In [1]:
```
import os
from pathlib import Path
from keras.preprocessing import image
import matplotlib.pyplot as plt
```

In [2]:
```
p = Path(r"///Users/tanmaypriyadarshi/Desktop/pokemon")
```

In [3]:
```
dirs = p.glob("*")
```

In [4]:
```
image_data = []
labels = []
image_path = []
label_dict = {"Pikachu":0,"Bulbasaur":1, "Meowth":2}
for folder_dirs in dirs:
    label = str(folder_dirs).split('/')[-1]
    print(label)
    cnt = 0
    for img_path in folder_dirs.glob("*.jpg"):
        img = image.load_img(img_path,target_size=(40,40))
        img_array = image.img_to_array(img)
        image_data.append(img_array)
        labels.append(label_dict[label])
        cnt+=1
    print(cnt)
```

```
Pikachu
199
Bulbasaur
174
Meowth
70
```

Visualisation

In [5]:
```
import numpy as np
```

In [6]:
```
print(len(image_data))
print(len(labels))
print(labels)
```

```
443
443
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2]
```

In [7]:
```
import random
random.seed(10)
```

In [8]:
```
from sklearn.utils import shuffle
```

In [9]:
```
x = np.array(image_data)
y = np.array(labels)
x,y = shuffle(x,y,random_state=2)
print(x.shape)
print(y.shape)
print(type(x))
x = x/255.0
#print(x)
print(y)
```

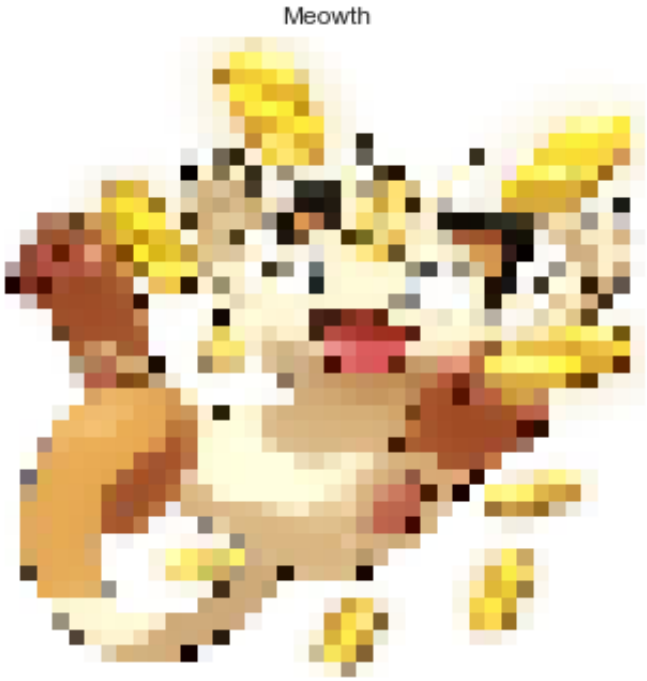
```
(443, 40, 40, 3)
(443,)
<class 'numpy.ndarray'>
[1 2 0 0 2 0 1 0 1 0 1 1 1 1 0 1 2 0 1 2 2 1 2 0 1 0 0 2 0 0 1 0 0
1 1 2 0
 2 0 0 1 1 1 0 0 0 1 1 1 2 1 0 0 1 2 1 1 0 0 0 2 0 0 0 0 0 0 1 0 0
1 1 0 1
 0 1 0 2 2 1 2 1 1 1 0 2 1 0 0 0 0 0 2 1 0 1 0 0 0 0 0 0 0 1 0 0 2
2 0 2 1
 1 1 0 2 2 0 0 0 0 1 1 1 0 1 1 1 1 1 2 1 1 0 0 1 1 0 0 1 1 2 1 1 2
0 1 1 0
 0 2 0 1 1 0 1 0 1 1 0 1 0 1 1 0 1 0 1 2 1 0 1 1 0 0 2 2 1 0 0 1 1
0 0 0 1
 1 1 0 1 0 0 1 0 1 0 1 2 1 0 1 0 0 2 2 1 0 1 1 0 0 1 0 0 0 1 1 2 2
0 0 1 0
 1 0 0 0 1 2 0 0 0 0 0 1 0 1 0 1 1 1 0 1 2 0 2 2 0 1 0 0 0 1 0 2 1
1 1 0 0
 1 0 0 0 1 0 1 0 1 0 2 0 1 1 1 0 1 1 1 1 0 0 1 1 2 0 0 0 0 1 1 2 0
1 0 2 1
 0 0 1 2 2 0 1 0 2 1 2 2 1 0 0 1 2 2 2 1 2 1 0 1 2 0 2 0 1 0 1 0 0
0 2 1 0
 0 0 1 2 2 2 0 0 1 0 1 2 1 2 0 0 1 0 1 2 1 1 2 1 0 0 0 0 2 1 0 0 0
1 1 0 1
 0 1 0 1 0 0 1 0 0 0 1 1 1 0 0 2 1 1 1 1 0 1 0 1 2 2 0 0 1 0 0 0 0
1 1 1 0
 0 1 1 1 2 0 0 0 1 0 1 1 0 0 0 1 0 0 2 1 1 1 0 1 2 0 2 0 1 1 0 1 0
0 0 0]
```

Let's draw some image

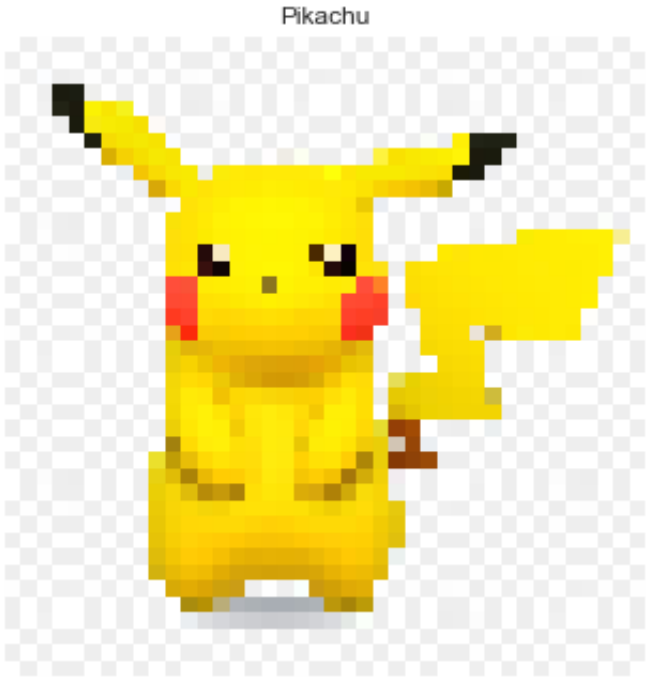
In [10]:
```
def drawImg(img,label):
    name = [key for key in label_dict.keys() if label_dict[key] ==
label]
    print(name)
    plt.title(name[0])
    plt.style.use("seaborn")
    plt.axis("off")
    plt.imshow(img)
    plt.show()
```

In [13]:
```
for i in range(1,20):
    drawImg(x[i].reshape(40,40,3),y[i])
```

['Meowth']



['Pikachu']



['Pikachu']

Pikachu

```
['Meowth']
```

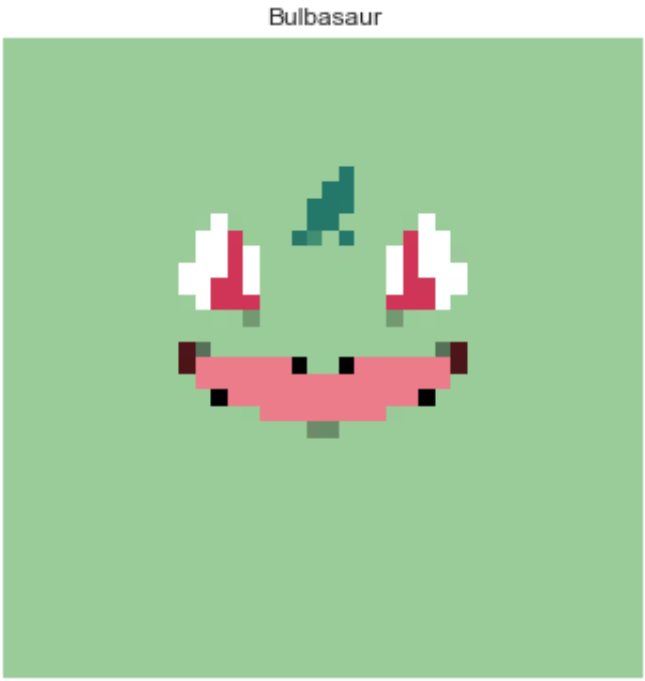
Meowth

```
['Pikachu']
```


```
['Bulbasaur']
```

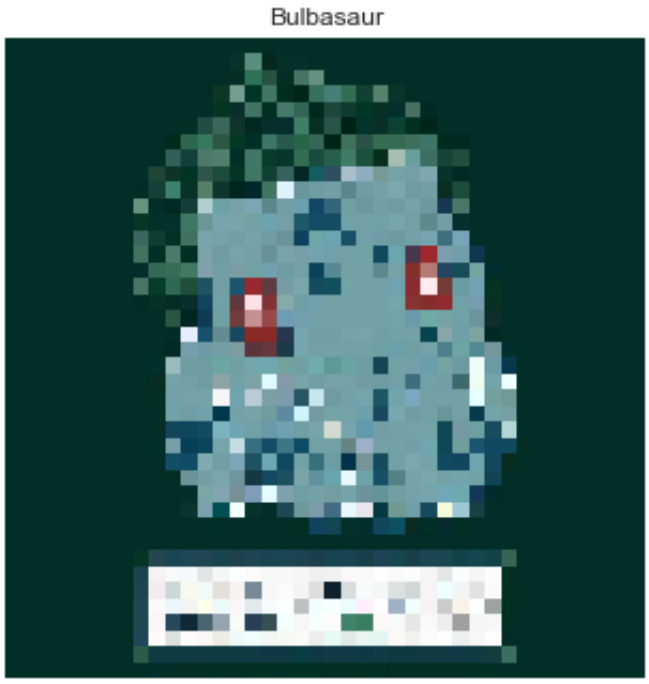

```
['Pikachu']
```


```
['Bulbasaur']
```

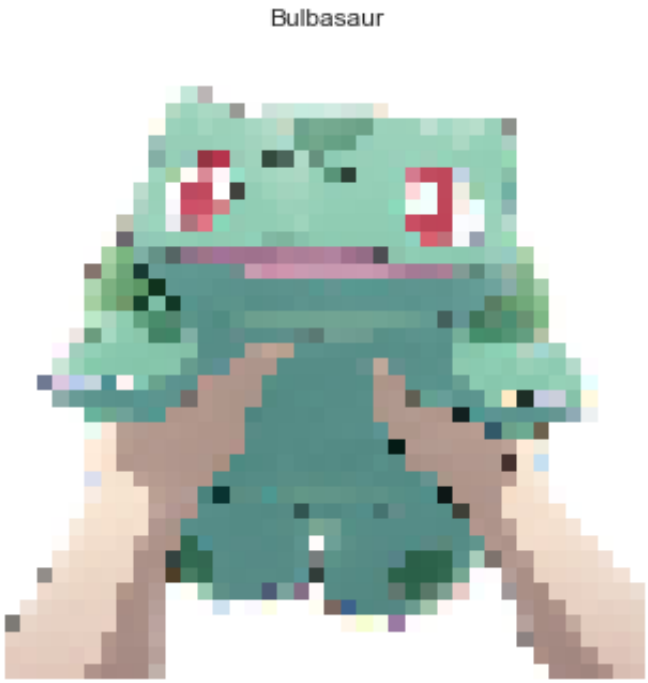



```
['Pikachu']
```

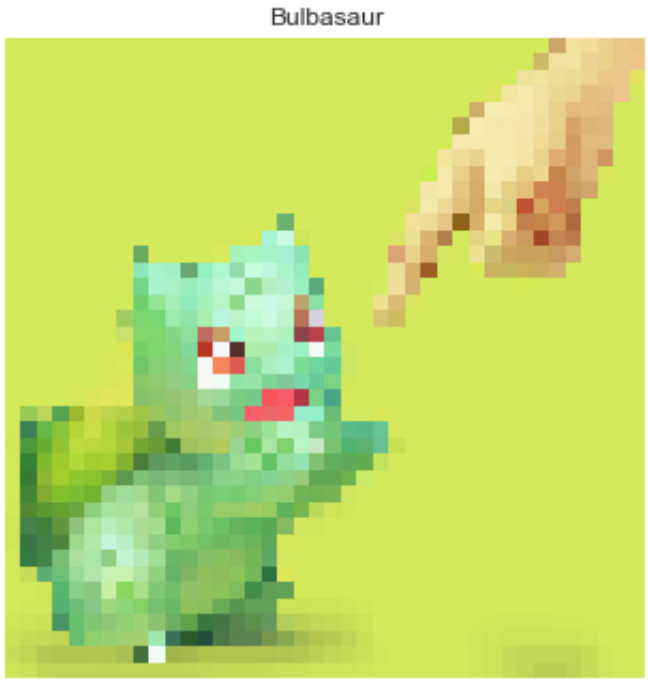

```
['Bulbasaur']
```




```
['Bulbasaur']
```

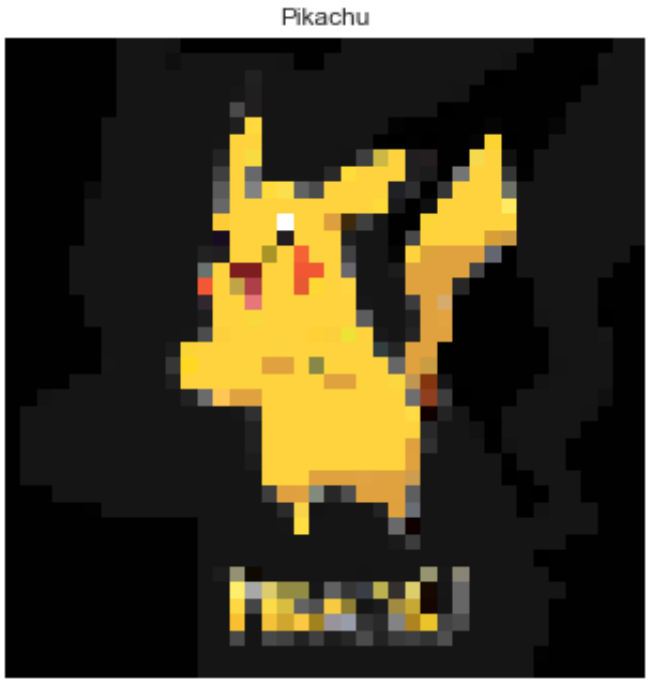



```
['Bulbasaur']
```




```
['Bulbasaur']
```


```
['Pikachu']
```

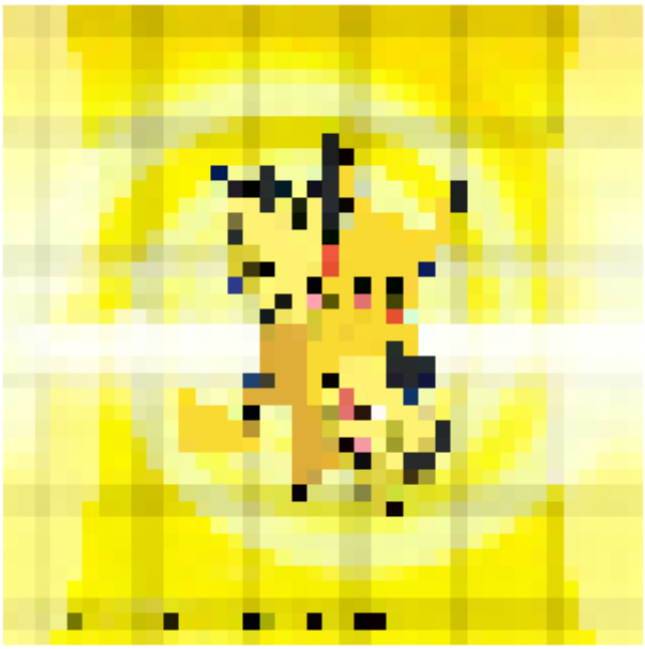



```
['Bulbasaur']
```


```
['Meowth']
```


```
['Pikachu']
```

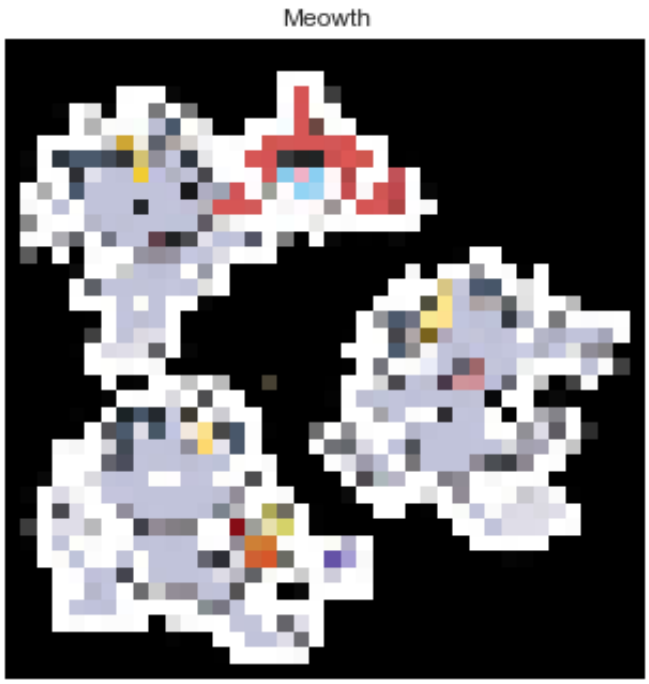
Pikachu



```
['Bulbasaur']
```

Bulbasaur

```
['Meowth']
```



training and testing dataset.....

In [14]:
```
split = int(x.shape[0]*0.8)
print(split)
```

```
354
```

In [15]:
```
x_ = np.array(x)
y_ = np.array(y)
```

In [16]:
```
# training set

x = x_[:split,:]
y = y_[:split]
```

In [17]:
```
#testing set

xtest = x_[split:,:]
ytest = y_[split:]
```

In [18]:
```
print(x.shape)
print(y.shape)
print(xtest.shape)
print(ytest.shape)
```

```
(354, 40, 40, 3)
(354,)
(89, 40, 40, 3)
(89,)
```

In [19]:
```
def softmax(z):
    ex_pa = np.exp(z)
    ans = ex_pa/np.sum(ex_pa,axis=1,keepdims=True)
    return ans
```

In [20]:
```
class NeuralNetwork:
    def __init__(self,input_size,layer,output_size):
        np.random.seed(0)

        model = {}

        model['w1']= np.random.randn(input_size,layer[0])
        model['b1']= np.zeros((1,layer[0]))

        model['w2']= np.random.randn(layer[0],layer[1])
        model['b2']= np.zeros((1,layer[1]))

        model['w3']= np.random.randn(layer[1],output_size)
        model['b3']= np.zeros((1,output_size))

        self.model = model




    def forward(self,x):
        w1,w2,w3 = self.model['w1'],self.model['w2'],self.model['w3
']
        b1,b2,b3 = self.model['b1'],self.model['b2'],self.model['b3
']

        z1 = np.dot(x,w1) + b1
        a1 = np.tanh(z1)

        z2 = np.dot(a1,w2) + b2
        a2 = np.tanh(z2)

        z3 = np.dot(a2,w3) + b3
        y_ = softmax(z3)

        self.activation_outputs = (a1,a2,y_)
        return y_
```

```
    def backward(self,x,y,learning_rate=0.001):
        w1,w2,w3 = self.model['w1'],self.model['w2'],self.model['w3
']
        b1,b2,b3 = self.model['b1'],self.model['b2'],self.model['b3
']
        m = x.shape[0]

        a1,a2,y_ = self.activation_outputs

        delta3 = y_ - y
        dw3 = np.dot(a2.T,delta3)
        db3 = np.sum(delta3,axis=0)/float(m)

        delta2 = (1-np.square(a2))*np.dot(delta3,w3.T)
        dw2 = np.dot(a1.T,delta2)
        db2 = np.sum(delta2,axis=0)/float(m)

        delta1 = (1-np.square(a1))*np.dot(delta2,w2.T)
        dw1 = np.dot(x.T,delta1)
        db1 = np.sum(delta1,axis=0)/float(m)

        # update the model parameter using gradient descent...

        self.model['w1'] -= learning_rate*dw1
        self.model['b1'] -= learning_rate*db1

        self.model['w2'] -= learning_rate*dw2
        self.model['b2'] -= learning_rate*db2

        self.model['w3'] -= learning_rate*dw3
        self.model['b3'] -= learning_rate*db3


    def predict(self,x):

        y_out = self.forward(x)
        return np.argmax(y_out,axis=1)

    def summary(self):

        w1,w2,w3 = self.model['w1'],self.model['w2'],self.model['w3
']
        a1,a2,y_ = self.activation_outputs

        print("w1",w1.shape)
        print("a1",a1.shape)


        print("w2",w2.shape)
        print("a2",a2.shape)
```

```
        print("w3",w3.shape)
        print("y_",y_.shape)
```

In [21]:
```
def loss(y_hot,p):
    l = -np.mean(y_hot*np.log(p))
    return l

def one_hot(y,depth):

    m = y.shape[0]
    y_oht = np.zeros((m,depth))
    print(y_oht.shape)
    y_oht[np.arange(m),y]=1

    return y_oht
```

In [29]:
```
def train(x,y,model,epochs,learning_rate,logs=True):
    training_losses = []

    classes = len(np.unique(y))
    Y_OHT = one_hot(y,classes)

    for ix in range(epochs):

        y_ = model.forward(x)
        l = loss(Y_OHT,y_)
        model.backward(x,Y_OHT,learning_rate)
        training_losses.append(l)

        if logs and ix%50==0:
            print("Epoch %d loss %.4f"%(ix,l))

    return training_losses
```

In [30]:
```
model = NeuralNetwork(input_size=4800,layer=[100,50],output_size=3)
```

In [31]:
```
print(x.shape)
```

```
(354, 4800)
```

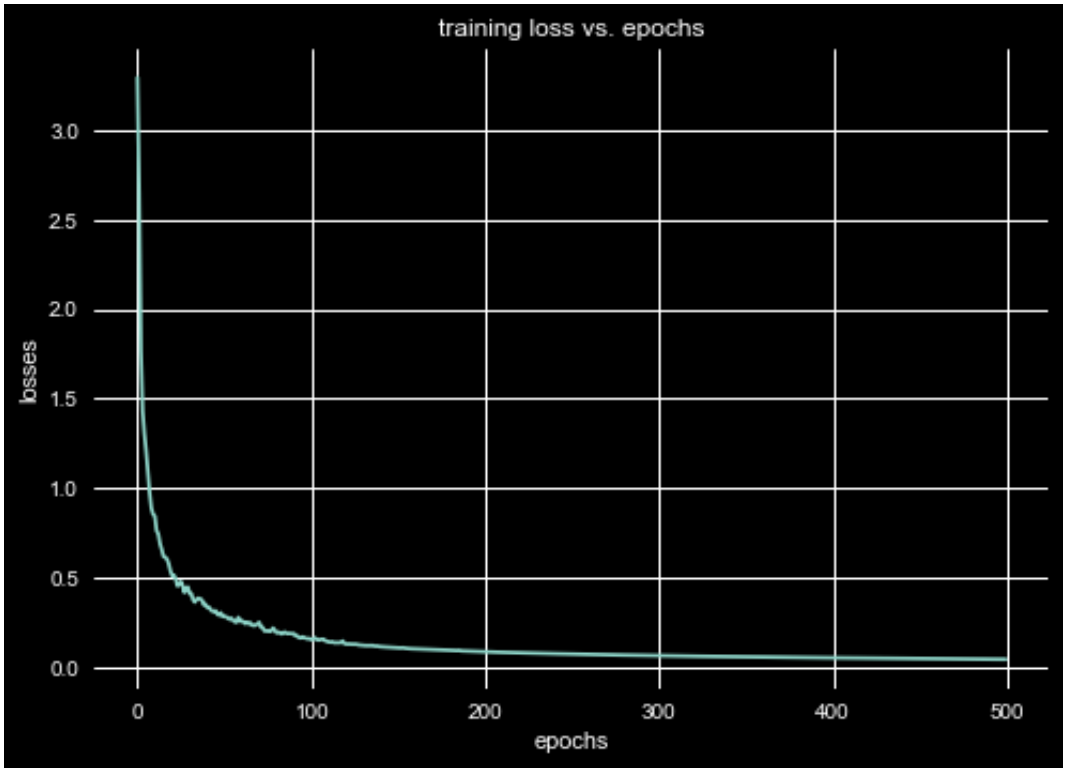
In [32]:
```
x = x.reshape(x.shape[0],-1)
print(x.shape)
xtest = xtest.reshape(xtest.shape[0],-1)
print(xtest.shape)
```

```
(354, 4800)
(89, 4800)
```

In [33]:
```
losses = train(x,y,model,500,0.0002)
```

```
(354, 3)
Epoch 0 loss 3.2952
Epoch 50 loss 0.2811
Epoch 100 loss 0.1500
Epoch 150 loss 0.1046
Epoch 200 loss 0.0824
Epoch 250 loss 0.0697
Epoch 300 loss 0.0606
Epoch 350 loss 0.0540
Epoch 400 loss 0.0487
Epoch 450 loss 0.0439
```

In [34]:
```
plt.style.use("dark_background")
plt.title("training loss vs. epochs")
plt.xlabel("epochs")
plt.ylabel("losses")
plt.plot(losses)
plt.show()
```



finding the accuracy....

In [35]:
```
def getAccuracy(x,y,model):
    outputs = model.predict(x)
    acc = np.sum(outputs==y)/y.shape[0]
    return acc
```

In [36]:
```
print("Train Acc is %.4f"%getAccuracy(x,y,model))
print("Test Acc is %.4f"%getAccuracy(xtest,ytest,model))
```

```
Train Acc is 0.9633
Test Acc is 0.6629
```

In [ ]:
```

```