In [1]:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from matplotlib import pyplot as plt

from google.colab import drive
drive.mount('sample_data/drive')
```

```
Mounted at sample_data/drive
```

In [ ]:

```
!unzip "sample_data/drive/MyDrive/Colab Notebooks/Lab_3/flowers6.zip"  -d "sample_data/dri
ve/MyDrive/Colab Notebooks/Lab_3"
```

```
Archive:  sample_data/drive/MyDrive/Colab Notebooks/Lab_3/flowers6.zip
replace sample_data/drive/MyDrive/Colab Notebooks/Lab_3/flowers6/test/buttercup/image_1194.
jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: N
```

In [2]:

```
batch_size_train = 32
batch_size_test = 32
learning_rate = .006
epochs = 100
```

In [ ]:

```

```

In [3]:

```
train_dir = "sample_data/drive/MyDrive/Colab Notebooks/Lab_3/flowers6/train"
test_dir = "sample_data/drive/MyDrive/Colab Notebooks/Lab_3/flowers6/test"

train_transforms = transforms.Compose([transforms.RandomRotation(30),
                                       transforms.RandomResizedCrop(224),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.ToTensor(),
                                       transforms.Normalize((0.485,0.456,0.406),
                                                            (0.229,0.224,0.225))])

test_transforms = transforms.Compose([transforms.Resize(256),
                                      transforms.CenterCrop(224),
                                      transforms.ToTensor(),
                                      transforms.Normalize((0.485,0.456,0.406),
                                                           (0.229,0.224,0.225))])

train_data = datasets.ImageFolder(train_dir, transform = train_transforms)
test_data = datasets.ImageFolder(test_dir, transform = test_transforms)

train_loader = torch.utils.data.DataLoader(train_data, batch_size = batch_size_train, shuf
fle = True)
test_loader = torch.utils.data.DataLoader(test_data, batch_size = batch_size_test, shuffle
= True)
```

In [4]:

```
class CNN(nn.Module):

  def __init__(self):
      super().__init__()

      self.conv1 = nn.Sequential(
          nn.Conv2d(3,64, kernel_size = 3, padding = 1),
          nn.ReLU(),
          nn.MaxPool2d(2),
      )

      self.conv2 = nn.Sequential(
          nn.Conv2d(64,128, kernel_size = 3, padding = 1),
          nn.ReLU(),
          nn.MaxPool2d(2),
      )

      self.conv3 = nn.Sequential(
          nn.Conv2d(128,256, kernel_size = 3, padding = 1),
          nn.ReLU(),
          nn.MaxPool2d(2),
      )


      self.conv4 = nn.Sequential(
          nn.Conv2d(256,512, kernel_size = 3, padding = 1),
          nn.ReLU(),
          nn.MaxPool2d(2),
      )

      self.fc = nn.Sequential(
          nn.Linear(14*14*512, 1000),
          nn.ReLU(inplace = True),
          nn.Linear(1000,1000),
          nn.ReLU(inplace = True),
          nn.Linear(1000,6),
      )

  def forward(self, x):
    x = self.conv1(x)
    x = self.conv2(x)
    x = self.conv3(x)
    x = self.conv4(x)
    x = x.view(x.size(0), -1)
    x = self.fc(x)
    return x
```

In [5]:

```
device = torch.device('cuda:0')
net = CNN().to(device)

print(net)
optimizer = optim.SGD(net.parameters(), lr = learning_rate, momentum = .9)
criteon = nn.CrossEntropyLoss().to(device)
```

```
CNN(
  (conv1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc): Sequential(
    (0): Linear(in_features=100352, out_features=1000, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=1000, out_features=1000, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=1000, out_features=6, bias=True)
  )
)
```

In [6]:

```python
for epoch in range(epochs):
  train_loss = 0
  train_acc = 0

  for data, target in train_loader:

    data, target = data.to(device), target.to(device)
    logits = net(data)
    loss = criteon(logits, target)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    _,pred = logits.max(1)
    num_correct = (pred==target).sum().item()
    acc = num_correct / data.shape[0]
    train_acc += acc
    train_loss += loss.data

  print("Train Epoch: {} Loss: {:.6f} Training Accuracy {:.6f}".format(epoch+1, train_loss
/len(train_loader), train_acc/len(train_loader)))
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:718: UserWarning: Named tenso
rs and all their associated APIs are an experimental feature and subject to change. Please
do not use them for anything important until they are released as stable. (Triggered intern
ally at  /pytorch/c10/core/TensorImpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)
```

```
Train Epoch: 1 Loss: 1.787632 Training Accuracy 0.186688
Train Epoch: 2 Loss: 1.737386 Training Accuracy 0.358360
Train Epoch: 3 Loss: 1.391102 Training Accuracy 0.359781
Train Epoch: 4 Loss: 1.311032 Training Accuracy 0.383117
Train Epoch: 5 Loss: 1.219138 Training Accuracy 0.388596
Train Epoch: 6 Loss: 1.155427 Training Accuracy 0.461242
Train Epoch: 7 Loss: 1.223086 Training Accuracy 0.462054
Train Epoch: 8 Loss: 1.243663 Training Accuracy 0.436485
Train Epoch: 9 Loss: 1.011380 Training Accuracy 0.540990
Train Epoch: 10 Loss: 1.107696 Training Accuracy 0.509943
Train Epoch: 11 Loss: 0.927240 Training Accuracy 0.584010
Train Epoch: 12 Loss: 0.828141 Training Accuracy 0.630682
Train Epoch: 13 Loss: 0.833163 Training Accuracy 0.627841
Train Epoch: 14 Loss: 0.793864 Training Accuracy 0.663352
Train Epoch: 15 Loss: 0.765739 Training Accuracy 0.661323
Train Epoch: 16 Loss: 0.679900 Training Accuracy 0.681818
Train Epoch: 17 Loss: 0.678440 Training Accuracy 0.683644
Train Epoch: 18 Loss: 0.663602 Training Accuracy 0.699269
Train Epoch: 19 Loss: 0.623527 Training Accuracy 0.732955
Train Epoch: 20 Loss: 0.615511 Training Accuracy 0.738231
Train Epoch: 21 Loss: 0.608164 Training Accuracy 0.727070
Train Epoch: 22 Loss: 0.607898 Training Accuracy 0.760552
Train Epoch: 23 Loss: 0.567824 Training Accuracy 0.765219
Train Epoch: 24 Loss: 0.555227 Training Accuracy 0.759131
Train Epoch: 25 Loss: 0.502841 Training Accuracy 0.808239
Train Epoch: 26 Loss: 0.524409 Training Accuracy 0.792817
Train Epoch: 27 Loss: 0.534170 Training Accuracy 0.778409
Train Epoch: 28 Loss: 0.507639 Training Accuracy 0.799513
Train Epoch: 29 Loss: 0.471292 Training Accuracy 0.820414
Train Epoch: 30 Loss: 0.499150 Training Accuracy 0.799513
Train Epoch: 31 Loss: 0.507312 Training Accuracy 0.839692
Train Epoch: 32 Loss: 0.474931 Training Accuracy 0.839489
Train Epoch: 33 Loss: 0.462745 Training Accuracy 0.845170
Train Epoch: 34 Loss: 0.367359 Training Accuracy 0.849635
Train Epoch: 35 Loss: 0.424208 Training Accuracy 0.846185
Train Epoch: 36 Loss: 0.431620 Training Accuracy 0.823255
Train Epoch: 37 Loss: 0.401054 Training Accuracy 0.836242
Train Epoch: 38 Loss: 0.476120 Training Accuracy 0.834213
Train Epoch: 39 Loss: 0.578674 Training Accuracy 0.777597
Train Epoch: 40 Loss: 0.504952 Training Accuracy 0.822849
Train Epoch: 41 Loss: 0.371539 Training Accuracy 0.837459
Train Epoch: 42 Loss: 0.441138 Training Accuracy 0.827110
Train Epoch: 43 Loss: 0.391585 Training Accuracy 0.860998
Train Epoch: 44 Loss: 0.400659 Training Accuracy 0.844968
Train Epoch: 45 Loss: 0.387436 Training Accuracy 0.849635
Train Epoch: 46 Loss: 0.421499 Training Accuracy 0.859781
Train Epoch: 47 Loss: 0.396071 Training Accuracy 0.851664
Train Epoch: 48 Loss: 0.369274 Training Accuracy 0.855317
Train Epoch: 49 Loss: 0.404373 Training Accuracy 0.840706
Train Epoch: 50 Loss: 0.351063 Training Accuracy 0.878450
Train Epoch: 51 Loss: 0.332931 Training Accuracy 0.884131
Train Epoch: 52 Loss: 0.410611 Training Accuracy 0.845373
Train Epoch: 53 Loss: 0.365093 Training Accuracy 0.865463
Train Epoch: 54 Loss: 0.318837 Training Accuracy 0.889813
Train Epoch: 55 Loss: 0.274144 Training Accuracy 0.903206
Train Epoch: 56 Loss: 0.354248 Training Accuracy 0.887378
Train Epoch: 57 Loss: 0.354453 Training Accuracy 0.870942
Train Epoch: 58 Loss: 0.320849 Training Accuracy 0.878653
Train Epoch: 59 Loss: 0.335198 Training Accuracy 0.871753
Train Epoch: 60 Loss: 0.325188 Training Accuracy 0.876623
Train Epoch: 61 Loss: 0.246846 Training Accuracy 0.898742
Train Epoch: 62 Loss: 0.296158 Training Accuracy 0.889610
Train Epoch: 63 Loss: 0.301152 Training Accuracy 0.894481
Train Epoch: 64 Loss: 0.461608 Training Accuracy 0.830763
Train Epoch: 65 Loss: 0.366020 Training Accuracy 0.847606
Train Epoch: 66 Loss: 0.463243 Training Accuracy 0.791599
Train Epoch: 67 Loss: 0.400920 Training Accuracy 0.823052
Train Epoch: 68 Loss: 0.382613 Training Accuracy 0.857346
Train Epoch: 69 Loss: 0.365740 Training Accuracy 0.855317
Train Epoch: 70 Loss: 0.341648 Training Accuracy 0.878450
Train Epoch: 71 Loss: 0.323712 Training Accuracy 0.879667
Train Epoch: 72 Loss: 0.345939 Training Accuracy 0.877435
Train Epoch: 73 Loss: 0.287535 Training Accuracy 0.900771
Train Epoch: 74 Loss: 0.294239 Training Accuracy 0.896307
Train Epoch: 75 Loss: 0.255426 Training Accuracy 0.883320
Train Epoch: 76 Loss: 0.313188 Training Accuracy 0.865463
Train Epoch: 77 Loss: 0.314263 Training Accuracy 0.897321
Train Epoch: 78 Loss: 0.251040 Training Accuracy 0.919846
Train Epoch: 79 Loss: 0.234626 Training Accuracy 0.898539
Train Epoch: 80 Loss: 0.219078 Training Accuracy 0.909700
Train Epoch: 81 Loss: 0.274910 Training Accuracy 0.903409
Train Epoch: 82 Loss: 0.420239 Training Accuracy 0.835633
Train Epoch: 83 Loss: 0.344406 Training Accuracy 0.888393
Train Epoch: 84 Loss: 0.294478 Training Accuracy 0.891843
Train Epoch: 85 Loss: 0.242416 Training Accuracy 0.907468
Train Epoch: 86 Loss: 0.208982 Training Accuracy 0.911932
Train Epoch: 87 Loss: 0.259192 Training Accuracy 0.912946
Train Epoch: 88 Loss: 0.262358 Training Accuracy 0.909700
Train Epoch: 89 Loss: 0.239718 Training Accuracy 0.912338
Train Epoch: 90 Loss: 0.200386 Training Accuracy 0.934456
Train Epoch: 91 Loss: 0.198848 Training Accuracy 0.920860
Train Epoch: 92 Loss: 0.256305 Training Accuracy 0.922078
Train Epoch: 93 Loss: 0.287760 Training Accuracy 0.899554
Train Epoch: 94 Loss: 0.312683 Training Accuracy 0.893060
Train Epoch: 95 Loss: 0.269023 Training Accuracy 0.906453
Train Epoch: 96 Loss: 0.194007 Training Accuracy 0.934253
Train Epoch: 97 Loss: 0.199188 Training Accuracy 0.935268
Train Epoch: 98 Loss: 0.260907 Training Accuracy 0.903003
Train Epoch: 99 Loss: 0.236349 Training Accuracy 0.914367
Train Epoch: 100 Loss: 0.191170 Training Accuracy 0.946429
```

In [7]:

```
correct = 0
net.eval()
for data, target in test_loader:
  data, target = data.to(device), target.cuda()
  logits = net(data)

  pred = logits.argmax(dim = 1)
```

```
    correct += pred.eq(target).float().sum().item()

total_num = len(test_loader.dataset)
acc = correct / total_num
print('Test Accuracy: ', acc)
```

```
Test Accuracy:  0.8809523809523809
```

In [8]:

```
import numpy as np

def plot_image(img, prediction, label):

  fig = plt.figure()
  for i in range(6):
    plt.imshow(np.transpose(img[i].reshape((3,224,224)), (1,2,0)))
    plt.title("Prediction = {} Label = {}".format(prediction[i].item(), label[i].item()))
    plt.show()
```

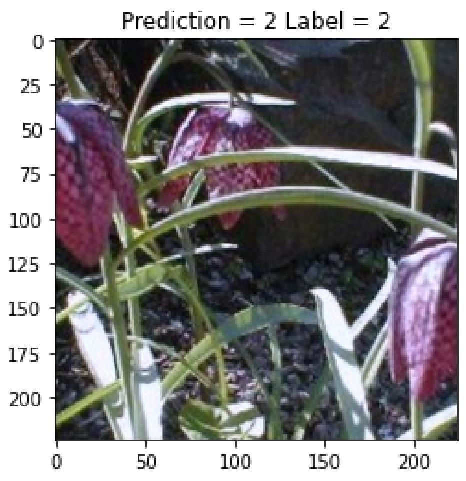
In [9]:

```
inv_normalize = transforms.Normalize(
    mean = [-0.485/0.229, -0.456/0.224, -0.406/0.225],
    std = [1/0.229, 1/0.224, 1/0.255]
)
```

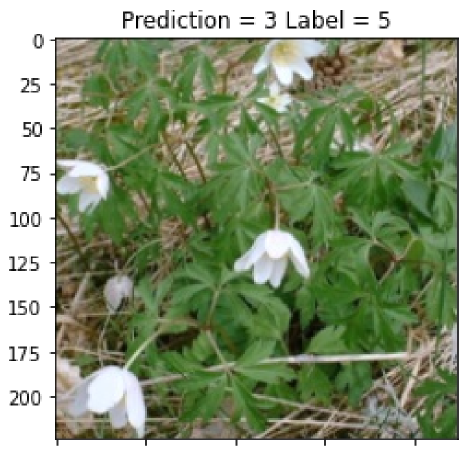
In [10]:

```
x, y = next(iter(test_loader))
x, y = x.to(device), y.cuda()
out = net(x)
pred = out.argmax(dim = 1)
x = inv_normalize(x)
plot_image(x.detach().cpu().numpy(), pred, y)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
55] for integers).
```



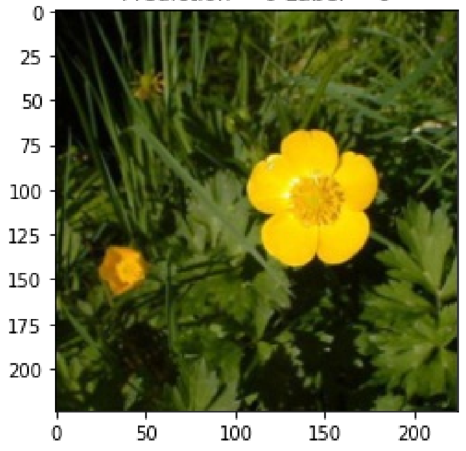
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
55] for integers).
```



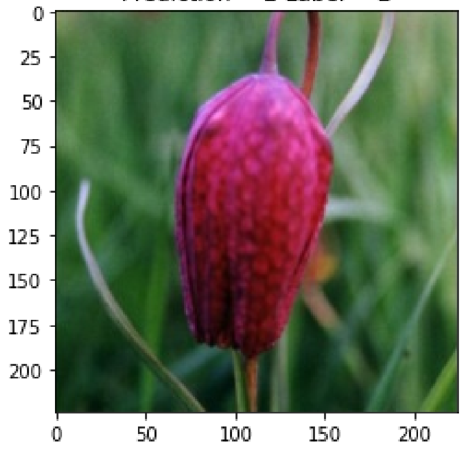
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
55] for integers).
```

Prediction = 3 Label = 3

Prediction = 0 Label = 0



Prediction = 2 Label = 2



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
55] for integers).
```

Prediction = 1 Label = 1