CS 5787 Assignment 2

Ran Ji

rj369@

Problem 1

Part 1 - Using Pre-Trained Deep CNN (5 points)

Model used: resnet152

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

train_loader = torch.utils.data.DataLoader(
        datasets.ImageFolder("./q1_train_img",
        transforms.Compose([transforms.ToTensor(),normalize,])),
        batch_size=10)

pepper_image = Variable(next(iter(train_loader))[0])

resnet152 = models.resnet152(pretrained=True)
resnet152.eval()

resnet152.conv1.register_forward_hook(get_activation('early'))
resnet152.layer2[1].conv3.register_forward_hook(get_activation('middle'))
resnet152.layer4[1].conv3.register_forward_hook(get_activation('end'))

output = resnet152(pepper_image)

prob = sum(np.exp(output[0].detach().numpy()))
index_file = json.load(open("./imagenet_class_index.txt"))

index_name = [0]*len(index_file)
for i in range(len(index_file)):
    index_name[i] = index_file[str(i)][1]
```

```
output2 = output[0].sort()[1][-3:]
output2
```

```
tensor([831, 943, 945])
```

```
for index in reversed(output2):
    print(index_name[index], end ="")
    print(" probability is ", end ="")
    print(np.exp(output[0][index].detach().numpy())/prob)
```

```
bell_pepper probability is 0.9002557140463503
cucumber probability is 0.08118853944603413
studio_couch probability is 0.005074153097658631
```

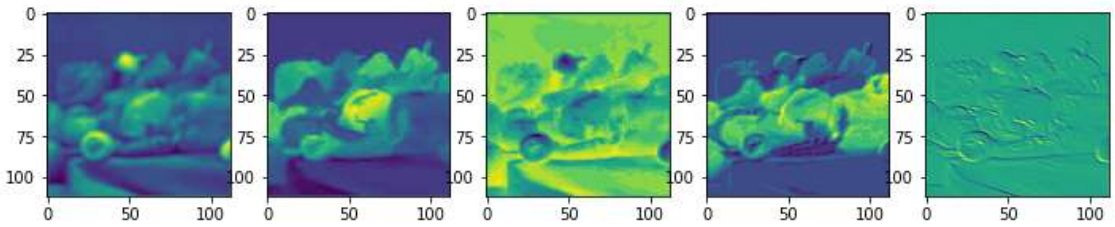
Part 2 - Visualizing Feature Maps (5 points)

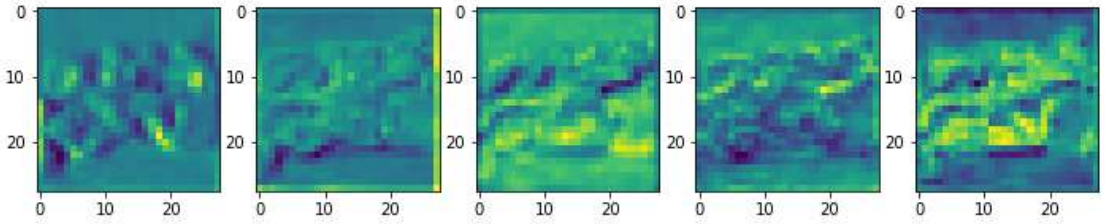
```
bell_pepper probability is 0.9002557140463503
cucumber probability is 0.08118853944603413
studio_couch probability is 0.005074153097658631
```

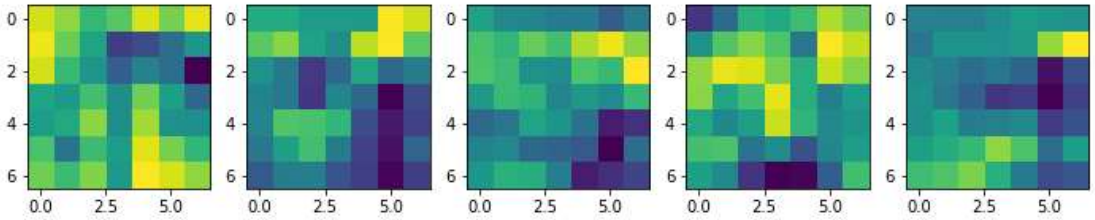
```
print(activation['early'].shape)
print(activation['middle'].shape)
print(activation['end'].shape)
```

```
torch.Size([1, 64, 112, 112])
torch.Size([1, 512, 28, 28])
torch.Size([1, 2048, 7, 7])
```

```
def print_feature(layer):
    layer = layer - layer.min()
    layer = layer / layer.max()
    fig = plt.figure(figsize=(12,8))
    for i in range(5):
        image = layer[i]
        fig.add_subplot(1, 5, i+1)
        plt.imshow(image)
    plt.show()

print_feature(activation['early'][0])
print_feature(activation['middle'][0])
print_feature(activation['end'][0])
```







Problem 2

Extract features:

```
import numpy as np
import json
import torch
import torch.nn as nn
import torch.nn.functional as F
import random
import torch.optim as optim

import torchvision.models as models
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.autograd import Variable
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib

from PIL import Image
from random import shuffle
```

```
content = open("annotations/trainval.txt","r").readlines()
PERCENT = 20

train_data = {}
for line in content:
    if line.strip() == "":
        continue
    if random.randint(1,PERCENT) % PERCENT != 0: # remove
        continue
    name = line.split()[0].strip()
    value = int(line.split()[1].strip())
    train_data[name] = value


content = open("annotations/test.txt","r").readlines()

test_data = {}
for line in content:
    if line.strip() == "":
        continue
#     if random.randint(1,PERCENT) % PERCENT != 0: # remove
#         continue
    name = line.split()[0].strip()
    value = int(line.split()[1].strip())
    test_data[name] = value

```

```
FEATURE_NUM = 2048
CLASS_NUM = 37

train_features = np.zeros((len(train_data), FEATURE_NUM))
train_label = []
train_items = list(train_data.items())

test_features = np.zeros((len(test_data), FEATURE_NUM))
test_label = []
test_items = list(test_data.items())

model = models.resnet152(pretrained=True)
extra_feature_model = nn.Sequential(*list(model.children()))[:-1]
```

```python
def extract_features(is_train):
    global model

    batch_size = 20
    items = None
    features = None
    labels = None

    if (is_train):
        items = train_items
        features = train_features
        labels = train_label
    else:
        items = test_items
        features = test_features
        labels = test_label

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

    for i in range(0, len(items), batch_size):
        current_items = items[i: i+batch_size]
        batch_input = None
        for name, value in current_items:
            path = "images/"+name+".jpg"
            image = Image.open(path).convert('RGB')
            labels.append(value)
            tensor = trans(image).unsqueeze(0)
            if batch_input is None:
                batch_input = tensor
            else:
                batch_input = torch.cat((batch_input, tensor), 0)

        input_var = torch.autograd.Variable(batch_input)
        output = extra_feature_model(input_var)
        features[i:i+batch_size] = output.data.numpy().reshape((-1, FEATURE_NUM))
        if (i%100 == 0):
            print(i)
```

```python
extract_features(True)
extract_features(False)
```

```python
print(train_features.shape)
print(test_features.shape)
```

```
torch.Size([3680, 2048])
torch.Size([3669, 2048])
```

Train linear classier and classify test features:

```python
# L2 normalize
train_features = F.normalize(Variable(torch.from_numpy(train_features)), p=2, dim=1)
test_features = F.normalize(Variable(torch.from_numpy(test_features)), p=2, dim=1)

train_label = torch.from_numpy(np.asarray(train_label)).long()
test_label = np.asarray(test_label)
```

```python
linear_classfier = LinearDiscriminantAnalysis()
linear_classfier.fit(train_features, train_label)
print(linear_classfier.score(test_features, test_label))
# linear_classfier.score(train_features, train_label)
final_result = linear_classfier.predict(test_features)
```

```
0.17879531207413465
```

Mean-per-class accuracy:

```
mean_per_class = {}
```

```
TEST_NUM = len(final_result)
for i in range(TEST_NUM):
    class_num = final_result[i]
    result = 0
    if test_label[i] == final_result[i]:
        result = 1

    if class_num in mean_per_class:
        mean_per_class[class_num].append(result)
    else:
        mean_per_class[class_num] = [result]
```

```
for key in range(38):
    if key in mean_per_class:
        print("class " + str(key) + " accuracy is " + str(sum(mean_per_class[key])/len(mean_per_class[key])))
```

```
class 1 accuracy is 0.17
class 2 accuracy is 0.10810810810810811
class 3 accuracy is 0.07142857142857142
class 4 accuracy is 0.20930232558139536
class 5 accuracy is 0.18181818181818182
class 6 accuracy is 0.15217391304347827
class 7 accuracy is 0.1743119266055046
class 8 accuracy is 0.0967741935483871
class 9 accuracy is 0.17557251908396945
class 10 accuracy is 0.16161616161616163
class 11 accuracy is 0.10204081632653061
class 12 accuracy is 0.23958333333333334
class 13 accuracy is 0.16666666666666666
class 14 accuracy is 0.12745098039215685
class 15 accuracy is 0.21621621621621623
class 16 accuracy is 0.1724137931034483
class 17 accuracy is 0.18691588785046728
class 18 accuracy is 0.2807017543859649
class 19 accuracy is 0.2553191489361702
class 20 accuracy is 0.21904761904761905
class 21 accuracy is 0.15384615384615385
class 22 accuracy is 0.18518518518518517
class 23 accuracy is 0.22641509433962265
class 24 accuracy is 0.18181818181818182
class 25 accuracy is 0.175
class 26 accuracy is 0.25688073394495414
class 27 accuracy is 0.11206896551724138
class 28 accuracy is 0.09523809523809523
class 29 accuracy is 0.22522522522522523
class 30 accuracy is 0.25
class 31 accuracy is 0.2169811320754717
class 32 accuracy is 0.22448979591836735
class 33 accuracy is 0.15555555555555556
class 34 accuracy is 0.18518518518518517
class 35 accuracy is 0.09090909090909091
class 36 accuracy is 0.2619047619047619
class 37 accuracy is 0.15555555555555556
```

Classier used:

I used the LinearDiscriminantAnalysis classier.

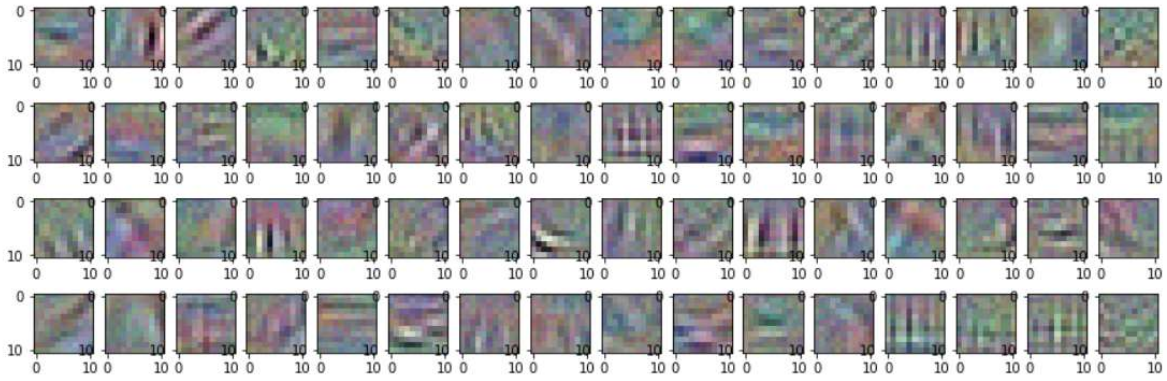
Problem 3

Part 1

IMAGE SHOWING THE FILTERS

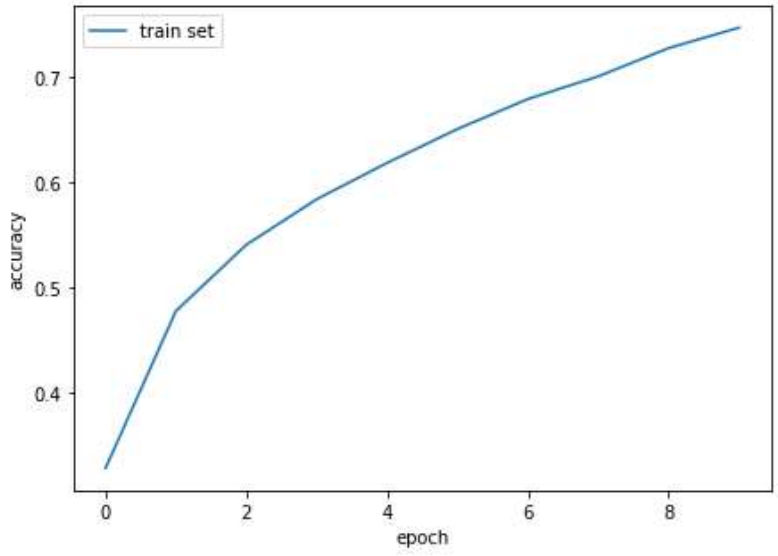
```
def print_filter():
    print(model.layer1.weight.detach().shape)
    layer = model.layer1.weight.detach().permute(0,2,3,1)
    layer = layer - layer.min()
    layer = layer / layer.max()
    fig=plt.figure(figsize=(15,5))
    for i in range(64):
        f = layer[i]
        fig.add_subplot(4,16, i+1)
        plt.imshow(f)
    plt.show()

print_filter()
```

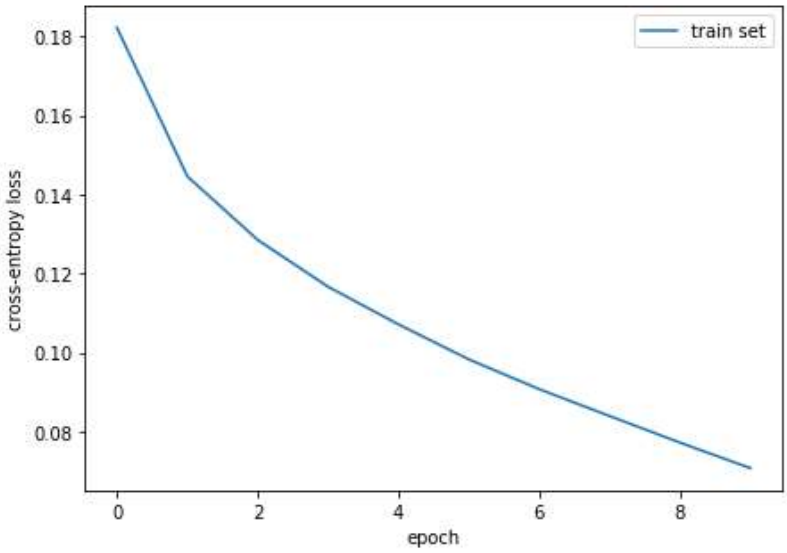
```
torch.Size([64, 3, 11, 11])
```



TRAINING LOSS AS FUNCTION OF EPOCHS

```
plt.subplots(figsize=(7,5))
plt.plot(range(epoch_num), train_acc, label="train set")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend()
plt.show()

plt.subplots(figsize=(7,5))
plt.plot(range(epoch_num), train_error, label="train set")
plt.ylabel('cross-entropy loss')
plt.xlabel('epoch')
plt.legend()
plt.show()
```





TEST DATA ACCURACY

```
# run on test data

loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

test_correct_num = 0
test_total_num = 0
test_error = 0
for i, data in enumerate(test_loader, 0):
#     if random.randint(1,1000) % 1000 != 0: # remove
#         continue
    if (i% 1000 == 0):
        print(i)

    inputs, labels = data

    optimizer.zero_grad()
    outputs = model(inputs)
    current_loss = loss(outputs, labels)
    _, result = torch.max(outputs.data, 1)

    test_correct_num = test_correct_num + (result == labels).sum().item()
    test_total_num = test_total_num + labels.size(0)
    test_error = test_error + current_loss.data.item()

print("Test data acc is " + str(test_correct_num/test_total_num) + ", error is " + str(test_error/test_total_num))
```

```
0
Test data acc is 0.6791, error is 0.0927662309229374
```

WEIGHT INITIALIZATION INFORMATION

The weight is initialized randomly.

DESCRIBE HYPER-PARAMETERS

learning rate=0.01

momentum=0.9

epoch_num = 10

batch_size = 10

num_workers = 2

Full Code

```
import numpy as np
import random

import torch
import torch.optim as optim
import torch.nn as nn
import torch.nn.functional as F

import torchvision
import torchvision.models as models
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.autograd import Variable
import matplotlib.pyplot as plt
```

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

train_data = torchvision.datasets.CIFAR10(root='./cifar-10-python/train', train=True, download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(train_data, batch_size=10, shuffle=True, num_workers=2)

test_data = torchvision.datasets.CIFAR10(root='./cifar-10-python/test', train=False, download=True, transform=transform)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=10, shuffle=False, num_workers=2)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

```
loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

epoch_num = 10

train_acc = [0] * epoch_num
train_error = [0] * epoch_num

print('Start')
for times in range(epoch_num):
    correct_num = 0
    total_num = 0
    for i, data in enumerate(train_loader, 0):
#         if random.randint(1,1000) % 1000 != 0: # remove
#             continue
        if (i% 1000 == 0):
            print(i)

        inputs, labels = data

        optimizer.zero_grad()
        outputs = model(inputs)
        current_loss = loss(outputs, labels)
        current_loss.backward()
        optimizer.step()
        _, result = torch.max(outputs.data, 1)

        correct_num = correct_num + (result == labels).sum().item()
        total_num = total_num + labels.size(0)

        train_error[times] = train_error[times] + current_loss.data.item()

    train_acc[times] = correct_num/total_num
    print(str(times) + ": acc is " + str(train_acc[times]) + ", error is " + str(train_error[times]/total_num))

print('Done')
```

```
train_error2 = train_error.copy()
for i in range(len(train_error)):
    train_error[i] = train_error[i]/50000
```

```
# run on test data

loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

test_correct_num = 0
test_total_num = 0
test_error = 0
for i, data in enumerate(test_loader, 0):
#     if random.randint(1,1000) % 1000 != 0: # remove
#         continue
    if (i% 1000 == 0):
        print(i)

    inputs, labels = data

    optimizer.zero_grad()
    outputs = model(inputs)
    current_loss = loss(outputs, labels)
    _, result = torch.max(outputs.data, 1)

    test_correct_num = test_correct_num + (result == labels).sum().item()
    test_total_num = test_total_num + labels.size(0)
    test_error = test_error + current_loss.data.item()

print("Test data acc is " + str(test_correct_num/test_total_num) + ", error is " + str(test_error/test_total_num))
```

```
0
Test data acc is 0.6791, error is 0.0927662309229374
```

```
plt.subplots(figsize=(7,5))
plt.plot(range(epoch_num), train_acc, label="train set")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend()
plt.show()

plt.subplots(figsize=(7,5))
plt.plot(range(epoch_num), train_error, label="train set")
plt.ylabel('cross-entropy loss')
plt.xlabel('epoch')
plt.legend()
plt.show()
```

```
def print_filter():
    print(model.layer1.weight.detach().shape)
    layer = model.layer1.weight.detach().permute(0,2,3,1)
    layer = layer - layer.min()
    layer = layer / layer.max()
    fig=plt.figure(figsize=(15,5))
    for i in range(64):
        f = layer[i]
        fig.add_subplot(4,16, i+1)
        plt.imshow(f)
    plt.show()

print_filter()
```

```
torch.Size([64, 3, 11, 11])
```

Part 2 batch normalization

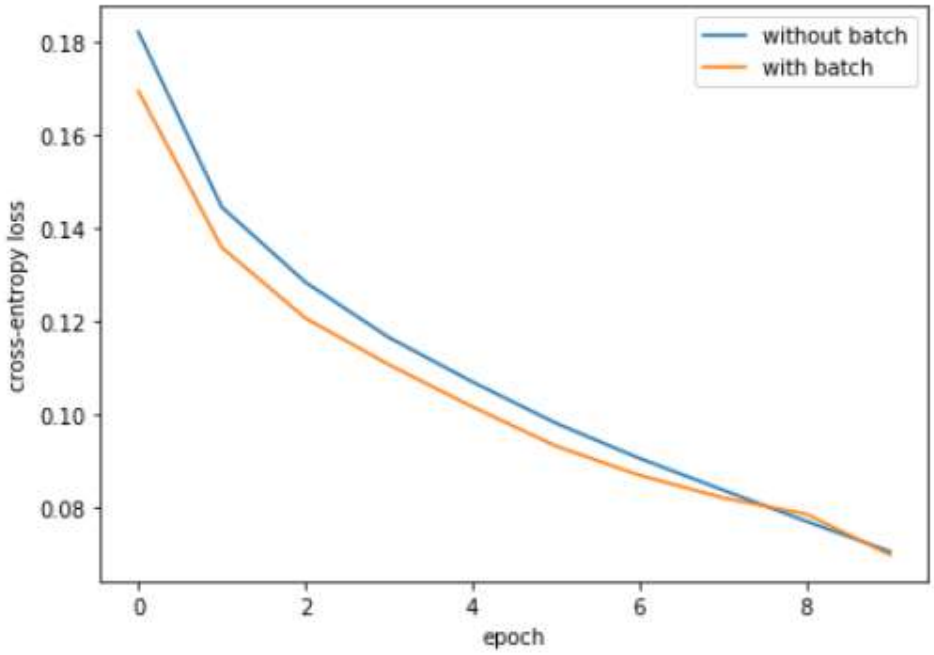
Structure

The new structure is defined as below, other code remains the same.

```
# input image: 3 * 32 * 32

class cnn(nn.Module):
    def __init__(self):
        super(cnn, self).__init__()
        self.layer1 = nn.Conv2d(3, 64, 11)
        self.pool1 = nn.MaxPool2d(2, 2)

        self.layer2 = nn.Conv2d(64, 128, 3)
        self.layer3 = nn.Conv2d(128, 128, 3)

        self.pool2 = nn.AvgPool2d(3, 2)
        self.fc1 = nn.Linear(128*3*3, 512)
        self.fc2 = nn.Linear(512, 10)

        self.batch1 = nn.BatchNorm2d(64)
        self.batch2 = nn.BatchNorm2d(128)
        self.batch3 = nn.BatchNorm2d(128)

    def forward(self, x):
        x = self.pool1(self.layer1(x))
        x = self.batch1(x)
        x = F.relu(self.layer2(x))
        x = self.batch2(x)
        x = F.relu(self.layer3(x))
        x = self.batch3(x)
        x = self.pool2(x)
        x = x.reshape(-1, 128 * 3 * 3)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = cnn()
```

Compare training loss

```
plt.subplots(figsize=(7,5))
plt.plot(range(epoch_num), train_error, label="without batch")
plt.plot(range(epoch_num), train_error_batch, label="with batch")
plt.ylabel('cross-entropy loss')
plt.xlabel('epoch')
plt.legend()
plt.show()
```



Final test error

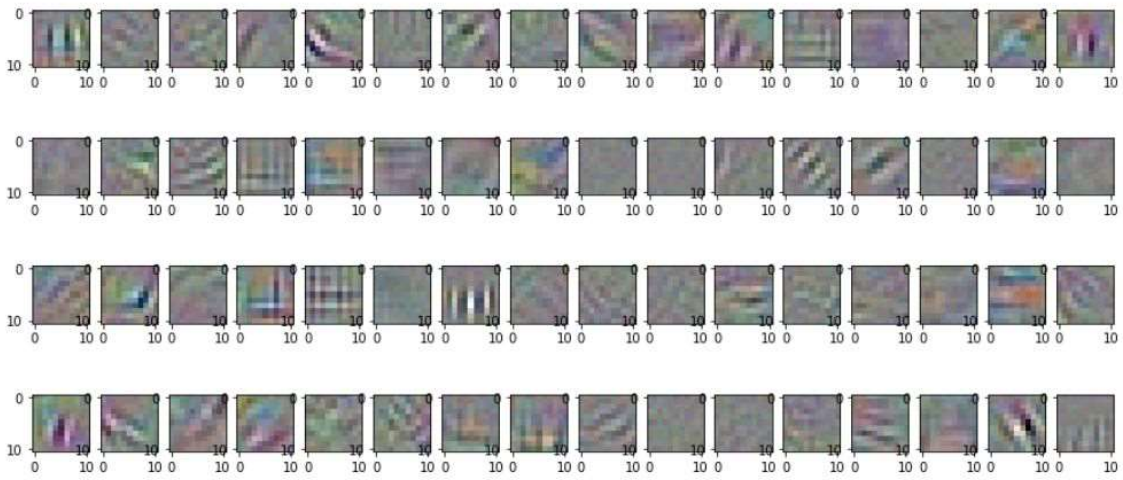
The final test error is 0.7016, which is better than the previous 0.6791.

```
# run on test data

loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

test_correct_num = 0
test_total_num = 0
test_error = 0
for i, data in enumerate(test_loader, 0):
#     if random.randint(1,1000) % 1000 != 0: # remove
#         continue
#     if (i% 1000 == 0):
#         print(i)

    inputs, labels = data

    optimizer.zero_grad()
    outputs = model(inputs)
    current_loss = loss(outputs, labels)
    _, result = torch.max(outputs.data, 1)

    test_correct_num = test_correct_num + (result == labels).sum().item()
    test_total_num = test_total_num + labels.size(0)
    test_error = test_error + current_loss.data.item()

print("Test data acc is " + str(test_correct_num/test_total_num))
```

```
Test data acc is 0.7016
```

Filters

```
def print_filter():
#     print(model.layer1.weight.detach().shape)
    layer = model.layer1.weight.detach().permute(0,2,3,1)
    layer = layer - layer.min()
    layer = layer / layer.max()
    fig=plt.figure(figsize=(15,5))
    for i in range(64):
        f = layer[i]
        fig.add_subplot(4,16, i+1)
        plt.imshow(f)
    plt.show()

print_filter()
```



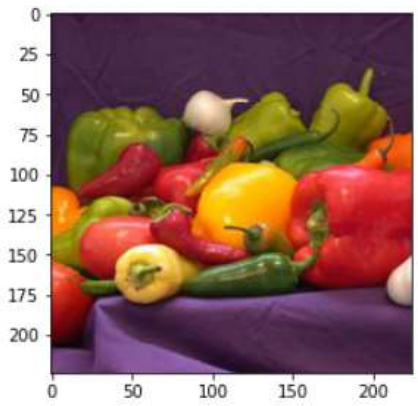
Part 3 Optimize

By replacing the 11x11 filters with 3x3 filters and adding one more layer with convolution(128, 256) in the end, it will give a test result 0.7182 which is better that the previous result 0.7016 and 0.6791.

Problem 4

```
for index in reversed(output2):
    print(index_name[index], end ="")
    print(" probability is ", end ="")
    print(np.exp(output[0][index].detach().numpy())/prob)

imgplot = plt.imshow(img)
plt.show()
```

```
bell_pepper probability is 0.9002557140463503
```



The rest is too hard :(

Problem 5

ht is a three-dimension vector, two number represent two input unit and one number represent the carry on bit. Let first two number represent two input and last represent the carry on.

For current operation, we only need the carry on bit from the previous result, so

W = [[0,0,0] [0,0,0] [0,0,1]]

We want [0,0]*U = [0,0,0], [1,1]*U = [1,1,0], [1,0]*U = [1,0,0], [0,1]*U = [0,1,0]

So U = [[1,0], [0,1], [0,0]]

h0 = [0,0,0] as there is no previous value to add.

by = 0 as there is no bias

bh = [0,0,0]

vt = [1,1,1], as we need to keep information from all three values.