In [1]:
```
import numpy as np
import random
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader,TensorDataset
from PIL import Image
import torchvision.transforms.functional as TF
import torchvision.models as models
import torchvision.datasets as datasets
import split_folders
import matplotlib.pyplot as plt
import os
```

In [2]:
```
class CNNNet(nn.Module):
    def __init__(self,):
        super().__init__()

        self.conv1 = nn.Conv2d(3,16,kernel_size = 3,padding = 1)
        self.act1 = nn.ReLU()
        self.bn1 = nn.BatchNorm2d(16)
        self.pool1 = nn.MaxPool2d(2)
        self.conv2 = nn.Conv2d(16,8,kernel_size = 3,padding = 2)
        self.act2 = nn.ReLU()
        self.bn2 = nn.BatchNorm2d(8)
        self.pool2 = nn.MaxPool2d(2)
        self.fc1 =  nn.Linear(25992,1024)
        self.act3 = nn.ReLU()
        self.out = nn.Linear(1024,102)

    def forward(self,x):
        out = self.pool1(self.act1(self.conv1(x)))
        out = self.bn1(out)
        out = self.pool2(self.act2(self.conv2(out)))
        out = self.bn2(out)
        out = out.view(out.size(0),-1)
        out = self.act3(self.fc1(out))
        out = self.out(out)
        return out
```

I chose these specific layers after experimenting a LOT with Batch Norms and deeper Networks. It turns out that Batchnorm layers significantly affect the performance of the model, along with the slight regularization effect they provide.

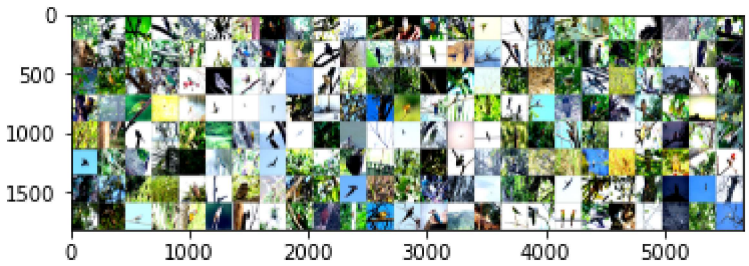
In [3]:
```
transformations = transforms.Compose([transforms.Resize((224,224)),transforms.
ToTensor()
                                      ,transforms.Normalize(mean = [0.485,0.45
6,0.406],std = [0.229,0.224,0.225])])
```

In [4]:
```
train_set = datasets.ImageFolder("C:\\Users\\heman\\Desktop\\fbird_trnval\\tra
in", transform = transformations)
val_set = datasets.ImageFolder("C:\\Users\\heman\\Desktop\\fbird_trnval\\val",
transform = transformations)
```

In [5]:
```
train_loader = DataLoader(train_set,batch_size = 200,shuffle = True)
val_loader = DataLoader(train_set,batch_size = 100,shuffle = True)
```

In [6]:
```
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(train_loader)
images, labels = dataiter.next()

# show images
imshow(torchvision.utils.make_grid(images,nrow = 25))
# print labels
print(labels)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```




```
tensor([12, 85,  8, 41, 32, 44, 29, 80, 56, 66, 61, 77, 76, 35, 60, 39, 39, 8
2,
        83, 85, 81, 34, 79, 60, 10,  5, 31, 87,  3, 29, 77, 37, 33, 74, 30, 6
5,
         1, 40, 39, 24,  8, 37, 85, 41, 49, 45, 41, 26, 28, 88, 29, 82, 80, 6
4,
        40, 20, 77, 65, 37, 82, 12, 70, 29, 15, 88, 70, 10, 19, 77, 34, 81, 5
1,
        31, 61, 19, 52, 83, 45, 69, 78, 28, 63, 28, 87, 17, 85, 57, 77, 37, 4
6,
        15, 31, 41, 16, 37, 58, 31, 77, 24, 77, 85, 54, 75, 80, 85, 36, 57, 5
8,
        22, 86, 36, 46, 22, 45, 58, 39, 46, 29, 55, 32, 26, 44,  2, 27, 78, 3
4,
        41, 55, 63, 77, 39,  0, 63,  5, 75, 41, 33, 75, 81, 60, 82, 27, 31, 7
6,
        40, 73, 40,  2, 77, 69, 83, 83, 77, 69, 59, 46, 11, 74, 50, 87, 26, 7
3,
        78, 85, 57, 24, 52, 82, 28, 82, 88, 55,  2, 26, 62, 44, 31, 68, 72, 5
5,
        71, 48, 31, 56, 56, 55, 81, 62, 37, 26, 11, 39, 61, 38,  3, 41, 61, 4
1,
        62,  8])
```

The above is a snapshot of the data, as the data is too large to be uploaded to this repository.

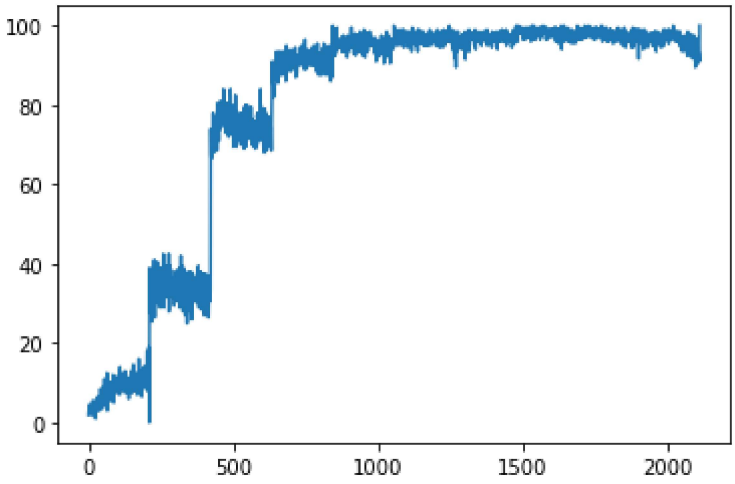
In [7]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNNNet()
```

In [8]:
```
model = model.to(device)
tr_losshist = []#----------------- Training Loss
tr_acchist = []#----------------- Training Accuracy
val_acchist = []#----------------- Validation Accuracy
n_epoch = 10
optimizer = optim.Adam(model.parameters(),lr = 1e-3)
crit = nn.CrossEntropyLoss()

for epoch in range(n_epoch):
    model.train()
    for i, (img,lbl) in enumerate(train_loader):
        img,lbl = img.to(device),lbl.to(device)

        out = model(img)
        loss = crit(out,lbl)
        tr_losshist.append(loss.item())
        loss.backward()#---------------- calculating gradients
        optimizer.step()#--------------- updating the weight values
        optimizer.zero_grad()#---------- zeroing the gradients
        _,tr_pred = torch.max(out.data,1)
        crrct = (tr_pred==lbl).sum().item()
        total = lbl.size(0)
        tr_acchist.append(100*crrct/total)

        if((i+1)%100 == 0):
            print('Epoch [{}/{}],  Loss: {:.4f}, Accuracy: {:.2f}%'
                  .format(epoch + 1, n_epoch, loss.item(),
                          (crrct / total) * 100))

    model.eval()
    with torch.no_grad():
        val_totcrrct = 0
        val_total = 0
        for img,lbl in val_loader:
            img,lbl = img.to(device),lbl.to(device)

            out = model(img)

            _,val_pred = torch.max(out.data,1)

            crrct = (val_pred==lbl).sum().item()
            total = lbl.size(0)
            val_acchist.append(100*crrct/total)
            val_totcrrct+=crrct
            val_total+=total

        print("Val accuracy of model :{:.3f} %".format((val_totcrrct/val_total
)*100))
```

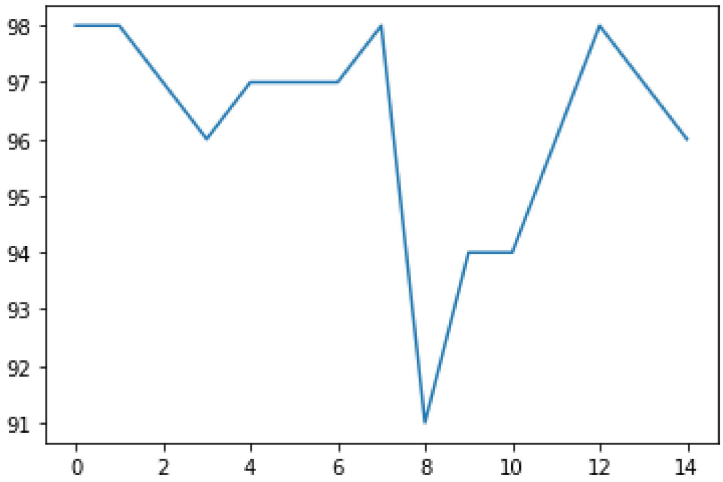
```
Epoch [1/10],  Loss: 3.9652, Accuracy: 8.00%
Epoch [1/10],  Loss: 3.6235, Accuracy: 9.50%
Val accuracy of model :33.217 %
Epoch [2/10],  Loss: 2.6126, Accuracy: 32.50%
Epoch [2/10],  Loss: 2.7994, Accuracy: 33.00%
Val accuracy of model :73.650 %
Epoch [3/10],  Loss: 0.8686, Accuracy: 75.00%
Epoch [3/10],  Loss: 1.1704, Accuracy: 72.00%
Val accuracy of model :90.412 %
Epoch [4/10],  Loss: 0.3657, Accuracy: 90.50%
Epoch [4/10],  Loss: 0.4420, Accuracy: 89.00%
Val accuracy of model :95.361 %
Epoch [5/10],  Loss: 0.1268, Accuracy: 97.00%
Epoch [5/10],  Loss: 0.1422, Accuracy: 95.00%
Val accuracy of model :96.282 %
Epoch [6/10],  Loss: 0.0705, Accuracy: 98.50%
Epoch [6/10],  Loss: 0.0605, Accuracy: 97.50%
Val accuracy of model :97.901 %
Epoch [7/10],  Loss: 0.0749, Accuracy: 98.00%
Epoch [7/10],  Loss: 0.1014, Accuracy: 97.00%
Val accuracy of model :98.262 %
Epoch [8/10],  Loss: 0.0695, Accuracy: 98.00%
Epoch [8/10],  Loss: 0.0749, Accuracy: 97.00%
Val accuracy of model :98.210 %
Epoch [9/10],  Loss: 0.0650, Accuracy: 98.00%
Epoch [9/10],  Loss: 0.1827, Accuracy: 96.00%
Val accuracy of model :97.682 %
Epoch [10/10],  Loss: 0.1312, Accuracy: 97.50%
Epoch [10/10],  Loss: 0.2739, Accuracy: 93.50%
Val accuracy of model :95.389 %
```

In [9]:
```
path = ".\\custcnn.pth"
torch.save(model.state_dict(),path)
```

In [15]:
```
plt.plot(tr_acchist,label = "Training accuracy")
plt.show()
```



In [16]:
```
plt.plot(val_acchist,label = "val Accuracy")
plt.show()
```



In [ ]:
```

```