# ECE 60146 HW4 Report

Zhengxin Jiang
(jiang839@purdue.edu)

## 1 Answers to The Questions

1. The adding of padding has a slight performance improvemance on both training loss and validation accuracy.

2. The net3 does training much slower than net1 and net2 on first 20 epochs, which could be something like vanishing gradient.

3. The net2 with padding performs best.

4. The classes of cat an dog are more difficult to classify, Since the two kinds of animals have many similar characteristics in pictures.

5. Maybe some combinations of tenser transforms in the Dataset class will improve performance.

## 2 Images From My Own Dataset

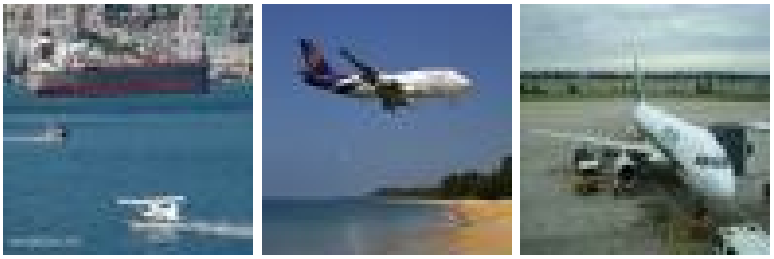


Figure 1: Three images from the class of airplane

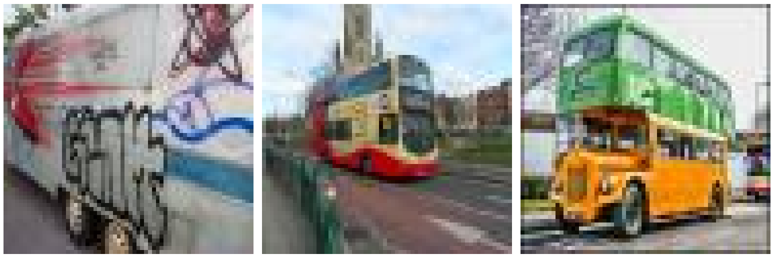


Figure 2: Three images from the class of bus

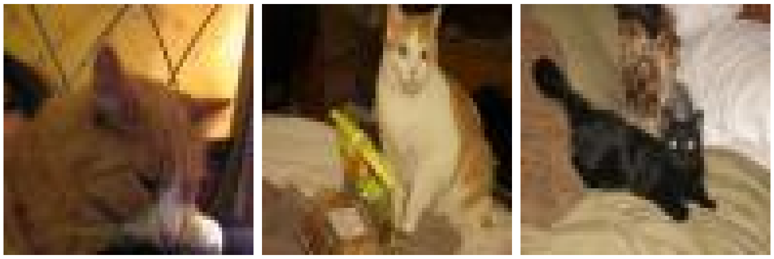


Figure 3: Three images from the class of cat

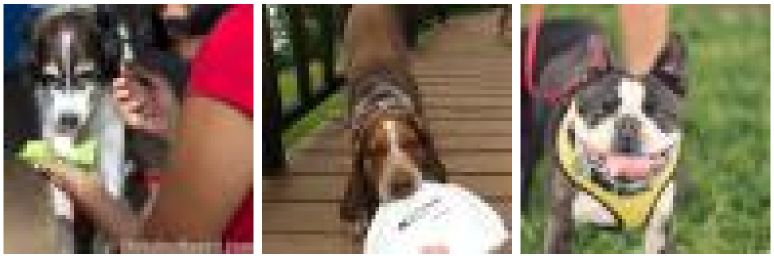


Figure 4: Three images from the class of dog

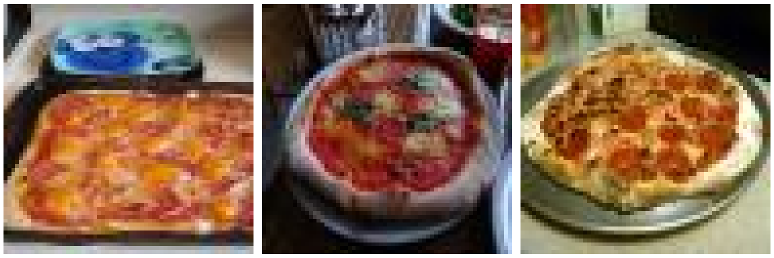


Figure 5: Three images from the class of pizza

# 3 Plots of The Training Loss

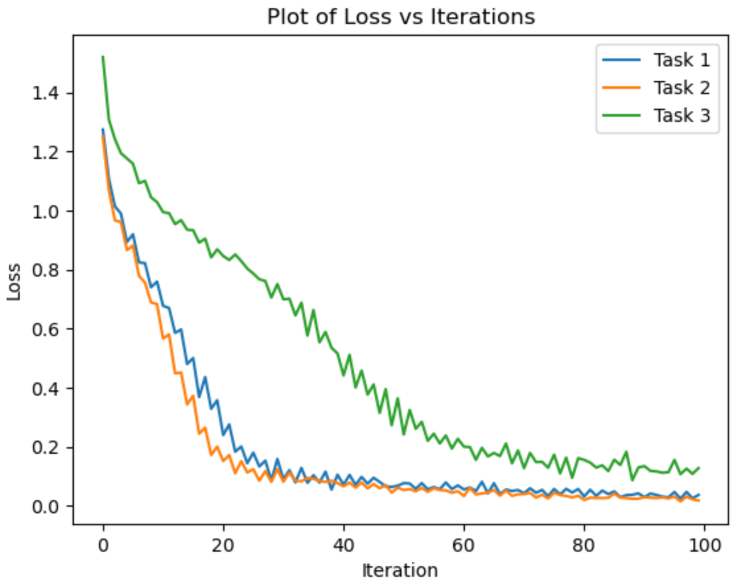
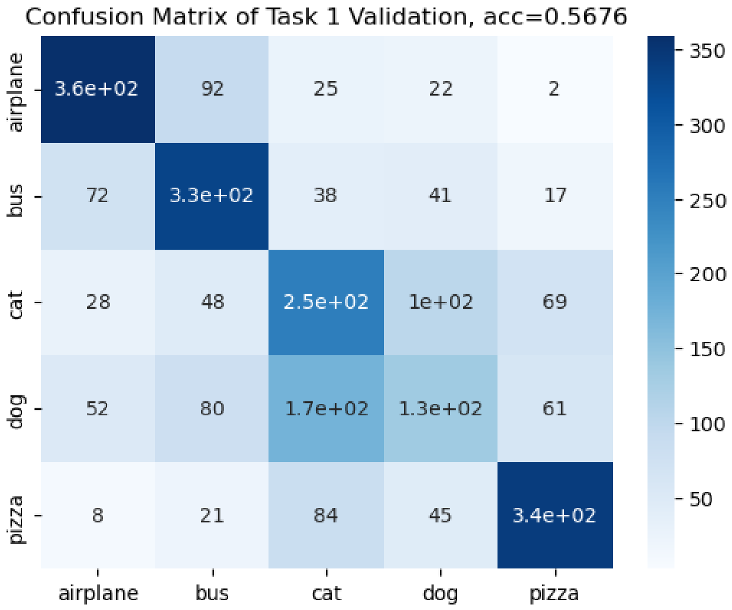


Figure 6: Plots of the training loss for all tasks

# 4 Confusion Matrices of Validation Results



| | airplane | bus | cat | dog | pizza |
|---|---|---|---|---|---|
| airplane | 3.6e+02 | 92 | 25 | 22 | 2 |
| bus | 72 | 3.3e+02 | 38 | 41 | 17 |
| cat | 28 | 48 | 2.5e+02 | 1e+02 | 69 |
| dog | 52 | 80 | 1.7e+02 | 1.3e+02 | 61 |
| pizza | 8 | 21 | 84 | 45 | 3.4e+02 |

Figure 7: The confusion matrix of task 1

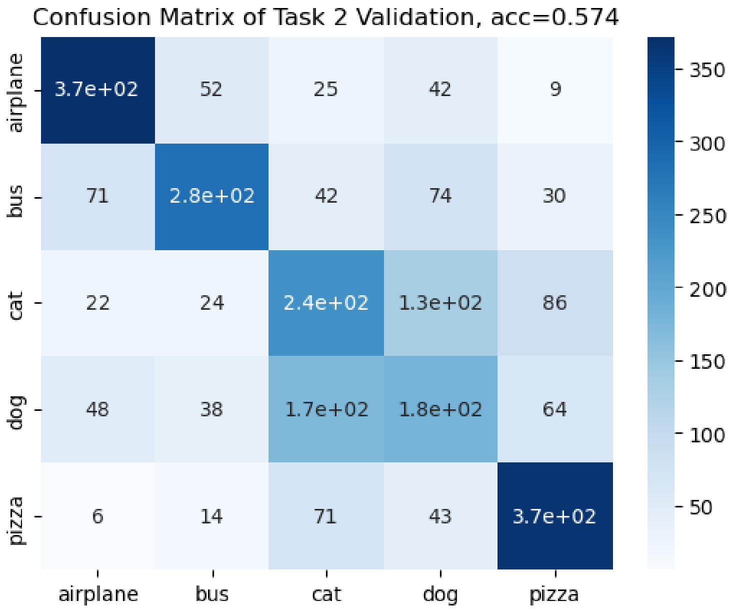

Confusion Matrix of Task 2 Validation, acc=0.574

Figure 8: The confusion matrix of task 2

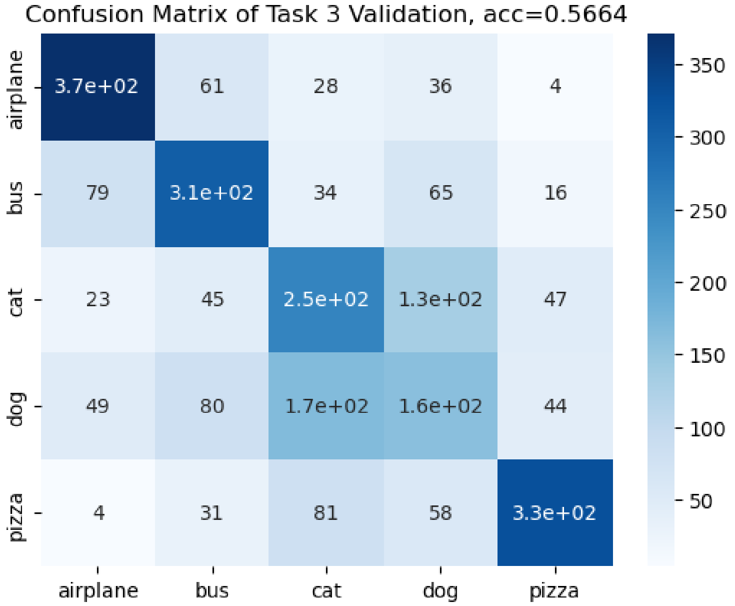

Confusion Matrix of Task 3 Validation, acc=0.5664

Figure 9: The confusion matrix of task 3

## 5 Source code

```
# ECE60146 HW4
# Zhengxin Jiang
# jiang839

import numpy as np
import os
import matplotlib.pyplot as plt
from PIL import Image
from pycocotools.coco import COCO
import seaborn as sn
import random

import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as tvt
from torch.utils.data import DataLoader

# torch.cuda.is_available()
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")


# Function for preparing the training data
def prepData(rawDataDir, hwDataDir):

    coco = COCO(’{}/annotations/instances_train2014.json’.format(rawDataDir))

    catIds = coco.getCatIds(catNms=[’airplane’,’bus’,’cat’,’dog’,’pizza’])

    for catCount,catId in enumerate(catIds):

        ImgIds = coco.getImgIds(catIds=catId)
        random.shuffle(ImgIds)

        for imgCount,imgId in enumerate(ImgIds):

            imgName = coco.loadImgs(imgId)[0][’file_name’]
            img = Image.open(rawDataDir+’/’+imgName)

            if img.mode != "RGB":
                img = img.convert(mode="RGB")
            img = img.resize((64, 64), Image.BOX)


            # Save training and validation images
            if imgCount<1500:

                imgNewName = str(catCount*1500+imgCount) + ’.jpg’
                fp = open(’{}/train/{}’.format(hwDataDir, imgNewName), ’w’)
                img.save(fp)
            elif imgCount<2000:

                imgNewName = str(catCount*500+imgCount-1500) + ’.jpg’
                fp = open(’{}/val/{}’.format(hwDataDir, imgNewName), ’w’)
                img.save(fp)

            else:
                break

    return

# The Dataset class for hw4
class hwDataset(torch.utils.data.Dataset):

    def __init__(self, root, tasktype):
        super().__init__()
        self.root = root
        self.tasktype = tasktype

    def __len__(self):
        if self.tasktype == ’training’:
            return 7500

        if self.tasktype == ’validation’:
            return 2500

    def __getitem__(self, index):
        name = str(index)+’.jpg’
        img = Image.open(os.path.join(self.root, name))

        tr = tvt.Compose([
            tvt.ToTensor(),
            tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
        ])

        img_tensor = tr(img)

        if self.tasktype == ’training’:
            return img_tensor, index//1500

        if self.tasktype == ’validation’:
            return img_tensor, index//500

# The network class
class HW4Net(nn.Module):
    def __init__(self, task):
        super(HW4Net, self).__init__()

        self.task = task

        # task 1
        if self.task == ’task1’:
            self.conv1 = nn.Conv2d(3, 16, 3)
            self.pool = nn.MaxPool2d(2, 2)
            self.conv2 = nn.Conv2d(16, 32, 3)
            self.fc1 = nn.Linear(32*14*14, 64)
            self.fc2 = nn.Linear(64, 5)

        # task 2
        if self.task == ’task2’:
            self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
            self.pool = nn.MaxPool2d(2, 2)
            self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
            self.fc1 = nn.Linear(32*16*16, 64)
            self.fc2 = nn.Linear(64, 5)

        # task3
        if self.task == ’task3’:
            self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
            self.pool = nn.MaxPool2d(2, 2)
            self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
            self.conv3 = nn.Conv2d(32, 32, 3, padding=1)
            self.fc1 = nn.Linear(32*16*16, 64)
            self.fc2 = nn.Linear(64, 5)


    def forward(self, x):

        # task 1 & 2
        if self.task == ’task1’ or self.task == ’task2’:
            x = self.pool(F.relu(self.conv1(x)))
            x = self.pool(F.relu(self.conv2(x)))
            x = x.view(x.shape[0], -1)
            x = F.relu(self.fc1(x))
            x = self.fc2(x)

        # task3
        if self.task == ’task3’:
            x = self.pool(F.relu(self.conv1(x)))
            x = self.pool(F.relu(self.conv2(x)))
            # 10 extra layers
            for i in range(10):
                x = F.relu(self.conv3(x))
            x = x.view(x.shape[0], -1)
            x = F.relu(self.fc1(x))
            x = self.fc2(x)

        return x

# Training function
def netTraining(saving_path, net, train_data_loader, epochs):

    net = net.to(device)
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(net.parameters(), lr=1e-3, betas=(0.9, 0.99))

    loss_list = []
    for epoch in range(epochs):
        running_loss = 0.0
        for i, data in enumerate(train_data_loader):
            inputs, labels = data
            inputs = inputs.to(device)
            labels = labels.to(device)
            optimizer.zero_grad()
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            if (i+1) % 100 == 0:
                print("[␣epoch␣:␣%d,␣batch␣:␣%5d]␣loss␣:␣%.3f" % (epoch + 1, i + 1,
                    running_loss / 100))
                loss_list.append(running_loss / 100)
                running_loss = 0.0

    # saving the learned parameters
    torch.save(net.state_dict(), saving_path)
    return loss_list


def validation(net, val_data_loader):

    cm = torch.zeros(5,5)
    true_count = 0

    # no grad for inference
    with torch.no_grad():
        for i, data in enumerate(val_data_loader):
            inputs, labels = data
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = net(inputs)

            # The predicted labels
            max_vals, predicted_labels = torch.max(outputs, 1)

            for i in range(len(labels)):
                cm[labels[i]][predicted_labels[i]] += 1
                if labels[i] == predicted_labels[i]:
                    true_count += 1

    return cm, true_count/2500


##### Main #####

rawDataDir = 'D:/coco/train2014'
hwDataDir = 'D:/coco/hw4'

prepData(rawDataDir, hwDataDir)

# Training
```

```
root = ’D:/coco/hw4/train’

traindataset = hwDataset(root, ’training’)
train_data_loader = DataLoader(traindataset, batch_size=32, num_workers=0, shuffle=True)

task = ’task3’
net = HW4Net(task)
epochs = 50
saving_path = task+’.pth’

loss = netTraining(saving_path, net, train_data_loader, epochs)

# # load trained parameters
# task = ’task1’
# net = HW4Net(task)
# net = net.to(device)

# net.load_state_dict(torch.load(task+’.pth’, map_location=torch.device(device)))

# Validation
root = ’D:/coco/hw4/val’
valdataset = hwDataset(root, ’validation’)
val_data_loader = DataLoader(valdataset, batch_size=32, num_workers=0, shuffle=True)

confusion_matrix, acc = validation(net, val_data_loader)
print(acc)

plt.figure()
plt.title("Confusion␣Matrix␣of␣Task␣3␣Validation,␣acc="+str(acc))
sn.heatmap(confusion_matrix, annot=True, cmap="Blues",xticklabels=[’airplane’,’bus’,’cat’
    ,’dog’,’pizza’], yticklabels=[’airplane’,’bus’,’cat’,’dog’,’pizza’])
```