# LAB 10

### Try CNN on "Fruit" dataset. Also modify number of layers and observe the performance difference:

### [https://www.kaggle.com/moltean/fruits (https://www.kaggle.com/moltean/fruits)](https://www.kaggle.com/moltean/fruits)

### Or (In a case if you don't have that much dataPack available, download 20 images of apple and 20 images of orange from the internet and work on it with RANDOM state=Rollnumber stratergy, 80-20% training-testing division)

In [2]:
```
# Import libraries
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
from torch.autograd import Variable
from torch.utils.data import DataLoader
from sklearn.datasets import load_files
from sklearn.model_selection import train_test_split
```

In [3]:
```
# Load Data Directory
data_dir = '/home/nihar/Desktop/SEM 7/ML/Lab/Lab9/sample-fruits-360'
```

In [4]:
```
# Function for load images
def load_dataset(path):
    data = load_files(path)
    files = np.array(data['filenames'])
    targets = np.array(data['target'])
    target_labels = np.array(data['target_names'])
    return files, targets, target_labels
```

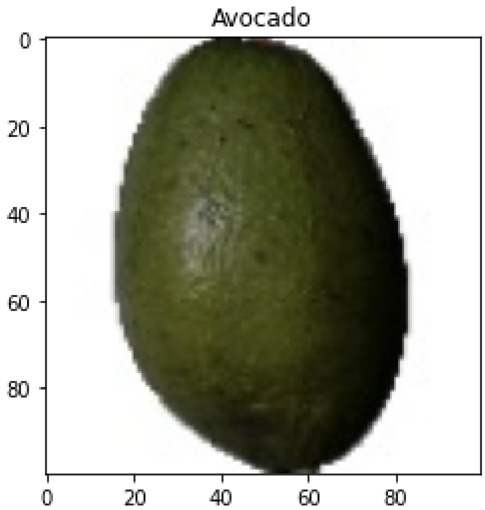
In [5]:
```
# Load Dataset
x, y, target_labels = load_dataset(data_dir)
print("Dataset Loaded !")

# Get Trainning size and Test size
print('Total set size : ',x.shape)
print('Total targets : ',len(target_labels) )
```

```
Dataset Loaded !
Total set size :  (975,)
Total targets :  65
```

In [6]:
```
# Function for convert image to array
def convert_image_to_array(files):
    images_as_array=[]
    for file in files:
        images_as_array.append(keras.preprocessing.image.img_to_array(keras.
preprocessing.image.load_img(file)))
    return images_as_array

# Convert images to numpy array using keras.preprocessing library
x = np.array(convert_image_to_array(x),np.float32)
print(x.shape)
```

```
(975, 100, 100, 3)
```

In [7]:
```
# Plot image on random data
plt.imshow(x[1]/255)
plt.title(target_labels[y[1]])
plt.show()
```




In [8]:
```
# Flatten the features of image
x = x.reshape([-1,100*100*3])
x = x/255
print("final shape : " , x.shape)
```

```
final shape :  (975, 30000)
```

In [9]:
```
# Train and Test split
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_st
ate=129)

# Get size of all set
print("X Train size : ", X_train.shape)
print("X Test size : ", X_test.shape)
print("Y Train size : ", y_train.shape)
print("Y Test size : ", y_test.shape)
```

```
X Train size :  (780, 30000)
X Test size :  (195, 30000)
Y Train size :  (780,)
Y Test size :  (195,)
```

In [10]:
```
# Convert numpy array to torch
X_train = torch.from_numpy(X_train)
y_train = torch.from_numpy(y_train).type(torch.LongTensor)
X_test = torch.from_numpy(X_test)
y_test = torch.from_numpy(y_test).type(torch.LongTensor)
```

In [11]:
```
# Define no of iteration, batch size, num_epochs
batch_size=100
n_iters = 1000
num_epochs = n_iters / (len(X_train) / batch_size)
num_epochs = int(num_epochs)
```

In [12]:
```
# Set train and test
train = torch.utils.data.TensorDataset(X_train,y_train)
test = torch.utils.data.TensorDataset(X_test,y_test)
train_loader = DataLoader(train, batch_size = batch_size, shuffle = False)
test_loader = DataLoader(test, batch_size = batch_size, shuffle = False)
```

### CNN with 2 convolutional layer and 1 fully connected layer

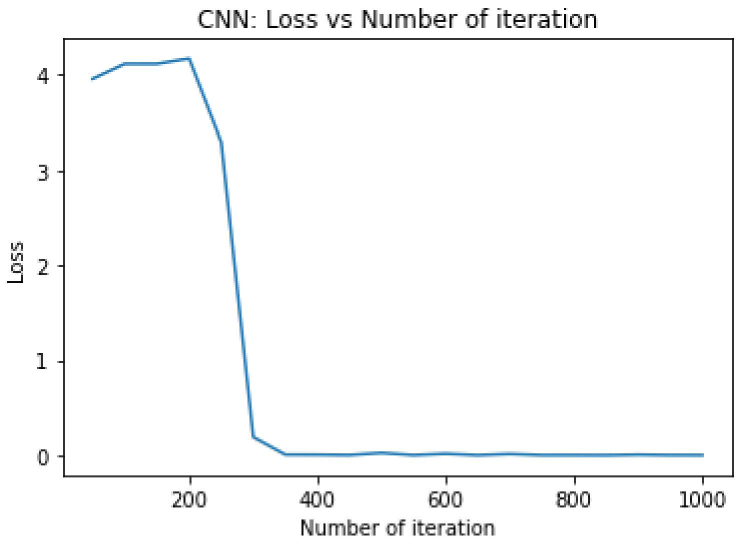
In [13]:
```
# Create CNN Model
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()

        self.cnn1 = nn.Conv2d(in_channels=3,out_channels=16,kernel_size=5,st
ride=1,padding=0)
        self.relu1 = nn.ReLU()

        self.maxpool1 = nn.MaxPool2d(kernel_size=2)

        self.cnn2 = nn.Conv2d(in_channels=16,out_channels=32,kernel_size=5,s
tride=1,padding=0)
        self.relu2 = nn.ReLU()

        self.maxpool2 = nn.MaxPool2d(kernel_size=2)

        self.fc1 = nn.Linear(15488,len(target_labels));

    def forward(self,x):

        out=self.cnn1(x)
        out=self.relu1(out)
        out=self.maxpool1(out)

        out=self.cnn2(out)
        out=self.relu2(out)
        out=self.maxpool2(out)

        out = out.view(out.size(0), -1)
        out=self.fc1(out)

        return out
```

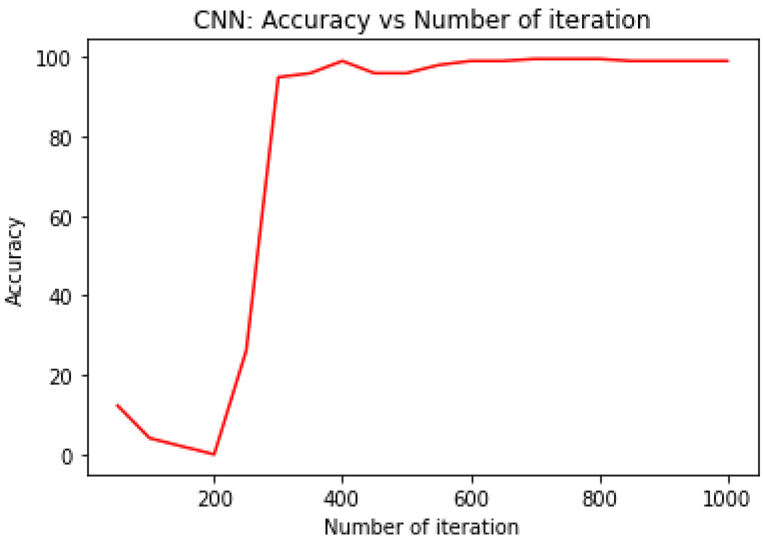
In [14]:
```
# Initialize Parameters and fit the model
model = CNNModel()
error = nn.CrossEntropyLoss()
learning_rate = 0.1
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

In [15]:
```python
# CNN model training
count = 0
loss_list = []
iteration_list = []
accuracy_list = []
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        train = Variable(images.view(-1,3,100,100))
        #print(train.shape)
        labels = Variable(labels)
        optimizer.zero_grad()
        outputs = model(train)
        loss = error(outputs, labels)
        loss.backward()
        optimizer.step()
        count += 1

        if count % 50 == 0:
            correct = 0
            total = 0
            for images, labels in test_loader:
                test = Variable(images.view(-1,3,100,100))
                outputs = model(test)
                predicted = torch.max(outputs.data, 1)[1]
                total += len(labels)
                correct += (predicted == labels).sum()
            accuracy = 100 * correct / float(total)
            loss_list.append(loss.data)
            iteration_list.append(count)
            accuracy_list.append(accuracy)
            if count % 5 == 0:
                print('Iteration: {}  Loss: {}  Accuracy: {} %'.format(count
, loss.data, accuracy))
```

```
Iteration: 50  Loss: 3.9584412574768066  Accuracy: 12.307692527770996 %
Iteration: 100  Loss: 4.116231918334961  Accuracy: 4.102564334869385 %
Iteration: 150  Loss: 4.11642599105835  Accuracy: 2.0512821674346924 %
Iteration: 200  Loss: 4.173018455505371  Accuracy: 0.0 %
Iteration: 250  Loss: 3.294015884399414  Accuracy: 26.153846740722656 %
Iteration: 300  Loss: 0.19131530821323395  Accuracy: 94.87179565429688 %
Iteration: 350  Loss: 0.006540663074702025  Accuracy: 95.8974380493164 %
Iteration: 400  Loss: 0.005745238158851862  Accuracy: 98.97435760498047 %
Iteration: 450  Loss: 0.0023304293863475323  Accuracy: 95.8974380493164 %
Iteration: 500  Loss: 0.02435666136443615  Accuracy: 95.8974380493164 %
Iteration: 550  Loss: 0.0016752103110775352  Accuracy: 97.94871520996094 %
Iteration: 600  Loss: 0.01584899052977562  Accuracy: 98.97435760498047 %
Iteration: 650  Loss: 0.0010343559551984072  Accuracy: 98.97435760498047 %
Iteration: 700  Loss: 0.013168991543352604  Accuracy: 99.4871826171875 %
Iteration: 750  Loss: 0.001453787088394165  Accuracy: 99.4871826171875 %
Iteration: 800  Loss: 0.0012704887194558978  Accuracy: 99.4871826171875 %
Iteration: 850  Loss: 0.00036563488538376987  Accuracy: 98.97435760498047 %
Iteration: 900  Loss: 0.005948456469923258  Accuracy: 98.97435760498047 %
Iteration: 950  Loss: 0.0012212992878630757  Accuracy: 98.97435760498047 %
Iteration: 1000  Loss: 0.0007654182845726609  Accuracy: 98.97435760498047 %
```

In [16]:
```
# visualization loss
plt.plot(iteration_list,loss_list)
plt.xlabel("Number of iteration")
plt.ylabel("Loss")
plt.title("CNN: Loss vs Number of iteration")
plt.show()

# visualization accuracy
plt.plot(iteration_list,accuracy_list,color = "red")
plt.xlabel("Number of iteration")
plt.ylabel("Accuracy")
plt.title("CNN: Accuracy vs Number of iteration")
plt.show()
```




## CNN with 3 convolutional layer and 1 fully connected

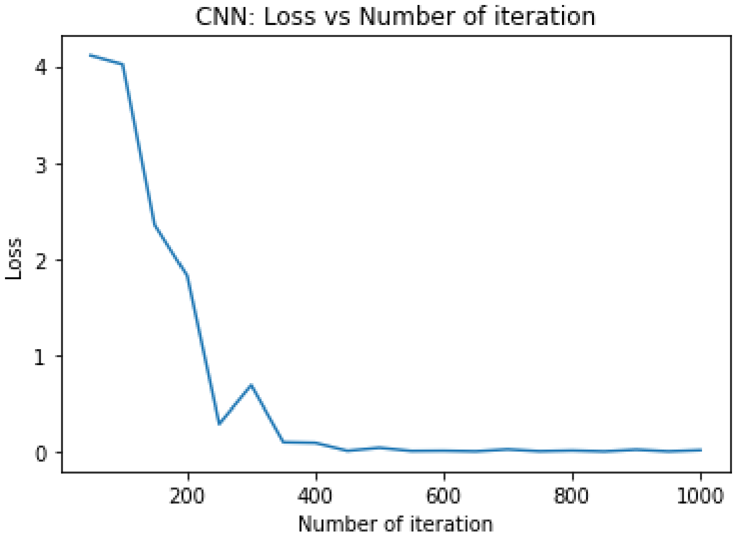
In [17]:
```python
# Create CNN Model
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()

        self.cnn1 = nn.Conv2d(in_channels=3,out_channels=16,kernel_size=5,stride=1,padding=0)
        self.relu1 = nn.ReLU()

        self.maxpool1 = nn.MaxPool2d(kernel_size=2)

        self.cnn2 = nn.Conv2d(in_channels=16,out_channels=32,kernel_size=5,stride=1,padding=0)
        self.relu2 = nn.ReLU()

        self.maxpool2 = nn.MaxPool2d(kernel_size=2)

        self.cnn3 = nn.Conv2d(in_channels=32,out_channels=64,kernel_size=5,stride=1,padding=0)
        self.relu3 = nn.ReLU()

        self.maxpool3 = nn.MaxPool2d(kernel_size=2)

        self.fc1 = nn.Linear(5184,len(target_labels));

    def forward(self,x):

        out=self.cnn1(x)
        out=self.relu1(out)
        out=self.maxpool1(out)

        out=self.cnn2(out)
        out=self.relu2(out)
        out=self.maxpool2(out)

        out=self.cnn3(out)
        out=self.relu3(out)
        out=self.maxpool3(out)

        out = out.view(out.size(0), -1)
        out = self.fc1(out)

        return out
```

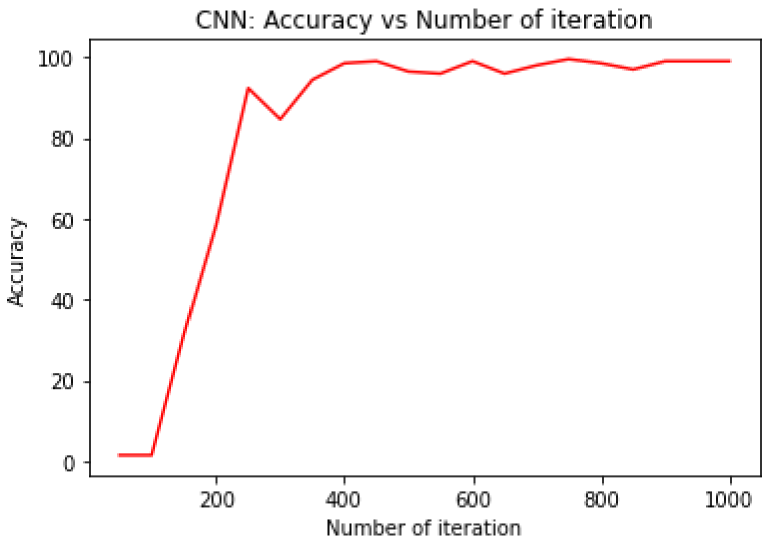
In [18]:
```python
# Initialize Parameters and fit the model
model = CNNModel()
error = nn.CrossEntropyLoss()
learning_rate = 0.02
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

In [19]:
```
# CNN model training
count = 0
loss_list = []
iteration_list = []
accuracy_list = []
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        train = Variable(images.view(-1,3,100,100))
        labels = Variable(labels)
        optimizer.zero_grad()
        outputs = model(train)
        loss = error(outputs, labels)
        loss.backward()
        optimizer.step()
        count += 1

        if count % 50 == 0:
            correct = 0
            total = 0
            for images, labels in test_loader:
                test = Variable(images.view(-1,3,100,100))
                outputs = model(test)
                predicted = torch.max(outputs.data, 1)[1]
                total += len(labels)
                correct += (predicted == labels).sum()
            accuracy = 100 * correct / float(total)
            loss_list.append(loss.data)
            iteration_list.append(count)
            accuracy_list.append(accuracy)
            if count % 5 == 0:
                print('Iteration: {}  Loss: {}  Accuracy: {} %'.format(count
, loss.data, accuracy))
```

```
Iteration: 50  Loss: 4.111763000488281  Accuracy: 1.5384615659713745 %
Iteration: 100  Loss: 4.019026279449463  Accuracy: 1.5384615659713745 %
Iteration: 150  Loss: 2.353355646133423  Accuracy: 31.28205108642578 %
Iteration: 200  Loss: 1.8257973194122314  Accuracy: 58.46154022216797 %
Iteration: 250  Loss: 0.28569337725639343  Accuracy: 92.30769348144531 %
Iteration: 300  Loss: 0.6871101260185242  Accuracy: 84.61538696289062 %
Iteration: 350  Loss: 0.09699603170156479  Accuracy: 94.35897064208984 %
Iteration: 400  Loss: 0.09078714996576309  Accuracy: 98.46154022216797 %
Iteration: 450  Loss: 0.007882218807935715  Accuracy: 98.97435760498047 %
Iteration: 500  Loss: 0.03974886238574982  Accuracy: 96.4102554321289 %
Iteration: 550  Loss: 0.006276494357734919  Accuracy: 95.8974380493164 %
Iteration: 600  Loss: 0.008571005426347256  Accuracy: 98.97435760498047 %
Iteration: 650  Loss: 0.002602524124085903  Accuracy: 95.8974380493164 %
Iteration: 700  Loss: 0.022816266864538193  Accuracy: 97.94871520996094 %
Iteration: 750  Loss: 0.0035299521405249834  Accuracy: 99.4871826171875 %
Iteration: 800  Loss: 0.010895995423197746  Accuracy: 98.46154022216797 %
Iteration: 850  Loss: 0.001701981294900179  Accuracy: 96.92308044433594 %
Iteration: 900  Loss: 0.019954847171902657  Accuracy: 98.97435760498047 %
Iteration: 950  Loss: 0.0021584550850093365  Accuracy: 98.97435760498047 %
Iteration: 1000  Loss: 0.014821499586105347  Accuracy: 98.97435760498047 %
```

In [20]:
```python
# visualization loss
plt.plot(iteration_list,loss_list)
plt.xlabel("Number of iteration")
plt.ylabel("Loss")
plt.title("CNN: Loss vs Number of iteration")
plt.show()

# visualization accuracy
plt.plot(iteration_list,accuracy_list,color = "red")
plt.xlabel("Number of iteration")
plt.ylabel("Accuracy")
plt.title("CNN: Accuracy vs Number of iteration")
plt.show()
```




**As we can see from above two model , if first I take 2 convolutional layer and 1 full connected layer with learning rate 0.1, next I take 3 convolutional layer and 1 fully connected layer with learning rate 0.02, keeping all parameters same I get good performance but If I take other learning rate with different numbers of convolutinal layer and fully connected layer then I get bad performance or underfitting model.**