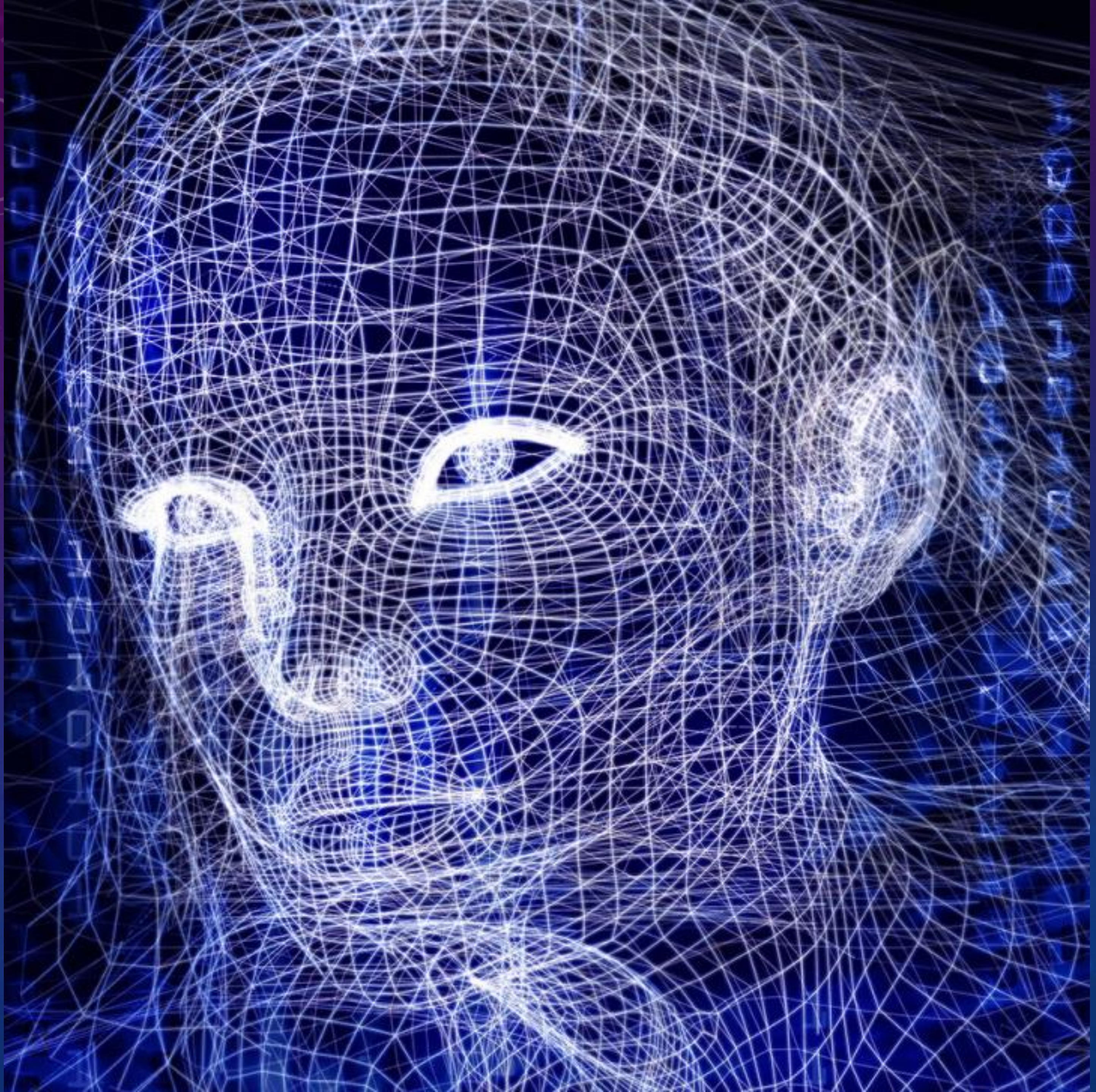
COMPUTER VISION AND ITS APPLICATION

CODING SKILLS FOR USING CNN

徐繼聖

Gee-Sern Jison Hsu

National Taiwan University of Science and Technology



Content Overview

- Feature Map Visualization and Its Dimension
- How To Training A Classifier and Codes of Building A Neural Network
- Other Examples For Studying
- CNN Review Problem

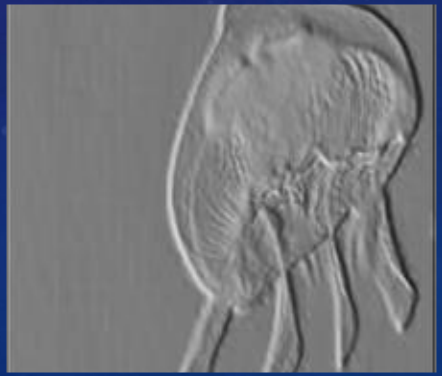
Content Overview

- Feature Map Visualization and Its Dimension
- How To Training A Classifier and Codes of Building A Neural Network
- Other Examples For Studying
- CNN Review Problem

Feature Map Visualization and Its Dimension

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.



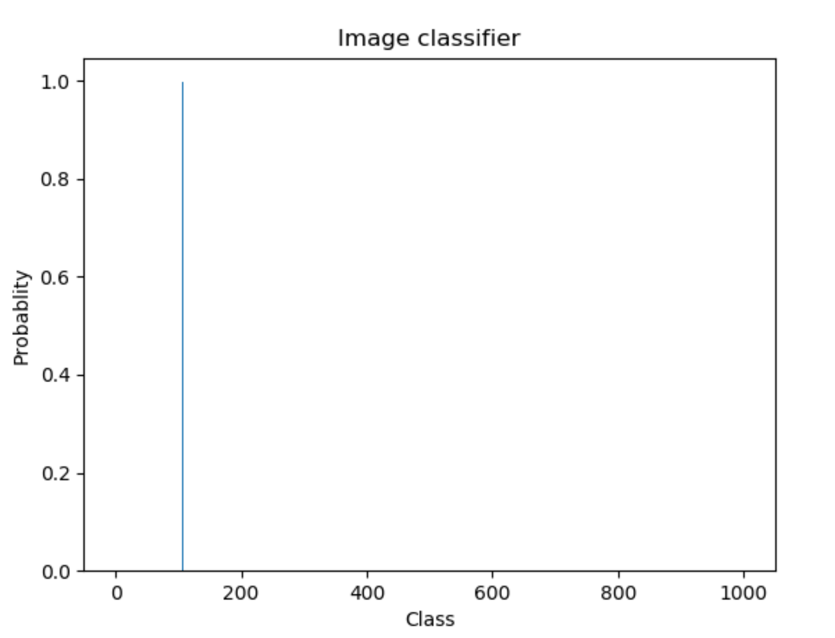
Example 2-1: Feature Map Visualization

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



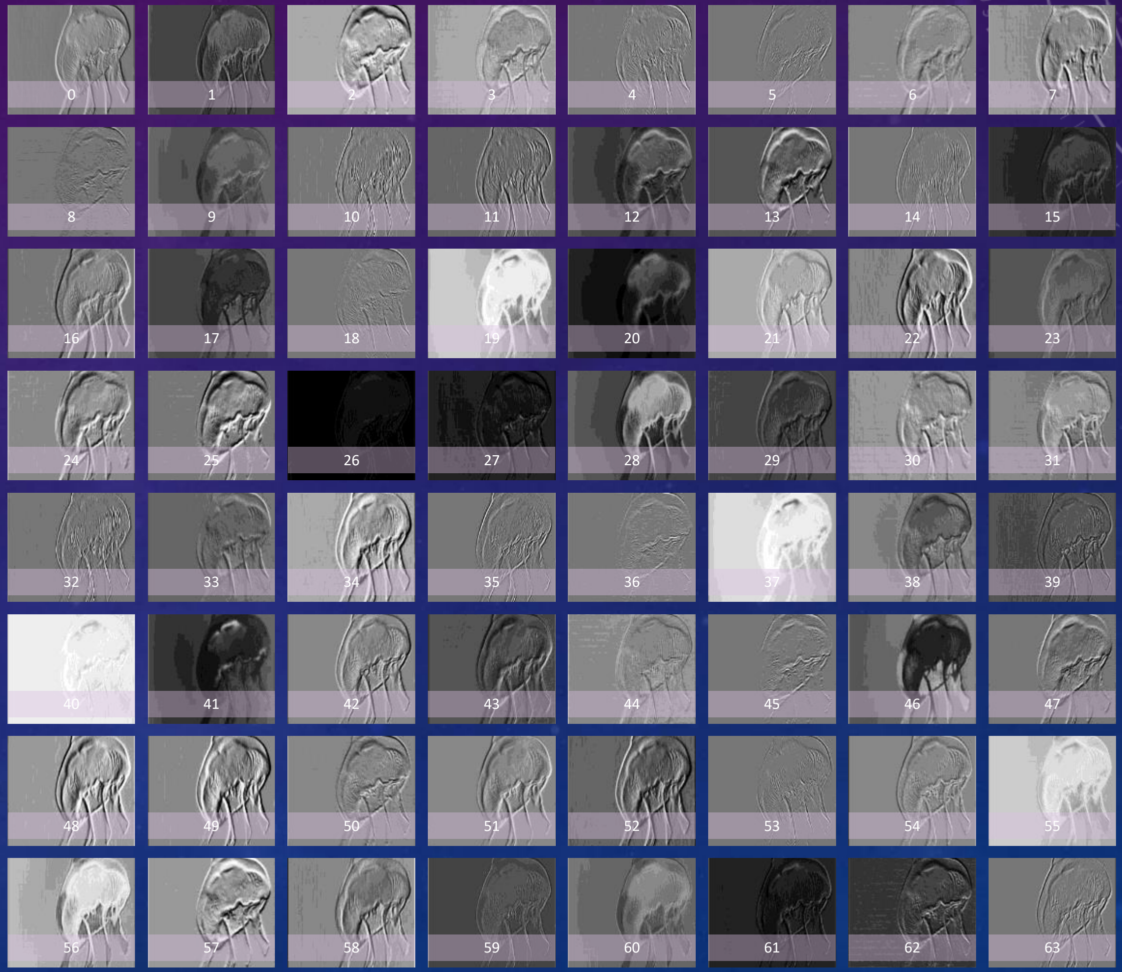
Probability of the classes

```
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

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

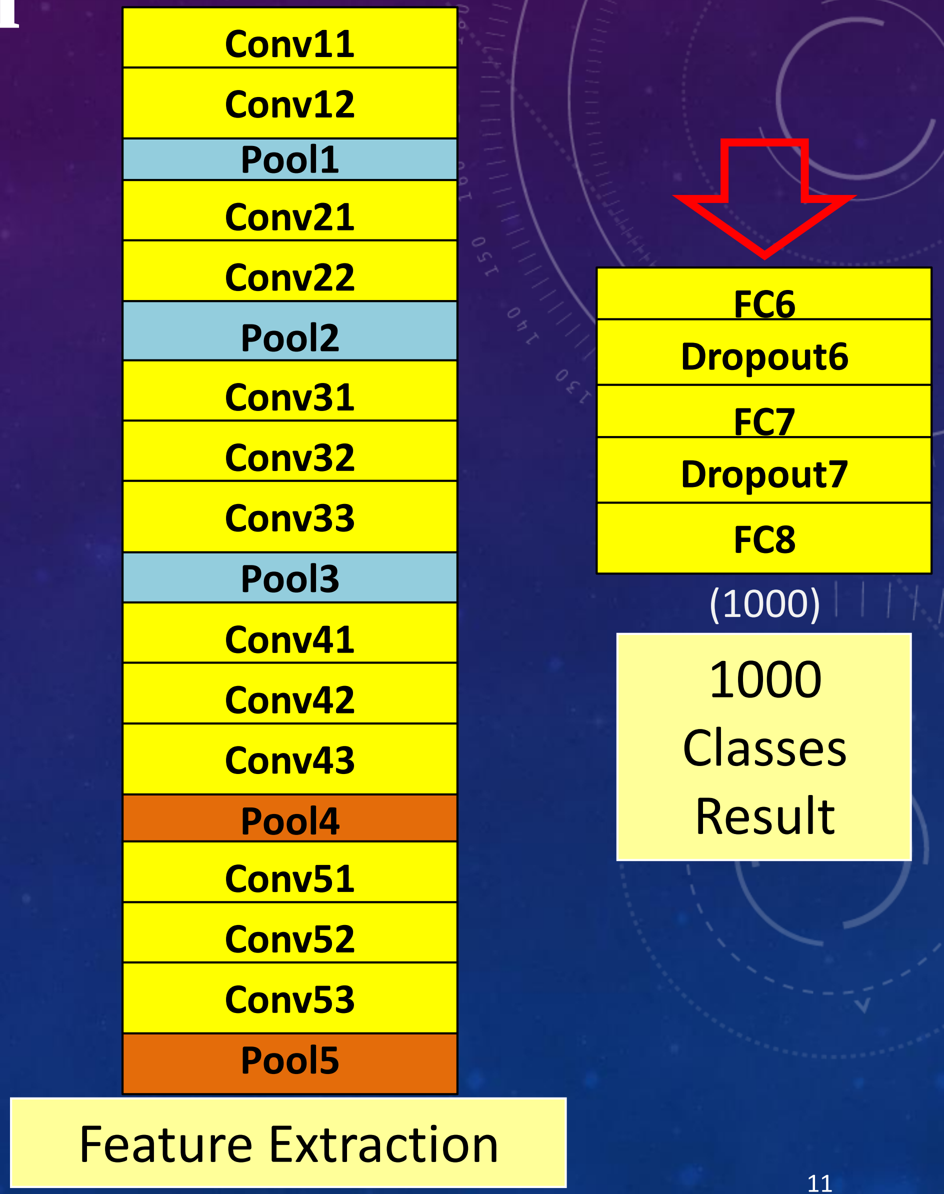
Example 2-1 : Coding Explanation

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

```python
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.

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

        # Resize image
1.      if resize_im:
                cv2im = cv2.resize(cv2im, (224, 224))
        im_as_arr = np.float32(cv2im)
2.      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
3.      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
4.      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

```
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)

```
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 .

Exercise 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.
- Choose your own images from Internet.
- Compare the feature maps that extract from layer 5 and observe the size and dimension of the feature maps.

Please write down your results and codes in MS Word, then upload to the Moodle.

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))
```

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:0.00702690239995718
Predicted: 'isopod'

TOP_3
Probablity:0.001932158716954290 9
Predicted: 'nematode
```

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


TOP_1
Probablity:0.2607852518558502
Predicted: 'sports car

TOP_2
Probablity:0.2007479369640350 3
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 “2-1_Feature_map_visualization.zip” on the Moodle and choose your own images from Internet.
- Upload the 2-1_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 2-3: Feature Comparison

- Please download the "2-3_Feature_map_visualization.zip" on the Moodle
- Upload the 2-3_Feature_map_visualization.ipynb and imagenet1000_clsidx_to_labels.txt to the Google Colab.
- Please use the ResNet-50 pretrained model provided by Pytorch Package.
- Compare the similarity of the images that contain two classes and different variations (pose, occlusion, age).
- Please write down result and your code in MS Words to the Moodle




Codes of Building A Neural Network and How To Training A Classifier

Example 2-4: Build A Classifier

Define a Convolutional Neural Network

Input size : 3 * 32 * 32

Input channel

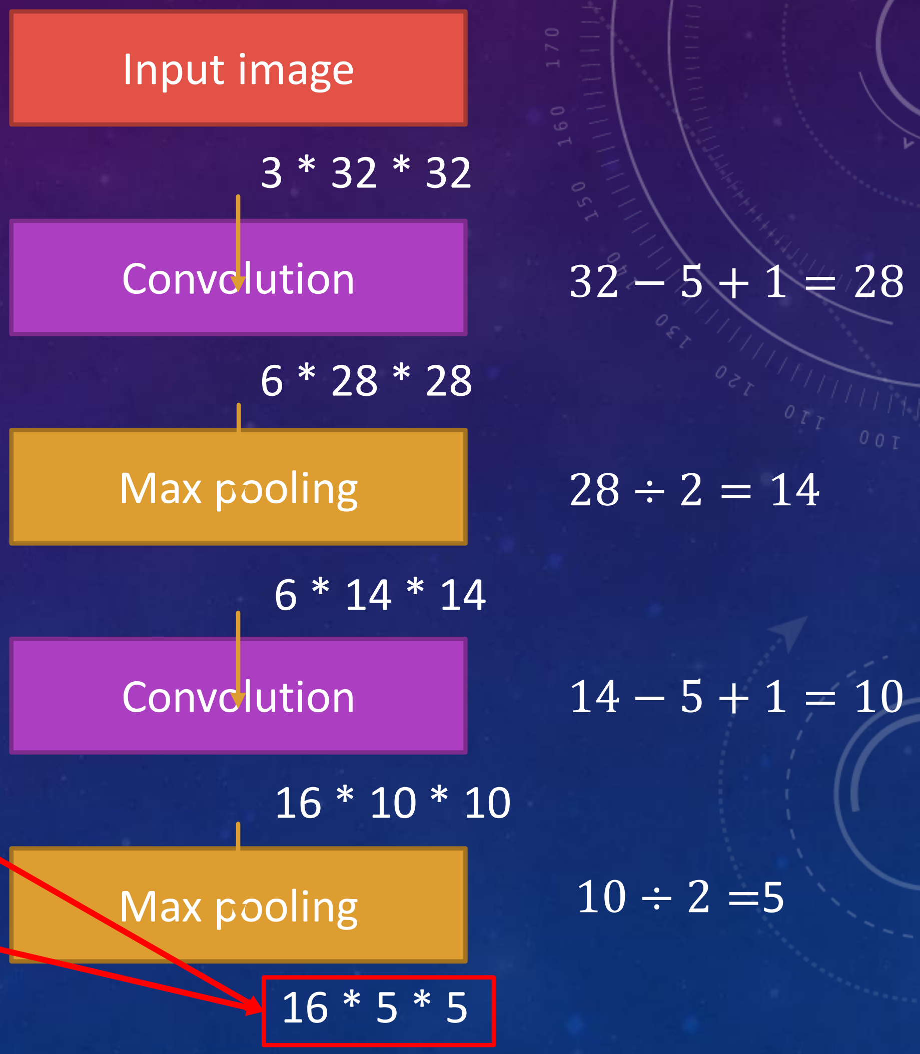
Output channel

Kernel size

```python
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
```

View function

Define forward path



View function

View function is a tool to Reshape the tensor.

```
import torch
a = torch.range(1, 16)
```

a = a tensor has 16 elements from 1 to 16

Try to reshape the tensor to 4 x 4

```
a = a.view(4, 4)
```

What is the meaning of parameter -1?

If there is any situation that you don't know how many rows you want but are sure of the number of columns, then you can specify this with a -1. (Note that you can extend this to tensors with more dimensions. Only one of the axis value can be -1).

This is a way of telling the library: "give me a tensor that has these many columns and you compute the appropriate number of rows that is necessary to make this happen".

Note : after the reshape the total number of elements need to remain the same

Example 2-4: Build A Classifier

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


def save_checkpoint(state, save_dir, filename='checkpoint.pth'):
    save_name = save_dir + '_{}'.format(filename)
    torch.save(state, save_name)
```

Set the filename of the weight file and use the function "torch.save" to save the weight file (Hint: "torch.load" can be used for loading the weight file.

Normalize a tensor image with mean and standard deviation. Given mean: (mean[1],...,mean[n]) and std: (std[1],..,std[n]) for n channels, this transform will normalize each channel of the input torch.*Tensor
i.e., output[channel] = (input[channel] - mean[channel]) / std[channel]

```
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()
```

CIFAR10 dataset in Pytorch

Setting the datasets path

There are 10 classes in CIFAR10

Example 2-4: Build A Classifier

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

Select Cross-Entropy loss for classification

Using SGD optimizer, learning rate :0.001, momentum: 0.9

```
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')
```

Setting the value of epoch (Hint: range(3) means 0 to 2)

Use for loop to iter the batch data.

Each batch data contains the image data and the corresponding label.

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

Print the loss every 2000 steps

Call the function to save the weight file.

Example 2-4: Build A Classifier

```
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

Image Classification

CS231n: Convolutional Neural Networks for Visual Recognition

http://cs231n.github.io/classification/

Example - Training A Classifier

1. Loading and normalizing CIFAR10

Setting the datasets path

```
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’)
```

There are 10 classes in CIFAR10

Example - Training A Classifier

3. Define a Loss function and optimizer

Select Cross-Entropy loss for classification

```
criterion = nn.CrossEntropyLoss()

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

Using SGD optimizer

Example - Training A Classifier

4. Train the network and save the checkpoint

```
for epoch in range(10):

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data

        optimizer.zero_grad()

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 2000 == 1999:
            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))
```

Define the epoch

Backpropagation the loss to update the weights.

Save the checkpoint

Model Training

```
outputs = model(inputs)
loss = criterion(outputs, targets)
# Backward and optimize
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

- optimizer.zero_grad():Because every time a variable is back propagated through, the gradient will be accumulated instead of being replaced.
- loss.backward(): Backward
- optimizer.step(): Parameters update based on the current gradient.

Example - Training A Classifier

5. Test the network on the test data

```
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 of the network on the 10000 test images: %d %%' % (100 * correct / total))
```

In the testing stage, the weights are fixed.

Check if predicted is the same as the labeled (G.T.)

Result:

```
Epoch : 10 steps : 8000 Training Loss : 0.8649811625033617
Epoch : 10 steps : 10000 Training Loss : 0.8769527244269848
Epoch : 10 steps : 12000 Training Loss : 0.8830881424993277
Finished Training
Accuracy : 61 %
```

Example: Result Comparison on CIFAR10

Use Pretrain model testing CIFAR10 Dataset

```
transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=100, shuffle=False)

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

# Choose Your Testing Model
print('==> Building model..')
net = VGG('VGG16')
#net = ResNet18()
#net = GoogLeNet()
```

Load pretrain architecture

```
# Load checkpoint.
print('==> Loading pretrained model from checkpoint..')
assert os.path.isdir('checkpoint'), 'Error: no checkpoint directory found!'
checkpoint = torch.load('./checkpoint/VGG16.pth')#checkpoint path
net.load_state_dict(checkpoint['net'])
```

**Load pretrain model

VGG16

```
-----Start Testing------
Loss: 0.745 | Acc: 75.590% (7559/10000)
```

ResNet18

```
------Start Testing------
Loss: 0.727 | Acc: 76.210% (7621/10000)
```

GoogleNet

```
------Start Testing------
Loss: 0.938 | Acc: 68.100% (6810/10000)
```

Use Your training model testing CIFAR10 Dataset

```
# your train model net
net= Net()
print(net)
net = net.to(device)
if device == 'cuda':
    net = torch.nn.DataParallel(net)
    cudnn.benchmark = True
```

**Load your training architecture

Your training model result

```
------Start Testing------
Loss: 2.372 | Acc: 47.000% (4700/10000)
```

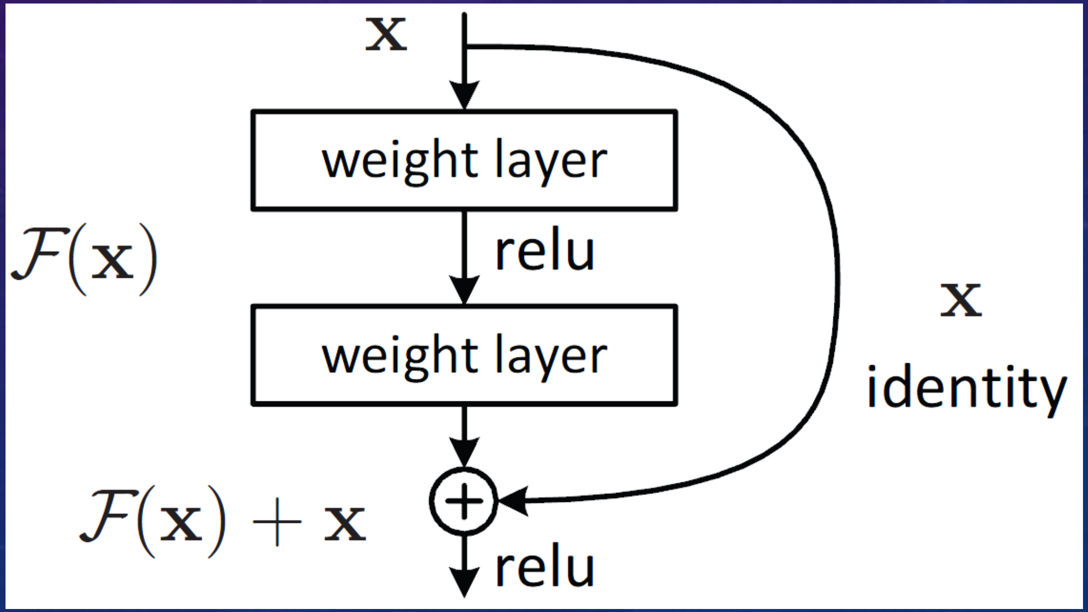
```
# Load your taining checkpoint.
checkpoint = torch.load('./checkpoint/test_epoch3_checkpoint.pth')
net.module.load_state_dict(checkpoint['net'])
```

**Load your training model

Comparison of Deep Networks

ResNet

- Since AlexNet, the state-of-the-art CNN architecture is going deeper and deeper.
- However, increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the notorious vanishing gradient problem
- The core idea of ResNet is introducing a so-called “identity shortcut connection” that skips one or more layers, as shown in the following figure:



Why ResNets Work

https://www.youtube.com/watch?v=RYth6EbBUqM&ab_channel=Deeplearning.ai[9:12]

Example 2-5: Comparison of Deep Networks

- Considering the following networks
  - N_1 has 10 conv layers and 1 Fc layer with Setup-1, please train N_1 on the CIFAR-10 for 3 epochs;
  - N_2 has 10 conv layers with specified shortcut connections and 1 Fc layer with Setup-2, please train N_2 on the CIFAR-10 for 3 epochs;

Example 2-5: Comparison of Deep Networks

Loading and normalizing CIFAR10

Setting the datasets path

```
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=8)

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=8)

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

Normalize a tensor image with mean and standard deviation. Given mean: (mean[1],...,mean[n]) and std: (std[1],..,std[n]) for n channels, this transform will normalize each channel of the input torch.*Tensor i.e., output[channel] = (input[channel] - mean[channel]) / std[channel]

There are 10 classes in CIFAR10

Example 2-5: Comparison of Deep Networks

Setup-1 Define network(without shortcut)(10conv 1fc)

```
net = Net().to(device)
```

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(),)
        self.conv2 = nn.Sequential(
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(),)
```

```
        self.conv6 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(),)
        self.conv7 = nn.Sequential(
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(),)
```

```
self.conv10 = nn.Sequential(
    nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1, bias=False),
    nn.BatchNorm2d(128),
    nn.ReLU(),)
self.fc1 = nn.Linear(131072, 10)
```

```
def forward(self, x):
    x = self.conv1(x)
    x = self.conv2(x)
    x = self.conv3(x)
    x = self.conv4(x)
    x = self.conv5(x)
    x = self.conv6(x)
    x = self.conv7(x)
    x = self.conv8(x)
    x = self.conv9(x)
    x = self.conv10(x)


    x = x.view(-1, 131072)
    x = F.relu(self.fc1(x))

    return x
```

Define forward path

Example 2-5: Comparison of Deep Networks

Setup-1   Print network(without shortcut)(10conv 1fc)

```
Net(
  (conv1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv2): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv3): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv4): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv5): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
```

```
  (conv6): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv7): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv8): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv9): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv10): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (fc1): Linear(in_features=131072, out_features=10, bias=True)
```

Example 2-5: Deep Network Comparison

Setup-2 Define network(with shortcut)(10conv 1fc)

```
net = ResNet18().to(device)
```

```
class ResidualBlock(nn.Module):
    def __init__(self, inchannel, outchannel, stride=1):
        super(ResidualBlock, self).__init__()
        self.left = nn.Sequential(
            nn.Conv2d(inchannel, outchannel, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(outchannel),
            nn.ReLU(inplace=True),
            nn.Conv2d(outchannel, outchannel, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(outchannel)
        )
        self.shortcut = nn.Sequential()
        if stride != 1 or inchannel != outchannel:
            self.shortcut = nn.Sequential(
                nn.Conv2d(inchannel, outchannel, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(outchannel)
            )

    def forward(self, x):
        out = self.left(x)
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

Define shortcut

```
def ResNet18():

    return ResNet(ResidualBlock)
```

```
class ResNet(nn.Module):
    def __init__(self, ResidualBlock, num_classes=10):
        super(ResNet, self).__init__()
        self.inchannel = 64
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(),
        )
        self.layer1 = self.make_layer(ResidualBlock, 64,  2, stride=1)
        self.layer2 = self.make_layer(ResidualBlock, 128, 2, stride=2)
        #self.layer3 = self.make_layer(ResidualBlock, 256, 2, stride=2)
        #self.layer4 = self.make_layer(ResidualBlock, 512, 2, stride=2)
        self.fc = nn.Linear(2048, num_classes)

    def make_layer(self, block, channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)   #strides=[1,1]
        layers = []
        for stride in strides:
            layers.append(block(self.inchannel, channels, stride))
            self.inchannel = channels
        return nn.Sequential(*layers)

    def forward(self, x):
        out = self.conv1(x)
        out = self.layer1(out)
        out = self.layer2(out)
        #out = self.layer3(out)
        #out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out
```

Define forward path

Example 2-5: Deep Network Comparison

Setup-2 Print network(with shortcut)(10conv 1fc)

```
ResNet(
  (conv1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (layer1): Sequential(
    (0): ResidualBlock(
      (left): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (shortcut): Sequential()
    )
    (1): ResidualBlock(
      (left): Sequential(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
        (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (shortcut): Sequential()
    )
  )
```

```
(layer2): Sequential(
  (0): ResidualBlock(
    (left): Sequential(
      (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (shortcut): Sequential(
      (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): ResidualBlock(
    (left): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (shortcut): Sequential()
  )
)
(fc): Linear(in_features=2048, out_features=10, bias=True)
```

shortcut

Example 2-5: Comparison of Deep Networks

Define a Loss function and optimizer

Select Cross-Entropy loss for classification

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

Using Adam optimizer

Example 2-5: Comparison of Deep Networks

Train the network and save the checkpoint

```
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
        inputs, labels = inputs.to(device), labels.to(device)

        # 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')
```

Define the epoch

Backpropagation the loss to update the weights.

Save the checkpoint

Example 2-5: Comparison of Deep Networks

Test the network on the test data

```
correct = 0
total = 0
with torch.no_grad():
        for data in testloader:
                images, labels = data
                images, labels =images.to(device), labels.to(device)
                outputs = net(images)

                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

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

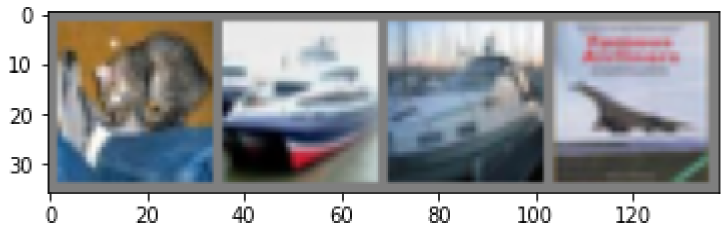
In the testing stage, the weights are fixed.

Check if predicted is the same as the labeled (G.T.)

Example 2-5: Comparison of Deep Networks

Result(with shortcut)

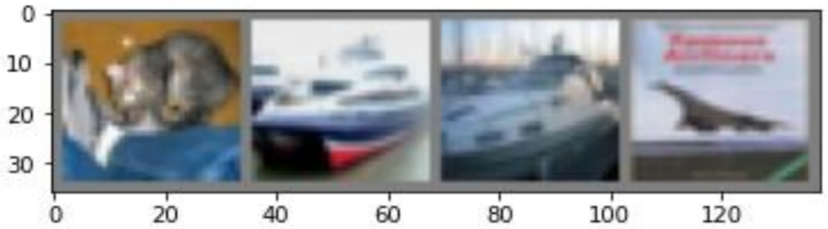
```
Epoch : 1 steps : 2000 Training Loss : 2.314868042707443
Epoch : 1 steps : 4000 Training Loss : 2.3025851249694824
Epoch : 1 steps : 6000 Training Loss : 2.3025851249694824
Epoch : 1 steps : 8000 Training Loss : 2.3025851249694824
Epoch : 1 steps : 10000 Training Loss : 2.3025851249694824
Epoch : 1 steps : 12000 Training Loss : 2.3025851249694824
Epoch : 2 steps : 2000 Training Loss : 2.3025851249694824
Epoch : 2 steps : 4000 Training Loss : 2.3025851249694824
Epoch : 2 steps : 6000 Training Loss : 2.3025851249694824
Epoch : 2 steps : 8000 Training Loss : 2.3025851249694824
Epoch : 2 steps : 10000 Training Loss : 2.3025851249694824
Epoch : 2 steps : 12000 Training Loss : 2.3025851249694824
Epoch : 3 steps : 2000 Training Loss : 2.3025851249694824
Epoch : 3 steps : 4000 Training Loss : 2.3025851249694824
Epoch : 3 steps : 6000 Training Loss : 2.3025851249694824
Epoch : 3 steps : 8000 Training Loss : 2.3025851249694824
Epoch : 3 steps : 10000 Training Loss : 2.3025851249694824
Epoch : 3 steps : 12000 Training Loss : 2.3025851249694824
Finished Training
```




```
GroundTruth:    cat  ship  ship plane
Predicted:  horse horse horse horse
Accuracy : 10 %
```

Result(without shortcut)

```
Epoch : 1 steps : 2000 Training Loss : 1.8878891510665416
Epoch : 1 steps : 4000 Training Loss : 1.4781636514812708
Epoch : 1 steps : 6000 Training Loss : 1.2983864627033472
Epoch : 1 steps : 8000 Training Loss : 1.1348232387583703
Epoch : 1 steps : 10000 Training Loss : 1.0744273342452944
Epoch : 1 steps : 12000 Training Loss : 0.9819176591066644
Epoch : 2 steps : 2000 Training Loss : 0.8823310074708425
Epoch : 2 steps : 4000 Training Loss : 0.8428827857617289
Epoch : 2 steps : 6000 Training Loss : 0.8335133502772077
Epoch : 2 steps : 8000 Training Loss : 0.8076976113315905
Epoch : 2 steps : 10000 Training Loss : 0.7748700085091405
Epoch : 2 steps : 12000 Training Loss : 0.7552551930428016
Epoch : 3 steps : 2000 Training Loss : 0.6636658706957823
Epoch : 3 steps : 4000 Training Loss : 0.6618142743564677
Epoch : 3 steps : 6000 Training Loss : 0.6509611685594427
Epoch : 3 steps : 8000 Training Loss : 0.6368713211108697
Epoch : 3 steps : 10000 Training Loss : 0.650435102526215
Epoch : 3 steps : 12000 Training Loss : 0.6216317716715858
Finished Training
```




```
GroundTruth:    cat  ship  ship plane
Predicted:    dog  ship   car plane
Accuracy : 76 %
```

Exercise 2-5: Comparison of Deep Networks

- Please download 2-5 Deep network comparsion.ipynb from moodle
- Modify the **Net's** architecture and make its gradient not disappear (can be deleted layer, or change the components)

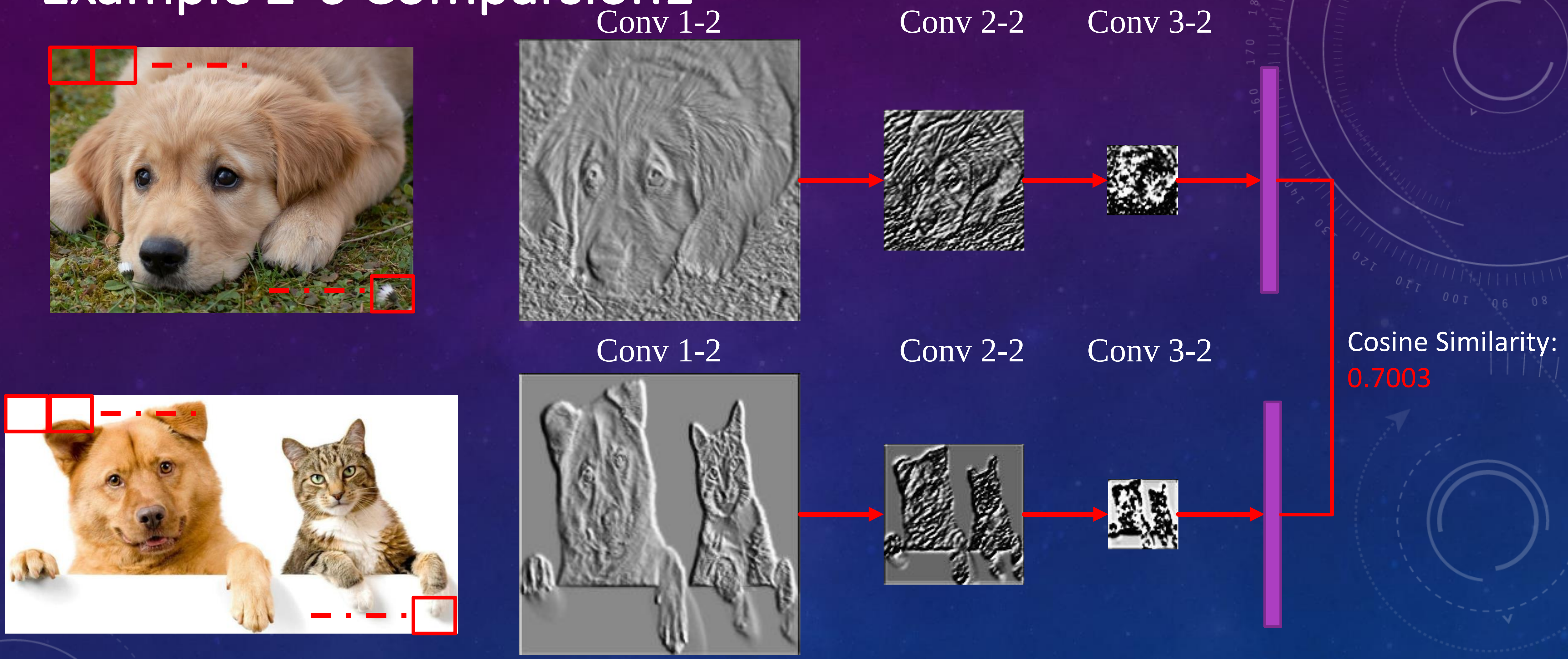
Upload your observations, comments and your code to Moodle in a docx.

Comparison of Latent Vector

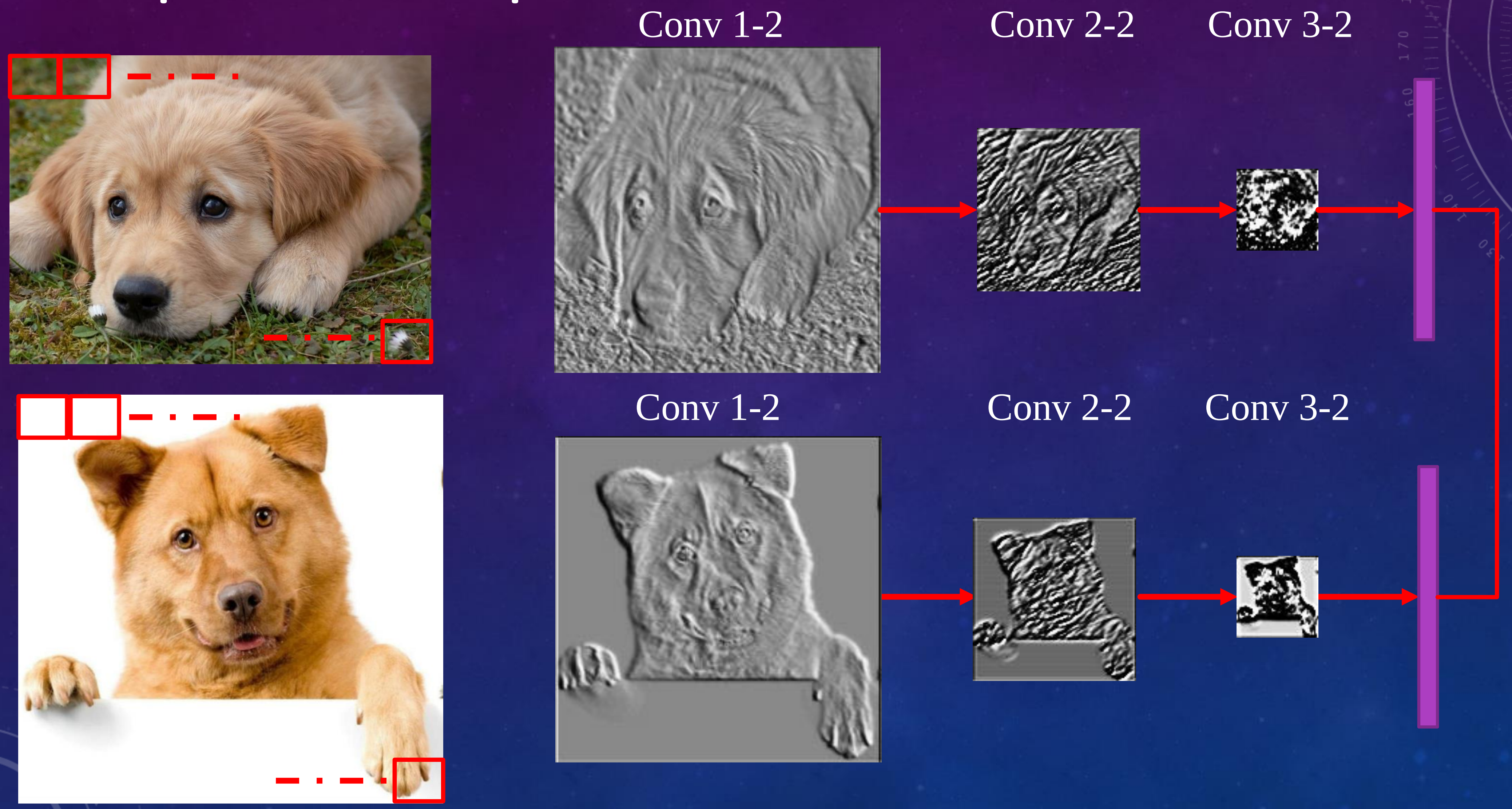
Example 2-6: Comparison of Latent Vector

- Use VGG16-pretrain to predict the image and compare the similarity between two images, when there is only one class and two classes, and the difference the scores when the crop is in different places.

Example 2-6 Comparsion1

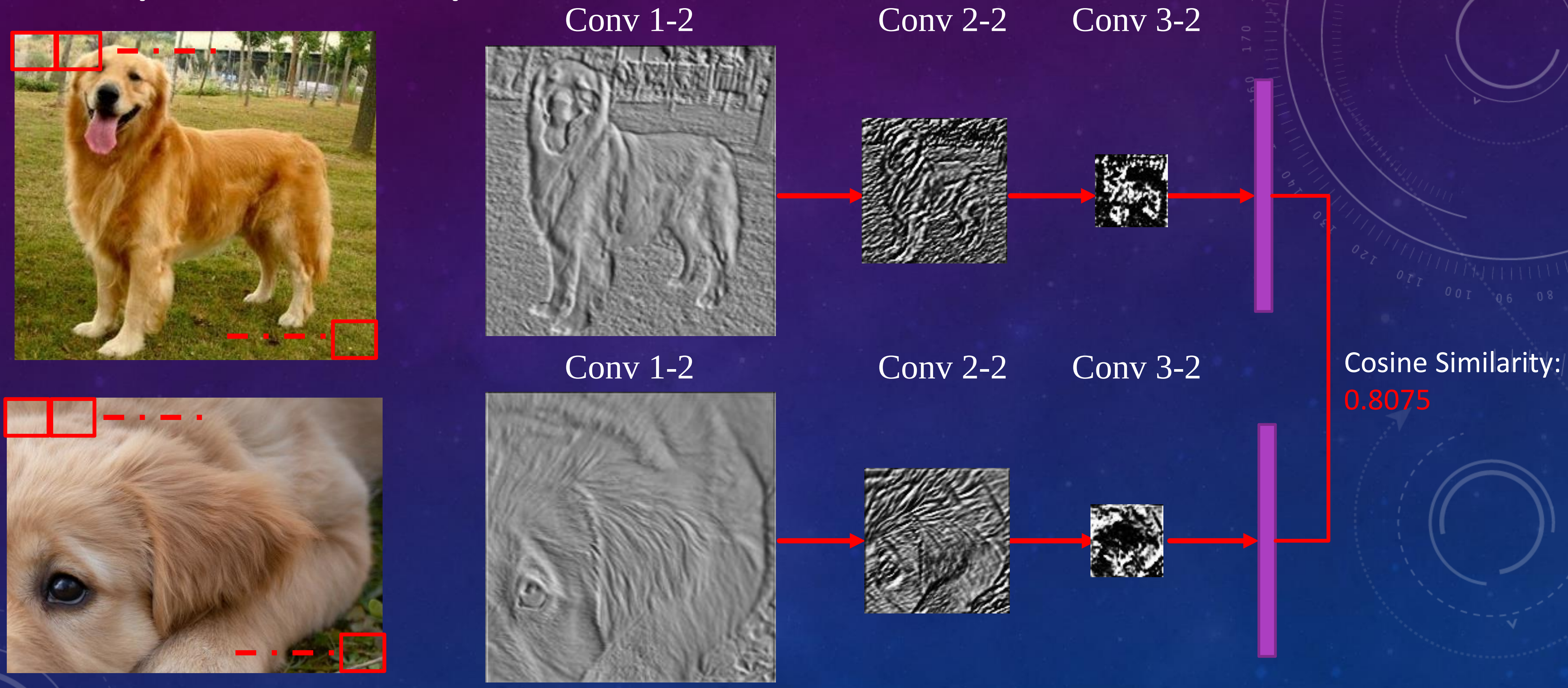


Example 2-6 Comparsion1

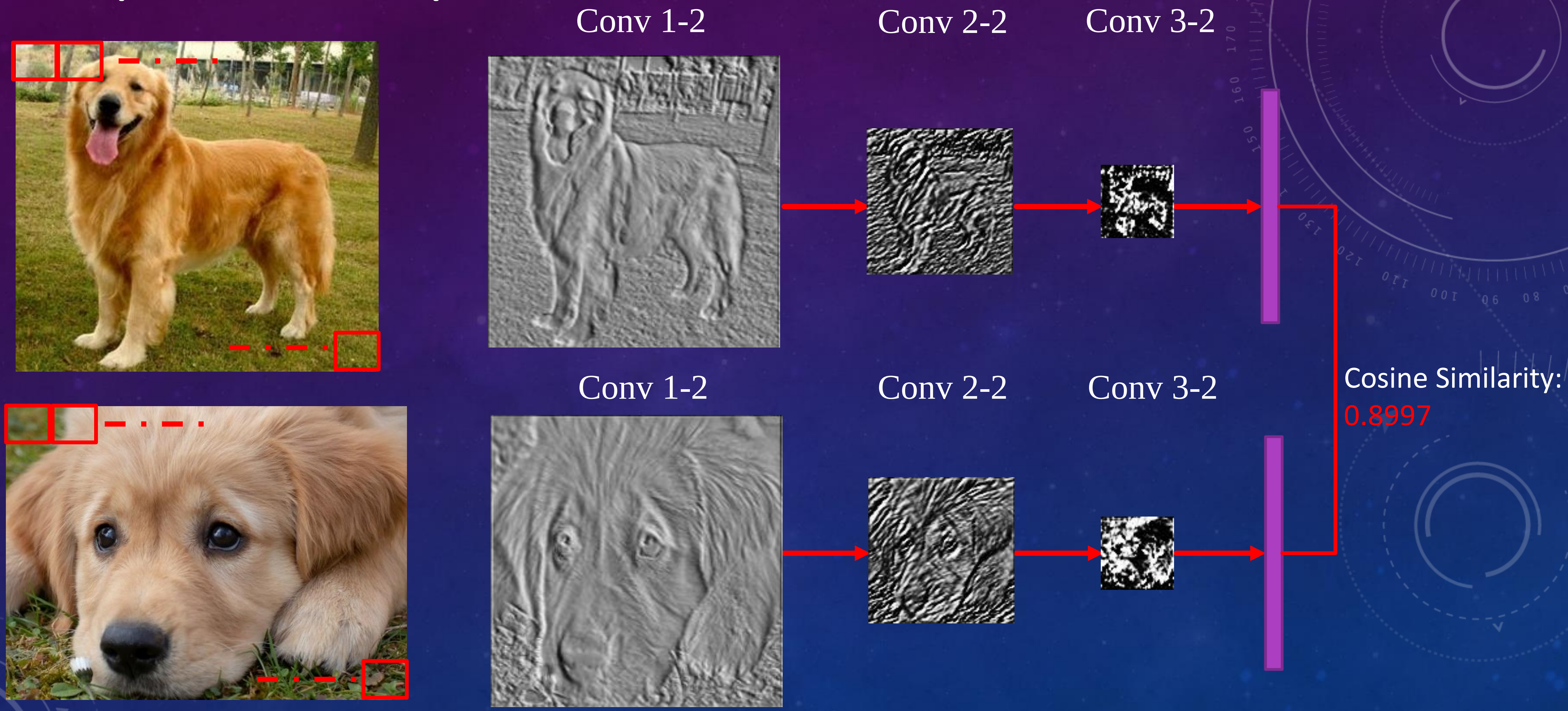


Cosine Similarity:
0.7871

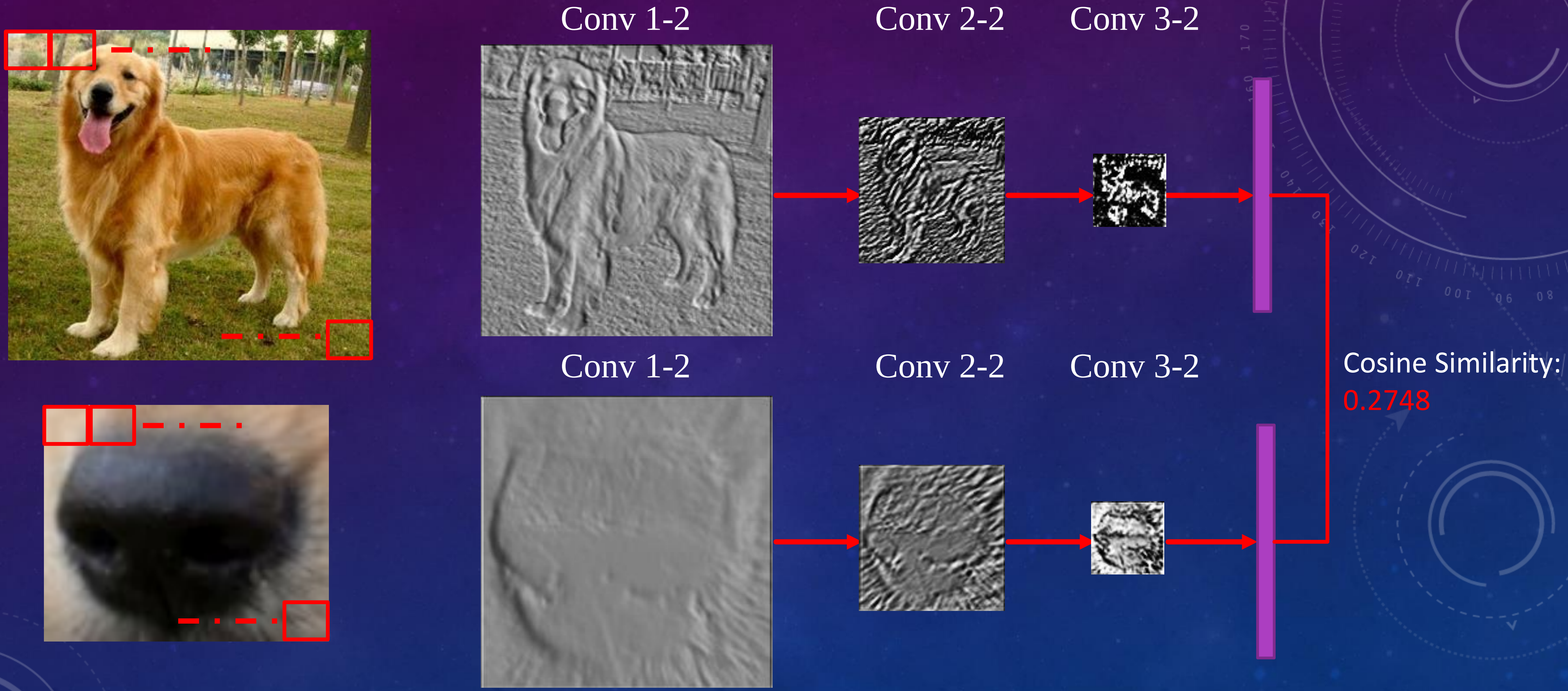
Example 2-6 Comparsion2



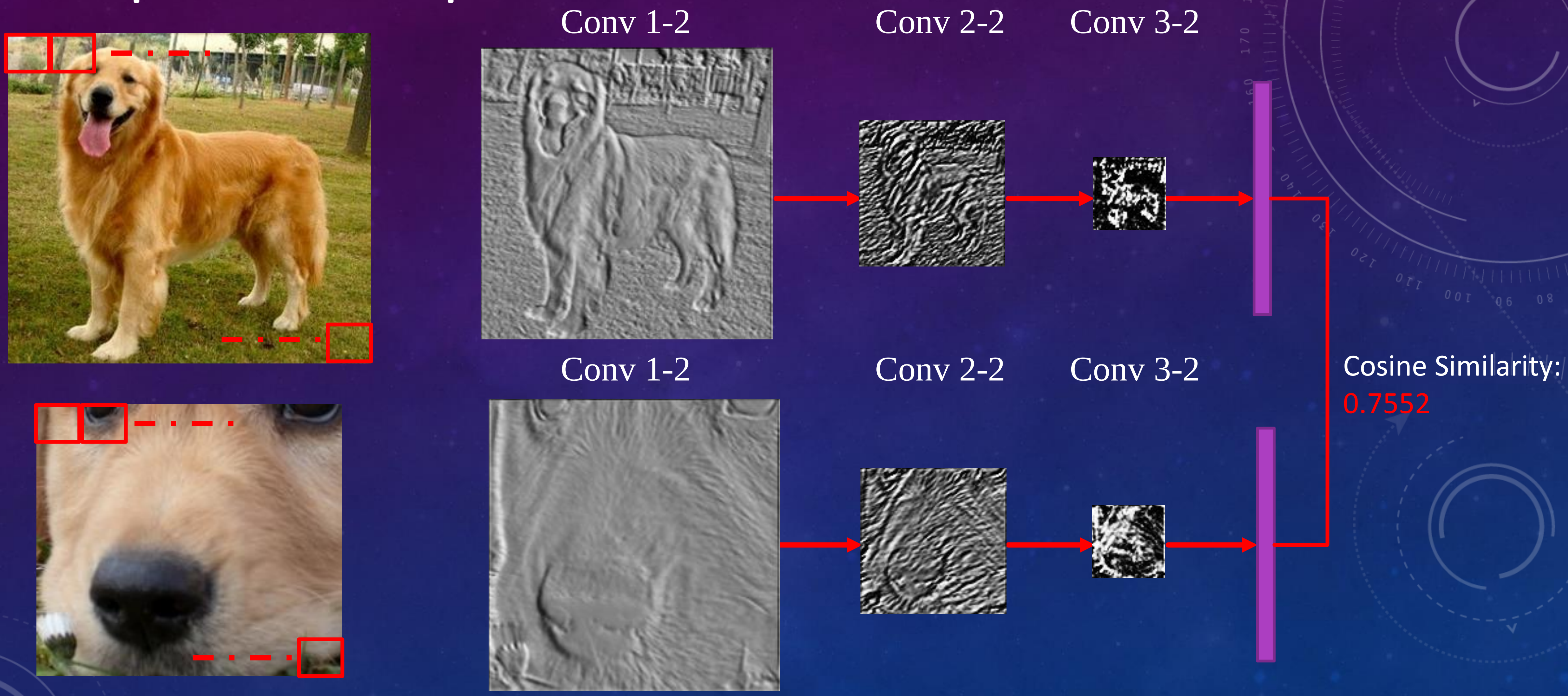
Example 2-6 Comparsion2



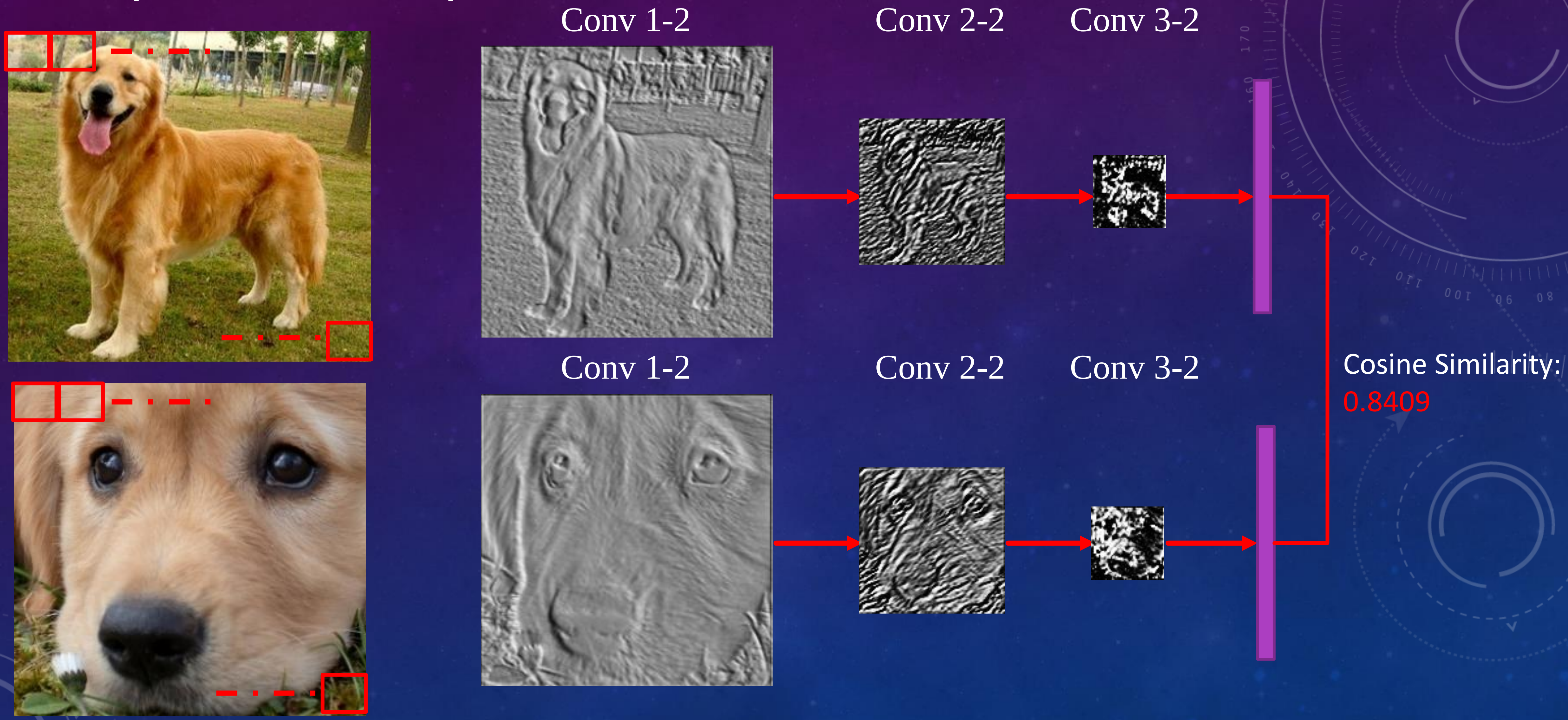
Example 2-6 Comparsion3



Example 2-6 Comparsion3



Example 2-6 Comparsion3



Review Sample

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)

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)

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
AvgPool			4	1
Conv2	8	16	4	2
MaxPool			2	2
Conv3	16	32	2	1
MaxPool			2	1
Conv4	32	64	3	2
AvgPool			3	1
Conv5	64	128	7	1

Example 2-4, Page 24