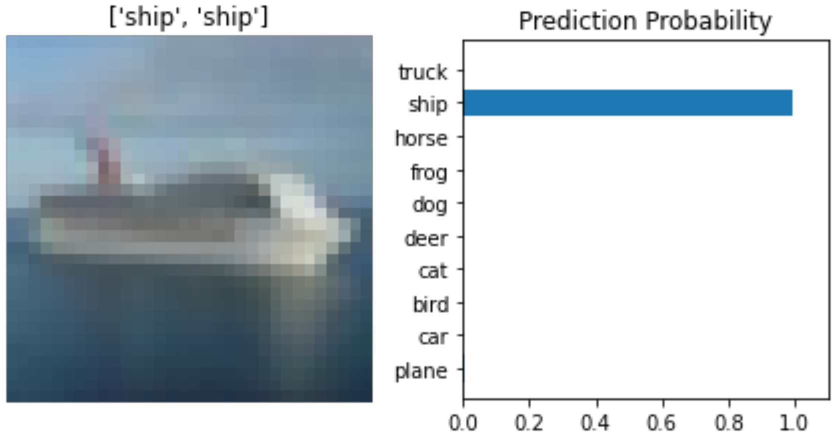

['plane', 'plane']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0
0.2
0.4
0.6
0.8
1.0


['truck', 'car']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0
0.2
0.4
0.6
0.8
1.0


['bird', 'bird']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0
0.2
0.4
0.6
0.8
1.0


['bird', 'bird']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0
0.2
0.4
0.6
0.8
1.0

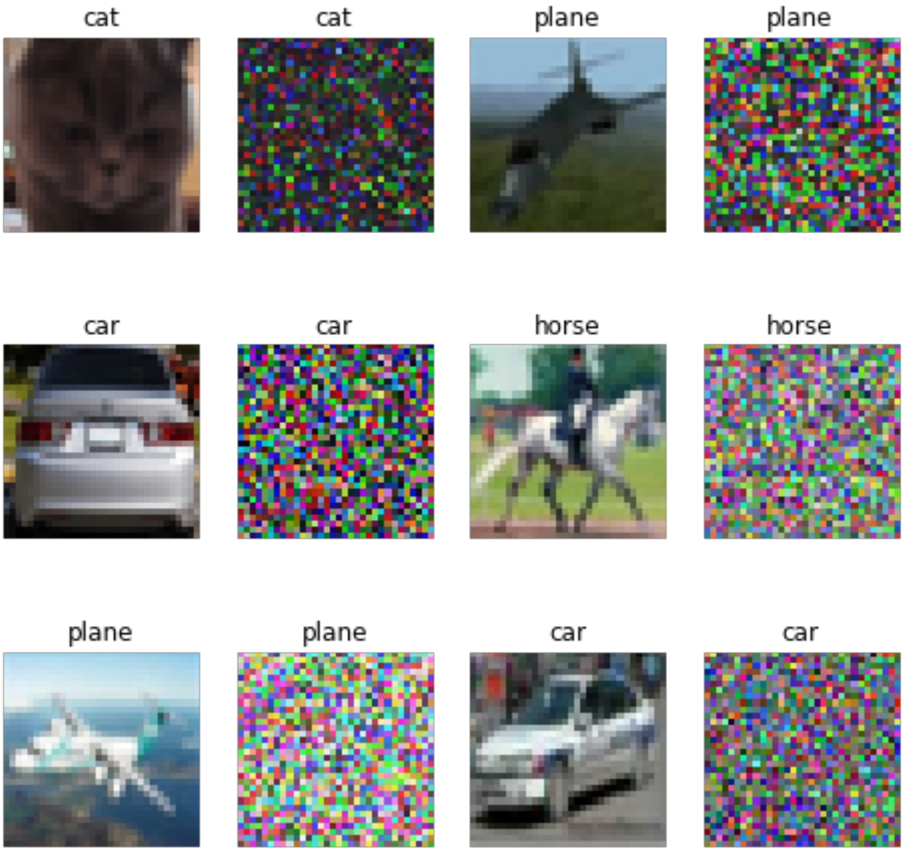

['deer', 'deer']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0
0.2
0.4
0.6
0.8
1.0



Does the Convolutional Network use "Visual Information" ?

In [17]:

```
fixed_perm = torch.randperm(3072) # Fix a permutation of the image pi
xels; We apply the same permutation to all images

# show some training images
plt.figure(figsize=(8, 8))

# fetch a batch of train images; RANDOM
image_batch, label_batch = next(iter(train_loader))

for i in range(6):
    image = image_batch[i]
    image_perm = image.view(-1, 32*32*3).clone()
    image_perm = image_perm[:, fixed_perm]
    image_perm = image_perm.view(3, 32, 32)

    label = label_batch[i].item()
    plt.subplot(3,4 , 2*i + 1)
    #image, label = train_loader.dataset.__getitem__(i)
    #plt.imshow(image.squeeze().numpy())
    imshow(image, plt)
    plt.axis('off')
    plt.title(classNames[label])
    plt.subplot(3, 4, 2*i+2)
    #plt.imshow(image_perm.squeeze().numpy())
    imshow(image_perm, plt)
    plt.axis('off')
    plt.title(classNames[label])
```


cat
cat
plane
plane
car
car
horse
horse
plane
plane
car
car

In [18]:

```
accuracy_list = []

def scramble_train(epoch, model, perm=torch.arange(0, 3072).long()):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        # send to device
        data, target = data.to(device), target.to(device)

        # permute pixels
        data = data.view(-1, 32*32*3)
        data = data[:, perm]
        data = data.view(-1, 3, 32, 32)

        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.f
ormat(
                epoch, batch_idx * len(data), len(train_loader.datase
t),
                100. * batch_idx / len(train_loader), loss.item()))

def scramble_test(model, perm=torch.arange(0, 3072).long()):
    model.eval()
    test_loss = 0
    correct = 0
    for data, target in test_loader:
        # send to device
        data, target = data.to(device), target.to(device)

        # permute pixels
        data = data.view(-1, 32*32*3)
        data = data[:, perm]
        data = data.view(-1, 3, 32, 32)

        output = model(data)
        test_loss += F.nll_loss(output, target, reduction='sum').item
() # sum up batch loss
        pred = output.data.max(1, keepdim=True)[1] # get the index of
```

```
the max log-probability
        correct += pred.eq(target.data.view_as(pred)).cpu().sum().ite
m()

    test_loss /= len(test_loader.dataset)
    accuracy = 100. * correct / len(test_loader.dataset)
    accuracy_list.append(accuracy)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0
f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        accuracy))
```

In [19]:

```
print("Training on ", device)
model_cnn_2 = CNN(input_size, output_size)
model_cnn_2.to(device)
optimizer = optim.SGD(model_cnn_2.parameters(), lr=0.01, momentum=
0.5)
print('Number of parameters: {}'.format(get_n_params(model_cnn_2)))

for epoch in range(0, 10):
    scramble_train(epoch, model_cnn_2, fixed_perm)
    scramble_test(model_cnn_2, fixed_perm)
```

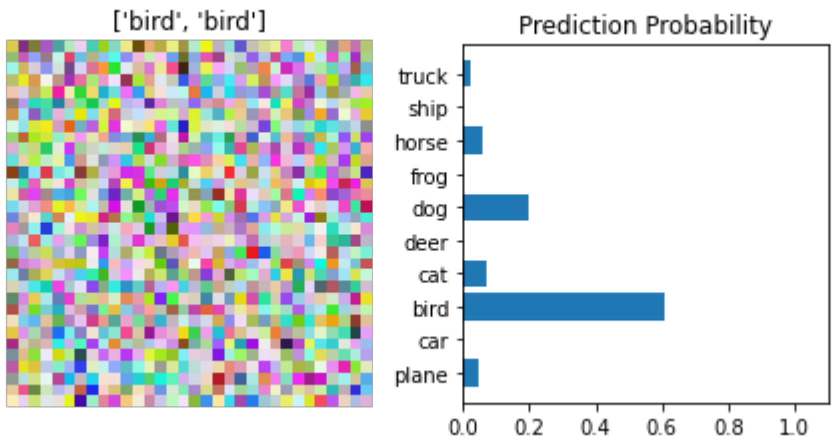
```
Training on  cuda:0
Number of parameters: 1558026
Train Epoch: 0 [0/50000 (0%)]   Loss: 3.028028
Train Epoch: 0 [6400/50000 (13%)]       Loss: 2.300369
Train Epoch: 0 [12800/50000 (26%)]      Loss: 2.244884
Train Epoch: 0 [19200/50000 (38%)]      Loss: 1.888256
Train Epoch: 0 [25600/50000 (51%)]      Loss: 2.022595
Train Epoch: 0 [32000/50000 (64%)]      Loss: 1.813951
Train Epoch: 0 [38400/50000 (77%)]      Loss: 1.716965
Train Epoch: 0 [44800/50000 (90%)]      Loss: 1.678029

Test set: Average loss: 2.0855, Accuracy: 2921/10000 (29%)

Train Epoch: 1 [0/50000 (0%)]   Loss: 2.023972
Train Epoch: 1 [6400/50000 (13%)]       Loss: 1.653855
Train Epoch: 1 [12800/50000 (26%)]      Loss: 1.713514
Train Epoch: 1 [19200/50000 (38%)]      Loss: 1.412678
Train Epoch: 1 [25600/50000 (51%)]      Loss: 1.551034
Train Epoch: 1 [32000/50000 (64%)]      Loss: 1.313132
Train Epoch: 1 [38400/50000 (77%)]      Loss: 1.473628
Train Epoch: 1 [44800/50000 (90%)]      Loss: 1.876570

Test set: Average loss: 1.6481, Accuracy: 4199/10000 (42%)

Train Epoch: 2 [0/50000 (0%)]   Loss: 1.699377
Train Epoch: 2 [6400/50000 (13%)]       Loss: 1.390427
Train Epoch: 2 [12800/50000 (26%)]      Loss: 1.271938
Train Epoch: 2 [19200/50000 (38%)]      Loss: 1.369706
Train Epoch: 2 [25600/50000 (51%)]      Loss: 1.371320
Train Epoch: 2 [32000/50000 (64%)]      Loss: 1.632250
Train Epoch: 2 [38400/50000 (77%)]      Loss: 1.467190
Train Epoch: 2 [44800/50000 (90%)]      Loss: 1.493528

Test set: Average loss: 1.6855, Accuracy: 4030/10000 (40%)

Train Epoch: 3 [0/50000 (0%)]   Loss: 1.276010
Train Epoch: 3 [6400/50000 (13%)]       Loss: 1.168302
Train Epoch: 3 [12800/50000 (26%)]      Loss: 1.239101
Train Epoch: 3 [19200/50000 (38%)]      Loss: 1.354253
Train Epoch: 3 [25600/50000 (51%)]      Loss: 1.107637
Train Epoch: 3 [32000/50000 (64%)]      Loss: 1.481267
Train Epoch: 3 [38400/50000 (77%)]      Loss: 1.196063
```

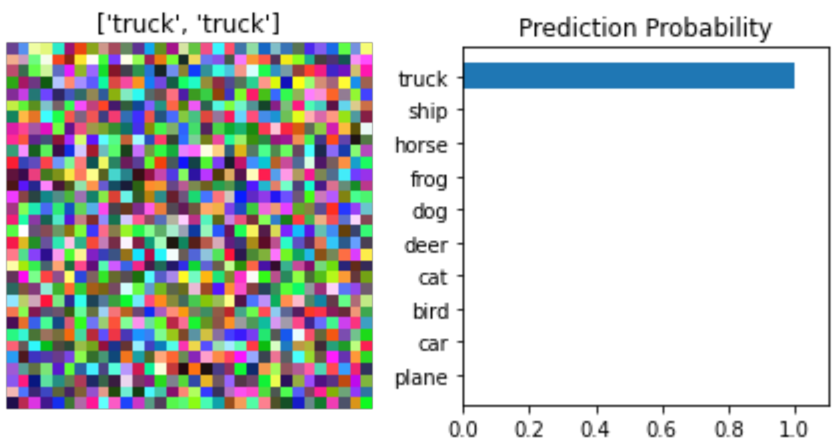
```
Train Epoch: 3 [44800/50000 (90%)]	Loss: 1.182608

Test set: Average loss: 1.7236, Accuracy: 4260/10000 (43%)

Train Epoch: 4 [0/50000 (0%)]	Loss: 1.230324
Train Epoch: 4 [6400/50000 (13%)]	Loss: 1.108968
Train Epoch: 4 [12800/50000 (26%)]	Loss: 1.294881
Train Epoch: 4 [19200/50000 (38%)]	Loss: 1.166198
Train Epoch: 4 [25600/50000 (51%)]	Loss: 0.866034
Train Epoch: 4 [32000/50000 (64%)]	Loss: 0.948421
Train Epoch: 4 [38400/50000 (77%)]	Loss: 1.300652
Train Epoch: 4 [44800/50000 (90%)]	Loss: 1.149651

Test set: Average loss: 1.9053, Accuracy: 4099/10000 (41%)

Train Epoch: 5 [0/50000 (0%)]	Loss: 1.078488
Train Epoch: 5 [6400/50000 (13%)]	Loss: 0.915758
Train Epoch: 5 [12800/50000 (26%)]	Loss: 0.866374
Train Epoch: 5 [19200/50000 (38%)]	Loss: 0.861203
Train Epoch: 5 [25600/50000 (51%)]	Loss: 0.963555
Train Epoch: 5 [32000/50000 (64%)]	Loss: 1.037355
Train Epoch: 5 [38400/50000 (77%)]	Loss: 0.898149
Train Epoch: 5 [44800/50000 (90%)]	Loss: 0.943985

Test set: Average loss: 1.9834, Accuracy: 4083/10000 (41%)

Train Epoch: 6 [0/50000 (0%)]	Loss: 0.959099
Train Epoch: 6 [6400/50000 (13%)]	Loss: 0.790870
Train Epoch: 6 [12800/50000 (26%)]	Loss: 0.697684
Train Epoch: 6 [19200/50000 (38%)]	Loss: 0.600765
Train Epoch: 6 [25600/50000 (51%)]	Loss: 0.703402
Train Epoch: 6 [32000/50000 (64%)]	Loss: 0.682824
Train Epoch: 6 [38400/50000 (77%)]	Loss: 0.679988
Train Epoch: 6 [44800/50000 (90%)]	Loss: 0.709877

Test set: Average loss: 1.9719, Accuracy: 4414/10000 (44%)

Train Epoch: 7 [0/50000 (0%)]	Loss: 0.557418
Train Epoch: 7 [6400/50000 (13%)]	Loss: 0.388350
Train Epoch: 7 [12800/50000 (26%)]	Loss: 0.398211
Train Epoch: 7 [19200/50000 (38%)]	Loss: 0.418651
Train Epoch: 7 [25600/50000 (51%)]	Loss: 0.518414
Train Epoch: 7 [32000/50000 (64%)]	Loss: 0.558018
```

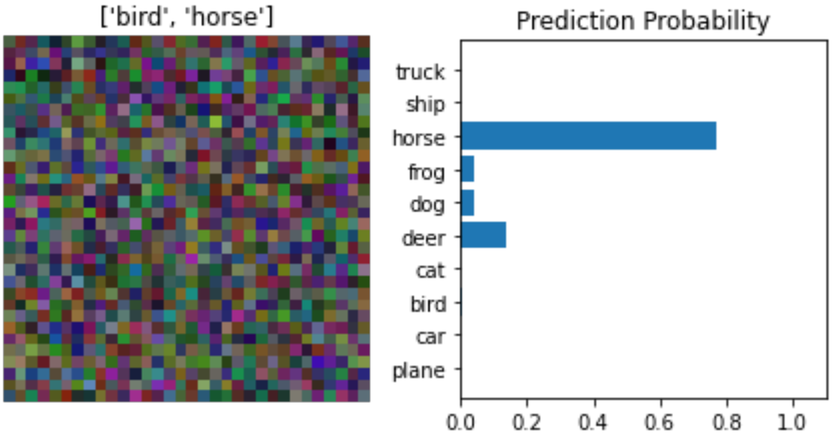
```
Train Epoch: 7 [38400/50000 (77%)]      Loss: 0.663281
Train Epoch: 7 [44800/50000 (90%)]      Loss: 0.844735

Test set: Average loss: 2.4950, Accuracy: 3988/10000 (40%)

Train Epoch: 8 [0/50000 (0%)]   Loss: 0.619049
Train Epoch: 8 [6400/50000 (13%)]       Loss: 0.309629
Train Epoch: 8 [12800/50000 (26%)]      Loss: 0.222102
Train Epoch: 8 [19200/50000 (38%)]      Loss: 0.489032
Train Epoch: 8 [25600/50000 (51%)]      Loss: 0.276393
Train Epoch: 8 [32000/50000 (64%)]      Loss: 0.350856
Train Epoch: 8 [38400/50000 (77%)]      Loss: 0.486811
Train Epoch: 8 [44800/50000 (90%)]      Loss: 0.275683

Test set: Average loss: 3.4336, Accuracy: 3515/10000 (35%)

Train Epoch: 9 [0/50000 (0%)]   Loss: 1.023970
Train Epoch: 9 [6400/50000 (13%)]       Loss: 0.094431
Train Epoch: 9 [12800/50000 (26%)]      Loss: 0.184453
Train Epoch: 9 [19200/50000 (38%)]      Loss: 0.139975
Train Epoch: 9 [25600/50000 (51%)]      Loss: 0.226777
Train Epoch: 9 [32000/50000 (64%)]      Loss: 0.153643
Train Epoch: 9 [38400/50000 (77%)]      Loss: 0.145197
Train Epoch: 9 [44800/50000 (90%)]      Loss: 0.203180

Test set: Average loss: 2.6585, Accuracy: 4442/10000 (44%)
```

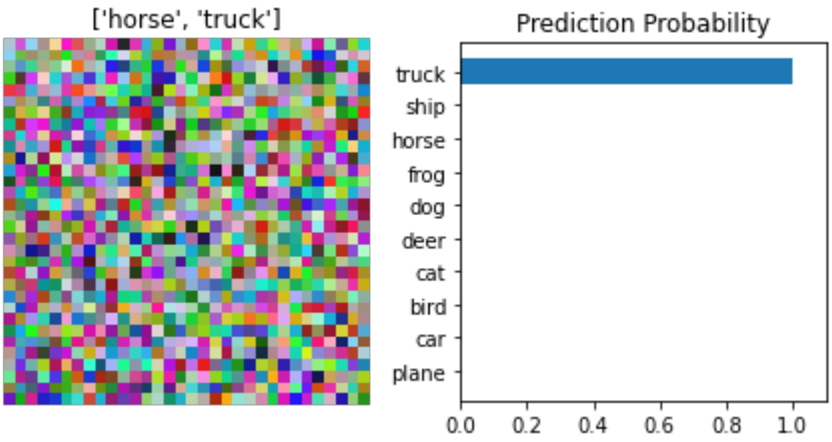
In [20]:

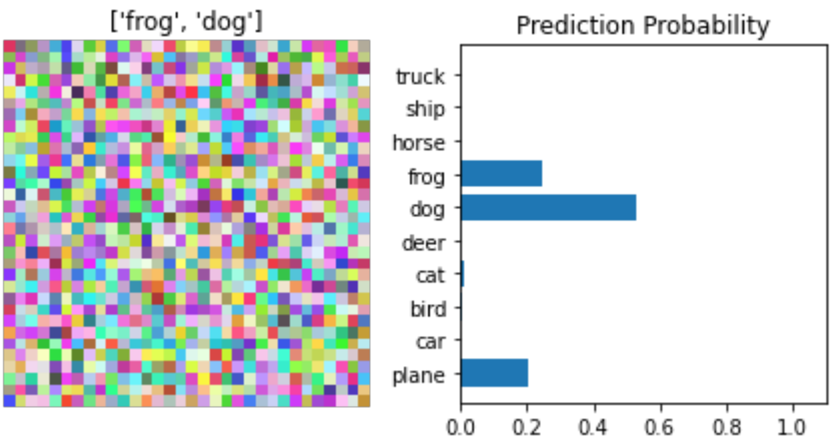
```
model_cnn_2.to('cpu')

# fetch a batch of test images
image_batch, label_batch = next(iter(test_loader))
image_batch_scramble = image_batch.view(-1, 32*32*3)
image_batch_scramble = image_batch_scramble[:, fixed_perm]
image_batch_scramble = image_batch_scramble.view(-1, 3, 32, 32)
# Turn off gradients to speed up this part
with torch.no_grad():
    log_pred_prob_batch = model_cnn_2(image_batch_scramble)
for i in range(10):
    img = image_batch[i]
    img_perm = image_batch_scramble[i]
    real_label = label_batch[i].item()
    log_pred_prob = log_pred_prob_batch[i]
    # Output of the network are log-probabilities, need to take expone
ntial for probabilities
    pred_prob = torch.exp(log_pred_prob).data.numpy().squeeze()
    visualize_pred(img_perm, pred_prob, real_label)
```

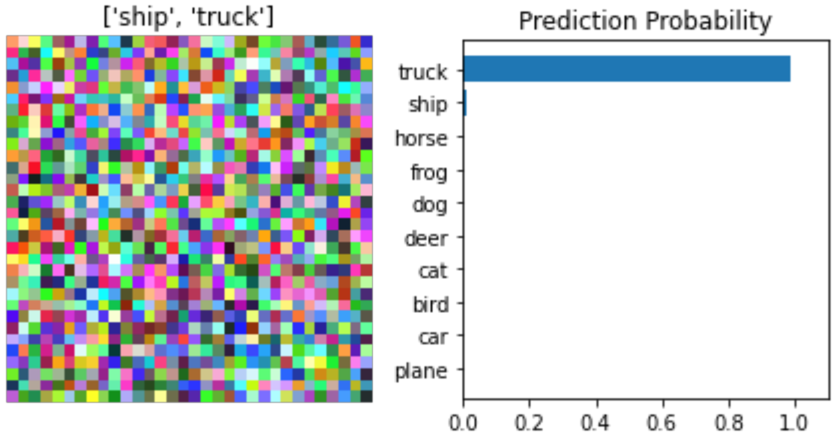


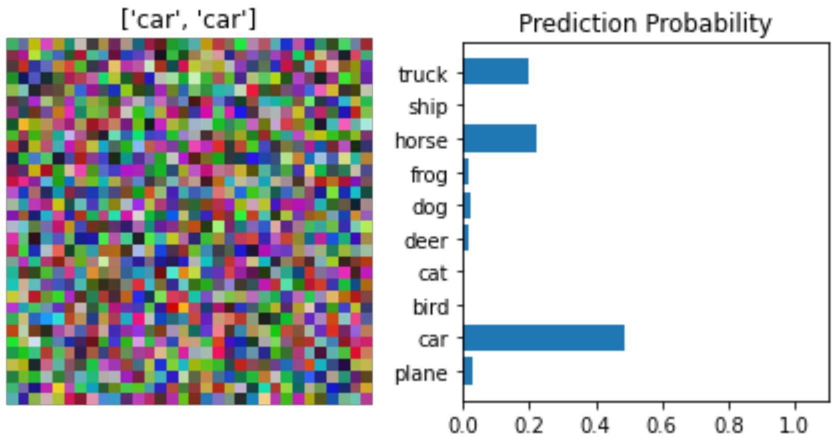


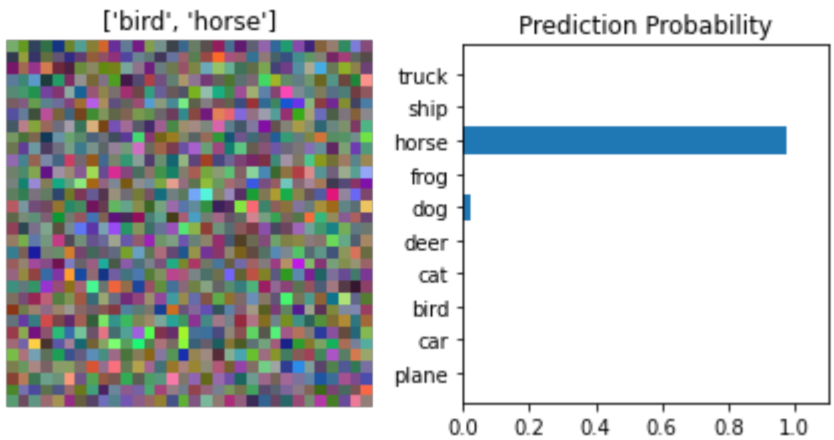

['bird', 'horse']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0 0.2 0.4 0.6 0.8 1.0

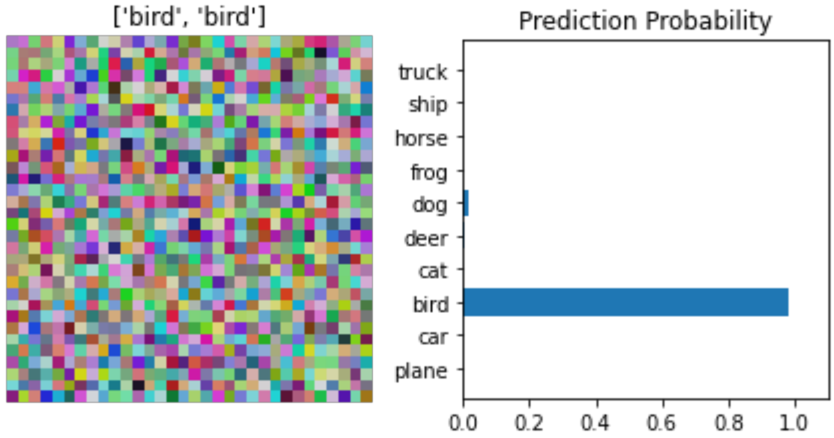

['horse', 'truck']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0 0.2 0.4 0.6 0.8 1.0

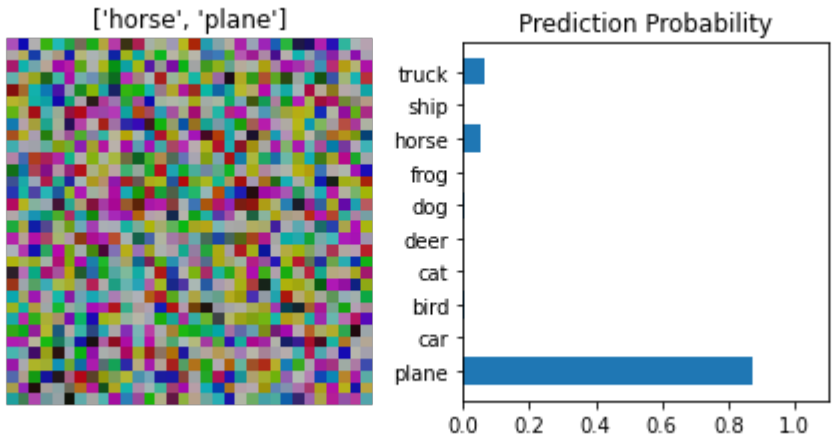

['frog', 'dog']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0 0.2 0.4 0.6 0.8 1.0


['ship', 'truck']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0
0.2
0.4
0.6
0.8
1.0


['car', 'car']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0
0.2
0.4
0.6
0.8
1.0


['bird', 'horse']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0
0.2
0.4
0.6
0.8
1.0


['bird', 'bird']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0
0.2
0.4
0.6
0.8
1.0


['horse', 'plane']
Prediction Probability
truck
ship
horse
frog
dog
deer
cat
bird
car
plane
0.0
0.2
0.4
0.6
0.8
1.0