In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import torch
from torch import autograd
import torch.nn.functional as F
import csv
import time

images = np.load("D:/work/JHUschoolStuff/machinelearning/project1/cs475_pr
oject_data/images.npy")
labels = np.load("D:/work/JHUschoolStuff/machinelearning/project1/cs475_pr
oject_data/labels.npy")
test = np.load("D:/work/JHUschoolStuff/machinelearning/project1/cs475_proj
ect_data/part_2_test_images.npy")
height = images.shape[1]
width = images.shape[2]
size = height * width
images = (images - images.mean()) / images.std()
data = images.reshape(images.shape[0],size)
data = torch.from_numpy(data).float()
labels = torch.from_numpy(labels).float()
test_data = test.reshape(test.shape[0], size)
test_data = (test_data - test_data.mean()) / test_data.std()
test_data = torch.from_numpy(test_data).float()
batch_size = 1
NUM_OPT_STEPS = 5000
train_seqs = data[0:45000,:]
train_labels = labels[0:45000]
val_seqs = data[45000:,:]
val_labels = labels[45000:]
NUM_CLASSES = 5
```

In [2]:
```
class TooSimpleConvNN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        # 3x3 convolution that takes in an image with one channel
        # and outputs an image with 8 channels.
        self.conv1 = torch.nn.Conv2d(1, 16, kernel_size=3, stride = 2)
        # 3x3 convolution that takes in an image with 8 channels
        # and outputs an image with 16 channels. The output image
        # has approximately half the height and half the width
        # because of the stride of 2.
        self.conv2 = torch.nn.Conv2d(16, 32, kernel_size=3, stride = 2)
        # 1x1 convolution that takes in an image with 16 channels and
        # produces an image with 5 channels. Here, the 5 channels
        # will correspond to class scores.
        self.final_conv = torch.nn.Conv2d(32, 5, kernel_size=1)
    def forward(self, x):
        # Convolutions work with images of shape
        # [batch_size, num_channels, height, width]
        x = x.view(-1, height, width).unsqueeze(1)
        x = F.relu(self.conv1(x))
```

```
        x = F.relu(self.conv2(x))
        n, c, h, w = x.size()
        x = F.avg_pool2d(x, kernel_size=[h, w])
        x = self.final_conv(x).view(-1, NUM_CLASSES)
        return x
```

In [3]:
```
def train(model, optimizer, batch_size):
    model.train()
    # i is is a 1-D array with shape [batch_size]
    i = np.random.choice(train_seqs.shape[0], size=batch_size, replace=Fal
se)
    i = torch.from_numpy(i).long()
    x = autograd.Variable(train_seqs[i, :])
    y = autograd.Variable(train_labels[i]).long()
    optimizer.zero_grad()
    y_hat_ = model(x)
    loss = F.cross_entropy(y_hat_, y)
    loss.backward()
    optimizer.step()
    return loss.data[0]
```

In [4]:
```
def approx_train_accuracy(model):
    i = np.random.choice(train_seqs.shape[0], size=1000, replace=False)
    i = torch.from_numpy(i).long()
    x = autograd.Variable(train_seqs[i, :])
    y = autograd.Variable(train_labels[i]).long()
    y_hat_ = model(x)
    y_hat = np.zeros(1000)
    for i in range(1000):
        y_hat[i] = torch.max(y_hat_[i,:].data, 0)[1][0]
    return accuracy(y_hat, y.data.numpy())
```

In [5]:
```
def val_accuracy(model):
    x = autograd.Variable(val_seqs)
    y = autograd.Variable(val_labels)
    y_hat_ = model(x)
    y_hat = np.zeros(5000)
    for i in range(5000):
        y_hat[i] = torch.max(y_hat_[i,:].data, 0)[1][0]
    return accuracy(y_hat, y.data.numpy())
```

In [6]:
```
def accuracy(y, y_hat):
    return (y == y_hat).astype(np.float).mean()
```

In [7]:
```
def plot(train_accs, val_accs):
    plt.figure(200)
    plt.title('Training Accuracy')
    plt.xlabel('Iteration')
    plt.ylabel('Accuracy')
    plt.plot(train_accs, 'b')
    plt.show()
    plt.figure(300)
    plt.title('Validation Accuracy')
    plt.xlabel('Iteration')
    plt.ylabel('Accuracy')
```

```
    plt.plot(val_accs, 'b')
    plt.show()
```

In [8]:
```
def runModel(model, batch_size, NUM_OPT_STEPS, optimizer):
    train_accs, val_accs = [], []
    for i in range(NUM_OPT_STEPS):
        train(model, optimizer, batch_size)
        if i % 100 == 0:
            train_accs.append(approx_train_accuracy(model))
            val_accs.append(val_accuracy(model))
            print("%6d %5.2f %5.2f" % (i, train_accs[-1], val_accs[-1]))
    plot(train_accs, val_accs)
```

In [20]:
```
model = TooSimpleConvNN()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

In [21]:
```
runModel(model, 32, 30000, optimizer)
```

```
     0  0.21  0.19
   100  0.52  0.50
   200  0.59  0.61
   300  0.65  0.65
   400  0.67  0.69
   500  0.66  0.70
   600  0.70  0.71
   700  0.70  0.70
   800  0.71  0.72
   900  0.71  0.74
  1000  0.73  0.74
  1100  0.70  0.73
  1200  0.74  0.76
  1300  0.73  0.75
  1400  0.75  0.75
  1500  0.73  0.75
  1600  0.77  0.78
  1700  0.75  0.77
  1800  0.75  0.77
  1900  0.76  0.76
  2000  0.77  0.77
  2100  0.75  0.78
  2200  0.78  0.77
  2300  0.77  0.77
  2400  0.75  0.78
  2500  0.77  0.79
  2600  0.77  0.78
  2700  0.78  0.78
  2800  0.79  0.78
  2900  0.77  0.79
  3000  0.77  0.80
  3100  0.79  0.79
  3200  0.77  0.79
  3300  0.80  0.80
  3400  0.81  0.80
  3500  0.79  0.81
  3600  0.80  0.81
  3700  0.82  0.80
```

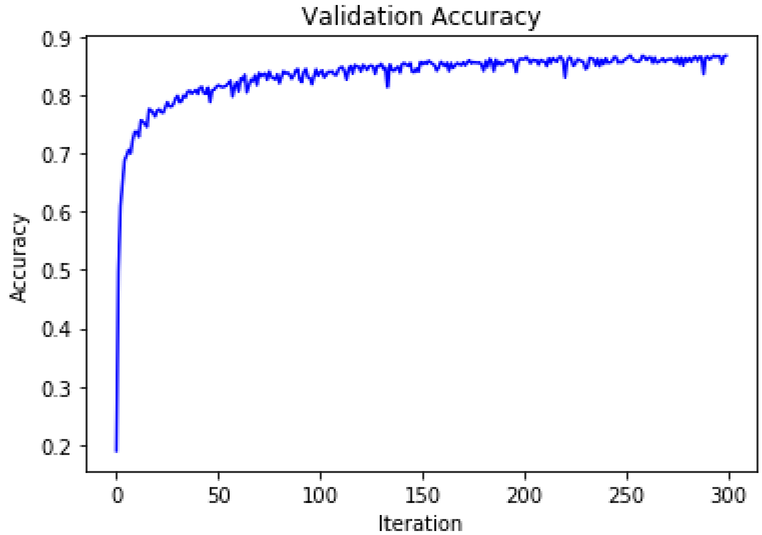
```
3800  0.80  0.81
3900  0.81  0.81
4000  0.81  0.80
4100  0.83  0.81
4200  0.78  0.81
4300  0.81  0.80
4400  0.80  0.80
4500  0.80  0.81
4600  0.80  0.79
4700  0.80  0.81
4800  0.81  0.81
4900  0.84  0.81
5000  0.79  0.82
5100  0.81  0.82
5200  0.80  0.81
5300  0.82  0.81
5400  0.81  0.82
5500  0.81  0.82
5600  0.83  0.82
5700  0.80  0.80
5800  0.81  0.81
5900  0.82  0.82
6000  0.81  0.81
6100  0.81  0.83
6200  0.83  0.83
6300  0.82  0.84
6400  0.79  0.81
6500  0.84  0.82
6600  0.80  0.83
6700  0.83  0.82
6800  0.85  0.83
6900  0.81  0.82
7000  0.85  0.84
7100  0.83  0.83
7200  0.84  0.84
7300  0.82  0.84
7400  0.83  0.83
7500  0.83  0.84
7600  0.83  0.83
7700  0.83  0.83
7800  0.81  0.83
7900  0.85  0.84
8000  0.79  0.82
8100  0.83  0.83
8200  0.82  0.84
8300  0.82  0.84
8400  0.84  0.84
8500  0.82  0.84
8600  0.83  0.83
8700  0.81  0.83
8800  0.83  0.84
8900  0.86  0.85
9000  0.83  0.83
9100  0.82  0.82
9200  0.85  0.84
9300  0.84  0.85
9400  0.84  0.83
```

```
 9500  0.84  0.83
 9600  0.83  0.82
 9700  0.85  0.83
 9800  0.85  0.84
 9900  0.84  0.83
10000  0.82  0.84
10100  0.83  0.84
10200  0.83  0.83
10300  0.82  0.83
10400  0.84  0.84
10500  0.84  0.84
10600  0.85  0.84
10700  0.82  0.84
10800  0.84  0.83
10900  0.85  0.84
11000  0.85  0.85
11100  0.83  0.85
11200  0.84  0.84
11300  0.84  0.83
11400  0.85  0.85
11500  0.85  0.85
11600  0.84  0.84
11700  0.85  0.85
11800  0.85  0.85
11900  0.85  0.85
12000  0.84  0.84
12100  0.87  0.85
12200  0.85  0.85
12300  0.84  0.84
12400  0.85  0.85
12500  0.86  0.85
12600  0.84  0.85
12700  0.83  0.84
12800  0.85  0.85
12900  0.84  0.85
13000  0.84  0.85
13100  0.84  0.85
13200  0.84  0.85
13300  0.83  0.81
13400  0.85  0.85
13500  0.84  0.84
13600  0.82  0.84
13700  0.83  0.85
13800  0.85  0.85
13900  0.82  0.84
14000  0.86  0.85
14100  0.85  0.86
14200  0.85  0.85
14300  0.86  0.85
14400  0.87  0.86
14500  0.84  0.84
14600  0.85  0.84
14700  0.85  0.85
14800  0.82  0.84
14900  0.84  0.86
15000  0.86  0.85
15100  0.86  0.86
```

```
15200  0.86  0.85
15300  0.85  0.86
15400  0.87  0.86
15500  0.87  0.85
15600  0.84  0.85
15700  0.84  0.84
15800  0.87  0.85
15900  0.84  0.86
16000  0.84  0.85
16100  0.84  0.85
16200  0.85  0.86
16300  0.85  0.84
16400  0.87  0.86
16500  0.85  0.85
16600  0.85  0.85
16700  0.87  0.86
16800  0.86  0.85
16900  0.87  0.85
17000  0.85  0.86
17100  0.83  0.85
17200  0.87  0.86
17300  0.85  0.86
17400  0.85  0.85
17500  0.85  0.86
17600  0.85  0.86
17700  0.86  0.86
17800  0.86  0.86
17900  0.84  0.85
18000  0.85  0.84
18100  0.88  0.86
18200  0.85  0.85
18300  0.88  0.86
18400  0.86  0.86
18500  0.87  0.84
18600  0.88  0.86
18700  0.87  0.85
18800  0.86  0.86
18900  0.87  0.85
19000  0.86  0.85
19100  0.85  0.86
19200  0.88  0.86
19300  0.86  0.86
19400  0.85  0.86
19500  0.87  0.85
19600  0.84  0.84
19700  0.85  0.86
19800  0.84  0.86
19900  0.84  0.86
20000  0.87  0.86
20100  0.88  0.87
20200  0.88  0.86
20300  0.88  0.85
20400  0.88  0.86
20500  0.86  0.86
20600  0.87  0.86
20700  0.87  0.85
20800  0.86  0.86
```

20900	0.87	0.86
21000	0.83	0.86
21100	0.88	0.85
21200	0.84	0.86
21300	0.87	0.86
21400	0.86	0.86
21500	0.86	0.86
21600	0.83	0.86
21700	0.87	0.87
21800	0.88	0.87
21900	0.86	0.86
22000	0.82	0.83
22100	0.86	0.86
22200	0.88	0.87
22300	0.87	0.86
22400	0.85	0.85
22500	0.88	0.85
22600	0.87	0.86
22700	0.86	0.86
22800	0.85	0.86
22900	0.86	0.86
23000	0.88	0.85
23100	0.86	0.85
23200	0.86	0.86
23300	0.87	0.86
23400	0.88	0.86
23500	0.85	0.85
23600	0.89	0.86
23700	0.86	0.85
23800	0.86	0.86
23900	0.87	0.85
24000	0.87	0.86
24100	0.87	0.86
24200	0.86	0.86
24300	0.88	0.86
24400	0.86	0.87
24500	0.87	0.87
24600	0.87	0.86
24700	0.86	0.86
24800	0.85	0.86
24900	0.84	0.86
25000	0.88	0.86
25100	0.86	0.87
25200	0.89	0.87
25300	0.86	0.86
25400	0.86	0.86
25500	0.88	0.86
25600	0.83	0.86
25700	0.86	0.86
25800	0.88	0.87
25900	0.87	0.87
26000	0.85	0.86
26100	0.87	0.86
26200	0.89	0.86
26300	0.85	0.86
26400	0.88	0.86
26500	0.86	0.86

```
26600  0.88  0.86
26700  0.86  0.86
26800  0.87  0.86
26900  0.86  0.86
27000  0.85  0.86
27100  0.89  0.86
27200  0.89  0.86
27300  0.86  0.86
27400  0.88  0.86
27500  0.88  0.87
27600  0.86  0.85
27700  0.87  0.86
27800  0.88  0.85
27900  0.88  0.86
28000  0.88  0.86
28100  0.88  0.86
28200  0.87  0.86
28300  0.90  0.86
28400  0.87  0.87
28500  0.84  0.86
28600  0.87  0.87
28700  0.89  0.87
28800  0.85  0.84
28900  0.86  0.86
29000  0.90  0.87
29100  0.85  0.86
29200  0.88  0.86
29300  0.89  0.87
29400  0.88  0.87
29500  0.88  0.87
29600  0.87  0.87
29700  0.87  0.85
29800  0.89  0.87
29900  0.89  0.87
```



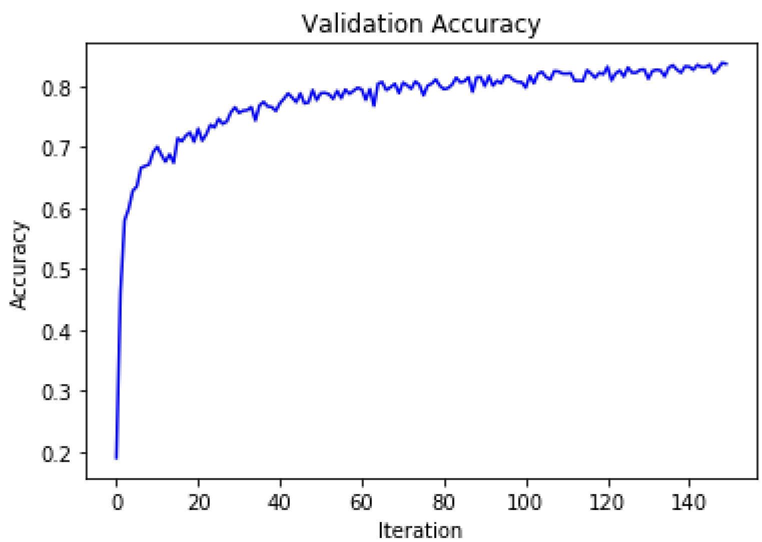
In [9]:
```
model = TooSimpleConvNN()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
runModel(model, 32, 15000, optimizer) #using stride = 1
```

```
   0  0.19  0.19
 100  0.45  0.46
 200  0.56  0.58
 300  0.56  0.60
 400  0.60  0.63
 500  0.62  0.64
 600  0.65  0.67
 700  0.67  0.67
 800  0.68  0.67
 900  0.66  0.69
1000  0.70  0.70
1100  0.69  0.69
1200  0.69  0.68
1300  0.68  0.69
1400  0.67  0.67
1500  0.73  0.71
1600  0.72  0.71
1700  0.72  0.72
1800  0.74  0.72
1900  0.69  0.71
2000  0.72  0.73
2100  0.68  0.71
2200  0.69  0.72
2300  0.74  0.74
2400  0.75  0.73
2500  0.73  0.75
2600  0.75  0.74
2700  0.73  0.74
2800  0.75  0.76
2900  0.76  0.77
3000  0.75  0.76
3100  0.79  0.76
3200  0.77  0.76
3300  0.77  0.77
3400  0.76  0.74
```

```
3500  0.76  0.77
3600  0.79  0.77
3700  0.76  0.77
3800  0.77  0.77
3900  0.77  0.76
4000  0.79  0.77
4100  0.79  0.78
4200  0.79  0.79
4300  0.79  0.78
4400  0.75  0.77
4500  0.76  0.79
4600  0.78  0.77
4700  0.79  0.77
4800  0.79  0.79
4900  0.79  0.78
5000  0.79  0.79
5100  0.78  0.79
5200  0.79  0.79
5300  0.78  0.78
5400  0.81  0.79
5500  0.78  0.78
5600  0.80  0.79
5700  0.80  0.79
5800  0.81  0.79
5900  0.80  0.80
6000  0.81  0.79
6100  0.76  0.78
6200  0.79  0.79
6300  0.76  0.77
6400  0.78  0.80
6500  0.82  0.81
6600  0.81  0.79
6700  0.81  0.80
6800  0.80  0.80
6900  0.79  0.79
7000  0.81  0.81
7100  0.82  0.80
7200  0.80  0.80
7300  0.82  0.81
7400  0.79  0.80
7500  0.79  0.78
7600  0.80  0.80
7700  0.81  0.80
7800  0.82  0.81
7900  0.79  0.80
8000  0.78  0.79
8100  0.81  0.80
8200  0.79  0.80
8300  0.81  0.81
8400  0.80  0.81
8500  0.82  0.81
8600  0.81  0.81
8700  0.79  0.79
8800  0.83  0.81
8900  0.83  0.81
9000  0.80  0.80
9100  0.83  0.82
```

```
 9200  0.81  0.80
 9300  0.82  0.81
 9400  0.81  0.80
 9500  0.83  0.82
 9600  0.80  0.82
 9700  0.82  0.81
 9800  0.81  0.81
 9900  0.81  0.81
10000  0.79  0.80
10100  0.80  0.82
10200  0.79  0.80
10300  0.82  0.82
10400  0.82  0.82
10500  0.80  0.81
10600  0.80  0.81
10700  0.83  0.82
10800  0.81  0.82
10900  0.86  0.82
11000  0.83  0.82
11100  0.85  0.82
11200  0.82  0.81
11300  0.83  0.81
11400  0.82  0.81
11500  0.85  0.83
11600  0.80  0.82
11700  0.83  0.81
11800  0.83  0.82
11900  0.84  0.82
12000  0.85  0.83
12100  0.81  0.81
12200  0.81  0.82
12300  0.83  0.83
12400  0.83  0.82
12500  0.82  0.83
12600  0.81  0.82
12700  0.82  0.82
12800  0.83  0.83
12900  0.81  0.83
13000  0.83  0.81
13100  0.83  0.82
13200  0.82  0.83
13300  0.82  0.83
13400  0.81  0.82
13500  0.85  0.83
13600  0.83  0.83
13700  0.82  0.83
13800  0.82  0.82
13900  0.84  0.83
14000  0.84  0.83
14100  0.85  0.83
14200  0.84  0.83
14300  0.83  0.83
14400  0.86  0.83
14500  0.84  0.83
14600  0.85  0.82
14700  0.84  0.83
14800  0.85  0.84
```

```
14900  0.85  0.84
```



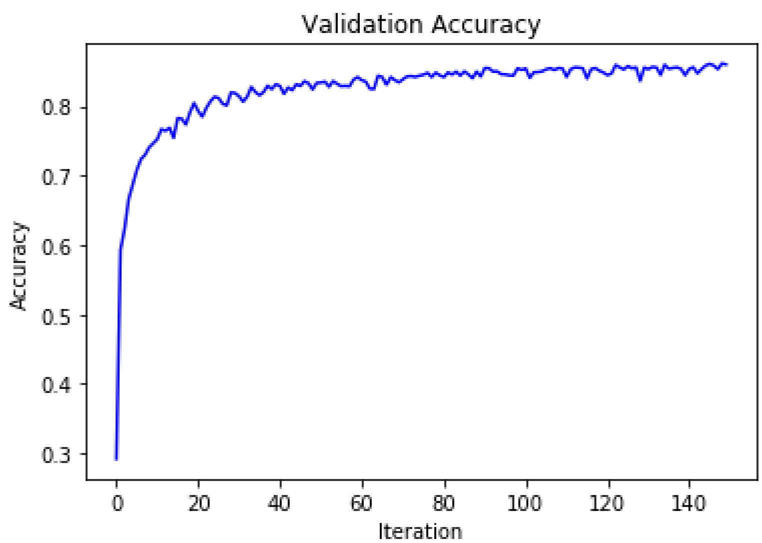
In [23]:
```
model = TooSimpleConvNN()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
runModel(model, 64, 15000, optimizer) #using stride = 2
```

```
   0  0.27  0.29
 100  0.59  0.59
 200  0.61  0.62
 300  0.65  0.67
 400  0.69  0.69
 500  0.69  0.71
 600  0.71  0.72
 700  0.71  0.73
 800  0.72  0.74
 900  0.76  0.75
1000  0.73  0.75
1100  0.75  0.77
1200  0.74  0.76
1300  0.74  0.77
1400  0.76  0.75
1500  0.78  0.78
1600  0.77  0.78
```

```
1700  0.78  0.77
1800  0.81  0.79
1900  0.79  0.80
2000  0.80  0.79
2100  0.79  0.78
2200  0.80  0.80
2300  0.80  0.81
2400  0.79  0.81
2500  0.81  0.81
2600  0.79  0.80
2700  0.79  0.80
2800  0.79  0.82
2900  0.81  0.82
3000  0.82  0.81
3100  0.80  0.81
3200  0.79  0.81
3300  0.80  0.83
3400  0.81  0.82
3500  0.83  0.82
3600  0.83  0.82
3700  0.82  0.83
3800  0.83  0.82
3900  0.83  0.83
4000  0.82  0.83
4100  0.80  0.82
4200  0.83  0.83
4300  0.83  0.82
4400  0.83  0.83
4500  0.83  0.83
4600  0.80  0.84
4700  0.82  0.83
4800  0.81  0.82
4900  0.83  0.83
5000  0.82  0.83
5100  0.86  0.84
5200  0.81  0.83
5300  0.85  0.84
5400  0.83  0.83
5500  0.85  0.83
5600  0.85  0.83
5700  0.83  0.83
5800  0.84  0.84
5900  0.83  0.84
6000  0.83  0.84
6100  0.82  0.84
6200  0.83  0.83
6300  0.82  0.82
6400  0.85  0.84
6500  0.84  0.84
6600  0.84  0.83
6700  0.83  0.84
6800  0.84  0.84
6900  0.81  0.83
7000  0.84  0.84
7100  0.84  0.84
7200  0.82  0.84
7300  0.84  0.84
```

```
 7400  0.82  0.84
 7500  0.84  0.85
 7600  0.85  0.85
 7700  0.85  0.84
 7800  0.85  0.85
 7900  0.86  0.84
 8000  0.83  0.84
 8100  0.84  0.85
 8200  0.85  0.85
 8300  0.86  0.85
 8400  0.84  0.84
 8500  0.85  0.85
 8600  0.86  0.85
 8700  0.84  0.84
 8800  0.85  0.85
 8900  0.83  0.84
 9000  0.85  0.85
 9100  0.84  0.85
 9200  0.84  0.85
 9300  0.86  0.85
 9400  0.85  0.85
 9500  0.87  0.85
 9600  0.84  0.84
 9700  0.83  0.84
 9800  0.86  0.85
 9900  0.86  0.85
10000  0.86  0.85
10100  0.85  0.84
10200  0.87  0.85
10300  0.87  0.85
10400  0.85  0.85
10500  0.87  0.85
10600  0.86  0.85
10700  0.86  0.85
10800  0.86  0.85
10900  0.86  0.85
11000  0.83  0.84
11100  0.87  0.85
11200  0.84  0.86
11300  0.86  0.86
11400  0.86  0.85
11500  0.85  0.84
11600  0.86  0.85
11700  0.86  0.85
11800  0.86  0.85
11900  0.87  0.85
12000  0.85  0.84
12100  0.85  0.85
12200  0.85  0.86
12300  0.86  0.86
12400  0.86  0.85
12500  0.84  0.86
12600  0.87  0.85
12700  0.85  0.86
12800  0.84  0.84
12900  0.85  0.86
13000  0.87  0.85
```

```
13100  0.85  0.86
13200  0.87  0.86
13300  0.87  0.85
13400  0.87  0.86
13500  0.86  0.85
13600  0.88  0.86
13700  0.88  0.86
13800  0.86  0.85
13900  0.84  0.84
14000  0.86  0.85
14100  0.87  0.86
14200  0.85  0.85
14300  0.87  0.85
14400  0.86  0.86
14500  0.86  0.86
14600  0.87  0.86
14700  0.84  0.85
14800  0.88  0.86
14900  0.88  0.86
```



The best validation accuracy I achieved after changing the stride to 2 was 86. I used a batch size of 64, 15000 optimization steps, and Adam as my optimizer with a learning rate of 0.001. My training and validation accuracies were about the same after running them for 15000 steps, however increasing the steps and batch size seems to give me a much higher training accuracy than validation accuracy which suggests that I had begun to overfit my training data. To increase performance further, possibly more convolutional layers may help me detect more complex features and give me a better accuracy. Increasing the channels may also help increase the accuracy of my predictions. I could also add in max pooling between the convolution layers to help with down sampling and reducing computational cost, which in turn will help me reduce overfitting.