hw4_ChinHongKennethLee

February 10, 2025

Name: Chin Hong Kenneth Lee

ECE 60146

0.0.1 Task 3.2: We select 5 different classes from each constraints specified on Piazza with at least 500 images.

I use the given pesudocode in the assignment and modify the `else` statement logic to enforce one additional logical statement: `len(obj_counts) <= len(cat_names)` to ensure there are only specified categories in the images across all dataset variations. This will help prevent getting multiple categories in the images for the single instance case. Also, I set to `break` in that function when it reaches 500 images, which is the least amount of images I need to get for the assignment. I first explore COCO API to see what categories are available. Then, I count the number of images per category that fits the requirmeents.

For the single-instance dataset, there are total of 6 categories that have at least 500 images. I selected the following categories: 'airplane', 'train', 'bird', 'giraffe', 'toilet'. This is done by setting `min_instances=1, max_instances=1,dataset_type="single_instance", multiple_categories=False` in the function `extract_images`. For the same object with multiple instances, we set the arguments `min_instances=2, max_instances=10,dataset_type="multi_instance_same", multiple_categories=False` for the function `extract_images`. Only 'zebra' and 'giraffe' have at least 500 images. I used these images along with 3 additional categories: cow, sheep, and elephant. Since cow, sheep, and elephant do not have more than 500 images, I augmented each of these categories with an additional of 58, 87, 32 images respectively. I took the code I had from HW2 to carry out the augmentation task. The augmentation steps are horizontal flip, random rotation and normalization. For the multiple-instance (different objects) case, I collected the following 5 classes: 'dog', 'vase', 'suitcase', 'fire hydrant', 'refrigerator'. I modified the argument in `extract_images` to `min_instances=2, max_instances=10, multiple_categories=True`in this case. There are at least 500 images for each of these classes. Thus, there is no need for augmentation.

[79]:
```python
from pycocotools.coco import COCO
import os
import shutil
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
```

```python
# Set COCO dataset paths
data_dir = "coco_dataset"
ann_file = os.path.join(data_dir , "annotations/instances_train2014.json")
image_dir = os.path.join(data_dir , "train2014")
output_dir = "output_datasets"

# Load COCO dataset
coco = COCO(ann_file)

# Ensure output directories exist
os.makedirs(output_dir , exist_ok=True)

def save_image(img_info , category_name , dataset_type):
    """ Saves the extracted image into a structured output directory."""
    img_path = os.path.join(image_dir , img_info['file_name'])
    save_dir = os.path.join(output_dir , dataset_type,
                            category_name)
    os.makedirs(save_dir , exist_ok=True)
    # Load and resize image
    img = Image.open(img_path).resize ((64 , 64))
    img.save(os.path.join(save_dir , img_info['file_name']))

def extract_images(cat_names, min_instances=1, max_instances=1,␣
 ↪multiple_categories=False, dataset_type="single_instance", saveImage=False,␣
 ↪num_image_limit=500,  printAllImagesNum=False):
    # retrieve category IDs for the specified object categories
    cat_ids = coco.getCatIds(catNms=cat_names)
    # get images that contains the category
    img_ids = coco.getImgIds(catIds=cat_ids)
    extracted = 0
    for img_id in img_ids:
        img_info = coco.loadImgs(img_id)[0]
        ann_ids = coco.getAnnIds(imgIds=img_id, iscrowd=False)
        anns = coco.loadAnns(ann_ids)

        obj_counts = {}
        for ann in anns:
            obj_category = coco.loadCats(ann['category_id'])[0]['name']
            obj_counts[obj_category] = obj_counts.get(obj_category, 0) + 1

        if multiple_categories:
            if all(obj in obj_counts for obj in cat_names) and len(obj_counts)␣
 ↪>= 2:
                if saveImage:
                    save_image(img_info, cat_names[0], dataset_type)
                extracted += 1
        else:
            if all(obj in obj_counts for obj in cat_names) and len(obj_counts)␣
↪<= len(cat_names) and min_instances <= obj_counts[cat_names[0]] <=␣
↪max_instances:
                if saveImage:
                    save_image(img_info, cat_names[0], dataset_type)
                extracted += 1

        if extracted >= num_image_limit:  # Limit for quick testing
            print(f"{cat_names[0]} has {num_image_limit} images")
            break
    if printAllImagesNum:
        print(f'{cat_names[0]}, count:{extracted}')
```

```
loading annotations into memory…
Done (t=5.66s)
creating index…
index created!
```

[9]:
```
##### The following code is from COCO API demo notebook ###############
from pycocotools.coco import COCO
import os
import shutil
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
dataDir='coco_dataset'
dataType='train2014'
annFile='{}/annotations/instances_{}.json'.format(dataDir,dataType)
# initialize COCO api for instance annotations
coco=COCO(annFile)
# display COCO categories and supercategories
cats = coco.loadCats(coco.getCatIds())
nms=[cat['name'] for cat in cats]
print('COCO categories: \n{}\n'.format(' '.join(nms)))
```

```
loading annotations into memory…
Done (t=5.18s)
creating index…
index created!
COCO categories:
person bicycle car motorcycle airplane bus train truck boat traffic light fire
hydrant stop sign parking meter bench bird cat dog horse sheep cow elephant bear
zebra giraffe backpack umbrella handbag tie suitcase frisbee skis snowboard
sports ball kite baseball bat baseball glove skateboard surfboard tennis racket
bottle wine glass cup fork knife spoon bowl banana apple sandwich orange
broccoli carrot hot dog pizza donut cake chair couch potted plant bed dining
table toilet tv laptop mouse remote keyboard cell phone microwave oven toaster
sink refrigerator book clock vase scissors teddy bear hair drier toothbrush
```

[38]:
```
# Extract single instance whose category has at least 500 images
for k in nms:
    extract_images([k], min_instances=1,␣
 ↪max_instances=1,dataset_type="single_instance",saveImage=False)
```

```
airplane has 500 images
train has 500 images
bird has 500 images
giraffe has 500 images
toilet has 500 images
clock has 500 images
```

[80]:
```
# Extract single instance whose category has at least 500 images
for k in ['airplane', 'train', 'bird', 'giraffe', 'toilet']:
    extract_images([k], min_instances=1,␣
 ↪max_instances=1,dataset_type="single_instance",saveImage=True)
```

```
airplane has 500 images
train has 500 images
bird has 500 images
giraffe has 500 images
toilet has 500 images
```

[44]:
```
# Extract multiple instances for 500 images
for k in nms:
    extract_images([k], min_instances=2,␣
 ↪max_instances=10,dataset_type="multi_instance_same", saveImage=False,␣
 ↪num_image_limit=500)
```

```
zebra has 500 images
giraffe has 500 images
```

[45]:
```
# Extract multiple instances for 500 images
for k in ['zebra', 'giraffe']:
    extract_images([k], min_instances=2,␣
 ↪max_instances=10,dataset_type="multi_instance_same", saveImage=True,␣
 ↪num_image_limit=500)
```

```
zebra has 500 images
giraffe has 500 images
```

[43]:
```
# Extract multiple instances
for k in nms:
    extract_images([k], min_instances=2,␣
 ↪max_instances=10,dataset_type="multi_instance_same", saveImage=False,␣
 ↪num_image_limit=400)
```

```
sheep has 400 images
```

```
cow has 400 images
elephant has 400 images
zebra has 400 images
giraffe has 400 images
```

[46]:
```
# Extract multiple instances
for k in ['cow','sheep', 'elephant']:
    extract_images([k], min_instances=2,␣
 ↪max_instances=10,dataset_type="multi_instance_same", saveImage=True,␣
 ↪num_image_limit=500,printAllImagesNum=True)
```

```
cow, count:442
sheep, count:413
elephant, count:468
```

[49]:
```
# we need to augment cow, sheep, and elephant images
# cow needs 58 augmented images
# sheep needs 87 augmented images
# elephant needs 32 augmented images
import torch
from torchvision import datasets , transforms
from torch.utils.data import DataLoader , Subset
import os
from PIL import Image

# Step 2: Create a custom dataset
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self , root , transform=None):
        self.root = root
        self.image_paths = [os.path.join(root , img) for img in os.
 ↪listdir(root)]
        self.transform = transform
    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self , index):
        img_path = self.image_paths[index]
        image = Image.open(img_path).convert("RGB")
        if self.transform:
            image = self.transform(image)
        return image , 0 # Assuming no class labels for simplicity

# Load 20 custom images and apply augmentation
transform_custom = transforms.Compose([
    # fill code here
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

custom_dataset = CustomDataset(root='output_datasets/multi_instance_same/cow', ␣
↪transform=transform_custom)

# augment 58 images
augmented_images = [custom_dataset[i % len(custom_dataset)][0] for i in␣
↪range(58)]
output_augmented_dir = 'output_datasets/multi_instance_same/cow'
os.makedirs(output_augmented_dir, exist_ok=True)  # Create directory if it␣
↪doesn't exist

# save the images to the directory
for i, img_tensor in enumerate(augmented_images):
    # # Convert tensor to PIL image
    img = transforms.ToPILImage()(img_tensor)
    img.save(os.path.join(output_augmented_dir, f'augmented_image_{i}.png'))  #␣
↪Save each image
```

[50]:
```
curr_dir = 'output_datasets/multi_instance_same/sheep'
custom_dataset = CustomDataset(root=curr_dir ,  transform=transform_custom)
# augment 87 images
augmented_images = [custom_dataset[i % len(custom_dataset)][0] for i in␣
↪range(87)]
output_augmented_dir = curr_dir
os.makedirs(output_augmented_dir, exist_ok=True)  # Create directory if it␣
↪doesn't exist

# save the images to the directory
for i, img_tensor in enumerate(augmented_images):
    # # Convert tensor to PIL image
    img = transforms.ToPILImage()(img_tensor)
    img.save(os.path.join(output_augmented_dir, f'augmented_image_{i}.png'))  #␣
↪Save each image
```

[51]:
```
curr_dir = 'output_datasets/multi_instance_same/elephant'
custom_dataset = CustomDataset(root=curr_dir ,  transform=transform_custom)
# augment 87 images
augmented_images = [custom_dataset[i % len(custom_dataset)][0] for i in␣
↪range(32)]
output_augmented_dir = curr_dir
os.makedirs(output_augmented_dir, exist_ok=True)  # Create directory if it␣
↪doesn't exist

# save the images to the directory
```

```
for i, img_tensor in enumerate(augmented_images):
    # # Convert tensor to PIL image
    img = transforms.ToPILImage()(img_tensor)
    img.save(os.path.join(output_augmented_dir, f'augmented_image_{i}.png'))  #␣
 ↪Save each image
```

```
[65]: for i in ['dog', 'vase', 'suitcase', 'fire hydrant', 'refrigerator']:
          extract_images([i], multiple_categories=True,␣
       ↪dataset_type="multi_instance_different", saveImage=True)
```

```
dog has 500 images
vase has 500 images
suitcase has 500 images
fire hydrant has 500 images
refrigerator has 500 images
```

0.0.2 Data visualization: at least three images per class

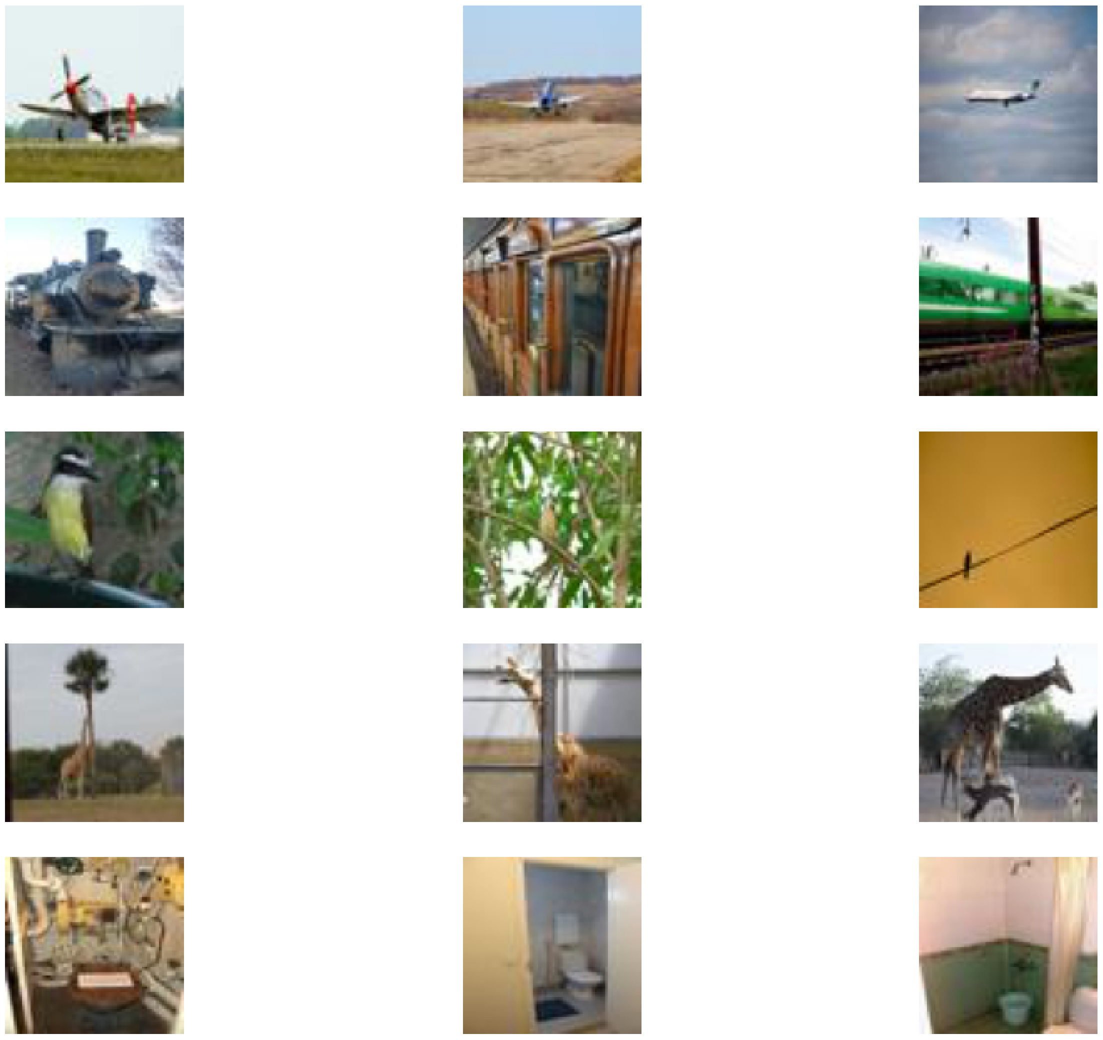
Here, I first loop through the specified categories that I found from the previous section for each dataset type. Then, I randomly pick 3 images from the directory and then plot all them in a 5x3 table where each row is a category and each column represents an image in the category. I reference COCO API to use `skimage` package to read the image from the directory and use `matplotlib.pyplot` to visualize these images

```
[87]: import skimage.io as io
      import random
      import os

      def split_data_and_show_train_data(dataset_type, ls_of_categories):
          """
              a helper function to split the data into train and test set and show 3␣
       ↪images per classs for
              the training dataset in the specified data_type
          """
          fig, axs = plt.subplots(5, 3, figsize=(10, 8))
          for rowidx, cat in enumerate(ls_of_categories):
              # chnage the path
              image_dir = f'output_datasets/{dataset_type}/{cat}/'
              image_files = os.listdir(image_dir)
              train_dir = os.path.join(image_dir, 'train')
              test_dir = os.path.join(image_dir, 'test')
              os.makedirs(train_dir, exist_ok=True)
              os.makedirs(test_dir, exist_ok=True)
              selected_train_images = random.sample(image_files, 400)
              # move the images
              for img_file in selected_train_images:
                  shutil.move(os.path.join(image_dir, img_file), os.path.
       ↪join(train_dir, img_file))
```

```
        # move the rest to test set
        remaining_images = set(image_files) - set(selected_train_images)
        for img_file in remaining_images:
            shutil.move(os.path.join(image_dir, img_file), os.path.
 ↪join(test_dir, img_file))

        # randomly pick three images
        selected_images = random.sample(os.listdir(train_dir), 3)
        for colidx, img_file in enumerate(selected_images):
            # plot the images
            I = io.imread(os.path.join(train_dir, img_file))
            axs[rowidx, colidx].axis('off')
            axs[rowidx, colidx].imshow(I)
    plt.show()
```

Single-instance (Training datasets)

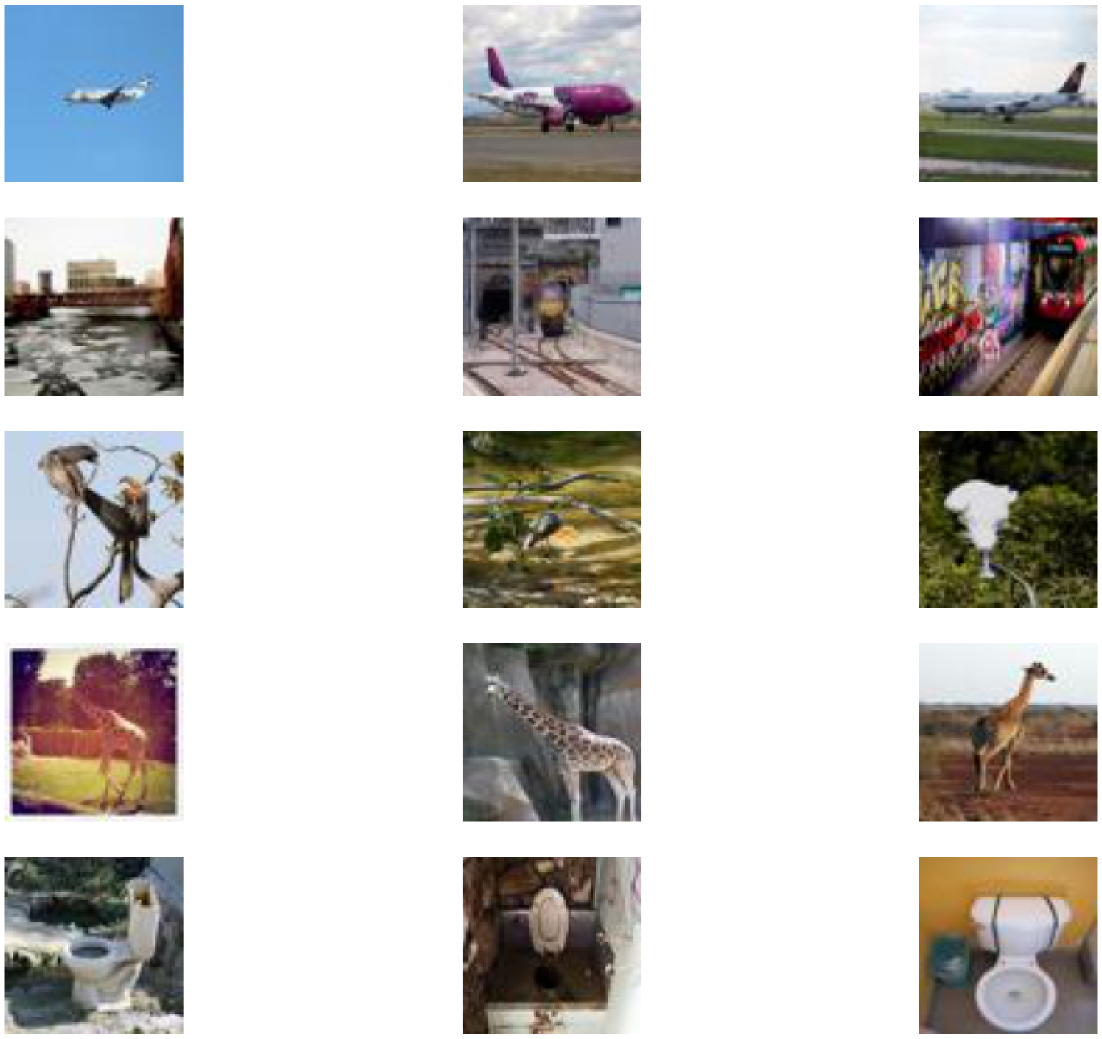
[81]:
```
import skimage.io as io
import random
import os
split_data_and_show_train_data('single_instance', ['airplane', 'train', 'bird',␣
 ↪'giraffe', 'toilet'])
```



Single-instance (Validation dataset)

```
[83]: import skimage.io as io
      import random
      fig, axs = plt.subplots(5, 3, figsize=(10, 8))
      dataset_type = 'single_instance'

      for rowidx, cat in enumerate(['airplane', 'train', 'bird', 'giraffe',␣
        ↪'toilet']):
          # chnage the path
          image_dir = f'output_datasets/{dataset_type}/{cat}/'
          test_dir = os.path.join(image_dir, 'test')
          image_files = os.listdir(test_dir)
          # randomly pick three images
          selected_images = random.sample(image_files, 3)
```

```
    for colidx, img_file in enumerate(selected_images):
        # plot the images
        I = io.imread(os.path.join(test_dir, img_file))
        axs[rowidx, colidx].axis('off')
        axs[rowidx, colidx].imshow(I)
plt.show()
```



Multiple instances (same objects) (Training data)

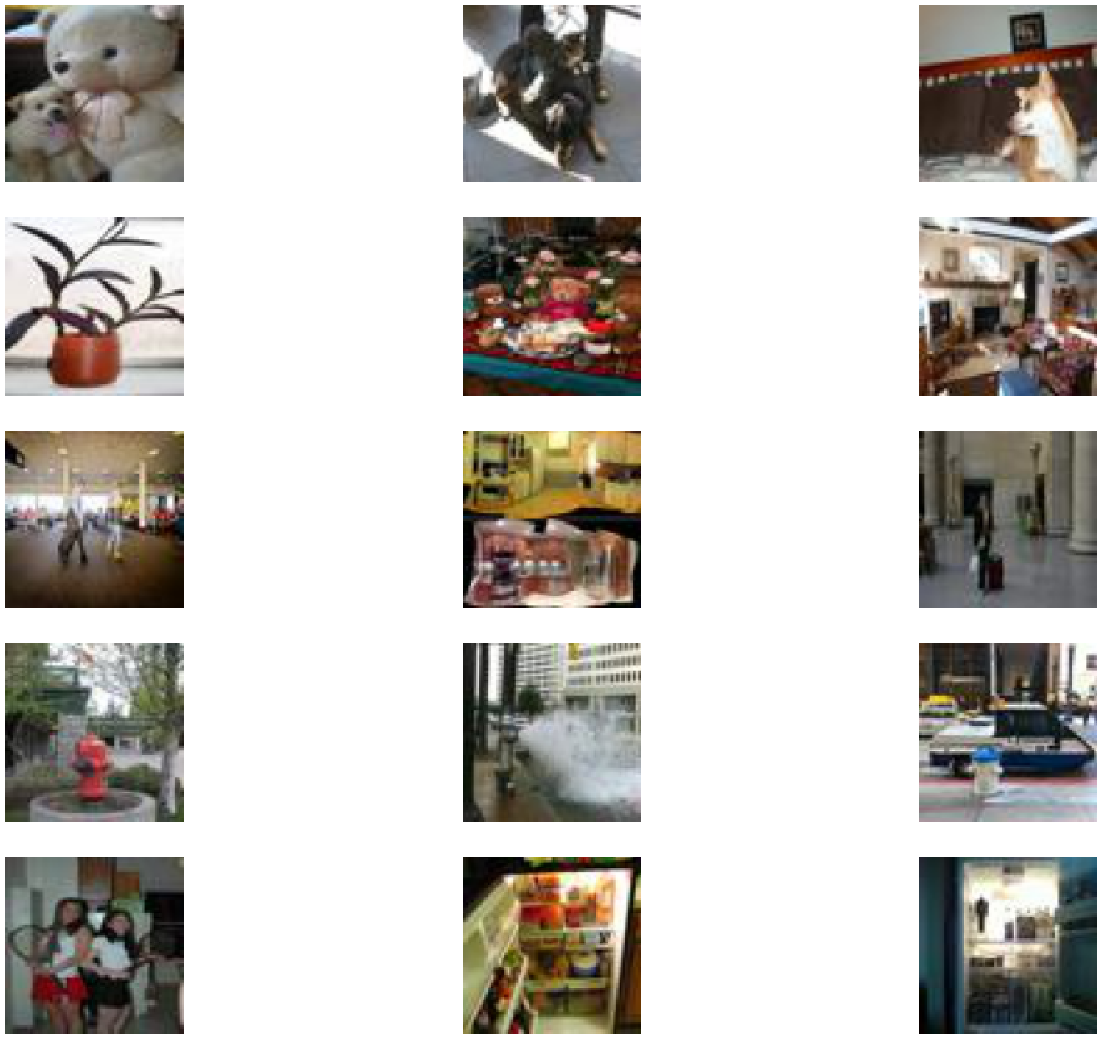
```
[84]: dataset_type = 'multi_instance_same'
      split_data_and_show_train_data(dataset_type, ['zebra', 'giraffe','cow','sheep',␣
       ↪'elephant'])
```

Multiple instances (same objects) (Validation data)

[85]:
```
import skimage.io as io
import random
fig, axs = plt.subplots(5, 3, figsize=(10, 8))
dataset_type = 'multi_instance_same'

for rowidx, cat in enumerate(['zebra', 'giraffe','cow','sheep', 'elephant']):
    # chnage the path
    image_dir = f'output_datasets/{dataset_type}/{cat}/'
    test_dir = os.path.join(image_dir, 'test')
    image_files = os.listdir(test_dir)
    # randomly pick three images
    selected_images = random.sample(image_files, 3)
    for colidx, img_file in enumerate(selected_images):
        # plot the images
        I = io.imread(os.path.join(test_dir, img_file))
        axs[rowidx, colidx].axis('off')
        axs[rowidx, colidx].imshow(I)
plt.show()
```

Multiple instances (different objects) (Training data)

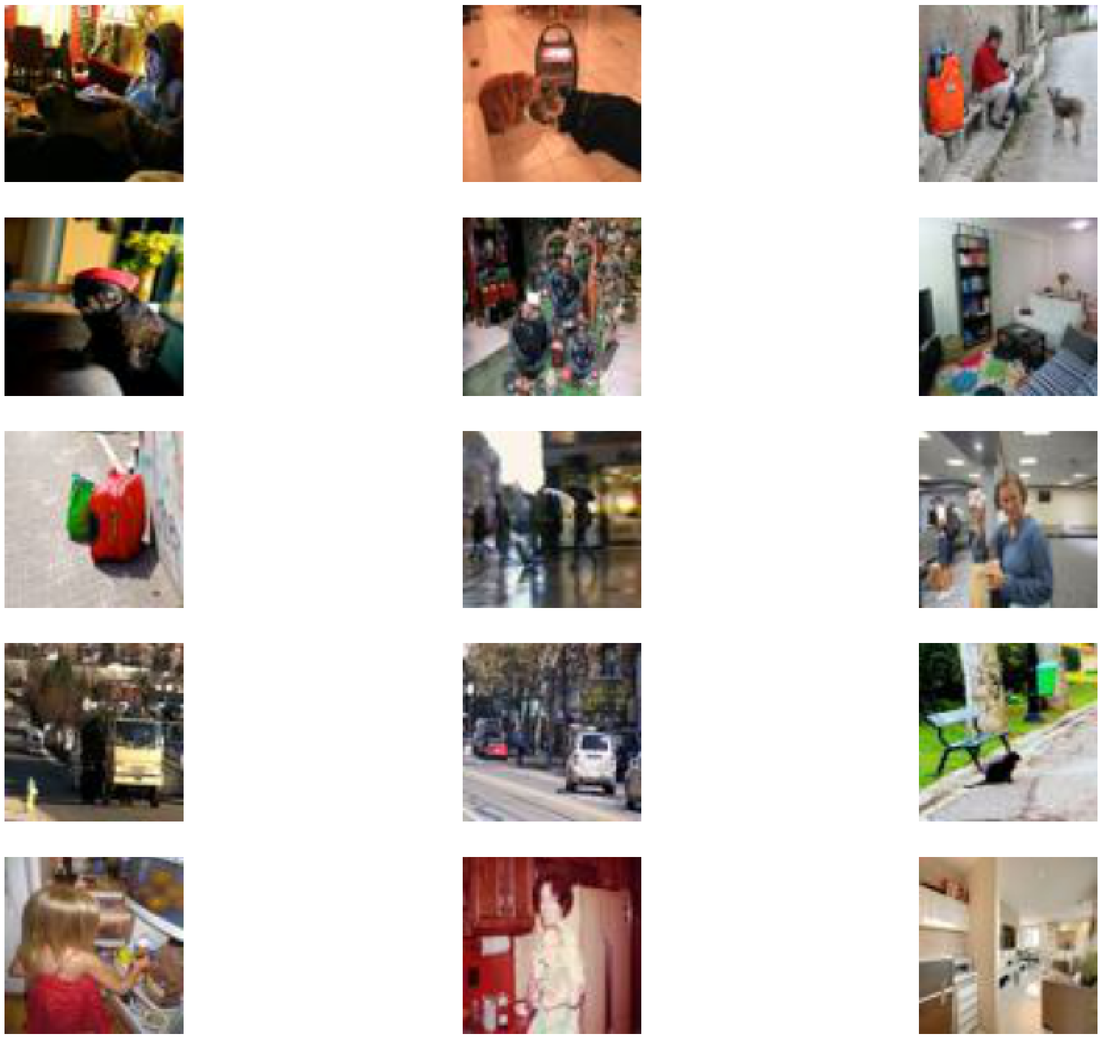
[88]:
```
dataset_type = 'multi_instance_different'
split_data_and_show_train_data(dataset_type, ['dog', 'vase', 'suitcase', 'fire␣
↪hydrant', 'refrigerator'])
```



Multiple instances (different objects) (Validation data)

```
[89]: import skimage.io as io
      import random
      fig, axs = plt.subplots(5, 3, figsize=(10, 8))
      dataset_type = 'multi_instance_different'

      for rowidx, cat in enumerate(['dog', 'vase', 'suitcase', 'fire hydrant',␣
       ↪'refrigerator']):
          # chnage the path
          image_dir = f'output_datasets/{dataset_type}/{cat}/'
          test_dir = os.path.join(image_dir, 'test')
          image_files = os.listdir(test_dir)
          # randomly pick three images
          selected_images = random.sample(image_files, 3)
```

```
    for colidx, img_file in enumerate(selected_images):
        # plot the images
        I = io.imread(os.path.join(test_dir, img_file))
        axs[rowidx, colidx].axis('off')
        axs[rowidx, colidx].imshow(I)
plt.show()
```



0.0.3 Implement and Evaluate a CNN model

Since the task is 5-class classification problem, we set `self.fc2 = nn.Linear(64,X)` to `self.fc2 = nn.Linear(64,5)`. The input size to `HW4Net` has 3 color channels and it's 64 by 64 images as instructed by the assignment. We can compute the XXXX for `self.fc1 = nn.Linear(XXXX,64)` as follows: given the first layer is a convolutional layer specified by `nn.Conv2d(3, 16, 3)`, it implies its input channel is 3, the output channel is 16 and uses a 3x3 kernel (K). The default for `stride (S)` parameter is 1 and `padding (P)` parameter is 0. I reference Pytorch documentaiton:(https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#torch.nn.Conv2d), which says given an input (C_{in}, H_{in}, W_{in}), where C_{in} is the number of input channels, H_{in} is the height of the input, W_{in} is the width of the input. The output can be computed using the following formula:

$$H_{out} = \frac{H_{in} + 2 * P - D(K-1) - 1}{S} + 1 = \frac{64 + 2 * 0 - 1(3-1) - 1}{1} + 1 = 62,$$

and

$$W_{out} = \frac{W_{in} + 2 * P - D(K-1) - 1}{S} + 1 = \frac{64 + 2 * 0 - 1(3-1) - 1}{1} + 1 = 62$$

Therefore, the output of `nn.Conv2d(3, 16, 3)` has size of 16 x 62 x 62.

Next, the layer `nn.MaxPool2d(2, 2)` is used via `self.pool()`, the formula for the output (C, H_{out}, W_{out}) can also be referened from PyTorch documantion: (https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html#torch.nn.MaxPool2d). As the parameter input to `nn.MaxPool2d(2, 2)` gives the kernel size 2, the output size is: 16 x $\frac{62}{2}$ x $\frac{62}{2}$ = 16 x 31 x 31.

For the second convolutional layer `nn.Conv2d(16, 32, 3)`, we repeat the same procedure as before using the formula with the input being of size 16 x 31 x 31 such that $H_{in} = 31, W_{in} = 31$ such that H_{out} and W_{out} are as follows:

$$H_{out} = \frac{31 + 2 * P - D(K-1) - 1}{S} + 1 = \frac{31 + 2 * 0 - 1(3-1) - 1}{1} + 1 = 29,$$

and

$$W_{out} = \frac{31 + 2 * P - D(K-1) - 1}{S} + 1 = \frac{31 + 2 * 0 - 1(3-1) - 1}{1} + 1 = 29$$

Since the kernel size is 3 and the size of output channel to be 32, the output of `nn.Conv2d(16, 32, 3)` has size of 32 x 29 x 29. Then, the layer `nn.MaxPool2d(2, 2)` is again used via `self.pool()`, so, we will have the output size to be 32 x 29/2 x 29/2 = 32 x 14 x 14 = 6272, which is the input size to the linear layer `nn.Linear(XXXX,64)` as we flatten the input via `view()`.

[98]:
```
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader

class HW4Net(nn.Module):
    def __init__(self):
        super(HW4Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, 3)

        # fill XXXX and XX
        self.fc1 = nn.Linear(6272,64)
        self.fc2 = nn.Linear(64,5) # we set to 5 because there are only 5␣
↪classes

    def forward (self, x):
        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)
        return x
```

I wrote several helper functions for reporting:

- `plot_learning_curve_results`: it is used to plotting the loss curve for all three different data configurations. I modified a function from my hw2 assignment to create this function
- `plot_confusion_matrix`: this takes a list of true labels and a list of predicted labels and the corresponding label names from the image categories. It uses scikit-learn library to compute the confusion matrix as hinted by the assignment. It then uses seaborn to visualize the confusion matrix
- `get_param_num`: I reference a PyTorch (https://pytorch.org/docs/stable/generated/torch.nn.parameter.Para and https://pytorch.org/docs/stable/generated/torch.numel.html#torch.numel) to know how to get the number of model parameters and this function will return the total number of model parameters from the given neural network model. The numel() mainly counts the number of learning parameters.
- `get_data_loader`: this function calls on a modified class `CustomDataset` that I modified from HW2. `CustomDataset` class takes inputs e.g. dataset_type, category_names, train_or_test,transform, to determine which dataset we want to create, whether it is a single-instance training set or multi-instance test set. It will look up the directory to find the images and tranform the images when they get iterated by padding the directory with various specified data types. It creates both the training data loader and validation data loader for training the model.
- `train_model`: this is modified from the given code in the assignment. I added a list called `losses` to keep track of the training loss and return the loss for plotting purposes. I changed the `% 100` to `% 63` since I am using `batch_size=32`, which gives me only 63 batches for a total of 2000 training images from 5 classes.
- `test_model`: this functio used to evaluate the given model with the given validation data loader. In order to avoid having PyTorch to store computational graphs in memory for every operation. I use `torch.no_grad()` to skip storing these graphs to reduce memory consumption. It then traverse every test image from the data loader and pass it to the model for getting the predicted label. It returns two lists: a list of true labels and a list of predicted labels. The `outputs` is the raw output logits from the model (HW4Net), which has shape (batch_size, num_classes). The line `torch.max(outputs, 1)` computes the maximum value along dimension 1 (class dimension). The first output (which we ignore with _) contains the actual max values (not needed for classification). The second output (preds) contains the indices of the maximum values, which represent the predicted class labels.

To get the result, I first use `torch.device("cuda" if torch.cuda.is_available() else "cpu")` to assign GPU if available or CPU if there is not GPU that PyTorch should use for model training. Then, I get three different instances from the class `HW4Net` for training them separately.

Then, I get the results using the helper functions. Below is the table summarzing parameters counts and classification accuracy. I use sckit-learn library `accuracy_score` function to get the average accuracy based on the predicted and true labels across classes. Below I will report the accuracy per class with respect to each dataset.

For single-instance dataset: class 1 is airplane, class 2 is train, class 3 is bird, class 4 is giraffe, class 5 is toilet

For multi-instance (same) dataset: class 1 is zebra, class 2 is giraffe, class 3 is cow, class 4 is sheep, class 5 is elephant

For multi-instance (different) dataset: class 1 is dog, class 2 is vase, class 3 is suitcase, class 4 is fire hydrant, class 5 is regrigerator

Configuration	Num of parameters	Accuracy Class 1	Accuracy Class 2	Accuracy Class 3	Accuracy Class 4	Accuracy Class 5
Single-instane	406885	0.78	0.73	0.31	0.51	0.81
Multi-instance(same)	406885	0.54	0.50	0.37	0.34	0.65
Multi-instance(different)	406885	0.26	0.33	0.46	0.47	0.31

We report 5 misclassified images from the first batch of test images for each dataset configuration. For the single-instance dataset, I check the misclassified label for the first image, the model actually predicts it to be toliet, which in my opinion, is quite interesting because the airplane does look similar to a toliet when the airplane is far away on the ground. For the multi-instance (same) case, the model often predicts a zebra to be giraffle, I think this is because when the image is zoomed out, the body shapes of the zebras do look quite similar to giraffes. For the multi-instance (different) dataset, I also find that the model predicts a dog to be a suitcase. It makes sense because when the dog lying on the bed with the same plain color does look like a suitcase.

By observing your classification accuracies, which dataset think are more difficult to correctly differentiate and why?

- It's clear that as the dataset has more different objects, it's harder to correctly differentiate. It is because other objects that appear on the same image may also appear in other images that have been labelled differently. This is an issue with spurious correlation. In short, the more objects appear in a single image, it's easier to overlap with other images that have a different label, making it difficult to correctly distinguish the correct class label.

By observing your confusion matrices, which class or classes do you think are more difficult to correctly differentiate and why?

- For the single instance case, the hardest class to detect is birds. One reason could be that birds can appear very small in the images as they may fly high in the sky. This makes the model to confuse birds with other objects. Another reason I think it's that the bird class has the highest number of augmented graphs due to insufficient images (e.g. <500 images). This poses challenges to the model as the data quality is low.

- For the multi-instance (same) case, it appears that giraffe is the hardest to classify. Based on the confusion matrix, the model often detects it to be zebra. This is possible because these two animals may often appear together in an image. Although there may be multiple giraffes, the fact that both zebras and giraffes live in the same area make these two classes look very similar. A similar pattern also apply to the class 'cow' and 'sheep'.
- For the multi-instance (different): this dataset also a similar issue as in the multi-instance (same) case, the object 'vase' appears to have low accuracy according to the confusion matrix. It often gets labels to be refrigrator. This is not surprising as vase and regrigrator are often place at home or near to the kitchen area in a home. Allowing multiple different objects to appear in a single images amplify the issue of supurious correlation between labels and images.

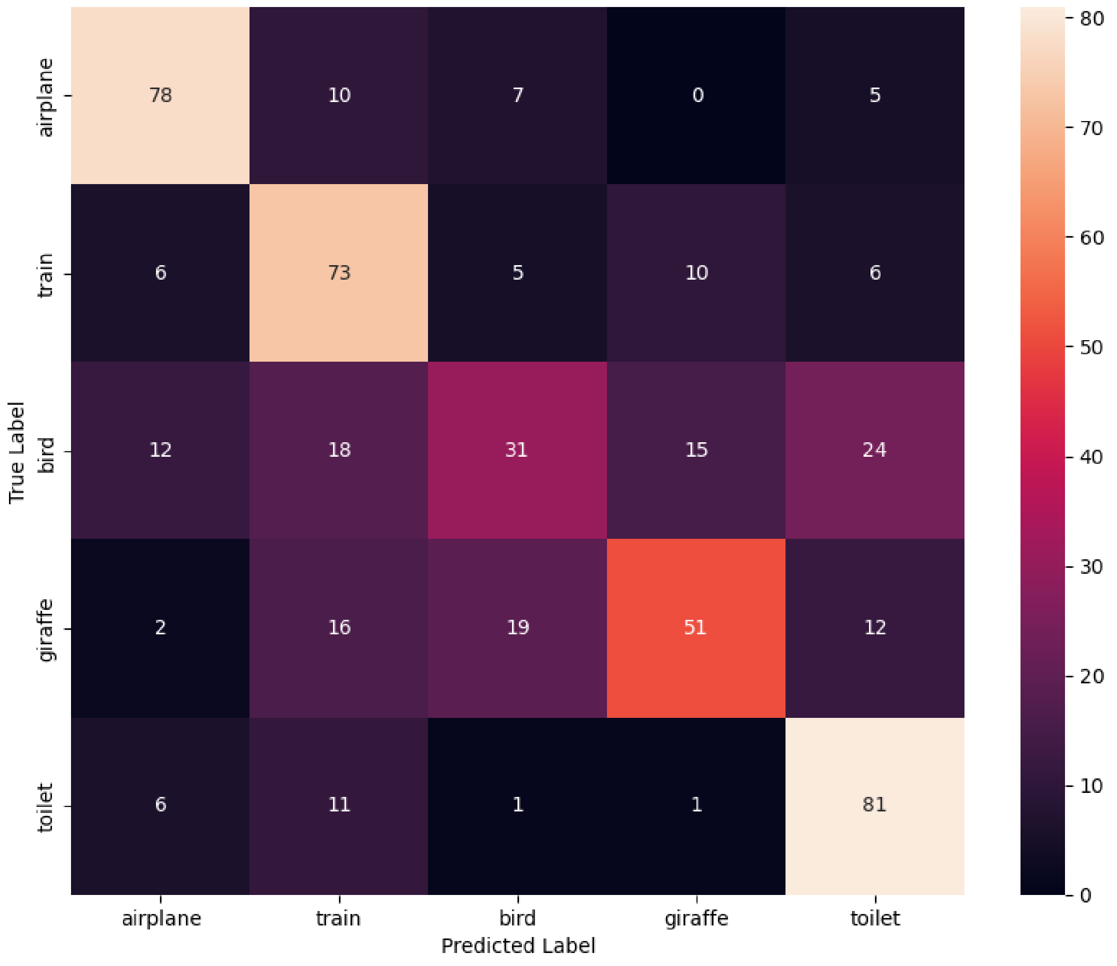
What is one thing that you propose to make the classification performance better?

- I would try not to resize the images to be 64 by 64. This compression can lose information from the original image, compromising the model performance.

[151]:
```python
# load the data and set seed
import seaborn as sns
from sklearn.metrics import confusion_matrix,  accuracy_score



seed = 60146
random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
np.random.seed(seed)
torch.backends.cudnn.deterministic=True
torch.backends.cudnn.benchmarks=False
os.environ['PYTHONHASHSEED'] = str(seed)



def plot_learning_curve_results(train_loss_d1, train_loss_d2, train_loss_d3):
    # taken from hw2 and modify it for plotting three different configuration␣
 ↪for the same network
    plt.figure(figsize=(10,8))
    plt.plot(train_loss_d1, label='Single-instance')
    plt.plot(train_loss_d2, label='Multiple-instance(same)')
    plt.plot(train_loss_d3, label='Multiple-instance(different)')
    plt.ylabel('Loss')
    plt.xlabel('Iteration')
    plt.title(f'Training Loss by Iteration for HW4Net')
    plt.legend()
    plt.show()

def plot_confusion_matrix(true, pred, category_names):
    # use sklearn package to get the confusion matrix
    cm = confusion_matrix(true, pred)
    plt.figure(figsize=(10,8))
    # plot the confusion matrix using seaborn package
    sns.heatmap(cm, annot=True, xticklabels=category_names,␣
 ↪yticklabels=category_names)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

def get_param_num(model):
    # referece: https://pytorch.org/docs/stable/generated/torch.nn.parameter.
 ↪Parameter.html and https://pytorch.org/docs/stable/generated/torch.numel.
 ↪html#torch.numel
    # get the total number of parameters
    return sum(param.numel() for param in model.parameters() if param.
 ↪requires_grad)


def get_data_loader(dataset_type, category_names):
    transform_custom = transforms.Compose([
        transforms.ToTensor()
    ])
    train_dataset = CustomDataset(dataset_type, category_names, train_or_test =␣
 ↪'train', transform=transform_custom)
    # transform train data
    transformed_train_data = [i for i in train_dataset]
    train_data_loader = DataLoader(transformed_train_data, batch_size=32,␣
 ↪shuffle=True)
    val_dataset = CustomDataset(dataset_type, category_names, train_or_test =␣
 ↪'test', transform=transform_custom)
    transformed_test_data = [i for i in val_dataset]
    val_loader = DataLoader(transformed_test_data, batch_size=32)
    return train_data_loader, val_loader

def train_model(model, train_data_loader, epochs = 10):
    criterion= torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(
        model.parameters(), lr=1e-3, betas = (0.9, 0.99)
    )
    losses = []
    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 = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            if (i + 1) % 63 == 0:
                losses.append(running_loss)
                print("[epoch : %d, batch: %5d] loss: %.3f"\
                    %(epoch +1, i +1, running_loss/100))
                running_loss = 0.0
    return losses

def test_model(model, val_loader):
    true = []
    pred = []
    # set the model to eval mode
    model.eval()
    # disable gradient calculation
    with torch.no_grad():
        for inputs, labels in val_loader:
            # move the batch of images and labels to device
            inputs = inputs.to(device)
            labels = labels.to(device)
            # pass them to nn model
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            true.extend(labels.tolist())
            pred.extend(preds.tolist())
    return true, pred

class CustomDataset(torch.utils.data.Dataset):
    def __init__(self , dataset_type, category_names, train_or_test =␣
 ↪'train',transform=None):
        # self.root = root
        #self.image_paths = [os.path.join(root , img) for img in os.
 ↪listdir(root)]
        self.transform = transform
        self.data = []
        for label, cat in enumerate(category_names):
            image_dir = f'output_datasets/{dataset_type}/{cat}/'
            train_or_test_dir = os.path.join(image_dir, train_or_test)
            image_files = os.listdir(train_or_test_dir)
            for img_path in image_files:
                modified_path = os.path.join(train_or_test_dir, img_path)
                self.data.append((modified_path, label))

    def __len__(self):
        return len(self.data)

    def __getitem__(self , index):
        img_path, label = self.data[index]
        # print(img_path)
        image = Image.open(img_path).convert("RGB")
        if self.transform:
            image = self.transform(image)
        return image, label


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
net1 = HW4Net().to(device)
net2 = HW4Net().to(device)
net3 = HW4Net().to(device)
```

[152]:
```
dataset_type = 'single_instance'
category_names = ['airplane', 'train', 'bird', 'giraffe', 'toilet']
train_data_loader,val_loader = get_data_loader(dataset_type, category_names)
d1loss = train_model(net1, train_data_loader, epochs = 10)
true, pred = test_model(net1, val_loader)
plot_confusion_matrix(true, pred, category_names)
accuracy = accuracy_score(true, pred)
print("Accuracy score:{}".format(accuracy))
# get the number of parameters
print("Number of parameters:{}".format(get_param_num(net1)))
```

```
[epoch : 1, batch:    63] loss: 0.919
[epoch : 2, batch:    63] loss: 0.778
[epoch : 3, batch:    63] loss: 0.687
[epoch : 4, batch:    63] loss: 0.621
[epoch : 5, batch:    63] loss: 0.589
[epoch : 6, batch:    63] loss: 0.565
[epoch : 7, batch:    63] loss: 0.532
[epoch : 8, batch:    63] loss: 0.507
[epoch : 9, batch:    63] loss: 0.454
[epoch : 10, batch:    63] loss: 0.438
```



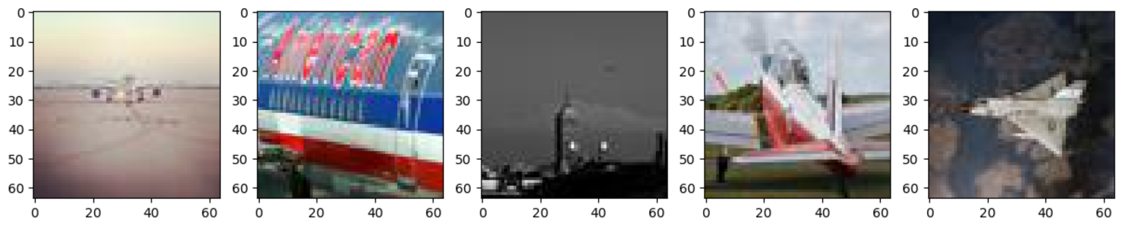
```
Accuracy score:0.628
Number of parameters:406885
```

[161]:
```python
# plot mispecified images
batch = next(iter(val_loader))
first_batch_images = batch[0]
mispecified_images_indices = []

for i in range(len(true)):
    if true[i] != pred[i]:
        mispecified_images_indices.append(i)
print(f'Misclassiified images indices: {mispecified_images_indices}')
```

```
Misclassiified images indices: [4, 10, 17, 25, 28, 31, 33, 45, 47, 48, 56, 57,
58, 66, 68, 69, 74, 77, 83, 85, 94, 95, 101, 103, 104, 109, 116, 117, 124, 126,
128, 131, 132, 134, 139, 142, 145, 146, 149, 154, 157, 163, 166, 173, 175, 188,
190, 191, 197, 200, 201, 203, 204, 205, 207, 208, 210, 211, 212, 213, 214, 216,
217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232,
234, 237, 238, 239, 240, 243, 244, 245, 246, 247, 250, 251, 252, 253, 254, 256,
257, 259, 260, 261, 265, 268, 269, 270, 272, 273, 274, 277, 278, 279, 282, 284,
285, 286, 287, 288, 289, 292, 293, 294, 301, 302, 303, 304, 305, 306, 307, 308,
309, 310, 311, 313, 314, 315, 316, 318, 322, 329, 332, 334, 339, 340, 349, 350,
353, 356, 358, 362, 363, 365, 367, 368, 370, 371, 372, 373, 374, 375, 379, 381,
382, 383, 384, 386, 387, 388, 390, 391, 394, 407, 408, 409, 420, 422, 432, 433,
434, 443, 446, 454, 464, 465, 466, 470, 479, 480, 482, 483]
```

[ ]:
```
# get the first batch of test images
batch = next(iter(val_loader))
first_batch_images = batch[0]
```

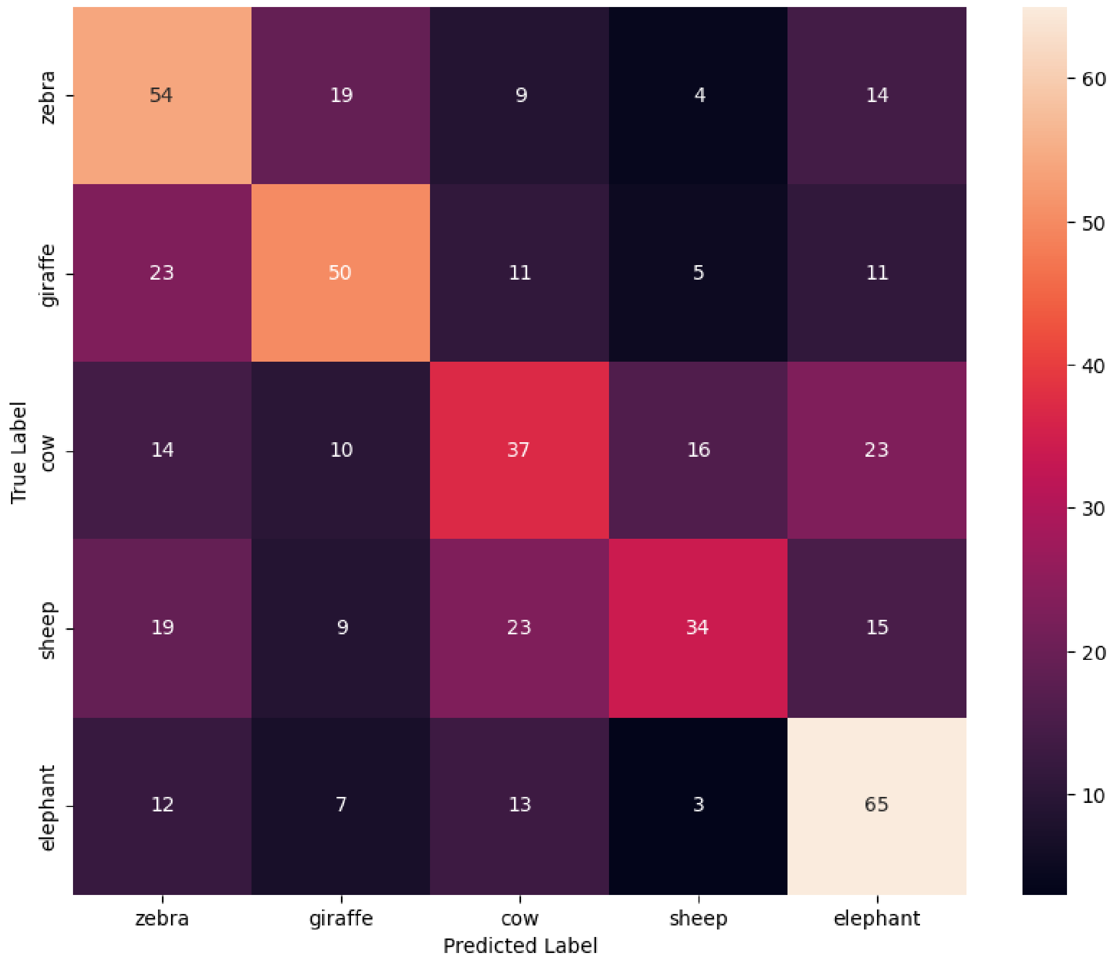
[159]:
```
plt.figure(figsize=(15, 5))
counter = 0
for i in range(len(first_batch_images)):
    if i in mispecified_images_indices:
        # create one row with 5 columns since we want to display 5 images, the␣
 ↪index i controls which column we want to disploy the image
        plt.subplot(1, 5, counter + 1)
        # show the image i and convert the format (C, H, W) to (H, W, C). Here␣
 ↪1, 2, 0 means the dimensions of the np array
        # 0 is the dimension for channel, 1 is the dimension for Height and 2␣
 ↪is the dimension for Width
        plt.imshow(first_batch_images[i].permute(1, 2, 0))
        counter += 1
    if counter ==5:
        break
```



[162]:
```
dataset_type = 'multi_instance_same'
category_names = ['zebra', 'giraffe','cow','sheep', 'elephant']
train_data_loader,val_loader = get_data_loader(dataset_type, category_names)
d2loss = train_model(net2, train_data_loader, epochs = 10)
true, pred = test_model(net2, val_loader)
plot_confusion_matrix(true, pred, category_names)
accuracy = accuracy_score(true, pred)
print("Accuracy score:{}".format(accuracy))
# get the number of parameters
```

```
print("Number of parameters:{}".format(get_param_num(net1)))
```

```
[epoch : 1, batch:    63] loss: 1.010
[epoch : 2, batch:    63] loss: 0.951
[epoch : 3, batch:    63] loss: 0.898
[epoch : 4, batch:    63] loss: 0.859
[epoch : 5, batch:    63] loss: 0.824
[epoch : 6, batch:    63] loss: 0.761
[epoch : 7, batch:    63] loss: 0.718
[epoch : 8, batch:    63] loss: 0.683
[epoch : 9, batch:    63] loss: 0.632
[epoch : 10, batch:    63] loss: 0.587
```

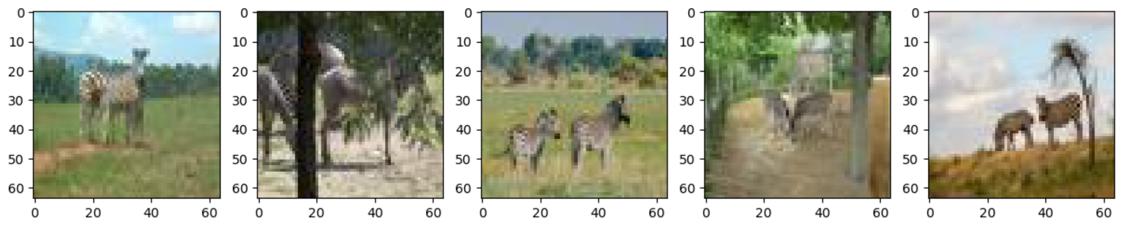


```
Accuracy score:0.48
Number of parameters:406885
```

```
[163]: # plot mispecified images
batch = next(iter(val_loader))
first_batch_images = batch[0]
mispecified_images_indices = []

for i in range(len(true)):
    if true[i] != pred[i]:
        mispecified_images_indices.append(i)
print(f'Misclassiified images indices: {mispecified_images_indices}')
```

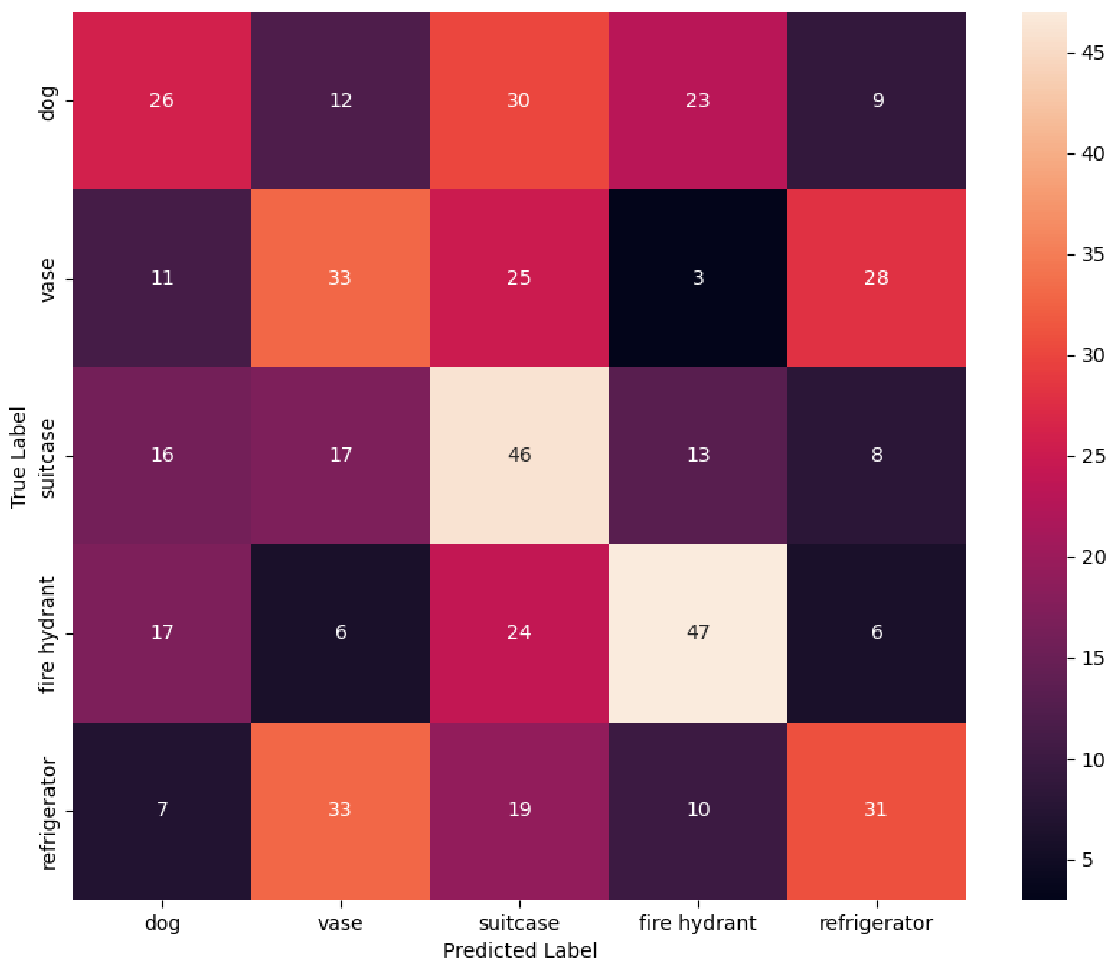
```
Misclassiified images indices: [3, 5, 8, 9, 11, 13, 14, 18, 21, 27, 28, 29, 31,
35, 37, 38, 40, 43, 45, 46, 47, 49, 52, 55, 56, 57, 58, 62, 63, 64, 65, 68, 73,
77, 79, 80, 81, 84, 85, 87, 90, 92, 94, 96, 98, 99, 101, 105, 106, 107, 110,
112, 116, 118, 119, 120, 124, 125, 128, 130, 131, 133, 136, 139, 140, 141, 142,
143, 144, 145, 146, 148, 149, 150, 153, 154, 155, 157, 158, 159, 160, 161, 162,
164, 168, 174, 179, 184, 185, 187, 188, 189, 190, 194, 198, 199, 200, 201, 202,
203, 204, 205, 207, 209, 210, 211, 212, 213, 216, 218, 219, 221, 223, 224, 225,
230, 231, 232, 233, 234, 238, 239, 242, 243, 246, 247, 248, 251, 252, 253, 254,
255, 259, 261, 262, 263, 264, 266, 267, 268, 269, 270, 271, 272, 273, 278, 279,
280, 282, 284, 285, 288, 290, 292, 293, 295, 296, 297, 298, 300, 303, 304, 306,
307, 309, 310, 312, 313, 315, 316, 317, 318, 321, 323, 326, 327, 329, 330, 331,
333, 334, 335, 336, 337, 338, 339, 340, 341, 344, 345, 348, 349, 350, 352, 354,
356, 358, 359, 360, 361, 362, 363, 366, 367, 368, 369, 370, 371, 372, 373, 374,
377, 379, 381, 385, 386, 387, 388, 389, 390, 391, 393, 395, 397, 398, 400, 401,
402, 406, 411, 412, 413, 417, 421, 422, 425, 428, 431, 437, 438, 440, 442, 444,
445, 446, 447, 450, 453, 454, 455, 460, 469, 470, 471, 472, 477, 480, 484, 486,
490]
```

```
[164]: plt.figure(figsize=(15, 5))
counter = 0
for i in range(len(first_batch_images)):
    if i in mispecified_images_indices:
        # create one row with 5 columns since we want to display 5 images, the␣
 ↪index i controls which column we want to disploy the image
        plt.subplot(1, 5, counter + 1)
        # show the image i and convert the format (C, H, W) to (H, W, C). Here␣
 ↪1, 2, 0 means the dimensions of the np array
        # 0 is the dimension for channel, 1 is the dimension for Height and 2␣
 ↪is the dimension for Width
        plt.imshow(first_batch_images[i].permute(1, 2, 0))
        counter += 1
    if counter ==5:
        break
```



```
[165]: dataset_type = 'multi_instance_different'
       category_names = ['dog', 'vase', 'suitcase', 'fire hydrant', 'refrigerator']
       train_data_loader,val_loader = get_data_loader(dataset_type, category_names)
       d3loss = train_model(net3, train_data_loader, epochs = 10)
       true, pred = test_model(net3, val_loader)
       plot_confusion_matrix(true, pred, category_names)
       accuracy = accuracy_score(true, pred)
       print("Accuracy score:{}".format(accuracy))
       # get the number of parameters
       print("Number of parameters:{}".format(get_param_num(net1)))
```

```
[epoch : 1, batch:    63] loss: 1.007
[epoch : 2, batch:    63] loss: 0.946
[epoch : 3, batch:    63] loss: 0.914
[epoch : 4, batch:    63] loss: 0.888
[epoch : 5, batch:    63] loss: 0.871
[epoch : 6, batch:    63] loss: 0.831
[epoch : 7, batch:    63] loss: 0.795
[epoch : 8, batch:    63] loss: 0.762
[epoch : 9, batch:    63] loss: 0.705
[epoch : 10, batch:    63] loss: 0.659
```

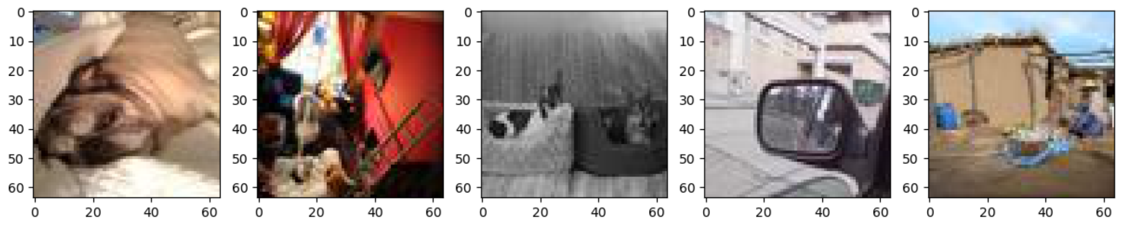


```
Accuracy score:0.366
Number of parameters:406885
```

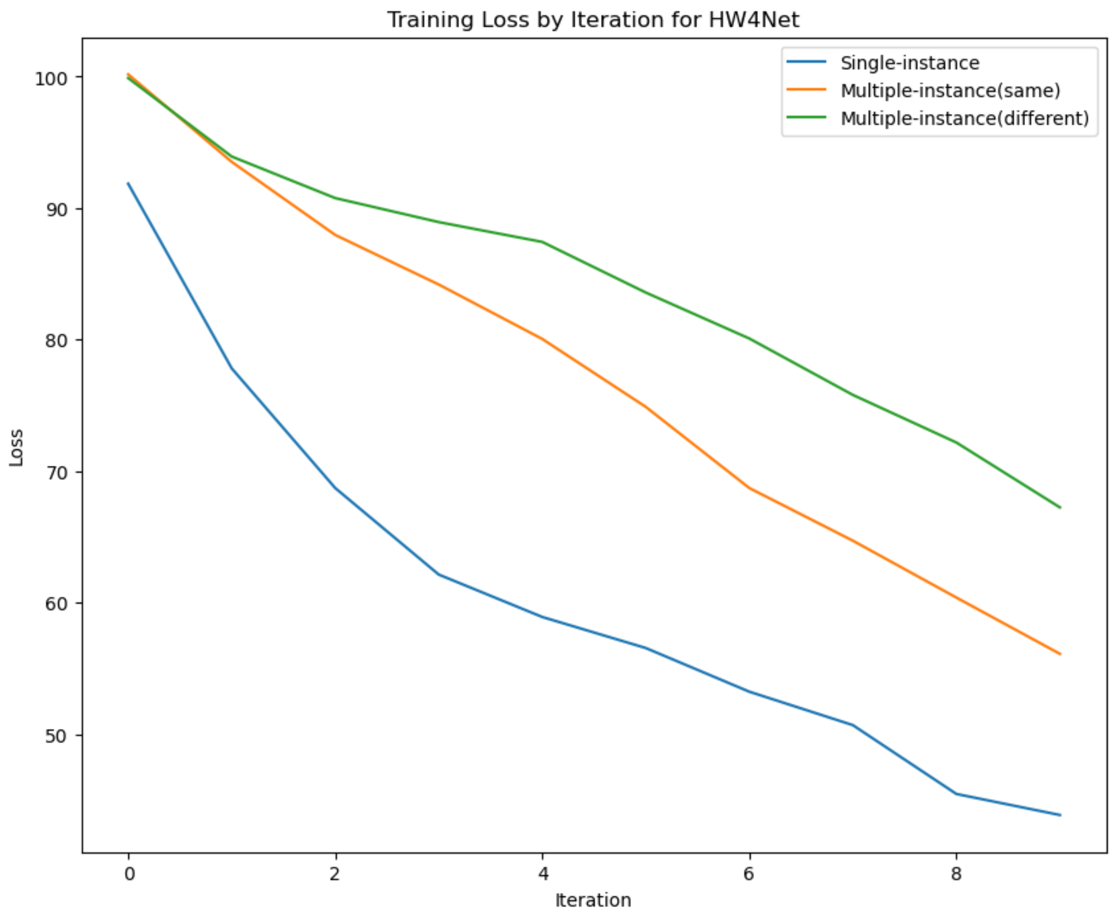
[166]:
```python
# plot mispecified images
batch = next(iter(val_loader))
first_batch_images = batch[0]
mispecified_images_indices = []

for i in range(len(true)):
    if true[i] != pred[i]:
        mispecified_images_indices.append(i)
print(f'Misclassiified images indices: {mispecified_images_indices}')
```

```
Misclassiified images indices: [0, 1, 2, 3, 5, 6, 7, 8, 10, 12, 13, 14, 16, 17,
18, 19, 20, 23, 24, 25, 27, 28, 29, 30, 31, 32, 33, 35, 36, 37, 38, 39, 40, 41,
42, 43, 45, 46, 47, 48, 49, 50, 52, 53, 54, 55, 59, 60, 61, 63, 65, 66, 67, 70,
72, 74, 75, 76, 77, 79, 83, 85, 86, 87, 89, 91, 92, 93, 94, 95, 96, 97, 98, 99,
100, 101, 104, 105, 107, 109, 110, 111, 112, 113, 114, 116, 117, 119, 120, 121,
123, 124, 125, 129, 131, 133, 134, 135, 136, 140, 141, 144, 145, 146, 147, 148,
149, 151, 152, 154, 156, 157, 159, 160, 161, 162, 164, 165, 166, 168, 170, 172,
174, 177, 178, 180, 182, 183, 185, 186, 187, 189, 191, 192, 193, 194, 195, 196,
197, 198, 199, 200, 202, 203, 204, 206, 207, 209, 210, 211, 213, 214, 215, 216,
217, 218, 219, 220, 222, 224, 225, 226, 227, 229, 232, 233, 234, 236, 238, 240,
241, 242, 247, 253, 254, 255, 257, 258, 262, 263, 264, 270, 273, 274, 275, 282,
283, 284, 285, 291, 292, 293, 294, 295, 299, 300, 301, 302, 303, 304, 306, 307,
310, 314, 316, 318, 320, 321, 323, 325, 326, 330, 332, 335, 337, 338, 339, 342,
345, 346, 347, 348, 349, 351, 355, 356, 358, 359, 360, 364, 366, 371, 372, 374,
376, 377, 378, 379, 380, 382, 384, 385, 386, 387, 391, 392, 395, 398, 400, 401,
405, 409, 410, 411, 412, 413, 414, 416, 417, 418, 419, 420, 421, 422, 423, 424,
425, 426, 427, 431, 432, 433, 436, 437, 439, 440, 441, 442, 443, 447, 449, 450,
452, 454, 455, 457, 458, 459, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471,
472, 476, 477, 478, 480, 482, 484, 485, 487, 488, 489, 490, 491, 492, 494, 495,
496, 497, 498]
```

[167]:
```
plt.figure(figsize=(15, 5))
counter = 0
for i in range(len(first_batch_images)):
    if i in mispecified_images_indices:
        # create one row with 5 columns since we want to display 5 images, the␣
↪index i controls which column we want to disploy the image
        plt.subplot(1, 5, counter + 1)
        # show the image i and convert the format (C, H, W) to (H, W, C). Here␣
↪1, 2, 0 means the dimensions of the np array
        # 0 is the dimension for channel, 1 is the dimension for Height and 2␣
↪is the dimension for Width
        plt.imshow(first_batch_images[i].permute(1, 2, 0))
        counter += 1
    if counter ==5:
        break
```



[138]:
```
plot_learning_curve_results(d1loss, d2loss, d3loss)
```



0.0.4 Bonus

To avoid a lengthy report on the calculation, I comment on the output size for each layer in the class. I use the same calculation method as in how I calculated the output size for each layer in HW4Net. I specifically change the default parameter `padding` to be `1` to avoid shrinking the input too fast in the network for both 5-layer and 10 layer-networks to equalize the size subtraction due to the kernel size in the formula. I feed both network with single-instance data with 2000 training images with 400 images per class and the model is tested on 500 test images with 100 images per class. I calculate the input size for `nn.Linear()` after the last pooling layer by mulitplying the output size for both networks as these inputs get flatten.

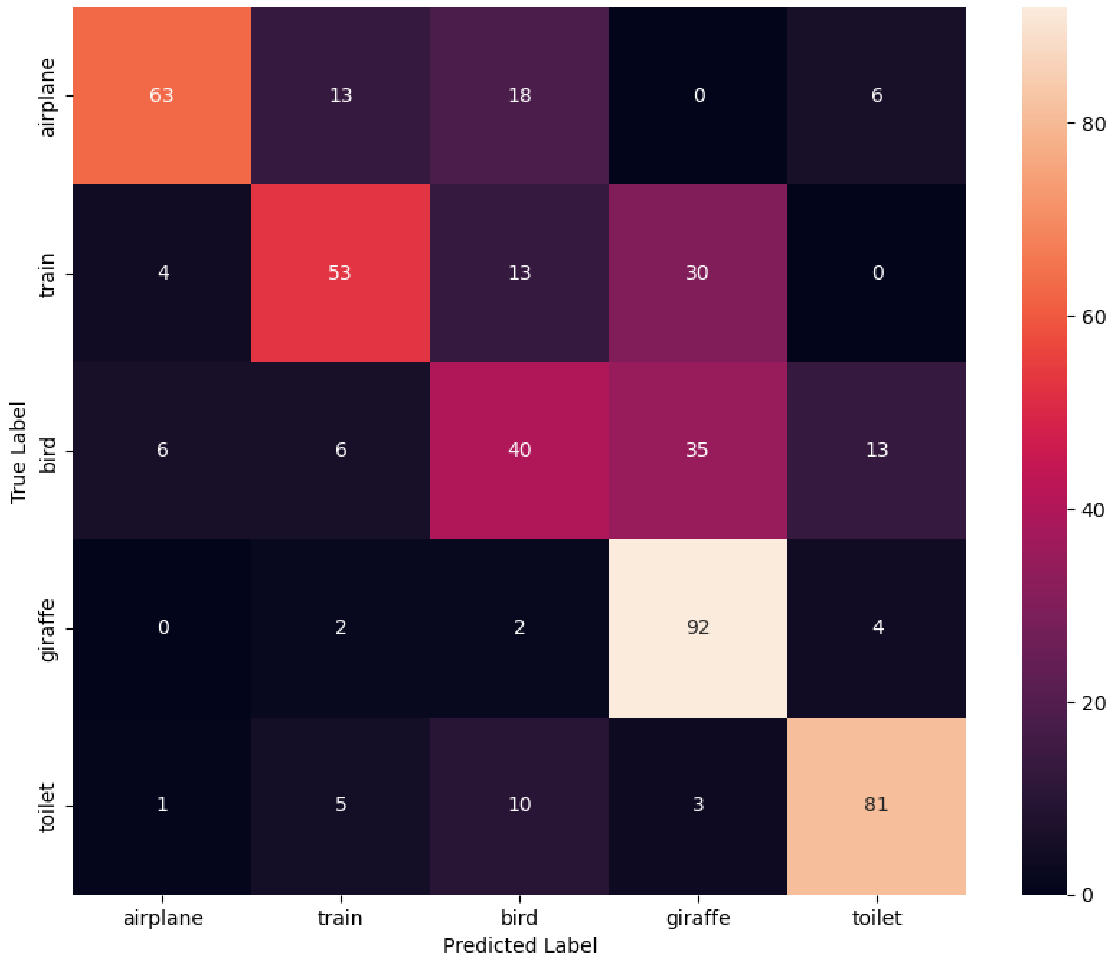
Does a deeper network always result in better performance? - Based on the accuracy score, the 5-layer network has slightly improved the performance going from 0.628 to 0.658. We can see even 'Bird' is still the most difficult category to classify, However, for 10-layer network, the accuracy drastically drops to 0.228. Therefore, deeper networks do not necessarily perform better. The training loss curve does not seem to converge for the 10-layer network. It mostly predict the class label to be either airplane or giraffe.

[145]:
```
import torch.nn as nn
import torch.nn.functional as F

class HWNet1(nn.Module):
    def __init__(self):
        super(HWNet1, self).__init__()
        self.layers = nn.Sequential(
            nn.Conv2d(3, 16, 3, padding=1), # output size: 16x 64 x 64
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output size: 16 x 32 x 32

            nn.Conv2d(16, 32, 3, padding=1), # output size: 32 x 32 x 32
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output size: 32 x16 x16

            nn.Conv2d(32, 64, 3, padding=1), # output size: 64 x16 x16
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output size: 64 x 8 x 8

            nn.Conv2d(64, 128, 3, padding=1),# output size: 128 x 8 x 8
            nn.ReLU(),
            nn.MaxPool2d(2, 2),# output size: 128 x 4 x 4

            nn.Conv2d(128, 256, 3, padding=1), # output size: 256 x 4 x 4
            nn.ReLU(),
            nn.MaxPool2d(2, 2) # output size: 256 x 2 x 2
        )

        # fill XXXX and XX
        self.fc1 = nn.Linear(1024,128)
        self.fc2 = nn.Linear(128,5) # we set to 5 because there are only 5␣
 ↪classes

    def forward (self, x):
        x = self.layers(x)
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

[146]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
dnet1 = HWNet1().to(device)

dataset_type = 'single_instance'
category_names = ['airplane', 'train', 'bird', 'giraffe', 'toilet']
train_data_loader,val_loader = get_data_loader(dataset_type, category_names)
d1loss = train_model(dnet1, train_data_loader, epochs = 10)
```

```
true, pred = test_model(dnet1, val_loader)
plot_confusion_matrix(true, pred, category_names)
accuracy = accuracy_score(true, pred)
print("Accuracy score:{}".format(accuracy))
```

```
[epoch : 1, batch:     63] loss: 1.010
[epoch : 2, batch:     63] loss: 0.912
[epoch : 3, batch:     63] loss: 0.847
[epoch : 4, batch:     63] loss: 0.834
[epoch : 5, batch:     63] loss: 0.794
[epoch : 6, batch:     63] loss: 0.749
[epoch : 7, batch:     63] loss: 0.712
[epoch : 8, batch:     63] loss: 0.674
[epoch : 9, batch:     63] loss: 0.625
[epoch : 10, batch:     63] loss: 0.587
```

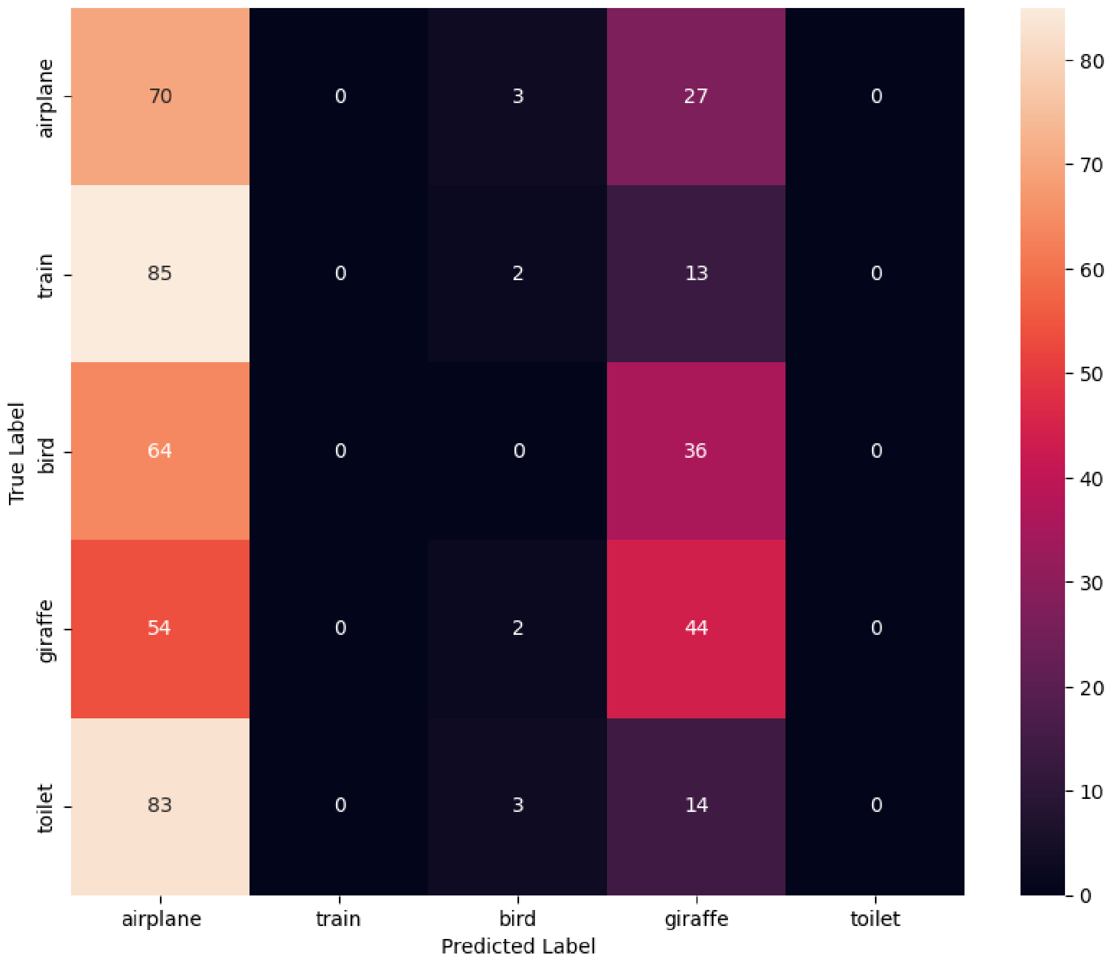


```
Accuracy score:0.658
```

[149]:
```
class HWNet2(nn.Module):
    def __init__(self):
        super(HWNet2, self).__init__()
        self.layers = nn.Sequential(
            nn.Conv2d(3, 16, 3, padding=1), # output size: 16x 64 x 64
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, padding=1), # output size: 32x 64 x 64
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output size: 32x 32 x 32

            nn.Conv2d(32, 64, 3, padding=1), # output size: 64x 32 x 32
            nn.ReLU(),
            nn.Conv2d(64, 128, 3, padding=1),  # output size: 128x 32 x 32
            nn.ReLU(),
            nn.MaxPool2d(2, 2),  # output size: 128x 16 x 16

            nn.Conv2d(128, 256, 3, padding=1), # output size: 256 x 16 x 16
            nn.ReLU(),
            nn.Conv2d(256, 512, 3, padding=1), # output size: 512 x 16 x 16
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output size: 512 x 8 x 8

            nn.Conv2d(512, 512, 3, padding=1), # output size: 512 x 8 x 8
            nn.ReLU(),
            nn.Conv2d(512, 512, 3, padding=1),  # output size: 512 x 8 x 8
            nn.ReLU(),
            nn.MaxPool2d(2, 2),  # output size: 512 x 4 x 4

            nn.Conv2d(512, 512, 3, padding=1), # output size: 512 x 4 x 4
            nn.ReLU(),
            nn.Conv2d(512, 512, 3, padding=1),  # output size: 512 x 4 x 4
            nn.ReLU(),
            nn.MaxPool2d(2, 2)  # output size: 512 x 2 x 2
        )
        # fill XXXX and XX
        self.fc1 = nn.Linear(2048, 256)
        self.fc2 = nn.Linear(256,5) # we set to 5 because there are only 5␣
 ↪classes


    def forward (self, x):
        x = self.layers(x)
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

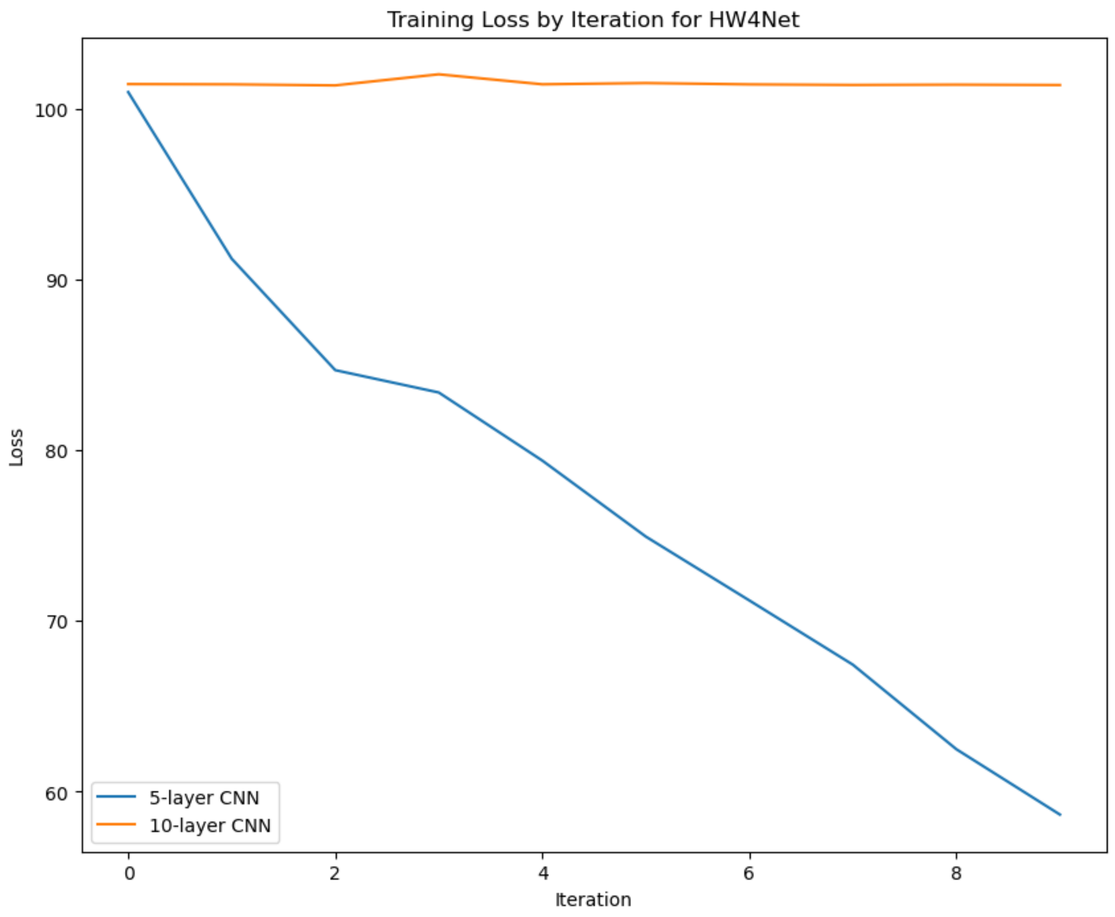
```
dnet2 = HWNet2().to(device)
d2loss = train_model(dnet2, train_data_loader, epochs = 10)
true, pred = test_model(dnet2, val_loader)
plot_confusion_matrix(true, pred, category_names)
accuracy = accuracy_score(true, pred)
print("Accuracy score:{}".format(accuracy))
```

```
[epoch : 1, batch:    63] loss: 1.015
[epoch : 2, batch:    63] loss: 1.014
[epoch : 3, batch:    63] loss: 1.014
[epoch : 4, batch:    63] loss: 1.020
[epoch : 5, batch:    63] loss: 1.014
[epoch : 6, batch:    63] loss: 1.015
[epoch : 7, batch:    63] loss: 1.014
[epoch : 8, batch:    63] loss: 1.014
[epoch : 9, batch:    63] loss: 1.014
[epoch : 10, batch:    63] loss: 1.014
```



```
Accuracy score:0.228
```

[150]:
```
plt.figure(figsize=(10,8))
plt.plot(d1loss, label='5-layer CNN')
plt.plot(d2loss, label='10-layer CNN')
plt.ylabel('Loss')
plt.xlabel('Iteration')
plt.title(f'Training Loss by Iteration for HW4Net')
plt.legend()
plt.show()
```



[ ]:
```

```