MLproject_ImageClassification

May 31, 2018

1 Monkey Species Classification

USING CONVOLUTIONAL NEURAL NETWORKS
Statistics and Machine Learning | 23rd April 22, 2018
TEAM
Shubham Anjankar
Venkata Avinash Paturu
Tanmay Sandav
Apoorv Upadhyaya
Sanat SN
Geet Kamat

OVERVIEW
The general idea of this project is to develop a convolutional neural network model to classify monkeys based on their species.
Dataset consists of several images of monkeys from different species.
Model identifies features that are similar as well as distinct in all the species.
Classification is done on the basis of distinct features pertaining to species.

Project dependencies

```
In [1]: import os
        import math
        from glob import glob
        import matplotlib.pyplot as plt
        import random
        import cv2
        import collections
        import pandas as pd
        import numpy as np
        import skimage
        from skimage.transform import resize
        import matplotlib.gridspec as gridspec
        import tensorflow as tf
        from keras.preprocessing.image import ImageDataGenerator
        from sklearn.metrics import confusion_matrix
        import time
        from datetime import timedelta
        import math
        import os
        import prettytensor as pt
        %matplotlib inline
```

```
/Users/avinash2don/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:34: FutureWarning: Co
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

2 UNDER THE HOOD

This project is loosely based on image recognition works by Andrej Karpathy, Fei-Fei Li and Matthew Zeiler, Rob Fergus.

Karpathy, Andrej, and Li Fei-Fei. "Deep Visual-Semantic Alignments for Generating Image Descriptions." 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, doi:10.1109/cvpr.2015.7298932.

Zeiler, Matthew D., and Rob Fergus. "Visualizing and Understanding Convolutional Networks." Computer Vision âĂŞ ECCV 2014 Lecture Notes in Computer Science, 2014, pp. 818–833., doi:10.1007/978-3-319-10590-1_53. -We identify common features in the monkeys namely eyes, fur, ears, nose, mouth etc.
-We then compare how distinct these features are among species.
-Conventional facial recognition techniques are limited to just identifying these features. If the algorithm detects all of these features, it will label it as a face.
-This algorithm operates more on a macro level. It will identify the features and then compare the difference of these features with respect to other features.
-Based on the differences (or similarity), it will classify the monkey in the image to be in the species where the differences(or similarity) in features match the most.

3 Dataset

```
In [2]: cols = ['Label','Latin Name', 'Common Name','Train Images', 'Validation Images']
        labels = pd.read_csv("./10-monkey-species/monkey_labels.txt", names=cols, skiprows=1)
        labels
```

```
Out[2]:    Label                  Latin Name                      Common Name  \
        0  n0          alouatta_palliata\t    mantled_howler
        1  n1         erythrocebus_patas\t    patas_monkey
        2  n2         cacajao_calvus\t        bald_uakari
        3  n3         macaca_fuscata\t        japanese_macaque
        4  n4        cebuella_pygmea\t        pygmy_marmoset
        5  n5        cebus_capucinus\t        white_headed_capuchin
        6  n6        mico_argentatus\t        silvery_marmoset
        7  n7       saimiri_sciureus\t        common_squirrel_monkey
```

```
8  n8       aotus_nigriceps\t        black_headed_night_monkey
9  n9       trachypithecus_johnii    nilgiri_langur

   Train Images  Validation Images
0           131                 26
1           139                 28
2           137                 27
3           152                 30
4           131                 26
5           141                 28
6           132                 26
7           142                 28
8           133                 27
9           132                 26
```

```
In [3]: labels = labels['Common Name']
        labels.shape
        print(labels)
```

```
0     mantled_howler
1     patas_monkey
2     bald_uakari
3     japanese_macaque
4     pygmy_marmoset
5     white_headed_capuchin
6     silvery_marmoset
7     common_squirrel_monkey
8     black_headed_night_monkey
9     nilgiri_langur
Name: Common Name, dtype: object
```

```
In [4]: train_dir = "./10-monkey-species/training/"
        test_dir =  "./10-monkey-species/validation/"
        from tqdm import tqdm
        def get_data(folder):
            X = []
            y = []
            for folderName in os.listdir(folder):
                if not folderName.startswith('.'):
                    if folderName in ['n0']:
                        label = 0
                    elif folderName in ['n1']:
                        label = 1
                    elif folderName in ['n2']:
                        label = 2
                    elif folderName in ['n3']:
                        label = 3
                    elif folderName in ['n4']:
                        label = 4
                    elif folderName in ['n5']:
                        label = 5
                    elif folderName in ['n6']:
                        label = 6
                    elif folderName in ['n7']:
                        label = 7
                    elif folderName in ['n8']:
                        label = 8
                    elif folderName in ['n9']:
                        label = 9
                    else:
                        label = 10
                    for image_filename in tqdm(os.listdir(folder + folderName)):
                        img_file = cv2.imread(folder + folderName + '/' + image_filename)
                        if img_file is not None:
                            img_file = skimage.transform.resize(img_file, (250, 250, 3))
                            img_arr = np.asarray(img_file)
                            X.append(img_arr)
                            y.append(label)
            X = np.asarray(X)
            y = np.asarray(y)
            return X,y
        x_train, y_train = get_data(train_dir)
        x_val, y_val= get_data(test_dir)
        from sklearn.cross_validation import train_test_split
        x_train, x_test, y_train, y_test = train_test_split(x_train, y_train,stratify=y_train,
```

```
100%|| 105/105 [00:08<00:00, 12.98it/s]
100%|| 105/105 [00:10<00:00, 10.39it/s]
100%|| 122/122 [00:11<00:00, 10.27it/s]
100%|| 106/106 [00:13<00:00,  7.75it/s]
100%|| 110/110 [00:09<00:00, 12.20it/s]
100%|| 111/111 [00:05<00:00, 19.47it/s]
100%|| 114/114 [00:07<00:00, 15.85it/s]
100%|| 106/106 [00:18<00:00,  5.77it/s]
100%|| 113/113 [00:09<00:00, 11.73it/s]
100%|| 106/106 [00:12<00:00,  8.19it/s]
100%|| 26/26 [00:00<00:00, 26.57it/s]
100%|| 26/26 [00:03<00:00,  7.04it/s]
100%|| 30/30 [00:06<00:00,  4.73it/s]
100%|| 26/26 [00:01<00:00, 16.94it/s]
100%|| 27/27 [00:02<00:00,  9.25it/s]
100%|| 28/28 [00:01<00:00, 24.12it/s]
100%|| 28/28 [00:02<00:00,  9.58it/s]
100%|| 26/26 [00:05<00:00,  5.03it/s]
100%|| 28/28 [00:03<00:00,  7.92it/s]
```

```
100%|| 27/27 [00:01<00:00, 13.81it/s]
/home/pv_avi99/.local/lib/python3.5/site-packages/sklearn/cross_validation.py:41: DeprecationWa
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
In [5]: from keras.utils.np_utils import to_categorical
        labels_train = to_categorical(y_train, num_classes = 10)
        labels_test = to_categorical(y_test, num_classes = 10)
        labels_val = to_categorical(y_val, num_classes = 10)
```

```
In [6]: print(x_train.shape)
        print(y_train.shape)
        print(labels_train.shape)
        print(x_val.shape)
        print(y_val.shape)
        print(labels_val.shape)
        print(x_test.shape)
        print(y_test.shape)
        print(labels_test.shape)
```

```
(988, 250, 250, 3)
(988,)
(988, 10)
(272, 250, 250, 3)
(272,)
(272, 10)
(110, 250, 250, 3)
(110,)
(110, 10)
```

4 Network architecture

4.1 Hyperparameters

```
In [7]: # Convolutional Layer 1.
        filter_size1 = 3
        num_filters1 = 128

        # Convolutional Layer 2.
        filter_size2 = 3
        num_filters2 = 64

        # Convolutional Layer 3.
        filter_size3 = 3
        num_filters3 = 64

        # Fully-connected layer.
```

```
fc_size = 512

# Number of color channels for the images
num_channels = 3

# image dimensions
img_size = 250

# Size of image when flattened to a single dimension
img_size_flat = img_size * img_size * num_channels

# Tuple with height and width of images used to reshape arrays.
img_shape = (img_size, img_size)

# class info
num_classes = 10

# batch size
batch_size = 68

# how long to wait after validation loss stops improving before terminating training
early_stopping = None
```

4.2 Helper functions for CNN and Fully connected layers

```
In [9]: def plot_images(images, cls_true, cls_pred=None):

            if len(images) == 0:
                print("no images to show")
                return
            else:
                random_indices = random.sample(range(len(images)), min(len(images), 9))


            images, cls_true  = zip(*[(images[i], cls_true[i]) for i in random_indices])

            fig, axes = plt.subplots(3, 3)
            fig.subplots_adjust(hspace=0.3, wspace=0.3)

            for i, ax in enumerate(axes.flat):
                ax.imshow(images[i].reshape(img_size, img_size, num_channels))
                cls_true_name = labels[cls_true[i]]

                if cls_pred is None:
                    xlabel = "True: {0}".format(cls_true_name)
                else:
                    cls_pred_name = labels[cls_pred[i]]
                    xlabel = "True: {0}, Pred: {1}".format(cls_true_name, cls_pred_name)
                ax.set_xlabel(xlabel)

                ax.set_xticks([])
                ax.set_yticks([])
            plt.show()
```

```
In [10]: # Get some random images and their labels from the train set.

         images, cls_true  = x_train, y_train

         # Plot the images and labels using our helper-function above.
         plot_images(images=images, cls_true=cls_true)
```

```
In [11]: def new_weights(shape):
             return tf.Variable(tf.truncated_normal(shape, stddev=0.05))

In [12]: def new_biases(length):
             return tf.Variable(tf.constant(0.05, shape=[length]))

In [13]: def new_conv_layer(input,
                            num_input_channels,
                            filter_size,
                            num_filters,
                            use_pooling=True):

             shape = [filter_size, filter_size, num_input_channels, num_filters]
```

```
              weights = new_weights(shape=shape)

              biases = new_biases(length=num_filters)

              # strides are set to 1 in all dimensions.
              layer = tf.nn.conv2d(input=input,
                                   filter=weights,
                                   strides=[1, 1, 1, 1],
                                   padding='SAME')

              layer += biases

              if use_pooling:
                  # 2x2 max-pooling
                  layer = tf.nn.max_pool(value=layer,
                                         ksize=[1, 2, 2, 1],
                                         strides=[1, 2, 2, 1],
                                         padding='SAME')

              # Rectified Linear Unit (ReLU).
              layer = tf.nn.relu(layer)

              return layer, weights
```

```
In [14]: def flatten_layer(layer):
              layer_shape = layer.get_shape()
              num_features = layer_shape[1:4].num_elements()
              layer_flat = tf.reshape(layer, [-1, num_features])
              return layer_flat, num_features
```

```
In [15]: def new_fc_layer(input,
                          num_inputs,
                          num_outputs,
                          use_relu=True):

              weights = new_weights(shape=[num_inputs, num_outputs])
              biases = new_biases(length=num_outputs)
              layer = tf.matmul(input, weights) + biases

              if use_relu:
                  layer = tf.nn.relu(layer)

              return layer
```

5 Tensorflow implementation

We are using tensorflow to construct this model

In a nutshell the graph computes various mathemetical operations by defining the functions and corresponding variables befor hand and inputting the data with placeholder variables. Then we also define varaibles that will be used to optimize the model, In this case our one-hot encoded labels.

```
In [16]: x = tf.placeholder(tf.float32, shape=[None, img_size_flat], name='x')
         x_image = tf.reshape(x, [-1, img_size, img_size, num_channels])
         y_true = tf.placeholder(tf.float32, shape=[None, num_classes], name='y_true')
         y_true_cls = tf.argmax(y_true, dimension=1)
```

The connvolutional step takes in an images and constructs filters which will be used to identify monkey features. A 2X2 maxpooling is used to down sample the image to identify macro features.

This is one of the key steps which make or break the problem, A small filter will help us to identify micro features but then again larger filter will help us to identenfy larger features. Its depends on the dataset to identify right size of the filter to identify the features which matters with out loosing data.

At the end we connect a fully connected layer to identify the many to many relationship betwee these macro features which will help us to classify the monkeys correctly.

```
In [19]: layer_conv1, weights_conv1 = \
             new_conv_layer(input=x_image,
                            num_input_channels=num_channels,
                            filter_size=filter_size1,
                            num_filters=num_filters1,
                            use_pooling=True)
         layer_conv2, weights_conv2 = \
             new_conv_layer(input=layer_conv1,
                            num_input_channels=num_filters1,
                            filter_size=filter_size2,
                            num_filters=num_filters2,
                            use_pooling=True)
         layer_conv3, weights_conv3 = \
             new_conv_layer(input=layer_conv2,
                            num_input_channels=num_filters2,
                            filter_size=filter_size3,
                            num_filters=num_filters3,
                            use_pooling=True)

         layer_flat, num_features = flatten_layer(layer_conv3)

         layer_fc1 = new_fc_layer(input=layer_flat,
                                  num_inputs=num_features,
                                  num_outputs=fc_size,
                                  use_relu=True)
         layer_fc2 = new_fc_layer(input=layer_fc1,
                                  num_inputs=fc_size,
                                  num_outputs=fc_size,
                                  use_relu=True)
```

```
layer_fc3 = new_fc_layer(input=layer_fc2,
                         num_inputs=fc_size,
                         num_outputs=num_classes,
                         use_relu=False)
```

The final Fully connected layer fires up our 10 label neurons, as we are using RELU we need to use softmax to get the probability of these 10 labels to work with the optimizer. The neuron with highes probability is selected as our predicted class Cross entropy is used as the cost function Cost function that can be used to guide the optimization of the variables. ADAM optimizer is used to updates the variables

```
In [20]: y_pred = tf.nn.softmax(layer_fc3)
         y_pred_cls = tf.argmax(y_pred, dimension=1)
         cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=layer_fc3,
                                                                 labels=y_true)
         cost = tf.reduce_mean(cross_entropy)

         optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(cost)

         correct_prediction = tf.equal(y_pred_cls, y_true_cls)

         accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

5.1 Tensor flow session

```
In [21]: session = tf.Session()
```

5.2 Initializing the variables

```
In [22]: session.run(tf.global_variables_initializer())
```

```
In [23]: train_batch_size = batch_size
```

Helper functions to print progress and select a random batch for calculations so that we dont exhaust the memory.

```
In [24]: def print_progress(epoch, feed_dict_train, feed_dict_validate, val_loss):
             acc = session.run(accuracy, feed_dict=feed_dict_train)
             val_acc = session.run(accuracy, feed_dict=feed_dict_validate)
             msg = "Epoch {0} --- Training Accuracy: {1:>6.1%}, Validation Loss: {3:.3f}"
             print(msg.format(epoch + 1, acc, val_acc, val_loss))

         def random_batch(x_train, y_train):
             num_images = len(x_train)
             idx = np.random.choice(num_images,
                                    size=train_batch_size,
                                    replace=False)
             x_batch = x_train[idx, :, :, :]
             y_batch = labels_train[idx, :]

             return x_batch, y_batch
```

5.3 Optimization Step

```
In [25]: total_iterations = 0

         def optimize(num_iterations):
             global total_iterations
             start_time = time.time()

             best_val_loss = float("inf")
             patience = 0

             for i in range(total_iterations,
                            total_iterations + num_iterations):

                 # random batch of training validation samples.
                 x_batch, y_true_batch = random_batch(x_train, labels_train)
                 x_valid_batch, y_valid_batch =random_batch( x_val, labels_val)

                 # reshaping the 4D array to 1D array

                 x_batch = x_batch.reshape(train_batch_size, img_size_flat)
                 x_valid_batch = x_valid_batch.reshape(train_batch_size, img_size_flat)

                 feed_dict_train = {x: x_batch,
                                    y_true: y_true_batch}

                 feed_dict_validate = {x: x_valid_batch,
                                       y_true: y_valid_batch}

                 # Optimizer using the random batch
                 session.run(optimizer, feed_dict=feed_dict_train)


                 # Print status at end of each epoch.
                 if i % int(len(x_train)/batch_size) == 0:
                     val_loss = session.run(cost, feed_dict=feed_dict_validate)
                     epoch = int(i / int(len(x_train)/batch_size))

                     print_progress(epoch, feed_dict_train, feed_dict_validate, val_loss)

                     if early_stopping:
                         if val_loss < best_val_loss:
                             best_val_loss = val_loss
                             patience = 0
                         else:
                             patience += 1

                         if patience == early_stopping:
```

```
                    break

            total_iterations += num_iterations

            end_time = time.time()

            time_dif = end_time - start_time

            print("Time elapsed: " + str(timedelta(seconds=int(round(time_dif)))))
```

6 Visualizing the results

```
In [26]: def plot_example_errors(cls_pred, correct):
             incorrect = (correct == False)
             # incorrectly classified.
             images = x_val[incorrect]

             # Get the predicted classes for those images.
             cls_pred = cls_pred[incorrect]

             # Get the true classes for those images.
             cls_true = y_val[incorrect]

             # Plot the first 9 images.
             plot_images(images=images[0:9],
                         cls_true=cls_true[0:9],
                         cls_pred=cls_pred[0:9])
```

```
In [27]: def plot_confusion_matrix(cls_pred):
             cls_true = y_val

             cm = confusion_matrix(y_true=cls_true,
                                   y_pred=cls_pred)
             print(cm)
             plt.matshow(cm)
             plt.colorbar()
             tick_marks = np.arange(num_classes)
             plt.xticks(tick_marks, range(num_classes))
             plt.yticks(tick_marks, range(num_classes))
             plt.xlabel('Predicted')
             plt.ylabel('True')
             plt.show()
```

```
In [28]: def print_validation_accuracy(show_example_errors=False,
                                       show_confusion_matrix=False):

             num_test = len(x_val)
             cls_pred = np.zeros(shape=num_test, dtype=np.int)
```

```
    i = 0

    while i < num_test:
        j = min(i + batch_size, num_test)

        images = x_val[i:j, :, :, :].reshape(batch_size, img_size_flat)

        labels = labels_val[i:j, :]

        feed_dict = {x: images,
                     y_true: labels}

        cls_pred[i:j] = session.run(y_pred_cls, feed_dict=feed_dict)

        i = j


    cls_true = np.array(y_val)

    correct = (cls_true == cls_pred)

    correct_sum = correct.sum()

    acc = float(correct_sum) / num_test


    msg = "Accuracy on Test-Set: {0:.1%} ({1} / {2})"
    print(msg.format(acc, correct_sum, num_test))


    if show_example_errors:
        print("Example errors:")
        plot_example_errors(cls_pred=cls_pred, correct=correct)

    if show_confusion_matrix:
        print("Confusion Matrix:")
        plot_confusion_matrix(cls_pred=cls_pred)
```

```
In [29]: len(x_val)
```

```
Out[29]: 272
```

```
In [30]: optimize(num_iterations=2000)
```

```
Epoch 1 --- Training Accuracy:  16.2%, Validation Loss: 2.466
Epoch 2 --- Training Accuracy:  35.3%, Validation Loss: 2.410
Epoch 3 --- Training Accuracy:  52.9%, Validation Loss: 2.795
Epoch 4 --- Training Accuracy:  52.9%, Validation Loss: 2.899
Epoch 5 --- Training Accuracy:  51.5%, Validation Loss: 3.915
Epoch 6 --- Training Accuracy:  55.9%, Validation Loss: 3.786
Epoch 7 --- Training Accuracy:  72.1%, Validation Loss: 3.728
Epoch 8 --- Training Accuracy:  79.4%, Validation Loss: 3.817
Epoch 9 --- Training Accuracy:  76.5%, Validation Loss: 4.423
Epoch 10 --- Training Accuracy:  82.4%, Validation Loss: 4.031
Epoch 11 --- Training Accuracy:  82.4%, Validation Loss: 4.227
Epoch 12 --- Training Accuracy:  79.4%, Validation Loss: 5.483
Epoch 13 --- Training Accuracy:  88.2%, Validation Loss: 5.745
Epoch 14 --- Training Accuracy:  92.6%, Validation Loss: 4.871
Epoch 15 --- Training Accuracy:  98.5%, Validation Loss: 6.279
Epoch 16 --- Training Accuracy: 100.0%, Validation Loss: 5.633
Epoch 17 --- Training Accuracy: 100.0%, Validation Loss: 6.468
Epoch 18 --- Training Accuracy:  95.6%, Validation Loss: 7.171
Epoch 19 --- Training Accuracy: 100.0%, Validation Loss: 7.095
Epoch 20 --- Training Accuracy:  97.1%, Validation Loss: 7.320
Epoch 21 --- Training Accuracy: 100.0%, Validation Loss: 6.797
Epoch 22 --- Training Accuracy: 100.0%, Validation Loss: 8.044
Epoch 23 --- Training Accuracy: 100.0%, Validation Loss: 7.095
Epoch 24 --- Training Accuracy: 100.0%, Validation Loss: 8.303
Epoch 25 --- Training Accuracy: 100.0%, Validation Loss: 7.579
Epoch 26 --- Training Accuracy: 100.0%, Validation Loss: 9.128
Epoch 27 --- Training Accuracy: 100.0%, Validation Loss: 8.667
Epoch 28 --- Training Accuracy: 100.0%, Validation Loss: 8.554
Epoch 29 --- Training Accuracy: 100.0%, Validation Loss: 8.339
Epoch 30 --- Training Accuracy: 100.0%, Validation Loss: 8.001
Epoch 31 --- Training Accuracy: 100.0%, Validation Loss: 8.514
Epoch 32 --- Training Accuracy: 100.0%, Validation Loss: 9.335
Epoch 33 --- Training Accuracy: 100.0%, Validation Loss: 9.350
Epoch 34 --- Training Accuracy: 100.0%, Validation Loss: 8.915
Epoch 35 --- Training Accuracy: 100.0%, Validation Loss: 9.503
Epoch 36 --- Training Accuracy: 100.0%, Validation Loss: 9.070
Epoch 37 --- Training Accuracy: 100.0%, Validation Loss: 10.841
Epoch 38 --- Training Accuracy: 100.0%, Validation Loss: 10.986
Epoch 39 --- Training Accuracy: 100.0%, Validation Loss: 11.397
Epoch 40 --- Training Accuracy: 100.0%, Validation Loss: 9.687
Epoch 41 --- Training Accuracy: 100.0%, Validation Loss: 10.300
Epoch 42 --- Training Accuracy: 100.0%, Validation Loss: 11.271
Epoch 43 --- Training Accuracy: 100.0%, Validation Loss: 10.220
Epoch 44 --- Training Accuracy: 100.0%, Validation Loss: 9.502
Epoch 45 --- Training Accuracy: 100.0%, Validation Loss: 8.815
Epoch 46 --- Training Accuracy: 100.0%, Validation Loss: 9.527
Epoch 47 --- Training Accuracy: 100.0%, Validation Loss: 13.269
Epoch 48 --- Training Accuracy: 100.0%, Validation Loss: 12.911
Epoch 49 --- Training Accuracy: 100.0%, Validation Loss: 10.384
Epoch 50 --- Training Accuracy: 100.0%, Validation Loss: 10.656
Epoch 51 --- Training Accuracy: 100.0%, Validation Loss: 11.433
Epoch 52 --- Training Accuracy: 100.0%, Validation Loss: 11.676
Epoch 53 --- Training Accuracy: 100.0%, Validation Loss: 9.593
Epoch 54 --- Training Accuracy: 100.0%, Validation Loss: 9.462
Epoch 55 --- Training Accuracy: 100.0%, Validation Loss: 12.115
Epoch 56 --- Training Accuracy: 100.0%, Validation Loss: 11.755
Epoch 57 --- Training Accuracy: 100.0%, Validation Loss: 10.314
Epoch 58 --- Training Accuracy: 100.0%, Validation Loss: 9.825
Epoch 59 --- Training Accuracy: 100.0%, Validation Loss: 12.120
Epoch 60 --- Training Accuracy: 100.0%, Validation Loss: 12.198
Epoch 61 --- Training Accuracy: 100.0%, Validation Loss: 11.560
Epoch 62 --- Training Accuracy: 100.0%, Validation Loss: 12.252
Epoch 63 --- Training Accuracy: 100.0%, Validation Loss: 11.633
Epoch 64 --- Training Accuracy: 100.0%, Validation Loss: 12.300
Epoch 65 --- Training Accuracy: 100.0%, Validation Loss: 11.475
Epoch 66 --- Training Accuracy: 100.0%, Validation Loss: 11.743
Epoch 67 --- Training Accuracy: 100.0%, Validation Loss: 11.821
Epoch 68 --- Training Accuracy: 100.0%, Validation Loss: 11.466
Epoch 69 --- Training Accuracy: 100.0%, Validation Loss: 13.183
Epoch 70 --- Training Accuracy: 100.0%, Validation Loss: 12.861
Epoch 71 --- Training Accuracy: 100.0%, Validation Loss: 11.340
Epoch 72 --- Training Accuracy: 100.0%, Validation Loss: 11.005
Epoch 73 --- Training Accuracy: 100.0%, Validation Loss: 12.150
Epoch 74 --- Training Accuracy: 100.0%, Validation Loss: 10.769
Epoch 75 --- Training Accuracy: 100.0%, Validation Loss: 12.226
Epoch 76 --- Training Accuracy: 100.0%, Validation Loss: 11.332
Epoch 77 --- Training Accuracy: 100.0%, Validation Loss: 11.879
Epoch 78 --- Training Accuracy: 100.0%, Validation Loss: 11.990
Epoch 79 --- Training Accuracy: 100.0%, Validation Loss: 13.937
Epoch 80 --- Training Accuracy: 100.0%, Validation Loss: 12.359
Epoch 81 --- Training Accuracy: 100.0%, Validation Loss: 12.852
Epoch 82 --- Training Accuracy: 100.0%, Validation Loss: 11.970
Epoch 83 --- Training Accuracy: 100.0%, Validation Loss: 14.081
Epoch 84 --- Training Accuracy: 100.0%, Validation Loss: 12.702
Epoch 85 --- Training Accuracy: 100.0%, Validation Loss: 13.021
Epoch 86 --- Training Accuracy: 100.0%, Validation Loss: 12.188
Epoch 87 --- Training Accuracy: 100.0%, Validation Loss: 13.429
Epoch 88 --- Training Accuracy: 100.0%, Validation Loss: 13.440
Epoch 89 --- Training Accuracy: 100.0%, Validation Loss: 13.202
Epoch 90 --- Training Accuracy: 100.0%, Validation Loss: 12.159
Epoch 91 --- Training Accuracy: 100.0%, Validation Loss: 12.190
Epoch 92 --- Training Accuracy: 100.0%, Validation Loss: 10.972
Epoch 93 --- Training Accuracy: 100.0%, Validation Loss: 12.373
Epoch 94 --- Training Accuracy: 100.0%, Validation Loss: 12.424
Epoch 95 --- Training Accuracy: 100.0%, Validation Loss: 13.225
Epoch 96 --- Training Accuracy: 100.0%, Validation Loss: 13.297
Epoch 97 --- Training Accuracy: 100.0%, Validation Loss: 13.679
Epoch 98 --- Training Accuracy: 100.0%, Validation Loss: 13.033
Epoch 99 --- Training Accuracy: 100.0%, Validation Loss: 13.278
Epoch 100 --- Training Accuracy: 100.0%, Validation Loss: 12.464
Epoch 101 --- Training Accuracy: 100.0%, Validation Loss: 12.099
Epoch 102 --- Training Accuracy: 100.0%, Validation Loss: 13.307
Epoch 103 --- Training Accuracy: 100.0%, Validation Loss: 13.725
Epoch 104 --- Training Accuracy: 100.0%, Validation Loss: 15.329
Epoch 105 --- Training Accuracy: 100.0%, Validation Loss: 12.207
Epoch 106 --- Training Accuracy: 100.0%, Validation Loss: 12.706
Epoch 107 --- Training Accuracy: 100.0%, Validation Loss: 12.864
Epoch 108 --- Training Accuracy: 100.0%, Validation Loss: 13.017
Epoch 109 --- Training Accuracy: 100.0%, Validation Loss: 11.108
Epoch 110 --- Training Accuracy: 100.0%, Validation Loss: 12.597
Epoch 111 --- Training Accuracy: 100.0%, Validation Loss: 12.514
Epoch 112 --- Training Accuracy: 100.0%, Validation Loss: 12.464
Epoch 113 --- Training Accuracy: 100.0%, Validation Loss: 13.814
Epoch 114 --- Training Accuracy: 100.0%, Validation Loss: 16.175
Epoch 115 --- Training Accuracy: 100.0%, Validation Loss: 13.925
Epoch 116 --- Training Accuracy: 100.0%, Validation Loss: 13.711
Epoch 117 --- Training Accuracy: 100.0%, Validation Loss: 15.386
Epoch 118 --- Training Accuracy: 100.0%, Validation Loss: 13.819
Epoch 119 --- Training Accuracy: 100.0%, Validation Loss: 13.201
Epoch 120 --- Training Accuracy: 100.0%, Validation Loss: 15.125
Epoch 121 --- Training Accuracy: 100.0%, Validation Loss: 14.388
Epoch 122 --- Training Accuracy: 100.0%, Validation Loss: 15.340
Epoch 123 --- Training Accuracy: 100.0%, Validation Loss: 13.982
Epoch 124 --- Training Accuracy: 100.0%, Validation Loss: 13.698
Epoch 125 --- Training Accuracy: 100.0%, Validation Loss: 13.604
Epoch 126 --- Training Accuracy: 100.0%, Validation Loss: 13.569
Epoch 127 --- Training Accuracy: 100.0%, Validation Loss: 12.613
Epoch 128 --- Training Accuracy: 100.0%, Validation Loss: 13.706
Epoch 129 --- Training Accuracy: 100.0%, Validation Loss: 13.897
Epoch 130 --- Training Accuracy: 100.0%, Validation Loss: 12.205
Epoch 131 --- Training Accuracy: 100.0%, Validation Loss: 14.913
Epoch 132 --- Training Accuracy: 100.0%, Validation Loss: 13.665
Epoch 133 --- Training Accuracy: 100.0%, Validation Loss: 13.850
Epoch 134 --- Training Accuracy: 100.0%, Validation Loss: 13.901
Epoch 135 --- Training Accuracy: 100.0%, Validation Loss: 14.654
Epoch 136 --- Training Accuracy: 100.0%, Validation Loss: 15.725
Epoch 137 --- Training Accuracy: 100.0%, Validation Loss: 14.617
Epoch 138 --- Training Accuracy: 100.0%, Validation Loss: 13.602
Epoch 139 --- Training Accuracy: 100.0%, Validation Loss: 14.102
Epoch 140 --- Training Accuracy: 100.0%, Validation Loss: 12.953
Epoch 141 --- Training Accuracy: 100.0%, Validation Loss: 15.132
Epoch 142 --- Training Accuracy: 100.0%, Validation Loss: 14.549
Epoch 143 --- Training Accuracy: 100.0%, Validation Loss: 12.701
Time elapsed: 6:33:29
```

```
In [31]: print_validation_accuracy()
```

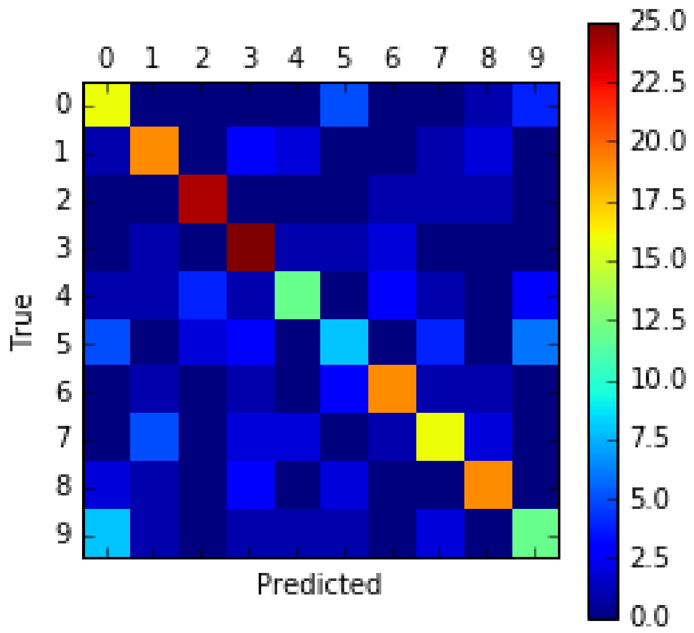
```
Accuracy on Test-Set: 62.5% (170 / 272)
```

7 Validation accuracy and confusion matrix

```
In [32]: print_validation_accuracy(show_example_errors=True, show_confusion_matrix=True)
```

```
Accuracy on Test-Set: 62.5% (170 / 272)
Example errors:
```

```
Confusion Matrix:
[[16  0  0  0  0  5  0  0  1  4]
 [ 1 19  0  3  2  0  0  1  2  0]
 [ 0  0 24  0  0  0  1  1  1  0]
 [ 0  1  0 25  1  1  2  0  0  0]
 [ 1  1  4  1 12  0  3  1  0  3]
 [ 5  0  2  3  0  8  0  4  0  6]
 [ 0  1  0  1  0  3 19  1  1  0]
 [ 0  5  0  2  2  0  1 16  2  0]
 [ 2  1  0  3  0  2  0  0 19  0]
 [ 8  1  0  1  1  1  0  2  0 12]]
```

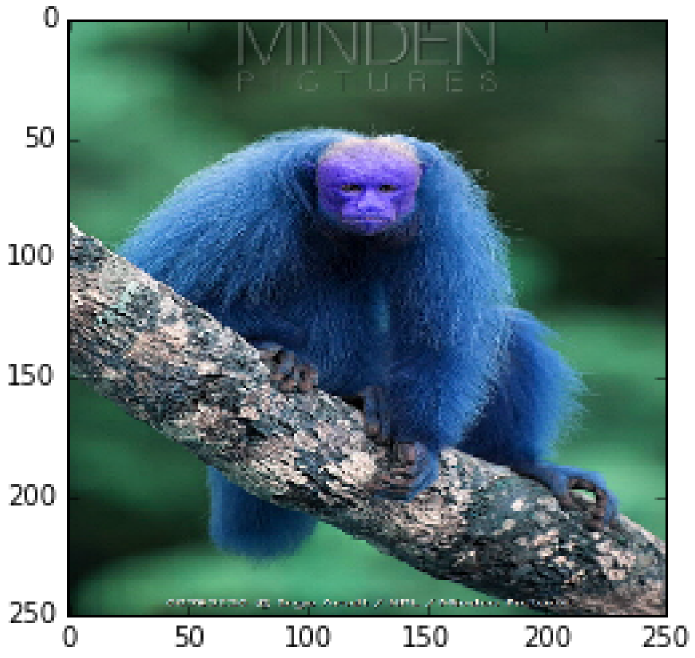


```
In [33]: def plot_conv_weights(weights, input_channel=0):

             w = session.run(weights)

             w_min = np.min(w)
             w_max = np.max(w)

             num_filters = w.shape[3]

             num_grids = math.ceil(math.sqrt(num_filters))

             fig, axes = plt.subplots(num_grids, num_grids, figsize=(5,5))

             for i, ax in enumerate(axes.flat):
                 if i<num_filters:

                     img = w[:, :, input_channel, i]

                     ax.imshow(img, vmin=w_min, vmax=w_max,
                               interpolation='nearest', cmap='seismic')

                 ax.set_xticks([])
                 ax.set_yticks([])

             plt.show()
```

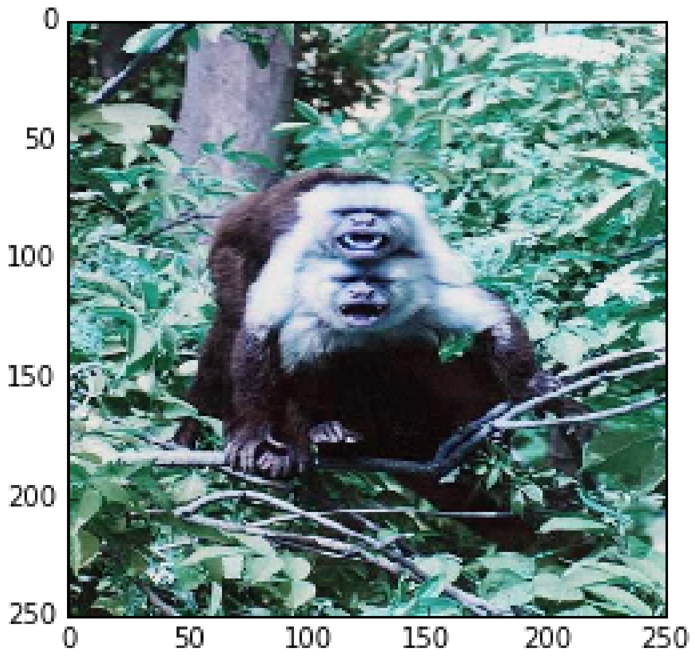
```
        def plot_conv_layer(layer, image):

            image = image.reshape(img_size_flat)

            feed_dict = {x: [image]}

            values = session.run(layer, feed_dict=feed_dict)

            num_filters = values.shape[3]

            num_grids = math.ceil(math.sqrt(num_filters))

            fig, axes = plt.subplots(num_grids, num_grids,figsize=(5,5))

            for i, ax in enumerate(axes.flat):
                if i<num_filters:

                    img = values[0, :, :, i]

                    ax.imshow(img, interpolation='nearest', cmap='binary')

                ax.set_xticks([])
                ax.set_yticks([])

            plt.show()
```

```
In [34]: def plot_image(image):
             plt.imshow(image.reshape(img_size, img_size, num_channels),
                        interpolation='nearest')
             plt.show()
```

```
In [35]: image1 = x_test[0]
         plot_image(image1)
```

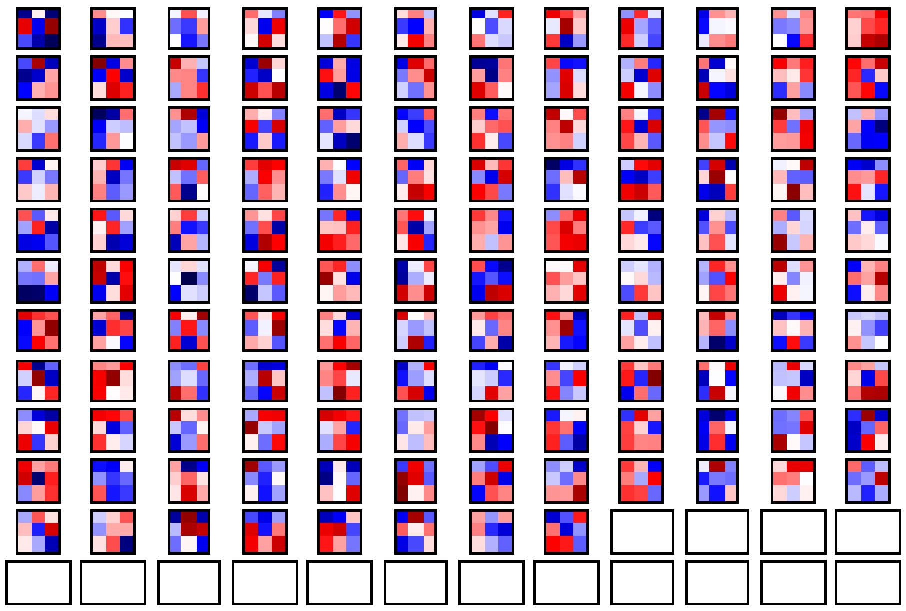


```
In [36]: image2 = x_test[36]
         plot_image(image2)
```

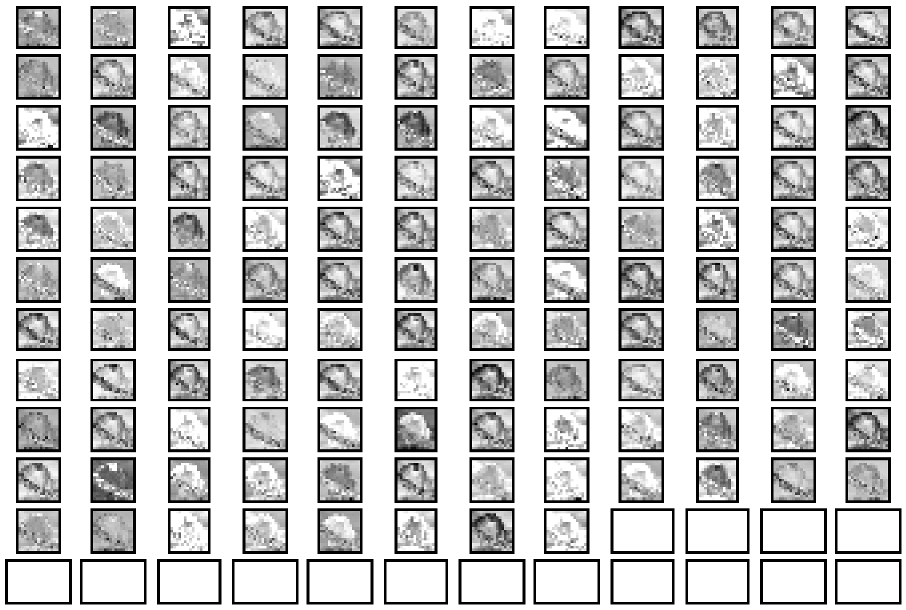


8 Visualization of Weights and Layers

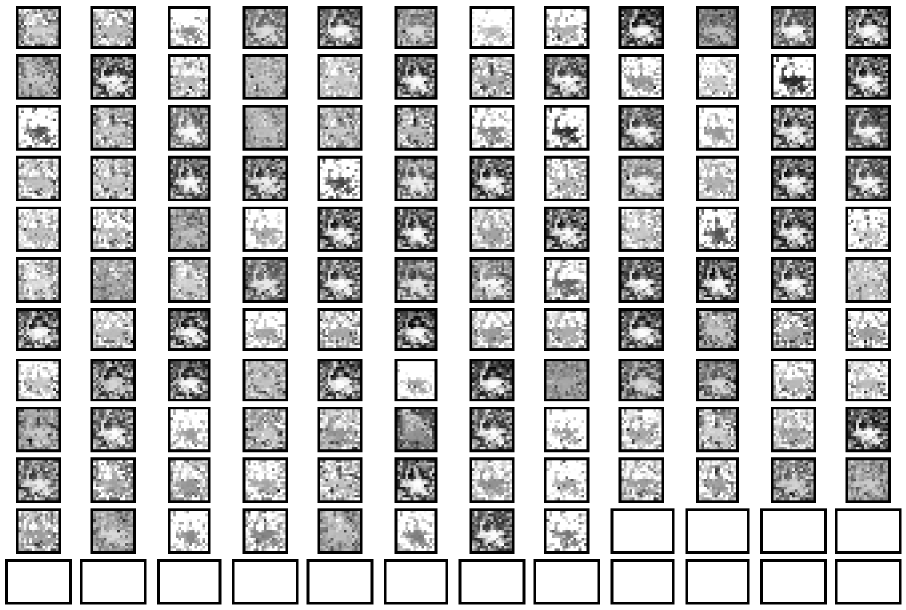
In [37]:
```
plot_conv_weights(weights=weights_conv1)
```



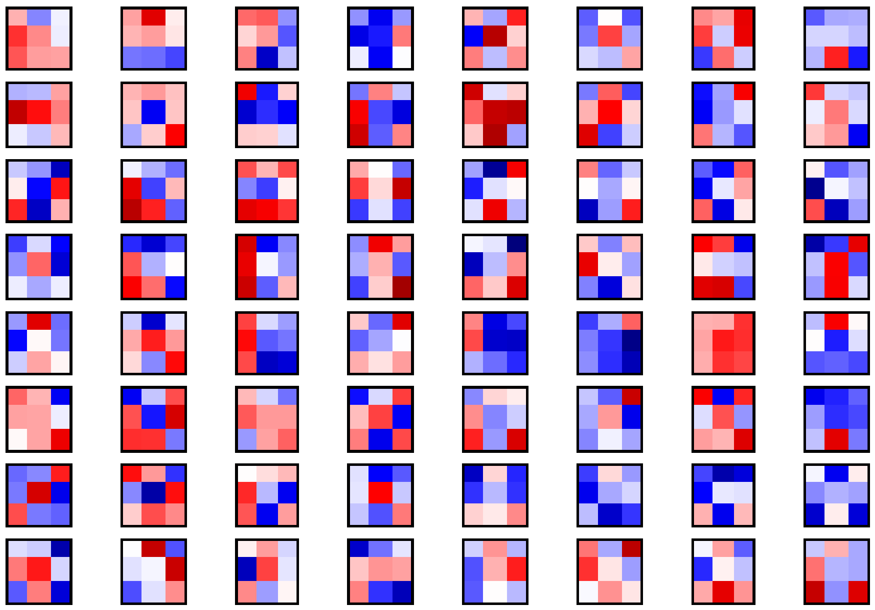
In [38]:
```
plot_conv_layer(layer=layer_conv1, image=image1)
```



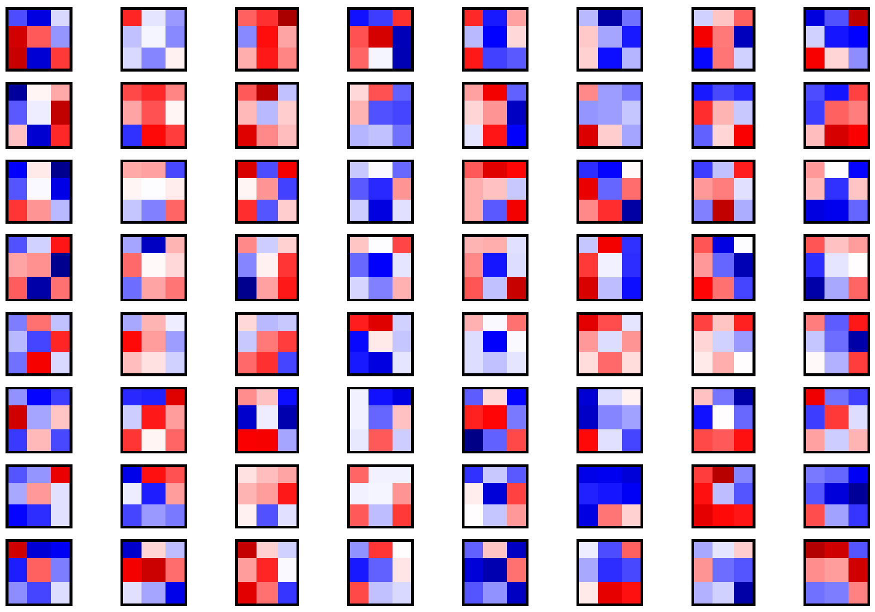
```
In [39]: plot_conv_layer(layer=layer_conv1, image=image2)
```



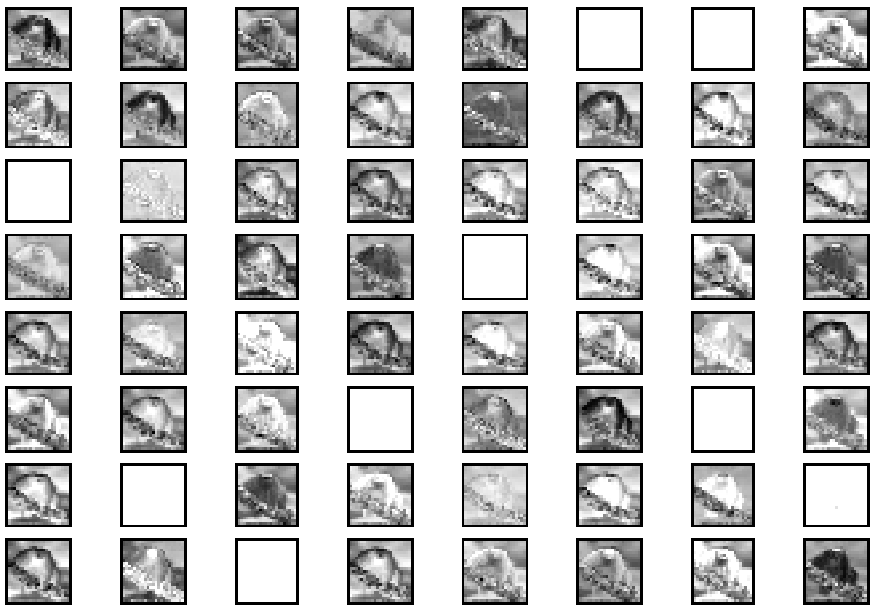
```
In [40]: plot_conv_weights(weights=weights_conv2, input_channel=0)
```



```
In [41]: plot_conv_weights(weights=weights_conv2, input_channel=1)
```



```
In [42]: plot_conv_layer(layer=layer_conv2, image=image1)
```



```
In [43]: plot_conv_layer(layer=layer_conv2, image=image2)
```

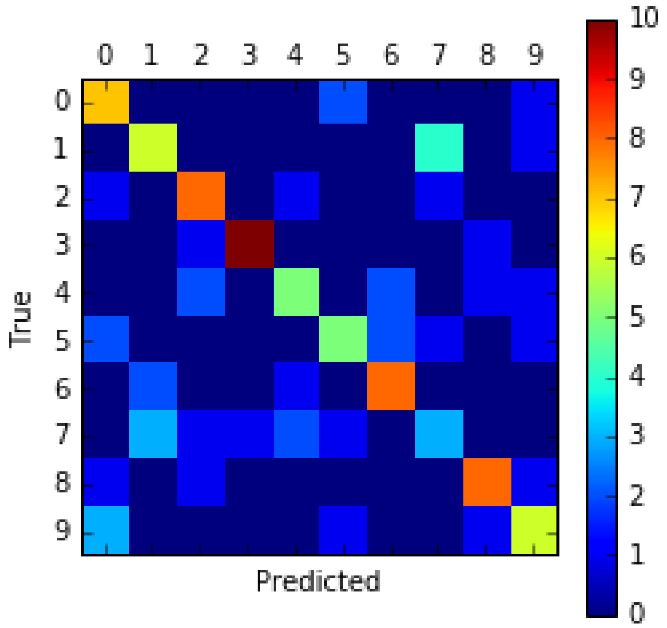
```
In [47]: batch_size1=110
         def print_test_accuracy(show_example_errors=False,
                                 show_confusion_matrix=False):

             num_test = len(x_test)

             cls_pred = np.zeros(shape=num_test, dtype=np.int)

             i = 0

             while i < num_test:
                 j = min(i + batch_size1, num_test)

                 images = x_test[i:j, :, :, :].reshape(batch_size1, img_size_flat)

                 labels = labels_test[i:j, :]

                 feed_dict = {x: images,
                              y_true: labels}

                 cls_pred[i:j] = session.run(y_pred_cls, feed_dict=feed_dict)

                 i = j

             cls_true = np.array(y_test)

             correct = (cls_true == cls_pred)
```

```
correct_sum = correct.sum()

acc = float(correct_sum) / num_test


msg = "Accuracy on Test-Set: {0:.1%} ({1} / {2})"
print(msg.format(acc, correct_sum, num_test))

cls_true = y_test


cm = confusion_matrix(y_true=cls_true,
                      y_pred=cls_pred)

print(cm)

plt.matshow(cm)

plt.colorbar()
tick_marks = np.arange(num_classes)
plt.xticks(tick_marks, range(num_classes))
plt.yticks(tick_marks, range(num_classes))
plt.xlabel('Predicted')
plt.ylabel('True')

plt.show()
```

9 Test accuracy and confusion matrix

```
In [48]: print_test_accuracy()
```

```
Accuracy on Test-Set: 60.0% (66 / 110)
[[ 7  0  0  0  0  2  0  0  0  1]
 [ 0  6  0  0  0  0  0  4  0  1]
 [ 1  0  8  0  1  0  0  1  0  0]
 [ 0  0  1 10  0  0  0  0  1  0]
 [ 0  0  2  0  5  0  2  0  1  1]
 [ 2  0  0  0  0  5  2  1  0  1]
 [ 0  2  0  0  1  0  8  0  0  0]
 [ 0  3  1  1  2  1  0  3  0  0]
 [ 1  0  1  0  0  0  0  0  8  1]
 [ 3  0  0  0  0  1  0  0  1  6]]
```


0
1
2
3
4
5
6
7
8
9
True
Predicted
10