Assignment3 Part1

In [4]:

```
import torch
import torchvision
import torchvision.transforms as transforms
```

In [5]:

```
#Checking CUDA Availability
train_on_gpu = torch.cuda.is_available()

if not train_on_gpu:
    print('CUDA is not available!  Training on CPU...')
else:
    print('CUDA is available!  Training on GPU ...')
```

```
CUDA is available!  Training on GPU ...
```

In [6]:

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

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
```

In [7]:

```
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()

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

```
<Figure size 640x480 with 1 Axes>
```

```
truck  deer  bird truck
```

In [8]:

```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
print(net)
```

```
Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

In [5]:

```python
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

In [6]:

```python
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

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

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0
```

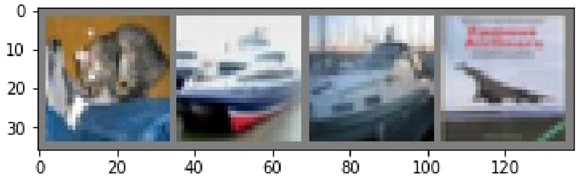
```
print('Finished Training')
```

```
[1,  2000] loss: 2.217
[1,  4000] loss: 1.861
[1,  6000] loss: 1.682
[1,  8000] loss: 1.586
[1, 10000] loss: 1.502
[1, 12000] loss: 1.450
[2,  2000] loss: 1.394
[2,  4000] loss: 1.352
[2,  6000] loss: 1.339
[2,  8000] loss: 1.326
[2, 10000] loss: 1.294
[2, 12000] loss: 1.286
Finished Training
```

Test the Network

In [8]:

```
dataiter = iter(testloader)
images, labels = dataiter.next()

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




```
GroundTruth:    cat  ship  ship plane
```

In [9]:

```
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```

```
Predicted:  plane plane plane   dog
```

In [15]:

```
correct = 0
total = 0
confusion_matrix = np.zeros([10,10], int)
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()
        for i, l in enumerate(labels):
            confusion_matrix[l.item(), predicted[i].item()] += 1


print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 52 %
```

In [11]:

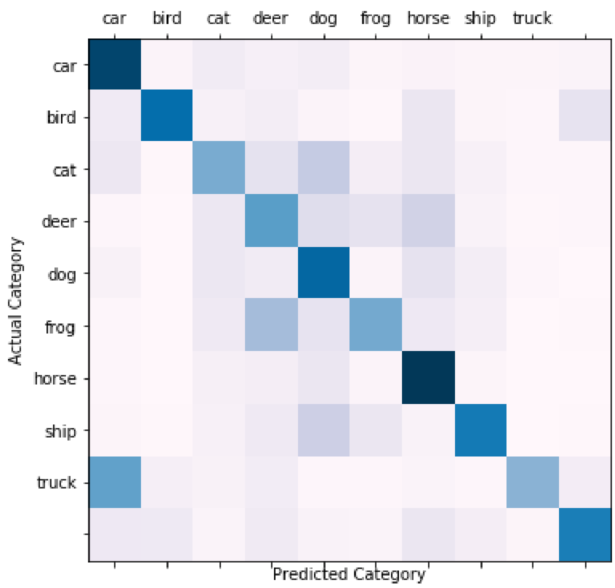
```python
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of plane : 71 %
Accuracy of   car : 57 %
Accuracy of  bird : 37 %
Accuracy of   cat : 41 %
Accuracy of  deer : 60 %
Accuracy of   dog : 37 %
Accuracy of  frog : 75 %
Accuracy of horse : 53 %
Accuracy of  ship : 33 %
Accuracy of truck : 52 %
```

Confusion Matrix

In [41]:

```python
import matplotlib.ticker as ticker

fig, ax = plt.subplots(1,1,figsize=(8,6))
cax=ax.matshow(confusion_matrix,cmap=plt.get_cmap('PuBu'))
#fig.colorbar(cax)
plt.ylabel('Actual Category')
plt.yticks(range(10), classes)
plt.xlabel('Predicted Category')
plt.xticks(range(10), classes)
ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
plt.show()
```



In [29]:

```python
print('actual/pred'.ljust(16), end='')
for i,c in enumerate(classes):
    print(c.ljust(10), end='')
```

```python
    print(c.ljust(10), end='')
print()
for i,r in enumerate(confusion_matrix):
    print(classes[i].ljust(16), end='')
    for idx, p in enumerate(r):
        print(str(p).ljust(10), end='')
    print()

    r = r/np.sum(r)
    print(''.ljust(16), end='')
    for idx, p in enumerate(r):
        print(str(p).ljust(10), end='')
    print()
```

```
actual/pred     plane     car       bird      cat       deer      dog       frog      horse     shi
truck
plane           717       19        69        43        58        12        26        17        17
22
                0.717     0.019     0.069     0.043     0.058     0.012     0.026     0.017     0.0
0.022
car             74        576       39        58        19        4         95        16        8
111
                0.074     0.576     0.039     0.058     0.019     0.004     0.095     0.016     0.0
0.111
bird            89        3         370       116       215       54        100       37        8
8
                0.089     0.003     0.37      0.116     0.215     0.054     0.1       0.037     0.0
0.008
cat             9         2         92        419       138       114       185       27        3
11
                0.009     0.002     0.092     0.419     0.138     0.114     0.185     0.027     0.0
0.011
deer            33        2         94        66        605       19        113       58        9
1
                0.033     0.002     0.094     0.066     0.605     0.019     0.113     0.058     0.0
0.001
dog             7         2         78        285       112       378       85        48        1
4
                0.007     0.002     0.078     0.285     0.112     0.378     0.085     0.048     0.0
0.004
frog            9         1         46        56        98        18        758       12        0
2
                0.009     0.001     0.046     0.056     0.098     0.018     0.758     0.012     0.0
0.002
horse           14        4         37        78        198       99        29        537       1
3
                0.014     0.004     0.037     0.078     0.198     0.099     0.029     0.537     0.0
0.003
ship            406       48        33        64        11        10        23        9         336
60
                0.406     0.048     0.033     0.064     0.011     0.01      0.023     0.009     0.3
0.06
truck           83        82        22        75        27        22        96        54        14
525
                0.083     0.082     0.022     0.075     0.027     0.022     0.096     0.054     0.0
0.525
```

Assignment3 Part2

In [1]:

```
import torch
import torchvision
import torchvision.transforms as transforms
```

In [2]:

```
#Checking CUDA Availability
train_on_gpu = torch.cuda.is_available()

if not train_on_gpu:
    print('CUDA is not available!  Training on CPU...')
else:
    print('CUDA is available!  Training on GPU ...')
```

```
CUDA is available!  Training on GPU ...
```

In [3]:

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

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
```

In [4]:

```
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()

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

```
<Figure size 640x480 with 1 Axes>
```

```
plane plane  deer truck
```

Selecting mask size of 3x3

In [8]:

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        #3 Convolutional layers
        self.conv1 = nn.Conv2d(3, 6, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 3)
        self.fc1 = nn.Linear(16 * 6 * 6, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        #Add dropout if data is overfitting
        #self.dropout=nn.Dropout(.15)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        #x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 16 * 6 * 6)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x


net = Net()
print(net)
#if train_on_gpu:
#    net.cuda()
```

```
Net(
  (conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=576, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

In [9]:

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

In [10]:

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    net.train()
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
```

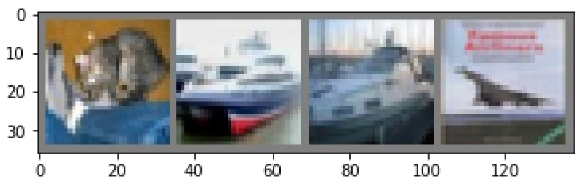
```
        loss.backward()
        optimizer.step()

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

print('Finished Training')
```

```
[1,  2000] loss: 2.253
[1,  4000] loss: 1.806
[1,  6000] loss: 1.612
[1,  8000] loss: 1.497
[1, 10000] loss: 1.451
[1, 12000] loss: 1.408
[2,  2000] loss: 1.326
[2,  4000] loss: 1.287
[2,  6000] loss: 1.282
[2,  8000] loss: 1.230
[2, 10000] loss: 1.226
[2, 12000] loss: 1.231
Finished Training
```

Test the Network

In [11]:

```
dataiter = iter(testloader)
images, labels = dataiter.next()

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




```
GroundTruth:    cat  ship  ship plane
```

In [12]:

```
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```

```
Predicted:   ship  deer  ship  deer
```

In [13]:

```
correct = 0
total = 0
confusion_matrix = np.zeros([10,10], int)
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()
        for i, l in enumerate(labels):
            confusion_matrix[l.item(), predicted[i].item()] += 1
```

```python
print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 57 %
```

In [1]:

```python
##Increase in accuracy by 5% by changing the mask size
```

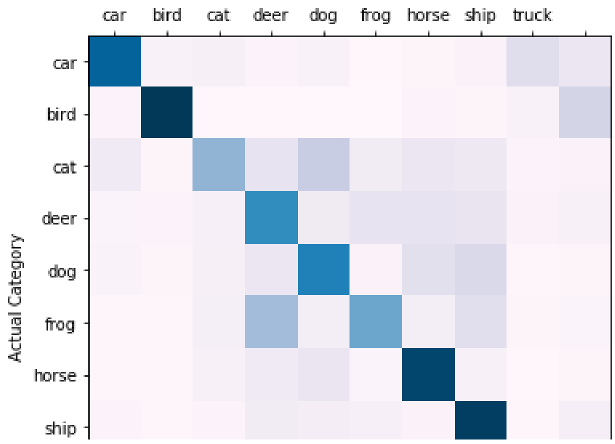
In [14]:

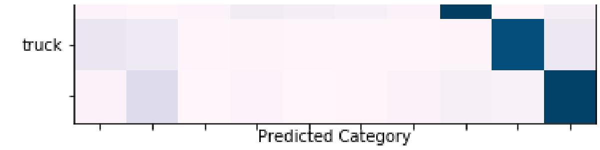
```python
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of plane : 60 %
Accuracy of   car : 73 %
Accuracy of  bird : 31 %
Accuracy of   cat : 46 %
Accuracy of  deer : 49 %
Accuracy of   dog : 37 %
Accuracy of  frog : 69 %
Accuracy of horse : 71 %
Accuracy of  ship : 66 %
Accuracy of truck : 70 %
```

In [15]:

```python
import matplotlib.ticker as ticker

fig, ax = plt.subplots(1,1,figsize=(8,6))
cax=ax.matshow(confusion_matrix,cmap=plt.get_cmap('PuBu'))
#fig.colorbar(cax)
plt.ylabel('Actual Category')
plt.yticks(range(10), classes)
plt.xlabel('Predicted Category')
plt.xticks(range(10), classes)
ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
plt.show()
```





In [16]:

```python
print('actual/pred'.ljust(16), end='')
for i,c in enumerate(classes):
    print(c.ljust(10), end='')
print()
for i,r in enumerate(confusion_matrix):
    print(classes[i].ljust(16), end='')
    for idx, p in enumerate(r):
        print(str(p).ljust(10), end='')
    print()

    r = r/np.sum(r)
    print(''.ljust(16), end='')
    for idx, p in enumerate(r):
        print(str(p).ljust(10), end='')
    print()
```

```
actual/pred     plane     car       bird      cat       deer      dog       frog      horse     shi
truck
plane           600       32        46        19        32        4         18        26        130
93
                0.6       0.032     0.046     0.019     0.032     0.004     0.018     0.026     0.1
0.093
car             20        730       5         6         1         2         19        16        31
170
                0.02      0.73      0.005     0.006     0.001     0.002     0.019     0.016     0.0
0.17
bird            72        15        312       109       200       64        97        86        19
26
                0.072     0.015     0.312     0.109     0.2       0.064     0.097     0.086     0.0
0.026
cat             21        19        41        465       74        108       109       102       24
37
                0.021     0.019     0.041     0.465     0.074     0.108     0.109     0.102     0.0
0.037
deer            28        14        44        96        498       26        121       151       9
13
                0.028     0.014     0.044     0.096     0.498     0.026     0.121     0.151     0.0
0.013
dog             12        7         51        278       57        375       56        127       16
21
                0.012     0.007     0.051     0.278     0.057     0.375     0.056     0.127     0.0
0.021
frog            7         12        36        71        99        21        692       39        6
17
                0.007     0.012     0.036     0.071     0.099     0.021     0.692     0.039     0.0
0.017
horse           19        8         19        64        55        43        26        712       7
47
                0.019     0.008     0.019     0.064     0.055     0.043     0.026     0.712     0.0
0.047
ship            99        77        10        13        9         9         10        17        669
87
                0.099     0.077     0.01      0.013     0.009     0.009     0.01      0.017     0.6
0.087
truck           24        136       6         19        4         7         25        46        32
701
                0.024     0.136     0.006     0.019     0.004     0.007     0.025     0.046     0.0
0.701
```

Assignment3 Question3

In [1]:

```
import torch
import torchvision
import torchvision.transforms as transforms
```

In [2]:

```
#Checking CUDA Availability
train_on_gpu = torch.cuda.is_available()

if not train_on_gpu:
    print('CUDA is not available!  Training on CPU...')
else:
    print('CUDA is available!  Training on GPU ...')
```

```
CUDA is available!  Training on GPU ...
```

In [3]:

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

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
```

In [4]:

```
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()

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

```
<Figure size 640x480 with 1 Axes>
```

deer bird cat ship

Increasing no. of filters in conv1

In [11]:

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        #3 Convolutional layers changing no of filters to 12 from 6 for conv1
        self.conv1 = nn.Conv2d(3, 12, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(12, 16, 3)
        self.fc1 = nn.Linear(16 * 6 * 6, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        #Add dropout if data is overfitting
        #self.dropout=nn.Dropout(.15)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        #x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 16 * 6 * 6)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x


net = Net()
print(net)
#if train_on_gpu:
#    net.cuda()
```

```
Net(
  (conv1): Conv2d(3, 12, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(12, 16, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=576, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

In [12]:

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

In [13]:

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    net.train()
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
```

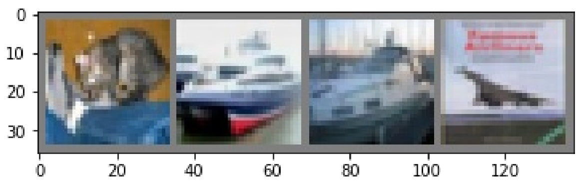
```
        loss.backward()
        optimizer.step()

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

print('Finished Training')
```

```
[1,  2000] loss: 2.225
[1,  4000] loss: 1.751
[1,  6000] loss: 1.603
[1,  8000] loss: 1.516
[1, 10000] loss: 1.423
[1, 12000] loss: 1.400
[2,  2000] loss: 1.303
[2,  4000] loss: 1.282
[2,  6000] loss: 1.256
[2,  8000] loss: 1.216
[2, 10000] loss: 1.168
[2, 12000] loss: 1.166
Finished Training
```

Test the Network

In [14]:

```
dataiter = iter(testloader)
images, labels = dataiter.next()

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




```
GroundTruth:    cat  ship  ship plane
```

In [15]:

```
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```

```
Predicted:   ship  ship   dog   car
```

In [16]:

```
correct = 0
total = 0
confusion_matrix = np.zeros([10,10], int)
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()
        for i, l in enumerate(labels):
            confusion_matrix[l.item(), predicted[i].item()] += 1
```

```
print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 60 %
```

Increase in accuracy with increase in conv1 filters is seen by 3%

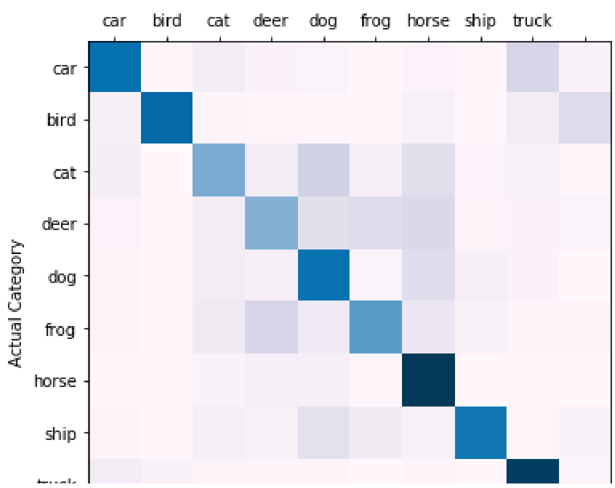
In [17]:

```
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of plane : 61 %
Accuracy of   car : 64 %
Accuracy of  bird : 39 %
Accuracy of   cat : 37 %
Accuracy of  deer : 60 %
Accuracy of   dog : 45 %
Accuracy of  frog : 82 %
Accuracy of horse : 59 %
Accuracy of  ship : 80 %
Accuracy of truck : 68 %
```

In [18]:

```
import matplotlib.ticker as ticker

fig, ax = plt.subplots(1,1,figsize=(8,6))
cax=ax.matshow(confusion_matrix,cmap=plt.get_cmap('PuBu'))
#fig.colorbar(cax)
plt.ylabel('Actual Category')
plt.yticks(range(10), classes)
plt.xlabel('Predicted Category')
plt.xticks(range(10), classes)
ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
plt.show()
```



In [19]:

```python
print('actual/pred'.ljust(16), end='')
for i,c in enumerate(classes):
    print(c.ljust(10), end='')
print()
for i,r in enumerate(confusion_matrix):
    print(classes[i].ljust(16), end='')
    for idx, p in enumerate(r):
        print(str(p).ljust(10), end='')
    print()

    r = r/np.sum(r)
    print(''.ljust(16), end='')
    for idx, p in enumerate(r):
        print(str(p).ljust(10), end='')
    print()
```

```
actual/pred     plane     car       bird      cat       deer      dog       frog      horse     shi
truck
plane           613       13        70        27        25        6         23        7         182
34
                0.613     0.013     0.07      0.027     0.025     0.006     0.023     0.007     0.1
0.034
car             51        645       20        8         8         4         42        4         66
152
                0.051     0.645     0.02      0.008     0.008     0.004     0.042     0.004     0.C
0.152
bird            62        4         399       70        202       54        141       21        34
13
                0.062     0.004     0.399     0.07      0.202     0.054     0.141     0.021     0.C
0.013
cat             22        5         68        374       137       151       169       20        29
25
                0.022     0.005     0.068     0.374     0.137     0.151     0.169     0.02      0.C
0.025
deer            16        3         67        50        607       25        150       52        27
3
                0.016     0.003     0.067     0.05      0.607     0.025     0.15      0.052     0.C
0.003
dog             15        1         84        181       81        459       105       37        17
20
                0.015     0.001     0.084     0.181     0.081     0.459     0.105     0.037     0.C
0.02
frog            5         3         33        54        47        11        823       7         10
7
                0.005     0.003     0.033     0.054     0.047     0.011     0.823     0.007     0.C
0.007
horse           17        3         56        44        130       75        41        591       11
32
                0.017     0.003     0.056     0.044     0.13      0.075     0.041     0.591     0.C
0.032
ship            66        36        16        19        15        3         17        3         801
24
                0.066     0.036     0.016     0.019     0.015     0.003     0.017     0.003     0.8
0.024
truck           56        82        14        21        13        6         34        14        71
689
                0.056     0.082     0.014     0.021     0.013     0.006     0.034     0.014     0.C
0.689
```

Assignment3 Question4 Part1 with test subset of 100 images

In [55]:

```
import torch
import torchvision
import torchvision.transforms as transforms
```

In [56]:

```
#Checking CUDA Availability
train_on_gpu = torch.cuda.is_available()

if not train_on_gpu:
    print('CUDA is not available!  Training on CPU...')
else:
    print('CUDA is available!  Training on GPU ...')
```

```
CUDA is available!  Training on GPU ...
```

In [57]:

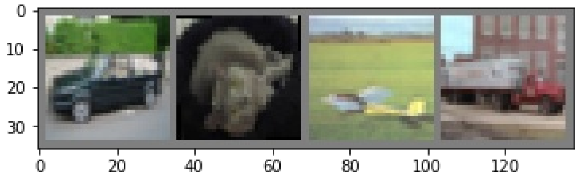
```
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(), # randomly flip
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

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
```

In [58]:

```
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()

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

```
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```




```
  car   cat plane truck
```

In [59]:

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        #3 Convolutional layers
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        #Max Pooling Layer
        self.pool = nn.MaxPool2d(2, 2)
        #Fully Connected Linear Layer
        self.fc1 = nn.Linear(64 * 4 * 4, 800)
        self.fc2 = nn.Linear(800, 400)
        self.fc3=nn.Linear(400,10)
        self.dropout = nn.Dropout(0.20)
        #Add dropout if data is overfitting
        #self.dropout=nn.Dropout(.15)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 64 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        return x


net = Net()
#if train_on_gpu:
#    net.cuda()
```

In [60]:

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

In [61]:

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    net.train()
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
```

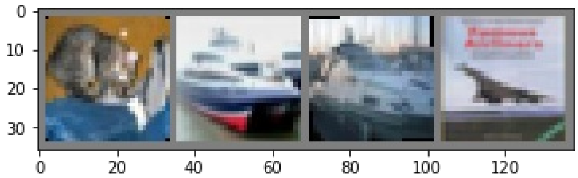
```
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

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

print('Finished Training')
```

```
[1,  2000] loss: 2.252
[1,  4000] loss: 1.963
[1,  6000] loss: 1.734
[1,  8000] loss: 1.602
[1, 10000] loss: 1.508
[1, 12000] loss: 1.451
[2,  2000] loss: 1.356
[2,  4000] loss: 1.328
[2,  6000] loss: 1.255
[2,  8000] loss: 1.249
[2, 10000] loss: 1.184
[2, 12000] loss: 1.180
Finished Training
```

Test the Network

In [62]:

```
dataiter = iter(testloader)
images, labels = dataiter.next()

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




```
GroundTruth:    cat  ship  ship plane
```

In [63]:

```
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```

```
Predicted:   frog   dog truck horse
```

In [64]:

```
from torch.utils.data.sampler import SubsetRandomSampler
num_test=len(testset)
indices = list(range(num_test))
np.random.shuffle(indices)
split = int(np.floor(.01 * num_test))
test_idx, subset_idx = indices[split:], indices[:split]

print(len(subset_idx), "Test subset created")

#test_sampler = SubsetRandomSampler(test_idx)
subset_sampler = SubsetRandomSampler(subset_idx)
```

```
#trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                        #    sampler=train_sampler, num_workers=2)

subsetloader = torch.utils.data.DataLoader(testset, batch_size=4,
    sampler=subset_sampler, num_workers=2)
```

```
100 Test subset created
```

In [65]:

```
correct = 0
total = 0

confusion_matrix = np.zeros([10,10], int)


with torch.no_grad():
    for data in subsetloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        labels=labels
        #print(len(labels))
        total += labels.size(0)
        correct += ((predicted == labels).sum().item())
        for i, l in enumerate(labels):
            confusion_matrix[l.item(), predicted[i].item()] += 1

#print(correct, total)
print('Accuracy of the network on the 100 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 100 test images: 60 %
```

In [66]:

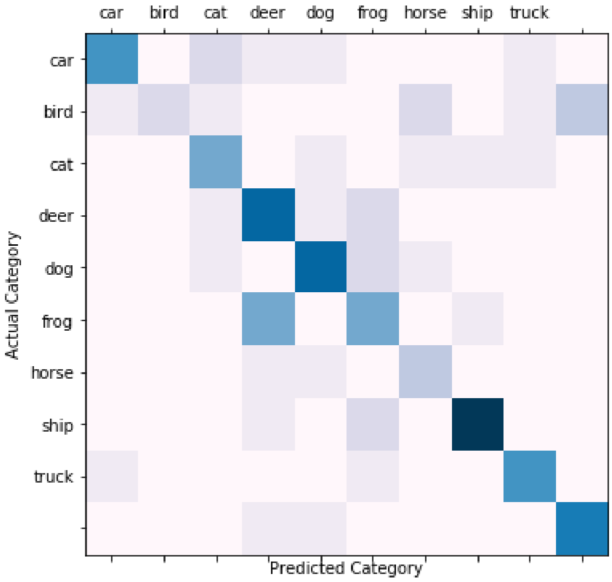
```
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in subsetloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of plane : 63 %
Accuracy of   car : 20 %
Accuracy of  bird : 33 %
Accuracy of   cat : 50 %
Accuracy of  deer : 58 %
Accuracy of   dog : 45 %
Accuracy of  frog : 80 %
Accuracy of horse : 53 %
Accuracy of  ship : 87 %
Accuracy of truck : 55 %
```

In [67]:

```
import matplotlib.ticker as ticker

fig, ax = plt.subplots(1,1,figsize=(8,6))
cax=ax.matshow(confusion_matrix,cmap=plt.get_cmap('PuBu'))
#fig.colorbar(cax)
```

```
plt.ylabel('Actual Category')
plt.yticks(range(10), classes)
plt.xlabel('Predicted Category')
plt.xticks(range(10), classes)
ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
plt.show()
```



Assignment3 Question4 Part2 with flip and additonal conv layer

In [37]:

```
import torch
import torchvision
import torchvision.transforms as transforms
```

In [38]:

```
#Checking CUDA Availability
train_on_gpu = torch.cuda.is_available()

if not train_on_gpu:
    print('CUDA is not available!  Training on CPU...')
else:
    print('CUDA is available!  Training on GPU ...')
```

```
CUDA is available!  Training on GPU ...
```

In [63]:

```
#Adding random flipping & rotation
```

In [39]:

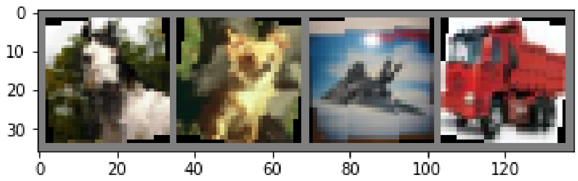
```
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(), # randomly flip
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

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
```

In [40]:

```
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()
```

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




```
horse   dog plane truck
```

In [ ]:

```
#Network achitecture is modified with increasing the depth and adding 3 convolutional layers and
#3 fully connected layers along with droput
```

In [54]:

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        #3 Convolutional layers
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        #Max Pooling Layer
        self.pool = nn.MaxPool2d(2, 2)
        #Fully Connected Linear Layer
        self.fc1 = nn.Linear(64 * 4 * 4, 800)
        self.fc2 = nn.Linear(800, 400)
        self.fc3=nn.Linear(400,10)
        self.dropout = nn.Dropout(0.25)
        #Add dropout if data is overfitting
        #self.dropout=nn.Dropout(.15)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 64 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        return x


net = Net()
#if train_on_gpu:
#    net.cuda()
```

In [55]:

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

In [56]:

```
for epoch in range(5):  # loop over the dataset multiple times
```

```
    running_loss = 0.0
    net.train()
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

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

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

print('Finished Training')
```

```
[1,  2000] loss: 2.283
[1,  4000] loss: 1.991
[1,  6000] loss: 1.742
[1,  8000] loss: 1.601
[1, 10000] loss: 1.524
[1, 12000] loss: 1.454
[2,  2000] loss: 1.381
[2,  4000] loss: 1.338
[2,  6000] loss: 1.292
[2,  8000] loss: 1.250
[2, 10000] loss: 1.199
[2, 12000] loss: 1.192
[3,  2000] loss: 1.122
[3,  4000] loss: 1.093
[3,  6000] loss: 1.098
[3,  8000] loss: 1.057
[3, 10000] loss: 1.067
[3, 12000] loss: 1.027
[4,  2000] loss: 0.994
[4,  4000] loss: 0.980
[4,  6000] loss: 0.960
[4,  8000] loss: 0.945
[4, 10000] loss: 0.946
[4, 12000] loss: 0.936
[5,  2000] loss: 0.901
[5,  4000] loss: 0.879
[5,  6000] loss: 0.881
[5,  8000] loss: 0.876
[5, 10000] loss: 0.889
[5, 12000] loss: 0.866
Finished Training
```

Test the Network

In [57]:

```
dataiter = iter(testloader)
images, labels = dataiter.next()

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

0 20 40 60 80 100 120

```
GroundTruth:    cat  ship  ship plane
```

In [58]:

```
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```

```
Predicted:   frog  frog   car   dog
```

In [59]:

```
correct = 0
total = 0
confusion_matrix = np.zeros([10,10], int)
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()
        for i, l in enumerate(labels):
            confusion_matrix[l.item(), predicted[i].item()] += 1


print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 69 %
```

Accuracy achieved is 69% which is 9% higher than the last best case with smaller subset of test images

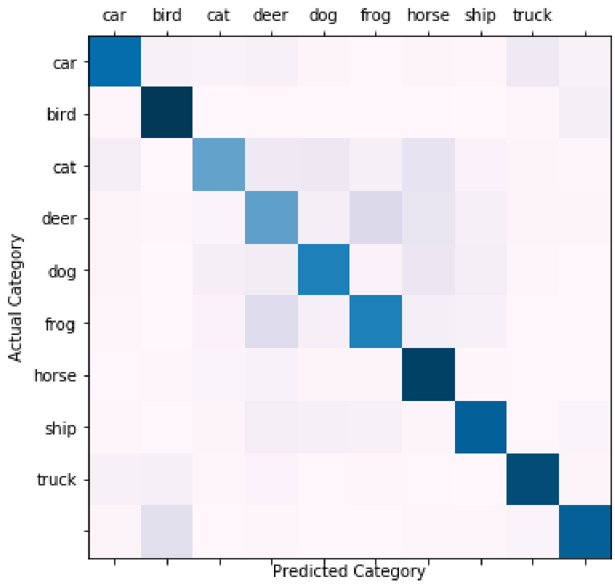
In [60]:

```
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of plane : 65 %
Accuracy of   car : 89 %
Accuracy of  bird : 47 %
Accuracy of   cat : 48 %
Accuracy of  deer : 61 %
Accuracy of   dog : 60 %
Accuracy of  frog : 86 %
Accuracy of horse : 74 %
Accuracy of  ship : 81 %
Accuracy of truck : 74 %
```

In [61]:

```python
import matplotlib.ticker as ticker

fig, ax = plt.subplots(1,1,figsize=(8,6))
cax=ax.matshow(confusion_matrix,cmap=plt.get_cmap('PuBu'))
#fig.colorbar(cax)
plt.ylabel('Actual Category')
plt.yticks(range(10), classes)
plt.xlabel('Predicted Category')
plt.xticks(range(10), classes)
ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
plt.show()
```



In [62]:

```python
print('actual/pred'.ljust(16), end='')
for i,c in enumerate(classes):
    print(c.ljust(10), end='')
print()
for i,r in enumerate(confusion_matrix):
    print(classes[i].ljust(16), end='')
    for idx, p in enumerate(r):
        print(str(p).ljust(10), end='')
    print()

    r = r/np.sum(r)
    print(''.ljust(16), end='')
    for idx, p in enumerate(r):
        print(str(p).ljust(10), end='')
    print()
```

```
actual/pred     plane     car       bird      cat       deer      dog       frog      horse     shi
truck
plane           678       47        35        48        21        7         16        9         98
41
                0.678     0.047     0.035     0.048     0.021     0.007     0.016     0.009     0.C
0.041
car             12        890       2         7         2         2         7         4         11
63
                0.012     0.89      0.002     0.007     0.002     0.002     0.007     0.004     0.C
0.063
bird            65        7         474       96        105       57        132       31        20
13
                0.065     0.007     0.474     0.096     0.105     0.057     0.132     0.031     0.C
0.013
cat             16        12        29        482       63        182       128       53        18
17
                0.016     0.012     0.029     0.482     0.063     0.182     0.128     0.053     0.C
0.017
deer            18        3         58        77        611       32        114       71        13
3
                0.018     0.003     0.058     0.077     0.611     0.032     0.114     0.071     0.C
0.003
```

dog	9	4	30	168	54	611	62	47	7	8
	0.009	0.004	0.03	0.168	0.054	0.611	0.062	0.047	0.0	0.008
frog	7	9	29	38	17	18	857	10	8	7
	0.007	0.009	0.029	0.038	0.017	0.018	0.857	0.01	0.0	0.007
horse	9	5	21	67	51	50	18	743	8	28
	0.009	0.005	0.021	0.067	0.051	0.05	0.018	0.743	0.0	0.028
ship	45	53	9	23	8	9	8	4	824	17
	0.045	0.053	0.009	0.023	0.008	0.009	0.008	0.004	0.8	0.017
truck	20	147	7	15	4	4	14	15	29	745
	0.02	0.147	0.007	0.015	0.004	0.004	0.014	0.015	0.0	0.745