In [18]:

```
import numpy as np
from sklearn.utils import shuffle
from skimage import io
from skimage.transform import resize
from skimage.color import rgba2rgb
import matplotlib.pyplot as plt
import os
```

In [19]:

```
class perceptron:
    def logistic_regression(self, w, b, X):
        z = np.dot(w.T, X) + b
        return z
    def sigmoid(self, z):
        return 1/(1+np.exp(-z))
    def propagate(self, X, Y, w, b):
        m = X.shape[1]
        z = self.logistic_regression(w, b, X)
        A = self.sigmoid(1/(1+np.exp(-z)))
        cost = -((1 / m) * (np.sum(Y * np.log(A) + (1 - Y) * np.log(1 - A))))
        dw = (1 / 2) * np.dot(X, (A - Y).T)
        db = (1 / 2) * np.sum(A - Y)
        grads = {"dw":dw, "db":db}
        return grads, cost
    def optimize(self, w, b, X, Y, ni, lr):
        for i in range(ni):
            grads, cost = self.propagate(X, Y, w, b,)
            dw = grads["dw"]
            db = grads["db"]
            w = w - lr * dw
            b = b - lr * db
            if i % 100 == 0:
                print("Cost: " + str(cost))
        params = {"w":w, "b":b}
        return params
    def predict(self, w, b, X):
        m = X.shape[1]
        y_predict = np.zeros((1, m))
        z = self.logistic_regression(w, b, X)
        A = self.sigmoid(z)
        for i in range(A.shape[1]):
            y_predict[0,i] = round(A[0,i])
        return y_predict
    def model(self, X, Y, Xt, Yt, ni=2000, lr=0.5):
        w, b = np.zeros((X.shape[0],1)), 0
        params = self.optimize(w, b, X, Y, ni, lr)
        w = params["w"]
        b = params["b"]
        y_predict_train = self.predict(w, b, X)
        y_predict_test = self.predict(w, b, Xt)
        print("Exactitud de entrenamiento: \t", str( 100 - np.mean(np.abs(y_predict_train - Y) * 100)))
        print("Exactitud de prueba: \t", str(100 - np.mean(np.abs(y_predict_test - Yt) * 100)))
        d = {"y_predict_train":y_predict_train, "y_predict_test":y_predict_test}
        return d
```

In [20]:

```
class dataset:
    def calculate(self, amount, percent):
        train = amount * percent / 100
        test = amount - train
        e =     {"train":round(train),
                "test":round(test)}
        return e
    def rename(self, path):
        for folder in os.listdir(path):
            new_name=0
            folder_path = os.path.join(path,folder)
            print("[+] Renaming data in", folder_path)
            if os.path.isdir(folder_path):
                for img in os.listdir(folder_path):
                    try:
                        extension = os.path.splitext(img)[1]
```

```python
                        img_path = os.path.join(folder_path, img)
                        new_img_path = os.path.join(folder_path,"img_"+str(new_name) + extension)
                        img_file = io.imread(img_path)
                        if img_file.shape[-1]==4:
                            print("[*] Converting",img,"RGBA to RGB")
                            rgb_img = rgba2rgb(img_file)
                            io.imsave(os.path.join(folder_path,"img_"+str(new_name) + ".jpg"), rgb_img)
                            os.remove(img_path)
                        else:
                            os.rename(img_path, new_img_path)
                        new_name+=1
                    except FileExistsError as e:
                        print("[*] File already exists. Resolving...")
                        os.remove(img_path)
                        io.imsave(os.path.join(folder_path,"img_"+str(new_name) + extension), img_file)
        print("[*] Done!")
    def create(self, path, *size):
        folders = os.listdir(path)
        if "yes" in folders and "no" in folders:
            x_train_set_orig = []
            y_train_set = []
            x_test_set_orig = []
            y_test_set = []
            print("[*] Folder 'yes' and 'no' found")
            for folder in folders:
                folder_path = os.path.join(path, folder)
                if os.path.isdir(folder_path):
                    if folder=="yes" or folder=="no":
                        folder_content = os.listdir(folder_path)
                        e = self.calculate(len(folder_content), 65)
                        train = e["train"]
                        test = e["test"]
                        print("\n[*] Folder:\t", folder)
                        print("[*] Images:\t", len(folder_content))
                        print("[*] Training:\t", train)
                        print("[*] Test:\t", test)
                        for img in folder_content:
                            img_path = os.path.join(folder_path, img)
                            array_img = io.imread(img_path)
                            image = resize(array_img, size[0],anti_aliasing=False, preserve_range=True)
                            if folder_content.index(img) < train:
                                x_train_set_orig.append(image)
                                if folder == "yes":
                                    y_train_set.append(1)
                                else:
                                    y_train_set.append(0)
                            else:
                                x_test_set_orig.append(image)
                                if folder == "yes":
                                    y_test_set.append(1)
                                else:
                                    y_test_set.append(0)
                    else:
                        print("[*] Folder", folder, "ignored")
            print("\n[*] Successfully generated dataset!")
        else:
            raise Exception("[!] No folder 'yes' or 'no' found")
        x_train_set_orig = np.array(x_train_set_orig)
        x_test_set_orig = np.array(x_test_set_orig)
        y_train_set = np.array(y_train_set)
        y_test_set = np.array(y_test_set)

        x_train_set_orig, y_train_set = shuffle(x_train_set_orig, y_train_set)
        x_test_set_orig, y_test_set = shuffle(x_test_set_orig, y_test_set)
        x_train_set_flat = x_train_set_orig.reshape(x_train_set_orig.shape[0],-1).T
        x_test_set_flat = x_test_set_orig.reshape(x_test_set_orig.shape[0],-1).T

        x_train_set = x_train_set_flat/255
        x_test_set = x_test_set_flat/255

        cds = {"x_train_set":x_train_set,
            "x_test_set":x_test_set,
            "y_train_set":y_train_set,
            "y_test_set":y_test_set}
        return cds
```

In [21]:

```
ds = dataset()
#ds.rename("C:/Users/nico/Dropbox/Coursera/DeepLearning/datasets/pikachu")
```

In [22]:

```
#Run!
cds = ds.create("C:/Users/nico/Dropbox/Coursera/DeepLearning/datasets/pikachu",(100,100))
x_train_set = cds["x_train_set"]
y_train_set = cds["y_train_set"]
x_test_set = cds["x_test_set"]
y_test_set = cds["y_test_set"]
```

```
[*] Folder 'yes' and 'no' found

[*] Folder:  no
[*] Images:  54
[*] Training:  35
[*] Test:  19

[*] Folder:  yes
[*] Images:  77
[*] Training:  50
[*] Test:  27

[*] Successfully generated dataset!
```

In [23]:

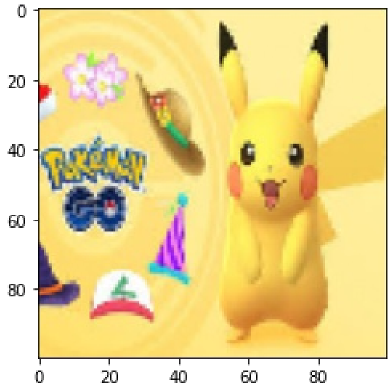
```
p = perceptron()
d = p.model(x_train_set, y_train_set, x_test_set, y_test_set, lr=0.01)
```

```
Cost: 0.6799593371212832
Cost: 0.5056258605902915
<ipython-input-19-7082772f7f10>:10: RuntimeWarning: overflow encountered in exp
  A = self.sigmoid(1/(1+np.exp(-z)))
Cost: 0.5068426203788917
Cost: 0.5049792435824382
Cost: 0.5049788293085258
Cost: 0.5049704829172775
Cost: 0.5065084304902002
Cost: 0.5049792700850599
Cost: 0.5049766992005591
Cost: 0.5049157483503709
Cost: 0.5049792457356457
Cost: 0.5049792447150518
Cost: 0.5049759856298176
Cost: 0.5049791650504627
Cost: 0.5049155444885103
Cost: 0.5047469571448172
Cost: 0.504924590922717
Cost: 0.5049792437743846
Cost: 0.5055697728466513
Cost: 0.5049789244661735
Exactitud de entrenamiento:   92.94117647058823
Exactitud de prueba:   89.13043478260869
<ipython-input-19-7082772f7f10>:6: RuntimeWarning: overflow encountered in exp
  return 1/(1+np.exp(-z))
```

In [28]:

```
index=4
plt.imshow(x_test_set[:,index].reshape(100,100,3))
print("y = " + str(y_test_set[index]) + ", you predicted: " + str(d["y_predict_test"][0,index]))
```

```
y = 1, you predicted: 1.0
```



In [ ]: