feed_forward

February 22, 2019

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import imread
```

In [2]:
```
def sigmoid(x):
    return 1/(1 + np.exp(-x))

def relu(x):
    x[x < 0] = 0
    return x

def tanh(x):
    return np.tanh(x)

def softmax(x):
    return np.exp(x) / np.sum(np.exp(x))
```

In [3]:
```
def conv2d(image,filters, kernel, stride, activation, padding = 'valid'):
    filter_shape = kernel
    if(len(image.shape) > 2):
        filter_shape = kernel + (image.shape[2],)
    weights = np.random.normal(size = ((filters,) + filter_shape))
    bias = np.random.rand(filters)
    out = []
    for i in range(filters):
        if( i == 0):
            output = np.zeros((((image.shape[0] - kernel[0])/stride[0]) + 1 ,((image.sh
#     print output.shape
    if(padding == 'same' and i == 0):
        output = np.zeros(image.shape[0:2])
        temp = ((image.shape[0] - 1)* stride[0]) + kernel[0] - image.shape[0]
        pad_size_rows = temp + temp%2
        temp = ((image.shape[1] - 1)* stride[1]) + kernel[1] - image.shape[1]
        pad_size_cols = temp + temp%2
        if(len(image.shape) > 2):
            image = np.pad(image, ((pad_size_rows/2,),(pad_size_cols/2,),(0,)), 'co
        else:
            image = np.pad(image, ((pad_size_rows/2,),(pad_size_cols/2,)), 'constan
```

```
            img_cur_row = 0
            out_cur_row = 0
            while(img_cur_row + kernel[0] <= image.shape[0] - 1):
                img_cur_col = 0
                out_cur_col = 0
                while(img_cur_col + kernel[1] <= image.shape[1] - 1):
        #             print(img_cur_row, img_cur_col, out_cur_row, out_cur_col)
                    if(len(image.shape) > 2):
                        output[out_cur_row , out_cur_col] = np.sum(np.multiply(image[img_cu
                    else:
                        output[out_cur_row , out_cur_col] = np.sum(np.multiply(image[img_cu
                    img_cur_col = img_cur_col + stride[1]
                    out_cur_col = out_cur_col + 1
                img_cur_row = img_cur_row + stride[0]
                out_cur_row = out_cur_row + 1
        #         print(123,img_cur_row, img_cur_col, out_cur_row, out_cur_col)
            if(activation == 'sigmoid'):
                output = sigmoid(output)
            elif(activation == 'relu'):
                output = relu(output)
            elif(activation == 'tanh'):
                output = tanh(output)
            out.append(output)
        #         print output.shape
        out = np.array(out)
        out = np.moveaxis(out, 0,2)
        #     print out.shape
        return out , weights, bias
```

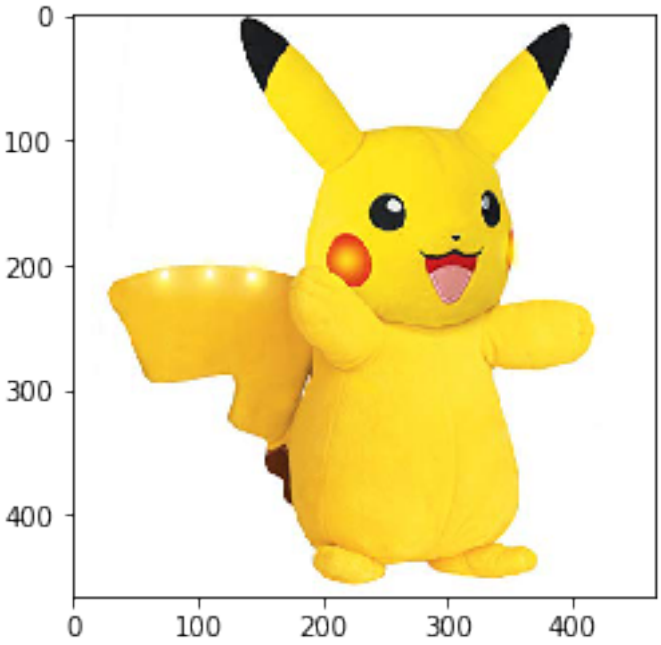
In [4]:
```
def pooling(image, kernel, stride, pool_func = 'max'):
    if(len(image.shape) > 2):
        output = np.zeros((((image.shape[0] - kernel[0])/stride[0]) + 1 ,((image.shape
    else:
        output = np.zeros((((image.shape[0] - kernel[0])/stride[0]) + 1 ,((image.shape
    #     print output.shape
    img_cur_row = 0
    out_cur_row = 0
    while(img_cur_row + kernel[0] <= image.shape[0] - 1):
        img_cur_col = 0
        out_cur_col = 0
        while(img_cur_col + kernel[1] <= image.shape[1] - 1):
            if(pool_func == 'max'):
                output[out_cur_row , out_cur_col] = np.amax(np.amax(image[img_cur_row:
            elif(pool_func == 'min'):
                output[out_cur_row , out_cur_col] = np.amin(np.amin(image[img_cur_row:
            elif(pool_func == 'average'):
                output[out_cur_row , out_cur_col] = np.mean(np.mean(image[img_cur_row:
            img_cur_col = img_cur_col + stride[1]
```

```
                out_cur_col = out_cur_col + 1
            img_cur_row = img_cur_row + stride[0]
            out_cur_row = out_cur_row + 1
        return output

In [5]: def flatten(inp, output_length = -1):
        inp = inp.flatten()
        if(output_length != -1):
            mat = np.random.uniform(size = (output_length, len(inp)))
            out = np.matmul(mat, inp)
            return out
        else:
            return inp

In [6]: def fully_connected(inp, nodes, activation):
        weights = np.asmatrix(np.random.rand(nodes, len(inp)))
        output_raw = np.matmul(weights, inp)
        output_raw = output_raw/np.max(output_raw)
        if(activation == 'sigmoid'):
            output = sigmoid(output_raw)
        elif(activation == 'relu'):
            output = relu(output_raw)
        elif(activation == 'tanh'):
            output = tanh(output_raw)
        elif(activation == 'softmax'):
            output = softmax(output_raw)
        if(output.shape[0] == 1):
            output = np.moveaxis(output, 0,1)
            output_raw = np.moveaxis(output_raw, 0,1)
        return output, output_raw

In [7]: def feed_forward(feed_dict):
        final_out = []
        inp = feed_dict['input']
        layers = feed_dict['layers']

        for i in range(len(layers)):
            if(layers[i]['type'] == 'conv'):
                output, weights, bias = conv2d(inp, filters = layers[i]['filters'], kernel
                out_dict = {'layer_number': i , 'type': 'conv', 'output': output, 'weights
                final_out.append(out_dict)
            elif(layers[i]['type'] == 'pool'):
                output= pooling(inp, kernel = layers[i]['kernel'] , stride = layers[i]['st
                out_dict = {'layer_number': i , 'type': 'pool', 'output': output}
                final_out.append(out_dict)
            elif(layers[i]['type'] == 'fc'):
                output, output_raw = fully_connected(inp, nodes = layers[i]['nodes'], acti
                out_dict = {'layer_number': i , 'type': 'fc', 'output': output, 'output_ra
```

```
                final_out.append(out_dict)
            elif(layers[i]['type'] == 'flat'):
                output = flatten(inp, output_length = layers[i]['output_length'])
                out_dict = {'layer_number': i , 'type': 'flat', 'output': output}
                final_out.append(out_dict)
            inp = output
        return final_out
```

In [8]:
```
img = imread("10.jpg")
print img.shape
plt.imshow(img)
plt.show()

feed_dict = {}
feed_dict['input'] = img
feed_dict['layers'] = [{'type': 'conv', 'filters': 4 , 'kernel': (3,3) , 'stride': (2,2
                        {'type': 'conv', 'filters': 4 , 'kernel': (3,3) , 'stride': (2,2
                        {'type': 'pool', 'kernel': (2,2) , 'stride': (1,1),  'pool_func
                        {'type': 'conv', 'filters': 8 , 'kernel': (2,2) , 'stride': (1,1
                        {'type': 'conv', 'filters': 8 , 'kernel': (2,2) , 'stride': (1,1
                        {'type': 'pool', 'kernel': (2,2) , 'stride': (2,2),  'pool_func
                        {'type': 'flat', 'output_length': 2048},
                        {'type': 'fc',  'nodes': 1024, 'activation' : 'sigmoid'},
                        {'type': 'fc',  'nodes': 1024, 'activation' : 'sigmoid'},
                        {'type': 'fc',  'nodes': 10, 'activation' : 'sigmoid'}]


output = feed_forward(feed_dict)

for i in range(len(output)):
    print output[i]['layer_number'],output[i]['output'].shape

# print output[len(output) - 1]['output']
```

```
(466, 466, 3)


/home/legion/.local/lib/python2.7/site-packages/ipykernel_launcher.py:1: DeprecationWarning: `i
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  """Entry point for launching an IPython kernel.
```




```
0 (232, 232, 4)
1 (115, 115, 4)
2 (114, 114, 4)
3 (113, 113, 8)
4 (112, 112, 8)
5 (56, 56, 8)
6 (2048,)
7 (1024, 1)
8 (1024, 1)
9 (10, 1)
```

```
In [9]: print output[len(output) - 1]['output']
```

```
[[0.7248013 ]
 [0.72731453]
 [0.72857369]
 [0.71918596]
 [0.73105858]
 [0.7271002 ]
 [0.7256236 ]
 [0.72806374]
 [0.73023833]
 [0.73043772]]
```