HW1-jy2913

February 12, 2019

Sole team member team: Name: Jin Yan UNI: jy2913 Link to Colaboratory: https://colab.research.google.com/drive/1LFp0QTjDMMv0wDrZK9CqP4pFcUrZGIzs#scrollTo=qsH_VY6Aqw

In [0]:
```
#download CIFAR10 dataset from pytorch
import torch, torchvision
transform = torchvision.transforms.Compose([torchvision.transforms.ToTensor(),torchvis
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True,trans
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, trans
```

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

In [0]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.misc
```

In [0]:
```
""" Pre-processing the dataset """
# get X input from trainset
X_train = trainset.train_data
print(X_train.shape)
# flatten (50000,32,32,3) size of X into (3072,50000) size of matrix
# normalize each pixel value to between 0 and 1
X_train = X_train.reshape(X_train.shape[0],-1).T / 255
print(X_train.shape)
# did the same thing on testset: got X input and flatten it
X_test = testset.test_data
print(X_test.shape)
X_test = X_test.reshape(X_test.shape[0],-1).T / 255
print(X_test.shape)
# get label y from trainset
y_train = trainset.train_labels
print(y_train)
# reshape y from list to numpy array
y_train = np.reshape(np.asarray(y_train),(50000,))
print(y_train.shape)
# did the same thing on testset: got y label and reshape it
y_test = testset.test_labels
```

```
y_test = np.reshape(np.asarray(y_test),(10000,))
print(y_test.shape)

# function takes X input and y labels and ratio to split validation set
def split(X, y, val_ratio):
  val_number = int(val_ratio * X.shape[1])
  random_indice = np.random.permutation(X.shape[1])
  return X[:, random_indice[val_number:]],y[random_indice[val_number:]], X[:, random_i
# split validation set from training set
X_train, y_train, X_val, y_val = split(X_train, y_train, val_ratio = 0.1)
```

```
(50000, 32, 32, 3)
(3072, 50000)
(10000, 32, 32, 3)
(3072, 10000)
[6, 9, 9, 4, 1, 1, 2, 7, 8, 3, 4, 7, 7, 2, 9, 9, 9, 3, 2, 6, 4, 3, 6, 6, 2, 6, 3, 5, 4, 0, 0, 9
(50000,)
(10000,)
```

In [0]:
```
class NeuralNetwork():

    def __init__(self, layer_dims):
        """
        Arguments:
        layer_dims -- A list contains the dimensions of each layer in CNN.

        Attributes generated:
        parameters -- a dict contains parameters "W1", "b1", ..., "WL", "bL"of each co
                    Wl -- weight matrix of shape (layer_dims[l], layer_dims[l-1])
                    bl -- bias vector of shape (layer_dims[l], 1)
        num_layers -- the length of layer_dims list
        """

        np.random.seed(1)
        self.num_layers = len(layer_dims)
        self.layer_dims = layer_dims
        self.parameters = {}
        L = len(self.layer_dims)
        for l in range(1, L):
            self.parameters['W'+ str(l)] = np.random.randn(layer_dims[l], layer_dims[l
            self.parameters['b'+ str(l)] = np.zeros([layer_dims[l], 1])
        self.X_val = None
        self.y_val = None


    def affineForward(self, A, W, b):
        """
```

```
         Implement the linear portion of CNN's forward propagation.

          Arguments:
          A -- activation from previous layer
          W -- weights matrix
          b -- bias vector

          Returns:
          Z -- the input of the activation function
          cache -- a dict stores "A", "W" and "b" during forward propagation
        """
        Z = np.matmul(W, A) + b
        cache = (A, W, b, Z)
        return Z, cache

    def activationForward(self, Z, activation="relu"):
        """
         Implement the linear to activation portion of CNN's forward propagation

          Arguments:
          Z -- output from linear portion of forward propagation
          activation(type) -- "relu" in this case

          Returns:
          A -- the ouput of relu activation
          cache -- a dict stores "A", "W" and "b" during forward propagation
        """
        A = np.maximum(0, Z)
        assert (Z.shape == A.shape)
        return A


    def forwardPropagation(self, X):
        """
        Implement the forward propagation

        Arguments:
        X -- input from input layer as the starting point of forward propagation

        Returns:
        A -- the output of the activation function for each layer
        caches -- a list stores parameters (cache) for each layer during forward propa
        """
        A = X
        caches = []
        for l in range(1, self.num_layers):
            Z, cache = self.affineForward(A,self.parameters['W'+ str(l)], self.paramete
            caches.append(cache)
```

```python
            if (l <= self.num_layers - 1):
                A = self.activationForward(Z)
        return A, caches


    def softmax(self,AL):
        """
        Implement the softmax function

        Arguments:
        AL -- output of linear portion of the output layer (last layer)

        Returns:
        p -- softmax probability
        """
        m = AL.shape[1]
        p = np.exp(AL - np.max(AL, axis=0, keepdims=True))
        p /= np.sum(p, axis=0, keepdims=True)
        return p


    def costFunction(self, AL, y):
        """
        Implement the cross entrophy loss

        Arguments:
        AL -- output of linear portion of the output layer (last layer)
        y -- the labels of data

        Returns:
        cost -- the cross entropy loss value of each iteration of forward propagation
        """
        m = AL.shape[1]
        p = self.softmax(AL)
        cost = -np.sum(np.log(p[y, np.arange(m)])) / m
        return cost


    def derivative_cost(self, AL, y):
        """
        Implement the first step of back propagation: the derivative of cost function

        Arguments:
        AL -- output of linear portion of the output layer (last layer)
        y -- the labels of data

        Returns:
        dAL -- the derivative of cost function over AL
```

```
        """
        m = AL.shape[1]
        p = self.softmax(AL)
        dAL = p.copy()
        dAL[y, np.arange(m)] -= 1
        dAL /= m
        return dAL

    def affineBackward(self, dA_prev, cache):
        """
        Implement the linear portion of backward propagation of one layer (layer l)

        Arguments:
        dAL -- derivative of the cost with respect to the activation output (of curren
        cache -- releases A, W, b, Z values stored from the tuple during forward propa

        Returns:
        dA -- Gradient of the cost over activation output from previous layer (layer l
        dW -- Gradient of the cost over W for the current layer(layerl)
        db -- Gradient of the cost over b for the current layer(layer1)
        """
        m = A.shape[1]
        A, W, b, Z = cache
        dA = np.matmul(W.T, dA_prev)
        dW = np.matmul(dA_prev, A.T) / m
        db = np.sum(dA_prev, axis = 1, keepdims=True) / m
        return dA, dW, db

    def activationBackward(self, dA, cache, activation="relu"):
        """
        Implement the derivative of cost function over relu activation input Z

        Arguments:
        dA -- the gradient of cost function over relu activation ouput A
        cache -- a tuple stores A,W,b,Z for each layer, cache[3] is Z
        activation(type) -- 'relu'

        Returns:
        relu_backward -- the derivative of cost function over Z
        """

        relu_backward = self.derivative_relu(dA, cache[3])
        return relu_backward

    def derivative_relu(self, dA, cache):
        """
        Implement the derivative calculation of relu activation during backpropagation
```

```
        Arguments:
        dA -- the gradient of cost function over relu activation ouput A
        cache -- a tuple stores A,W,b,Z for each layer, cache[3] is Z

        Returns:
        dZ -- the derivative of cost function over Z
        """
        calcu = np.maximum(0, cache)
        calcu[out > 0] = 1
        dZ = calcu * dA
        return dZ

    def backPropagation(self, dAL, y, caches):
        """
        Implement backpropagation for each layer

        Arguments:
        dAL -- the gradient of cost function over the activation output of last layer:
        y -- the labels of data
        caches -- a list stores tuple of A,W,b,Z for each layer

        Returns:
        grads -- a dict stores dWl and dbl for each layer l
        """
        dA = dAL
        grads = {}
        for l in reversed range(0, self.num_layers-1):
            if l < self.num_layers - 1:
                dA = self.activationBackward(dA, caches[l-1])
            dA, dW, db = self.affineBackward(dA, caches[l-1])
            grads['W'+ str(l)] = dW
            grads['b'+ str(l)] = db
        return grads

    def updateParameters(self, grads, alpha):
        """
        Use gradient descent to implement parameters update

        Arguments:
        grads -- a dict stores all parameters gradients for each layer
        alpha -- learning rate

        Returns:
        self.parameters -- a dict stores updated parameters for W and b of each layer
                    parameters["W" + str(l)] = ...
                    parameters["b" + str(l)] = ...
        """
        L = len(self.parameters) // 2
```

```python
        for l in range(1,L-1):
            self.parameters["W" + str(l)] -=  alpha*grads["W"+str(l)]
            self.parameters["b" + str(l)] -=  alpha*grads["b"+str(l)]
        return self.parameters


    def normalized_X(self,X):
        """
        Standardize the input data by substract mean and divided by variance

        Arguments:
        X -- input data

        Return:
        norm_X -- standardized data
        """

        standardize = {}
        standardize['mean'] = np.mean(X, axis = 1, keepdims = True)
        standardize['var'] = np.var(X, axis = 1, keepdims = True)
        norm_X = (X - standardize['mean']) / np.sqrt(standardize['var'])
        return norm_X

    def train(self, X, y, iters, alpha, batch_size, print_every):
        """
        It takes advantage of every function in this class to implement training and v

        Arguments:
        X -- input data
        y -- labels of data
        iters -- number of iterations to run
        alpha -- learning rate
        batch_szie -- number of samples to assign to minibatch
        print_every -- number of iterations to print

        Return:
        Numbewr of iterations, train loss, train_acc, and valid_acc in every 100 itera

        """

        X = self.normalized_X(X)
        for i in range(0, iters):
            X_batch, y_batch = self.get_batch(X, y, batch_size)
            AL, cache = self.forwardPropagation(X_batch)
            loss = self.costFunction(AL, y_batch)
            dAL = self.derivative_cost(AL, y_batch)
            grads = self.backPropagation(dAL, y_batch, cache)
            self.updateParameters(grads, alpha)
```

```
            if i % print_every == 0:
                train_acc = self.score(self.predict(X), y)
                val_acc = self.score(self.predict(self.X_val), self.y_val)
                print('iter={:5}, loss={:.4f}, train_acc={:.4f}, validation_acc={:.4f}

    def predict(self, X):
        """
        It predicts the label given . input x
        Argument:
        X -- input of data
        Return:
        y_pred -- predicted label
        """
        X = self.normalized_X(X)
        AL, _ = self.forwardPropagation(X)
        y_pred = np.argmax(AL, axis = 0)
        return y_pred


    def score(self, y_pred, y_true):
        """
        It calculates the percentage of correct predicted labels over true labels

        Argument:
        y_pred-- predicted labels
        y_true -- true labels

        REturn: percentage of correct predicted labels
        """
        correct = np.mean(y_pred == y_true)
        return correct

    def load_validation_set(self, X_val, y_val):
        """
        Load validation set to CNN
        """
        self.X_val = X_val
        self.y_val = y_val


    def get_batch(self, X, y, batch_size):
        """
        Load minibatch to CNN
        """

        batch_index = np.random.randint(X.shape[1], size = batch_size)
        X_batch = X[:, batch_index]
        y_batch = y[batch_index]
```

```
            return X_batch, y_batch
```

```
In [0]: """
        Train CNN and validate at the same time
        """
        # I tried iddifferent architecture of CNN and it turns out 3 hidden layers of 1024, 25
        # I also tried different learning drate and batch size and it turns out alpha=1 and ba
        # In this trained CNN, I reach 54.12% validation accuracy and 52.99% test accuracy in
        layer_dims = [X_train.shape[0], 1024, 256, 128, 10]
        CNN = NeuralNetwork(layer_dims)
        CNN.load_validation_set(X_val, y_val)
        CNN.train(X_train, y_train, iters=10000, alpha=1, batch_size=100, print_every=100)
```

```
iter=    0, loss=2.3617, train_acc=0.0999, validation_acc=0.1064
iter=  100, loss=2.0751, train_acc=0.2959, validation_acc=0.2948
iter=  200, loss=1.8461, train_acc=0.3493, validation_acc=0.3530
iter=  300, loss=1.7975, train_acc=0.3786, validation_acc=0.3762
iter=  400, loss=1.6927, train_acc=0.4006, validation_acc=0.4002
iter=  500, loss=1.7257, train_acc=0.4153, validation_acc=0.4088
iter=  600, loss=1.6456, train_acc=0.4278, validation_acc=0.4176
iter=  700, loss=1.7837, train_acc=0.4386, validation_acc=0.4272
iter=  800, loss=1.7971, train_acc=0.4468, validation_acc=0.4314
iter=  900, loss=1.6531, train_acc=0.4553, validation_acc=0.4396
iter= 1000, loss=1.6428, train_acc=0.4612, validation_acc=0.4514
iter= 1100, loss=1.5892, train_acc=0.4674, validation_acc=0.4504
iter= 1200, loss=1.6917, train_acc=0.4736, validation_acc=0.4532
iter= 1300, loss=1.4878, train_acc=0.4779, validation_acc=0.4560
iter= 1400, loss=1.4416, train_acc=0.4837, validation_acc=0.4680
iter= 1500, loss=1.4070, train_acc=0.4885, validation_acc=0.4596
iter= 1600, loss=1.3062, train_acc=0.4958, validation_acc=0.4642
iter= 1700, loss=1.4820, train_acc=0.4989, validation_acc=0.4618
iter= 1800, loss=1.3706, train_acc=0.5010, validation_acc=0.4672
iter= 1900, loss=1.4472, train_acc=0.5081, validation_acc=0.4714
iter= 2000, loss=1.6072, train_acc=0.5118, validation_acc=0.4750
iter= 2100, loss=1.4100, train_acc=0.5090, validation_acc=0.4700
iter= 2200, loss=1.4180, train_acc=0.5184, validation_acc=0.4834
iter= 2300, loss=1.4381, train_acc=0.5197, validation_acc=0.4812
iter= 2400, loss=1.4284, train_acc=0.5244, validation_acc=0.4872
iter= 2500, loss=1.3580, train_acc=0.5291, validation_acc=0.4840
iter= 2600, loss=1.2492, train_acc=0.5353, validation_acc=0.4894
iter= 2700, loss=1.5415, train_acc=0.5384, validation_acc=0.4938
iter= 2800, loss=1.4538, train_acc=0.5415, validation_acc=0.4924
iter= 2900, loss=1.2612, train_acc=0.5461, validation_acc=0.4922
iter= 3000, loss=1.3019, train_acc=0.5513, validation_acc=0.4980
iter= 3100, loss=1.2984, train_acc=0.5501, validation_acc=0.5016
iter= 3200, loss=1.1218, train_acc=0.5572, validation_acc=0.4976
iter= 3300, loss=1.3844, train_acc=0.5569, validation_acc=0.5042
```

```
iter= 3400, loss=1.3159, train_acc=0.5593, validation_acc=0.5064
iter= 3500, loss=1.3674, train_acc=0.5641, validation_acc=0.5018
iter= 3600, loss=1.3703, train_acc=0.5645, validation_acc=0.5016
iter= 3700, loss=1.4296, train_acc=0.5690, validation_acc=0.5096
iter= 3800, loss=1.4543, train_acc=0.5718, validation_acc=0.5084
iter= 3900, loss=1.2558, train_acc=0.5754, validation_acc=0.5092
iter= 4000, loss=1.2729, train_acc=0.5790, validation_acc=0.5084
iter= 4100, loss=1.1926, train_acc=0.5764, validation_acc=0.5186
iter= 4200, loss=1.3360, train_acc=0.5815, validation_acc=0.5102
iter= 4300, loss=1.1685, train_acc=0.5840, validation_acc=0.5126
iter= 4400, loss=1.0960, train_acc=0.5850, validation_acc=0.5154
iter= 4500, loss=1.2709, train_acc=0.5880, validation_acc=0.5202
iter= 4600, loss=1.1889, train_acc=0.5892, validation_acc=0.5116
iter= 4700, loss=1.2040, train_acc=0.5927, validation_acc=0.5222
iter= 4800, loss=1.1178, train_acc=0.5971, validation_acc=0.5148
iter= 4900, loss=1.0266, train_acc=0.6021, validation_acc=0.5226
iter= 5000, loss=1.3410, train_acc=0.6036, validation_acc=0.5230
iter= 5100, loss=1.1866, train_acc=0.6079, validation_acc=0.5294
iter= 5200, loss=1.0812, train_acc=0.6103, validation_acc=0.5268
iter= 5300, loss=1.1771, train_acc=0.6108, validation_acc=0.5256
iter= 5400, loss=1.3337, train_acc=0.6172, validation_acc=0.5304
iter= 5500, loss=1.1678, train_acc=0.6180, validation_acc=0.5302
iter= 5600, loss=1.2106, train_acc=0.6181, validation_acc=0.5242
iter= 5700, loss=0.8486, train_acc=0.6255, validation_acc=0.5272
iter= 5800, loss=1.0979, train_acc=0.6220, validation_acc=0.5260
iter= 5900, loss=0.9381, train_acc=0.6273, validation_acc=0.5298
iter= 6000, loss=1.1777, train_acc=0.6294, validation_acc=0.5304
iter= 6100, loss=1.1077, train_acc=0.6285, validation_acc=0.5290
iter= 6200, loss=1.2029, train_acc=0.6188, validation_acc=0.5264
iter= 6300, loss=1.1024, train_acc=0.6365, validation_acc=0.5316
iter= 6400, loss=0.9901, train_acc=0.6314, validation_acc=0.5316
iter= 6500, loss=1.0325, train_acc=0.6378, validation_acc=0.5340
iter= 6600, loss=1.0798, train_acc=0.6381, validation_acc=0.5304
iter= 6700, loss=0.9387, train_acc=0.6422, validation_acc=0.5296
iter= 6800, loss=1.2228, train_acc=0.6452, validation_acc=0.5368
iter= 6900, loss=1.1876, train_acc=0.6407, validation_acc=0.5322
iter= 7000, loss=0.9095, train_acc=0.6509, validation_acc=0.5354
iter= 7100, loss=1.0665, train_acc=0.6489, validation_acc=0.5364
iter= 7200, loss=1.2499, train_acc=0.6576, validation_acc=0.5372
iter= 7300, loss=1.0148, train_acc=0.6551, validation_acc=0.5394
iter= 7400, loss=1.0043, train_acc=0.6538, validation_acc=0.5356
iter= 7500, loss=1.0910, train_acc=0.6589, validation_acc=0.5412
iter= 7600, loss=0.9433, train_acc=0.6632, validation_acc=0.5458
iter= 7700, loss=1.0436, train_acc=0.6630, validation_acc=0.5364
iter= 7800, loss=0.8558, train_acc=0.6640, validation_acc=0.5358
iter= 7900, loss=0.9926, train_acc=0.6714, validation_acc=0.5422
iter= 8000, loss=1.1019, train_acc=0.6728, validation_acc=0.5422
iter= 8100, loss=0.9499, train_acc=0.6720, validation_acc=0.5338
```

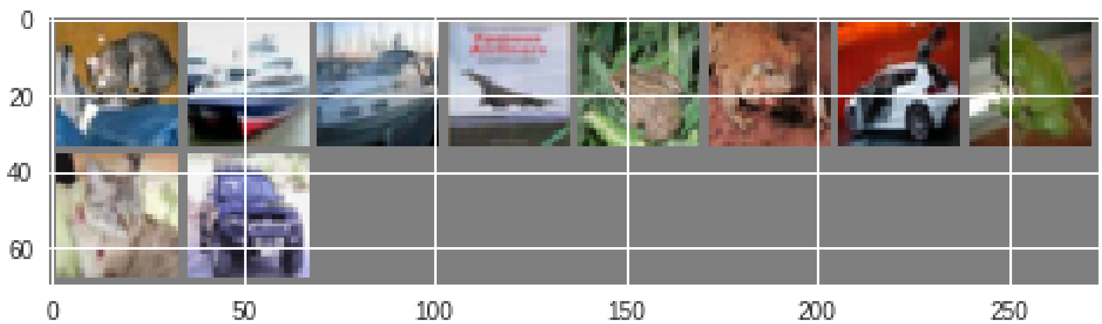
```
iter= 8200, loss=0.9922, train_acc=0.6770, validation_acc=0.5386
iter= 8300, loss=0.8384, train_acc=0.6788, validation_acc=0.5456
iter= 8400, loss=1.0024, train_acc=0.6706, validation_acc=0.5364
iter= 8500, loss=0.8492, train_acc=0.6794, validation_acc=0.5352
iter= 8600, loss=0.9987, train_acc=0.6816, validation_acc=0.5416
iter= 8700, loss=0.9400, train_acc=0.6886, validation_acc=0.5458
iter= 8800, loss=0.9616, train_acc=0.6854, validation_acc=0.5412
iter= 8900, loss=1.0930, train_acc=0.6908, validation_acc=0.5464
iter= 9000, loss=0.9944, train_acc=0.6914, validation_acc=0.5452
iter= 9100, loss=0.8386, train_acc=0.6916, validation_acc=0.5486
iter= 9200, loss=0.8467, train_acc=0.6893, validation_acc=0.5408
iter= 9300, loss=0.9941, train_acc=0.6972, validation_acc=0.5392
iter= 9400, loss=1.0225, train_acc=0.6920, validation_acc=0.5326
iter= 9500, loss=0.8249, train_acc=0.7022, validation_acc=0.5426
iter= 9600, loss=1.0521, train_acc=0.7055, validation_acc=0.5456
iter= 9700, loss=0.8433, train_acc=0.7016, validation_acc=0.5426
iter= 9800, loss=0.9047, train_acc=0.7047, validation_acc=0.5394
iter= 9900, loss=0.8353, train_acc=0.7116, validation_acc=0.5412
```

In [0]:
```
#test set accuracy
y_pred = CNN.predict(X_test)
test_acc = CNN.score(y_pred, y_test)
print('test_acc ={:4}'.format(test_acc))
```

```
test_acc =0.5299
```

In [0]:
```
#visualize the prediction result of trained CNN on some of test set images
import torchvision
import torchvision.transforms as transforms
testloader = torch.utils.data.DataLoader(testset, batch_size=10,
                                         shuffle=False, num_workers=2)

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

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

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print("testlabels: " + ' '.join('%5s' % classes[labels[j]] for j in range(10)))
print("predlabels: " + ' '.join('%5s' % classes[y_pred[j]] for j in range(10)))
```




```
testlabels:   cat  ship  ship plane  frog  frog   car  frog   cat   car
predlabels:   cat  ship plane plane  deer  frog   cat  frog   dog   car
```

In [0]:
```
import torch
import torchvision
import torchvision.transforms as transforms
from torchvision import datasets
import numpy as np
from torch.utils.data.sampler import SubsetRandomSampler
```

In [0]:
```
#download CIFAR10 data from pytorch

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


train_dataset = datasets.CIFAR10(root='./data', train=True,
                                download=True, transform=transform)
valid_dataset = datasets.CIFAR10(root='./data', train=True,
                                download=True, transform=transform)
```

```
#generate index for validation set
#use SubsetRandomSampler to split validation set
valid_size = 0.1
num_train = len(train_dataset)
indices = list(range(num_train))
split = int(np.floor(valid_size * num_train))
np.random.seed(0)
np.random.shuffle(indices)

train_idx, valid_idx = indices[split:], indices[:split]
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

#load train set, validation set, and test set

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=50,
                                           sampler=train_sampler,num_workers=2, pin_mem
valid_loader = torch.utils.data.DataLoader(train_dataset, batch_size=50,
                                           sampler=valid_sampler,num_workers=2, pin_mem
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=50,
                                         shuffle=False, num_workers=2)

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

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.
Files already downloaded and verified
Files already downloaded and verified
```

In [0]:
```
import matplotlib.pyplot as plt
import numpy as np


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



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

```
imshow(torchvision.utils.make_grid(images))

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

```
horse   dog  bird  frog
```

```
In [0]: import torch.nn as nn
        import torch.nn.functional as F
        # model_b is my baseline model. All my following models are generated upon this model
        #model_b consists of two convolutional layers and two fc layers
        # model_b has validation accuracy and test accuracy of 73%

        class CIFAR10_base(nn.Module):
            def __init__(self):
                super(CIFAR10_base, self).__init__()
                self.conv1 = nn.Conv2d(3, 32, 5, padding=2)
                self.conv2 = nn.Conv2d(32, 64, 5, padding=2)
                self.fc1 = nn.Linear(64*8*8, 1024)
                self.fc2 = nn.Linear(1024,10)
```

```
    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, 64*8*8) # reshape before sending to fc layer
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x)

model_b = CIFAR10_base()
```

In [0]:
```
print(model_b)
```

```
CIFAR10_base(
  (conv1): Conv2d(3, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv2): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (fc1): Linear(in_features=4096, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=10, bias=True)
)
```

In [0]:
```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model_b.parameters(), lr=0.01, momentum=0.9)
```

In [0]:
```
def train(model,train_loader,criterion,optimizer,epoch):
    model.cuda()
    i = 0

    model.train() # to set train mode for drop out
    train_loss, train_accu = [], []
    for images, labels in train_loader:
        # send tensors to GPU
        images, labels = images.cuda(), labels.cuda()

        optimizer.zero_grad()               # zero the parameter gradients
        outputs = model(images)             # calls the forward function of model, i.e
        loss = F.nll_loss(outputs, labels)  # calculate loss
        loss.backward()                     # calculate gradients
        train_loss.append(loss.item())
        optimizer.step()                    # update learnable parameters
        predictions = outputs.data.max(1)[1]# column at idx 1 has actual prob.

        accuracy = np.sum(predictions.cpu().numpy()==labels.cpu().numpy())/batch_size*
        #return loss.item(), accuracy
        train_accu.append(accuracy)
        if i % 1000 == 0:
            print('Train Step: {}\tTrain Loss: {:.3f}\tTrain Accuracy: {:.3f}'.format(e
        i += 1
```

```
In [0]: def validation(model, valid_loader, epoch):
            i = 0
            model.eval()
            correct = 0
            for images, labels in valid_loader:
                with torch.no_grad(): # so that computation graph history is not stored
                    images, labels = images.cuda(), labels.cuda() # send tensors to GPU
                    outputs = model(images)
                    predictions = outputs.data.max(1)[1]
                    correct += predictions.eq(labels.data).sum()
                    accuracy = 100.0 * correct / (len(valid_loader.dataset) * valid_size)
                    if i == 99:

                        print('Validation step: {}\tValidation  accuracy: {:.3f}'.format(epoch
                    i += 1
```

```
In [0]: batch_size = 50
        # send model to GPU

        #train_loss, train_accu, valid_accu = [], [],[]

        for epoch in range(20):
            train(model_b,train_loader,criterion,optimizer,epoch)
            validation(model_b,valid_loader,epoch)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:19: UserWarning: Implicit dimensi

Train Step: 0        Train Loss: 1.475        Train Accuracy: 46.000
Validation step: 0        Validation  accuracy: 67.000
Train Step: 1000        Train Loss: 0.680        Train Accuracy: 78.000
Validation step: 1000        Validation  accuracy: 70.000
Train Step: 2000        Train Loss: 0.768        Train Accuracy: 78.000
Validation step: 2000        Validation  accuracy: 72.000
Train Step: 3000        Train Loss: 0.296        Train Accuracy: 90.000
Validation step: 3000        Validation  accuracy: 72.000
Train Step: 4000        Train Loss: 0.227        Train Accuracy: 94.000
Validation step: 4000        Validation  accuracy: 73.000
Train Step: 5000        Train Loss: 0.195        Train Accuracy: 92.000
Validation step: 5000        Validation  accuracy: 72.000
Train Step: 6000        Train Loss: 0.091        Train Accuracy: 98.000
Validation step: 6000        Validation  accuracy: 73.000
Train Step: 7000        Train Loss: 0.081        Train Accuracy: 94.000
Validation step: 7000        Validation  accuracy: 73.000
```

```
Train Step: 8000       Train Loss: 0.012       Train Accuracy: 100.000
```

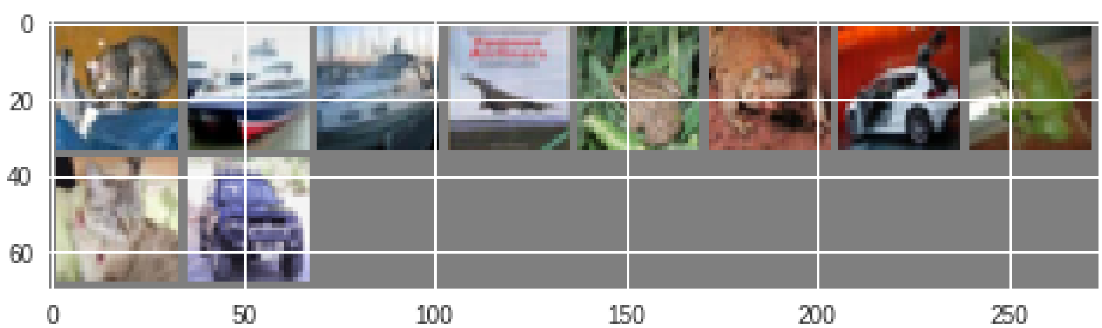
```
        ---------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call last)

        <ipython-input-16-bec414c5a960> in <module>()
          5
          6 for epoch in range(20):
    ----> 7     train(model_b,train_loader,criterion,optimizer,epoch)
          8     validation(model_b,valid_loader,epoch)
          9


        <ipython-input-10-fdc09cd7c551> in train(model, train_loader, criterion, optimizer, epo
         17         predictions = outputs.data.max(1)[1]# column at idx 1 has actual prob.
         18
    ---> 19         accuracy = np.sum(predictions.cpu().numpy()==labels.cpu().numpy())/batch_s
         20         #return loss.item(), accuracy
         21         train_accu.append(accuracy)


        KeyboardInterrupt:
```

In [0]:
```
model_b.eval()
correct = 0
for images, labels in testloader:
    with torch.no_grad(): # so that computation graph history is not stored
        images, labels = images.cuda(), labels.cuda() # send tensors to GPU
        outputs = model_b(images)
        predictions_T = outputs.data.max(1)[1]
        correct += predictions_T.eq(labels.data).sum()


print('Test set accuracy: {:.2f}%'.format(100.0 * correct / len(testloader.dataset)))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:19: UserWarning: Implicit dimensic


Test set accuracy: 73.00%
```

In [0]:
```
import torchvision
import torchvision.transforms as transforms
testloader = torch.utils.data.DataLoader(testset, batch_size=10,
                                         shuffle=False, num_workers=2)
```

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

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

# show images
imshow(torchvision.utils.make_grid(images))
images, labels = images.cuda(), labels.cuda() # send tensors to GPU
outputs = model_b(images)
predictions_T = outputs.data.max(1)[1]
correct += predictions_T.eq(labels.data).sum()

# print labels
print("testlabels: " + ' '.join('%5s' % classes[labels.data[j]] for j in range(10)))
print("predlabels: " + ' '.join('%5s' % classes[predictions_T[j]] for j in range(10)))
```




```
testlabels:   cat  ship  ship plane  frog  frog   car  frog   cat   car
predlabels:   cat   car  ship plane  deer  frog   car  frog   cat   car
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:19: UserWarning: Implicit dimensio
```

```
In [0]: import torch.nn as nn
        import torch.nn.functional as F
        # model1 add a dropout layer between fc1 and fc2 of model_b
        # improve val acc to 74% and test acc to 75% compared to model_b

        class CIFAR10_1(nn.Module):
            def __init__(self):
                super(CIFAR10_1, self).__init__()
                self.conv1 = nn.Conv2d(3, 32, 5, padding=2)
                self.conv2 = nn.Conv2d(32, 64, 5, padding=2)
                self.fc1 = nn.Linear(64*8*8, 1024)
                self.fc2 = nn.Linear(1024,10)

            def forward(self, x):
                x = F.max_pool2d(F.relu(self.conv1(x)), 2)
                x = F.max_pool2d(F.relu(self.conv2(x)), 2)
                x = x.view(-1, 64*8*8) # reshape before sending to fc layer
                x = F.relu(self.fc1(x))
                x = F.dropout(x, training=self.training)
                x = self.fc2(x)
                return F.log_softmax(x)
                return F.log_softmax(x)

        model1 = CIFAR10_1()
```

```
In [0]: import torch.optim as optim

        criterion = nn.CrossEntropyLoss()
        optimizer = optim.SGD(model1.parameters(), lr=0.01, momentum=0.9)
```

```
In [0]: batch_size = 50
        # send model to GPU

        #train_loss, train_accu, valid_accu = [], [],[]

        for epoch in range(20):
            train(model1,train_loader,criterion,optimizer,epoch)
            validation(model1,valid_loader,epoch)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:21: UserWarning: Implicit dimensio


Train Step: 0        Train Loss: 2.314        Train Accuracy: 6.000
Validation step: 0        Validation  accuracy: 57.000
Train Step: 1000        Train Loss: 1.200        Train Accuracy: 58.000
Validation step: 1000        Validation  accuracy: 65.000
```

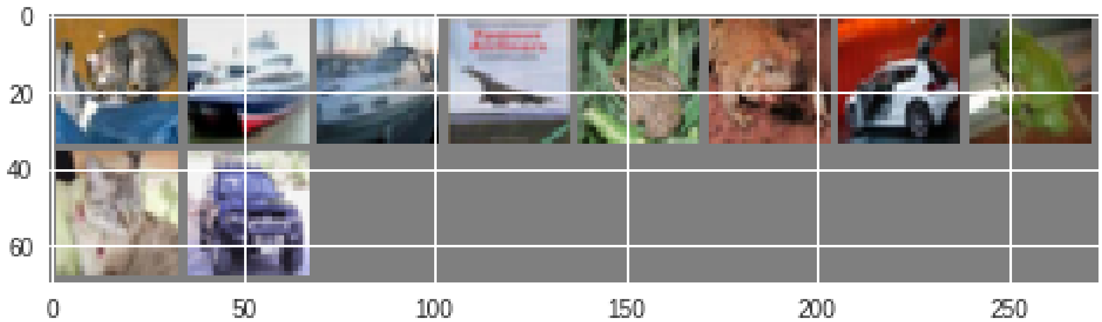
```
Train Step: 2000       Train Loss: 1.041       Train Accuracy: 66.000
Validation step: 2000       Validation  accuracy: 69.000
Train Step: 3000       Train Loss: 0.485       Train Accuracy: 84.000
Validation step: 3000       Validation  accuracy: 71.000
Train Step: 4000       Train Loss: 0.633       Train Accuracy: 76.000
Validation step: 4000       Validation  accuracy: 72.000
Train Step: 5000       Train Loss: 0.478       Train Accuracy: 92.000
Validation step: 5000       Validation  accuracy: 74.000
Train Step: 6000       Train Loss: 0.571       Train Accuracy: 88.000
Validation step: 6000       Validation  accuracy: 75.000
Train Step: 7000       Train Loss: 0.204       Train Accuracy: 90.000
Validation step: 7000       Validation  accuracy: 73.000
Train Step: 8000       Train Loss: 0.329       Train Accuracy: 90.000
Validation step: 8000       Validation  accuracy: 75.000
Train Step: 9000       Train Loss: 0.129       Train Accuracy: 98.000
Validation step: 9000       Validation  accuracy: 74.000
Train Step: 10000       Train Loss: 0.130       Train Accuracy: 96.000
Validation step: 10000       Validation  accuracy: 75.000
Train Step: 11000       Train Loss: 0.103       Train Accuracy: 96.000
Validation step: 11000       Validation  accuracy: 75.000
Train Step: 12000       Train Loss: 0.075       Train Accuracy: 98.000
Validation step: 12000       Validation  accuracy: 75.000
Train Step: 13000       Train Loss: 0.148       Train Accuracy: 94.000
Validation step: 13000       Validation  accuracy: 75.000
Train Step: 14000       Train Loss: 0.117       Train Accuracy: 96.000
Validation step: 14000       Validation  accuracy: 75.000
Train Step: 15000       Train Loss: 0.071       Train Accuracy: 98.000
Validation step: 15000       Validation  accuracy: 75.000
Train Step: 16000       Train Loss: 0.034       Train Accuracy: 98.000
Validation step: 16000       Validation  accuracy: 75.000
Train Step: 17000       Train Loss: 0.055       Train Accuracy: 98.000
Validation step: 17000       Validation  accuracy: 74.000
Train Step: 18000       Train Loss: 0.050       Train Accuracy: 96.000
Validation step: 18000       Validation  accuracy: 74.000
Train Step: 19000       Train Loss: 0.128       Train Accuracy: 94.000
Validation step: 19000       Validation  accuracy: 74.000
```

```
In [0]: model1.eval()
        correct = 0
        for images, labels in testloader:
            with torch.no_grad(): # so that computation graph history is not stored
                images, labels = images.cuda(), labels.cuda() # send tensors to GPU
                outputs = model1(images)
                predictions = outputs.data.max(1)[1]
                correct += predictions.eq(labels.data).sum()

        print('Test set accuracy: {:.2f}%'.format(100.0 * correct / len(testloader.dataset)))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:21: UserWarning: Implicit dimensio
```

```
Test set accuracy: 75.00%
```

In [0]:
```python
import torchvision
import torchvision.transforms as transforms
testloader = torch.utils.data.DataLoader(testset, batch_size=10,
                                          shuffle=False, num_workers=2)

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

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

# show images
imshow(torchvision.utils.make_grid(images))
images, labels = images.cuda(), labels.cuda() # send tensors to GPU
outputs = model1(images)
predictions_T = outputs.data.max(1)[1]
correct += predictions_T.eq(labels.data).sum()

# print labels
print("testlabels: " + ' '.join('%5s' % classes[labels.data[j]] for j in range(10)))
print("predlabels: " + ' '.join('%5s' % classes[predictions_T[j]] for j in range(10)))
```



```
testlabels:   cat  ship  ship plane  frog  frog   car  frog   cat   car
predlabels:   cat  ship  ship plane  frog  frog   car  frog   cat   car
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:21: UserWarning: Implicit dimensio
```

In [0]:
```
import torch.nn as nn
import torch.nn.functional as F

#model2 changes the size of kernel from 5x5 to 3x3
#model2 uses nn.Sequential mode to include a ReLu activation and a Dropout in the firs
#model 2 improve valid_acc and test_acc to 76%
class CIFAR10_2(nn.Module):
    def __init__(self):
        super(CIFAR10_2, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Dropout(p = 0.2),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Dropout(p = 0.2),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.drop_out = nn.Dropout()
        self.fc1 = nn.Linear(64*8*8, 1024)
        self.fc2 = nn.Linear(1024, 10)
    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = x.view(-1, 64*8*8) # reshape before sending to fc layer
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training) # default p=0.5
```

```
            x = self.fc2(x)
            return F.log_softmax(x)

    model2 = CIFAR10_2()
```

In [0]:
```
print(model2)
```

```
CIFAR10_2(
  (layer1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Dropout(p=0.2)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Dropout(p=0.2)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (drop_out): Dropout(p=0.5)
  (fc1): Linear(in_features=4096, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=10, bias=True)
)
```

In [0]:
```
import torch.nn.functional as F
loss = F.nll_loss(outputs,labels)
```

1 New Section

In [0]:
```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model2.parameters(), lr=0.01, momentum=0.9)
```

In [0]:
```
batch_size = 50
# send model to GPU

#train_loss, train_accu, valid_accu = [], [],[]

for epoch in range(40):
    train(model2,train_loader,criterion,optimizer,epoch)
    validation(model2,valid_loader,epoch)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:29: UserWarning: Implicit dimensi
```

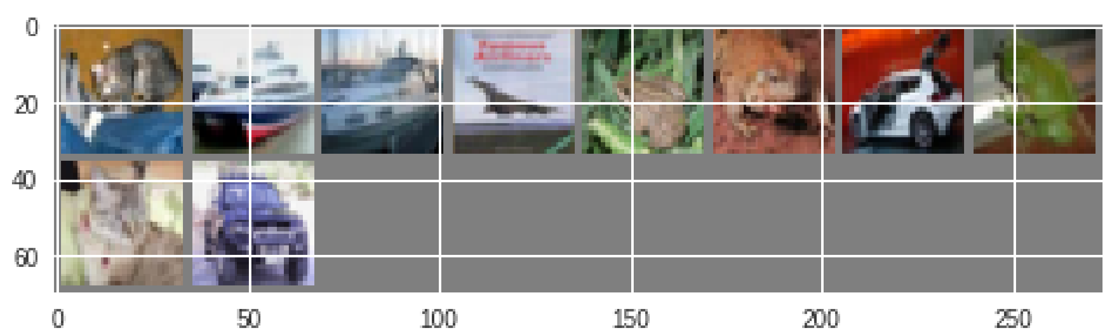
```
Train Step: 0        Train Loss: 2.305        Train Accuracy: 6.000
Validation step: 0        Validation  accuracy: 52.000
Train Step: 1000        Train Loss: 1.349        Train Accuracy: 52.000
Validation step: 1000        Validation  accuracy: 65.000
Train Step: 2000        Train Loss: 0.881        Train Accuracy: 72.000
Validation step: 2000        Validation  accuracy: 68.000
Train Step: 3000        Train Loss: 0.964        Train Accuracy: 68.000
Validation step: 3000        Validation  accuracy: 72.000
Train Step: 4000        Train Loss: 0.547        Train Accuracy: 86.000
Validation step: 4000        Validation  accuracy: 72.000
Train Step: 5000        Train Loss: 0.594        Train Accuracy: 80.000
Validation step: 5000        Validation  accuracy: 72.000
Train Step: 6000        Train Loss: 0.621        Train Accuracy: 74.000
Validation step: 6000        Validation  accuracy: 73.000
Train Step: 7000        Train Loss: 0.338        Train Accuracy: 86.000
Validation step: 7000        Validation  accuracy: 74.000
Train Step: 8000        Train Loss: 0.385        Train Accuracy: 88.000
Validation step: 8000        Validation  accuracy: 74.000
Train Step: 9000        Train Loss: 0.416        Train Accuracy: 84.000
Validation step: 9000        Validation  accuracy: 75.000
Train Step: 10000        Train Loss: 0.294        Train Accuracy: 92.000
Validation step: 10000        Validation  accuracy: 75.000
Train Step: 11000        Train Loss: 0.150        Train Accuracy: 94.000
Validation step: 11000        Validation  accuracy: 76.000
Train Step: 12000        Train Loss: 0.167        Train Accuracy: 96.000
Validation step: 12000        Validation  accuracy: 75.000
Train Step: 13000        Train Loss: 0.172        Train Accuracy: 94.000
Validation step: 13000        Validation  accuracy: 75.000
Train Step: 14000        Train Loss: 0.140        Train Accuracy: 94.000
Validation step: 14000        Validation  accuracy: 75.000
Train Step: 15000        Train Loss: 0.056        Train Accuracy: 98.000
Validation step: 15000        Validation  accuracy: 75.000
Train Step: 16000        Train Loss: 0.293        Train Accuracy: 90.000
Validation step: 16000        Validation  accuracy: 75.000
Train Step: 17000        Train Loss: 0.158        Train Accuracy: 94.000
Validation step: 17000        Validation  accuracy: 76.000
Train Step: 18000        Train Loss: 0.095        Train Accuracy: 96.000
Validation step: 18000        Validation  accuracy: 75.000
Train Step: 19000        Train Loss: 0.149        Train Accuracy: 94.000
Validation step: 19000        Validation  accuracy: 76.000
Train Step: 20000        Train Loss: 0.147        Train Accuracy: 98.000
Validation step: 20000        Validation  accuracy: 76.000
Train Step: 21000        Train Loss: 0.123        Train Accuracy: 96.000
Validation step: 21000        Validation  accuracy: 75.000
Train Step: 22000        Train Loss: 0.350        Train Accuracy: 94.000
Validation step: 22000        Validation  accuracy: 76.000
Train Step: 23000        Train Loss: 0.219        Train Accuracy: 96.000
Validation step: 23000        Validation  accuracy: 76.000
```

```
Train Step: 24000       Train Loss: 0.044       Train Accuracy: 100.000
Validation step: 24000       Validation  accuracy: 76.000
Train Step: 25000       Train Loss: 0.188       Train Accuracy: 94.000
Validation step: 25000       Validation  accuracy: 75.000
Train Step: 26000       Train Loss: 0.074       Train Accuracy: 98.000
Validation step: 26000       Validation  accuracy: 76.000
Train Step: 27000       Train Loss: 0.026       Train Accuracy: 98.000
Validation step: 27000       Validation  accuracy: 75.000
Train Step: 28000       Train Loss: 0.106       Train Accuracy: 96.000
Validation step: 28000       Validation  accuracy: 75.000
Train Step: 29000       Train Loss: 0.230       Train Accuracy: 94.000
Validation step: 29000       Validation  accuracy: 76.000
Train Step: 30000       Train Loss: 0.155       Train Accuracy: 94.000
Validation step: 30000       Validation  accuracy: 76.000
Train Step: 31000       Train Loss: 0.029       Train Accuracy: 100.000
Validation step: 31000       Validation  accuracy: 76.000
Train Step: 32000       Train Loss: 0.033       Train Accuracy: 100.000
Validation step: 32000       Validation  accuracy: 76.000
Train Step: 33000       Train Loss: 0.039       Train Accuracy: 100.000
Validation step: 33000       Validation  accuracy: 76.000
Train Step: 34000       Train Loss: 0.020       Train Accuracy: 100.000
Validation step: 34000       Validation  accuracy: 75.000
Train Step: 35000       Train Loss: 0.041       Train Accuracy: 96.000
Validation step: 35000       Validation  accuracy: 75.000
Train Step: 36000       Train Loss: 0.024       Train Accuracy: 100.000
Validation step: 36000       Validation  accuracy: 76.000
Train Step: 37000       Train Loss: 0.104       Train Accuracy: 96.000
Validation step: 37000       Validation  accuracy: 75.000
Train Step: 38000       Train Loss: 0.095       Train Accuracy: 94.000
Validation step: 38000       Validation  accuracy: 76.000
Train Step: 39000       Train Loss: 0.274       Train Accuracy: 90.000
Validation step: 39000       Validation  accuracy: 76.000
```

```
In [0]: model2.eval()
        correct = 0
        for images, labels in testloader:
            with torch.no_grad(): # so that computation graph history is not stored
                images, labels = images.cuda(), labels.cuda() # send tensors to GPU
                outputs = model2(images)
                predictions = outputs.data.max(1)[1]
                correct += predictions.eq(labels.data).sum()

        print('Test set accuracy: {:.2f}%'.format(100.0 * correct / len(testloader.dataset)))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:29: UserWarning: Implicit dimensi

Test set accuracy: 76.00%
```

In [0]:
```python
import torchvision
import torchvision.transforms as transforms
testloader = torch.utils.data.DataLoader(testset, batch_size=10,
                                          shuffle=False, num_workers=2)

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

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

# show images
imshow(torchvision.utils.make_grid(images))
images, labels = images.cuda(), labels.cuda() # send tensors to GPU
outputs = model2(images)
predictions_T = outputs.data.max(1)[1]
correct += predictions_T.eq(labels.data).sum()

# print labels
print("testlabels: " + ' '.join('%5s' % classes[labels.data[j]] for j in range(10)))
print("predlabels: " + ' '.join('%5s' % classes[predictions_T[j]] for j in range(10)))
```



```
testlabels:   cat  ship  ship plane  frog  frog   car  frog   cat   car
predlabels:   cat  ship  ship plane  frog  frog   car  frog   cat   car


/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:29: UserWarning: Implicit dimensi
```

```
In [0]: import torch.nn as nn
        import torch.nn.functional as F
        # Model3 adds batchNorm2d layer after ReLu activation in the first two convolutional l
        # Model3 add fc3 after fc2 in fully-connected layers
        # Model3 still incooperates dropout layers in both convolutional and fully-connected l
        # Model3 has improved valid_acc and test_acc to 78%

        class CIFAR10_3(nn.Module):
            def __init__(self):
                super(CIFAR10_3, self).__init__()
                self.layer1 = nn.Sequential(
                    nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
                    nn.ReLU(),
                    nn.BatchNorm2d(32),
                    nn.MaxPool2d(kernel_size=2, stride=2),
                    nn.Dropout(p=0.2))
                self.layer2 = nn.Sequential(
                    nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
                    nn.ReLU(),
                    nn.BatchNorm2d(64),
                    nn.MaxPool2d(kernel_size=2, stride=2),
                    nn.Dropout(p=0.2))
                self.drop_out = nn.Dropout()
                self.fc1 = nn.Linear(64*8*8, 1024)
                self.fc2 = nn.Linear(1024, 100)
                self.fc3 = nn.Linear(100, 10)
            def forward(self, x):
                x = self.layer1(x)
                x = self.layer2(x)
                x = x.view(-1, 64*8*8) # reshape before sending to fc layer
                x = F.relu(self.fc1(x))
                x = F.dropout(x, training=self.training)
                x = F.relu(self.fc2(x))
                x = F.dropout(x, training=self.training) # default p=0.5
                x = self.fc3(x)
                return F.log_softmax(x)


        model3 = CIFAR10_3()
```

In [0]:
```
print(model3)
```

```
CIFAR10_3(
  (layer1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.2)
  )
  (layer2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.2)
  )
  (drop_out): Dropout(p=0.5)
  (fc1): Linear(in_features=4096, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=100, bias=True)
  (fc3): Linear(in_features=100, out_features=10, bias=True)
)
```

In [0]:
```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model3.parameters(), lr=0.01, momentum=0.9)
```

In [0]:
```
batch_size = 50
# send model to GPU

#train_loss, train_accu, valid_accu = [], [],[]

for epoch in range(40):
    train(model3,train_loader,criterion,optimizer,epoch)
    validation(model3,valid_loader,epoch)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:36: UserWarning: Implicit dimensio

Train Step: 0        Train Loss: 2.317        Train Accuracy: 8.000
Validation step: 0        Validation  accuracy: 56.000
Train Step: 1000        Train Loss: 1.307        Train Accuracy: 48.000
Validation step: 1000        Validation  accuracy: 65.000
Train Step: 2000        Train Loss: 1.079        Train Accuracy: 60.000
Validation step: 2000        Validation  accuracy: 70.000
Train Step: 3000        Train Loss: 0.750        Train Accuracy: 78.000
Validation step: 3000        Validation  accuracy: 71.000
```

```
Train Step: 4000        Train Loss: 1.115        Train Accuracy: 66.000
Validation step: 4000        Validation  accuracy: 72.000
Train Step: 5000        Train Loss: 0.736        Train Accuracy: 72.000
Validation step: 5000        Validation  accuracy: 74.000
Train Step: 6000        Train Loss: 0.775        Train Accuracy: 70.000
Validation step: 6000        Validation  accuracy: 74.000
Train Step: 7000        Train Loss: 0.611        Train Accuracy: 74.000
Validation step: 7000        Validation  accuracy: 75.000
Train Step: 8000        Train Loss: 0.651        Train Accuracy: 72.000
Validation step: 8000        Validation  accuracy: 75.000
Train Step: 9000        Train Loss: 0.675        Train Accuracy: 76.000
Validation step: 9000        Validation  accuracy: 76.000
Train Step: 10000        Train Loss: 0.534        Train Accuracy: 84.000
Validation step: 10000        Validation  accuracy: 76.000
Train Step: 11000        Train Loss: 0.636        Train Accuracy: 68.000
Validation step: 11000        Validation  accuracy: 76.000
Train Step: 12000        Train Loss: 0.480        Train Accuracy: 86.000
Validation step: 12000        Validation  accuracy: 77.000
Train Step: 13000        Train Loss: 0.484        Train Accuracy: 82.000
Validation step: 13000        Validation  accuracy: 75.000
Train Step: 14000        Train Loss: 0.376        Train Accuracy: 90.000
Validation step: 14000        Validation  accuracy: 76.000
Train Step: 15000        Train Loss: 0.351        Train Accuracy: 92.000
Validation step: 15000        Validation  accuracy: 77.000
Train Step: 16000        Train Loss: 0.318        Train Accuracy: 86.000
Validation step: 16000        Validation  accuracy: 77.000
Train Step: 17000        Train Loss: 0.398        Train Accuracy: 84.000
Validation step: 17000        Validation  accuracy: 77.000
Train Step: 18000        Train Loss: 0.402        Train Accuracy: 88.000
Validation step: 18000        Validation  accuracy: 78.000
Train Step: 19000        Train Loss: 0.201        Train Accuracy: 94.000
Validation step: 19000        Validation  accuracy: 77.000
Train Step: 20000        Train Loss: 0.243        Train Accuracy: 96.000
Validation step: 20000        Validation  accuracy: 77.000
Train Step: 21000        Train Loss: 0.229        Train Accuracy: 92.000
Validation step: 21000        Validation  accuracy: 78.000
Train Step: 22000        Train Loss: 0.170        Train Accuracy: 94.000
Validation step: 22000        Validation  accuracy: 78.000
Train Step: 23000        Train Loss: 0.382        Train Accuracy: 84.000
Validation step: 23000        Validation  accuracy: 77.000
Train Step: 24000        Train Loss: 0.401        Train Accuracy: 88.000
Validation step: 24000        Validation  accuracy: 78.000
Train Step: 25000        Train Loss: 0.279        Train Accuracy: 86.000
Validation step: 25000        Validation  accuracy: 78.000
Train Step: 26000        Train Loss: 0.161        Train Accuracy: 94.000
Validation step: 26000        Validation  accuracy: 78.000
Train Step: 27000        Train Loss: 0.296        Train Accuracy: 92.000
Validation step: 27000        Validation  accuracy: 77.000
```

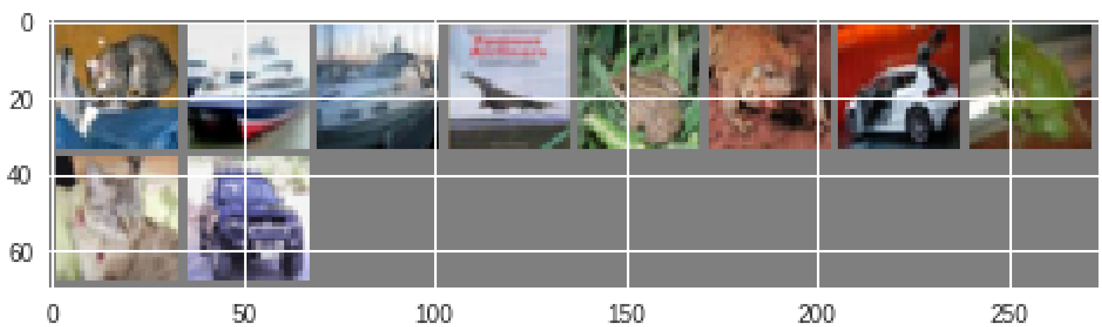
```
Train Step: 28000        Train Loss: 0.244        Train Accuracy: 94.000
Validation step: 28000        Validation  accuracy: 78.000
Train Step: 29000        Train Loss: 0.171        Train Accuracy: 92.000
Validation step: 29000        Validation  accuracy: 77.000
Train Step: 30000        Train Loss: 0.230        Train Accuracy: 94.000
Validation step: 30000        Validation  accuracy: 78.000
Train Step: 31000        Train Loss: 0.254        Train Accuracy: 88.000
Validation step: 31000        Validation  accuracy: 78.000
Train Step: 32000        Train Loss: 0.210        Train Accuracy: 96.000
Validation step: 32000        Validation  accuracy: 78.000
Train Step: 33000        Train Loss: 0.304        Train Accuracy: 92.000
Validation step: 33000        Validation  accuracy: 77.000
Train Step: 34000        Train Loss: 0.120        Train Accuracy: 96.000
Validation step: 34000        Validation  accuracy: 77.000
Train Step: 35000        Train Loss: 0.225        Train Accuracy: 94.000
Validation step: 35000        Validation  accuracy: 77.000
Train Step: 36000        Train Loss: 0.082        Train Accuracy: 96.000
Validation step: 36000        Validation  accuracy: 78.000
Train Step: 37000        Train Loss: 0.240        Train Accuracy: 90.000
Validation step: 37000        Validation  accuracy: 78.000
Train Step: 38000        Train Loss: 0.101        Train Accuracy: 96.000
Validation step: 38000        Validation  accuracy: 78.000
Train Step: 39000        Train Loss: 0.239        Train Accuracy: 90.000
Validation step: 39000        Validation  accuracy: 78.000
```

In [0]:
```
model3.eval()
correct = 0
for images, labels in testloader:
    with torch.no_grad(): # so that computation graph history is not stored
        images, labels = images.cuda(), labels.cuda() # send tensors to GPU
        outputs = model3(images)
        predictions = outputs.data.max(1)[1]
        correct += predictions.eq(labels.data).sum()

print('Test set accuracy: {:.2f}%'.format(100.0 * correct / len(testloader.dataset)))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:36: UserWarning: Implicit dimensi


Test set accuracy: 78.00%
```

In [0]:
```
import torchvision
import torchvision.transforms as transforms
testloader = torch.utils.data.DataLoader(testset, batch_size=10,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
```

```
                'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

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

# show images
imshow(torchvision.utils.make_grid(images))
images, labels = images.cuda(), labels.cuda() # send tensors to GPU
outputs = model3(images)
predictions_T = outputs.data.max(1)[1]
correct += predictions_T.eq(labels.data).sum()

# print labels
print("testlabels: " + ' '.join('%5s' % classes[labels.data[j]] for j in range(10)))
print("predlabels: " + ' '.join('%5s' % classes[predictions_T[j]] for j in range(10)))
```



```
testlabels:   cat  ship  ship plane  frog  frog   car  frog   cat   car
predlabels:   cat  ship  ship plane   cat  frog   car  frog   cat   car

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:36: UserWarning: Implicit dimensi
```

In [0]:
```
import torch.nn as nn
import torch.nn.functional as F
# Model3 adds batchNorm2d layer after ReLu activation in the first two convolutional la
# Model3 add fc3 after fc2 in fully-connected layers
# Model3 still incooperates dropout layers in both convolutional and fully-connected la
# Model3 has improved valid_acc and test_acc to 79%
# Model4 keeps kernel size at 5x5


class CIFAR10_4(nn.Module):
    def __init__(self):
        super(CIFAR10_4, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Dropout(p=0.2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Dropout(p=0.2))
        self.drop_out = nn.Dropout()
        self.fc1 = nn.Linear(64*8*8, 1024)
        self.fc2 = nn.Linear(1024, 100)
        self.fc3 = nn.Linear(100, 10)
    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = x.view(-1, 64*8*8) # reshape before sending to fc layer
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = F.relu(self.fc2(x))
        x = F.dropout(x, training=self.training) # default p=0.5
        x = self.fc3(x)
        return F.log_softmax(x)
model4 = CIFAR10_4()
```

In [0]:
```
print(model4)
```

```
CIFAR10_4(
  (layer1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
```

```
    (4): Dropout(p=0.2)
  )
  (layer2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.2)
  )
  (drop_out): Dropout(p=0.5)
  (fc1): Linear(in_features=4096, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=100, bias=True)
  (fc3): Linear(in_features=100, out_features=10, bias=True)
)
```

```
In [0]: import torch.nn.functional as F
        loss = F.nll_loss(outputs,labels)
```

```
In [0]: import torch.optim as optim

        criterion = nn.CrossEntropyLoss()
        optimizer = optim.SGD(model4.parameters(), lr=0.01, momentum=0.9)
```

```
In [0]: batch_size = 50
        # send model to GPU

        #train_loss, train_accu, valid_accu = [], [],[]

        for epoch in range(40):
            train(model4,train_loader,criterion,optimizer,epoch)
            validation(model4,valid_loader,epoch)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:34: UserWarning: Implicit dimensi


Train Step: 0        Train Loss: 2.322        Train Accuracy: 10.000
Validation step: 0        Validation  accuracy: 57.000
Train Step: 1000        Train Loss: 1.483        Train Accuracy: 56.000
Validation step: 1000        Validation  accuracy: 65.000
Train Step: 2000        Train Loss: 1.118        Train Accuracy: 62.000
Validation step: 2000        Validation  accuracy: 68.000
Train Step: 3000        Train Loss: 0.888        Train Accuracy: 68.000
Validation step: 3000        Validation  accuracy: 73.000
Train Step: 4000        Train Loss: 0.867        Train Accuracy: 74.000
Validation step: 4000        Validation  accuracy: 73.000
Train Step: 5000        Train Loss: 0.830        Train Accuracy: 76.000
Validation step: 5000        Validation  accuracy: 75.000
Train Step: 6000        Train Loss: 0.744        Train Accuracy: 66.000
```

```
Validation step: 6000        Validation  accuracy: 75.000
Train Step: 7000        Train Loss: 0.697        Train Accuracy: 78.000
Validation step: 7000        Validation  accuracy: 75.000
Train Step: 8000        Train Loss: 0.769        Train Accuracy: 70.000
Validation step: 8000        Validation  accuracy: 75.000
Train Step: 9000        Train Loss: 0.771        Train Accuracy: 80.000
Validation step: 9000        Validation  accuracy: 76.000
Train Step: 10000        Train Loss: 0.453        Train Accuracy: 88.000
Validation step: 10000        Validation  accuracy: 76.000
Train Step: 11000        Train Loss: 0.646        Train Accuracy: 76.000
Validation step: 11000        Validation  accuracy: 77.000
Train Step: 12000        Train Loss: 0.392        Train Accuracy: 84.000
Validation step: 12000        Validation  accuracy: 77.000
Train Step: 13000        Train Loss: 0.302        Train Accuracy: 90.000
Validation step: 13000        Validation  accuracy: 77.000
Train Step: 14000        Train Loss: 0.458        Train Accuracy: 80.000
Validation step: 14000        Validation  accuracy: 76.000
Train Step: 15000        Train Loss: 0.221        Train Accuracy: 90.000
Validation step: 15000        Validation  accuracy: 77.000
Train Step: 16000        Train Loss: 0.243        Train Accuracy: 92.000
Validation step: 16000        Validation  accuracy: 77.000
Train Step: 17000        Train Loss: 0.453        Train Accuracy: 88.000
Validation step: 17000        Validation  accuracy: 77.000
Train Step: 18000        Train Loss: 0.378        Train Accuracy: 84.000
Validation step: 18000        Validation  accuracy: 77.000
Train Step: 19000        Train Loss: 0.237        Train Accuracy: 90.000
Validation step: 19000        Validation  accuracy: 78.000
Train Step: 20000        Train Loss: 0.326        Train Accuracy: 92.000
Validation step: 20000        Validation  accuracy: 78.000
Train Step: 21000        Train Loss: 0.281        Train Accuracy: 94.000
Validation step: 21000        Validation  accuracy: 78.000
Train Step: 22000        Train Loss: 0.092        Train Accuracy: 98.000
Validation step: 22000        Validation  accuracy: 78.000
Train Step: 23000        Train Loss: 0.094        Train Accuracy: 96.000
Validation step: 23000        Validation  accuracy: 78.000
Train Step: 24000        Train Loss: 0.229        Train Accuracy: 94.000
Validation step: 24000        Validation  accuracy: 79.000
Train Step: 25000        Train Loss: 0.289        Train Accuracy: 88.000
Validation step: 25000        Validation  accuracy: 78.000
Train Step: 26000        Train Loss: 0.200        Train Accuracy: 92.000
Validation step: 26000        Validation  accuracy: 79.000
Train Step: 27000        Train Loss: 0.117        Train Accuracy: 96.000
Validation step: 27000        Validation  accuracy: 79.000
Train Step: 28000        Train Loss: 0.170        Train Accuracy: 94.000
Validation step: 28000        Validation  accuracy: 78.000
Train Step: 29000        Train Loss: 0.028        Train Accuracy: 100.000
Validation step: 29000        Validation  accuracy: 78.000
Train Step: 30000        Train Loss: 0.230        Train Accuracy: 92.000
```

```
Validation step: 30000       Validation  accuracy: 78.000
Train Step: 31000       Train Loss: 0.154       Train Accuracy: 92.000
Validation step: 31000       Validation  accuracy: 78.000
Train Step: 32000       Train Loss: 0.134       Train Accuracy: 96.000
Validation step: 32000       Validation  accuracy: 79.000
Train Step: 33000       Train Loss: 0.110       Train Accuracy: 94.000
Validation step: 33000       Validation  accuracy: 79.000
Train Step: 34000       Train Loss: 0.097       Train Accuracy: 98.000
Validation step: 34000       Validation  accuracy: 78.000
Train Step: 35000       Train Loss: 0.215       Train Accuracy: 94.000
Validation step: 35000       Validation  accuracy: 78.000
Train Step: 36000       Train Loss: 0.042       Train Accuracy: 100.000
Validation step: 36000       Validation  accuracy: 79.000
Train Step: 37000       Train Loss: 0.068       Train Accuracy: 96.000
Validation step: 37000       Validation  accuracy: 78.000
Train Step: 38000       Train Loss: 0.090       Train Accuracy: 96.000
Validation step: 38000       Validation  accuracy: 79.000
Train Step: 39000       Train Loss: 0.058       Train Accuracy: 98.000
Validation step: 39000       Validation  accuracy: 79.000
```

In [0]:
```
model4.eval()
correct = 0
for images, labels in testloader:
    with torch.no_grad(): # so that computation graph history is not stored
        images, labels = images.cuda(), labels.cuda() # send tensors to GPU
        outputs = model4(images)
        predictions = outputs.data.max(1)[1]
        correct += predictions.eq(labels.data).sum()

print('Test set accuracy: {:.2f}%'.format(100.0 * correct / len(testloader.dataset)))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:34: UserWarning: Implicit dimensi

Test set accuracy: 79.00%
```

In [ ]:
```
import torchvision
import torchvision.transforms as transforms
testloader = torch.utils.data.DataLoader(testset, batch_size=10,
                                         shuffle=False, num_workers=2)

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

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

```python
# 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(testloader)
images, labels = dataiter.next()

# show images
imshow(torchvision.utils.make_grid(images))
images, labels = images.cuda(), labels.cuda() # send tensors to GPU
outputs = model4(images)
predictions_T = outputs.data.max(1)[1]
correct += predictions_T.eq(labels.data).sum()

# print labels
print("testlabels: " + ' '.join('%5s' % classes[labels.data[j]] for j in range(10)))
print("predlabels: " + ' '.join('%5s' % classes[predictions_T[j]] for j in range(10)))
```

Disccusion for Part 2:

I start with my base line model of two convolutional layers: each of kernel size 5, the first layer has input channel 3 and output channel 32 and second layer has input channel 32 and ouput channel 64. Following each of these convolutional layers, I have maxpool layer of 2x2. After passing data through these two convolution layers, I then pass feature maps(tensors) to two fcs: fc1 (64x8x8, 1024), fc2(1024, 10). For fc1, there is relu activation while fc2 does not. After linear portion of fc2, data are passed through softmax layer. The cost function I chose was crossentropyloss and optimization I used was gradient descent . With this base line model cNN, I got 73% for validation and test accuracy. One problem I noticed was that the model was overfitted by training data during training. I had to stop training after 8000 iterations as training accuracy reach 100%.

The first thing I did to modify this model was to add a dropout layer after fc2. This was my model1. This obviously have controlled overfitting problem better. I was able to run more iterations. In the end, I was able to get 74% for validation accuracy and 75% for test accuracy after adding this dropout layer.

I also tried a number of other modifications: including change the kernel size, add more fcs, and add batchnorm layer and relu layer in convolutional layers of my model. Here are what I oberserved:

`modifications` `valid_acc` `test_acc`

model_b None 73% 73%

model1 dropout after fc1 (this dropout is included 75% 74% in model2,3,4)

model2 kernel 3x3, add relu and dropout 76% 76%
in convolutional layers

model3 kernel 3x3, add batch, relu and dropout 78% 78%
in convolutional layers, add fc3 after fc2 and dropout

model4 kernel 5x5(the same with model_b), 79% 79%
add batch, relu and dropout
in convolutional layers, add fc3 after fc2 and dropout

My conclusion: Dropout and batchnorm technique can improve generalization of CNN model and avoid overfitting of training data. In addition, altering architecture of CNN can also affect performance of CNN model. A relatively larger size of kernel seems to work better than smaller size kernel and it captures more information in feature maps.