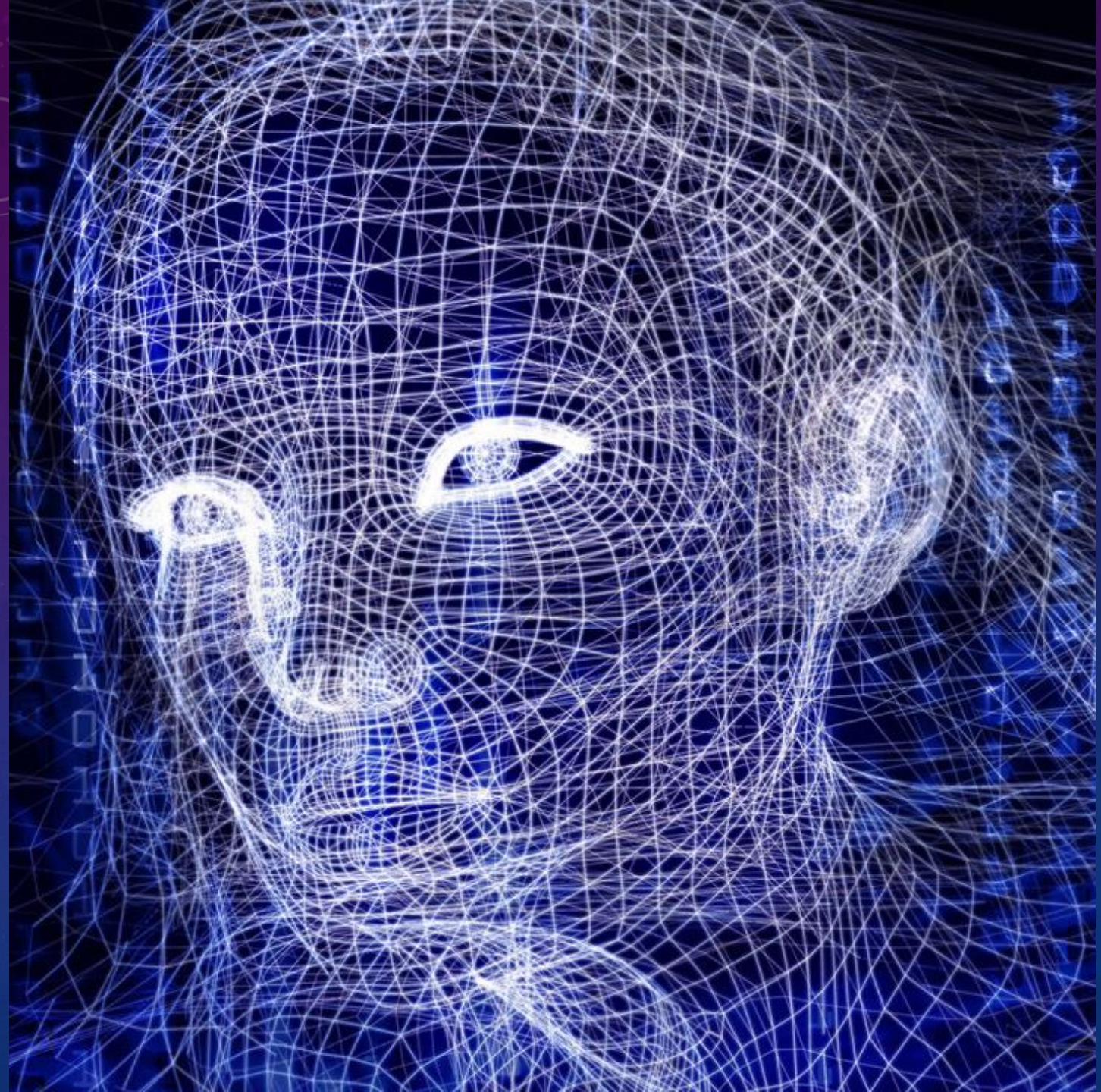
DIGITAL SURVEILLANCE SYSTEMS AND APPLICATION

EXERCISE REVIEW



徐繼聖

Gee-Sern Jison Hsu

National Taiwan University of Science and Technology

Content Overview

Chapter1

- Linear Regression
- Build A Neural Network

Chapter2

- Convolution Filter
- Pooling Practice
- Feature Map Visualization
- Training A Classifier

Chapter3

- Weight Initialization
- Comparison of Different Channel on Feature Map
- Comparison of Deep Networks
- Comparison of Latent Vector

Chapter4

- Faster RCNN
- License Plate Detector

Chapter1

Linear Regression

Define Model Structure

- Set up the base structure of this model in pytorch

```
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
plt.ion()

# Hyper-parameters
num_epochs = 50
learning_rate = 0.001
# Set initialize parameter y = ax + b
a = -0.5
b = 1
```

```
# Toy dataset
x_train = np.array([[3.3], [4.4], [5.5]], dtype=np.float32)

y_train = np.array([[1.7], [2.76], [2.09]], dtype=np.float32)
```

Define Model Structure

- Initialize the model type and declare the forward pass

```
# Define Linear regression model
model = nn.Linear(1, 1)
# Initialize parameter
model.weight.data.fill_(a)
model.bias.data.fill_(b)
```

Loss Function (Criterion) and Optimizer

Use the Mean Square Error (MSE), which is the most commonly used regression loss function

```
# Define Loss
criterion = nn.MSELoss()
```

Use Stochastic Gradient Descent (SGD) optimizer for the update of hyperparameters

```
# Define optimizer
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

Model Training

```
for epoch in range(num_epochs):
    # Convert numpy arrays to torch tensors
    inputs = torch.from_numpy(x_train)
    targets = torch.from_numpy(y_train)
    # Forward pass
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    # Get the model parameters (slope, bias)
    [a, b] = model.parameters()
    print ('Epoch [{}/{}], Loss: {:.4f}, New_a:{:.2f}, New_b:{:.2f}, y_predict:{}, a_gradient:{},
b_gradient:{}'.format(epoch+1, num_epochs, loss.item(),a.data[0][0],
b.data[0],outputs.data[0],model.weight.grad,model.bias.grad))
    # 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.

Visualize Linear Regression

- Download linear_regression_visualize.py
- Set initial linear function (In sample code: a = 0.5, b = 1)

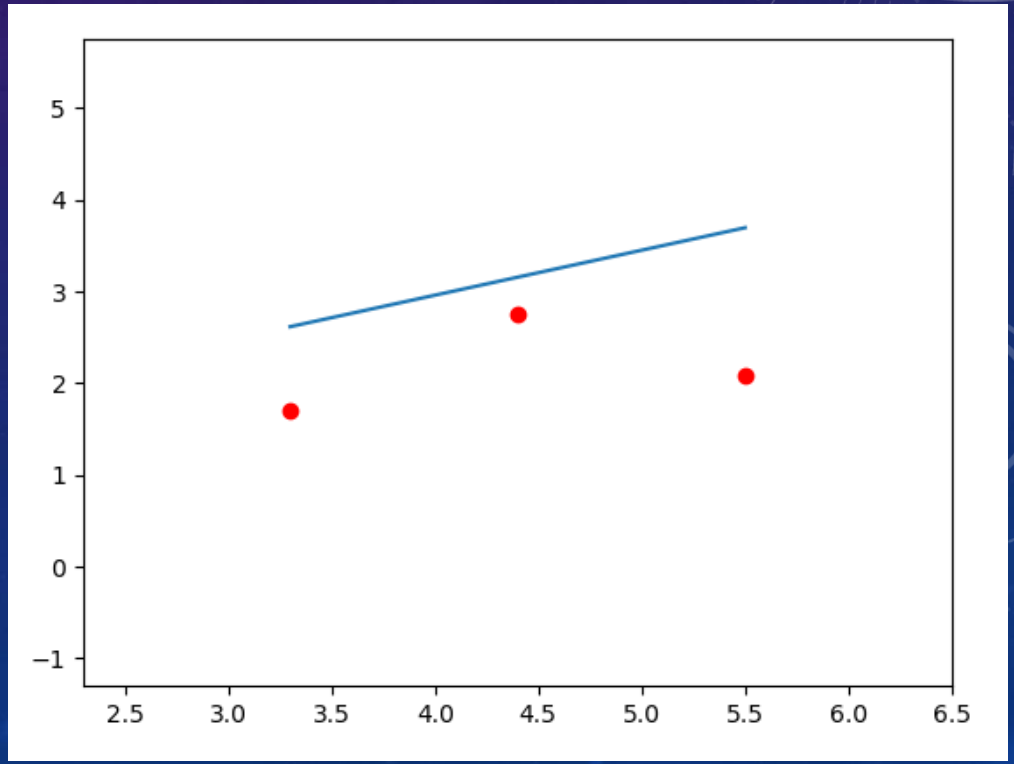
$$y = ax + b$$

- Execute the code, we can see:

Red dot : training set

Blue line : linear function

```
# Hyper-parameters
num_epochs = 1
learning_rate = 0.001
# Set initialize parameter y = ax + b
a = 0.5
b = 1
```



Visualize Linear Regression

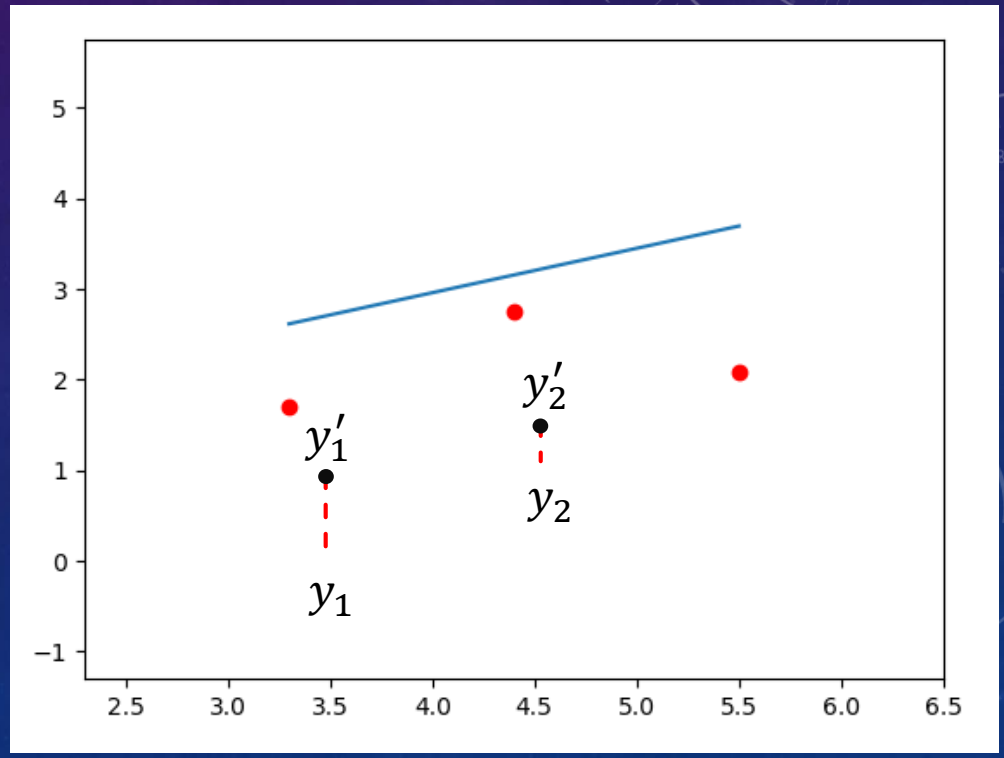
- Loss function: Mean Square Error (MSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2$$

- Code in PyTorch:

```
criterion = torch.nn.MSELoss(reduction='mean')
```

```
Initial Function: y = 0.50x + 1.00
Epoch [1/50], Loss: 1.2839, New_a:0.50, New_b:1.00
Epoch [2/50], Loss: 1.1921, New_a:0.49, New_b:1.00
Epoch [3/50], Loss: 1.1079, New_a:0.48, New_b:1.00
Epoch [4/50], Loss: 1.0307, New_a:0.47, New_b:0.99
Epoch [5/50], Loss: 0.9599, New_a:0.46, New_b:0.99
Epoch [6/50], Loss: 0.8949, New_a:0.46, New_b:0.99
```



How the parameter be updated will talk at later lecture

Exercise 1-1 : Linear Regression

- Please download 1-1_linear_regression.py and adjust the following parameters:

  Step 1 : Input Data

  x_train= [3.3], [4.4], [5.5],[8.2],[9.4]

  y_train= [1.7], [2.76], [2.09],[5.48],[4.99]

  Step 2 : Epoch

  Step 3 : Learning Rate

- Please upload your code and your comments in MS Word to Clouds.

Solution

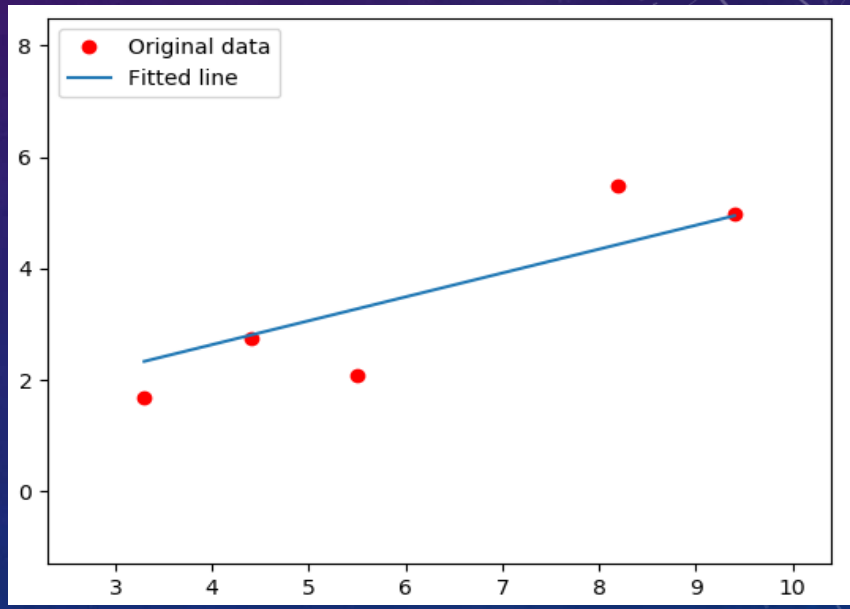
Adjust the input data

```
# Toy dataset
x_train = np.array([[3.3], [4.4], [5.5], [8.2], [9.4]], dtype=np.float32)
y_train = np.array([[1.7], [2.76], [2.09], [5.48], [4.99]], dtype=np.float32)
```

Adjust the lr and epoch

```
# %load 1-1_linear_regression.py
%matplotlib

import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
plt.ion() ##Display dynamic images or multiple windows: using plt.ion()

# Hyper-parameters
num_epochs = 100
learning_rate = 0.002

# Set initialize parameter y = ax + b
a = 0.5
b = 1
```

Result:



PyTorch: Neural Network — Feedforward / MLP

Andrea Eunbee Jang

https://medium.com/biaslyai/pytorch-introduction-to-neural-network-feedforward-neural-network-model-e7231cff47cb

Autograd

- Autograd, which is an automatic differentiation package provided by PyTorch.
- The gradient can only be calculated if your variable is a function of the variable you want to differentiate.

```
import torch

x = torch.ones(1, requires_grad=True)

print(x.grad)    # returns None
```

Autograd

- Example:

```
import torch

x = torch.ones(1, requires_grad=True)
y = x + 2
z = y * y * 2

z.backward()     # automatically
calculates the gradient
print(x.grad)    # ∂z/∂x = 12
```

$$\frac{\partial z}{\partial x} = \frac{\partial 2y^2}{\partial x}$$

$$= 4y \frac{\partial y}{\partial x}$$

$$= 4(x+2) \frac{\partial (x+2)}{\partial x}$$

$$= 4(x+2)$$

$$\rightarrow x = 1$$

$$\rightarrow \frac{\partial z}{\partial x} = 12$$

- .backward() : performs the backpropagation automatically

Perceptron Model

Example:

```
class Perceptron(torch.nn.Module):
    def __init__(self):
        super(Perceptron, self).__init__()
        self.fc = nn.Linear(1,1)
        self.relu = torch.nn.ReLU()
    def forward(self, x):
        output = self.fc(x)
        output = self.relu(x)
        return output
```

self.fc : outputs linear information

self.relu : makes it non-linear

Feedforward Neural Network

```
class Feedforward(torch.nn.Module):
    def __init__(self):
        super(Feedforward, self).__init__()
        self.input_size = 2
        self.hidden_size  = 2
        self.output_size  = 1
        self.fc1 = torch.nn.Linear(self.input_size, self.hidden_size)
        self.relu = torch.nn.ReLU()
        self.fc2 = torch.nn.Linear(self.hidden_size, self.output_size)
        self.sigmoid = torch.nn.Sigmoid()
    def forward(self, x):
        hidden = self.fc1(x)
        relu = self.relu(hidden)
        output = self.fc2(relu)
        output = self.sigmoid(output)
        return output
model=Feedforward()
print(model)
```

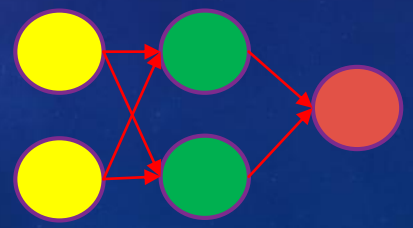
Parameters

Relu function

Sigmoid function

When you have more than two hidden layers, the model is also called the deep/multilayer feedforward model or MultiLayer Perceptron (MLP).

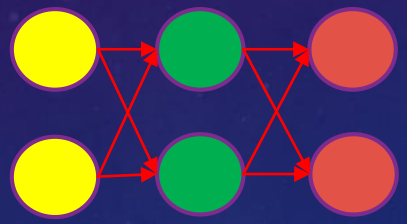
```
Feedforward(
  (fc1): Linear(in_features=2, out_features=2, bias=True)
  (relu): ReLU()
  (fc2): Linear(in_features=2, out_features=1, bias=True)
  (sigmoid): Sigmoid()
)
```



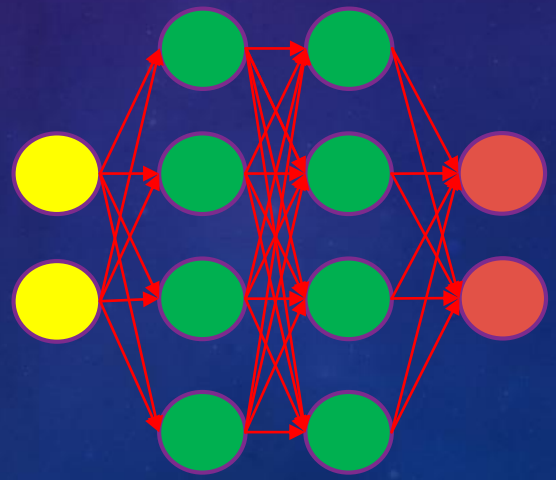
Exercise 1-2 – Build A Neural Network

- Please download 1-2_log_regression_visualize.py which contains a network “Original”.
- Please modify the settings in the file so that it can be structured as the “Modified”.
- See the configurations below for both networks.
- You also need to add the activation function “ReLU” behind each hidden layers
- Please upload your python code in MS Word to Clouds.

Original



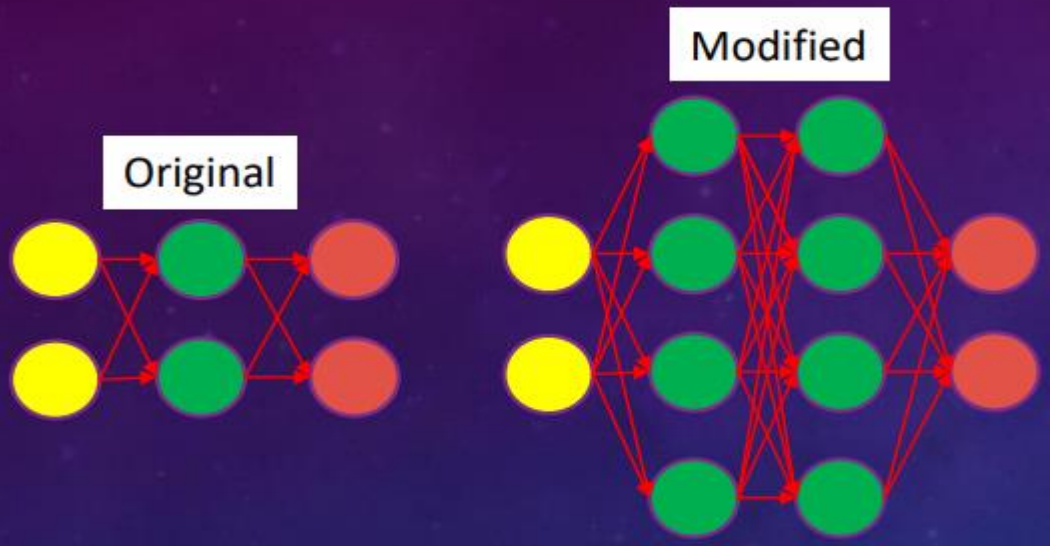
Modified



Yellow : Input layer
Green: Hidden layer
Orange:Output layer

More detail information in exercise ppt page 10~15

Solution



Yellow: Input layer | **Green**: Hidden layer | **Orange**: Output layer

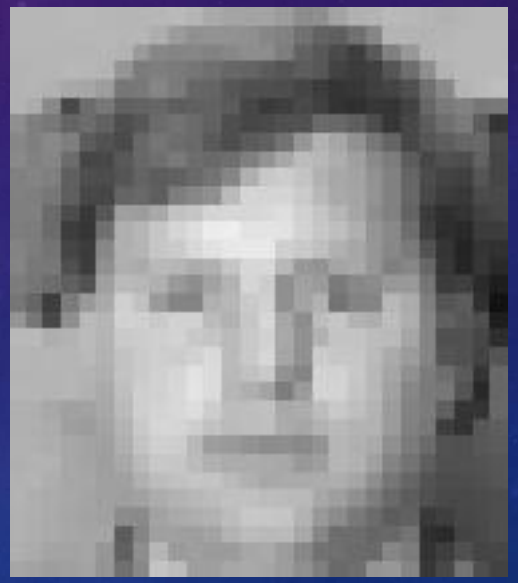
```
class Feedforward(torch.nn.Module):
    def __init__(self):
        super(Feedforward, self).__init__()
        self.input_size = 2
        self.hidden_size  = 4
        self.output_size  = 2
        self.fc1 = torch.nn.Linear(self.input_size, self.hidden_size)
        self.relu = torch.nn.ReLU()
        self.fc2 = torch.nn.Linear(self.hidden_size, self.hidden_size)
        self.fc3 = torch.nn.Linear(self.hidden_size, self.output_size)


    def forward(self, x):
        hidden = self.fc1(x)
        relu = self.relu(hidden)
        output1 = self.fc2(relu)
        relu2 = self.relu(output1)
        output2 = self.fc3(relu2)
        relu3 = self.relu(output2)
        return relu3
```

Chapter2

Example 2-1: Convolution Filter

- Given a face image I_{mg}, please generate F_{conv}, a 5x5 convolution filter with random coefficients in uniform distribution, and use "same zeros padding" with unit stride. Please compute the output by convolving I_{mg} with F_{conv} and calculate the information loss between original image and the one after convolving.

```
KERNAL_SIZE = 3
STRIDE = 1
PADDING =  (KERNAL_SIZE - STRIDE)/2
PADDING = int(PADDING)
img = cv2.imread('./006_01_01_051_08.png')
img = cv2.resize(img,(28,32))

img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
Conv_Filter =
np.random.rand(KERNAL_SIZE,KERNAL_SIZE)
#Conv_Filter =
np.random.normal(mean,std,(KERNAL_SIZE,KERNAL_SIZE))
#Normal distribution
Conv_Filter = Conv_Filter/np.sum(Conv_Filter)
img_F = img
```

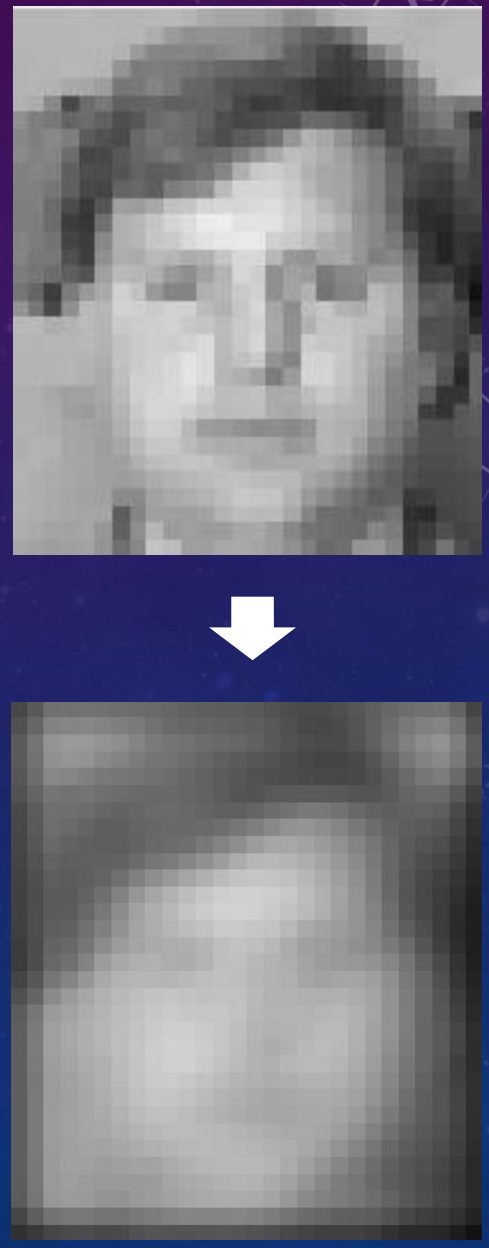


Input Image

Conv kernel

Example 2-1 ：Convolution Filter

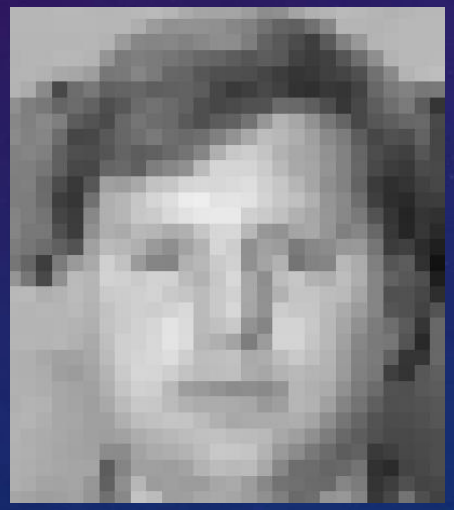
```
for h in range(int((H-KERNAL_SIZE)/STRIDE)+1):
    for w in range(int((W-KERNAL_SIZE)/STRIDE)+1):
        aa = img_F[h*STRIDE:h*STRIDE + (KERNAL_SIZE), w*STRIDE:w*STRIDE +
(KERNAL_SIZE)]*Conv_Filter

        new_feature[h,w] = np.sum(aa)

        img_S = img_F.astype(np.uint8)
        img_new = new_feature.astype(np.uint8)

        cv2.rectangle(img_S, (int(w*STRIDE), int(h*STRIDE)), (int((w*STRIDE
+ KERNAL_SIZE)), int((h*STRIDE + KERNAL_SIZE))), (255, 0, 0), 1)

        cv2.namedWindow('Conv_process', cv2.WINDOW_NORMAL)
        cv2.resizeWindow("Conv_process", 300, 300)
        cv2.imshow('Conv_process',img_S)

        cv2.namedWindow('Conv_result', cv2.WINDOW_NORMAL)
        cv2.resizeWindow("Conv_result", 300, 300)
        cv2.imshow('Conv_result',img_new)

        cv2.waitKey(100)
```



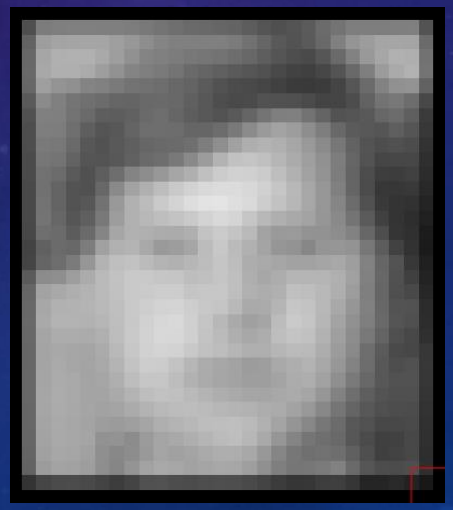
Example 2-1 : Convolution Filter

```
%% Percentage of information loss
imgd = reshape(double(img), numel(img),1);
convimgd = reshape(double(Conv_Img), numel(Conv_Img),1);

Diff = sum(sum(abs(double(imgd)./norm(double(imgd)) - double(convimgd)./norm(double(convimgd)))));
PercnetageDiff = Diff/sum(sum(double(imgd)./norm(double(imgd)))*100);

fprintf('The information loss using %dx%d convolution kernel is %.6f%%\n\n', KERNAL_SIZE, KERNAL_SIZE, PercnetageDiff);
```

Original Image



Convolution Processing



Feature Map

Exercise 2-1 – Design Convolution Filter

Please download 2-1_Convolution_Example.py and design your own convolution filter with a

(1) 3x3 convolution filter

(2) 5x5 convolution filter

(3) 7x7 convolution filter

with random coefficients in normal distribution (std=1, mean=5), and compute the output by convolving I_{mg} with F_{conv}. Then, calculate the information loss with original image. Please write down result and your code in MS Word to Moodle

Change it in to normal distribution

```
KERNAL_SIZE = 3
STRIDE = 1
PADDING =  (KERNAL_SIZE - STRIDE)/2
PADDING = int(PADDING)
img = cv2.imread('./006_01_01_051_08.png')
img = cv2.resize(img,(28,32))

img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
Conv_Filter = np.random.rand(KERNAL_SIZE,KERNAL_SIZE)
#Conv_Filter = np.random.normal(mean,std,(KERNAL_SIZE,KERNAL_SIZE)) #Normal distribution
Conv_Filter = Conv_Filter/np.sum(Conv_Filter)
img_F = img
```

Solution

```
KERNAL_SIZE = 3,5,7
STRIDE = 1
PADDING =  (KERNAL_SIZE - STRIDE)/2
PADDING = int(PADDING)
img = cv2.imread('./006_01_01_051_08.png')
img = cv2.resize(img,(28,32))

img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
#Conv_filter = np.random.rand(KERNAL_SIZE,KERNAL_SIZE)
Conv_Filter = np.random.normal(5,1,(KERNAL_SIZE,KERNAL_SIZE))
#print(Conv_filter)
 #Normal distribution
print(Conv_Filter)
Conv_Filter = Conv_Filter/np.sum(Conv_Filter)
print(Conv_Filter)
img_F = img
```

Change the kernel_size

Change it in to normal distribution

Exercise 2-2 – Pooling Practice

Please download 2-2_Convolution_Pooling_Example.py and design your own convolution filter with a 3x3 convolution filter with random coefficients in normal distribution (mean=2, std=0), and compute the output by convolving I_{mg} with F_{conv}. Then, upload the report in MS Word or PDF to Moodle

Change it in to normal distribution

```
KERNAL_SIZE = 3
STRIDE = 1
PADDING =  (KERNAL_SIZE - STRIDE)/2
PADDING = int(PADDING)
img = cv2.imread('./006_01_01_051_08.png')
img = cv2.resize(img,(28,32))

img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
Conv_Filter = np.random.rand(KERNAL_SIZE,KERNAL_SIZE)
#Conv_Filter = np.random.normal(mean,std,(KERNAL_SIZE,KERNAL_SIZE)) #Normal distribution
Conv_Filter = Conv_Filter/np.sum(Conv_Filter)
img_F = img
```

Please save result and your code in MS Word to Moodle

Solution

```
KERNAL_SIZE = 3
STRIDE = 1
PADDING =  (KERNAL_SIZE - STRIDE)/2
PADDING = int(PADDING)
print(PADDING)
img = cv2.imread('./006_01_01_051_08.png')
img = cv2.resize(img,(28,32))
img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
Conv_Filter = np.random.normal(2,0,(KERNAL_SIZE,KERNAL_SIZE))
img_F = img
```

Change the kernel_size

Change it in to normal distribution

Feature Map Visualization

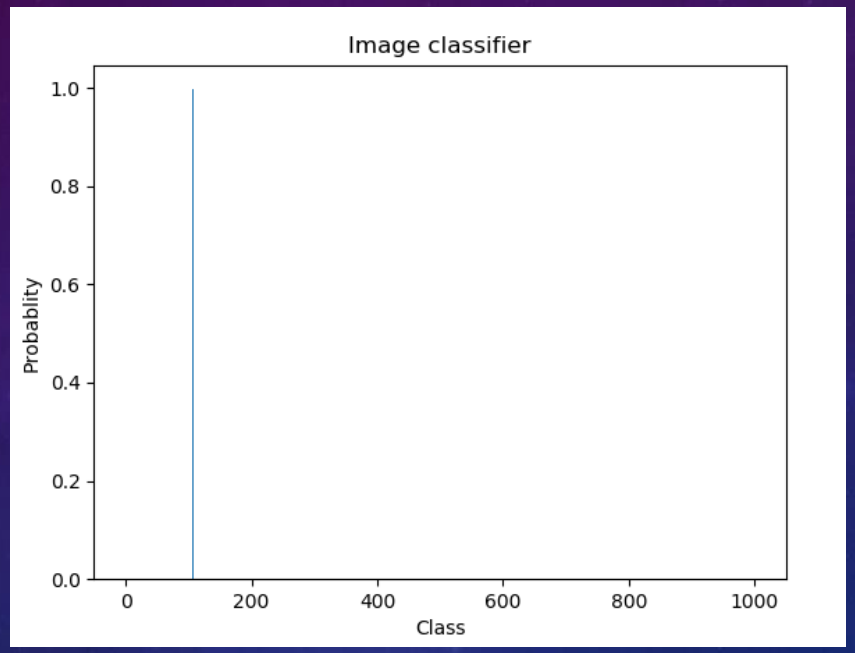
Example 2-3: Feature Map Visualization

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',
```

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

    #Compute the probablity
    s = torch.nn.Softmax(dim=1)
    outputs = s(outputs)
    self.plot_probablity(outputs)
    prob, predicted = outputs.sort(1,descending=True)
    prob = prob.data.numpy()
    predicted = predicted.data.numpy()

    #Output the TOP-3 probability
    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])]))
```

Example 2-3 : Feature Map Visualization




```
Probablity TOP-3:


TOP_1
Probablity:0.9995427131652832
Predicted: 'jellyfish'

TOP_2
Probablity:0.00025874192942865193
Predicted: 'isopod'

TOP_3
Probablity:9.464480535825714e-05
Predicted: 'chambered nautilus
```

Original image: jellyfish.jpg

Probability of the classes

Predicted class : jellyfish

Exercise 2-3: Feature Map Visualization

- Pip install opencv-python in your pytorch environment.
- Please download the “2-3_Feature_map_visualization” on Clouds and 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.

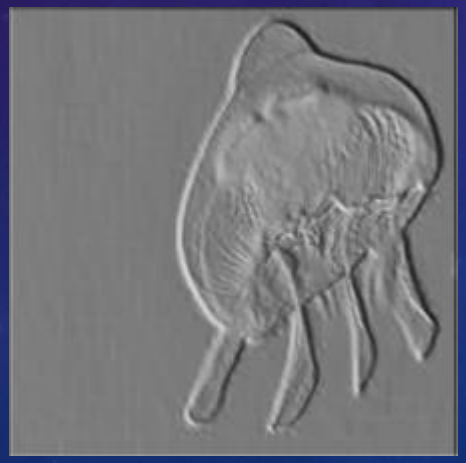
Please write down result and your code in MS Word and upload to Moodle

Solution

```
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()
```

Change the extract layer

Result: Original (layer2)



Modify (layer5)

Exercise 2-4: Feature Map Visualization

- Pip install opencv-python in your pytorch environment.
- Please download the “2-3_Feature_map_visualization” on Clouds and choose your own images from Internet.
- Compare the probability of the images that contain two classes and different variations (pose, occlusion, age).
- Please write down result and your code in MS Word to Moodle




Solution

```
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('./border_collie.jpg',2)
    print(myClass.pretrained_model2)

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

Your image path and layer

```
Probablity TOP-3:


TOP_1
Probablity:0.28741899132728577
Predicted: 'Siberian husky'

TOP_2
Probablity:0.21282021701335907
Predicted: 'Eskimo dog

TOP_3
Probablity:0.09047196805477142
Predicted: 'Border collie'
```

```
Probablity TOP-3:


TOP_1
Probablity:0.483703702688217l6
Predicted: 'Border collie'

TOP_2
Probablity:0.3483608365058899
Predicted: 'collie'

TOP_3
Probablity:0.037174716591835O2
Predicted: 'borzoi
```

Solution

```
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('./border_collie.jpg',2)
    print(myClass.pretrained_model2)

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

Your image path and layer

```
Probablity TOP-3:


TOP_1
Probablity:0.4282551407814026
Predicted: 'Border collie'

TOP_2
Probablity:0.32686054706573486
Predicted: 'EntleBucher'

TOP_3
Probablity:0.12311583012342453
Predicted: 'Appenzeller'
```

```
Probablity TOP-3:


TOP_1
Probablity:0.34360525012016296
Predicted: 'skunk

TOP_2
Probablity:0.15644530951976776
Predicted: 'Border collie'

TOP_3
Probablity:0.10190677642822266
Predicted: 'papillon'
```

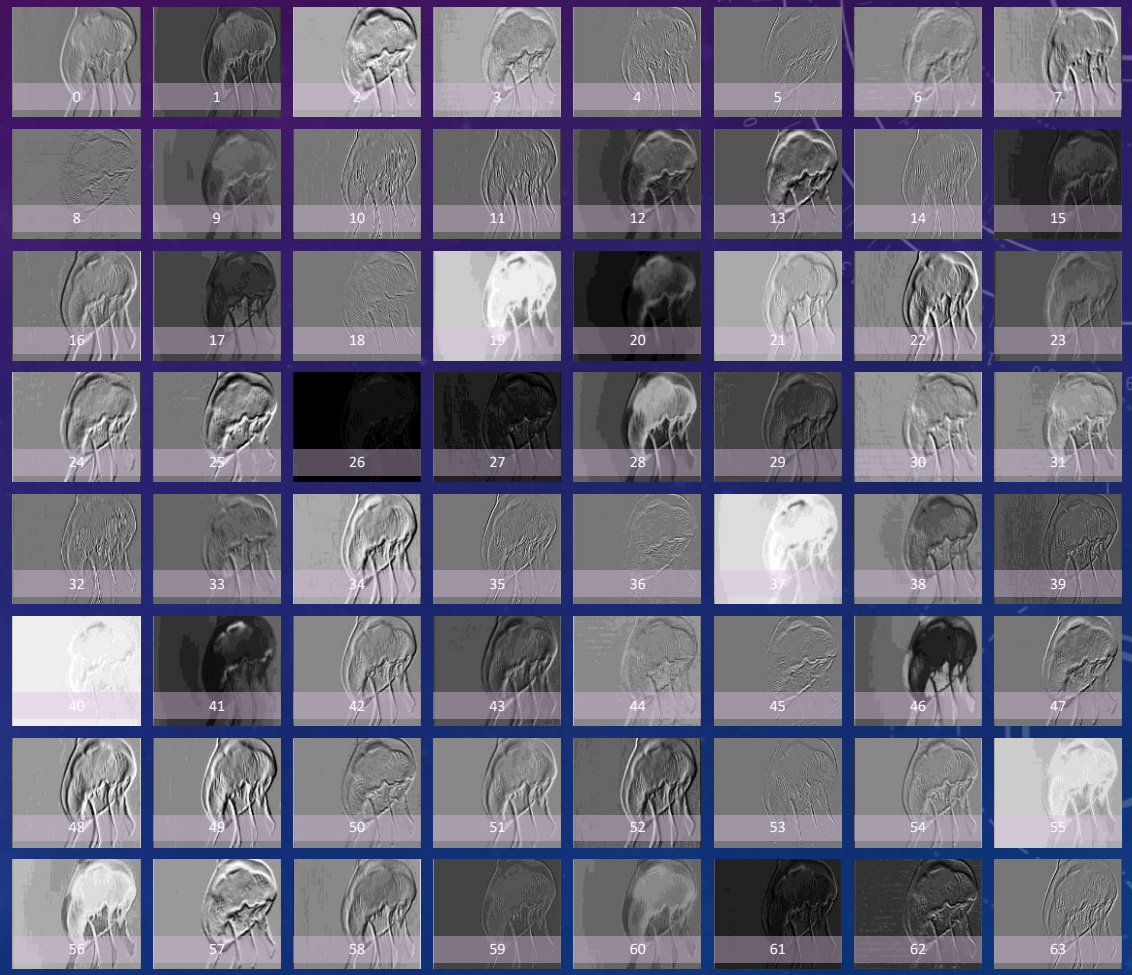
Example 2.5 : Feature Overview

Feature maps from same layer :

```
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',2)
    Compare=FeatureVisualization('./car.jpg',2)
    print(myClass.pretrained_model2)
```

Select layer 2

Choose two pictures to compare



In Folder './feat_2'

Example 2.5 : Feature Overview

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

On different layer how many feature maps we can get

Save the feature maps

Example 2.5 : Feature Overview

```
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 = cos(myClass_vector,Compare_vector)
  print("Verification:")
  if cosine_dist > 0.4:
     print("They are the same!")
     print("Their cosine_similarity:{}".format(cosine_dist))
  else:
     print("They are not the same!")
     print("Their cosine_similarity:{}".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 2.5 : 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:0.0019321587169542909
Predicted: 'nematode
```

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


TOP_1
Probablity:0.2607852518558502
Predicted: 'sports car

TOP_2
Probablity:0.20074793696403503
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 2-5: Feature Overview

- Pip install opencv-python in your pytorch environment.
- Please download the “2-5_Feature_Overview” on Clouds and choose your own images from Internet.
- Step 1: Compare different objects and the same objects by choosing images from Internet.
- Step 2: Compare the results of different crop sizes of the same image(e.g. Dog image only crop the tail)
- Upload your observations, comments and your code to Moodle in a docx.

Solution

Step1: Different objects

```
# Define image path and select the layer
myClass=FeatureVisualization('./jellyfish.jpg',2)
Compare=FeatureVisualization('./car.jpg',2)
print(myClass.pretrained_model2)
```

```
Verification:
They are not the same!
Their cosine_similarity:tensor([-0.0497], grad_fn=<DivBackward0>)
Their euclidean_dist:139.6898193359375
```

Same objects

```
# Define image path and select the layer
myClass=FeatureVisualization('./jellyfish.jpg',2)
Compare=FeatureVisualization('./jellyfish2.jpg',2)
print(myClass.pretrained_model2)
```

```
Verification:
They are the same!
Their cosine_similarity:tensor([0.6344], grad_fn=<DivBackward0>)
Their euclidean_dist:92.77845764160156
```

Image Classification

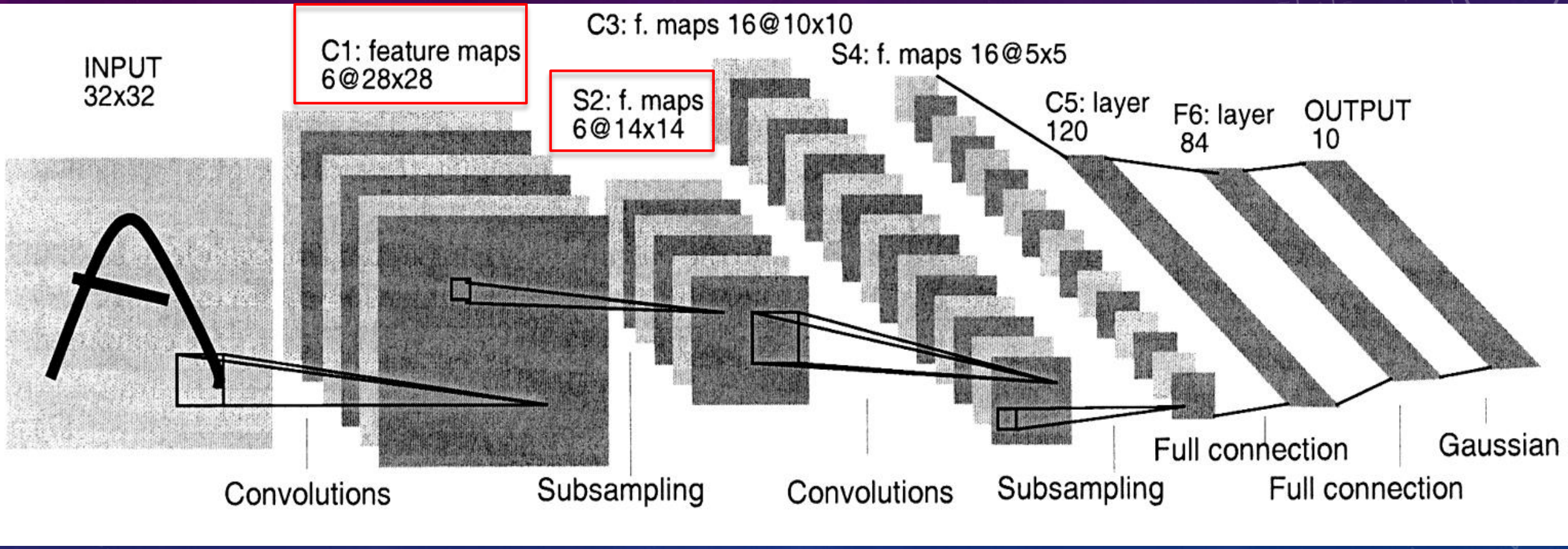
CS231n: Convolutional Neural Networks for Visual Recognition

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

Le Net

- Yann LeCun and his collaborators developed a really good recognizer for handwritten digits by using backpropagation in a feedforward net with:
  - Many hidden layers
  - Many maps of replicated units in each layer.
  - Pooling of the outputs of nearby replicated units.
  - A wide net that can cope with several characters at once even if they overlap.
  - A clever way of training a complete system, not just a recognizer.
- This net was used for reading ~10% of the checks in North America.
- Look the impressive demos of LENET at http://yann.lecun.com

The architecture of LeNet5



Example 2.6 - 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 2.6 - Training A Classifier

2. Define a Convolutional Neural Network(LeNet)

Input size : 3 * 32 * 32

```
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(5* 5* 16, 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, 5* 5* 16)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Input channel

Output channel

Kernel size

Define forward path

Example 2.6 - 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 2.6 - 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

Example 2.6 - 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 : 0.86498116250336l7
Epoch : 10 steps : 10000 Training Loss : 0.8769527244269848
Epoch : 10 steps : 12000 Training Loss : 0.8830881424993277
Finished Training
Accuracy : 61 %
```

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

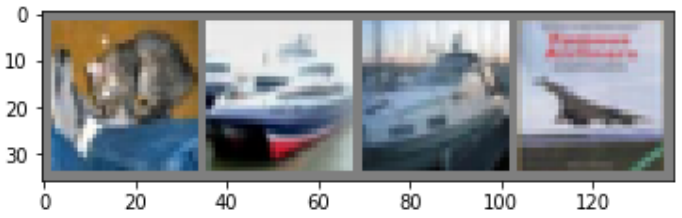
Exercise 2-6 – Build A Classifier

- Please install matplotlib with the command “pip install matplotlib”
- Download 2-6_CIFAR10.py and do the following steps:
- Step 1: Epoch
- Step 2: Optimizer learning rate
- Please upload your result in MS Word to Moodle

Solution

Original

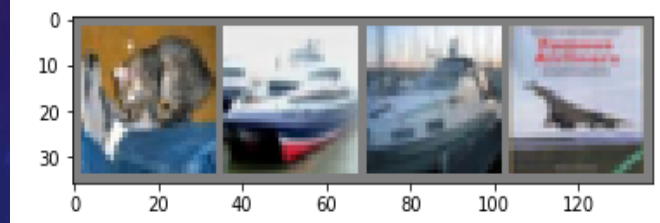
Epoch:5 Learning rate:0.001



```
GroundTruth:    cat  ship  ship plane
Predicted:    cat   car   car  ship
Accuracy : 60 %
```

Modify

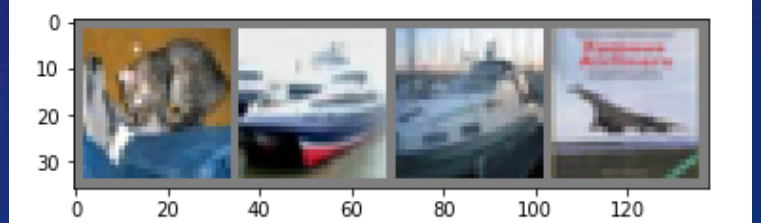
Epoch:10 Learning rate:0.001



```
GroundTruth:    cat  ship  ship plane
Predicted:    cat   car plane  ship
Accuracy : 63 %
```

Modify

Epoch:5 Learning rate:0.002



```
GroundTruth:    cat  ship  ship plane
Predicted:    cat  ship plane  ship
Accuracy : 59 %
```

Exercise 2-7

- Download and modify 2-6_CIFAR10.py to train MNIST dataset with following steps
  1. Download the MNIST Dataset from the following website
     Hint: https://pytorch.org/docs/stable/torchvision/datasets.html
  2. Since MNIST is a handwritten digits dataset, please change the name of classes.
  3. Since the size of images in MNIST dataset are (1,28,28), please check the following points (Input size of first convolution layer and fully-connected layer)
- Please write down the reason of the difference, result and your code in MS Word to Moodle

Solution

```
########################################################################
# The output of torchvision datasets are PILImage images of range [0, 1].
# We transform them to Tensors of normalized range [-1, 1].
if __name__=="__main__":
    def imshow(img):
        img = img / 2 + 0.5  # unnormalize
        npimg = img.numpy()
        plt.imshow(np.transpose(npimg, (1, 2, 0)))
        plt.show()

    transform = transforms.Compose(
        [transforms.Resize(32),transforms.ToTensor(),transforms.Normalize([0.5], [0.5])])
    #torchvision.transforms.Normalize(mean, std, inplace=False)v
    #mean (sequence) - Sequence of means for each channel.
    #std (sequence) - Sequence of standard deviations for each channel.

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

    classes = ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9')
```

Change the MNIST dataset

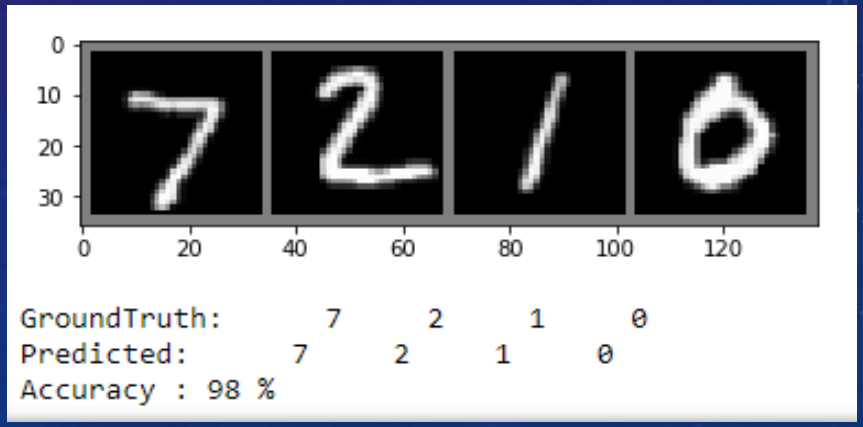
Change the MNIST class

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 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

net = Net()
net
```

Change the input channel

Change the output dimension

Result:



Example 2.8: Testing VGG16 on CIFAR10

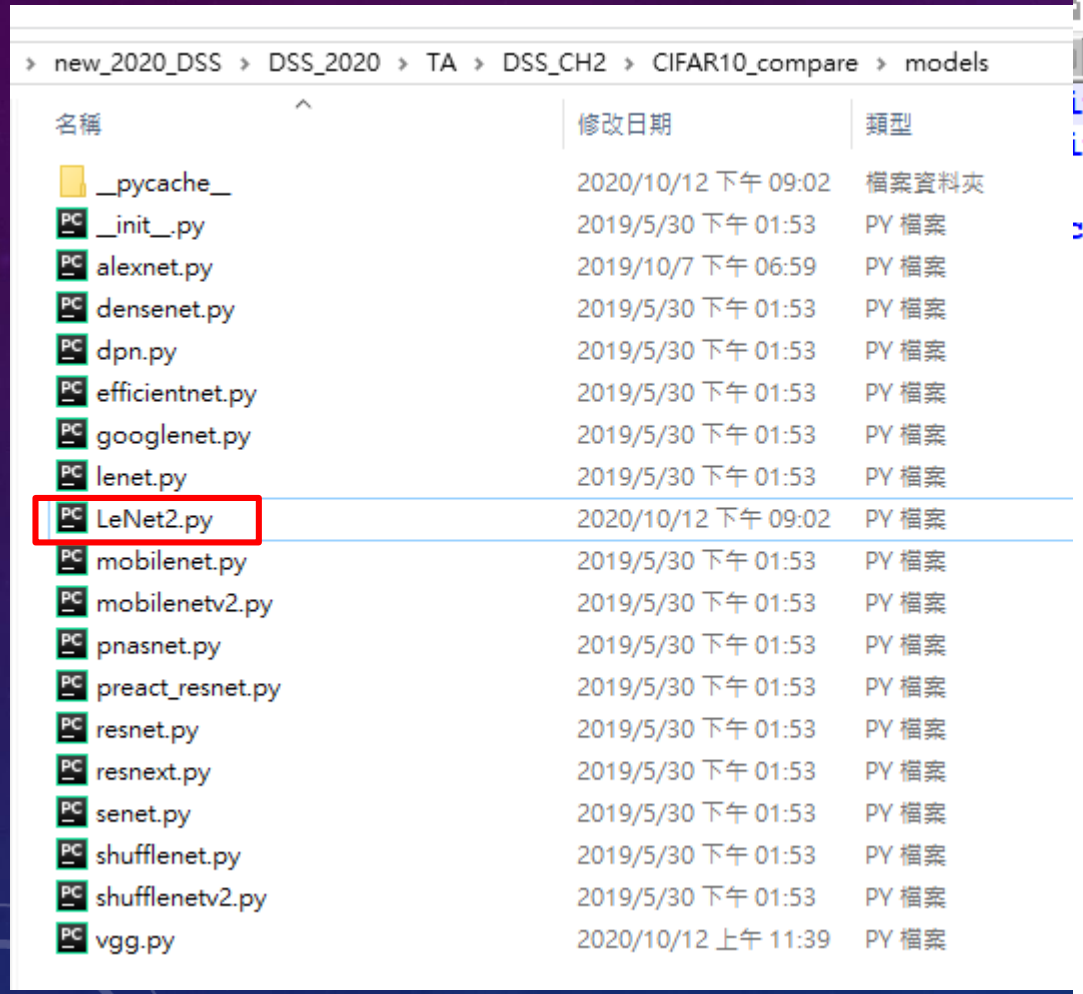
```
from models.LeNet2 import Net
# Load Data
print('==> Preparing data..')
transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),])

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=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()
# your train model net
#net= Net()
```

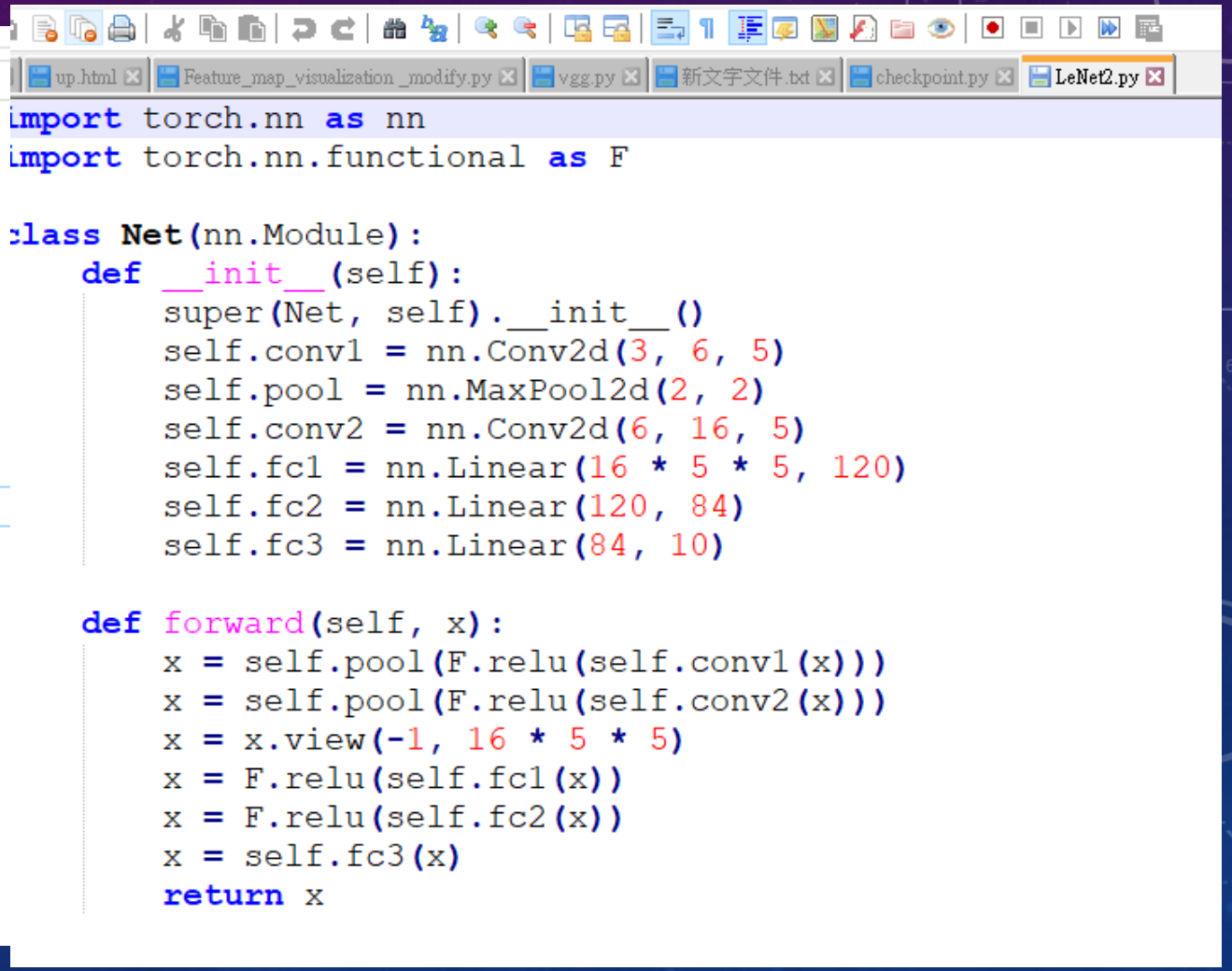
Import Your 2-6_CIFAR10.py training network

Define pretrain network

Your 2-6_CIFAR10.py training network

Example 2.8: Testing VGG16 on CIFAR10






```python
import torch.nn as nn
import torch.nn.functional as F

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
```

Put your training network on models folder

Example 2.8: Testing VGG16 on CIFAR10

```
# 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 your training checkpoint.
# checkpoint = torch.load('./checkpoint/test_pretrain3_checkpoint.pth')
# net.module.load_state_dict(checkpoint['net'])
```

Load pretrain model

Load your checkpoint model

Put your 2-6 CIFAR10 checkpoint into checkpoint folder

Example 2.8: Testing VGG16 on CIFAR10

```
criterion = nn.CrossEntropyLoss()
def test():
    global best_acc
    net.eval()
    test_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(testloader):
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = net(inputs)
            loss = criterion(outputs, targets)

            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()
        print('------Start Testing------')
        print('Loss: %.3f | Acc: %.3f%% (%d/%d)'
            % (test_loss/(batch_idx+1), 100.*correct/total, correct, total))
test()
```

Exercise 2-8: Result Compare on CIFAR10

- Please download 2-8 CIFAR10_compare.zip from moodle
- Modify test.py to use ResNet18, GoogleNet and VGG16 for testing CIFAR10 Dataset
- Modify test.py to use 2-6_CIFAR10.py checkpoint for testing CIFAR10 Dataset

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

Solution

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

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

Load your training model

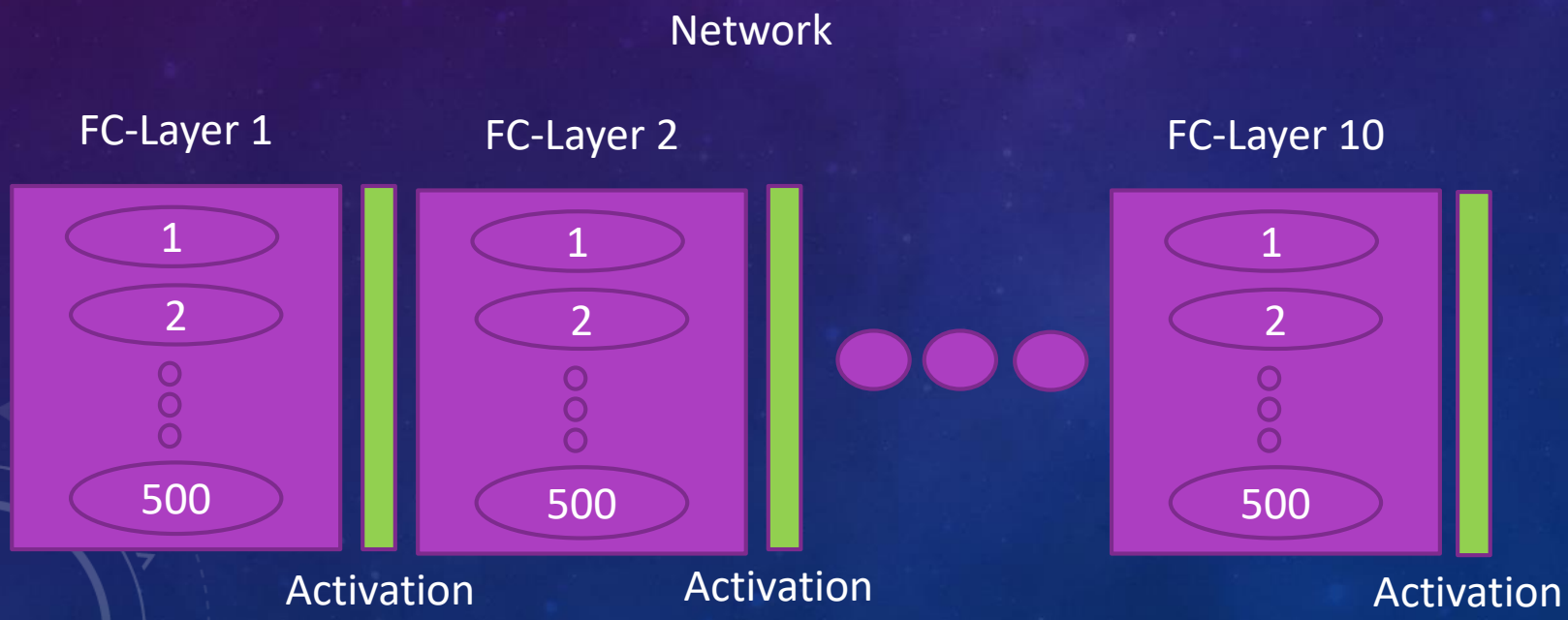
Your training model result

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

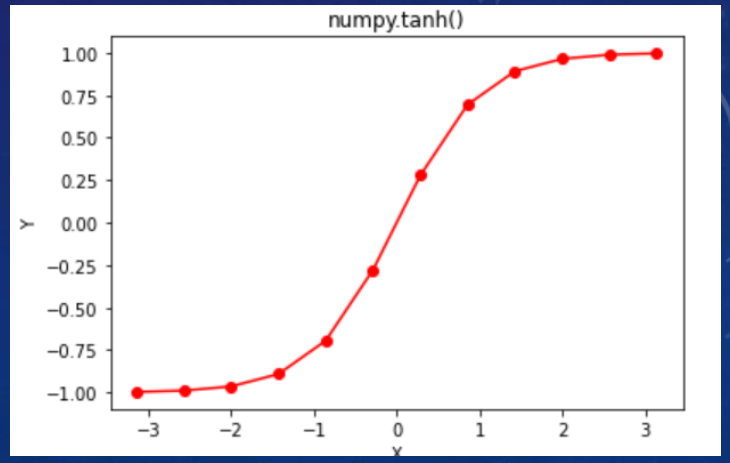
Chapter3

Example 3-1 – Weight Initialization

Example 1 examines a FC-Network with 500 Neurons on each layer and 10 layers in total. We are using tanh non-linearities as our activation functions and can change the initialization of the weights. We will try out a small, medium and large initialization. After initialization, we perform one forward pass. Then, we will look at the output values of the layers with a normally distributed input and estimate the Gradient.



Activation function tanh



Example 3-1 – Code Setup – Part 1

- For this example, we will use only the standard libraries numpy and matplotlib
- D is our input vector, which is created from a gaussian normal distribution
- In line 5, we define our hidden_layer_sizes.
- Line 6 sets up the activation functions. In our example, we only consider relu and tanh

```
1  #assume some unit gaussian 10-D input data
2  import numpy as np
3  import matplotlib.pyplot as plt
4  D = np.random.randn(1000, 500)
5  hidden_layer_sizes =[500]*10
6  nonlinearities =['relu']*len(hidden_layer_sizes)
```

Example 3-1 – Code Setup – Part 2

- For the activation function, we make use of the lambda function and dictionaries, which are both very useful in python.
- The for loop, starting in line 3, is activating our Network Architecture
  - Line 7 initializes the weight
  - Line 8 performs the forward pass
  - Line 9 is feeding the result of line 8 into our activation function
  - Finally we save our result in line 10 to Hs

```
1  act={'relu': lambda x: np.maximum(0,x), 'tanh': lambda x: np.tanh(x)}
2  Hs={}
3  for i in range(len(hidden_layer_sizes)):
4      X=D if i==0 else Hs[i-1] # input at this layer
5      fan_in = X.shape[1]
6      fan_out = hidden_layer_sizes[i]
7      W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2) #layer intialization
8      H=np.dot(X,W) #matrix multiplication
9      H=act[nonlinearities[i]](H)
10     Hs[i]=H
11
```

Lambda function in python

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.
- The expression is executed and the result is returned:
- A very simple example is the following
  - In this example, we define the lambda argument *a*
  - We then define the expression *a+10*
  - Can you guess the result?

```
x = lambda a: a + 10
print(x(5))
```

Help with commands

numpy.random.randn

numpy.random.**randn**(*d0*, *d1*, ..., *dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, **randn** generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.

This is a convenience function. If you want an interface that takes a tuple as the first argument, use numpy.random.standard_normal instead.

Parameters: d0, d1, ..., dn : *int, optional*

The dimensions of the returned array, should be all positive. If no argument is given a single Python float is returned.

Returns: Z : *ndarray or float*

A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

See also:

standard_normal Similar, but takes a tuple as its argument.

Example 3-1 – Code Setup – Part 3

- To examine our results, we make use of the numpy module
- We will look at each layer's mean and standard deviation, with *np.mean* and *np.std*

```
# look at distribution at each layer
print('input layer had mean %f and std %f'.format(np.mean(D), np.std(D)))
layer_means = [np.mean(H) for i, H in Hs.items()]
layer_stds = [np.std(H) for i, H in Hs.items()]
for i,H in Hs.items():
    print('hidden layer {} had mean {} and std {}'.format(i+1, layer_means[1], layer_stds[i]))
```

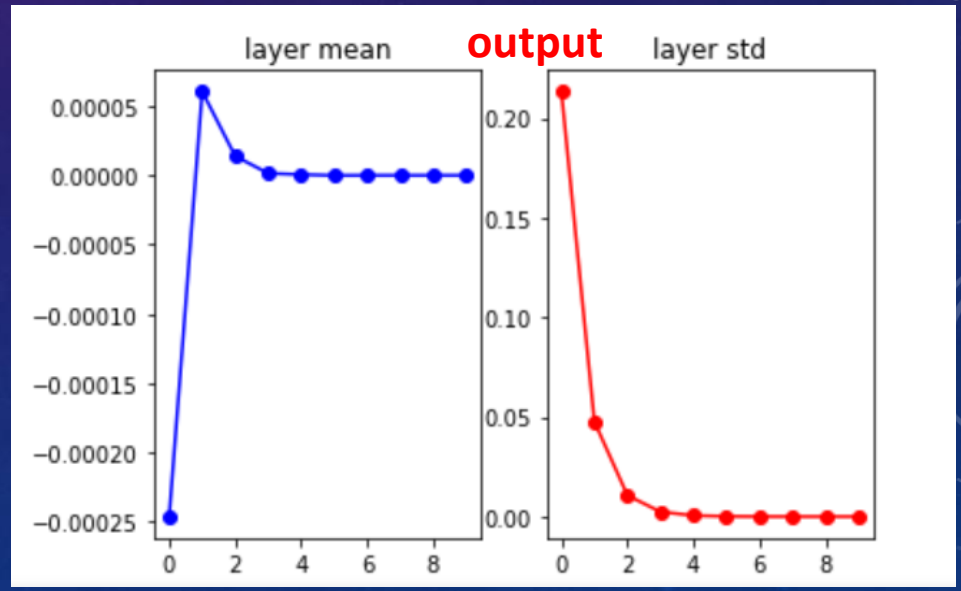
Example 3-1 – Code Setup – Part 4

- To plot our results, we make use of the matplotlib.pylot module
- We will plot the mean and standard deviation *std* from before.

code

```
#plot the means and standard deviations
plt.figure()
plt.subplot(121)
plt.plot(Hs.keys(), layer_means, 'ob-')
plt.title('layer mean')
plt.subplot(122)
plt.plot(Hs.keys(),layer_stds, 'or-')
plt.title('layer std')
```

output



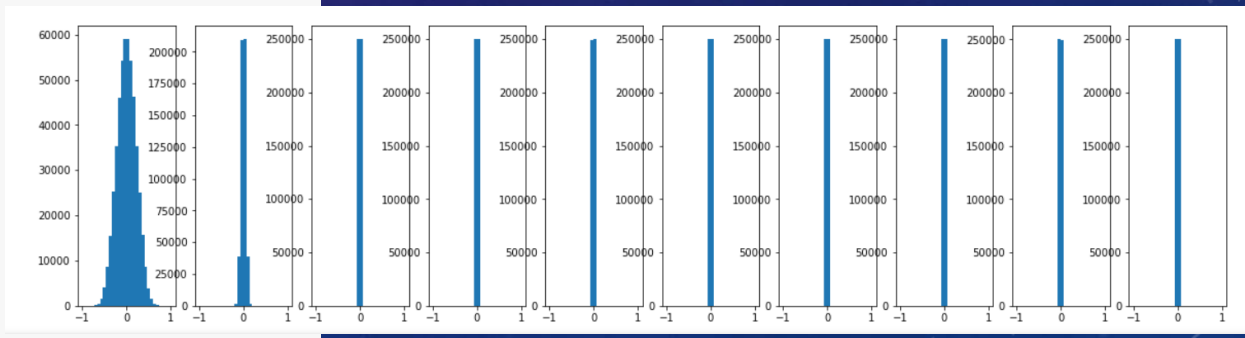
Example 3-1 – Code Setup – Part 5

- In the last part, we will draw a histogram of each layer's output after the activation function with the *plt.hist()* function in line 23. The *ravel()* command in line 23 is giving us a flattened array from the input array.

code

```
17  #plot the raw distributions
18
19  plt.figure(figsize=(20,5))
20  for i, H in Hs.items():
21      plt.subplot(1, len(Hs), i+1)
22      #print(H)
23      plt.hist(H.ravel(), 30, range=(-1,1))
```

output



Let’s have a look at the following examples

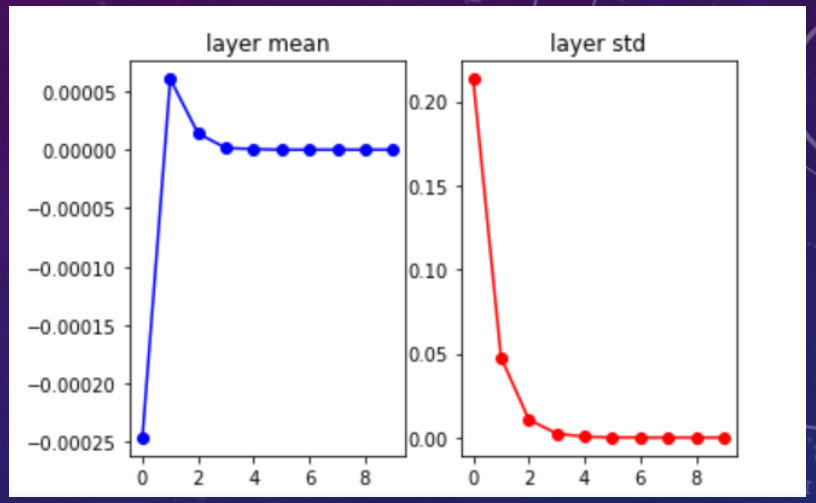
- Different dimensions of the input
  - Small
  - Medium
  - Large
- Normalized input
- How do the two activation functions tanh and ReLU behave?

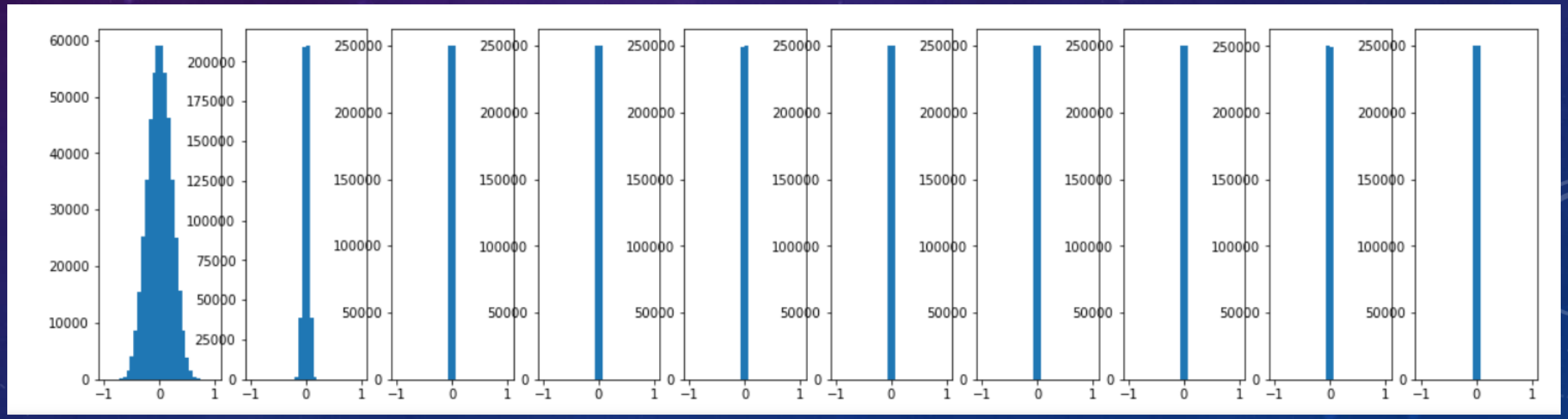
Example 3-1 Different Initialization of Neural Networks from Layer 1 to Layer 10

Small Weight Initialization:

Code Change:

```
W = np.random.randn(fan_in, fan_out)*0.01
```





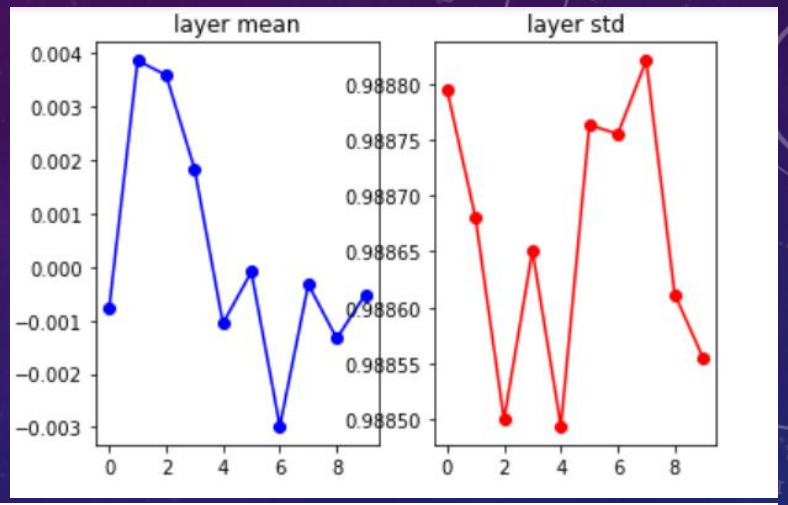
Issue: Activation functions approach 0. Therefore the gradients will be 0. How does the backward pass would look for this scenario?

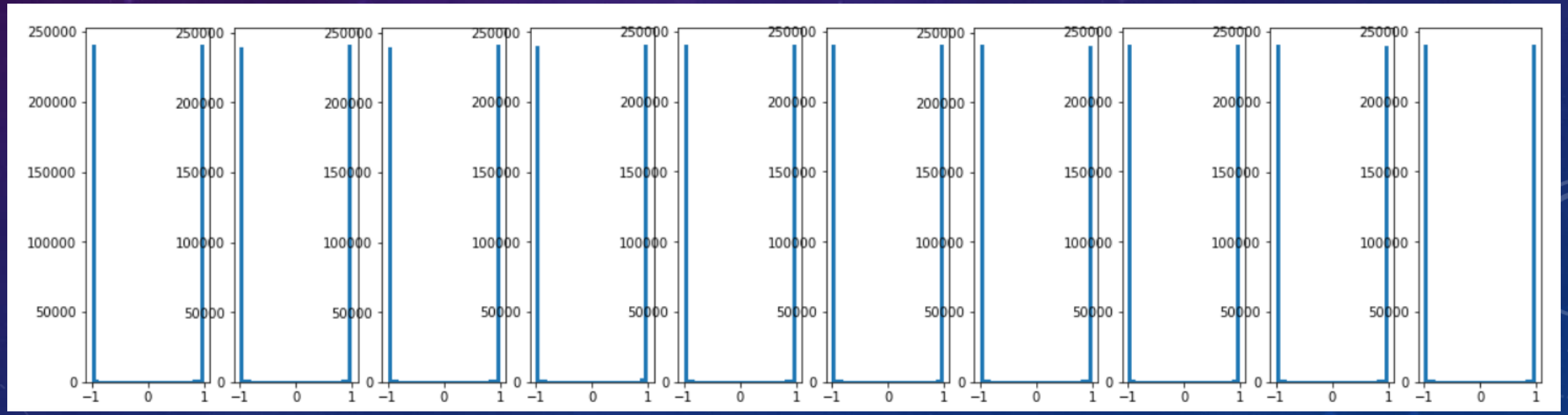
Example 3-1 Different Initialization of Neural Networks from Layer 1 to Layer 10

Large Weight Initialization:

Code Change:

```
W = np.random.randn(fan_in, fan_out)*1.6
```





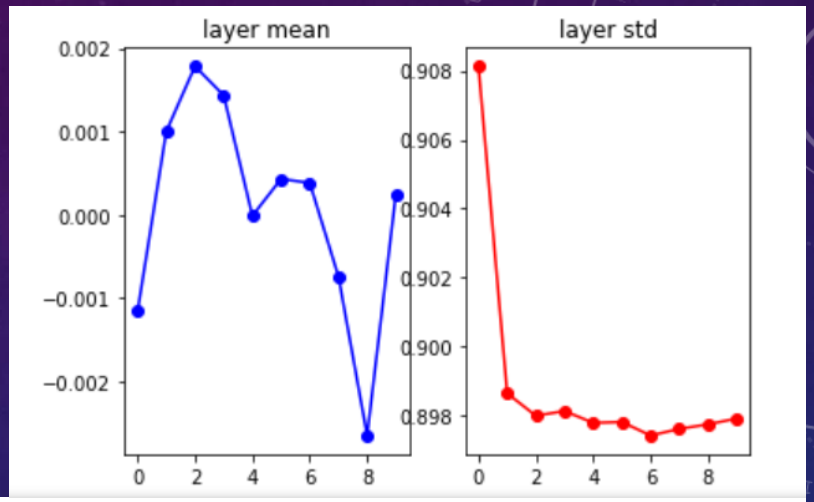
Issue: all neurons are completely saturated, either -1 or 1. Gradients will be zero. Backprop struggles and training is not possible

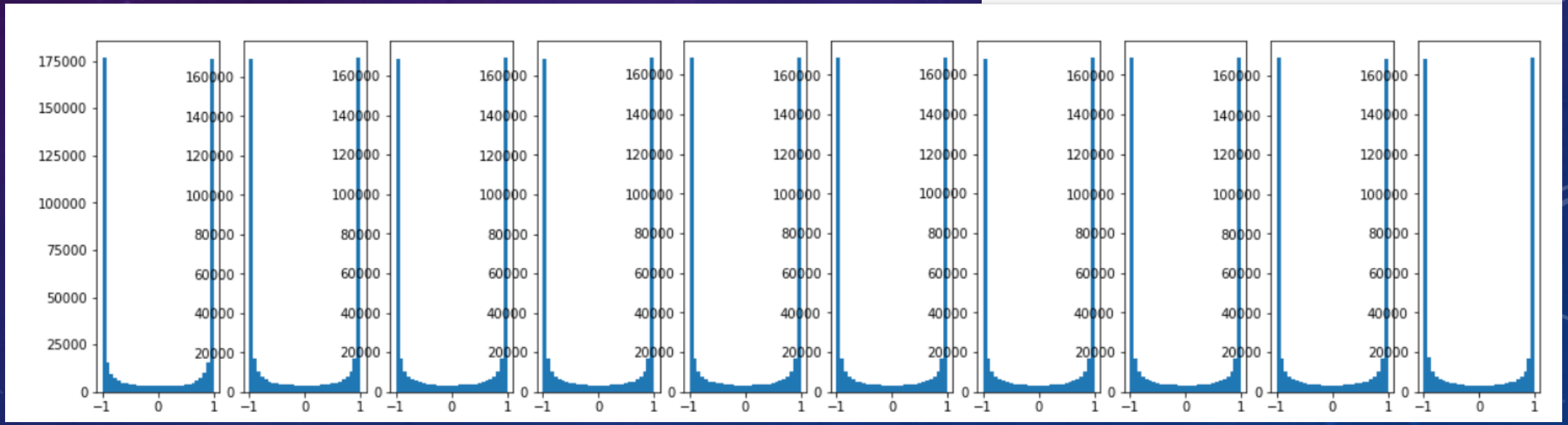
Example 3-1 Different Initialization of Neural Networks from Layer 1 to Layer 10

Code Change:

Medium Weight Initialization:

```
W = np.random.randn(fan_in, fan_out)*0.2
```





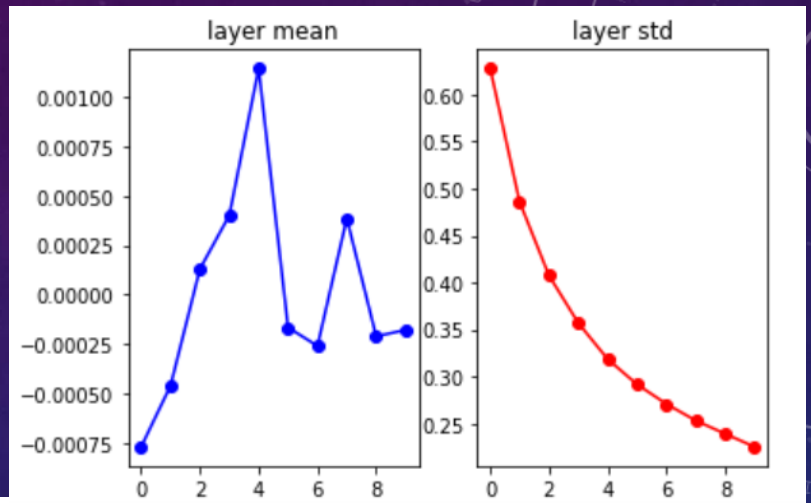
Observation: This looks already more like the result we are requesting in order to perform backprop, since the results are not collapsing in a single value.

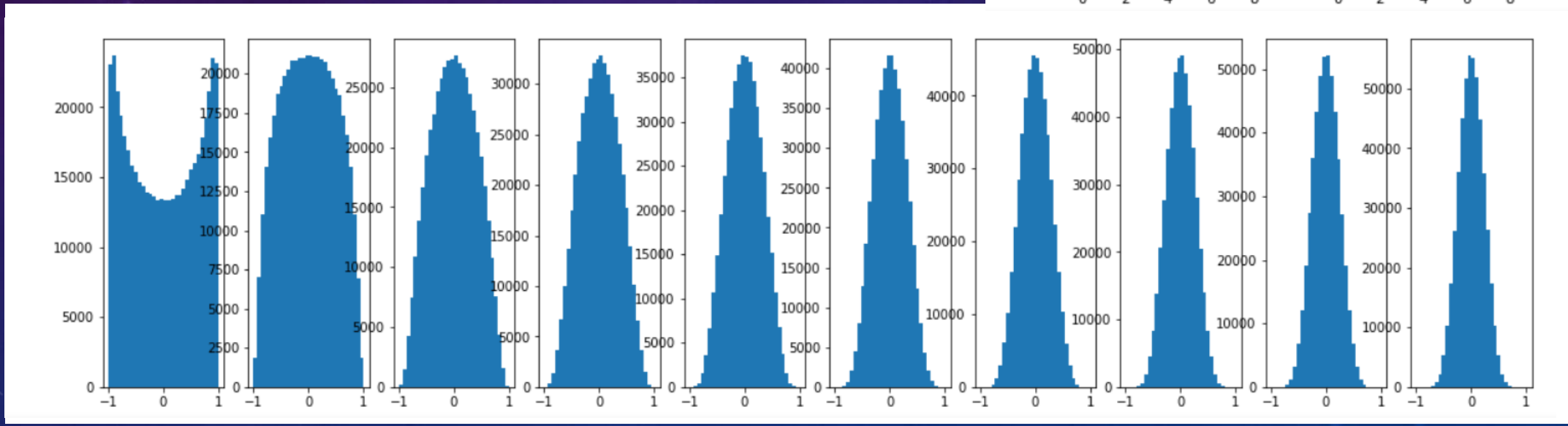
Example 3-1 Different Initialization of Neural Networks from Layer 1 to Layer 10

Now, we will use a kind of normalization.

Code Change:

```
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)
```





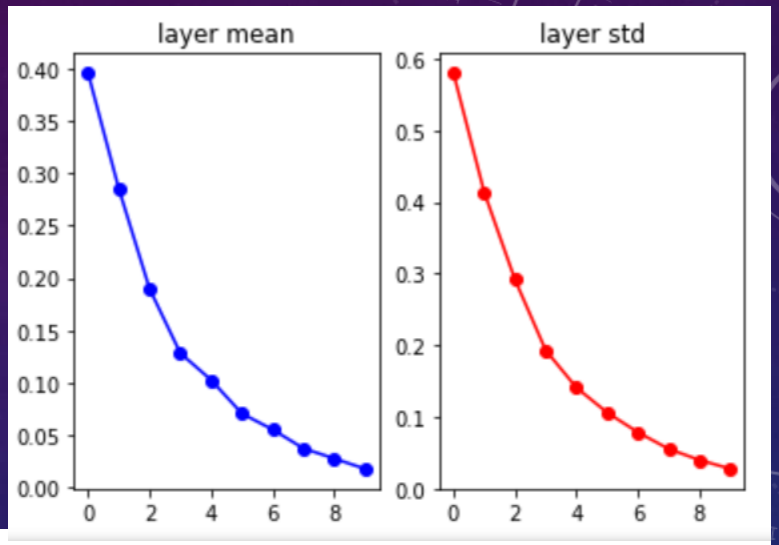
With this result, we receive a reasonable initialization for our Network setup, from where we have the base for a successful training.

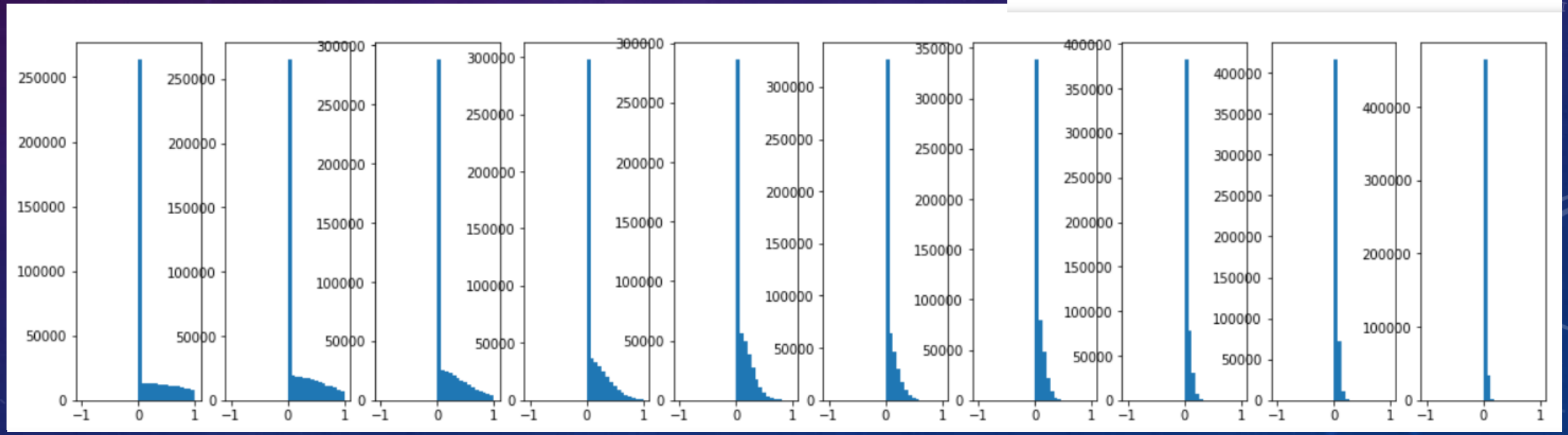
Example 3-1 Change Activation Function to Relu

Now, we will use normalization.

Code Change:

```
nonlinearities =['relu']*len(hidden_layer_sizes)
```





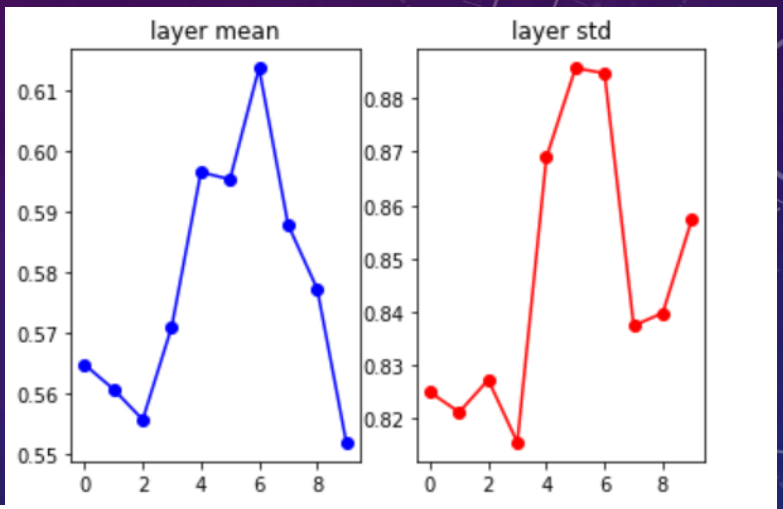
With a different activation function, our Network breaks again and backprop algorithm will fail.

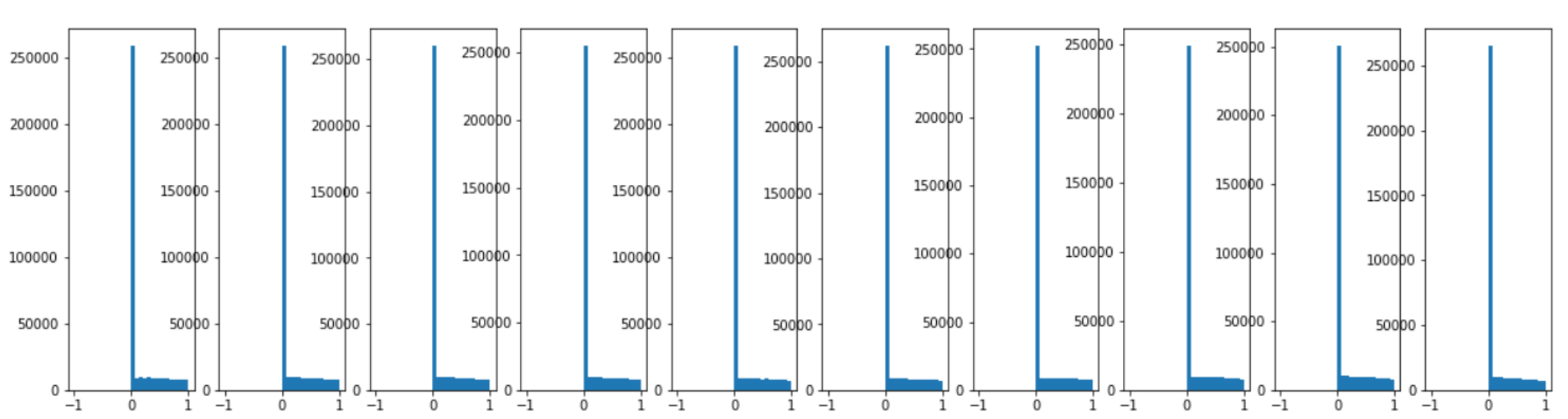
Example 3-1 Add a division by two during Initialization of W

Now, we will use a kind of normalization.

Code Change:

```
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

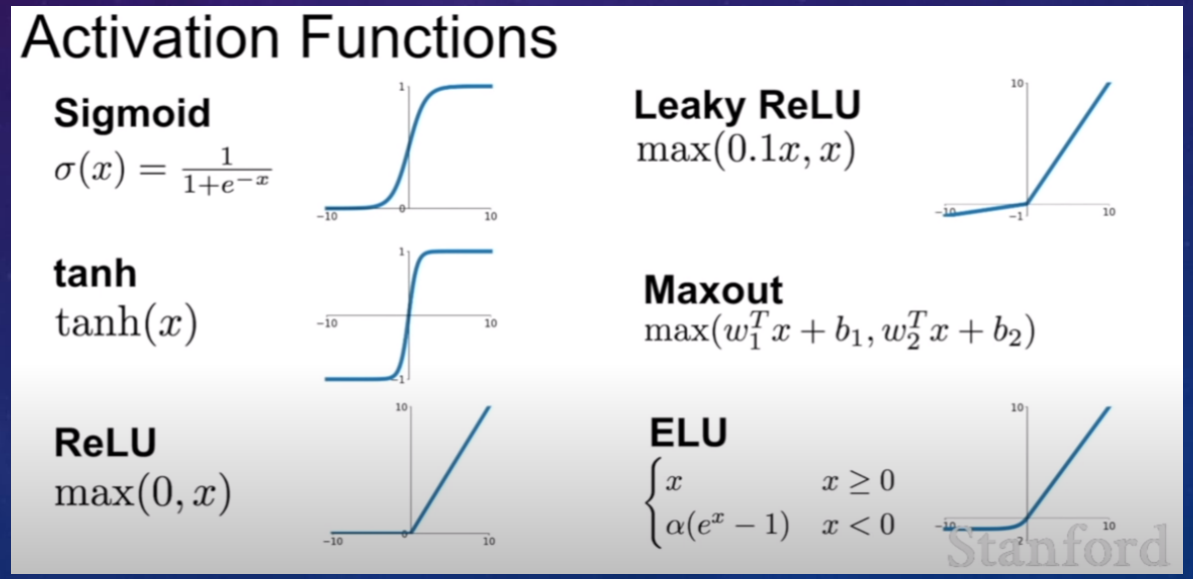




Small changes can have a big impact during initialization. The additional division by 2 enables back prop.

Exercise 3-1 – Weight Initialization and Forward Pass

- We have looked at two activation functions so far: tanh and ReLU. There are far more and it is an interesting area of research. Therefore, investigate the initialization behavior of two more activation functions of your choice. You can take some examples from the Stanford slides (below) or browse the internet for others. Use *Example1-1_Weight_Init_v2.ipynb* from moodle as a template.
- Upload your observations, comments and your code to Moodle.

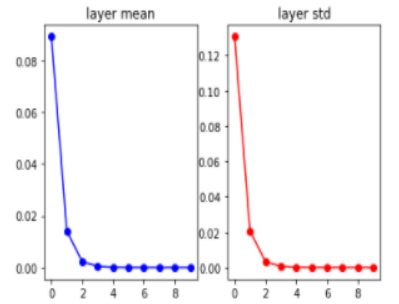


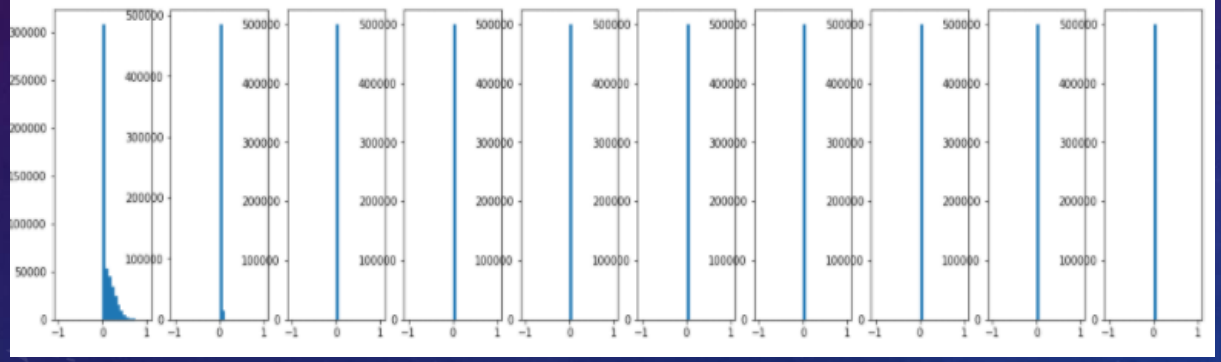
Solution

```
act={'relu': lambda x: np.maximum(0,x), 'tanh': lambda x: np.tanh(x),
    'leakyrelu': lambda x: np.maximum(0.1*x,x), 'sigmoid': lambda x: 1 / (1 + np.exp(-x))}
```

Relu

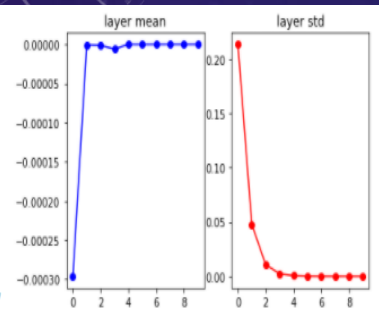
```
input layer had mean 0.0009335135909438322 and std 0.9997689831675833
hidden layer 1 had mean 0.014019580129699966 and std 0.1307316129065503
hidden layer 2 had mean 0.014019580129699966 and std 0.020647808432871017
hidden layer 3 had mean 0.014019580129699966 and std 0.003290683117809404
hidden layer 4 had mean 0.014019580129699966 and std 0.0005534896263596907
hidden layer 5 had mean 0.014019580129699966 and std 8.585049727234856e-05
hidden layer 6 had mean 0.014019580129699966 and std 1.342331666859198e-05
hidden layer 7 had mean 0.014019580129699966 and std 2.0283581252651962e-06
hidden layer 8 had mean 0.014019580129699966 and std 3.436872307399183e-07
hidden layer 9 had mean 0.014019580129699966 and std 5.6567040149436027e-08
hidden layer 10 had mean 0.014019580129699966 and std 9.587618192002277e-09
```

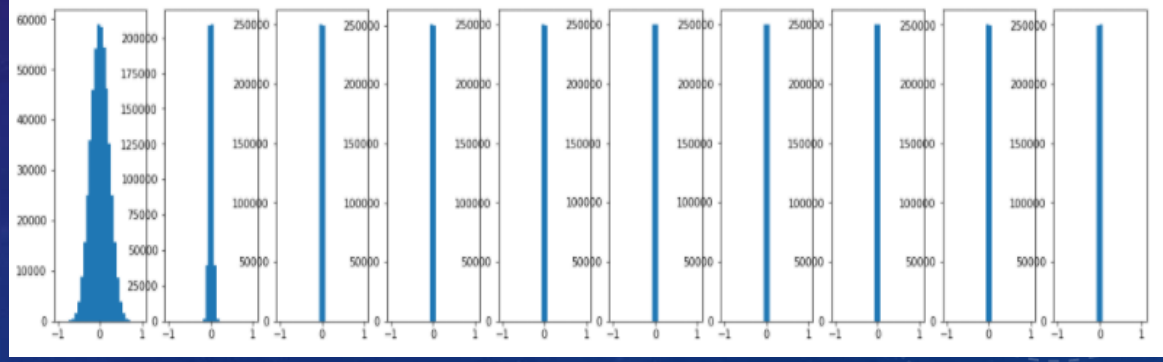




Tanh

```
input layer had mean 0.0016972016949942422 and std 0.9996007506323715
hidden layer 1 had mean -4.321378442543371e-07 and std 0.21346517154813444
hidden layer 2 had mean -4.321378442543371e-07 and std 0.04781186361357239
hidden layer 3 had mean -4.321378442543371e-07 and std 0.01062970942673882
hidden layer 4 had mean -4.321378442543371e-07 and std 0.0023768271338609486
hidden layer 5 had mean -4.321378442543371e-07 and std 0.0005320356428437244
hidden layer 6 had mean -4.321378442543371e-07 and std 0.0001190996457944263
hidden layer 7 had mean -4.321378442543371e-07 and std 2.657968168665321e-05
hidden layer 8 had mean -4.321378442543371e-07 and std 5.927058059929101e-06
hidden layer 9 had mean -4.321378442543371e-07 and std 1.3244777257742804e-06
hidden layer 10 had mean -4.321378442543371e-07 and std 2.9692949582887295e-07
```

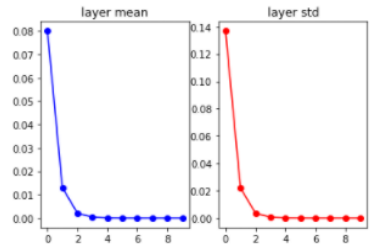


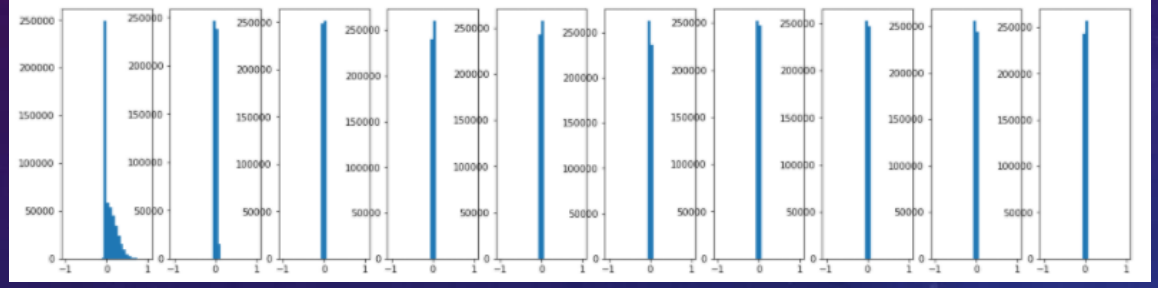


Solution

Leaky Relu

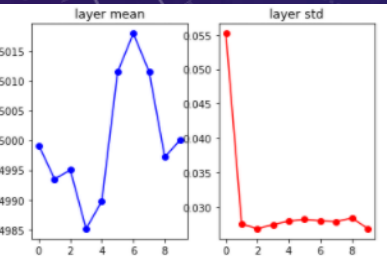
```
input layer had mean 0.0006237668063730286 and std 0.999450048502503
hidden layer 1 had mean 0.013004083297745514 and std 0.13689907198841048
hidden layer 2 had mean 0.013004083297745514 and std 0.021915925974818486
hidden layer 3 had mean 0.013004083297745514 and std 0.0035371501706149572
hidden layer 4 had mean 0.013004083297745514 and std 0.0005984772901338369
hidden layer 5 had mean 0.013004083297745514 and std 0.00010777243521971972
hidden layer 6 had mean 0.013004083297745514 and std 1.7138364266299613e-05
hidden layer 7 had mean 0.013004083297745514 and std 2.6710999722929167e-06
hidden layer 8 had mean 0.013004083297745514 and std 4.2799430670700576e-07
hidden layer 9 had mean 0.013004083297745514 and std 6.793395583726308e-08
hidden layer 10 had mean 0.013004083297745514 and std 1.0724698474700706e-08
```

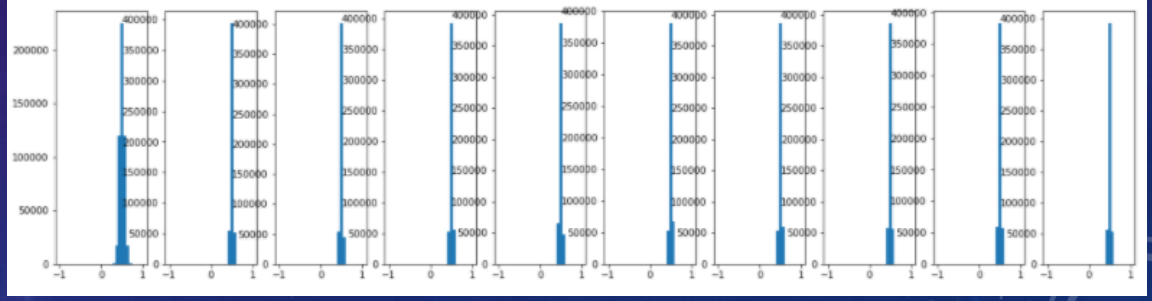




Sigmoid

```
input layer had mean -0.0016490615070105 92 and std 1.0004462164518235
hidden layer 1 had mean 0.499352139250211 and std 0.0552146079482946
hidden layer 2 had mean 0.499352139250211 and std 0.027524609047155174
hidden layer 3 had mean 0.499352139250211 and std 0.026817495834162094
hidden layer 4 had mean 0.499352139250211 and std 0.027431914881054995
hidden layer 5 had mean 0.499352139250211 and std 0.027929774245204337
hidden layer 6 had mean 0.499352139250211 and std 0.028174809718204567
hidden layer 7 had mean 0.499352139250211 and std 0.027973942477078708
hidden layer 8 had mean 0.499352139250211 and std 0.0278584540333884
hidden layer 9 had mean 0.499352139250211 and std 0.02838440757413669
hidden layer 10 had mean 0.499352139250211 and std 0.0268751728399357
```





Comparison of Different Channel on Feature Map

Example 3-3: Comparison of Different Channel on Feature Map

- Training the networks of the different output channels and compare the difference in feature maps.

Example 3-3: Comparison of Different Channel on Feature Map

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')
```

There are 10 classes in CIFAR10

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]

Example 3-3: Comparison of Different Channel on Feature Map

Define network(vgg16)

```
class Net(nn.Module):

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

        # calculate same padding:
        # (w - k + 2*p)/s + 1 = o
        # => p = (s(o-1) - w + k)/2

        self.feature = nn.Sequential(
            nn.Conv2d(in_channels=3,
                      out_channels=64,
                      kernel_size=(3, 3),
                      stride=(1, 1),
                      # (1(32-1)- 32 + 3)/2 = 1
                      padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.Conv2d(in_channels=64,
                      out_channels=64,
                      kernel_size=(3, 3),
                      stride=(1, 1),
                      padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(2, 2),
                         stride=(2, 2)),
```

You can change the output channel by yourself

```
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(2, 2),
                         stride=(2, 2))
        )

        self.classifier = nn.Sequential(
            nn.Linear(2048, 4096),
            nn.ReLU(True),
            nn.Dropout(p=0.65),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout(p=0.65),
            nn.Linear(4096, 10),
        )

        for m in self.modules():
            if isinstance(m, torch.nn.Conv2d) or isinstance(m, torch.nn.Linear):
                nn.init.kaiming_uniform_(m.weight, mode='fan_in', nonlinearity='leaky_relu')

                if m.bias is not None:
                    m.bias.detach().zero_()
```

```
def forward(self, x):

    x = self.feature(x)

    x = x.view(x.size(0), -1)
    x = self.classifier(x)

    return x
```

Define forward path

Example 3-3: Comparison of Different Channel on Feature Map

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 3-3: Comparison of Different Channel on Feature Map

Train the network and save the checkpoint

```
for epoch in range(1):    # 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(1)

Backpropagation the loss to update the weights.

Save the checkpoint

Example 3-3: Comparison of Different Channel on Feature Map

Compare the feature map and vector

```
myClass=FeatureVisualization('./jellyfish.jpg',2)
Compare=FeatureVisualization('./car.jpg',2)
print(myClass.pretrained_model2)

myClass.save_feature_to_img1()
Compare.save_feature_to_img2()
#  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  =  cos(myClass_vector,Compare_vector)
print("Verification:")
if  cosine_dist  >  0.4:
        print("They  are  the  same!")
        print("Their  cosine_similarity:{}".format(cosine_dist))
else:
        print("They  are  not  the  same!")
        print("Their  cosine_similarity:{}".format(cosine_dist))

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

Save the feature maps

Calculate the Euclidean distance between different pictures

Calculate the Cosine distance between different pictures

Define the threshold

Exercise 3-3: Comparison of Different Channel on Feature Map

- Please download 3-3 Comparison of Different Channel on Feature Map.ipynb from moodle
- Change the output channel of the same layer, observe the feature map is different, and increase the training epoch(choosing images from Internet)

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

Solution

Original

```
# We transform them to Tensors of normalized range [-1, 1].
class Net(nn.Module):

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

        # calculate same padding:
        # (w - k + 2*p)/s + 1 = o
        # => p = (s(o-1) - w + k)/2

        self.feature = nn.Sequential(
            nn.Conv2d(in_channels=3,
                      out_channels=64,
                      kernel_size=(3, 3),
                      stride=(1, 1),
                      # (1(32-1)- 32 + 3)/2 = 1
                      padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
```

Result

```
On layer:2, We can get the 64 feature maps
On layer:2, We can get the 64 feature maps
Verification:
They are the same!
Their cosine_similarity:tensor([0.7571], device='cuda:0', grad_fn=<DivBackward0>)
Their euclidean_dist:4.129570007324219
```

layer2

Modify

```
# calculate same padding:
# (w - k + 2*p)/s + 1 = o
# => p = (s(o-1) - w + k)/2

self.feature = nn.Sequential(
        nn.Conv2d(in_channels=3,
                  out_channels=128,
                  kernel_size=(3, 3),
                  stride=(1, 1),
                  # (1(32-1)- 32 + 3)/2 = 1
                  padding=1),
        nn.BatchNorm2d(128),
        nn.ReLU(),
```

Result

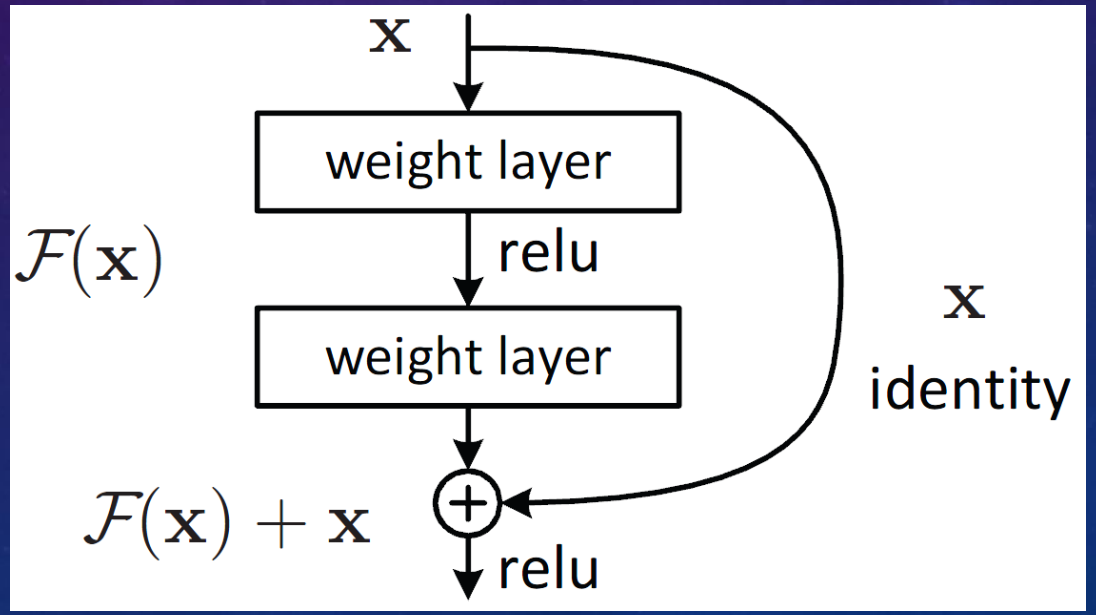
```
On layer:2, We can get the 128 feature maps
On layer:2, We can get the 128 feature maps
Verification:
They are the same!
Their cosine_similarity:tensor([0.7845], device='cuda:0', grad_fn=<DivBackward0>)
Their euclidean_dist:43.9280891418457
```

layer2

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 3-4: 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 3-4: 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')
```

There are 10 classes in CIFAR10

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]

Example 3-4: 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 3-4: 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 3-4: 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 3-4: 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 3-4: 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 3-4: 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 3-4: 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))
```

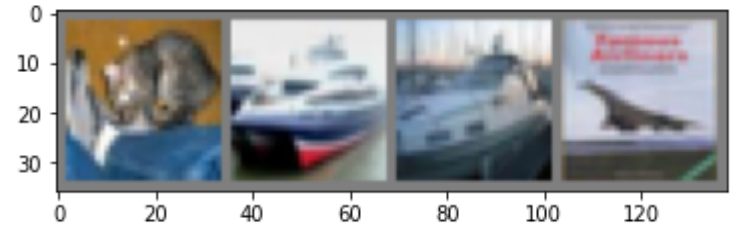
In the testing stage, the weights are fixed.

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

Example 3-4: 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 3-4: Comparison of Deep Networks

- Please download 3-4 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.

Solution

Define Net's network on Resnet18

```
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

```
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
        super(ResNet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.linear = nn.Linear(512*block.expansion, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(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.linear(out)
        return out

def Net():

    return ResNet(BasicBlock, [2, 2, 2, 2])
```

Solution

Result:

```
Epoch : 1 steps : 2000 Training Loss : 2.117974672973156
Epoch : 1 steps : 4000 Training Loss : 1.7780956227183342
Epoch : 1 steps : 6000 Training Loss : 1.5950266527533532
Epoch : 1 steps : 8000 Training Loss : 1.457605205014348
Epoch : 1 steps : 10000 Training Loss : 1.3227422883436084
Epoch : 1 steps : 12000 Training Loss : 1.2041479910723865
Epoch : 2 steps : 2000 Training Loss : 1.0617947282977402
Epoch : 2 steps : 4000 Training Loss : 1.030461528196931
Epoch : 2 steps : 6000 Training Loss : 0.9655739871114493
Epoch : 2 steps : 8000 Training Loss : 0.9034403795124963
Epoch : 2 steps : 10000 Training Loss : 0.8588768947087229
Epoch : 2 steps : 12000 Training Loss : 0.8167751074163243
Epoch : 3 steps : 2000 Training Loss : 0.7196904533319175
Epoch : 3 steps : 4000 Training Loss : 0.7030686599572655
Epoch : 3 steps : 6000 Training Loss : 0.6714745255545713
Epoch : 3 steps : 8000 Training Loss : 0.6320362791607622
Epoch : 3 steps : 10000 Training Loss : 0.6508280203730101
Epoch : 3 steps : 12000 Training Loss : 0.6577154291847255
Finished Training
```


```
GroundTruth:    cat  ship  ship plane
Predicted:      dog  ship   cat plane
Accuracy : 77 %
```

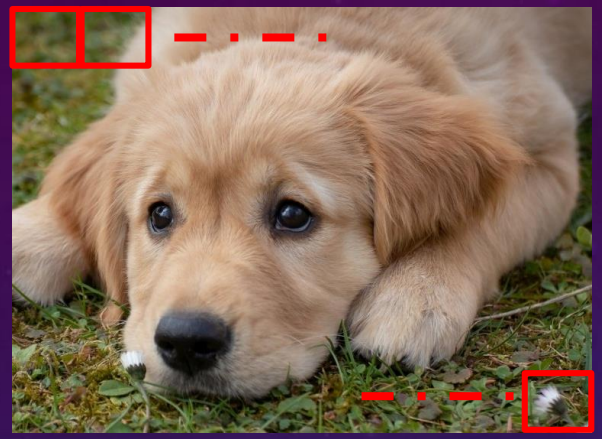
Loss will be decay

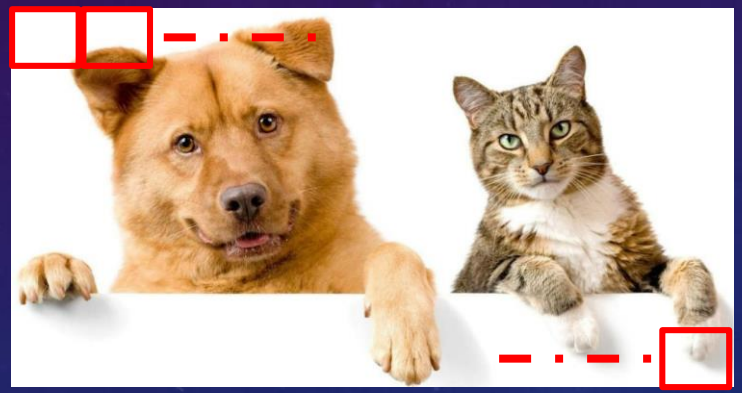
Comparison of Latent Vector

Example 3-5: Comparison of Latent Vector

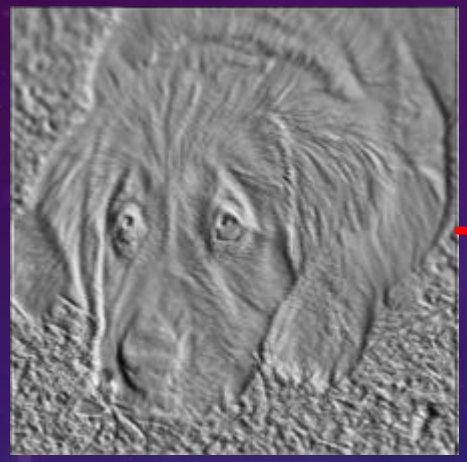
- Use VGG16-pretrain to predict the image and compare the similarity between two images, when there is only one class and two classes, and the difference the scores when the crop is in different places.

Example 3-5 Comparsion1

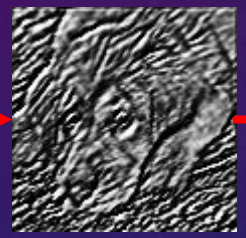




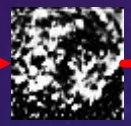
Conv 1-2



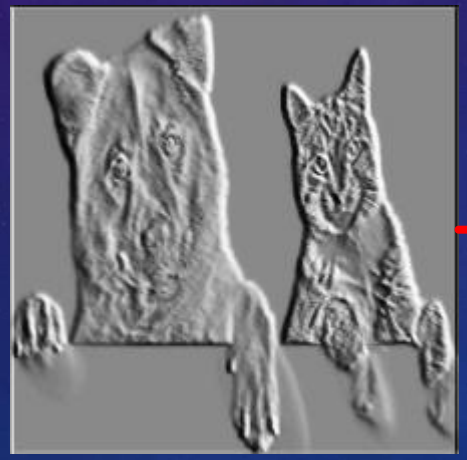
Conv 2-2



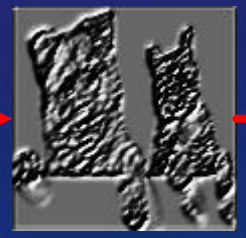
Conv 3-2



Conv 1-2



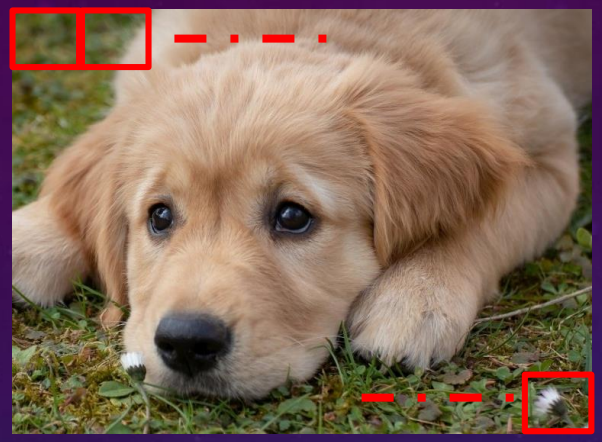
Conv 2-2

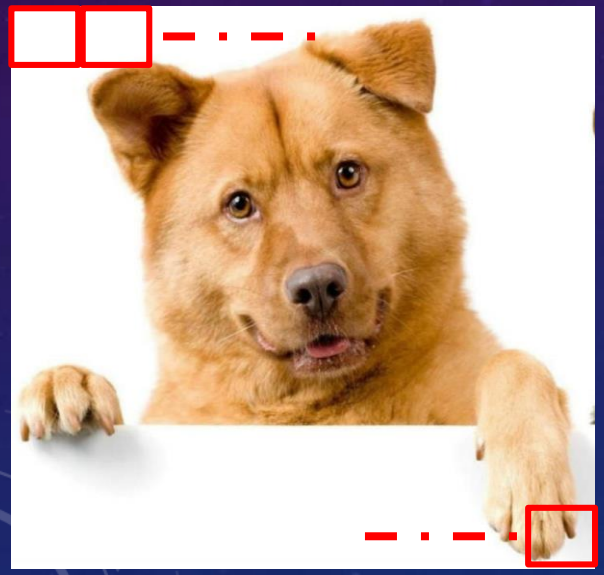


Conv 3-2

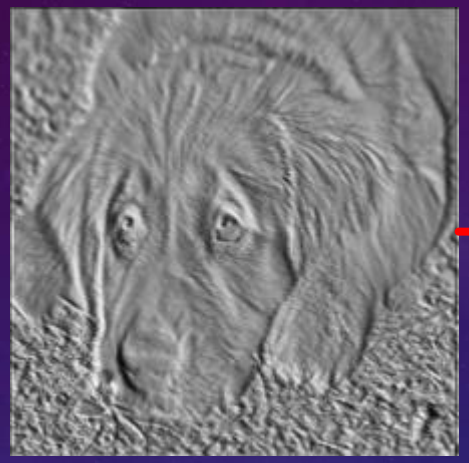
Cosine Similarity:
0.7003

Example 3-5 Comparsion1

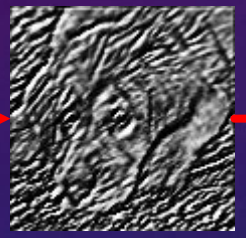




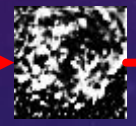
Conv 1-2



Conv 2-2

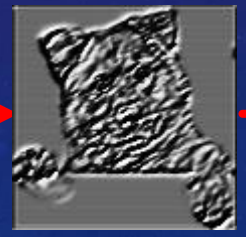


Conv 3-2



Conv 1-2

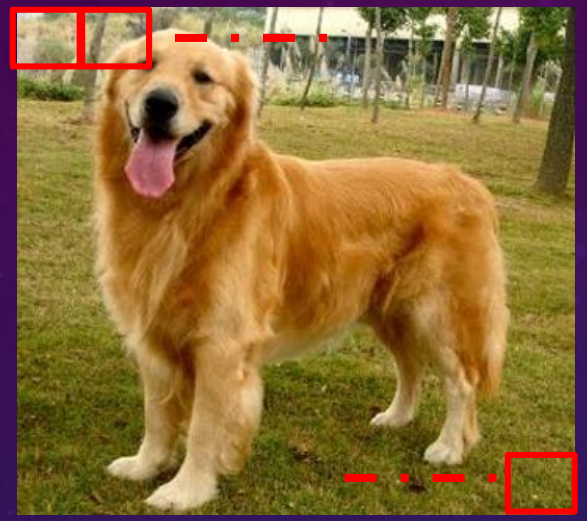
Conv 2-2

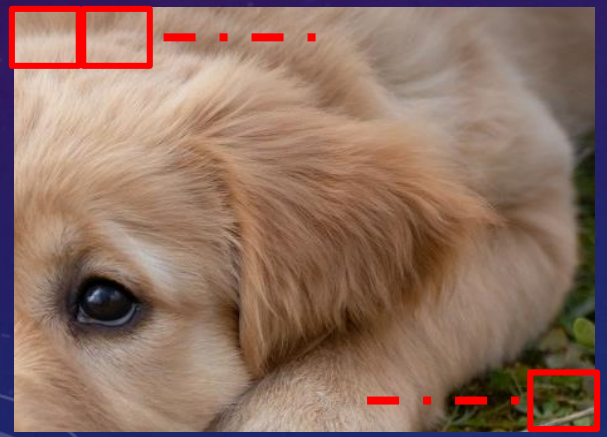


Conv 3-2

Cosine Similarity:
0.7871

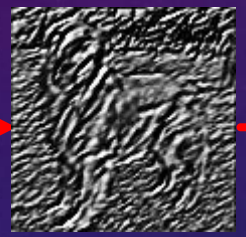
Example 3-5 Comparsion2





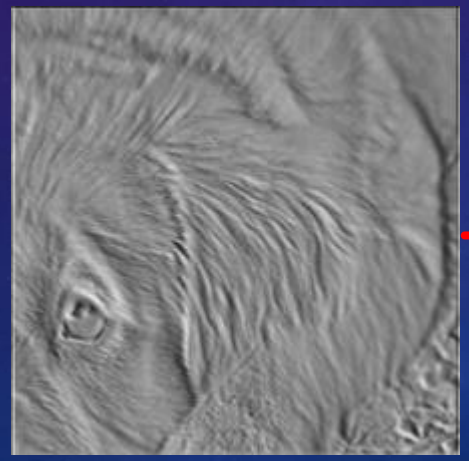
Conv 1-2

Conv 2-2

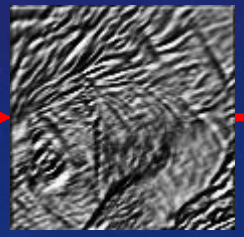


Conv 3-2

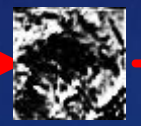
Conv 1-2



Conv 2-2

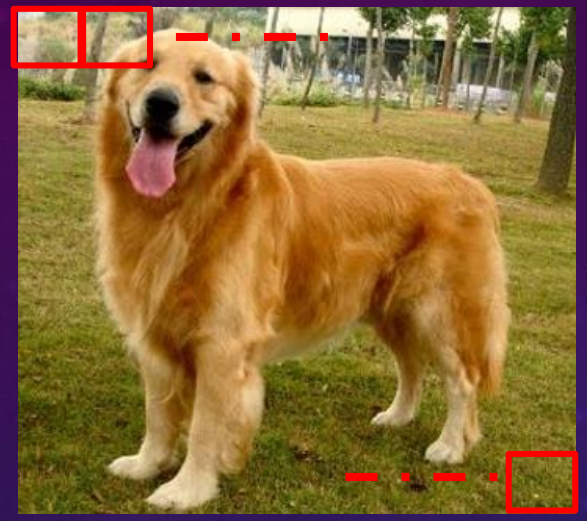


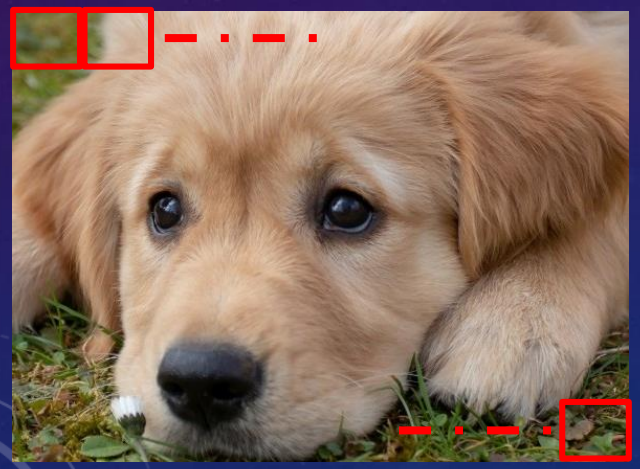
Conv 3-2



Cosine Similarity:
0.8075

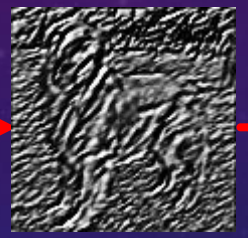
Example 3-5 Comparsion2





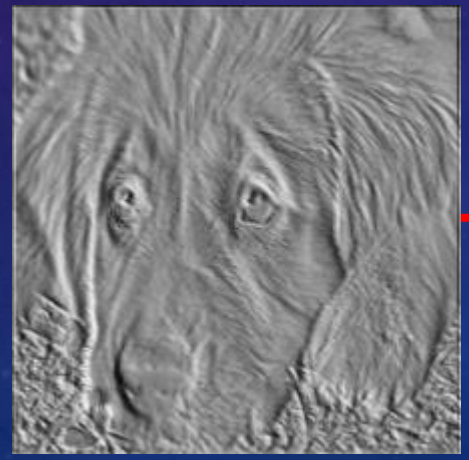
Conv 1-2

Conv 2-2

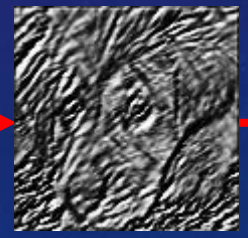


Conv 3-2

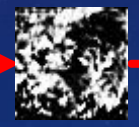
Conv 1-2



Conv 2-2

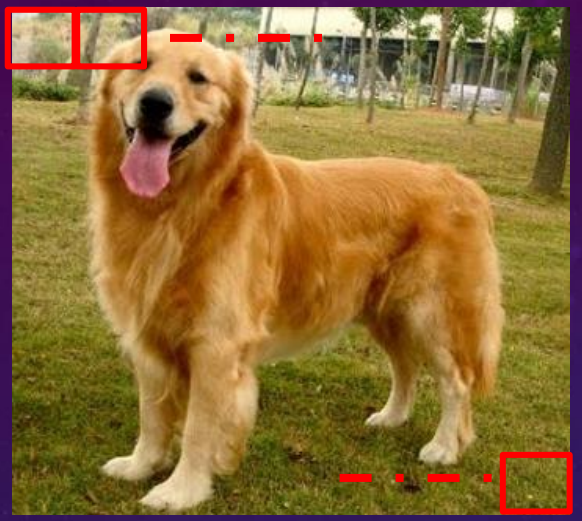


Conv 3-2



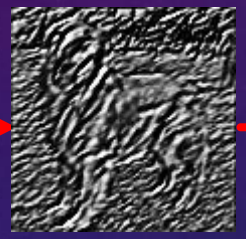
Cosine Similarity:
0.8997

Example 3-5 Comparsion3



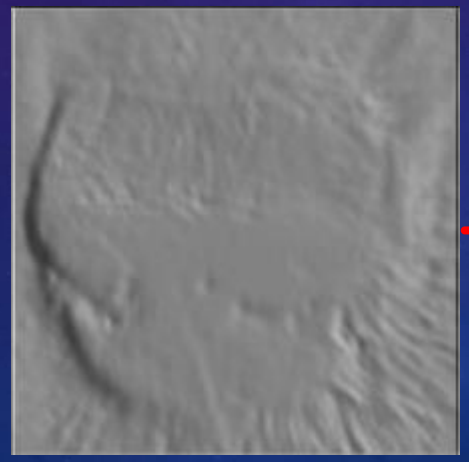
Conv 1-2

Conv 2-2

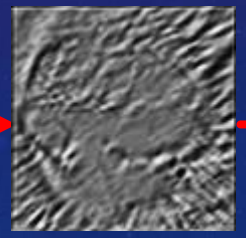


Conv 3-2

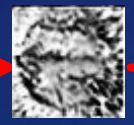
Conv 1-2



Conv 2-2

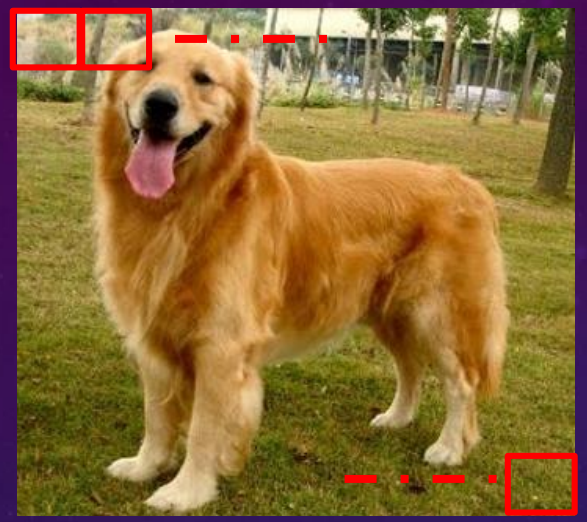


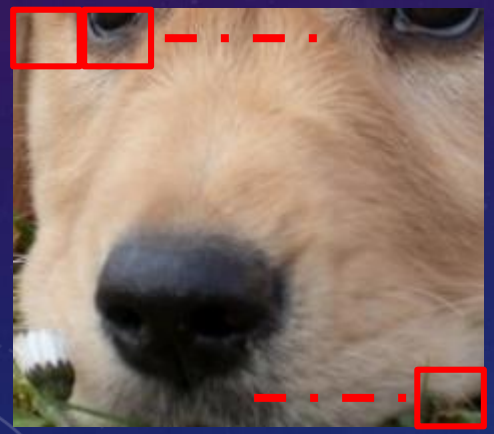
Conv 3-2



Cosine Similarity:
0.2748

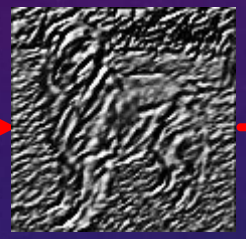
Example 3-5 Comparsion3





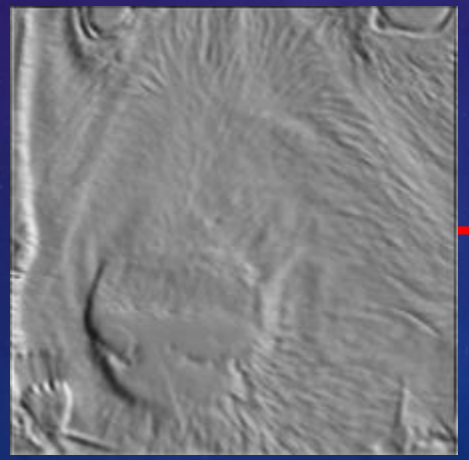
Conv 1-2

Conv 2-2

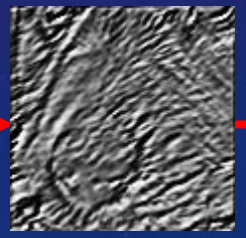


Conv 3-2

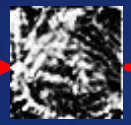
Conv 1-2



Conv 2-2

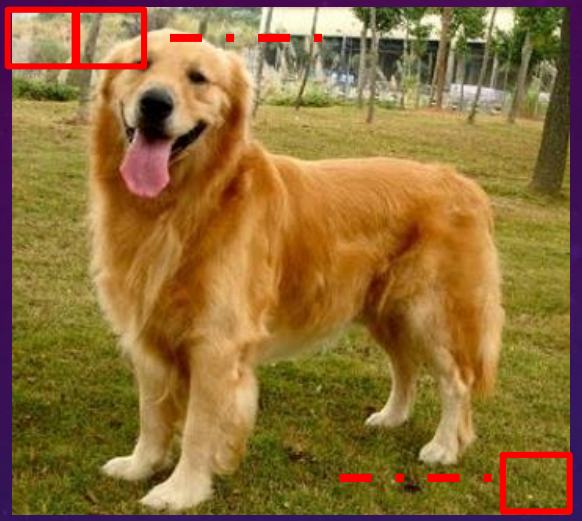


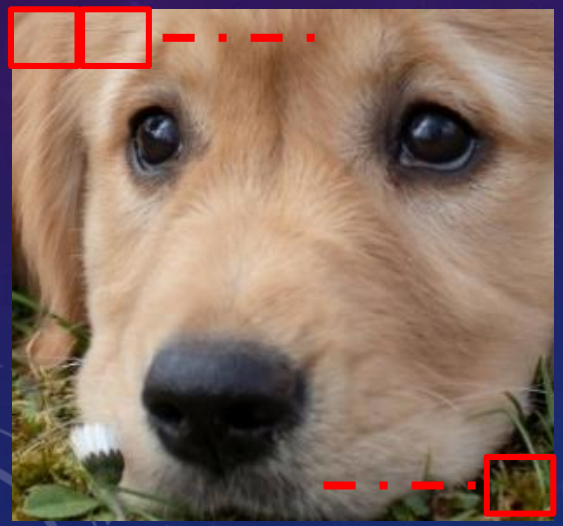
Conv 3-2



Cosine Similarity:
0.7552

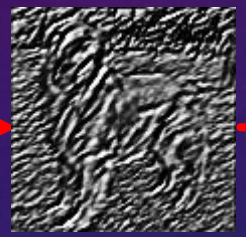
Example 3-5 Comparsion3





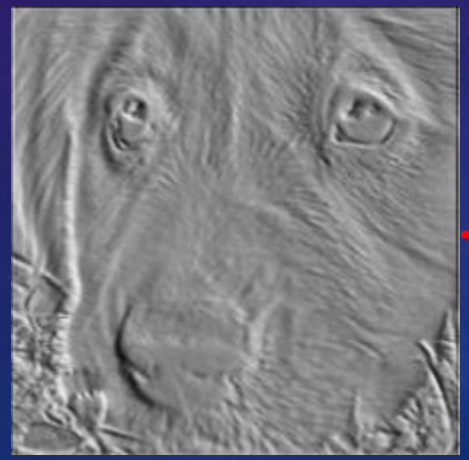
Conv 1-2

Conv 2-2

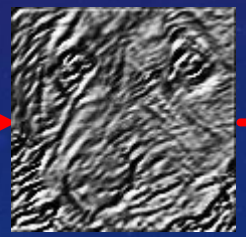


Conv 3-2

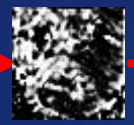
Conv 1-2



Conv 2-2



Conv 3-2



Cosine Similarity:
0.8409

Exercise 3-5: : Comparison of Latent Vector

- Please download example3-5 Comparison of Latent Vector.ipynb from moodle
- Please follow the example format to compare the similarity between the two images. Please find pictures online.
- Please use the Resnet18-pretrain model to compare the latent vector of the VGG16-pretrain model.
- Upload your observations, comments and your code to Moodle.

Solution

First:

Please refer to the example for the first question.

Second:

VGG16

```
# 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 = models.resnet18(pretrained=True)
self.pretrained_model2.eval()
```

```
Verification:
They are the same!
Their cosine_similarity:tensor([0.8409], grad_fn=<DivBackward0>)
Their euclidean_dist:60.07851791381836
```

Resnet18

```
# 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 = models.resnet18(pretrained=True)
self.pretrained_model2.eval()
```

```
Verification:
They are the same!
Their cosine_similarity:tensor([0.7690], grad_fn=<DivBackward0>)
Their euclidean_dist:56.894046783447266
```

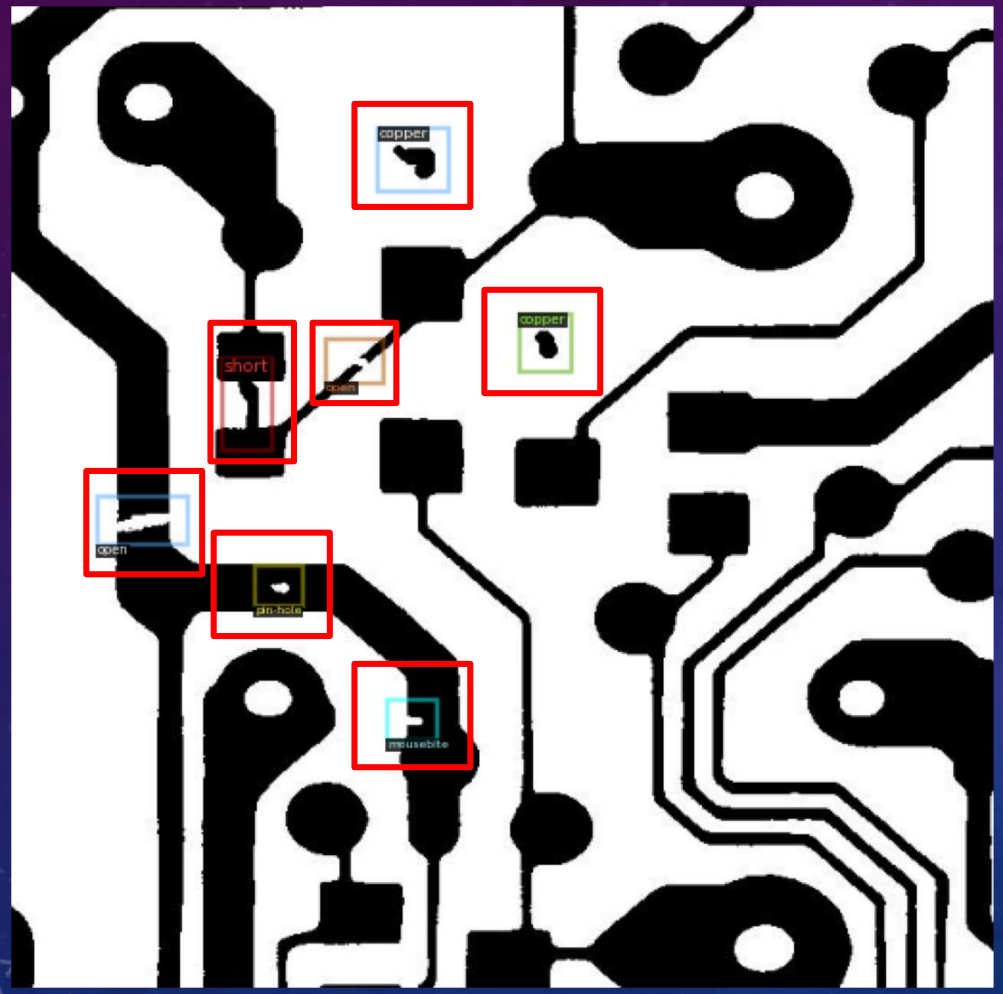
Chapter4

Example 4-1 – Faster RCNN

In this example, we want to use Faster RCNN to detect defects on a Printed Circuit Board (PCB). The problem is a classification of 6 different defect types. The defect types can be a short circuit or Pin-hole on the circuits for example. You can see an overview of all defect types in the following slide. We will use pretrained weights, which were trained on the COCO dataset and retrain on our own data.

Example 4-1 – Faster RCNN

PCB data sample:

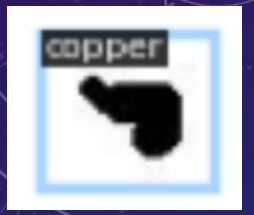


Category Type Sample:

Open

Spur

Short



Copper

Pin-hole

Copper

Mousebite

Training data :1000 images
Testing data : 500 images

Example 4-1 – Faster RCNN

Step 1 : Set up environment

```
!pip install -U torch==1.5 torchvision==0.6 -f https://download.pytorch.org/whl/cu101/torch_stable.html
!pip install cython pyyaml==5.1
!pip install -U 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'
import torch, torchvision
print(torch.__version__, torch.cuda.is_available())
!gcc --version
```

Result :

```
Successfully installed torch-1.5.0+cu101 torchvision-0.6.0+cu101
Requirement already satisfied: cython in /usr/local/lib/python3.6/dist-packages (0.29.20)
```

```
Successfully built pyyaml
```

```
Successfully installed pyyaml-5.1
```

```
Successfully installed pycocotools-2.0
```

Example 4-1 – Faster RCNN

Step 1 : Set up environment

```
!git clone https://github.com/tangsanli5201/DeepPCB
# install detectron2:
!pip install detectron2==0.1.3 -f https://dl.fbaipublicfiles.com/detectron2/wheels/cu101/torch1.5/index.html
```

Result :

```
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.6/dist-packages (from pyasn1-modules>=0.2.1->google-auth<2,>=1.6.3->tensc
Building wheels for collected packages: fvcore
  Building wheel for fvcore (setup.py) ... done
  Created wheel for fvcore: filename=fvcore-0.1.1.post20200630-cp36-none-any.whl size=41299 sha256=0973f00611a6330425c434401316cea811cc11c9d0ffe9159f9c
  Stored in directory: /root/.cache/pip/wheels/80/eb/49/83b9d20a804f1b4b163d1c1451c670a2067a00175662516f01
Successfully built fvcore
Installing collected packages: yacs, portalocker, fvcore, mock, detectron2
Successfully installed detectron2-0.1.3+cu101 fvcore-0.1.1.post20200630 mock-4.0.2 portalocker-1.7.0 yacs-0.1.7
```

Example 4-1 – Faster RCNN

Register PCB dataset

```
def get_PCB_dict(data_list):
    dataset_dicts = []

    for i,path in enumerate(data_list):
        filename = path[0]
        height, width = cv2.imread(filename).shape[:2]
        record = {}
        record['file_name'] = filename
        record['image_id'] = i
        record['height']= height
        record['width']= width

        objs = []
        with open(path[1]) as t:
            print(path[1])
            lines = t.readlines()
            print(lines)
```

The txt contains all the information in each image with all defect locations and types

Example 4-1 – Faster RCNN

We need to convert PCB-data into COCO format(.JSON) for training.
Here are some important information.

```
        for line in lines:
            if line[-1]=="\n":
              box = line[:-1].split(' ')
            else:
              box = line.split(' ')
            print(box)
            boxes = list(map(float,[box[0],box[1],box[2],box[3]]))
            category = int(box[4])
            print(boxes)
            obj = {
                "bbox": boxes,
                "bbox_mode": BoxMode.XYXY_ABS,
                #"segmentation": [poly], To draw a line, along to ballon
                #you will need this for mask RCNN
                "category_id": category-1,
                "iscrowd": 0
            }
            print(obj)
            objs.append(obj)
        record["annotations"] = objs
    dataset_dicts.append(record)
return dataset_dicts #list of dicts
```

The location and type

The location(point from top left to bottom right) [x1,y1,x2,y2]

Type:0-open 1-short 2-mousebite 3-spur 4-copper 5-pin-hole

Example 4-1 – Faster RCNN

Take a photo and print the information


```
{'file_name': './DeepPCB/PCBData/group20085/20085/20085000_test.jpg', 'image_id': 0, 'height': 640, 'width': 640, 'annotations':

[{'bbox': [409.0, 394.0, 435.0, 422.0], 'bbox_mode': <BoxMode.XYXY_ABS: 0>, 'category_id': 2, 'iscrowd': 0},
{'bbox': [275.0, 383.0, 319.0, 417.0], 'bbox_mode': <BoxMode.XYXY_ABS: 0>, 'category_id': 2, 'iscrowd': 0},
{'bbox': [8.0, 163.0, 36.0, 191.0], 'bbox_mode': <BoxMode.XYXY_ABS: 0>, 'category_id': 3, 'iscrowd': 0},
{'bbox': [244.0, 151.0, 270.0, 182.0], 'bbox_mode': <BoxMode.XYXY_ABS: 0>, 'category_id': 4, 'iscrowd': 0},
{'bbox': [338.0, 519.0, 364.0, 543.0], 'bbox_mode': <BoxMode.XYXY_ABS: 0>, 'category_id': 5, 'iscrowd': 0},
{'bbox': [476.0, 460.0, 502.0, 481.0], 'bbox_mode': <BoxMode.XYXY_ABS: 0>, 'category_id': 3, 'iscrowd': 0}]}
```

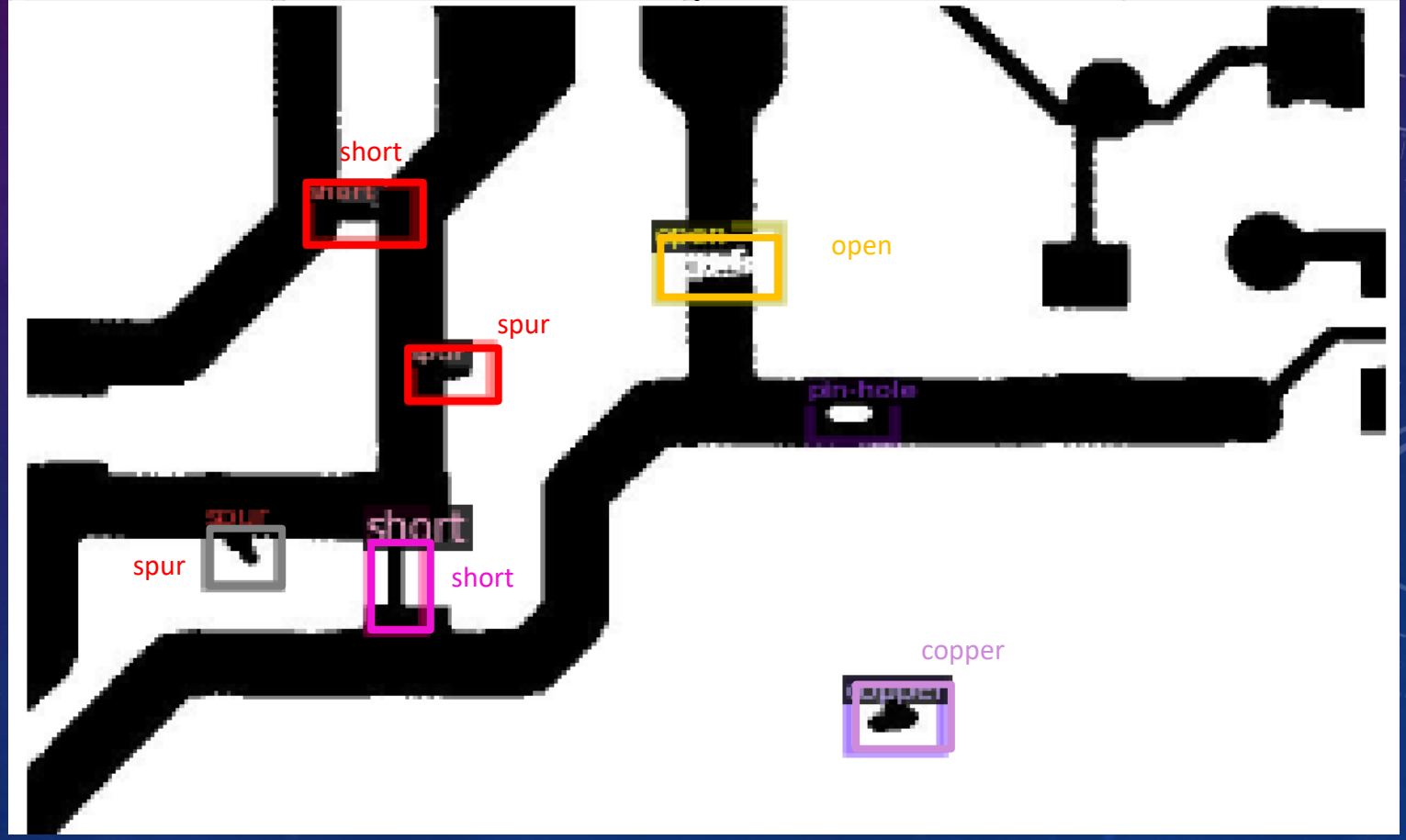

bbox: Defect bounding box coordinates
Bbox_mode: Bbox format
Category: Defect category
Iscrowd: 0: An object
1: A set of objects

Example 4-1 – Faster RCNN

Visualizing the Train Dataset

Randomly select 2 pictures from the train folder of the dataset and check the appearance of the bounding box.

```
for d in random.sample(dataset_dicts, 2):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::-1], metadata=PCB_metadata, scale=0.5)
    vis = visualizer.draw_dataset_dict(d)
    cv2_imshow(vis.get_image()[:, :, ::-1])
```



Example 4-1 – Faster RCNN

Define Hyper-parameter

In this part we select the faster_rcnn_R_50_FPN_3x pre-trained model in the model library. The model has been pre-trained on the COCO dataset.

```
from detectron2.engine import DefaultTrainer
from detectron2.config import get_cfg

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("PCB_train",)
cfg.DATASETS.TEST = ()
cfg.DATALOADER.NUM_WORKERS = 0
cfg.MODEL.WEIGHTS = "detectron2://COCO-Detection/faster_rcnn_R_50_FPN_3x/137849458/model_final_280758.pkl"  # Let training initialize from model zoo
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.00025  # pick a good LR
cfg.SOLVER.MAX_ITER = 300     # 300 iterations seems good enough for this toy dataset; you may need to train longer for a practical dataset
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 4096    # faster, and good e
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 6

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

Pretrained model

Define the max iterations of training

Set the trainer

Load last checkpoint or model.weights

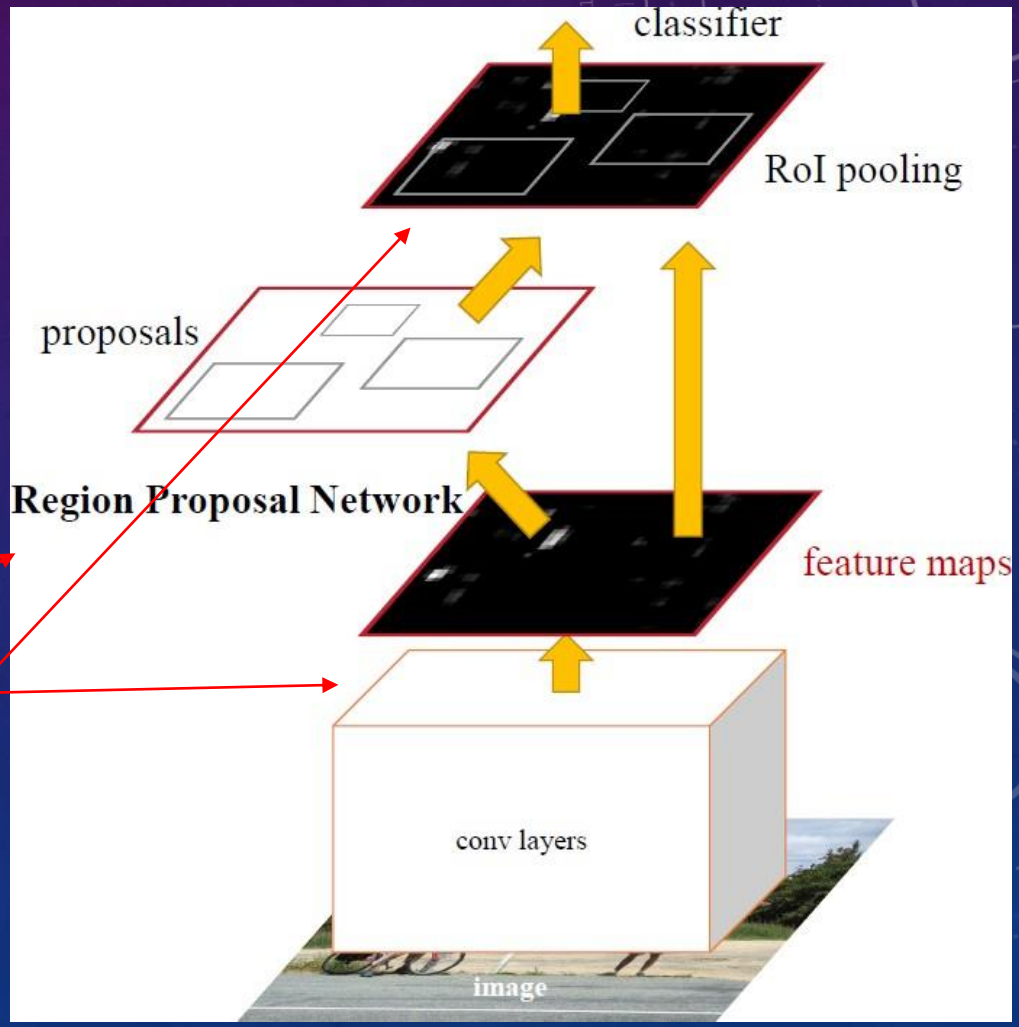
Start to train

Example 4-1 – Faster RCNN

This conv layers architecture uses FPN

```
class FPN(Backbone):
    """
    This module implements :paper:`FPN`.
    It creates pyramid features built on top of some input feature maps.
    """

    def __init__(
        self, bottom_up, in_features, out_channels, norm="", top_block=None, fuse_type="sum"
    ):
        """
```

```
features = self.backbone(images.tensors)
if isinstance(features, torch.Tensor):
    features = OrderedDict([('0', features)])
proposals, proposal_losses = self.rpn(images, features, targets)
detections, detector_losses = self.roi_heads(features, proposals, images.image_sizes, targets)
detections = self.transform.postprocess(detections, images.image_sizes, original_image_sizes)
```



Perform NMS and map the box back to the original image through original_images_size

Example 4-1 – Faster RCNN

rpn.py

```
def forward(self, images, features, gt_instances=None):
    """
    Args:
        images (ImageList): input images of length `N`
        features (dict[str: Tensor]): input data as a mapping from feature
            map name to tensor. Axis 0 represents the number of images `N` in
            the input data; axes 1-3 are channels, height, and width, which may
            vary between feature maps (e.g., if a feature pyramid is used).
        gt_instances (list[Instances], optional): a length `N` list of `Instances`s.
            Each `Instances` stores ground-truth instances for the corresponding image.

    Returns:
        proposals: list[Instances]: contains fields "proposal_boxes", "objectness_logits"
        loss: dict[Tensor] or None
    """
    features = [features[f] for f in self.in_features]
    pred_objectness_logits, pred_anchor_deltas = self.rpn_head(features)
    anchors = self.anchor_generator(features)
```

Realize the core function of the RPN network, output the classification vector objectness, and realize the background classification. Store the four offsets of the candidate box in pred_anchor_deltas

Generate bounding box

```
with torch.no_grad():
    # Find the top proposals by applying NMS and removing boxes that
    # are too small. The proposals are treated as fixed for approximate
    # joint training with roi heads. This approach ignores the derivative
    # w.r.t. the proposal boxes' coordinates that are also network
    # responses, so is approximate.
    proposals = find_top_rpn_proposals(
        outputs.predict_proposals(),
        outputs.predict_objectness_logits(),
        images,
        self.nms_thresh,
        self.pre_nms_topk[self.training],
        self.post_nms_topk[self.training],
        self.min_box_side_len,
        self.training,
    )

return proposals, losses
```

Use NMS to pick bounding box

Example 4-1 – Faster RCNN

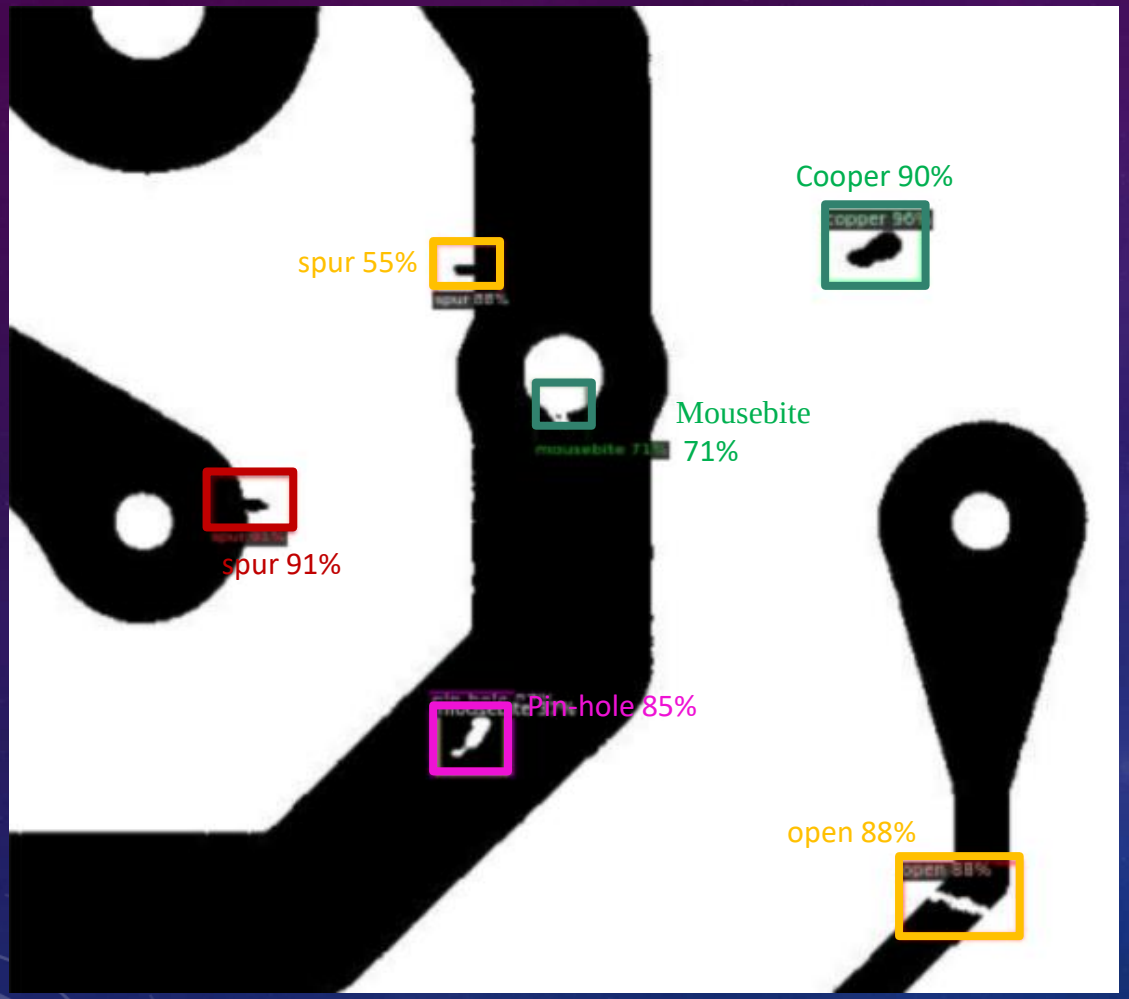
When starting to training , the colab will print the information for every 20 iterations

```
[08:57:18 d2.engine.train_loop]: Starting training from iteration 0
[08:57:26 d2.utils.events]:  eta: 0:18:07  iter: 19  total_loss: 2.585  loss_cls: 1.906  loss_box_reg: 0.042
[08:57:33 d2.utils.events]:  eta: 0:18:01  iter: 39  total_loss: 2.325  loss_cls: 1.703  loss_box_reg: 0.029
[08:57:40 d2.utils.events]:  eta: 0:17:50  iter: 59  total_loss: 1.812  loss_cls: 1.344  loss_box_reg: 0.034
[08:57:48 d2.utils.events]:  eta: 0:17:53  iter: 79  total_loss: 1.231  loss_cls: 0.869  loss_box_reg: 0.036
[08:57:55 d2.utils.events]:  eta: 0:17:50  iter: 99  total_loss: 0.912  loss_cls: 0.489  loss_box_reg: 0.038
[08:58:03 d2.utils.events]:  eta: 0:17:47  iter: 119  total_loss: 0.614  loss_cls: 0.247  loss_box_reg: 0.034
[08:58:11 d2.utils.events]:  eta: 0:17:48  iter: 139  total_loss: 0.466  loss_cls: 0.177  loss_box_reg: 0.068
[08:58:18 d2.utils.events]:  eta: 0:17:46  iter: 159  total_loss: 0.507  loss_cls: 0.176  loss_box_reg: 0.077
[08:58:27 d2.utils.events]:  eta: 0:17:47  iter: 179  total_loss: 0.521  loss_cls: 0.213  loss_box_reg: 0.098
```

```
loss_rpn_cls: 0.469  loss_rpn_loc: 0.154  time: 0.3619  data_time: 0.0438  lr: 0.000005  max_mem: 1826M
loss_rpn_cls: 0.416  loss_rpn_loc: 0.143  time: 0.3653  data_time: 0.0426  lr: 0.000010  max_mem: 1826M
loss_rpn_cls: 0.243  loss_rpn_loc: 0.135  time: 0.3647  data_time: 0.0437  lr: 0.000015  max_mem: 1826M
loss_rpn_cls: 0.162  loss_rpn_loc: 0.140  time: 0.3689  data_time: 0.0447  lr: 0.000020  max_mem: 1826M
loss_rpn_cls: 0.209  loss_rpn_loc: 0.122  time: 0.3682  data_time: 0.0422  lr: 0.000025  max_mem: 1826M
loss_rpn_cls: 0.185  loss_rpn_loc: 0.132  time: 0.3712  data_time: 0.0452  lr: 0.000030  max_mem: 1826M
loss_rpn_cls: 0.096  loss_rpn_loc: 0.119  time: 0.3742  data_time: 0.0428  lr: 0.000035  max_mem: 1826M
loss_rpn_cls: 0.098  loss_rpn_loc: 0.094  time: 0.3757  data_time: 0.0424  lr: 0.000040  max_mem: 1826M
loss_rpn_cls: 0.104  loss_rpn_loc: 0.084  time: 0.3790  data_time: 0.0440  lr: 0.000045  max_mem: 1826M
```

Example 4-1 – Faster RCNN

Visualizing the Testing Result



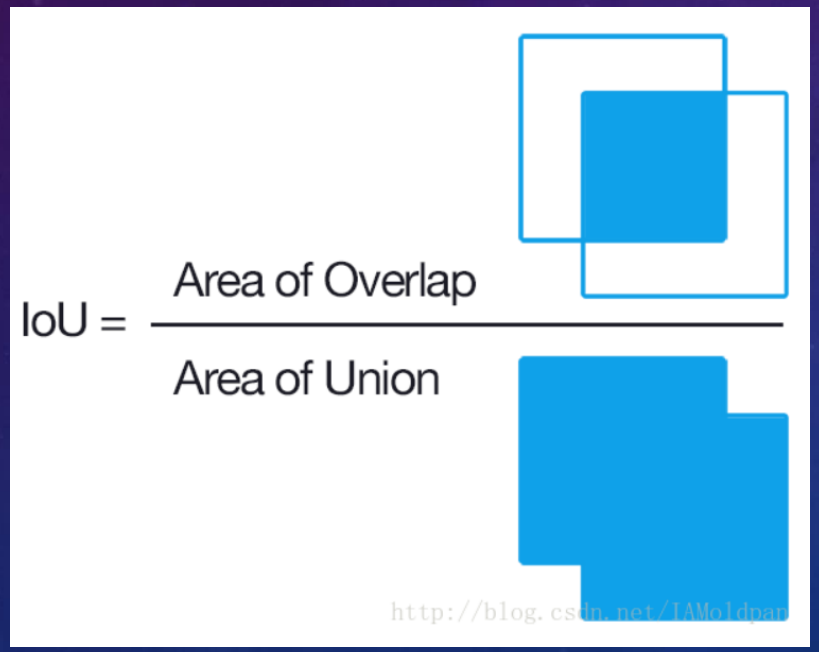

```
from detectron2.utils.visualizer import ColorMode
dataset_dicts = get_PCB_dict(test)
```

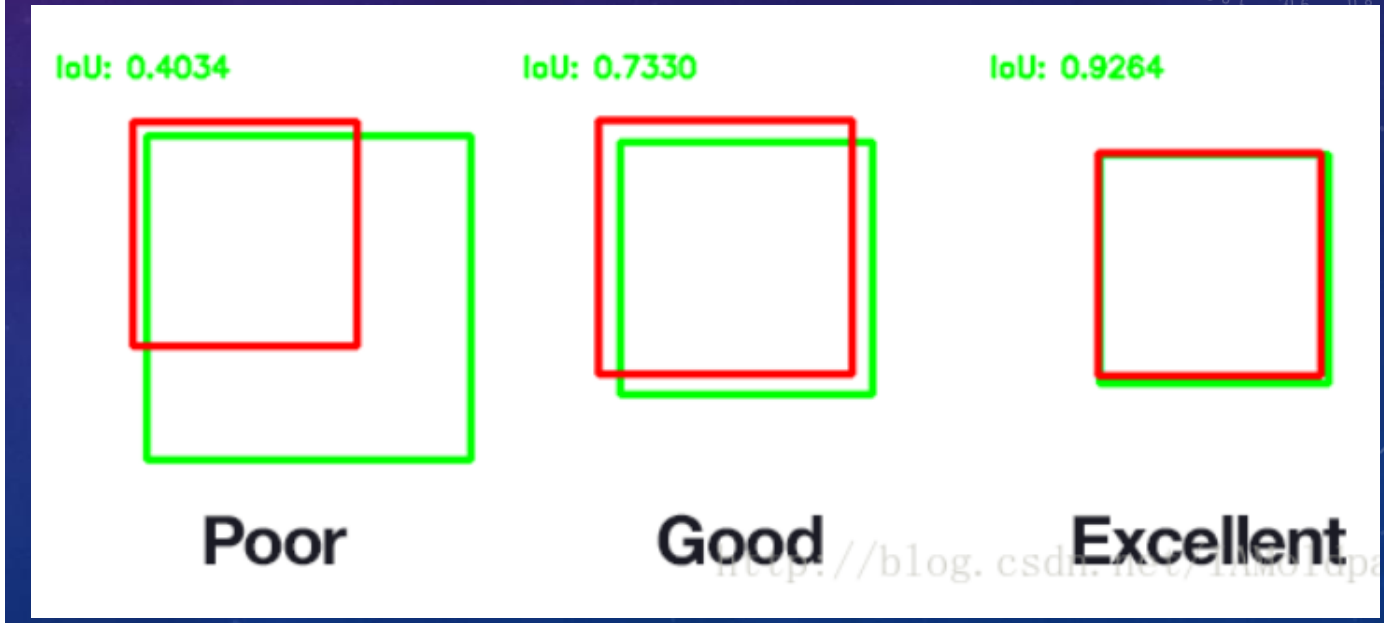
```
for d in random.sample(dataset_dicts, 10):
    im = cv2.imread(d["file_name"])
    outputs = predictor(im)
    v = Visualizer(im,
                   metadata=PCB_metadata,
                   scale=0.8,
                   instance_mode = ColorMode.IMAGE
                   )
                   # remove the colors of unsegmented pixels
    print(outputs['instances'].pred_classes)
    print(outputs["instances"].pred_boxes)

    v = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    cv2_imshow(v.get_image())
```

Example 4-1 – Faster RCNN

- Formally we define confidence as *Pr*(Object) * *IOU*(pred, truth) . If no object exists in that cell, the confidence score should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth.

- IOU





Example 4-1 – Faster RCNN

```python
if thresholds is None:
        step = 0.05
        thresholds = torch.arange(0.5, 0.95 + 1e-5, step, dtype=torch.float32)
recalls = torch.zeros_like(thresholds)
# compute recall for each iou threshold
for i, t in enumerate(thresholds):
        recalls[i] = (gt_overlaps >= t).float().sum() / float(num_pos)
# ar = 2 * np.trapz(recalls, thresholds)
ar = recalls.mean()
return {
        "ar": ar,
        "recalls": recalls,
        "thresholds": thresholds,
        "gt_overlaps": gt_overlaps,
        "num_pos": num_pos,
}
```

```python
# Compute per-category AP
# from https://github.com/facebookresearch/Detectron/blob/a6a835f5b8208c45d0dce217
precisions = coco_eval.eval["precision"]
# precision has dims (iou, recall, cls, area range, max dets)
assert len(class_names) == precisions.shape[2]

results_per_category = []
for idx, name in enumerate(class_names):
        # area range index 0: all area ranges
        # max dets index -1: typically 100 per image
        precision = precisions[:, :, idx, 0, -1]
        precision = precision[precision > -1]
        ap = np.mean(precision) if precision.size else float("nan")
        results_per_category.append(("{}".format(name), float(ap * 100)))
```

Compute recall for each IOU threshold

Compute per-category AP

Different IOU thresholds, from 0.5 to 0.95, step 0.05

Example 4-1 – Faster RCNN

Evaluate the trained model

```
from detectron2.evaluation import COCOEvaluator, inference_on_dataset, LVISEvaluator
from detectron2.data import build_detection_test_loader

evaluator = COCOEvaluator("PCB_test", cfg, False, output_dir="./output/")
val_loader = build_detection_test_loader(cfg, "PCB_test")
inference_on_dataset(trainer.model, val_loader, evaluator)
```

Result:

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.555
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.862
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.623
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.531
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.568
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.600
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.508
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.650
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.650
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.649
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.645
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.600
[10/26 07:08:07 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 55.525 | 86.206 | 62.343 | 53.122 | 56.825 | 60.000 |
[10/26 07:08:07 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP     | category   | AP     |
|:-----------|:-------|:-----------|:-------|:-----------|:-------|
| open       | 45.682 | short      | 30.675 | mousebite  | 53.424 |
| spur       | 56.391 | copper     | 82.105 | pin-hole   | 64.871 |
```

Area : target detection area
Small :
area<32*32
Medium :
32*32<area<96*96
Large :
area>96*96

AP of each category

Exercise 4-1

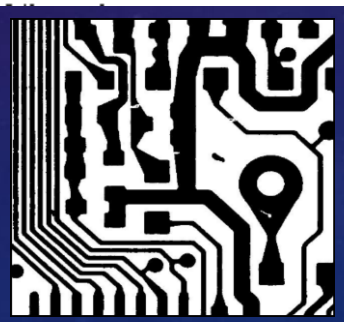
- Please download the “PCBdata_fasterRCNN_colab.ipynb” from the Moodle and open by the Colab.
- Follow the Colab code and pip install packages for environments.
- Visualize the PCB data and define the hyper-parameter for training.
- Compare different iterations and comment on the results.

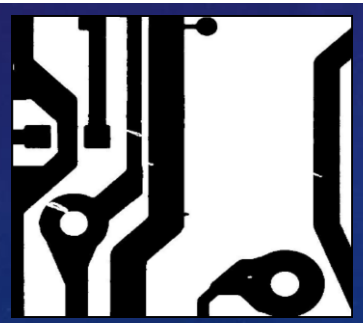
Please crop your results and code and paste to a MS Word, discuss the results, and then upload to the Moodles.

Solution

Iteration:30

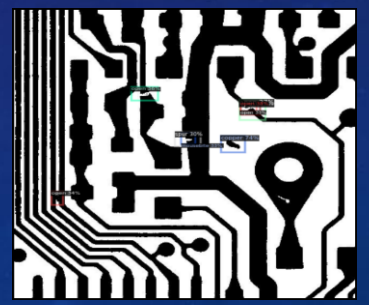
```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.001
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.007
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.004
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.005
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.013
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.015
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.001
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.019
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.000
[11/03 08:07:37 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|  AP   |  AP50  |  AP75  |  APs  |  APm  |  APl  |
|:-----:|:------:|:------:|:-----:|:-----:|:-----:|
| 0.109 | 0.674  | 0.000  | 0.000 | 0.354 | 0.000 |
[11/03 08:07:37 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category | AP    | category | AP    | category | AP    |
|:---------|:------|:---------|:------|:---------|:------|
| open     | 0.001 | short    | 0.000 | mousebite | 0.000 |
| spur     | 0.002 | copper   | 0.654 | pin-hole | 0.000 |
```

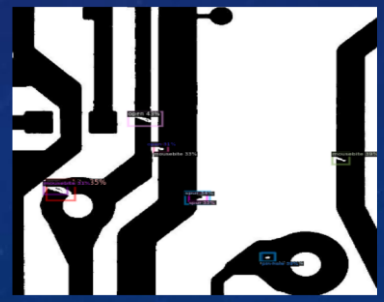




Iteration:1000

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.304
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.569
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.245
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.264
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.349
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.331
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.537
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.548
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.520
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.566
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.000
[11/03 08:50:23 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|  AP   |  AP50  |  AP75  |  APs  |  APm  |  APl  |
|:-----:|:------:|:------:|:-----:|:-----:|:-----:|
| 30.402 | 56.861 | 24.470 | 26.414 | 34.929 | 0.000 |
[11/03 08:50:23 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category | AP    | category | AP    | category | AP    |
|:---------|:------|:---------|:------|:---------|:------|
| open     | 20.531 | short    | 9.601 | mousebite | 19.532 |
| spur     | 22.547 | copper   | 72.572 | pin-hole | 37.631 |
```





Example 4-2 – License Plate Detector

In this example, we want to use Faster RCNN to detect license plate. We have to learn how to label the license plate and we have to change the data into COCO data format. We will use pretrained weights, which were trained on the COCO dataset and retrain on our own data and test our own data.

Example 4-2 : License Plate - labelme

Install labelme

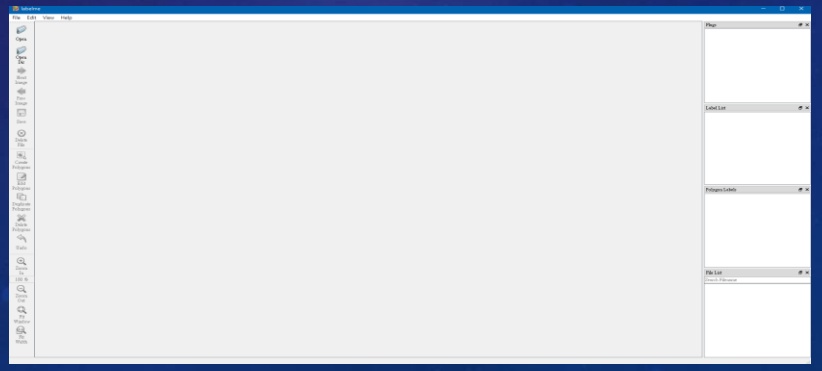
1. Python2 (Windows) Python 3 (Windows)

pip install pyqt pip install labelme	pip install pyqt5 pip install labelme

2. Cmd : labelme

```
(pythonGPU) C:\Users\andy>labelme
```

3. Start to label



Example 4-2 : Start to label License Plate

Example 4-2 : How to label License Plate

For example

Naming method:
(name+order)

Label list:

(normal class)plate1 plate2 ….
(yellow class)yellow_plate1 yellow_plate2 ….
(green class)green_plate1 green_plate2 ….
(red class)red_plate1 red_plate2 ….

Example 4-2 : Label Process

Step1:check the number of clear license plates in this image

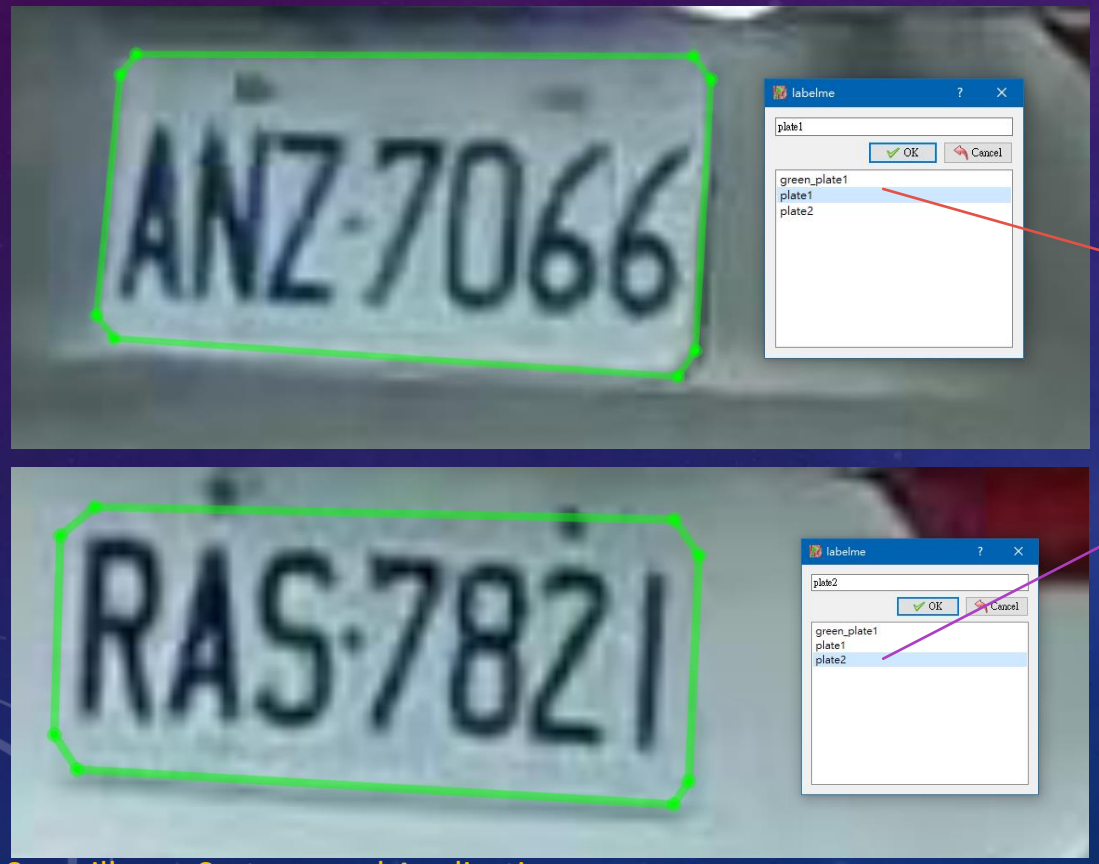
We can get two samples from this image

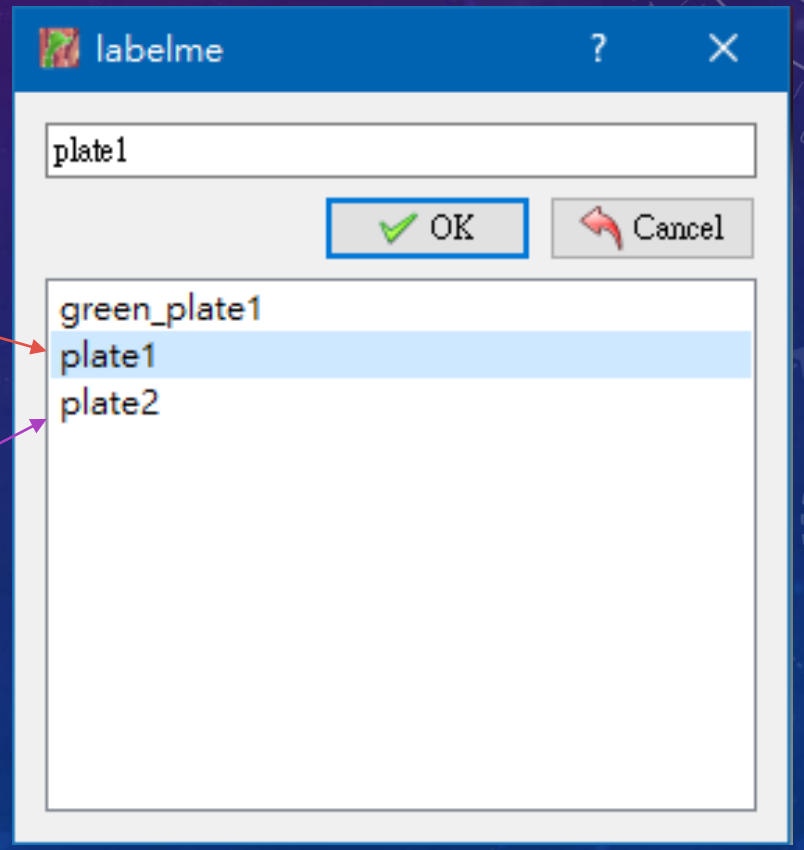
- There’re two clear license plates
- All the license plate are normal class so we can named plate1 and plate 2

Example 4-2 : Label Process

Step2:Partial enlargement the license plate and use the dots to bound it

Step3:Give the license plate their name and order





Example 4-2 : Label Process

From the labelme we can get several json files that contain bounding boxes in the images (1 json/img). If you want to use multiple json files for combining multiple images, please consider using labelme2coco.py (given in the labelme2coco.7z from Moodle)

Step1: Put your image folder in the labelme2coco folder

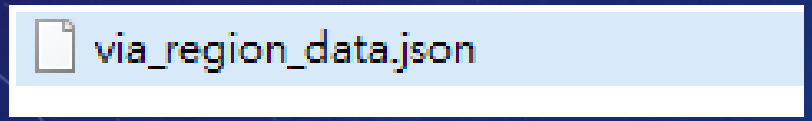
Step2: Make sure your json files and images are in the same folder

Step3: Open cmd and run labelme2coco.py [folder name], as shown below

```
(C:\Users\Micky\Anaconda3\envs\Pytorch10) Y:\108_Andy\henry\labelme2coco-master>python labelme2coco.py images
save coco json
via_region_data.json
```

The images folder name

Result : The file via_region_data.json combines all json files with the associated images.



Example 4-2：License Plate Detector

Step 1 : Set up environment

Install pytorch and detectron2

Step 2 : Prepare the “license plate dataset”

Download the dataset from https://drive.google.com/file/d/14wXRkpow5vlVc2ROhHbWelEjwCJCx9b5/view?usp=sharing

Step 3 : Write a function that loads the dataset into detectron2's standard format

Step 4 : Verify the data loading is correct

Step 5 : Fine-tune a pretrained model on the license plate dataset

Step 6 : Visualize the prediction results.

Example 4-2：License Plate Detector

Step 1 : Set up environment

```
!pip install -U torch torchvision
!pip install git+https://github.com/facebookresearch/fvcore.git
import torch, torchvision
torch.__version__
```

Result :

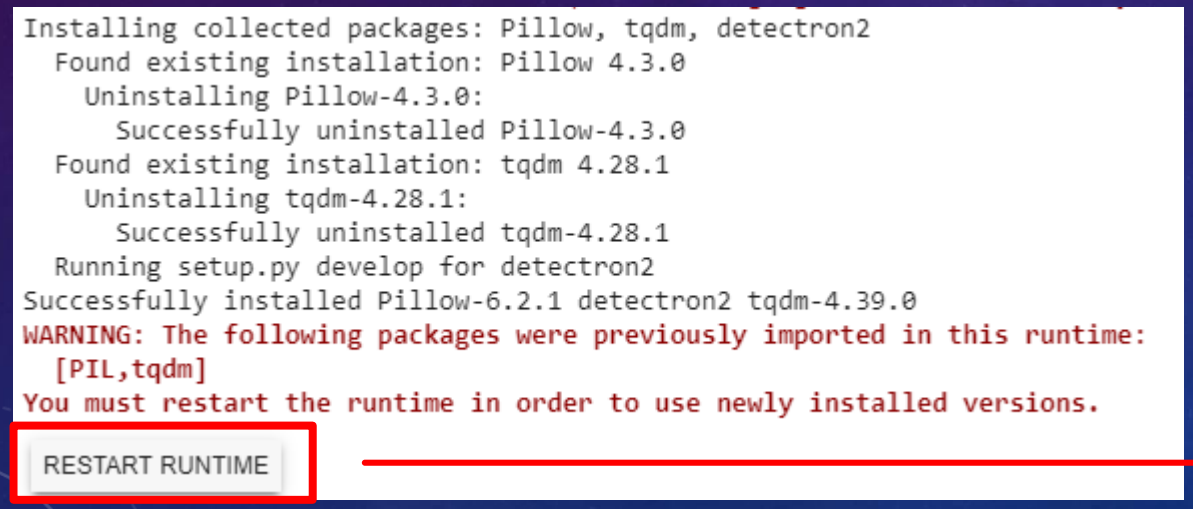
```
Successfully built fvcore pyyaml
Installing collected packages: pyyaml, yacs, portalocker, fvcore
  Found existing installation: PyYAML 3.13
    Uninstalling PyYAML-3.13:
      Successfully uninstalled PyYAML-3.13
Successfully installed fvcore-0.1 portalocker-1.5.2 pyyaml-5.1.2 yacs-0.1.6
'1.3.1'
```

Example 4-2：License Plate Detector

Step 1 : Set up environment

```
!git clone https://github.com/facebookresearch/detectron2 detectron2_repo
!pip install -e detectron2_repo
```

Result :



If you have already installed the detectron2, you need to “restart runtime”.

Example 4-2 : License Plate Detector

Step 2 : Prepare the “license plate dataset”

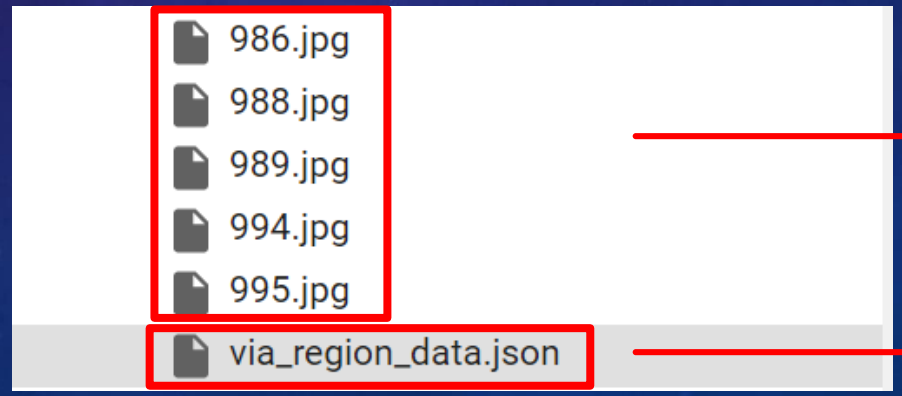
Option 1 : run !wget https://www.dropbox.com/s/rjra44iyzoh19te/plate.zip?dl=0 -O plate.zip

Option 2 : Download the plate.zip from
https://drive.google.com/file/d/14wXRkpow5vlVc2ROhHbWelEjwCJCx9b5/view?usp=sharing and upload it

```
# Unzip plate.zip.
!unzip plate.zip
```

Result :

In plate/train



Training data

Our label file

Example 4-2：License Plate Detector

Step 3 : Write a function that loads the dataset into detectron2's standard format

```python
def get_plate_dicts(img_dir):
    json_file = os.path.join(img_dir, "via_region_data.json") # via_region_data.json
    with open(json_file) as f:
        imgs_anns = json.load(f)
```

Open the “via_region_data.json”

```python
for i in range(len(filename_list)):
    record = {}
    filename = os.path.join(img_dir, filename_list[i]["file_name"])
    height, width = cv2.imread(filename).shape[:2]
    record["file_name"] = filename
    record["image_id"] = i
    record["height"] = 1080
    record["width"] = 1920
```

In plate/train/via_region_data.json

```
"images": [
    {
        "height": 1080,
        "width": 1920,
        "id": 0,
        "file_name": "10.jpg"
    },
```

Example 4-2：License Plate Detector

Step 3 : Write a function that loads the dataset into detectron2's standard format

```
for j in range(k,k+bbox_num[i]):
    if bbox_list[j]["image_id"] != record["image_id"]:
        assert False
    bbox = []
    bbox = [int(bbox_list[j]["bbox"][0]),int(bbox_list[j]["bbox"][1]),
            int(bbox_list[j]["bbox"][0])+int(bbox_list[j]["bbox"][2]),
            int(bbox_list[j]["bbox"][1])+int(bbox_list[j]["bbox"][3])]
    obj = {
        "bbox":bbox,
        "bbox_mode": BoxMode.XYXY_ABS,
        "segmentation":bbox_list[j]["segmentation"],
        "category_id":0,
        "iscrowd": 0
    }
    objs.append(obj)
record["annotations"] = objs
k += bbox_num[i]
dataset_dicts.append(record)
```

In plate/train/via_region_data.json

```
"bbox": [
    710.0,
    283.0,
    128.0,
    81.0
],
```

The width and height of plates

```
"segmentation": [
    [
        732.9621380846324,
        283.5189309576837,
        723.8307349665924,
        283.2962138084632,
        710.467706013363,
        327.8396436525612,
        714.6993318485523,
        331.62583518930956,
        815.5902004454342,
        364.81069042316255,
        821.826280623608,
        361.2472160356347,
        838.3073496659242,
        316.9265033407572,
        835.1893095768373,
        312.6948775055679
    ]
```

Example 4-2：License Plate Detector

Step 4 : Verify the data in the training folder

```
import random
dataset_dicts = get_plate_dicts("plate/train")
for d in random.sample(dataset_dicts, 3):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::-1], metadata=plate_metadata, scale=0.5)
    vis = visualizer.draw_dataset_dict(d)
    cv2_imshow(vis.get_image()[:, :, ::-1])
```

Randomly select 3 samples to verify.

Result :



Example 4-2：License Plate Detector

Step 5 : Fine-tune a pretrained model on the license plate dataset

```
from detectron2.engine import DefaultTrainer
from detectron2.config import get_cfg

cfg = get_cfg()
cfg.merge_from_file("./detectron2_repo/configs/COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")
cfg.DATASETS.TRAIN = ("plate_train",)
cfg.DATASETS.TEST = ()
cfg.DATALOADER.NUM_WORKERS = 5
cfg.MODEL.WEIGHTS = "detectron2://COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x/137849600/model_final_f10217.pkl"
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.00025
cfg.SOLVER.MAX_ITER = 2000
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 256
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1  # only has one class

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

Define the max iterations of training

Pretrained model

Example 4-2：License Plate Detector

Step 6：Visualize the prediction results.

```
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7   # set the testing threshold for this model
cfg.DATASETS.TEST = ("plate_val", )
predictor = DefaultPredictor(cfg)
```

Load trained model

```
from detectron2.utils.visualizer import ColorMode
dataset_dicts = get_plate_dicts("plate/val")
for d in random.sample(dataset_dicts, 10):
    im = cv2.imread(d["file_name"])
    outputs = predictor(im)
    v = Visualizer(im[:, :, ::-1],
                   metadata=plate_metadata,
                   scale=0.8,
                   instance_mode=ColorMode.IMAGE_BW
    )
    v = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    cv2_imshow(v.get_image()[:, :, ::-1])
```

Randomly select 10 samples to test.

Example 4-2：License Plate Detector

Result :

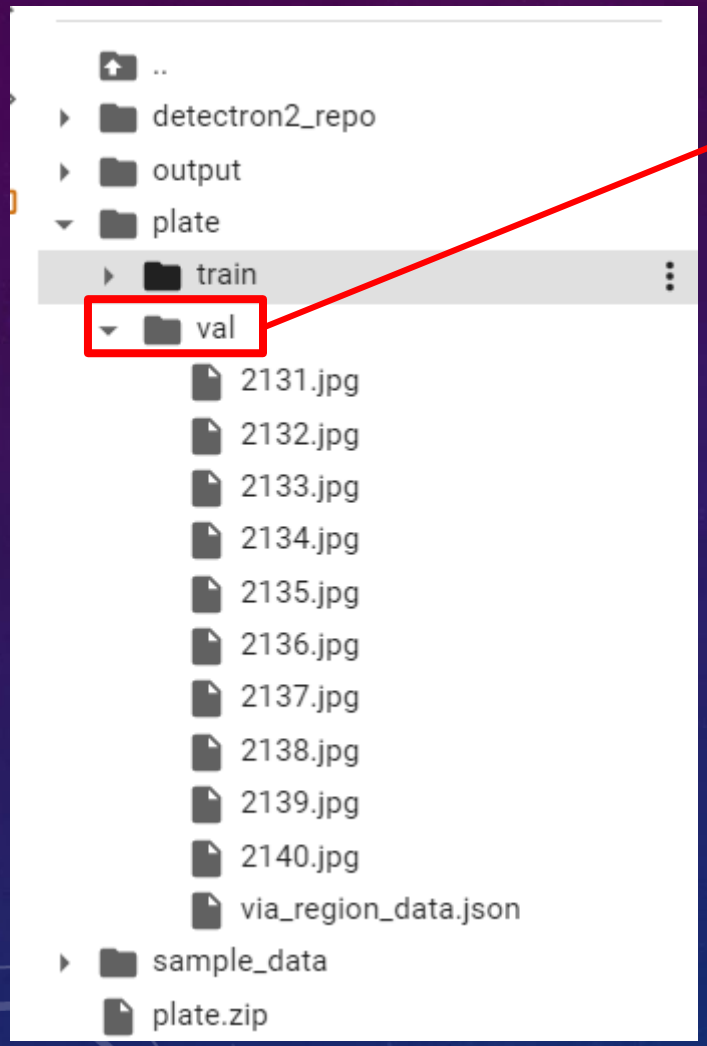
Exercise 4-2：License Plate Detector

- Please download “Detectron2_license_plate.ipynb” and “test_plate.zip” and open “Detectron2_license_plate.ipynb” by Colab.
- Use labelme tool to label the 10 images by yourself and test it on trained model.

  ※Hint: If you want to test, you need to upload your images and json file into plate/val folder.

- Please write down result and your code in MS Word to Moodle

Solution



Put your label img and via_region_data.json in val folder

Result: