COMPUTER VISION AND ITS APPLICATION

Solutions-

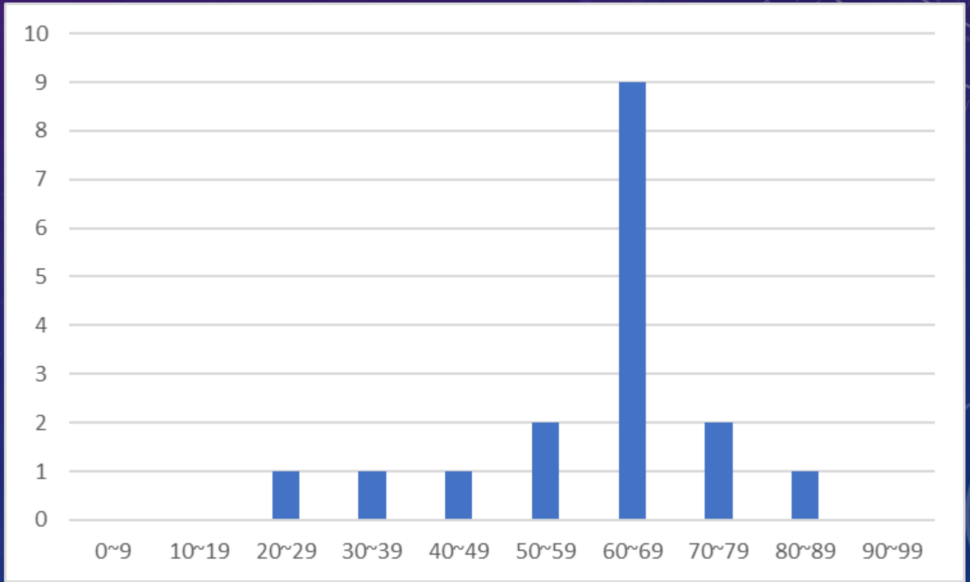
Evaluation of Your Understanding on using Pytorch for Setting up CNN

Jison Hsu
Artificial Vision Laboratory
Taiwan Tech

GRADES OVERVIEW

	Prob1	Prob2	Prob3	Total
B10630216	31	37	2	70
B10631030	25	25	15	65
B10603123	37	32	0	69
B10603043	15	32	15	62
D10803817	17	28	15	60
M10803816	18	29	7	54
M10815822	24	34	0	58
M10803339	22	28	15	65
D10903817	7	10	8	25
D10907801	21	39	8	68
M10903418	26	35	15	76
M10903429	31	0	15	46
M10903430	36	38	15	89
M10903807	21	37	4	62
M10903814	34	25	4	63
M10903151	16	14	2	32
M10903417	35	31	0	66
	24.47059	27.88235	8.235294	60.58824

Average score



Problem 1 [40/100]

1. Prob1.ipynb is a toy example, which shows you how to utilize the VGG-16 pre-trained model to extract the deep features of the input images. Please upload the files in "Prob1.zip" to the Google Colab, and answer the following questions:
   1) [2/40] Given the Q1.jpg as the input of the VGG-16, please show the feature maps and the feature dimensions of Layer-6 and Layer-10, and describe the differences.
   2) [5/40] Compute the Euclidean distance between the Q1.jpg and Q2.jpg using the Layer 6 feature maps.
   3) [5/40] Given the Q3.jpg as the input, please show the predicted class and top-3 probabilities.
   4) [2/40] Please compute the cosine similarities of the intra paired data (i.e., same classes): Q1.jpg and Q2.jpg
   5) [8/40] Please compute the cosine similarities of inter paired data (i.e., different classes):
      - 5.1) Q1.jpg and Q3.jpg
      - 5.2) Q2.jpg and Q4.jpg
      - 5.3) Q3.jpg and Q4.jpg

Problem 1 [40/100]

1. Prob1.ipynb is a toy example, which show you how to utilize the VGG-16 pre-trained model to extract the deep features of the input images. Please upload the files in "Prob1.zip" to the Google Colab., and answer the following questions:
   6) [6/40] Please describe what you observe in the four computed similarities from Prob 1.4) and 1.5).
   7) [6/40] Given a threshold = 0.5, please employ the computed similarities from Prob 1.4) and 1.5), and answer whether theses images are similar or not. (Hint: the similarity is greater than threshold, then they are similar !)
   8) [6/40] Given the predicted results in Prob1.7) and the ground-truth in Prob 1.4) and 1.5). Please show the accuracy rate.

Solution 1.1

Refer to [Example 2-1](), [Problem 1.A]()

Given the Q1.jpg as the input of the VGG-16, please show the feature maps and the feature dimensions of Layer-6 and Layer-10, and describe the differences.

Q1.jpg

Solution 1.1

Refer to
Example 2-1,
Problem 1.A

Feed the Q1.jpg into the VGG-16 pre-train model

```
Q1_6=FeatureVisualization('./Q1.jpg',6)
Q1_10=FeatureVisualization('./Q1.jpg',10)


Q1_6.save_feature_to_img()
Q1_10.save_feature_to_img1()
```

Selected Layer

Selected Layer

Save the extracted feature maps

Observation:
It can be found that the details are not clear in the deeper layer, but you can still find the primary contour. It means that the model will distill and preserve the major features and dispel the unnecessary ones.

```
On layer:6, We can get the 128 feature maps
On layer:10, We can get the 256 feature maps
```

Q1.jpg

Layer6

Layer10

Solution 1.2

Refer to
Example 2-1, 2-2,
Problem 1.B

Compute the Euclidean distance between the Q1.jpg and Q2.jpg using the Layer 6 feature maps.

```
Q2_6 = FeatureVisualization('./Q2.jpg',6)
A = Q1_6.get_feature()
B = Q2_6.get_feature()
Euclidean_dist = torch.dist(A,B,p=2)
print('Euclidean distance between the Q1.jpg and the Q2.jpg is {}'.format(Euclidean_dist))
```

```
def get_feature(self):
    # Image preprocessing
    input=self.process_image()
    #print("input.shape:{}".format(input.shape))
    x=input
    for index,layer in enumerate(self.pretrained_model):
        x=layer(x)
        # print("x:{}".format(x.shape))
        if (index == self.selected_layer):
            return x
```

```
Euclidean distance between the Q1.jpg and the Q2.jpg is 1017.1950073242188
```

Solution 1.2

Refer to
Example 2-1, 2-2,
Problem 1.B

Feed the Q2.jpg into the VGG-16 pre-train model

```
Q2_6 = FeatureVisualization('./Q2.jpg',6)
A = Q1_6.get_feature()
B = Q2_6.get_feature()
Euclidean_dist = torch.dist(A,B,p=2)
print('Euclidean distance between the Q1.jpg and the Q2.jpg is {}'.format(Euclidean_dist))
```

Q2.jpg and selected Layer6

Use get_feature() to obtain the feature maps

Compute the Euclidean_distance

```
Euclidean distance between the Q1.jpg and the Q2.jpg is 1017.1950073242188
```

Solution 1.3

Refer to Exercise 2-2

Given the Q3.jpg as the input, please show the predicted class and top-3 probabilities.

Q3.jpg

Solution 1.3

Refer to Exercise 2-2

```
Q3 = FeatureVisualization('./Q3.jpg',0)
print("The first picture classification predict:")
Q3_predict = Q3.predict()
```

```
def predict(self):
    input=self.process_image()
    outputs = self.pretrained_model2(input)

    s = torch.nn.Softmax(dim=1)
    result = s(outputs)
    self.plot_probablity(result)

    prob, predicted = result.sort(1,descending=True)
    prob = prob.data.numpy()

    predicted = predicted.data.numpy()

    print("Probablity TOP-3:\n")
    print("")
    for i in range(3):

        print("TOP_"+str(i+1))
        print("Probablity:{}".format(prob[0][i]))
        print("Predicted:{}\n".format(c[int(predicted[0][i])]))
    return outputs
```

Q3.jpg

```
Probablity TOP-3:


TOP_1
Probablity:0.7808881402015686
Predicted: 'Egyptian cat'

TOP_2
Probablity:0.09954452514648438
Predicted: 'tabby

TOP_3
Probablity:0.07643458992242813
Predicted: 'tiger cat'
```

Solution 1.3

Refer to Exercise 2-2

Feed the Q3.jpg into the VGG-16 pre-train model

```
Q3 = FeatureVisualization('./Q3.jpg',0)
print("The first picture classification predict:")
Q3_predict = Q3.predict()
```

Call the predict() function and it will print the TOP-3 probabilities and their corresponding classes

Q3.jpg

```
Probablity TOP-3:


TOP_1
Probablity:0.7808881402015686
Predicted: 'Egyptian cat'

TOP_2
Probablity:0.09954452514648438
Predicted: 'tabby

TOP_3
Probablity:0.07643458992242813
Predicted: 'tiger cat'
```

Solution 1.3

Refer to Exercise 2-2

```
def  predict(self):
        input=self.process_image()
        outputs  =  self.pretrained_model2(input)

        s  =  torch.nn.Softmax(dim=1)
        result  =  s(outputs)
        self.plot_probablity(result)

        prob,  predicted  =  result.sort(1,descending=True)
        prob  =  prob.data.numpy()

        predicted  =  predicted.data.numpy()

        print("Probablity  TOP-3:\n")
        print("")
        for  i  in  range(3):

                print("TOP_"+str(i+1))
                print("Probablity:{}".format(prob[0][i]))
                print("Predicted:{}\n".format(c[int(predicted[0][i])]))
        return  outputs
```

Get the outputs and transfer them to the probabilities by SoftMax function.

Sort these probabilities.

Show the top-3 probabilities and their corresponding classes

Solution 1.4

Refer to
Example 2-3
Exercise 2-3

Please compute the cosine similarities of the intra paired data (i.e., same classes): Q1.jpg and Q2.jpg

Q1.jpg

Q2.jpg

Intra Pair data

Solution 1.4

Refer to
Example 2-3
Exercise 2-3

Extract the latent feature by using extract_latent_feature_vgg()

```
Q1_latent = Q1_6.extract_latent_feature_vgg()
Q2_latent = Q2_6.extract_latent_feature_vgg()
cos = nn.CosineSimilarity(dim=1)

Q12_cosim = cos(Q1_latent,Q2_latent)
print('The cosine similarity between Q1 and Q2 is {}'.format(Q12_cosim.data))
```

Compute and show the similarity between Q1.jpg and Q2.jpg.

```
def extract_latent_feature_vgg(self):
    # Image preprocessing
    input=self.process_image()
    x=input

    x = self.pretrained_model2.features(x)
    x = self.pretrained_model2.avgpool(x)
    x = torch.flatten(x, 1)

    for index,layer in enumerate(self.pretrained_model2.classifier):
        x=layer(x)
        if (index == len(self.pretrained_model2.classifier)-2):
            return x
```

Extract the last second FC layer as the latent feature.

Q1.jpg

Q2.jpg

Intra Pair data

```
The cosine similarity between Q1 and Q2 is tensor([0.5561])
```

Solution 1.4

Refer to
Example 2-3
Exercise 2-3

Extract the latent feature by using extract_latent_feature_vgg()

```
Q1_latent = Q1_6.extract_latent_feature_vgg()
Q2_latent = Q2_6.extract_latent_feature_vgg()
cos = nn.CosineSimilarity(dim=1)

Q12_cosim = cos(Q1_latent,Q2_latent)
print('The cosine similarity between Q1 and Q2 is {}'.format(Q12_cosim.data))
```

Define the cosine similarity function

Compute and show the similarity between Q1.jpg and Q2.jpg.

Solution 1.4

Refer to
Example 2-3
Exercise 2-3

```python
def  extract_latent_feature_vgg(self):
    #  Image    preprocessing
    input=self.process_image()
    x=input

    x  =  self.pretrained_model2.features(x)
    x  =  self.pretrained_model2.avgpool(x)
    x  =  torch.flatten(x,   1)

    for  index,layer  in  enumerate(self.pretrained_model2.classifier):
        x=layer(x)
        if  (index  ==  len(self.pretrained_model2.classifier)-2):
            return  x
```

Extract the last second FC layer as the latent feature.

Solution 1.5

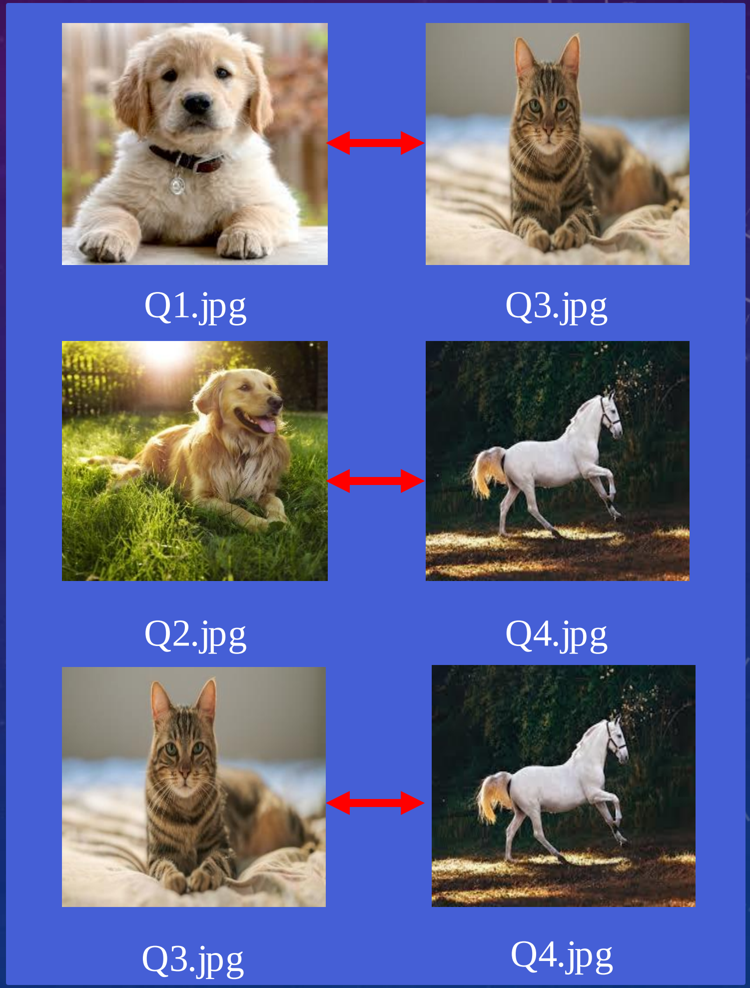
Refer to
Example 2-3
Exercise 2-3

Please compute the cosine similarities of inter paired data (i.e., different classes):

5.1) Q1.jpg and Q3.jpg

5.2) Q2.jpg and Q4.jpg

5.3) Q3.jpg and Q4.jpg



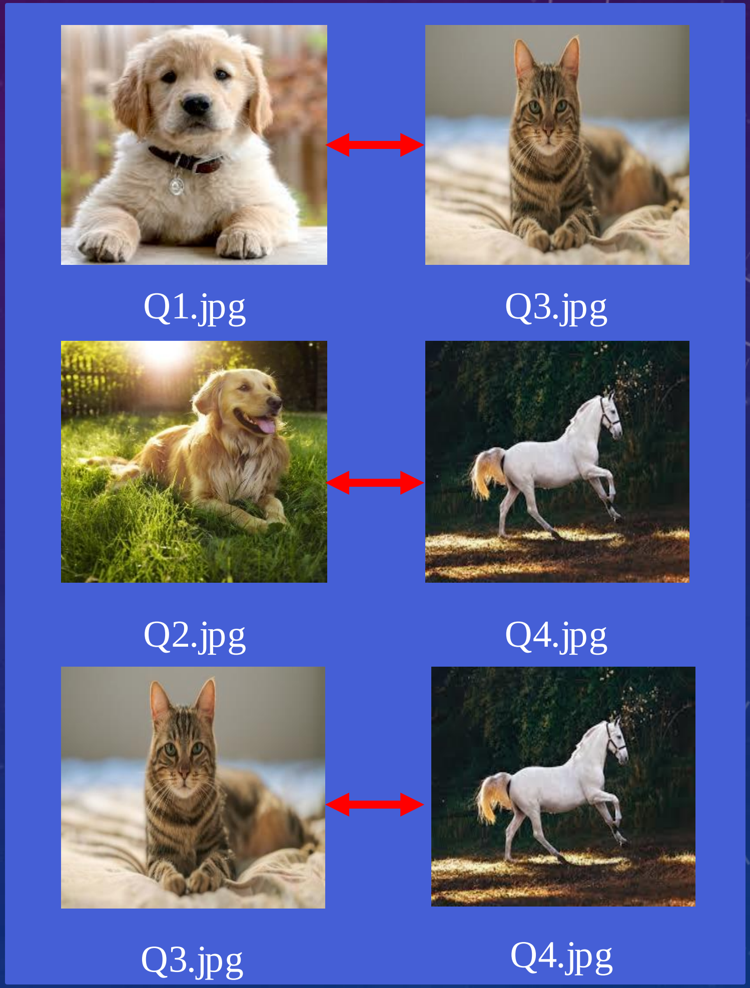
Inter Pair data

Solution 1.5

Refer to
Example 2-3
Exercise 2-3

```python
Q4 = FeatureVisualization('./Q4.jpg',0)
Q3_latent = Q3.extract_latent_feature_vgg()
Q4_latent = Q4.extract_latent_feature_vgg()

Q13_cosim = cos(Q1_latent,Q3_latent)
print('The cosine similarity between Q1 and Q3 is {}'.format(Q13_cosim.data))

Q24_cosim = cos(Q2_latent,Q4_latent)
print('The cosine similarity between Q2 and Q4 is {}'.format(Q24_cosim.data))

Q34_cosim = cos(Q3_latent,Q4_latent)
print('The cosine similarity between Q3 and Q4 is {}'.format(Q34_cosim.data))
```

```
The cosine similarity between Q1 and Q3 is tensor([0.0907])
The cosine similarity between Q2 and Q4 is tensor([0.3942])
The cosine similarity between Q3 and Q4 is tensor([0.1344])
```



Inter Pair data

Solution 1.6

Refer to
Example 2-3
Exercise 2-3

Please describe what you observe in the four computed similarities

from Prob 1.4) and 1.5).

```
The cosine similarity between Q1 and Q2 is tensor([0.5561])
```

```
The cosine similarity between Q1 and Q3 is tensor([0.0907])
The cosine similarity between Q2 and Q4 is tensor([0.3942])
The cosine similarity between Q3 and Q4 is tensor([0.1344])
```

Q1.jpg

Q2.jpg

Q3.jpg

Q4.jpg

Observation:
It can be seen that the similarity will be lower when comparing the different species. The value would be affected by the compositions of the two images.

Solution 1.7

Refer to
Example 2-3
Exercise 2-3

Given a threshold = 0.5, please employ the computed similarities from Prob 1.4) and 1.5), and answer whether theses images are similar or not. (Hint: the similarity is greater than threshold, then they are similar !)

Solution 1.7

Refer to
Example 2-3
Exercise 2-3

```python
threshold = 0.5
## Q1.jpg and Q2.jpg
if Q12_cosim >= threshold:
    print("Q1.jpg and Q2.jpg are similar, its similarity is {}".format(Q12_cosim.data))
else :
    print("Q1.jpg and Q3.jpg are not similar, its similarity is {}".format(Q12_cosim.data))
## Q1.jpg and Q3.jpg
if Q13_cosim >= threshold:
    print("Q1.jpg and Q3.jpg are similar, its similarity is {}".format(Q13_cosim.data))
else :
    print("Q1.jpg and Q3.jpg are not similar, its similarity is {}".format(Q13_cosim.data))
## Q2.jpg and Q4.jpg
if Q24_cosim >= threshold:
    print("Q2.jpg and Q4.jpg are similar, its similarity is {}".format(Q24_cosim.data))
else :
    print("Q2.jpg and Q4.jpg are not similar, its similarity is {}".format(Q24_cosim.data))
## Q3.jpg and Q4.jpg
if Q34_cosim >= threshold:
    print("Q3.jpg and Q4.jpg are similar, its similarity is {}".format(Q34_cosim.data))
else :
    print("Q3.jpg and Q4.jpg are not similar, its similarity is {}".format(Q34_cosim.data))
```

```
Q1.jpg and Q2.jpg are similar, its similarity is tensor([0.5561])
Q1.jpg and Q3.jpg are not similar, its similarity is tensor([0.0907])
Q2.jpg and Q4.jpg are not similar, its similarity is tensor([0.3942])
Q3.jpg and Q4.jpg are not similar, its similarity is tensor([0.1344])
```

Solution 1.8

Refer to
Example 2-3
Exercise 2-3

Given the predicted results in Prob1.7) and the ground-truth in Prob 1.4) and 1.5). Please show the accuracy rate.

```
threshold= 0.5
similarities = [Q12_cosim,Q13_cosim,Q24_cosim,Q34_cosim]
pair_Groundtruth = [1,0,0,0]
correct = 0
Total_pair = 4
for sim,GT in zip(similarities,pair_Groundtruth):
    if sim >= threshold :
        pred = 1
    else :
        pred = 0
    if pred == GT:
        correct +=1
Accuracy = correct/Total_pair
print('The accuracy rate is {} %'.format(Accuracy*100))
```

```
The accuracy rate is 100.0 %
```

Solution 1.8

Refer to
Example 2-3
Exercise 2-3

```
threshold= 0.5
similarities = [Q12_cosim,Q13_cosim,Q24_cosim,Q34_cosim]
pair_Groundtruth = [1,0,0,0]
correct = 0
Total_pair = 4
for sim,GT in zip(similarities,pair_Groundtruth):
    if sim >= threshold :
        pred = 1
    else :
        pred = 0
    if pred == GT:
        correct +=1
Accuracy = correct/Total_pair
print('The accuracy rate is {} %'.format(Accuracy*100))
```

```
The accuracy rate is 100.0 %
```

The similarities are obtained from 1.4 and 1.5

GT of the paired data, intra = 1, inter = 0

Compare the similarities with threshold

Check whether the prediction is equal to the GT

Problem 2 [45/100]

2. Prob2.ipynb shows you how to train a classifier. Please modify the Prob2.ipynb with the following requirements and set the Cross Entropy Loss, optimizer SGD, 0.002 learning rate and 0.9 momentum to train a Fashion-MNIST classifier :

1) [8/45] Design a model with the following structure and complete the table in the next page.
   - First Conv. layer: Input: Gray scale, Output Channel 32, second Conv. layer: Output Channel 64, third Conv. layer: Output Channel 128.
   - FC-Layer1: Input: Defined by the third convolutional Layer, Output: 1000
   - FC-Layer2: Input: From FC- Layer1, Output: 500
   - FC-Layer3: Input: From FC- Layer2, Output: equal to your class size
2) [2/45] Train the classifier and show the losses during the training process.
3) [2/45] Save the model and name it ‘Prob2.pth’ .
4) [2/45] Save the optimizer and name it as ‘Prob2_1.pth’.

Net ( )

Layer type	Input channel	Output channel	Filter size	Stride	padding
Conv1	A	B	3	1	1
ReLU					
MaxPool			2	2	0
Conv2	B	C	3	2	0
ReLU					
MaxPool			2	2	0
Conv3	C	D	4	2	1
ReLU					
AvgPool			2	1	1
Linear1	E	G			
ELU					
Linear2	G	F			
ELU					
Linear3	F	G			

Please crop the parts that you modify in Prob2.ipynb and paste to the solution .docx.

Problem 2

2. Prob2.ipynb shows how to train a classifier. Please modify the Prob2.ipynb with the following requirements and set the Cross Entropy Loss, optimizer SGD, 0.002 learning rate and 0.9 momentum to train a Fashion-MNIST classifier :

   5) [4/45] Change the learning rate :0.0002. Load the 'Prob2.pth and Prob2_1.pth' obtained from Prob 2.3). and Prob 2.4). as the pre-trained model.

   6) [4/45] Resume training the model on the Fashion-MNIST dataset.

   7) [2/30] Run the testing code and show the accuracy.

   8) [8/30] Modify your structure by following setting, train with 6 epochs and test the modified classifier.

      - FC-Layer1: Input: Defined by the third convolutional Layer, Output: 4096
      - FC-Layer2: Input: From FC- Layer1, Output: 2048
      - FC-Layer3: Input: From FC- Layer2, Output: equal to your class size.

Problem 2

2. Prob2.ipynb shows how to train a classifier. Please modify the Prob2.ipynb with the following requirements and set the Cross Entropy Loss, optimizer SGD, 0.002 learning rate and 0.9 momentum to train a Fashion-MNIST classifier :

   9) [4/30] By comparing with the two different structures, please describe what you observe according your results.

   10) [6/30] Please describe the why we need to use the fully-connected layer and the convolution layer.

   11) [3/30] If we change the dataset to CIFAR100 (RGB, 100 classes) from Fashion-MNIST (Gray, 10 classes), what we need to modify in our network.

Solution 2.1

Refer to Problem 2.A

Design a model with the following structure and complete the table in the next page.

- First Conv. layer: Input: Gray scale, Output Channel 32, second Conv. layer:
  - Output Channel 64, third Conv. layer: Output Channel 128.
- FC-Layer1: Input: Defined by the third convolutional Layer, Output: 1000
- FC-Layer2: Input: From FC- Layer1, Output: 500
- FC-Layer3: Input: From FC- Layer2, Output: equal to your class size

Solution 2.1

Refer to Problem 2.A

```python
class Net(nn.Module):
        def __init__(self):
            super(Net, self).__init__()
            self.relu = nn.ReLU()

            self.conv1 = nn.Conv2d(1,32,3,1,1)
            self.maxpool1 = nn.MaxPool2d(2,2,0)

            self.conv2 = nn.Conv2d(32,64,3,2,0)
            self.maxpool2 = nn.MaxPool2d(2,2,0)

            self.conv3 = nn.Conv2d(64,128,4,2,1)
            self.avgpool1 = nn.AvgPool2d(2,1,1)



            self.fc1 = nn.Linear(128*2*2,1000)
            self.elu1 = nn.ELU()
            self.fc2 = nn.Linear(1000,500)
            self.elu2 = nn.ELU()
            self.fc3 = nn.Linear(500,10)
```

Shape:
(Batch size, 128, 2, 2)

```python
def forward(self, x):
    batchsize = x.shape[0]
    x = self.maxpool1(self.relu(self.conv1(x)))
    x = self.maxpool2(self.relu(self.conv2(x)))
    x = self.avgpool1(self.relu(self.conv3(x)))
    # print(x.shape)
    x = x.view(batchsize,-1)
    #print(x.shape)
    x = self.elu1(self.fc1(x))
    x = self.elu2(self.fc2(x))
    x = self.fc3(x)


    return x
```

Solution 2.1

Refer to Problem 2.A

Shape: (Batch size, 128, 2, 2)

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.relu = nn.ReLU()

        self.conv1 = nn.Conv2d(1,32,3,1,1)
        self.maxpool1 = nn.MaxPool2d(2,2,0)

        self.conv2 = nn.Conv2d(32,64,3,2,0)
        self.maxpool2 = nn.MaxPool2d(2,2,0)

        self.conv3 = nn.Conv2d(64,128,4,2,1)
        self.avgpool1 = nn.AvgPool2d(2,1,1)



        self.fc1 = nn.Linear(128*2*2,1000)
        self.elu1 = nn.ELU()
        self.fc2 = nn.Linear(1000,500)
        self.elu2 = nn.ELU()
        self.fc3 = nn.Linear(500,10)
```

```
def forward(self, x):
    batchsize = x.shape[0]
    x = self.maxpool1(self.relu(self.conv1(x)))
    x = self.maxpool2(self.relu(self.conv2(x)))
    x = self.avgpool1(self.relu(self.conv3(x)))
    # print(x.shape)
    x = x.view(batchsize,-1)
    #print(x.shape)
    x = self.elu1(self.fc1(x))
    x = self.elu2(self.fc2(x))
    x = self.fc3(x)


    return x
```

Net ( )

Layer type	Input channel	Output channel	Filter size	Stride	padding
Conv1	A:1	B:32	3	1	1
ReLU					
MaxPool			2	2	0
Conv2	B:32	C:64	3	2	0
ReLU					
MaxPool			2	2	0
Conv3	C:64	D:128	4	2	1
ReLU					
AvgPool			2	1	1
Linear1	E:512	G:1000			
ELU					
Linear2	G:1000	F:500			
ELU					
Linear3	F:500	G:10			

Refer to Problem 2.A

Please crop the parts that you modify in Prob2.ipynb and paste to the solution .docx.

Solution 2.2

Refer to Problem 2.E

Train the classifier and show the losses during the training process.

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

trainset = torchvision.datasets.FashionMNIST(root='./data', train=True,download=True,transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,shuffle=True, num_workers=2)

testset = torchvision.datasets.FashionMNIST(root='./data', train=False,download=True,transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,shuffle=False, num_workers=2)


classes = ('T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal',
           'Shirt', 'Sneaker', 'Bag', 'Ankle boot')
```

The original codes are ((0.5,0.5,0.5),(0.5,0.5,0.5))
It represents your datasets are 3 channels.
If you want to use the gray-scale dataset, you need to modify the codes,
((0.5),(0.5))

Solution 2.2

Refer to Problem 2.E

The original codes are ((0.5,0.5,0.5),(0.5,0.5,0.5))
It represents your datasets are 3 channels. If you want to use the gray-scale dataset, you need to modify the codes to ((0.5),(0.5))

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

trainset = torchvision.datasets.FashionMNIST(root='./data',  train=True,download=True,transform=transform)
trainloader = torch.utils.data.DataLoader(trainset,  batch_size=4,shuffle=True,  num_workers=2)

testset = torchvision.datasets.FashionMNIST(root='./data',  train=False,download=True,transform=transform)
testloader = torch.utils.data.DataLoader(testset,  batch_size=4,shuffle=False,  num_workers=2)



classes = ('T-shirt/top',  'Trouser',  'Pullover',  'Dress',  'Coat',  'Sandal',
                  'Shirt',  'Sneaker',  'Bag',  'Ankle  boot')
```

Solution 2.2

Refer to
Exercise 2-4 (P.61)
Exercise 2-4 (P.62)

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.002, momentum=0.9)
```

```
for epoch in range(3):  # loop over the dataset multiple times

        running_loss = 0.0
        for i, data in enumerate(trainloader):
            # get the inputs; data is a list of [inputs, labels]
            inputs, labels = data
            inputs = inputs.cuda()
            labels = labels.cuda()


            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
```

```
Epoch : 1 steps : 1000 Training Loss : 2.127310976743698
Epoch : 1 steps : 2000 Training Loss : 1.0009411244057118
Epoch : 1 steps : 3000 Training Loss : 0.7320705399112776
Epoch : 1 steps : 4000 Training Loss : 0.6135896655730904
Epoch : 1 steps : 5000 Training Loss : 0.570787304927595
```

```
Epoch : 3 steps : 10000 Training Loss : 0.30414618078O2189
Epoch : 3 steps : 11000 Training Loss : 0.29006112643110193
Epoch : 3 steps : 12000 Training Loss : 0.27181791267670635
Epoch : 3 steps : 13000 Training Loss : 0.2741309449586397
Epoch : 3 steps : 14000 Training Loss : 0.2826621224831997
Epoch : 3 steps : 15000 Training Loss : 0.2622677051232313
Finished Training
```

1. Feed the input into the model and get the prediction.
2. Use the defined loss function to calculate the loss between the prediction and the label.
3. Use backward() to compute the gradient and use the optimizer.step() to update the weight

Solution 2.2

Refer to
Exercise 2-4 (P.64)

Testing

```python
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        images = images.cuda()
        labels = labels.cuda()
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy : %d %%' % (100 * correct / total))
```

```
Accuracy : 89 %
```

1. Use for loop to get the entire testing data
2. Feed the testing data and calculate the accuracy

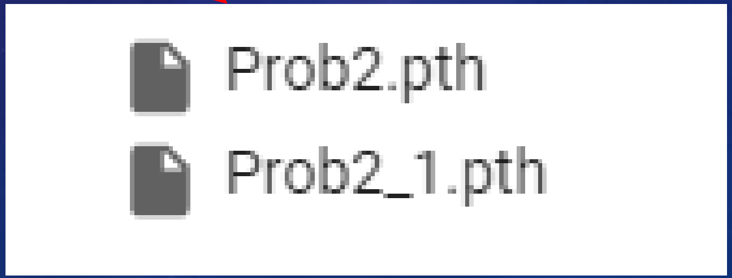
Solution 2.3 & 2.4

Refer to
Problem 2.B & 2.C

Save the model and name it as ‘Prob2.pth’ .
Save the optimizer and name it as ‘Prob2_1.pth’.

```
Path='Prob2.pth'
torch.save(net.state_dict(),Path)
```

```
Path=Path='Prob2_1.pth'
torch.save(optimizer.state_dict(),Path)
```



Solution 2.5

Refer to Problem 2.F

Change the learning rate :0.0002. Load the ‘Prob2.pth and Prob2_1.pth’ obtained from Prob 2.3). and Prob 2.4). as the pre-trained model.

```
net = Net()
checkpoint = 'Prob2.pth'
checkpoint = torch.load(checkpoint)
net.load_state_dict(checkpoint)
net = net.cuda()



criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.0002, momentum=0.9)

checkpoint = 'Prob2_1.pth'
checkpoint = torch.load(checkpoint)
optimizer.load_state_dict(checkpoint)
```

Solution 2.6

Refer to
Exercise 2-4 (P.62)

Resume training the model on the Fashion-MNIST dataset.

```
for epoch in range(3):   # loop over the dataset multiple times

        running_loss = 0.0
        for i, data in enumerate(trainloader):
            # get the inputs; data is a list of [inputs, labels]
            inputs, labels = data

            inputs = inputs.cuda()
            labels = labels.cuda()


            # zero the parameter gradients
            optimizer.zero_grad()
            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
```

```
Epoch : 1 steps : 1000 Training Loss : 0.27730414292975
Epoch : 1 steps : 2000 Training Loss : 0.251289471682419
Epoch : 1 steps : 3000 Training Loss : 0.263792005553687
Epoch : 1 steps : 4000 Training Loss : 0.2605055343461393
Epoch : 1 steps : 5000 Training Loss : 0.244047583428532
Epoch : 1 steps : 6000 Training Loss : 0.2572043398532777
Epoch : 1 steps : 7000 Training Loss : 0.2593977226401220
```

```
Epoch : 3 steps : 7000 Training Loss : 0.2087033114988908
Epoch : 3 steps : 8000 Training Loss : 0.2131072640818001
Epoch : 3 steps : 9000 Training Loss : 0.2019322334044352
Epoch : 3 steps : 10000 Training Loss : 0.2344798367654384
Epoch : 3 steps : 11000 Training Loss : 0.2149624526704774
Epoch : 3 steps : 12000 Training Loss : 0.2331448473241152
```

Solution 2.7

Refer to
Exercise 2-4 (P.64)

Testing

```
[ ] dataiter = iter(testloader)
    images, labels = dataiter.next()
```

```
[ ] correct = 0
    total = 0
    with torch.no_grad():
        for data in testloader:
            images, labels = data
            images, labels = images.cuda(), labels.cuda()
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print('Accuracy : %d %%' % (100 * correct / total))
```

```
Accuracy : 90 %
```

Solution 2.8

Refer to Problem 2.A

Modify your structure by following setting, train with 6 epochs and test the modified classifier.

- FC-Layer1: Input: Defined by the third convolutional Layer, Output: 4096
- FC-Layer2: Input: From FC- Layer1, Output: 2048
- FC-Layer3: Input: From FC- Layer2, Output: equal to your class size.

```
class Net2(nn.Module):
    def __init__(self):
        super(Net2, self).__init__()
        self.relu = nn.ReLU()

        self.conv1 = nn.Conv2d(1,32,3,1,1)
        self.maxpool1 = nn.MaxPool2d(2,2,0)

        self.conv2 = nn.Conv2d(32,64,3,2,0)
        self.maxpool2 = nn.MaxPool2d(2,2,0)

        self.conv3 = nn.Conv2d(64,128,4,2,1)
        self.avgpool1 = nn.AvgPool2d(2,1,1)



        self.fc1 = nn.Linear(128*2*2,4096)
        self.elu1 = nn.ELU()
        self.fc2 = nn.Linear(4096,2048)
        self.elu2 = nn.ELU()
        self.fc3 = nn.Linear(2048,10)
```

Solution 2.8

Refer to
Exercise 2-4 (P.62)

```
net2 = Net2()
net2 = net2.cuda()
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net2.parameters(), lr=0.002, momentum=0.9)
```

```
for epoch in range(6):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs = inputs.cuda()
        labels = labels.cuda()


        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net2(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

Solution 2.8

Refer to
Exercise 2-4 (P.62)

```
net2 = Net2()
net2 = net2.cuda()
```

Initialize the net2 parameter and transfer it to GPU mode

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net2.parameters(), lr=0.002, momentum=0.9)
```

Use SGD optimizer to update the parameters in the Net2.
Note:
Remember to use the net2.parameter instead of using net.parameter.

Solution 2.8

Refer to

Exercise 2-4 (P.62)

```
for epoch in range(6):    # loop over the dataset multiple times

        running_loss = 0.0
        for i, data in enumerate(trainloader):
            # get the inputs; data is a list of [inputs, labels]
            inputs, labels = data
            inputs = inputs.cuda()
            labels = labels.cuda()


            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net2(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
```

Remember to modify the codes for feeding data into the net2.

Solution 2.9 & 2.10

- By comparing with the two different structures, please describe what you observe according your results
- Solution 2.9:
  - More iteration times is needed to achieve the similar performance when enlarging the FC layer. It means that the modification will increase the complexities. However, the performance does not approve the better results in this modification. The fact shows that it is a batter way to design the appropriate FC layers according to the dataset.

- Please describe the why we need to use the fully-connected layer and the convolution layer.

- Solution 2.10:
  - CNNs have two main parts: A convolution mechanism that breaks up the images into features and analyzes them. A fully connected layer that takes the outputs of convolution and predicts the best answer to describe the image.

  - Convolution is used in deep neural networks, which can reduce parameters through parameter sharing and sparse connections. When using fully connected layer to connect two large dimension features, there will be a lot of parameters.

Solution 2.11

If we change the dataset to CIFAR100 (RGB, 100 classes) from Fashion-MNIST (Gray, 10 classes), what we need to modify in our network.

CIFAR100: RGB, 100 classes
Fashion-MNIST: Gray, 10 classes

When changing the dataset, we need to modify the first convolution layer, the first and third fully connected layers setting to match the dataset format. (i.e., image size, channels, classes)

Problem 3 [15/100]

Refer to Problem 3

3. The output dimensions of Conv4 is 128×96×56 (i.e., Chanel × Width × Hieght) , please calculate the dimensions of the inputs and the outputs from Conv1, Conv2, Conv3, Conv5.

Layer type	Input channel	Output channel	Filter size	Stride	Padding
Conv1	3	16	3	1	1
AvgPool			4	1	0
Conv2	16	16	4	2	0
MaxPool			2	2	0
Conv3	16	32	2	1	1
MaxPool			2	1	0
Conv4	32	128	3	2	2
AvgPool			3	1	1
Conv5	128	256	7	1	0

Solution3

$$Output = \frac{Input - kernel\ size + 2 \times Paddiing}{Stride} + 1$$

$$Input = (Output - 1) \times Stride\ - 2 \times Paddiing + kernel\ size$$

- Input-> [3,761,441]
- Conv1-> [8, 761, 441]
- AvgPool1-> [8, 758, 438]
- Conv2-> [16, 378, 218]
- MaxPool1-> [16, 189, 109]
- Conv3-> [32, 190, 110]
- MaxPool2-> [32, 189, 109]
- Conv4-> [128, 96, 56]

$761 = (761 - 1) \times 1\ - 2 \times 1{+}3$, $441 = (2065 - 1) \times 1\ - 2 \times 1{+}3$

$761 = (758 - 1) \times 1\ - 2 \times 0{+}4$, $441 = (438 - 1) \times 1\ - 2 \times 0{+}4$

$758 = (378 - 1) \times 2\ - 2 \times 0{+}4$, $438 = (218 - 1) \times 2\ - 2 \times 0{+}4$

$378 = (189 - 1) \times 2\ - 2 \times 0{+}2$, $218 = (515 - 1) \times 2\ - 2 \times 0{+}2$

$189 = (190 - 1) \times 1\ - 2 \times 1{+}2$, $109 = (110 - 1) \times 1\ - 2 \times 1{+}2$

$190 = (189 - 1) \times 1\ - 2 \times 0{+}2$, $110 = (109 - 1) \times 1\ - 2 \times 0{+}2$

$189 = (96 - 1) \times 2\ - 2 \times 2{+}3$, $109 = (56 - 1) \times 2\ - 2 \times 2{+}3$

Refer to Problem 3

Solution3

Refer to Problem 3

- AvgPool2->[128, 96, 56]
- Conv5-> [256, 90, 50]

$$96 = \frac{96 - 3 + 2 \times 1}{1} + 1\,, \qquad 56 = \frac{56 - 3 + 2 \times 1}{1} + 1$$

$$90 = \frac{96 - 7 + 2 \times 0}{1} + 1\,, \qquad 50 = \frac{56 - 7 + 2 \times 1}{1} + 1$$

Reference

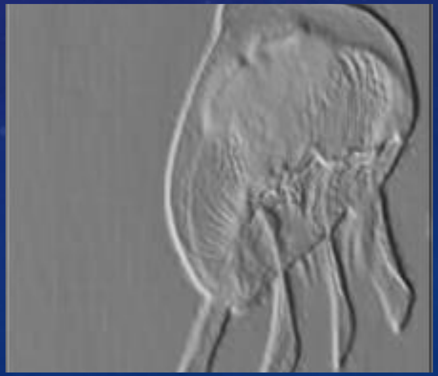
Content Overview

- The examples and the exercises
- The solutions to the sample problems

The Examples and The Exercises

Example 2-1: Feature Map Visualization

- Please download the "2-1_Feature_map_visualization.zip" from the Moodle, which is built on the VGG-16 trained on the ImageNet.
- Upload the 2-1_Feature_map_visualization.ipynb and imagenet1000_clsidx_to_labels.txt to the Google Colab.
- Upload the "jellyfish.jpg" to the Google Colab.
- Run the codes and get the feature map from the selected layer.



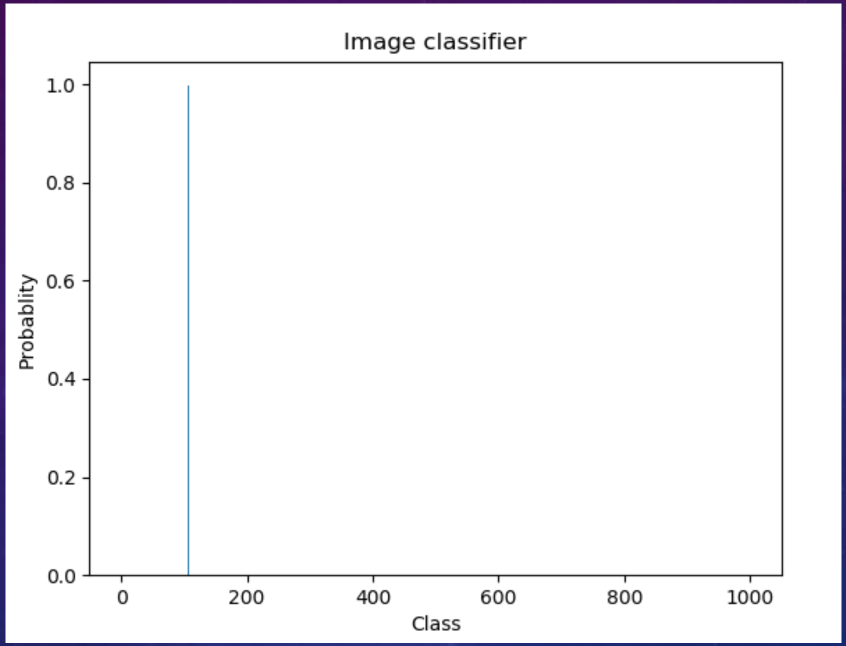
Example 2-1: Feature Map Visualization

imagenet1000_clsidx_to_labels.txt
(in “Feature_map_visualization_v2.7z”)

```
0: 'tench, Tinca tinca',
1: 'goldfish, Carassius auratus',
2: 'great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias',
3: 'tiger shark, Galeocerdo cuvieri',
4: 'hammerhead, hammerhead shark',
5: 'electric ray, crampfish, numbfish, torpedo',
6: 'stingray',
7: 'cock',
8: 'hen',
9: 'ostrich, Struthio camelus',
10: 'brambling, Fringilla montifringilla',
11: 'goldfinch, Carduelis carduelis',
12: 'house finch, linnet, Carpodacus mexicanus',
13: 'junco, snowbird',
14: 'indigo bunting, indigo finch, indigo bird, Passerina cyanea',
15: 'robin, American robin, Turdus migratorius',
16: 'bulbul',
17: 'jay',
18: 'magpie',
19: 'chickadee',
20: 'water ouzel, dipper',
21: 'kite',
22: 'bald eagle, American eagle, Haliaeetus leucocephalus',
23: 'vulture',
24: 'great grey owl, great gray owl, Strix nebulosa',
25: 'European fire salamander, Salamandra salamandra',
26: 'common newt, Triturus vulgaris',
27: 'eft',
28: 'spotted salamander, Ambystoma maculatum',
29: 'axolotl, mud puppy, Ambystoma mexicanum',
30: 'bullfrog, Rana catesbeiana',
31: 'tree frog, tree-frog',
32: 'tailed frog, bell toad, ribbed toad, tailed toad, Ascaphus trui',
33: 'loggerhead, loggerhead turtle, Caretta caretta',
```

Example 2-1 : Feature Map Visualization

Original image: jellyfish.jpg



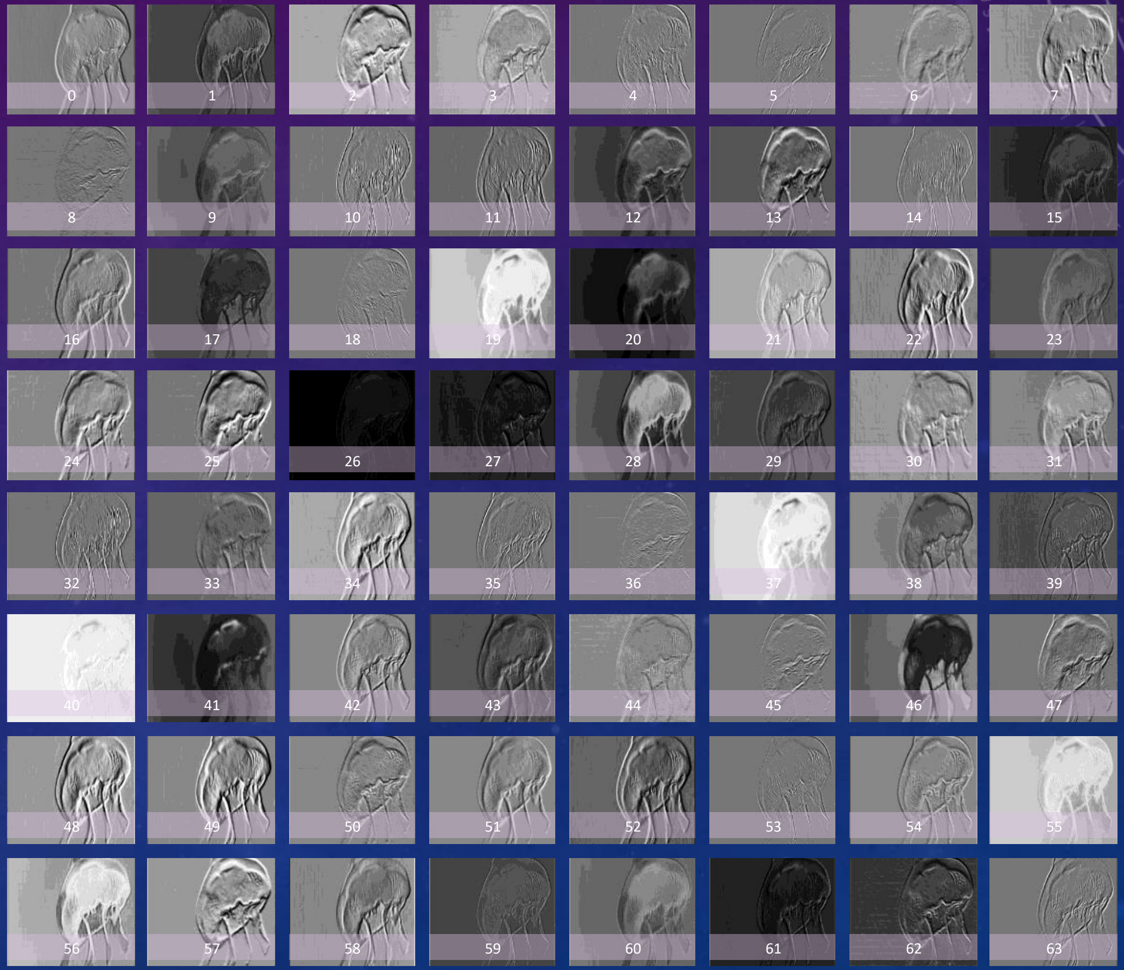
Probability of the classes

```
Probablity TOP-3:


TOP_1
Probablity:0.9990069270133972
Predicted: 'jellyfish'

TOP_2
Probablity:0.0008054533391259611
Predicted: 'isopod'

TOP_3
Probablity:8.906585571821779e-05
Predicted: 'chambered nautilus
```

Predicted class : jellyfish

Example 2-1 : Feature Map Visualization

Original image: jellyfish.jpg



Example 2-1 : Coding Explanation

Overview of the sample code:

- Main Process for executing
- Function - FeatureVisualization
  - "__init__ (i.e. initialization)" for setting the pretrained model i.e., vgg16 on ImageNet
  - "process_image" for the image preprocessing.
  - "get_multi_feature" for getting the feature maps.
  - "save_feature_to_img" for saving the feature maps.
  - "Predict" for getting the prediction from the given image.

Example 2-1 : Coding Explanation

Return solution 1.1

Main Process for executing

```
if __name__=='__main__':
    # get class
    c = {}
    with open("imagenet1000_clsidx_to_labels.txt") as f:
        for line in f:
            (key, val) = line.split(":")
            c[int(key)] = val.split(",")[0]
    # Define image path and select the layer
    myClass=FeatureVisualization('./jellyfish.jpg',5)
    print(myClass.pretrained_model2)

    myClass.save_feature_to_img()
    myClass.predict()
```

Open the txt file to get the information (Hint: You need to upload the file to Colab., or you will get the error "No such file…..")

Select the layer 5

Print the Network Architecture

1. Call the function to save the extracted feature from the selected layer in the VGG16.
2. Call the predict function to get the prediction from the given image

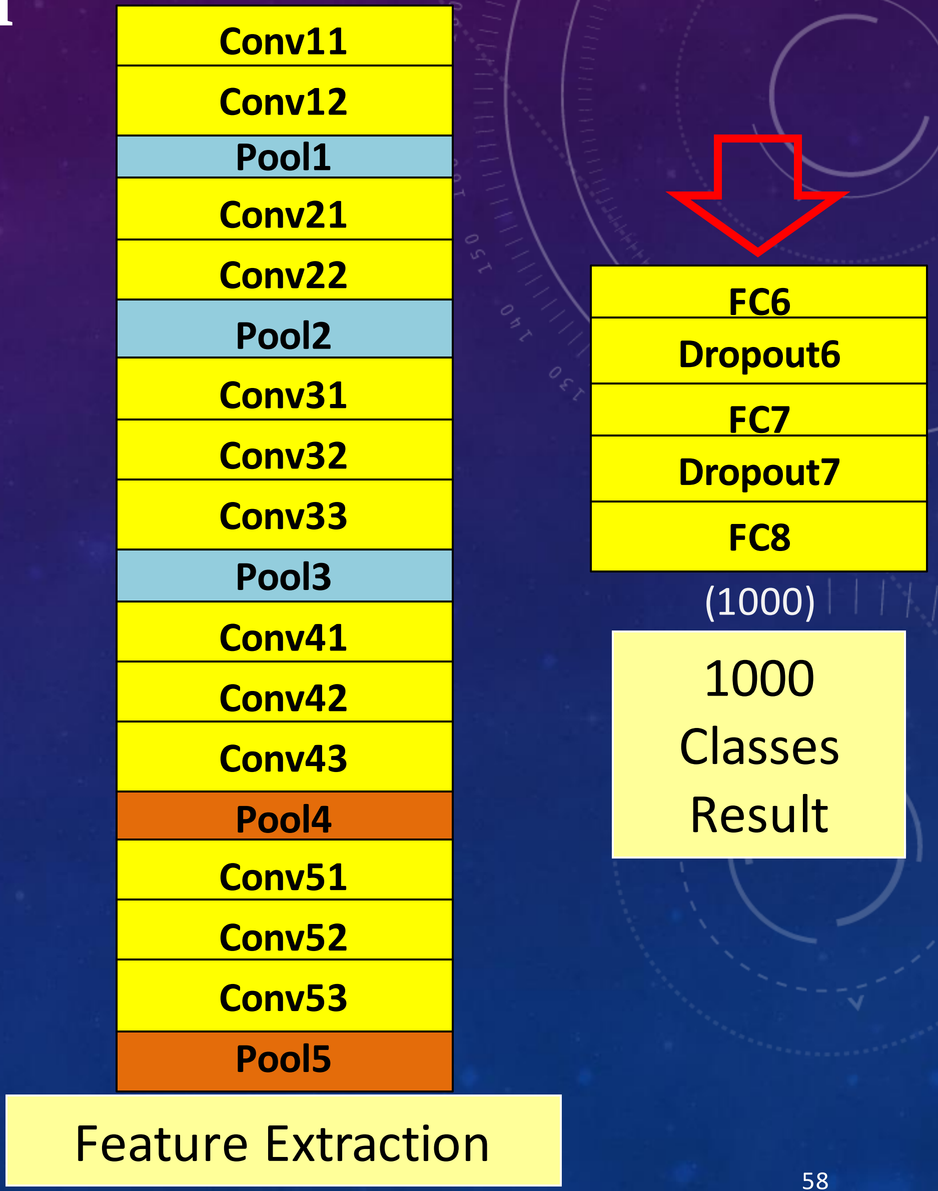
Example 2-1 : Coding Explanation

FeatureVisualization

```
def  __init__(self,img_path,selected_layer):
        self.img_path=img_path
        self.selected_layer=selected_layer
        #  Load  pretrained  model

        self.pretrained_model  =  models.vgg16(pretrained=True).features
        self.pretrained_model.eval()

        self.pretrained_model2  =  models.vgg16(pretrained=True)
        self.pretrained_model2.eval()
```

Call the feature part of vgg16 pretrained model.
“eval()” is for fixing the pretrained weight.

Call the entire vgg16 pretrained model (i.e. the feature part and classifier part)
“eval()” is for fixing the pretrained weight.



Example 2-1 : Coding Explanation

```
def process_image(self):
    img=cv2.imread(self.img_path)
    img=preprocess_image(img)
    return img
```

Read the given image and preprocess it before feeding it into the model.

```
def preprocess_image(cv2im, resize_im=True):

    # Resize image
    if resize_im:
        cv2im = cv2.resize(cv2im, (224, 224))
    im_as_arr = np.float32(cv2im)
    im_as_arr = np.ascontiguousarray(im_as_arr[..., ::-1])
    im_as_arr = im_as_arr.transpose(2, 0, 1)  # Convert array to D,W,H
    # Normalize the channels
    for channel, _ in enumerate(im_as_arr):
        im_as_arr[channel] /= 255
    # Convert to float tensor
    im_as_ten = torch.from_numpy(im_as_arr).float()
    # Add one more channel to the beginning. Tensor shape = 1,3,224,224
    im_as_ten.unsqueeze_(0)
    # Convert to Pytorch variable
    im_as_var = Variable(im_as_ten, requires_grad=True)
    return im_as_var
```

Preprocess:

1. Resize to the 224x224 (i.e. VGG16 input size)
2. Covert the dimension to match the format of PyTorch.
3. Normalize the value of the data (From 0 to 1, i.e., divide data by 255)
4. Convert the data type to PyTorch tensor type

Example 2-1 : Coding Explanation

Return solution 1.2

```python
def get_feature(self):
    # Image preprocessing
    input=self.process_image()
    #print("input.shape:{}".format(input.shape))
    x=input
    for index,layer in enumerate(self.pretrained_model):
        x=layer(x)
        #print("x:{}".format(x.shape))
        if (index == self.selected_layer):
            return x
```

Feed the preprocessed image into the feature part of VGG16 model to extract the feature from the given layer.
(i.e., the index of layer equal to the given value)

```python
def get_multi_feature(self):
    # Get the feature map
    features=self.get_feature()
    #print(features.shape)
    result_path = './feat_first' + str(self.selected_layer)

    if not os.path.exists(result_path):
        os.makedirs(result_path)
    print("On layer:{}, We can get the {} feature maps".format(self.selected_layer,features.shape[1]))
    #print(features.shape[1])
    for i in range(features.shape[1]):
        feature=features[:,i,:,:]
        feature=feature.view(feature.shape[1],feature.shape[2])
        feature = feature.data.numpy()
        feature = 1.0 / (1 + np.exp(-1 * feature))
        feature = np.round(feature * 255)
        save_name = result_path + '/' + str(i) + '.jpg'
        cv2.imwrite(save_name, feature)
```

1. Create the folder to save the feature map.
2. Use for loop to save the extracted feature maps

Example 2-1 : Coding Explanation

```
def  save_feature_to_img(self):
        #to  numpy
        feature=self.get_single_feature()
        self.get_multi_feature()
        feature=feature.data.numpy()

        #use  sigmod  to  [0,1]
        #  print(feature[0])
        feature=  1.0/(1+np.exp(-1*feature))

        #  to  [0,255]
        feature=np.round(feature*255)
        #print(self.selected_layer)
        save_name  =  './feat_first'  +  str(self.selected_layer)  +  '.jpg'
        cv2.imwrite(save_name,  feature)
```

Call the function of extracting feature maps

Save the sample feature map

Example 2-1 : Coding Explanation

```
def predict(self):
    input=self.process_image()
    outputs = self.pretrained_model2(input)

    s = torch.nn.Softmax(dim=1)
    result = s(outputs)
    self.plot_probablity(result)

    prob, predicted = result.sort(1,descending=True)
    prob = prob.data.numpy()

    predicted = predicted.data.numpy()

    print("Probablity TOP-3:\n")
    print("")
    for i in range(3):

        print("TOP_"+str(i+1))
        print("Probablity:{}".format(prob[0][i]))
        print("Predicted:{}\n".format(c[int(predicted[0][i])]))
    return outputs
```

Call the preprocessed data, feed it into the entire vgg16 pretrained model, and get the output from the classifier.

Call the Softmax function to transform the output value to the probability and plot the figure.

Sort the predicted probability and show the first three value and its class in the ImageNet .

Example 2-2 : Feature Map Visualization

- Please download the "2-2_Feature_map_visualization.zip" on the Moodle and choose your own images from Internet.
- Upload the 2-2_Feature_map_visualization.ipynb and imagenet1000_clsidx_to_labels.txt to the Google Colab.
- Upload the given images "g1.jpg" and "g2.jpg" to the Google Colab.
- Run the codes and get the probabilities of these images.

Example 2-3: Feature Comparison

- Please download the “2-2_Feature_map_visualization.zip” on the Moodle
- Upload the 2-2_Feature_map_visualization.ipynb and imagenet1000_clsidx_to_labels.txt to the Google Colab.
- Upload the given images “g3.jpg” and “g4.jpg” to the Google Colab.
- Run the codes and get the comparisons.




Example 2-2 & 2-3: Feature Map Visualization

```
if __name__=='__main__':
    # get class
    c = {}
    with open("imagenet1000_clsidx_to_labels.txt") as f:
        for line in f:
            (key, val) = line.split(":")
            c[int(key)] = val.split(",")[0]
    # Define image path and select the layer
    myClass=FeatureVisualization('./dog6.jpg',12)
    Compare=FeatureVisualization('./dog9.jpg',12)
    print(myClass.pretrained_model2)

    myClass.save_feature_to_img()
    Compare.save_feature_to_img1()
    print("The first picture classification predict:")
    myClass_vector = myClass.predict()
    print("The second picture classification predict:")
    Compare_vector = Compare.predict()
    #Define cosine similarity
    cos= nn.CosineSimilarity(dim=1)
    #Define Euclidean distance
    euclidean_dist = torch.dist(myClass_vector,Compare_vector,p=2)
    cosine_dist = 1-cos(myClass_vector,Compare_vector)
    print("Verification:")
    if cosine_dist < 0.6:
        print("They are the same!")
        print("Their cosine_distance:{}".format(cosine_dist))
    else:
        print("They are not the same!")
        print("Their cosine_distance:{}".format(cosine_dist))

    print("Their euclidean_dist:{}".format(euclidean_dist))
```

Return solution 1.2

solution 1.4, solution 1.5
solution 1.6, solution 1.7
solution 1.8

Define the Cosine Similarity function

Calculate the Euclidean distance between different pictures
Calculate the Cosine distance between different pictures

Define the threshold

Example 2-2 & 2-3 : Feature Overview

Results:

```
On layer:2, We can get the 64 feature maps
The first picture classification predict:
Probablity TOP-3:


TOP_1
Probablity:0.9882549047470093
Predicted: 'jellyfish'

TOP_2
Probablity:0.007026902399995718
Predicted: 'isopod'

TOP_3
Probablity:0.0019321587169542909
Predicted: 'nematode
```

```
The second picture classification predict:
Probablity TOP-3:


TOP_1
Probablity:0.2607852518558502
Predicted: 'sports car

TOP_2
Probablity:0.2007479369640350З
Predicted: 'beach wagon

TOP_3
Probablity:0.13690434396266937
Predicted: 'convertible'

Verification:
They are not the same!
Their cosine_similarity:tensor([0.0470], grad_fn=<DivBackward0>)
Their euclidean_dist:135.32333374023438
```

The first picture

The second picture

Exercise 2-2: Feature Map Visualization

- Please download the "2-2_Feature_map_visualization.zip" on the Moodle and choose your own images from Internet.
- Upload the 2-2_Feature_map_visualization.ipynb and imagenet1000_clsidx_to_labels.txt to the Google Colab.
- Compare the probability of the images that contain multi classes and different variations (pose, occlusion, age).
- Please write down results and your codes in MS Word to the Moodle




Exercise 2-2 & 2-3: Feature Map Visualization

```
if __name__=='__main__':
    # get class
    c = {}
    with open("imagenet1000_clsidx_to_labels.txt") as f:
        for line in f:
            (key, val) = line.split(":")
            c[int(key)] = val.split(",")[0]
    # Define image path and select the layer
    myClass=FeatureVisualization('./dog6.jpg',12)
    Compare=FeatureVisualization('./dog9.jpg',12)
    print(myClass.pretrained_model2)

    myClass.save_feature_to_img()
    Compare.save_feature_to_img1()
    print("The first picture classification predict:")
    myClass_vector = myClass.predict()
    print("The second picture classification predict:")
    Compare_vector = Compare.predict()
    #Define cosine similarity
    cos= nn.CosineSimilarity(dim=1)
    #Define Euclidean distance
    euclidean_dist = torch.dist(myClass_vector,Compare_vector,p=2)
    cosine_dist = 1-cos(myClass_vector,Compare_vector)
    print("Verification:")
    if cosine_dist < 0.6:
        print("They are the same!")
        print("Their cosine_distance:{}".format(cosine_dist))
    else:
        print("They are not the same!")
        print("Their cosine_distance:{}".format(cosine_dist))

    print("Their euclidean_dist:{}".format(euclidean_dist))
```

Return solution 1.3

Show the predicted classes and probabilities

Exercise 2-2 & 2-3: Feature Map Visualization

```
TOP_1
Probablity:0.730004072189331
Predicted: 'jellyfish'

TOP_2
Probablity:0.055244747549295425
Predicted: 'cup'

TOP_3
Probablity:0.023521317169070244
Predicted: 'vase'
```

```
TOP_1
Probablity:0.773815393447876
Predicted: 'Samoyed'

TOP_2
Probablity:0.09530658274888992
Predicted: 'West Highland white terrier'

TOP_3
Probablity:0.014327057637274265
Predicted: 'komondor'
```

Exercise 2-2 & 2-3: Feature Map Visualization

```
They are not the same!
Their cosine_distance:tensor([0.9956], grad_fn=<RsubBackward1>)
Their euclidean_dist:99.01476287841797
```

The cosine similarity is : 1-0.9956 = 0.0044

solution 1.4, solution 1.5
solution 1.6, solution 1.7
solution 1.8

Exercise 2-4 – Build A Classifier

- Please download the "2-4_CIFAE10.ipynb" on the Moodle, run the sample code, and change the following parameters:
  - Epoch
  - Learning Rate: 0.1, 0.01, 0.001
- Please upload your result and observations in MS Words to the Moodle.

Exercise 2-4 – Build A Classifier

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Define a Convolutional Neural Network

Exercise 2-4 – Build A Classifier

Preprocess function

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

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

net = Net()
```

Initialize the network

Define the dataset and its classes

Exercise 2-4 – Build A Classifier

Return to solution 2.2

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

Change the learning rate: 0.1, 0.01 and 0.001

1. Define the Cross Entropy Loss
2. Define the SGD optimizer which the learning rate is 0.001 and the momentum is 0.9.

Note: If you want to use Adam optimizer, betas = {0.9 and 0.999}, the codes are:

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001, betas=(0.9, 0.999))
```

1. Define the Cross Entropy Loss
2. Define the Adam optimizer which the learning rate is 0.001 and the betas are 0.9 and 0.999.

Exercise 2-4 – Build A Classifier

Return to solution 2.2, 2.8

Epoch number

```
for epoch in range(3):    # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data


        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:     # print every 2000 mini-batches

            print("Epoch : {} steps : {} Training Loss : {}".format(epoch + 1, i + 1, running_loss / 2000) )
            running_loss = 0.0
    save_checkpoint({'net':net.state_dict()}, 'test_epoch{}'.format(epoch+1))

print('Finished Training')
```

1. Feed the input into the model and get the prediction.
2. Use the defined loss function to calculate the loss between the prediction and the label.
3. Use backward() to compute the gradient and use the optimizer.step() to update the weight

Exercise 2-4 – Build A Classifier

The training process of different learning rate:

```
Epoch : 1 steps : 2000 Training Loss : 2.357952641606331
Epoch : 1 steps : 4000 Training Loss : 2.358549834549427
Epoch : 1 steps : 6000 Training Loss : 2.363395507276058
Epoch : 1 steps : 8000 Training Loss : 2.3620128685832023
Epoch : 1 steps : 10000 Training Loss : 2.3548867295980456
Epoch : 1 steps : 12000 Training Loss : 2.3602904160022735
Epoch : 2 steps : 2000 Training Loss : 2.364006470501423
Epoch : 2 steps : 4000 Training Loss : 2.3610252693295477
Epoch : 2 steps : 6000 Training Loss : 2.3646557998657225
Epoch : 2 steps : 8000 Training Loss : 2.3612379420399665
Epoch : 2 steps : 10000 Training Loss : 2.359113136589527
Epoch : 2 steps : 12000 Training Loss : 2.361136519730091
Epoch : 3 steps : 2000 Training Loss : 2.360108473300934
Epoch : 3 steps : 4000 Training Loss : 2.3645326865315437
Epoch : 3 steps : 6000 Training Loss : 2.3571521565318108
Epoch : 3 steps : 8000 Training Loss : 2.3616783508062364
Epoch : 3 steps : 10000 Training Loss : 2.356024751186371
Epoch : 3 steps : 12000 Training Loss : 2.360763477861881
Finished Training
```

Learning rate: 0.1

```
Epoch : 1 steps : 2000 Training Loss : 2.0899574621915815
Epoch : 1 steps : 4000 Training Loss : 1.9607795716822147
Epoch : 1 steps : 6000 Training Loss : 1.9563240223526954
Epoch : 1 steps : 8000 Training Loss : 1.957445238739252
Epoch : 1 steps : 10000 Training Loss : 1.9918364935815334
Epoch : 1 steps : 12000 Training Loss : 1.9577875487208367
Epoch : 2 steps : 2000 Training Loss : 2.011085530459881
Epoch : 2 steps : 4000 Training Loss : 2.0082346482574938
Epoch : 2 steps : 6000 Training Loss : 2.0100900876820087
Epoch : 2 steps : 8000 Training Loss : 2.0044543738663196
Epoch : 2 steps : 10000 Training Loss : 1.969518426090479
Epoch : 2 steps : 12000 Training Loss : 1.9845602488517762
Epoch : 3 steps : 2000 Training Loss : 1.9992908894717694
Epoch : 3 steps : 4000 Training Loss : 2.0016448673307896
Epoch : 3 steps : 6000 Training Loss : 2.016233505010605
Epoch : 3 steps : 8000 Training Loss : 2.028977326095104
Epoch : 3 steps : 10000 Training Loss : 2.01645450925827
Epoch : 3 steps : 12000 Training Loss : 2.056691348493099
Finished Training
```

Learning rate: 0.01

```
Epoch : 1 steps : 2000 Training Loss : 2.1889700249433517
Epoch : 1 steps : 4000 Training Loss : 1.8390005451440812
Epoch : 1 steps : 6000 Training Loss : 1.643870963960886
Epoch : 1 steps : 8000 Training Loss : 1.5679095338881015
Epoch : 1 steps : 10000 Training Loss : 1.492728895097971
Epoch : 1 steps : 12000 Training Loss : 1.4820199556872249
Epoch : 2 steps : 2000 Training Loss : 1.407832405924797
Epoch : 2 steps : 4000 Training Loss : 1.371456946671009
Epoch : 2 steps : 6000 Training Loss : 1.3651220782622695
Epoch : 2 steps : 8000 Training Loss : 1.3291044723726808
Epoch : 2 steps : 10000 Training Loss : 1.284140573028475
Epoch : 2 steps : 12000 Training Loss : 1.2962907982245087
Epoch : 3 steps : 2000 Training Loss : 1.2023748714327813
Epoch : 3 steps : 4000 Training Loss : 1.2107961179297417
Epoch : 3 steps : 6000 Training Loss : 1.1705148331448436
Epoch : 3 steps : 8000 Training Loss : 1.2100771391429006
Epoch : 3 steps : 10000 Training Loss : 1.185110432397574
Epoch : 3 steps : 12000 Training Loss : 1.1803923727944494
Finished Training
```

Learning rate: 0.001

Exercise 2-4 – Build A Classifier

Return to solution 2.2

```
dataiter = iter(testloader)
images, labels = dataiter.next()
```

Use next() to get 1 batch data as a test sample.

```
outputs = net(images)
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]for j in range(4)))
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy : %d %%' % (100 * correct / total))
```

1. Feed the input data into the model and get the value of CIFAR10's 10 classes.
2. Use torch.max to get the maximum score and its index

1. Use for loop to get the entire testing data
2. Feed the testing data and calculate the accuracy

```
Accuracy : 10 %
```

Learning rate: 0.1

```
Accuracy : 25 %
```

Learning rate: 0.01

```
Accuracy : 61 %
```

Learning rate: 0.001

Solutions To The Sample Problems

Problem 1 [30/100]

1. Prob1.ipynb gives you a VGG-16 trained on ImageNet. Upload the "imagenet1000_clsidx_to_labels.txt " and g1.jpg and g2.jpg to the Colab. Use Prob1.ipynb to show the following:

   A. The feature maps and dimensions extracted from Layer 10. [8/30] (Example 2-1, Page 10)

   B. Calculate the Euclidean distance between the images g1.jpg and g2.jpg, which are given with the code. [6/30] (Example 2-2 & 2-3, Page 19)

   C. Please list the changes of dimension when feeding a image to the VGG-16. [16/30] (Example 2-1, Page 10)

Solution 1A

Return solution 1.1

```
if __name__=='__main__':
        # get class
        c = {}
        with open("imagenet1000_clsidx_to_labels.txt") as f:
                for line in f:
                        (key, val) = line.split(":")
                        c[int(key)] = val.split(",")[0]
        # Define image path and select the layer
        first=FeatureVisualization('./g1.jpg',10)
        second=FeatureVisualization('./g2.jpg',10)
```

The feature maps and dimensions extracted from Layer 10.

```
def get_feature(self):
        # Image  preprocessing
        input=self.process_image()
        #print("input.shape:{}".format(input.shape))
        x=input
        for index,layer in enumerate(self.pretrained_model):
                x=layer(x)
                print("x:{}".format(x.shape))
                if (index == self.selected_layer):
                        return x
```

```
x:torch.Size([1, 256, 56, 56])
On layer:10, We can get the 256 feature maps
```

Solution 1B

Return solution 1.2

```
first.save_feature_to_img()
second.save_feature_to_img1()
print("The first picture classification predict:")
first_vector = first.predict()
print("The second picture classification predict:")
second_vector = second.predict()

#Define Euclidean distance
euclidean_dist = torch.dist(first_vector,second_vector,p=2)

print("Verification:")
print("Their euclidean_dist:{}".format(euclidean_dist))
```

```
Verification:
Their euclidean_dist:84.86775970458984
```

Define the Euclidean distance to calculate the distance between the given images.

Solution 1C

```
def  get_feature(self):
     #  Image   preprocessing
     input=self.process_image()
     #print("input.shape:{}".format(input.shape))
     x=input
     for  index,layer  in  enumerate(self.pretrained_model):
          x=layer(x)
          print("x:{}".format(x.shape))
          #  if  (index  ==  self.selected_layer):
          #          return  x
```

Use the “get_feature” function and delete the part of extracting features to make the process of inference complete and observe the feature maps in each layer.

```
x:torch.Size([1, 64, 224, 224])
x:torch.Size([1, 64, 224, 224])
x:torch.Size([1, 64, 224, 224])
x:torch.Size([1, 64, 224, 224])
x:torch.Size([1, 64, 112, 112])
x:torch.Size([1, 128, 112, 112])
```

.
.
.

```
x:torch.Size([1, 128, 112, 112])
x:torch.Size([1, 128, 112, 112])
x:torch.Size([1, 128, 112, 112])
x:torch.Size([1, 128, 56, 56])
x:torch.Size([1, 256, 56, 56])
```

C.Please list the changes of dimension when feeding a image to the VGG-16.

Answer:

[1, 64, 224, 224]

[1, 64, 112, 112]

[1, 128, 112, 112]

[1, 128, 56, 56]

[1, 256, 56, 56]

[1, 256, 28, 28]

[1, 512, 28, 28]

[1, 512, 14, 14]

[1, 512, 7, 7]

Problem 2 [45/100]

1. Please modify the Prob2.ipynb with the following requirements and set the Cross Entropy Loss, Adam Optimizer 0.002 learning rate and betas [0.5,0.999] to train a classifier :

   A. Design a model with the following structure. (Example 2-4, Page 21)

      - First Conv. layer: Input: RGB, Output Channel 16, second Conv. layer: Output Channel 32, third Conv. layer: Output Channel 64.
      - FC-Layer1: Input: Defined by the third convolutional Layer, Output: 1200
      - FC-Layer2: Input: From FC- Layer1, Output: 600
      - FC-Layer3: Input: From FC- Layer2, Output: equal to your class size [16/40]

   B. Save the model and name it as ‘Prob2.pth’ [4/40] (Example 2-4, Page 23)

   C. Save the optimizer and name it as ‘Prob2_1.pth’[6/40] (Example 2-4, Page 23)

Net ( )

Layer type	Input channel	Output channel	Filter size	Stride	padding
Conv1	A	B	3	1	2
ReLU					
AvgPool			2	1	1
Conv2	B	C	2	1	1
ReLU					
AvgPool			2	2	1
Conv3	C	D	2	1	1
ReLU					
AvgPool			2	3	1
Linear1	E	G			
ELU					
Linear2	G	F			
ELU					
Linear3	F	G			

Please crop the parts that you modify in Prob2.ipynb and paste to the solution .docx.

Problem 2

E. Change the dataset to CIFAR10 and the learning rate :0.0002 [5/40] (Example 2-4, Page 26)

F. Load the ‘Prob2.pth and Prob2_1.pth’ obtained from C as pretrained model[8/40] (Hint: torch.load function, Page 33)

G. Train the model on the CIFAR10 dataset[3/40] (Example 2-4, Page 26)

H. Save the model and name it as ‘Prob2_2.pth’ [3/40] (Example 2-4, Page 26)

Solution

Net ( )

Layer type	Input channel	Output channel	Filter size	Stride	padding
Conv1	A:3	B:16	3	1	2
ReLU					
AvgPool			2	1	1
Conv2	B:16	C:32	2	1	1
ReLU					
AvgPool			2	2	1
Conv3	C:32	D:64	2	1	1
ReLU					
AvgPool			2	3	1
Linear1	E:64*49	G:1200			
ELU					
Linear2	G:1200	F:600			
ELU					
Linear3	F:600	G:10			

Return to solution 2.1, 2.8

Solution 2A

Design a model with the given structure.

CIFAR10: RGB, 3 channel

(Input channel, Output channel, Kernel size, Stride, Padding)

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, 1, 2)
        self.relu1 = nn.ReLU()
        self.avgpool1 = nn.AvgPool2d(2, 1, 1)
        self.conv2 = nn.Conv2d(16, 32, 2, 1, 1)
        self.relu2 = nn.ReLU()
        self.avgpool2 = nn.AvgPool2d(2, 2, 1)
        self.conv3 = nn.Conv2d(32, 64, 2, 1, 1)
        self.relu3 = nn.ReLU()
        self.avgpool3 = nn.AvgPool2d(2, 3, 1)

        self.fc1 = nn.Linear(64*7*7, 1200)
        self.elu1 = nn.ELU()
        self.fc2 = nn.Linear(1200, 600)
        self.elu2 = nn.ELU()
        self.fc3 = nn.Linear(600, 10)
```

Channel × Width × Height

10 Classes in CIFAR10

```
def forward(self, x):
    batchsize = x.shape[0]
    x = self.avgpool1(self.relu1(self.conv1(x)))
    x = self.avgpool2(self.relu2(self.conv2(x)))
    x = self.avgpool3(self.relu3(self.conv3(x)))
    # print(x.shape)
    x = x.view(batchsize, -1)
    # print(x.shape)
    x = self.elu1(self.fc1(x))
    x = self.elu2(self.fc2(x))
    x = self.fc3(x)

    return x
```

Before feeding features into Fully-Connected layer, we need to straighten features as (Batchsize, Channel × Width × Height)

(Hint: Using "print" can help us observe the dimensions and can correctly reshape the features)

Solution 2B & 2C

Save the model and name it as ‘Prob2.pth’

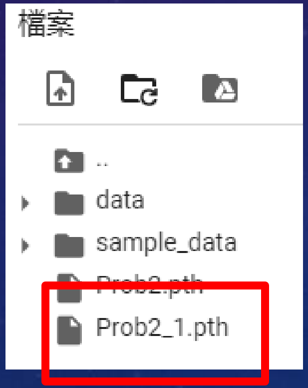
```
Path='Prob2.pth'
torch.save(net.state_dict(),Path)
```

Define the filename and use “torch.save” to save the model weight file.

Save the optimizer and name it as ‘Prob2_1.pth’

```
Path='Prob2_1.pth'
torch.save(optimizer.state_dict(),Path)
```

Define the filename and use “torch.save” to save the optimizer weight file.



Return to solution 2.2

Solution 2E

Change the dataset to CIFAR10 and the learning rate :0.0002

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


trainset = torchvision.datasets.CIFAR10(root='./data', train=True,download=True,transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,shuffle=False, num_workers=2)


classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.0002, momentum=0.9)
```

Return to solution 2.5

Solution 2F

Load the 'Prob2.pth and Prob2_1.pth' obtained from C as pretrained model

```
net = Net()
checkpoint = 'Prob2.pth'
checkpoint = torch.load(checkpoint)
net.load_state_dict(checkpoint)
net = net.cuda()
```

Initialize the Network

GPU mode

Use "torch.load" to load the model weight file.

Through the line, the net will load the pretrained weights in the model weight file.

```
checkpoint = 'Prob2_1.pth'
checkpoint = torch.load(checkpoint)
optimizer.load_state_dict(checkpoint)
```

Use "torch.load" to load the optimizer weight file.

Through the line, the optimizer will load the pretrained weights in the optimizer weight file.

Solution 2G

Train the model on the CIFAR10 dataset

```
for epoch in range(3):  # loop over the dataset multiple times

        running_loss = 0.0
        for i, data in enumerate(trainloader):
            # get the inputs; data is a list of [inputs, labels]
            inputs, labels = data

            inputs = inputs.cuda()
            labels = labels.cuda()

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            if i % 1000 == 999:    # print every 2000 mini-batches

                print("Epoch : {} steps : {} Training Loss : {}".format(epoch + 1, i + 1, running_loss / 1000) )
                running_loss = 0.0

print('Finished Training')
```

→ Covert data to GPU mode

1. Feed the input into the model and get the prediction.
2. Use the defined loss function to calculate the loss between the prediction and the label.
3. Use backward() to compute the gradient and use the optimizer.step() to update the weight

```
Epoch : 1 steps : 1000 Training Loss : 1.0656005302481353
Epoch : 1 steps : 2000 Training Loss : 1.0824362260140479
Epoch : 1 steps : 3000 Training Loss : 1.0971789803281427
Epoch : 1 steps : 4000 Training Loss : 1.0857224909588694
Epoch : 1 steps : 5000 Training Loss : 1.039100713431835
Epoch : 1 steps : 6000 Training Loss : 1.0577989703826607
Epoch : 1 steps : 7000 Training Loss : 1.0318068355247378
```

.
.
.

```
Epoch : 3 steps : 8000 Training Loss : 0.9148140549212694
Epoch : 3 steps : 9000 Training Loss : 0.8839120383523404
Epoch : 3 steps : 10000 Training Loss : 0.905096075758338
Epoch : 3 steps : 11000 Training Loss : 0.9149602136574686
Epoch : 3 steps : 12000 Training Loss : 0.8874567676372827
Finished Training
```

Solution 2H

Save the model and name it as ‘Prob2_2.pth’

```
Path='Prob2_2.pth'
torch.save(net.state_dict(),Path)
```

Define the filename and use “torch.save” to save the second model weight file.

Problem 3 [25/100]

3. The output dimension of the feature map from Conv4 is 64*224*256, please calculate the dimension of the Input and the feature maps from Conv1, Conv2, Conv3, Conv5

Layer type	**Input channel**	**Output channel**	**Filter size**	**Stride**
Conv1	3	8	3	1
AvgPool1			4	1
Conv2	8	16	4	2
MaxPool1			2	2
Conv3	16	32	2	1
MaxPool2			2	1
Conv4	32	64	3	2
AvgPool2			3	1
Conv5	64	128	7	1

Solution3

$$Output = \frac{Input - kernel\ size + 2 \times Padding}{Stride} + 1$$

$$Input = (Output - 1) \times Stride\ - 2 \times Padding + kernel\ size$$

- Input-> [3,1811,2067]
- Conv1-> [8, 1809, 2065]
- AvgPool1-> [8, 1806, 2062]
- Conv2-> [16, 902, 1030]
- MaxPool1-> [16, 451, 515]
- Conv3-> [32, 450, 514]
- MaxPool2-> [32, 449, 513]
- Conv4-> [64, 224, 256]

$1811 = (1809 - 1) \times 1\ - 2 \times 0 + 3$, $2067 = (2065 - 1) \times 1\ - 2 \times 0 + 3$

$1809 = (1806 - 1) \times 1\ - 2 \times 0 + 4$, $2065 = (2062 - 1) \times 1\ - 2 \times 0 + 4$

$1806 = (902 - 1) \times 2\ - 2 \times 0 + 4$, $2062 = (1030 - 1) \times 2\ - 2 \times 0 + 4$

$902 = (451 - 1) \times 2\ - 2 \times 0 + 2$, $1030 = (515 - 1) \times 2\ - 2 \times 0 + 2$

$451 = (450 - 1) \times 1\ - 2 \times 0 + 2$, $515 = (514 - 1) \times 1\ - 2 \times 0 + 2$

$450 = (449 - 1) \times 1\ - 2 \times 0 + 2$, $514 = (513 - 1) \times 1\ - 2 \times 0 + 2$

$449 = (224 - 1) \times 2\ - 2 \times 0 + 3$, $513 = (256 - 1) \times 2\ - 2 \times 0 + 3$

Return to solution 3

Solution3

- AvgPool2->[64, 222, 254]
- Conv5-> [128, 216, 248]

$$222 = \frac{224 - 3 + 2 \times 0}{1} + 1\,, \qquad 254 = \frac{256 - 3 + 2 \times 0}{1} + 1$$

$$216 = \frac{222 - 7 + 2 \times 0}{1} + 1\,, \qquad 248 = \frac{254 - 7 + 2 \times 0}{1} + 1$$