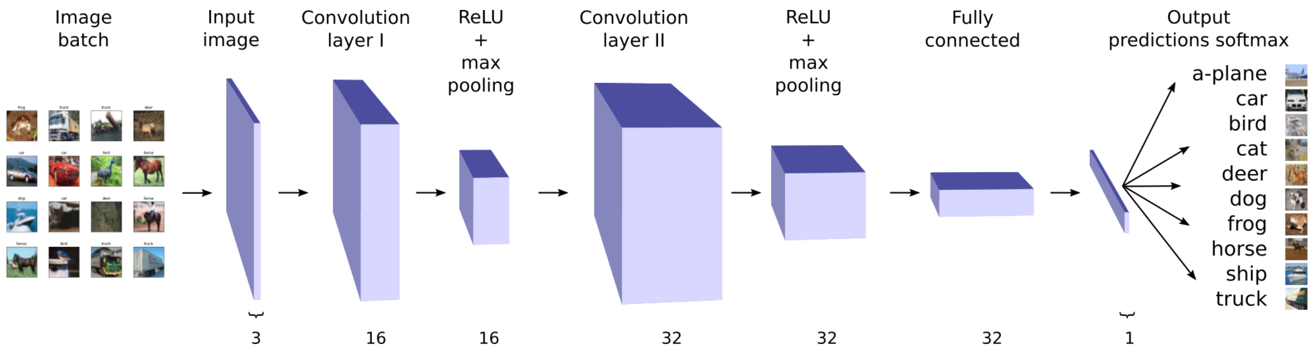
Convolutional Neural Networks Exercise: Image Classification

Paul J. Atzberger
http://atzberger.org/

We demonstrate here setup of a convolutional neural networks for image classification using the CIFAR10 and MNIST datasets [5-9]. The MNIST images are of handwritten digits and have a single gray-scale channel and are 28x28 pixels. The CIFAR10 images are of common objects (cars, trucks, cats, dogs, etc...) and have 3 RGB color channels and are 32x32 pixels. For processing, the datasets are represented respectively as tensors with dimensions Nx1x28x28 and Nx3x32x32. For the CIFAR10 image database we formulate a convolutional neural network with the following architecture, and a similar network for MNIST.



To demonstrate concepts, we use a basic convolutional neural network consisting of 2 layers with operations of Convolution -> ReLU -> Max-Pool. The first layer has a 16 channel convolutional layer with a kernel of size 5 pixels, a ReLU non-linearity, and a max pooling with kernel size of 2 units. The second layer has a 32 channel convolutional layer with a kernel of size 5 pixels, a ReLU non-linearity, and max pooling with kernel size of 2 units. This produces $8 * 8 * 32 = 2048$ features that we process with a fully-connected layer to use in a softmax assignment of probabilities to the classification categories. We train the model by minimizing the cross-entropy loss function.

In this exercise, you are to modify the architecture and hyper-parameters to get familiar with how convolutional neural networks work and to try to improve their ability to perform this classification task. You are welcome to experiment with other CNN architectures by stacking additional convolutional layers or by adjusting the width, kernel size, and depth of filters or by performing data augmentation or by making other modifications. For inspiration, see the papers below for discussions of both historic and modern techniques [1-4].

References

[1] *Hand written digit recognition with a back-propagation network*, Y. LeCun, In Proc. Advances in Neural Information Processing Systems, (1990).

[2] *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*, K. Fukushima, Biological Cybernetics, 36, 4, April (1980).

[3] *Receptive fields of single neurons in the cat's striate cortex*, Hubel, D.H.; Wiesel, T.N., J Physiol. 148 (3), (1959).

[4] *Deep learning*, LeCun, Y., Bengio, Y., and Hinton, G., Nature, 521, May, (2015).

[5] *Pytorch-tutorials*, https://pytorch.org/tutorials/ (https://pytorch.org/tutorials/), 2017.

[6] *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*, https://arxiv.org/abs/1603.04467 (https://arxiv.org/abs/1603.04467), 2015.

[7] CIFAR10 Dataset: Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

[8] Images CIFAR10 database: https://www.cs.toronto.edu/~kriz/cifar.html (https://www.cs.toronto.edu/~kriz/cifar.html)

[9] Images of MNIST database: http://yann.lecun.com/exdb/mnist/index.html (http://yann.lecun.com/exdb/mnist/index.html)

Setup Codes

In [1]:

```
import torch;
import torch.nn as nn;
import torchvision;
import torchvision.transforms as transforms;

import numpy as np;
import matplotlib.pyplot as plt;
import matplotlib.gridspec as gridspec

# Configure devices
if torch.cuda.is_available():
  device = torch.device('cuda:0');
else:
  device = torch.device('cpu');

# Setup the parameters
learningRate = 0.001;
batchSize = 100;
numCategories = 10;

numEpochs = 5;

flagDataSet = 'MNIST';
#flagDataSet = 'CIFAR10';

print("torch.__version__ = " + str(torch.__version__));
print("numpy.__version__ = " + str(np.__version__));

print("device = " + str(device));

```

```
torch.__version__ = 0.4.1
numpy.__version__ = 1.14.0
device = cuda:0
```

Function Definitions

In [2]:

```
def plot_images_in_array(axs,img_arr,label_arr=None,cmap=None):

    numSamples = len(img_arr);
    sqrtS      = int(np.sqrt(numSamples));

    I = 0;
    for i in range(0,sqrtS):
        for j in range(0,sqrtS):
            img = img_arr[I];

            if cmap is not None:
              axs[i][j].imshow(img, cmap=cmap);
            else:
              axs[i][j].imshow(img);
            if label_arr is not None:
              axs[i][j].set_title("%s"%label_arr[I]);
            axs[i][j].set_xticks([]);
            axs[i][j].set_yticks([]);
            I += 1;

```

In [3]:

```
def plot_image_array(img_arr,label_arr=None,title=None,figSize=(18,18),tit

    # determine number of images we need to plot
    numSamples = len(img_arr);
    sqrtS      = int(np.sqrt(numSamples));
    rows       = sqrtS;
    cols       = sqrtS;

    fig, axs = plt.subplots(nrows=rows, ncols=cols, figsize=figSize);

    plot_images_in_array(axs,img_arr,label_arr,cmap=cmap);

    if title is None:
      plt.suptitle("Collection of Images", fontsize=18,y=title_yp);
    else:
      plt.suptitle(title, fontsize=18,y=title_yp);

```

In [4]:

```python
def plot_gridspec_image_array(outer_h,img_arr,label_arr=None,title=None,fi

    fig = plt.gcf();

    sqrtS = int(np.sqrt(len(img_arr)));
    #print(sqrtS);

    inner = gridspec.GridSpecFromSubplotSpec(sqrtS, sqrtS,
                                             subplot_spec=outer_h, wspace=

    # collect the axes
    axs =[];
    I = 0;
    for i in range(0,sqrtS):
      axs_r = [];
      for j in range(0,sqrtS):
        ax = plt.Subplot(fig, inner[I]);
        fig.add_subplot(ax);
        axs_r.append(ax);
        I += 1;

      axs.append(axs_r);

    #print(axs);

    # plot the images
    plot_images_in_array(axs,img_arr,cmap="gray");

    if title is None:
      a = 1;
    else:
      axs[0][0].text(title_x,title_y,title,fontsize=title_fsize);
      #plt.suptitle(title, fontsize=18,y=title_yp);


```

Load the dataset

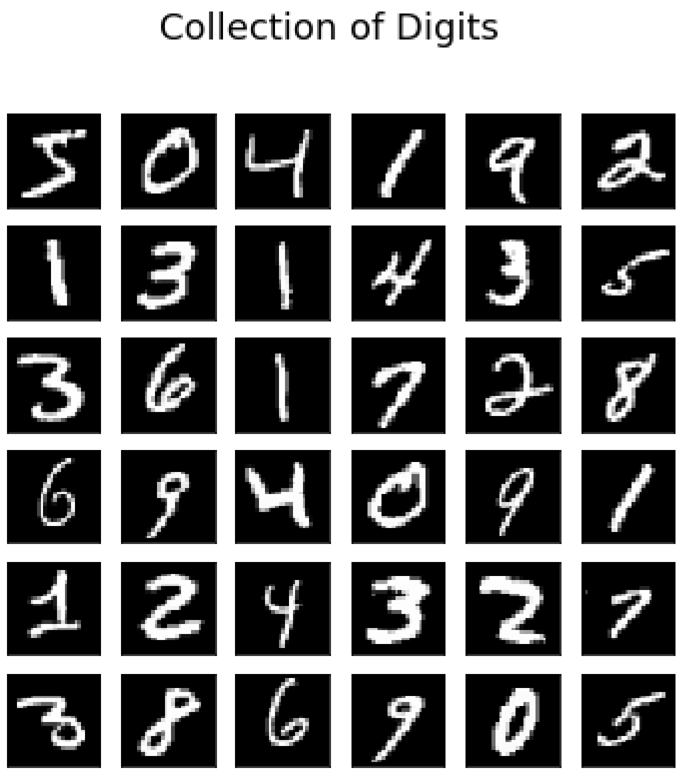
In [5]:

```
if flagDataSet == 'MNIST':
  # MNIST dataset
  train_dataset = torchvision.datasets.MNIST(root='./data/',
                                             train=True,
                                             transform=transforms.ToTensor
                                             download=True);

  test_dataset = torchvision.datasets.MNIST(root='./data/',
                                            train=False,
                                            transform=transforms.ToTensor(

  categoryNames = ['zero','one','two','three','four','five','six','seven',

elif flagDataSet == 'CIFAR10':
  # CIFAR10 dataset
  train_dataset = torchvision.datasets.CIFAR10(root='./data/',
                                             train=True,
                                             transform=transforms.ToTensor
                                             download=True);

  test_dataset = torchvision.datasets.CIFAR10(root='./data/',
                                            train=False,
                                            transform=transforms.ToTensor(

  #categoryNames = ['airplane','automobile','bird','cat','deer','dog','fro
  categoryNames = ['a-plane','car','bird','cat','deer','dog','frog','horse

else:
  print("flagDataSet not recognized.");
  print("flagDataSet = " + str(flagDataSet));
  raise;

# Data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batchSize,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batchSize,
                                          shuffle=False)

```

Show Subset of the Data

In [6]:

```
# Show subset of the data
img_arr = [];
label_str_arr = [];

numImages = len(train_dataset);
#II = np.random.permutation(numImages); # compute random collection of ind
II = np.arange(numImages);

if flagDataSet == 'MNIST':
  for I in np.arange(0,36):
    img_arr.append(np.array(train_dataset[II[I]][0][0]));
  plot_image_array(img_arr,title='Collection of Digits',figSize=(6,6),titl
elif flagDataSet == 'CIFAR10':
  for I in np.arange(0,36):
    img = np.array(train_dataset[II[I]][0]);
    img = np.transpose(img,(1,2,0));
    img_arr.append(img);
    l_true_str = categoryNames[train_dataset[II[I]][1]];
    label_str_arr.append(l_true_str);
  plot_image_array(img_arr,label_str_arr,title='CIFAR10 Images',figSize=(9
  plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace

```



Setup the Convolutional Neural Network (CNN)

In [7]:

```
# Convolutional neural network (two convolutional layers)
class ConvNet(nn.Module):
    def __init__(self, numCategories=10):
        super(ConvNet, self).__init__()
        if flagDataSet == 'MNIST':   # assumed images are given as 1x28x28
          self.layer1 = nn.Sequential(
              nn.Conv2d(in_channels=1, out_channels=16, kernel_size=5, str
              nn.BatchNorm2d(num_features=16),
              nn.ReLU(),
              nn.MaxPool2d(kernel_size=2, stride=2));
          self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, stri
            nn.BatchNorm2d(num_features=32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2));
          self.fc = nn.Linear(7*7*32, numCategories);
        elif flagDataSet == 'CIFAR10': # assumed images are given as 3x32x
          self.layer1 = nn.Sequential(
              nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5, str
              nn.BatchNorm2d(num_features=16),
              nn.ReLU(),
              nn.MaxPool2d(kernel_size=2, stride=2));
          self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, stri
            nn.BatchNorm2d(num_features=32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2));
          self.fc = nn.Linear(8*8*32, numCategories);

    def forward(self, x):
        out = self.layer1(x);
        out = self.layer2(out);
        out = out.reshape(out.size(0), -1);
        out = self.fc(out);
        return out;

model = ConvNet(numCategories).to(device);

```

Perform Training of the Neural Network

In [8]:

```
# setup the optimization method and loss function
optimizer = torch.optim.Adam(model.parameters(), lr=learningRate);
loss_func = nn.CrossEntropyLoss();

print("Training the CNN with:");
print("numEpochs = %d"%numEpochs);
print("batchSize = %d"%batchSize);
print(" ");

# Train the model
numSteps = len(train_loader);
for epoch in range(numEpochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device);
        labels = labels.to(device);

        # Forward pass
        outputs = model(images);
        loss = loss_func(outputs, labels);

        # Backward and optimize
        optimizer.zero_grad();
        loss.backward();
        optimizer.step();

        if ((i + 1) % 100) == 0:
          print ('Epoch: [%d/%d]; batchStep = [%d/%d]; Loss: %.4f.'%(epoch

```

```
Training the CNN with:
numEpochs = 5
batchSize = 100

Epoch: [1/5]; batchStep = [100/600]; Loss: 0.1731.
Epoch: [1/5]; batchStep = [200/600]; Loss: 0.0862.
Epoch: [1/5]; batchStep = [300/600]; Loss: 0.1007.
Epoch: [1/5]; batchStep = [400/600]; Loss: 0.0911.
Epoch: [1/5]; batchStep = [500/600]; Loss: 0.0698.
Epoch: [1/5]; batchStep = [600/600]; Loss: 0.0369.
Epoch: [2/5]; batchStep = [100/600]; Loss: 0.0530.
Epoch: [2/5]; batchStep = [200/600]; Loss: 0.0643.
Epoch: [2/5]; batchStep = [300/600]; Loss: 0.0089.
Epoch: [2/5]; batchStep = [400/600]; Loss: 0.0277.
Epoch: [2/5]; batchStep = [500/600]; Loss: 0.0757.
Epoch: [2/5]; batchStep = [600/600]; Loss: 0.0345.
Epoch: [3/5]; batchStep = [100/600]; Loss: 0.0322.
Epoch: [3/5]; batchStep = [200/600]; Loss: 0.0097.
Epoch: [3/5]; batchStep = [300/600]; Loss: 0.1274.
Epoch: [3/5]; batchStep = [400/600]; Loss: 0.0849.
Epoch: [3/5]; batchStep = [500/600]; Loss: 0.0462.
Epoch: [3/5]; batchStep = [600/600]; Loss: 0.0234.
Epoch: [4/5]; batchStep = [100/600]; Loss: 0.0477.
Epoch: [4/5]; batchStep = [200/600]; Loss: 0.0202.
Epoch: [4/5]; batchStep = [300/600]; Loss: 0.0041.
Epoch: [4/5]; batchStep = [400/600]; Loss: 0.0182.
```

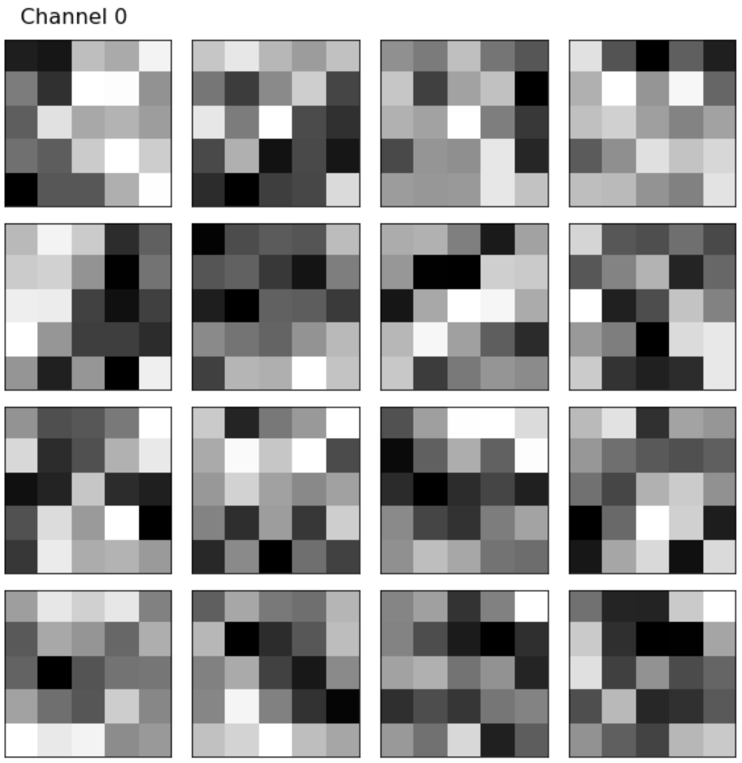
```
Epoch: [4/5]; batchStep = [500/600]; Loss: 0.0169.
Epoch: [4/5]; batchStep = [600/600]; Loss: 0.0643.
Epoch: [5/5]; batchStep = [100/600]; Loss: 0.0375.
Epoch: [5/5]; batchStep = [200/600]; Loss: 0.0204.
Epoch: [5/5]; batchStep = [300/600]; Loss: 0.0109.
Epoch: [5/5]; batchStep = [400/600]; Loss: 0.0201.
Epoch: [5/5]; batchStep = [500/600]; Loss: 0.0017.
Epoch: [5/5]; batchStep = [600/600]; Loss: 0.0040.
```

Show Convolution Layers

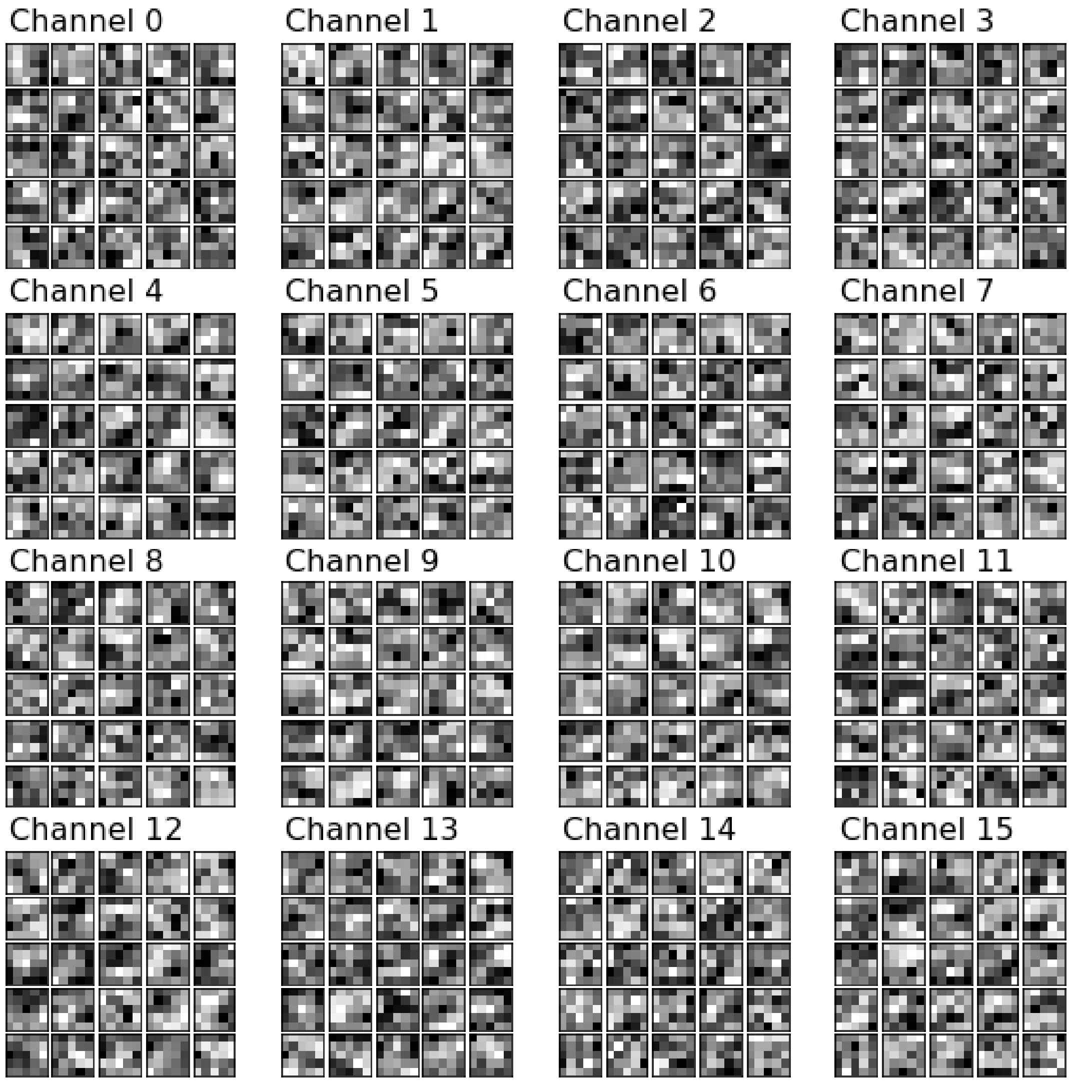
In [9]:

```python
#print(model.state_dict().keys());
state = model.state_dict();

fig = plt.figure(figsize=(10, 10))
w = state['layer1.0.weight'];
wp = np.transpose(w,(1,0,2,3));
numChannels = wp.shape[0];
layerNumber = 1;
numCols = 1;
outer = gridspec.GridSpec(numChannels, numChannels);
if flagDataSet=='MNIST':
  title_y = -1.0;
else:
  title_y = -2.0;
for d in range(0,numChannels):
  plot_gridspec_image_array(outer[d],img_arr=wp[d],label_arr=None,
                            title='Channel %d'%d,title_x=0.0,title_y=title
                            title_fsize=16,cmap="gray");

plt.suptitle("Convolution Layer 1", fontsize=18,y=0.95);

fig = plt.figure(figsize=(10, 10))
w = state['layer2.0.weight'];
wp = np.transpose(w,(1,0,2,3));
numChannels = wp.shape[0];
layerNumber = 2;
numCols = int(np.sqrt(numChannels));
outer = gridspec.GridSpec(numCols,numCols);
for d in range(0,numChannels):
  plot_gridspec_image_array(outer[d],img_arr=wp[d],label_arr=None,
                            title='Channel %d'%d,title_x=0.0,title_y=-2.0,
                            title_fsize=16,cmap="gray");

plt.suptitle("Convolution Layer 2", fontsize=18,y=0.95);

```

Convolution Layer 1



Convolution Layer 2



Test the Neural Network Predictions

In [10]:

```
# Test the model
print("Testing predictions of the neural network:");
print("");

# Save the first few to show as examples of labeling
saved_test_img = [];
saved_test_label_true = [];
saved_test_label_pred = [];

model.eval();  # eval mode (batchnorm uses moving mean/variance instead of
with torch.no_grad():
    correct = 0;
    total = 0;
    II = 0;
    for images, labels in test_loader: # loads data in batches and then su

        if (II >= 1000):
          print("tested on %d images"%total);
          II = 0;

        images = images.to(device);
        labels = labels.to(device);
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1);
        total += labels.size(0);
        correct += (predicted == labels).sum().item();

        #print(I);
        #print(images.shape);
        #print(labels.shape);
        II += labels.size(0);

        img = np.transpose(images[0],(1,2,0)); # sample just first image o
        saved_test_img.append(img);
        saved_test_label_true.append(labels[0].item());
        saved_test_label_pred.append(predicted[0].item());

    print("");
    print("Tested on a total of %d images."%total);
    print("");
    test_accuracy = correct/total;

    print("The neural network has an accurary of %.2f%% on the %d test ima

```

```
Testing predictions of the neural network:

tested on 1000 images
tested on 2000 images
tested on 3000 images
tested on 4000 images
tested on 5000 images
tested on 6000 images
tested on 7000 images
tested on 8000 images
tested on 9000 images
```

```
Tested on a total of 10000 images.

The neural network has an accuracy of 98.89% on the 10000 test im
ages.
```

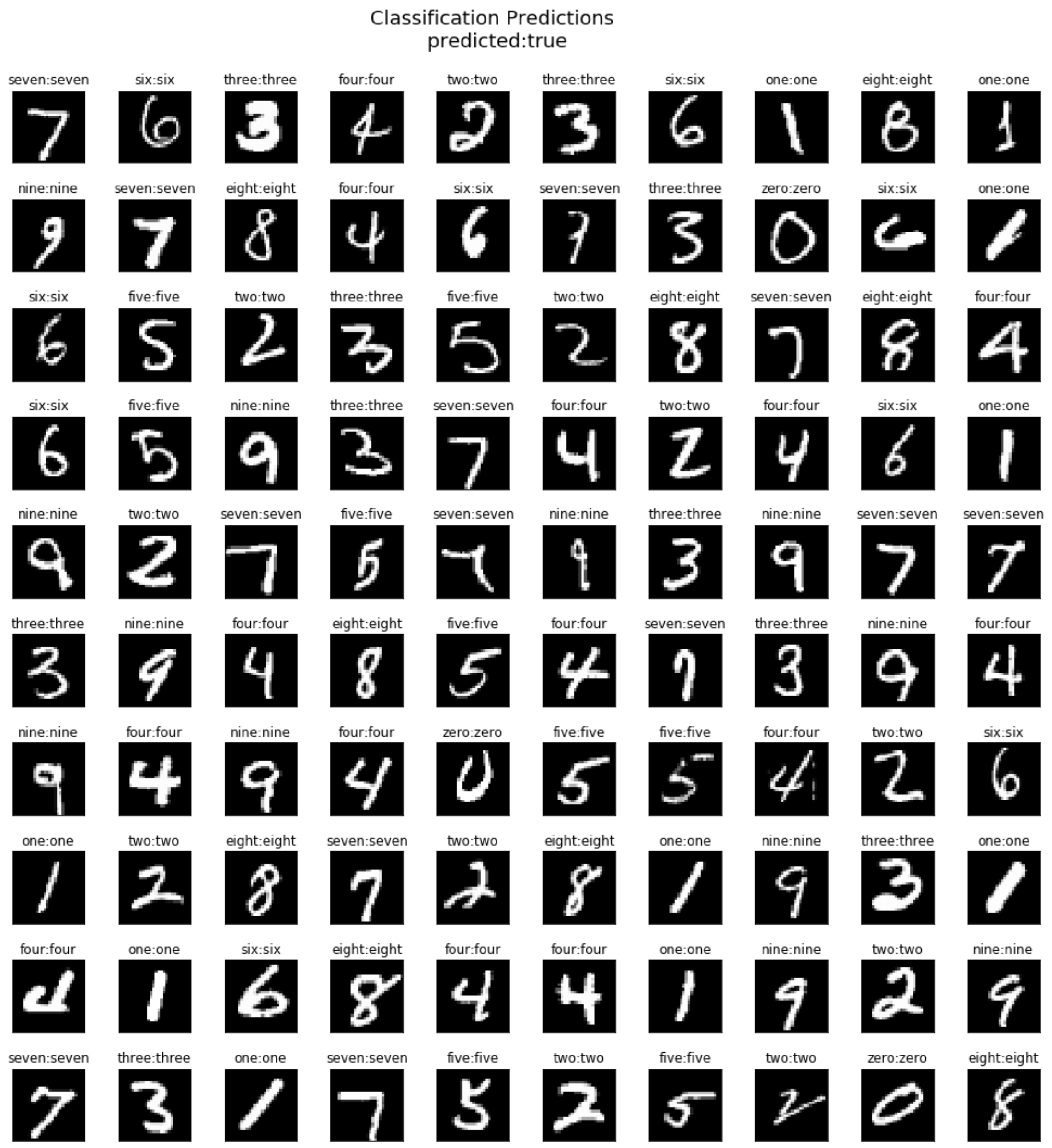
Show a Sample of the Predictions

In [11]:

```
# collect a subset of the data to show and attach named labels
numSamplesShow = 100;
ss_label_true = saved_test_label_true[0:numSamplesShow];
ss_label_pred = saved_test_label_pred[0:numSamplesShow];

# setup the labels
ss_label_str  = [];
numLabels = len(saved_test_label_true);
for i in range(0,numLabels):
  l_true = saved_test_label_true[i];
  l_pred = saved_test_label_pred[i];
  l_pred_str = categoryNames[l_pred];
  l_true_str = categoryNames[l_true];
  #sstr = "%s:%s"%(l_pred,l_true);
  sstr = "%s:%s"%(l_pred_str,l_true_str);
  ss_label_str.append(sstr);

if flagDataSet == 'MNIST':
  ss_img = [];
  for I in np.arange(0,numSamplesShow):
    #print(saved_test_img[I].shape);
    ss = np.transpose(np.array(saved_test_img[I]),(2,0,1));
    img = ss[0];
    #print(img.shape)
    ss_img.append(img);

elif flagDataSet == 'CIFAR10':
  ss_img   = saved_test_img[0:numSamplesShow];

```

In [12]:

```
plot_image_array(ss_img,ss_label_str,title="Classification Predictions \n
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0
```



Save the Model

In [13]:

```
# Write the weights to disk for possible later re-use.
# For instance, one could use a well-trained neural network and
# try to transfer the learned features for use in other tasks.
torch.save(model.state_dict(), 'cnn_trained.ckpt')
```

Summary

We have demonstrated how to formulate and train a basic CNN for image classication using the MNIST image dataset. This basic CNN used for demonstrating concepts can be improved by further development and adjustments. Experiment with the CNN architecture and hyperparameters. For instance, by stacking additional convolutional layers or by adjusting the width, kernel size, and depth of filters or by performing data augmentation or by making other modifications. For inspiration, see the references above for discussions of both historic and modern techniques [1-4].