1. Observations on 1 pixel kernel

In Line (I), the operation `self.in2out = nn.Conv2d(in_ch, out_ch, 1)` is used to change the number of channels from in_ch to out_ch. By using a 1×1 kernel, this convolution applies a transformation independently to each pixel across channels. Instead of considering neighboring pixels, it performs a per-pixel operation, essentially appling learned linear transformation to each pixel's channel values. This allows for a channel-wise transformation while preserving spatial structure.

In Lines (K) and (L), the operations `self.downsampler1 = nn.Conv2d(in_ch, in_ch, 1, stride=2)` and `self.downsampler2 = nn.Conv2d(out_ch, out_ch, 1, stride=2)` serve as downsamplers. By setting stride=2, the convolution layer skips every other pixel in both the width and height dimensions. This results in a reduction of spatial resolution while performing a per-pixel linear transformation across channels. Unlike conventional convolution, which considers spatial neighbors, this method reduces the number of pixels processed while retaining key features.

The reason why Lines (I), (K), and (L) make sense in using a 1×1 kernel is that they perform a linear transformation at each pixel location while maintaining spatial information. Line (I) adjusts the number of channels without altering spatial dimensions, whereas Lines (K) and (L) downsample the feature map while preserving channel-wise structure.

2. BMENet with maxpool

In [1]:
```
import os, sys
print(os.getcwd())
sys.path.append("./../DLStudio-2.5.1")

# from DLStudio import *
```

```
/home/chen4126/ece60146/HW6
```

In [4]:
```
import torch.nn as nn
from DLStudio import DLStudio
import torch

# this code is mainly borrowed from DLstudio, the code that i modify has a comment
class BMEnetModifiedForMaxpool(nn.Module):
    def __init__(self, dl_studio, skip_connections=True, depth=8):
        super().__init__()

        self.dl_studio = dl_studio
        self.depth = depth
        image_size = dl_studio.image_size
        ## num_ds stands for number of downsampling steps
        num_ds = 0
```

```
        self.conv = nn.Conv2d(3, 64, 3, padding=1)

        # layers of skipblock without downsample
        self.skip64_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip64_arr.append(self.SkipBlock(64, 64, skip_connections=skip_con
        # a skipblock with downsample
        self.skip64to128ds = self.SkipBlock(64, 128, downsample=True, skip_connecti
        num_ds += 1

        self.skip128_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip128_arr.append(self.SkipBlock(128, 128, skip_connections=skip_
        self.skip128to256ds = self.SkipBlock(128, 256, downsample=True, skip_connec
        num_ds += 1

        self.skip256_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip256_arr.append(self.SkipBlock(256, 256, skip_connections=skip_

        self.fc1 =  nn.Linear( (image_size[0]// (2 ** num_ds))  *  (image_size[1]//
        self.fc2 =  nn.Linear(1000, 10)



    def forward(self, x):
        x = nn.functional.relu(self.conv(x))
        for skip64 in self.skip64_arr:
            x = skip64(x)
        x = self.skip64to128ds(x)
        for skip128 in self.skip128_arr:
            x = skip128(x)
        x = self.skip128to256ds(x)
        for skip256 in self.skip256_arr:
            x = skip256(x)
        x  =  x.view( x.shape[0], - 1 )
        x = nn.functional.relu(self.fc1(x))
        x = self.fc2(x)
        return x

    def load_cifar_10_dataset(self):
        self.dl_studio.load_cifar_10_dataset()

    def load_cifar_10_dataset_with_augmentation(self):
        self.dl_studio.load_cifar_10_dataset_with_augmentation()



    class SkipBlock(nn.Module):
        def __init__(self, in_ch, out_ch, downsample=False, skip_connections=True):
            super().__init__()
            self.downsample = downsample
            self.skip_connections = skip_connections
            self.in_ch = in_ch
```

```
            self.out_ch = out_ch
            self.convo1 = nn.Conv2d(in_ch, in_ch, 3, stride=1, padding=1)
            self.convo2 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
            self.bn1 = nn.BatchNorm2d(in_ch)
            self.bn2 = nn.BatchNorm2d(out_ch)
            self.in2out  =  nn.Conv2d(in_ch, out_ch, 1)
            if downsample:
                self.maxpool = nn.MaxPool2d(2, 2) # do maxpool


        def forward(self, x):
            # identity holds the original input tensor to the skip block
            identity = x
            out = self.convo1(x)
            out = self.bn1(out)
            out = nn.functional.relu(out)
            out = self.convo2(out)
            out = self.bn2(out)
            out = nn.functional.relu(out)


            # Apply downsampling if needed
            if self.downsample:
                identity = self.maxpool(identity)       # channel still input size
                out = self.maxpool(out)                #  channel still output size

            # Handle skip connections
            if self.skip_connections:
                if (self.in_ch == self.out_ch) and (self.downsample is False):
                    out = out + identity
                elif (self.in_ch != self.out_ch) and (self.downsample is False):
                    identity = self.in2out( identity )
                    out = out + identity
                elif (self.in_ch != self.out_ch) and (self.downsample is True):
                    out = out + torch.cat((identity, identity), dim=1)

            return out


    def run_code_for_training(self, net, display_images=False):
        self.dl_studio.run_code_for_training(net, display_images)

    def save_model(self, model):
        '''
        Save the trained model to a disk file
        '''
        torch.save(model.state_dict(), self.dl_studio.path_saved_model)

    def run_code_for_testing(self, model, display_images=False):
        self.dl_studio.run_code_for_testing(model, display_images=False)


dls = DLStudio(
#                  dataroot = "/home/kak/ImageDatasets/CIFAR-10/",
                dataroot = "./../data/CIFAR-10/",
                image_size = [32,32],
```

```
                path_saved_model = "./saved_model",
                momentum = 0.9,
                learning_rate = 1e-5,
                epochs = 10,
                batch_size = 16,
                classes = ('plane','car','bird','cat','deer','dog','frog','horse','
                use_gpu = True,
            )

bme_net = BMEnetModifiedForMaxpool(dls, skip_connections=True, depth=8)         ##

bme_net.load_cifar_10_dataset()

## display network properties
number_of_learnable_params = sum(p.numel() for p in bme_net.parameters() if p.requi
print("\n\nThe number of learnable parameters in the model: %d" % number_of_learnab

## training and testing
bme_net.run_code_for_training(bme_net, display_images=False)
bme_net.run_code_for_testing(bme_net, display_images=False)
```

```
Files already downloaded and verified
Files already downloaded and verified


The number of learnable parameters in the model: 30092354


Starting training loop...



[epoch:1/10  iter=1000  elapsed_time=   25 secs]   Ground Truth:           bird
plane       frog       ship        car       deer       frog       ship       deer
horse       bird       deer       frog        car        dog       ship
[epoch:1/10  iter=1000  elapsed_time=   25 secs]   Predicted Labels:      plane
plane        car        car        car       deer       deer       deer        dog
dog       bird        cat        dog        dog        dog       bird
[epoch:1/10  iter=1000  elapsed_time=   25 secs]   Loss: 2.278


[epoch:1/10  iter=2000  elapsed_time=   51 secs]   Ground Truth:           frog
plane        dog      truck        cat      horse        car       bird       deer
deer        car       frog      horse       bird      horse      horse
[epoch:1/10  iter=2000  elapsed_time=   51 secs]   Predicted Labels:       frog
ship       bird        car        cat       ship        car       frog      truck
frog        car       frog      horse       frog        dog      truck
[epoch:1/10  iter=2000  elapsed_time=   51 secs]   Loss: 1.803


[epoch:1/10  iter=3000  elapsed_time=   75 secs]   Ground Truth:          plane
car       frog        cat       bird      horse      truck        car       ship
deer        cat        cat        car       frog        dog       bird
[epoch:1/10  iter=3000  elapsed_time=   75 secs]   Predicted Labels:      plane
horse        dog        cat       bird      plane      truck        car       ship
deer        dog       frog      truck       deer        dog       bird
[epoch:1/10  iter=3000  elapsed_time=   75 secs]   Loss: 1.668



[epoch:2/10  iter=1000  elapsed_time=  104 secs]   Ground Truth:          truck
deer      plane       bird        cat        dog       ship       bird       deer
bird       frog      truck        car       deer       ship      horse
[epoch:2/10  iter=1000  elapsed_time=  104 secs]   Predicted Labels:        car
deer       ship      plane        cat        dog        car       deer        car
dog       deer        dog        car       deer       ship      horse
[epoch:2/10  iter=1000  elapsed_time=  104 secs]   Loss: 1.578


[epoch:2/10  iter=2000  elapsed_time=  128 secs]   Ground Truth:           dog
deer        car      horse       deer      horse       frog        cat       ship
plane      plane        car       frog       bird        car       frog
[epoch:2/10  iter=2000  elapsed_time=  128 secs]   Predicted Labels:        dog
ship        car       deer       deer       deer       frog        dog      truck
plane      plane        car      horse       bird        car       frog
[epoch:2/10  iter=2000  elapsed_time=  128 secs]   Loss: 1.526
```

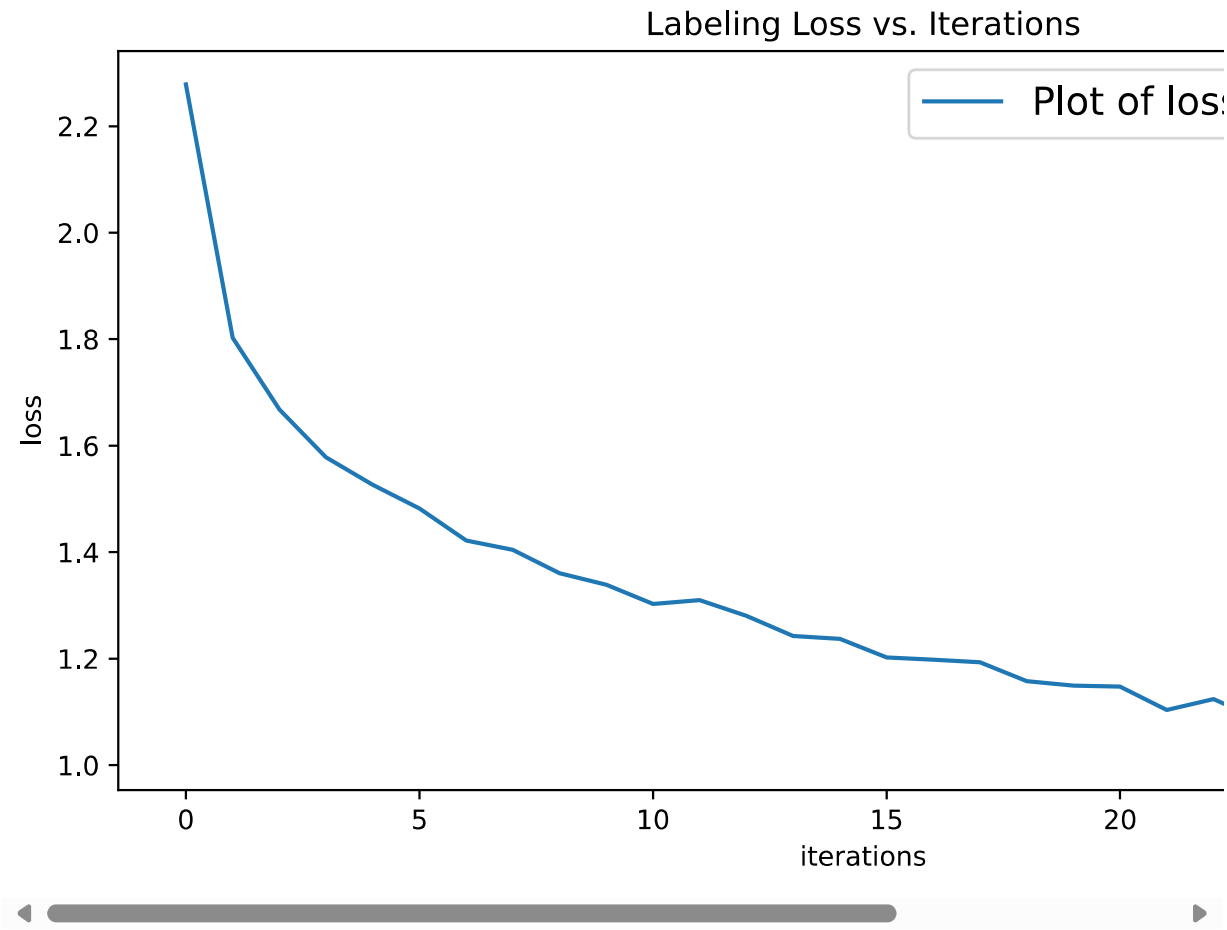
```
[epoch:2/10  iter=3000  elapsed_time=  153 secs]   Ground Truth:           bird
cat       deer      plane      horse       ship       deer       frog        dog
plane        dog      horse      plane        car        dog       frog
[epoch:2/10  iter=3000  elapsed_time=  153 secs]   Predicted Labels:        dog
horse       frog      plane        car      truck       deer       bird      horse
plane        dog        dog        car        car        dog        car
[epoch:2/10  iter=3000  elapsed_time=  153 secs]   Loss: 1.482



[epoch:3/10  iter=1000  elapsed_time=  183 secs]   Ground Truth:           ship
plane      horse        cat       bird       deer       frog       deer        cat
deer       bird       deer        cat      truck       bird      truck
[epoch:3/10  iter=1000  elapsed_time=  183 secs]   Predicted Labels:       ship
plane      horse        cat       bird      horse       frog        car      truck
horse       bird       deer        dog        car       bird      truck
[epoch:3/10  iter=1000  elapsed_time=  183 secs]   Loss: 1.422



[epoch:3/10  iter=2000  elapsed_time=  208 secs]   Ground Truth:           deer
horse        dog      truck       ship       bird       frog       bird       ship
truck       frog      plane      horse       frog        car      truck
[epoch:3/10  iter=2000  elapsed_time=  208 secs]   Predicted Labels:        dog
horse        cat      truck       bird       bird       frog      truck       ship
truck       frog       bird      horse       frog      truck      truck
[epoch:3/10  iter=2000  elapsed_time=  208 secs]   Loss: 1.405



[epoch:3/10  iter=3000  elapsed_time=  232 secs]   Ground Truth:            cat
deer       ship        cat      truck       bird       cat      plane      horse
car      horse      plane      horse      truck       deer        cat
[epoch:3/10  iter=3000  elapsed_time=  232 secs]   Predicted Labels:        dog
deer        car        dog        car       frog       dog      plane      horse
car       frog      plane      horse      plane      plane        dog
[epoch:3/10  iter=3000  elapsed_time=  232 secs]   Loss: 1.361



[epoch:4/10  iter=1000  elapsed_time=  260 secs]   Ground Truth:          horse
deer       frog       bird      horse       bird      truck      horse        dog
cat        dog      plane      plane        cat       frog       bird
[epoch:4/10  iter=1000  elapsed_time=  260 secs]   Predicted Labels:        cat
deer       deer       deer       deer       ship      truck      horse        cat
cat        cat      plane      plane        cat       frog       ship
[epoch:4/10  iter=1000  elapsed_time=  260 secs]   Loss: 1.339



[epoch:4/10  iter=2000  elapsed_time=  285 secs]   Ground Truth:          plane
horse       frog      truck       deer       bird       frog        dog      truck
car        car      truck      plane        dog      horse       deer
[epoch:4/10  iter=2000  elapsed_time=  285 secs]   Predicted Labels:      plane
plane      horse        car       deer       ship       frog        cat        cat
car      truck        car      plane        dog      horse       deer
[epoch:4/10  iter=2000  elapsed_time=  285 secs]   Loss: 1.303
```

```
[epoch:4/10  iter=3000  elapsed_time=  308 secs]   Ground Truth:           ship
deer       ship        dog        car      plane      plane       bird      truck
bird       deer        car      truck       bird        car        car
[epoch:4/10  iter=3000  elapsed_time=  308 secs]   Predicted Labels:       ship
bird       ship        dog        car      plane      plane       bird      truck
dog       deer        car      plane       bird        car       deer
[epoch:4/10  iter=3000  elapsed_time=  308 secs]   Loss: 1.310



[epoch:5/10  iter=1000  elapsed_time=  338 secs]   Ground Truth:            dog
horse      truck        dog       ship       bird       ship       frog       frog
car        car        dog       deer       bird      plane       bird
[epoch:5/10  iter=1000  elapsed_time=  338 secs]   Predicted Labels:       deer
horse      horse      horse       ship      horse        car       deer        cat
car      truck        dog        dog      plane      plane       bird
[epoch:5/10  iter=1000  elapsed_time=  338 secs]   Loss: 1.281



[epoch:5/10  iter=2000  elapsed_time=  363 secs]   Ground Truth:            dog
car       bird        car      plane       bird       frog       deer       deer
plane        cat      plane        car        dog        dog        dog
[epoch:5/10  iter=2000  elapsed_time=  363 secs]   Predicted Labels:        cat
car       bird       ship      plane       ship       deer        cat        cat
ship        dog      plane      truck      horse        cat       bird
[epoch:5/10  iter=2000  elapsed_time=  363 secs]   Loss: 1.242



[epoch:5/10  iter=3000  elapsed_time=  387 secs]   Ground Truth:            dog
car      truck       ship       frog       frog       deer       frog        cat
plane      truck       ship        cat        cat       deer      plane
[epoch:5/10  iter=3000  elapsed_time=  387 secs]   Predicted Labels:        cat
car        car       ship       frog        cat        cat       frog        cat
plane        car       ship        cat       deer       bird      plane
[epoch:5/10  iter=3000  elapsed_time=  387 secs]   Loss: 1.237



[epoch:6/10  iter=1000  elapsed_time=  416 secs]   Ground Truth:           deer
truck       bird       deer        car      horse        dog      horse        car
frog       frog        cat        cat       deer      plane        car
[epoch:6/10  iter=1000  elapsed_time=  416 secs]   Predicted Labels:       deer
truck       bird        cat        car      truck        dog      horse      truck
frog        cat        cat       frog       bird      plane      truck
[epoch:6/10  iter=1000  elapsed_time=  416 secs]   Loss: 1.203



[epoch:6/10  iter=2000  elapsed_time=  439 secs]   Ground Truth:          truck
ship       bird      truck      plane      plane       ship        dog       frog
truck      truck      horse      horse        cat      plane      horse
[epoch:6/10  iter=2000  elapsed_time=  439 secs]   Predicted Labels:       ship
ship      horse       ship      truck      plane       bird        cat        cat
truck      truck      horse       bird       deer      plane      horse
[epoch:6/10  iter=2000  elapsed_time=  439 secs]   Loss: 1.198
```

```
[epoch:6/10  iter=3000  elapsed_time=  462 secs]   Ground Truth:          horse
horse       bird       ship       bird        cat        dog        cat       ship
horse      plane      plane       bird        dog       ship        dog
[epoch:6/10  iter=3000  elapsed_time=  462 secs]   Predicted Labels:        dog
horse      plane        car       bird       frog        dog        dog       ship
horse       frog      plane       bird       deer       ship        dog
[epoch:6/10  iter=3000  elapsed_time=  462 secs]   Loss: 1.193



[epoch:7/10  iter=1000  elapsed_time=  491 secs]   Ground Truth:           frog
dog      truck      truck      truck      horse       ship      truck        cat
ship        dog      plane       deer      horse      truck        dog
[epoch:7/10  iter=1000  elapsed_time=  491 secs]   Predicted Labels:       frog
deer        car        cat        car      horse      plane        car        cat
ship        cat      plane       deer        dog        car        cat
[epoch:7/10  iter=1000  elapsed_time=  491 secs]   Loss: 1.158



[epoch:7/10  iter=2000  elapsed_time=  514 secs]   Ground Truth:           cat
bird       ship       frog        cat      truck      horse        car        car
ship       bird       bird       frog      horse      horse       deer
[epoch:7/10  iter=2000  elapsed_time=  514 secs]   Predicted Labels:        dog
plane      truck       frog       deer      plane      horse        car        car
plane       ship       bird       frog        dog      horse       deer
[epoch:7/10  iter=2000  elapsed_time=  514 secs]   Loss: 1.150



[epoch:7/10  iter=3000  elapsed_time=  537 secs]   Ground Truth:          plane
dog       bird        dog        cat      truck       deer        car        car
dog       ship       deer       ship      horse      horse       bird
[epoch:7/10  iter=3000  elapsed_time=  537 secs]   Predicted Labels:       deer
dog       deer        dog        dog      truck       deer        car      truck
horse        car       deer      truck       deer      horse       bird
[epoch:7/10  iter=3000  elapsed_time=  537 secs]   Loss: 1.148



[epoch:8/10  iter=1000  elapsed_time=  565 secs]   Ground Truth:          truck
deer       frog      plane       bird       frog       ship        dog      truck
plane        cat      plane      plane        cat       frog      horse
[epoch:8/10  iter=1000  elapsed_time=  565 secs]   Predicted Labels:      truck
bird       frog       ship       bird        cat       ship        dog      truck
plane        cat      plane        car        dog       frog      horse
[epoch:8/10  iter=1000  elapsed_time=  565 secs]   Loss: 1.104



[epoch:8/10  iter=2000  elapsed_time=  589 secs]   Ground Truth:           car
truck      plane        cat        car        car      plane      horse       bird
horse       ship       ship       deer       deer       ship      truck
[epoch:8/10  iter=2000  elapsed_time=  589 secs]   Predicted Labels:        car
truck      plane       bird        car      truck      plane      horse       bird
horse       ship      truck       frog       deer       ship      truck
[epoch:8/10  iter=2000  elapsed_time=  589 secs]   Loss: 1.124
```

```
[epoch:8/10  iter=3000  elapsed_time=  613 secs]   Ground Truth:           bird
horse       frog        dog       bird       frog       ship       deer      horse
cat       frog       frog      plane      plane        dog        car
[epoch:8/10  iter=3000  elapsed_time=  613 secs]   Predicted Labels:       bird
horse       frog        dog       bird       frog       ship       frog      horse
cat       frog       frog      plane        cat        cat        car
[epoch:8/10  iter=3000  elapsed_time=  613 secs]   Loss: 1.085



[epoch:9/10  iter=1000  elapsed_time=  642 secs]   Ground Truth:            dog
car      truck      horse        car       deer       bird       bird       ship
car        dog      truck       bird      truck       frog      plane
[epoch:9/10  iter=1000  elapsed_time=  642 secs]   Predicted Labels:       frog
car       frog       deer        car       deer       bird        dog       ship
truck        dog      truck       bird      truck        cat      horse
[epoch:9/10  iter=1000  elapsed_time=  642 secs]   Loss: 1.060



[epoch:9/10  iter=2000  elapsed_time=  665 secs]   Ground Truth:            dog
ship        dog       ship      truck        car       ship       deer       bird
truck        car       bird       deer       bird       frog        dog
[epoch:9/10  iter=2000  elapsed_time=  665 secs]   Predicted Labels:        cat
ship        dog       ship      truck        car       ship        cat       bird
truck        car       bird       deer       frog       frog       bird
[epoch:9/10  iter=2000  elapsed_time=  665 secs]   Loss: 1.055



[epoch:9/10  iter=3000  elapsed_time=  689 secs]   Ground Truth:           ship
frog       frog      truck      horse        car      plane       frog        car
plane      truck        car       bird        dog       ship        dog
[epoch:9/10  iter=3000  elapsed_time=  689 secs]   Predicted Labels:       ship
deer       frog      truck      horse      truck      plane       deer       ship
horse      truck      truck       bird        dog       ship        dog
[epoch:9/10  iter=3000  elapsed_time=  689 secs]   Loss: 1.054



[epoch:10/10  iter=1000  elapsed_time=  718 secs]   Ground Truth:           deer
deer        cat      plane       frog       bird        car      horse       bird
dog        car       deer      truck       ship       deer       frog
[epoch:10/10  iter=1000  elapsed_time=  718 secs]   Predicted Labels:       deer
deer       bird      plane       frog      plane        car      horse       deer
deer        car       deer      truck       ship       frog       frog
[epoch:10/10  iter=1000  elapsed_time=  718 secs]   Loss: 1.033



[epoch:10/10  iter=2000  elapsed_time=  745 secs]   Ground Truth:            car
ship        car       frog       bird      truck      truck      horse       frog
horse       deer      plane       frog        car       deer      horse
[epoch:10/10  iter=2000  elapsed_time=  745 secs]   Predicted Labels:        car
ship        car        dog      horse      truck      plane        cat       frog
horse        cat      plane       frog        car       deer      horse
[epoch:10/10  iter=2000  elapsed_time=  745 secs]   Loss: 1.016
```

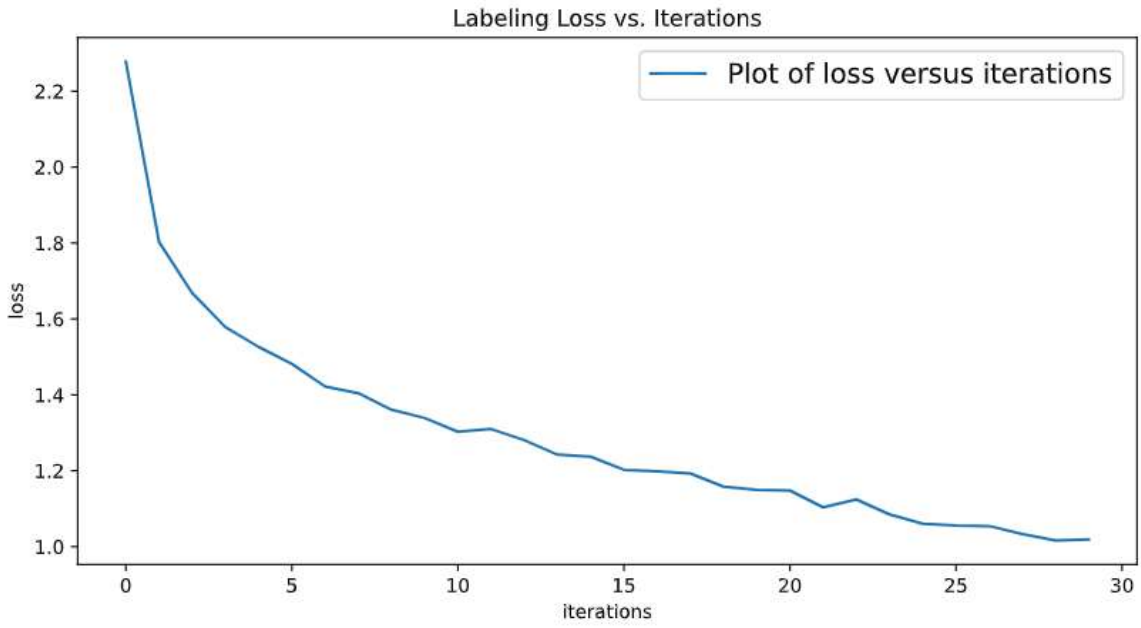
```
[epoch:10/10  iter=3000  elapsed_time=  777 secs]   Ground Truth:            cat
horse       deer        cat        car       ship      horse      horse      plane
dog       bird       frog       ship       ship      horse      plane
[epoch:10/10  iter=3000  elapsed_time=  777 secs]   Predicted Labels:        dog
horse       deer        cat        car       ship      horse      horse      plane
dog       bird       frog       ship       ship      horse       ship
[epoch:10/10  iter=3000  elapsed_time=  777 secs]   Loss: 1.019

Finished Training
```



```
Prediction accuracy for plane : 65 %
Prediction accuracy for   car : 79 %
Prediction accuracy for  bird : 37 %
Prediction accuracy for   cat : 52 %
Prediction accuracy for  deer : 61 %
Prediction accuracy for   dog : 35 %
Prediction accuracy for  frog : 68 %
Prediction accuracy for horse : 70 %
Prediction accuracy for  ship : 81 %
Prediction accuracy for truck : 77 %


Overall accuracy of the network on the 10000 test images: 62 %


Displaying the confusion matrix:
```

	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane:	65.40	4.10	2.70	2.20	2.70	0.30	1.00	1.30	12.20	8.10
car:	1.40	79.10	0.20	0.40	0.50	0.10	0.60	0.10	2.40	15.20
bird:	9.80	2.20	37.30	11.20	17.30	5.20	5.00	4.50	4.30	3.20
cat:	2.90	1.40	5.00	52.40	10.10	7.90	6.70	5.40	2.80	5.40
deer:	2.50	0.80	5.80	7.60	61.60	2.20	5.60	11.00	1.60	1.30
dog:	1.70	0.80	5.10	31.20	8.90	35.50	2.40	8.80	2.70	2.90
frog:	0.60	1.70	4.00	8.50	10.20	0.40	68.20	1.60	1.50	3.30
horse:	1.40	0.60	2.00	7.10	8.10	3.20	1.00	70.00	0.60	6.00
ship:	4.90	5.40	0.20	0.90	0.70	0.00	0.50	0.80	81.90	4.70
truck:	2.30	12.30	0.40	1.60	0.70	0.00	0.80	1.10	3.10	77.70

2.1 Train loss curve



2.2 Confusion Matrix

	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Plane	65.40	4.10	2.70	2.20	2.70	0.30	1.00	1.30	12.20	8.10
Car	1.40	79.10	0.20	0.40	0.50	0.10	0.60	0.10	2.40	15.20
Bird	9.80	2.20	37.30	11.20	17.30	5.20	5.00	4.50	4.30	3.20
Cat	2.90	1.40	5.00	52.40	10.10	7.90	6.70	5.40	2.80	5.40
Deer	2.50	0.80	5.80	7.60	61.60	2.20	5.60	11.00	1.60	1.30
Dog	1.70	0.80	5.10	31.20	8.90	35.50	2.40	8.80	2.70	2.90
Frog	0.60	1.70	4.00	8.50	10.20	0.40	68.20	1.60	1.50	3.30
Horse	1.40	0.60	2.00	7.10	8.10	3.20	1.00	70.00	0.60	6.00
Ship	4.90	5.40	0.20	0.90	0.70	0.00	0.50	0.80	81.90	4.70
Truck	2.30	12.30	0.40	1.60	0.70	0.00	0.80	1.10	3.10	77.70

3. BMENet with stride

In [3]:
```
# this code is mainly borrowed from DLstudio

dls = DLStudio(
#                   dataroot = "/home/kak/ImageDatasets/CIFAR-10/",
                  dataroot = "./../data/CIFAR-10/",
                  image_size = [32,32],
                  path_saved_model = "./saved_model",
                  momentum = 0.9,
                  learning_rate = 1e-4,
                  epochs = 6,
                  batch_size = 4,
                  classes = ('plane','car','bird','cat','deer','dog','frog','horse','
                  use_gpu = True,
              )

bme_net = dls.BMEnet(dls, skip_connections=True, depth=8)          ## if you want to

bme_net.load_cifar_10_dataset()

## display network properties
number_of_learnable_params = sum(p.numel() for p in bme_net.parameters() if p.requi
print("\n\nThe number of learnable parameters in the model: %d" % number_of_learnab

## training and testing
bme_net.run_code_for_training(bme_net, display_images=False)
bme_net.run_code_for_testing(bme_net, display_images=False)
```

```
ter=3000  elapsed_time=   93 secs]   Loss: 1.641


[epoch:1/6  iter=4000  elapsed_time=  122 secs]   Ground Truth:          horse
plane       ship        dog
[epoch:1/6  iter=4000  elapsed_time=  122 secs]   Predicted Labels:       ship
plane       ship       frog
[epoch:1/6  iter=4000  elapsed_time=  122 secs]   Loss: 1.556


[epoch:1/6  iter=5000  elapsed_time=  151 secs]   Ground Truth:          plane
bird      plane        car
[epoch:1/6  iter=5000  elapsed_time=  151 secs]   Predicted Labels:      plane
frog        dog        car
[epoch:1/6  iter=5000  elapsed_time=  151 secs]   Loss: 1.478


[epoch:1/6  iter=6000  elapsed_time=  181 secs]   Ground Truth:          plane
ship       deer        cat
[epoch:1/6  iter=6000  elapsed_time=  181 secs]   Predicted Labels:       bird
ship        dog        cat
[epoch:1/6  iter=6000  elapsed_time=  181 secs]   Loss: 1.384


[epoch:1/6  iter=7000  elapsed_time=  210 secs]   Ground Truth:           ship
horse       deer        cat
[epoch:1/6  iter=7000  elapsed_time=  210 secs]   Predicted Labels:        car
horse       bird        dog
[epoch:1/6  iter=7000  elapsed_time=  210 secs]   Loss: 1.332


[epoch:1/6  iter=8000  elapsed_time=  239 secs]   Ground Truth:          horse
plane       frog       bird
[epoch:1/6  iter=8000  elapsed_time=  239 secs]   Predicted Labels:      horse
plane       frog      plane
[epoch:1/6  iter=8000  elapsed_time=  239 secs]   Loss: 1.234


[epoch:1/6  iter=9000  elapsed_time=  268 secs]   Ground Truth:           dog
plane       deer        cat
[epoch:1/6  iter=9000  elapsed_time=  268 secs]   Predicted Labels:        dog
plane       ship      plane
[epoch:1/6  iter=9000  elapsed_time=  268 secs]   Loss: 1.208


[epoch:1/6  iter=10000  elapsed_time=  298 secs]   Ground Truth:          truck
horse        dog      horse
[epoch:1/6  iter=10000  elapsed_time=  298 secs]   Predicted Labels:      truck
truck        dog       frog
[epoch:1/6  iter=10000  elapsed_time=  298 secs]   Loss: 1.151


[epoch:1/6  iter=11000  elapsed_time=  328 secs]   Ground Truth:           frog
dog        dog        car
[epoch:1/6  iter=11000  elapsed_time=  328 secs]   Predicted Labels:       frog
frog        dog        car
```

```
[epoch:1/6  iter=11000  elapsed_time=  328 secs]   Loss: 1.153


[epoch:1/6  iter=12000  elapsed_time=  357 secs]   Ground Truth:            cat
frog        dog        dog
[epoch:1/6  iter=12000  elapsed_time=  357 secs]   Predicted Labels:        cat
frog        cat      horse
[epoch:1/6  iter=12000  elapsed_time=  357 secs]   Loss: 1.069



[epoch:2/6  iter=1000  elapsed_time=  406 secs]   Ground Truth:            car
horse        dog       ship
[epoch:2/6  iter=1000  elapsed_time=  406 secs]   Predicted Labels:        car
horse        dog       ship
[epoch:2/6  iter=1000  elapsed_time=  406 secs]   Loss: 0.895


[epoch:2/6  iter=2000  elapsed_time=  436 secs]   Ground Truth:            cat
dog        dog       frog
[epoch:2/6  iter=2000  elapsed_time=  436 secs]   Predicted Labels:       deer
cat        cat        car
[epoch:2/6  iter=2000  elapsed_time=  436 secs]   Loss: 0.883


[epoch:2/6  iter=3000  elapsed_time=  466 secs]   Ground Truth:          truck
truck        car       deer
[epoch:2/6  iter=3000  elapsed_time=  466 secs]   Predicted Labels:      truck
truck        car       deer
[epoch:2/6  iter=3000  elapsed_time=  466 secs]   Loss: 0.836


[epoch:2/6  iter=4000  elapsed_time=  497 secs]   Ground Truth:           deer
truck      plane        car
[epoch:2/6  iter=4000  elapsed_time=  497 secs]   Predicted Labels:      horse
truck      plane        car
[epoch:2/6  iter=4000  elapsed_time=  497 secs]   Loss: 0.835


[epoch:2/6  iter=5000  elapsed_time=  527 secs]   Ground Truth:          truck
plane      truck        car
[epoch:2/6  iter=5000  elapsed_time=  527 secs]   Predicted Labels:      truck
plane       bird        car
[epoch:2/6  iter=5000  elapsed_time=  527 secs]   Loss: 0.797


[epoch:2/6  iter=6000  elapsed_time=  557 secs]   Ground Truth:            car
plane       deer      horse
[epoch:2/6  iter=6000  elapsed_time=  557 secs]   Predicted Labels:        car
plane       deer      horse
[epoch:2/6  iter=6000  elapsed_time=  557 secs]   Loss: 0.842


[epoch:2/6  iter=7000  elapsed_time=  586 secs]   Ground Truth:            cat
deer        cat        cat
[epoch:2/6  iter=7000  elapsed_time=  586 secs]   Predicted Labels:       frog
```

```
deer       deer        cat
[epoch:2/6  iter=7000  elapsed_time=  586 secs]   Loss: 0.794


[epoch:2/6  iter=8000  elapsed_time=  616 secs]   Ground Truth:          truck
dog       ship       frog
[epoch:2/6  iter=8000  elapsed_time=  616 secs]   Predicted Labels:      truck
dog       ship       frog
[epoch:2/6  iter=8000  elapsed_time=  616 secs]   Loss: 0.774


[epoch:2/6  iter=9000  elapsed_time=  646 secs]   Ground Truth:            car
horse        car       frog
[epoch:2/6  iter=9000  elapsed_time=  646 secs]   Predicted Labels:      truck
horse        car      plane
[epoch:2/6  iter=9000  elapsed_time=  646 secs]   Loss: 0.801


[epoch:2/6  iter=10000  elapsed_time=  675 secs]   Ground Truth:            dog
bird       frog      truck
[epoch:2/6  iter=10000  elapsed_time=  675 secs]   Predicted Labels:        cat
cat       frog      truck
[epoch:2/6  iter=10000  elapsed_time=  675 secs]   Loss: 0.760


[epoch:2/6  iter=11000  elapsed_time=  705 secs]   Ground Truth:          horse
frog        dog        car
[epoch:2/6  iter=11000  elapsed_time=  705 secs]   Predicted Labels:      horse
frog        dog        car
[epoch:2/6  iter=11000  elapsed_time=  705 secs]   Loss: 0.732


[epoch:2/6  iter=12000  elapsed_time=  734 secs]   Ground Truth:           deer
truck        cat      horse
[epoch:2/6  iter=12000  elapsed_time=  734 secs]   Predicted Labels:        cat
horse      horse      horse
[epoch:2/6  iter=12000  elapsed_time=  734 secs]   Loss: 0.703



[epoch:3/6  iter=1000  elapsed_time=  781 secs]   Ground Truth:          truck
bird      truck        cat
[epoch:3/6  iter=1000  elapsed_time=  781 secs]   Predicted Labels:      truck
bird      truck        cat
[epoch:3/6  iter=1000  elapsed_time=  781 secs]   Loss: 0.462


[epoch:3/6  iter=2000  elapsed_time=  811 secs]   Ground Truth:          ship
plane        dog       bird
[epoch:3/6  iter=2000  elapsed_time=  811 secs]   Predicted Labels:       ship
plane        dog       bird
[epoch:3/6  iter=2000  elapsed_time=  811 secs]   Loss: 0.471


[epoch:3/6  iter=3000  elapsed_time=  840 secs]   Ground Truth:          ship
ship       ship      plane
```

```
[epoch:3/6  iter=3000  elapsed_time=  840 secs]   Predicted Labels:       ship
ship       ship      plane
[epoch:3/6  iter=3000  elapsed_time=  840 secs]   Loss: 0.459


[epoch:3/6  iter=4000  elapsed_time=  869 secs]   Ground Truth:           deer
ship       frog       bird
[epoch:3/6  iter=4000  elapsed_time=  869 secs]   Predicted Labels:       deer
ship       frog       bird
[epoch:3/6  iter=4000  elapsed_time=  869 secs]   Loss: 0.465


[epoch:3/6  iter=5000  elapsed_time=  899 secs]   Ground Truth:            cat
horse       frog        dog
[epoch:3/6  iter=5000  elapsed_time=  899 secs]   Predicted Labels:        cat
horse       frog        dog
[epoch:3/6  iter=5000  elapsed_time=  899 secs]   Loss: 0.463


[epoch:3/6  iter=6000  elapsed_time=  929 secs]   Ground Truth:          horse
dog      horse      horse
[epoch:3/6  iter=6000  elapsed_time=  929 secs]   Predicted Labels:       deer
dog        dog      horse
[epoch:3/6  iter=6000  elapsed_time=  929 secs]   Loss: 0.474


[epoch:3/6  iter=7000  elapsed_time=  960 secs]   Ground Truth:           frog
horse       deer       bird
[epoch:3/6  iter=7000  elapsed_time=  960 secs]   Predicted Labels:       frog
horse      horse       bird
[epoch:3/6  iter=7000  elapsed_time=  960 secs]   Loss: 0.485


[epoch:3/6  iter=8000  elapsed_time=  990 secs]   Ground Truth:          plane
car      horse       deer
[epoch:3/6  iter=8000  elapsed_time=  990 secs]   Predicted Labels:      plane
car      horse       deer
[epoch:3/6  iter=8000  elapsed_time=  990 secs]   Loss: 0.492


[epoch:3/6  iter=9000  elapsed_time= 1021 secs]   Ground Truth:           ship
horse        dog      plane
[epoch:3/6  iter=9000  elapsed_time= 1021 secs]   Predicted Labels:       ship
horse      plane      plane
[epoch:3/6  iter=9000  elapsed_time= 1021 secs]   Loss: 0.476


[epoch:3/6  iter=10000  elapsed_time= 1052 secs]   Ground Truth:           frog
horse        dog        dog
[epoch:3/6  iter=10000  elapsed_time= 1052 secs]   Predicted Labels:       frog
horse        dog        dog
[epoch:3/6  iter=10000  elapsed_time= 1052 secs]   Loss: 0.499


[epoch:3/6  iter=11000  elapsed_time= 1083 secs]   Ground Truth:            cat
car       ship        cat
```

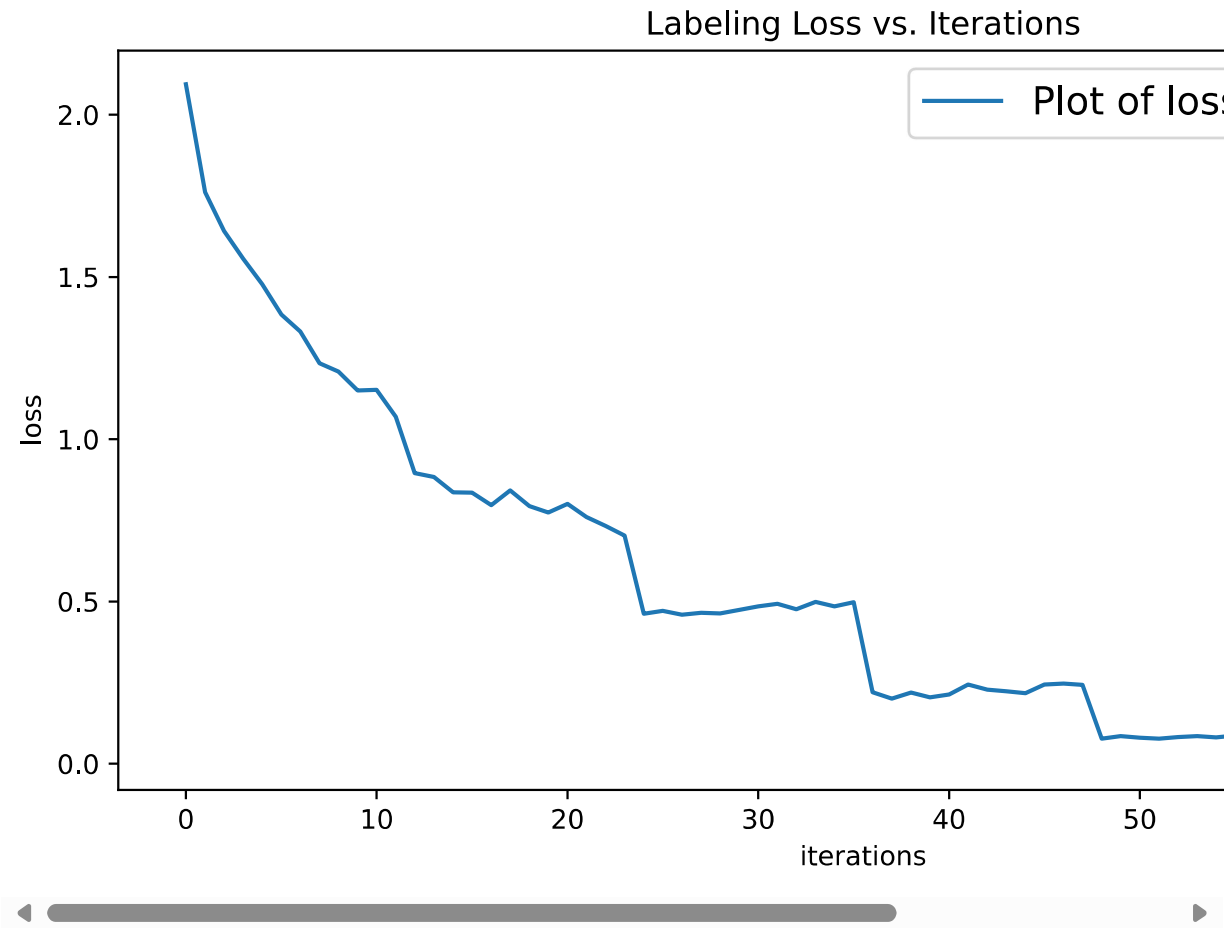
```
[epoch:3/6  iter=11000  elapsed_time= 1083 secs]   Predicted Labels:        cat
car       ship        cat
[epoch:3/6  iter=11000  elapsed_time= 1083 secs]   Loss: 0.485


[epoch:3/6  iter=12000  elapsed_time= 1113 secs]   Ground Truth:            dog
car       ship        dog
[epoch:3/6  iter=12000  elapsed_time= 1113 secs]   Predicted Labels:        dog
car       ship        dog
[epoch:3/6  iter=12000  elapsed_time= 1113 secs]   Loss: 0.498



[epoch:4/6  iter=1000  elapsed_time= 1162 secs]   Ground Truth:           bird
frog       bird       deer
[epoch:4/6  iter=1000  elapsed_time= 1162 secs]   Predicted Labels:       bird
frog      horse       deer
[epoch:4/6  iter=1000  elapsed_time= 1162 secs]   Loss: 0.221


[epoch:4/6  iter=2000  elapsed_time= 1192 secs]   Ground Truth:           bird
frog        dog       bird
[epoch:4/6  iter=2000  elapsed_time= 1192 secs]   Predicted Labels:       bird
frog        dog       bird
[epoch:4/6  iter=2000  elapsed_time= 1192 secs]   Loss: 0.200


[epoch:4/6  iter=3000  elapsed_time= 1222 secs]   Ground Truth:            cat
truck        car       bird
[epoch:4/6  iter=3000  elapsed_time= 1222 secs]   Predicted Labels:        cat
truck        car       bird
[epoch:4/6  iter=3000  elapsed_time= 1222 secs]   Loss: 0.219


[epoch:4/6  iter=4000  elapsed_time= 1253 secs]   Ground Truth:            cat
cat       ship       ship
[epoch:4/6  iter=4000  elapsed_time= 1253 secs]   Predicted Labels:        cat
cat       ship       ship
[epoch:4/6  iter=4000  elapsed_time= 1253 secs]   Loss: 0.204


[epoch:4/6  iter=5000  elapsed_time= 1283 secs]   Ground Truth:            cat
plane        car       frog
[epoch:4/6  iter=5000  elapsed_time= 1283 secs]   Predicted Labels:        cat
plane        car       frog
[epoch:4/6  iter=5000  elapsed_time= 1283 secs]   Loss: 0.213


[epoch:4/6  iter=6000  elapsed_time= 1313 secs]   Ground Truth:          horse
bird       ship       ship
[epoch:4/6  iter=6000  elapsed_time= 1313 secs]   Predicted Labels:      horse
bird       ship       ship
[epoch:4/6  iter=6000  elapsed_time= 1313 secs]   Loss: 0.244


[epoch:4/6  iter=7000  elapsed_time= 1343 secs]   Ground Truth:          plane
```

```
bird      plane       bird
[epoch:4/6  iter=7000  elapsed_time= 1343 secs]   Predicted Labels:      plane
bird      plane       bird
[epoch:4/6  iter=7000  elapsed_time= 1343 secs]   Loss: 0.228


[epoch:4/6  iter=8000  elapsed_time= 1373 secs]   Ground Truth:           bird
car      plane       ship
[epoch:4/6  iter=8000  elapsed_time= 1373 secs]   Predicted Labels:       bird
car      plane       ship
[epoch:4/6  iter=8000  elapsed_time= 1373 secs]   Loss: 0.223


[epoch:4/6  iter=9000  elapsed_time= 1403 secs]   Ground Truth:            car
horse       deer       frog
[epoch:4/6  iter=9000  elapsed_time= 1403 secs]   Predicted Labels:        car
horse       deer       frog
[epoch:4/6  iter=9000  elapsed_time= 1403 secs]   Loss: 0.217


[epoch:4/6  iter=10000  elapsed_time= 1434 secs]   Ground Truth:            dog
deer       bird        dog
[epoch:4/6  iter=10000  elapsed_time= 1434 secs]   Predicted Labels:        dog
deer       bird        dog
[epoch:4/6  iter=10000  elapsed_time= 1434 secs]   Loss: 0.244


[epoch:4/6  iter=11000  elapsed_time= 1463 secs]   Ground Truth:          horse
cat       frog       deer
[epoch:4/6  iter=11000  elapsed_time= 1463 secs]   Predicted Labels:        cat
cat       frog       deer
[epoch:4/6  iter=11000  elapsed_time= 1463 secs]   Loss: 0.247


[epoch:4/6  iter=12000  elapsed_time= 1494 secs]   Ground Truth:          horse
cat       deer       ship
[epoch:4/6  iter=12000  elapsed_time= 1494 secs]   Predicted Labels:      horse
cat       deer       ship
[epoch:4/6  iter=12000  elapsed_time= 1494 secs]   Loss: 0.243



[epoch:5/6  iter=1000  elapsed_time= 1542 secs]   Ground Truth:          deer
bird       frog        dog
[epoch:5/6  iter=1000  elapsed_time= 1542 secs]   Predicted Labels:       deer
bird       frog        dog
[epoch:5/6  iter=1000  elapsed_time= 1542 secs]   Loss: 0.078


[epoch:5/6  iter=2000  elapsed_time= 1571 secs]   Ground Truth:          deer
horse       frog       ship
[epoch:5/6  iter=2000  elapsed_time= 1571 secs]   Predicted Labels:       deer
horse       frog       ship
[epoch:5/6  iter=2000  elapsed_time= 1571 secs]   Loss: 0.085
```

```
[epoch:5/6  iter=3000  elapsed_time= 1600 secs]   Ground Truth:           frog
plane       bird       ship
[epoch:5/6  iter=3000  elapsed_time= 1600 secs]   Predicted Labels:       frog
plane       bird       ship
[epoch:5/6  iter=3000  elapsed_time= 1600 secs]   Loss: 0.080


[epoch:5/6  iter=4000  elapsed_time= 1629 secs]   Ground Truth:          horse
dog       frog       deer
[epoch:5/6  iter=4000  elapsed_time= 1629 secs]   Predicted Labels:      horse
dog       frog       deer
[epoch:5/6  iter=4000  elapsed_time= 1629 secs]   Loss: 0.077


[epoch:5/6  iter=5000  elapsed_time= 1659 secs]   Ground Truth:           frog
ship        dog       ship
[epoch:5/6  iter=5000  elapsed_time= 1659 secs]   Predicted Labels:       frog
ship        dog       ship
[epoch:5/6  iter=5000  elapsed_time= 1659 secs]   Loss: 0.082


[epoch:5/6  iter=6000  elapsed_time= 1689 secs]   Ground Truth:           bird
truck        dog      truck
[epoch:5/6  iter=6000  elapsed_time= 1689 secs]   Predicted Labels:       bird
truck        dog      truck
[epoch:5/6  iter=6000  elapsed_time= 1689 secs]   Loss: 0.085


[epoch:5/6  iter=7000  elapsed_time= 1718 secs]   Ground Truth:           frog
frog      horse      truck
[epoch:5/6  iter=7000  elapsed_time= 1718 secs]   Predicted Labels:       frog
frog      horse      truck
[epoch:5/6  iter=7000  elapsed_time= 1718 secs]   Loss: 0.081


[epoch:5/6  iter=8000  elapsed_time= 1748 secs]   Ground Truth:           ship
truck       ship       bird
[epoch:5/6  iter=8000  elapsed_time= 1748 secs]   Predicted Labels:       ship
truck       ship       bird
[epoch:5/6  iter=8000  elapsed_time= 1748 secs]   Loss: 0.087


[epoch:5/6  iter=9000  elapsed_time= 1778 secs]   Ground Truth:           frog
plane      plane        car
[epoch:5/6  iter=9000  elapsed_time= 1778 secs]   Predicted Labels:       frog
plane      plane        car
[epoch:5/6  iter=9000  elapsed_time= 1778 secs]   Loss: 0.084


[epoch:5/6  iter=10000  elapsed_time= 1807 secs]   Ground Truth:           bird
cat      plane       frog
[epoch:5/6  iter=10000  elapsed_time= 1807 secs]   Predicted Labels:       bird
cat      plane       frog
[epoch:5/6  iter=10000  elapsed_time= 1807 secs]   Loss: 0.081
```

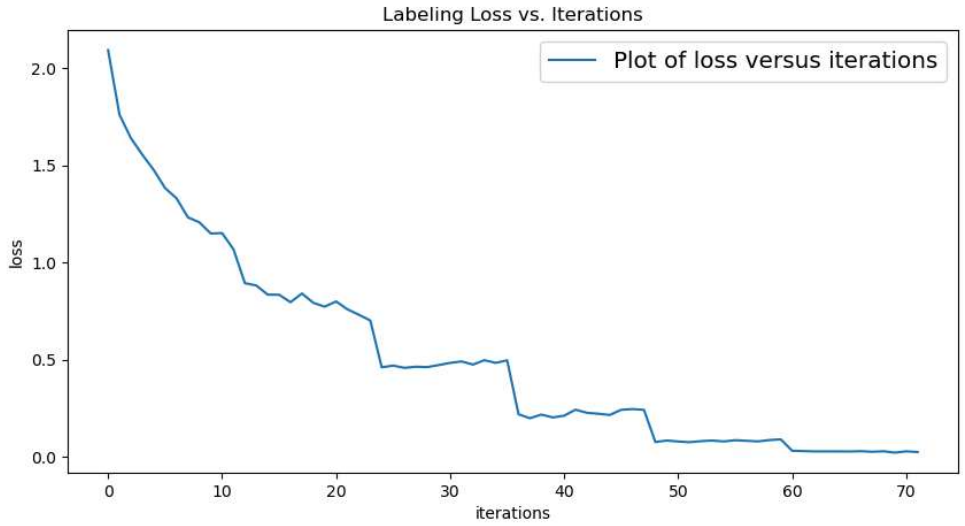
```
[epoch:5/6  iter=11000  elapsed_time= 1836 secs]   Ground Truth:          truck
deer      truck       frog
[epoch:5/6  iter=11000  elapsed_time= 1836 secs]   Predicted Labels:      truck
deer      truck       frog
[epoch:5/6  iter=11000  elapsed_time= 1836 secs]   Loss: 0.088


[epoch:5/6  iter=12000  elapsed_time= 1866 secs]   Ground Truth:            dog
truck      plane       frog
[epoch:5/6  iter=12000  elapsed_time= 1866 secs]   Predicted Labels:        dog
truck      plane       frog
[epoch:5/6  iter=12000  elapsed_time= 1866 secs]   Loss: 0.091



[epoch:6/6  iter=1000  elapsed_time= 1912 secs]   Ground Truth:            cat
frog        car      horse
[epoch:6/6  iter=1000  elapsed_time= 1912 secs]   Predicted Labels:        cat
frog        car      horse
[epoch:6/6  iter=1000  elapsed_time= 1912 secs]   Loss: 0.033


[epoch:6/6  iter=2000  elapsed_time= 1941 secs]   Ground Truth:          truck
ship       deer      plane
[epoch:6/6  iter=2000  elapsed_time= 1941 secs]   Predicted Labels:      truck
ship       deer      plane
[epoch:6/6  iter=2000  elapsed_time= 1941 secs]   Loss: 0.031


[epoch:6/6  iter=3000  elapsed_time= 1970 secs]   Ground Truth:          truck
cat      truck      truck
[epoch:6/6  iter=3000  elapsed_time= 1970 secs]   Predicted Labels:      truck
cat      truck      truck
[epoch:6/6  iter=3000  elapsed_time= 1970 secs]   Loss: 0.029


[epoch:6/6  iter=4000  elapsed_time= 2001 secs]   Ground Truth:          plane
deer       ship       deer
[epoch:6/6  iter=4000  elapsed_time= 2001 secs]   Predicted Labels:      plane
deer       ship       deer
[epoch:6/6  iter=4000  elapsed_time= 2001 secs]   Loss: 0.030


[epoch:6/6  iter=5000  elapsed_time= 2030 secs]   Ground Truth:          horse
cat      horse       frog
[epoch:6/6  iter=5000  elapsed_time= 2030 secs]   Predicted Labels:      horse
cat      horse       frog
[epoch:6/6  iter=5000  elapsed_time= 2030 secs]   Loss: 0.029


[epoch:6/6  iter=6000  elapsed_time= 2061 secs]   Ground Truth:           bird
cat      plane        dog
[epoch:6/6  iter=6000  elapsed_time= 2061 secs]   Predicted Labels:       bird
cat       ship        dog
[epoch:6/6  iter=6000  elapsed_time= 2061 secs]   Loss: 0.029
```

```
[epoch:6/6  iter=7000  elapsed_time= 2090 secs]   Ground Truth:           frog
plane        car      truck
[epoch:6/6  iter=7000  elapsed_time= 2090 secs]   Predicted Labels:       frog
plane        car      truck
[epoch:6/6  iter=7000  elapsed_time= 2090 secs]   Loss: 0.031


[epoch:6/6  iter=8000  elapsed_time= 2118 secs]   Ground Truth:           bird
horse      horse        dog
[epoch:6/6  iter=8000  elapsed_time= 2118 secs]   Predicted Labels:       bird
horse      horse        dog
[epoch:6/6  iter=8000  elapsed_time= 2118 secs]   Loss: 0.028


[epoch:6/6  iter=9000  elapsed_time= 2148 secs]   Ground Truth:           deer
dog      plane      horse
[epoch:6/6  iter=9000  elapsed_time= 2148 secs]   Predicted Labels:       deer
dog      plane      horse
[epoch:6/6  iter=9000  elapsed_time= 2148 secs]   Loss: 0.030


[epoch:6/6  iter=10000  elapsed_time= 2177 secs]   Ground Truth:          plane
frog        dog       ship
[epoch:6/6  iter=10000  elapsed_time= 2177 secs]   Predicted Labels:      plane
frog        dog       ship
[epoch:6/6  iter=10000  elapsed_time= 2177 secs]   Loss: 0.023


[epoch:6/6  iter=11000  elapsed_time= 2206 secs]   Ground Truth:          horse
plane        car       bird
[epoch:6/6  iter=11000  elapsed_time= 2206 secs]   Predicted Labels:      horse
plane        car       bird
[epoch:6/6  iter=11000  elapsed_time= 2206 secs]   Loss: 0.030


[epoch:6/6  iter=12000  elapsed_time= 2236 secs]   Ground Truth:           frog
deer      truck      truck
[epoch:6/6  iter=12000  elapsed_time= 2236 secs]   Predicted Labels:       frog
deer      truck      truck
[epoch:6/6  iter=12000  elapsed_time= 2236 secs]   Loss: 0.026

Finished Training
```


Labeling Loss vs. Iterations
Plot of loss
2.0
1.5
1.0
0.5
0.0
loss
0
10
20
30
40
50
iterations

```
/home/chen4126/ece60146/HW6/./../DLStudio-2.5.1/DLStudio/DLStudio.py:2964: FutureWar
ning: You are using `torch.load` with `weights_only=False` (the current default valu
e), which uses the default pickle module implicitly. It is possible to construct mal
icious pickle data which will execute arbitrary code during unpickling (See https://
github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details).
In a future release, the default value for `weights_only` will be flipped to `True`.
This limits the functions that could be executed during unpickling. Arbitrary object
s will no longer be allowed to be loaded via this mode unless they are explicitly al
lowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you s
tart setting `weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues related to this ex
perimental feature.
  net.load_state_dict(torch.load(self.path_saved_model))


[i=1000:] Ground Truth:      frog  bird  bird truck
[i=1000:] Predicted Labels:  frog  deer  bird truck


[i=2000:] Ground Truth:     horse   car  bird  frog
[i=2000:] Predicted Labels: horse   car  bird  frog
Prediction accuracy for plane : 78 %
Prediction accuracy for   car : 93 %
Prediction accuracy for  bird : 73 %
Prediction accuracy for   cat : 67 %
Prediction accuracy for  deer : 71 %
Prediction accuracy for   dog : 68 %
Prediction accuracy for  frog : 86 %
Prediction accuracy for horse : 82 %
Prediction accuracy for  ship : 87 %
Prediction accuracy for truck : 84 %



Overall accuracy of the network on the 10000 test images: 79 %


Displaying the confusion matrix:

           plane    car   bird    cat   deer    dog   frog  horse   ship  truck

 plane:    78.10   1.60   7.50   2.30   0.70   0.50   1.20   0.50   4.60   3.00
   car:     0.30  93.20   0.10   0.60   0.00   0.10   1.00   0.10   1.40   3.20
  bird:     3.60   0.60  73.90   6.20   4.40   4.20   3.90   1.30   1.00   0.90
   cat:     1.00   0.70   6.10  67.60   3.10  12.00   5.40   2.30   1.20   0.60
  deer:     1.30   0.60   8.10   6.80  71.30   2.30   4.80   4.40   0.40   0.00
   dog:     0.90   0.20   4.00  18.70   1.50  68.20   2.20   3.50   0.40   0.40
  frog:     0.40   0.60   3.70   5.70   1.20   1.00  86.70   0.10   0.40   0.20
 horse:     1.40   0.10   2.60   5.20   2.80   3.60   0.60  82.60   0.20   0.90
  ship:     3.50   3.40   1.30   1.10   0.30   0.50   0.70   0.10  87.50   1.60
 truck:     2.00   7.30   0.80   2.10   0.30   0.40   0.60   0.80   1.50  84.20
```

3.1 Train loss curve



3.2 Confusion Matrix

	Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Plane	78.10	1.60	7.50	2.30	0.70	0.50	1.20	0.50	4.60	3.00
Car	0.30	93.20	0.10	0.60	0.00	0.10	1.00	0.10	1.40	3.20
Bird	3.60	0.60	73.90	6.20	4.40	4.20	3.90	1.30	1.00	0.90
Cat	1.00	0.70	6.10	67.60	3.10	12.00	5.40	2.30	1.20	0.60
Deer	1.30	0.60	8.10	6.80	71.30	2.30	4.80	4.40	0.40	0.00
Dog	0.90	0.20	4.00	18.70	1.50	68.20	2.20	3.50	0.40	0.40
Frog	0.40	0.60	3.70	5.70	1.20	1.00	86.70	0.10	0.40	0.20
Horse	1.40	0.10	2.60	5.20	2.80	3.60	0.60	82.60	0.20	0.90
Ship	3.50	3.40	1.30	1.10	0.30	0.50	0.70	0.10	87.50	1.60
Truck	2.00	7.30	0.80	2.10	0.30	0.40	0.60	0.80	1.50	84.20

4. Table1: Overall accuracy of 2 models table

	Maxpool	Stride
Accuracy	62%	79%

5. Table2: Per class accuracy of 2 models (10x2 table)

	Maxpool	Stride
Plane	65	78
Car	79	93
Bird	37	73
Cat	52	67
Deer	61	71
Dog	35	68
Frog	68	86
Horse	70	82
Ship	81	87
Truck	77	84

6. Observations of Maxpool vs Stride

Conclusion first, the stride method significantly outperforms the maxpool method in this experiment.

The likely reason for this difference is that, while both approaches aim to reduce spatial dimensions, the stride method learns how to downsample the image while simultaneously extracting relevant features. In contrast, max pooling is a fixed operation that simply selects the most prominent value in a local region, without learning additional transformations that might preserve more discriminative information.

When using stride-based convolutions, the network jointly optimizes feature extraction and downsampling, ensuring that critical patterns are retained even as resolution decreases. This allows for a more adaptive reduction in spatial dimensions, where the network can learn to emphasize key edges, textures, or shapes based on training data.

On the other hand, MaxPooling is a static operation—it does not learn any parameters, and it simply selects the maximum value from a given region. While this can help with translation invariance, it may also result in information loss, as it discards non-maximum activations that could be useful in later layers. This rigid downsampling process might lead to a degradation in representational power, making it harder for the model to distinguish between similar classes effectively.

Given these findings, stride-based downsampling appears to be a more effective strategy for this particular architecture and dataset.

Furthermore, compared to HW5, where I used max pooling without shortcut connections, the current model shows significant improvement in performance. This suggests that both stride-based downsampling and shortcut connections contribute to better feature retention and overall model efficiency.

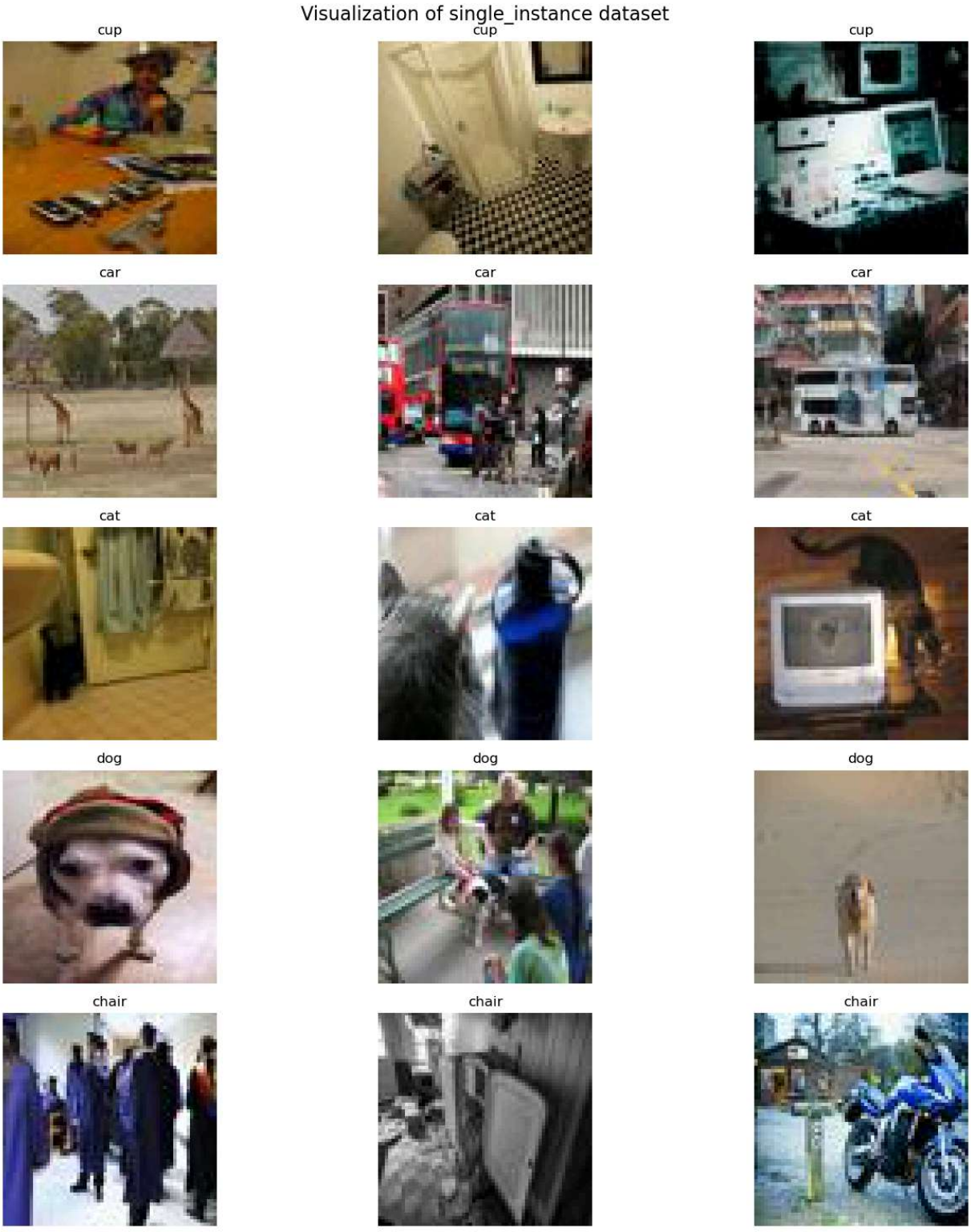
7. Skip Connections with MSCOCO

In [ ]:
```
import torch.nn as nn
# skipblock is borrowed from DLstudio
class SkipBlock(nn.Module):
    """
    Class Path:   DLStudio  ->  BMEnet  ->  SkipBlock
    """
    def __init__(self, in_ch, out_ch, downsample=False, skip_connections=True):
        super().__init__()
        self.downsample = downsample
        self.skip_connections = skip_connections
        self.in_ch = in_ch
        self.out_ch = out_ch
        self.convo1 = nn.Conv2d(in_ch, in_ch, 3, stride=1, padding=1)
        self.convo2 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(in_ch)
        self.bn2 = nn.BatchNorm2d(out_ch)
        self.in2out  =  nn.Conv2d(in_ch, out_ch, 1)
        if downsample:
            ##  Setting stride to 2 and kernel_size to 1 amounts to retaining every
            ##  other pixel in the image --- which halves the size of the image:
            self.downsampler1 = nn.Conv2d(in_ch, in_ch, 1, stride=2)
            self.downsampler2 = nn.Conv2d(out_ch, out_ch, 1, stride=2)

    def forward(self, x):
        identity = x
        out = self.convo1(x)
        out = self.bn1(out)
        out = nn.functional.relu(out)
        out = self.convo2(out)
        out = self.bn2(out)
        out = nn.functional.relu(out)
        if self.downsample:
            identity = self.downsampler1(identity)
            out = self.downsampler2(out)
        if self.skip_connections:
            if (self.in_ch == self.out_ch) and (self.downsample is False):
                out = out + identity
            elif (self.in_ch != self.out_ch) and (self.downsample is False):
                identity = self.in2out( identity )
                out = out + identity
            elif (self.in_ch != self.out_ch) and (self.downsample is True):
                out = out + torch.cat((identity, identity), dim=1)
        return out
```

```
# ------------------ HW6 NET------------------
# some of this code is borrowed from DLstudio, the code that i modify has a comment

class HW6Net(nn.Module): # nn.Module is the base class for all neural network model
    def __init__(self, num_class = 5, skip_connections=True):
        super().__init__()
        num_ds = 0

        self.depth = 6 # Since each SkipBlock uses two instances of nn.Conv2d, your

        self.initial_block = SkipBlock(3, 64, skip_connections=False)  # this is ju


        # these for loop block is mainly borrow from DLstudio, but I added a additi
        self.skip64_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip64_arr.append(SkipBlock(64, 64, skip_connections=skip_connecti
        self.skip64to128ds = SkipBlock(64, 128, downsample=True, skip_connections=s
        num_ds += 1

        self.skip128_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip128_arr.append(SkipBlock(128, 128, skip_connections=skip_conne
        self.skip128to256ds = SkipBlock(128, 256, downsample=True, skip_connections
        num_ds += 1

        self.skip256_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip256_arr.append(SkipBlock(256, 256, skip_connections=skip_conne



        self.fc1 =  nn.Linear( (32// (2 ** num_ds))  *  (32//(2 ** num_ds))  *256,
        self.fc2 =  nn.Linear(512, num_class) # reduce the size of fc layers

    def forward(self, x):

        x = self.initial_block(x)  # initial block 3->64

        for skip64 in self.skip64_arr:
            x = skip64(x)
        x = self.skip64to128ds(x)
        for skip128 in self.skip128_arr:
            x = skip128(x)
        x = self.skip128to256ds(x)
        for skip256 in self.skip256_arr:
            x = skip256(x)


        x  =  x.view( x.shape[0], - 1 )
        x = nn.functional.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

7.1 5x3 images

In [19]:
```
import matplotlib.pyplot as plt
import random

def visualize_dataset(dataset_type, classes, output_dir="output_datasets"):
    fig, axes = plt.subplots(5, 3, figsize=(15, 15))  # 5 rows x 3 columns grid
    fig.suptitle(f"Visualization of {dataset_type} dataset", fontsize=16)

    image_paths = []

    for category in classes:
        category_path = os.path.join(output_dir, dataset_type, 'train', category)
        if os.path.exists(category_path) and os.listdir(category_path):
            category_images = [os.path.join(category_path, img) for img in os.listd
            random.shuffle(category_images)
            image_paths.extend(category_images[:3]) # Select 3 images per class

    image_paths = image_paths[:15]  # Ensure only 15 images for the grid

    for ax, img_path in zip(axes.ravel(), image_paths):
        img = Image.open(img_path)
        ax.imshow(img)
        ax.axis("off")
        ax.set_title(os.path.basename(os.path.dirname(img_path)))  # Display catego

    plt.tight_layout()
    plt.show()

target_classes = ["cup", "car", "cat", "dog", "chair"]

visualize_dataset("single_instance", target_classes)
```



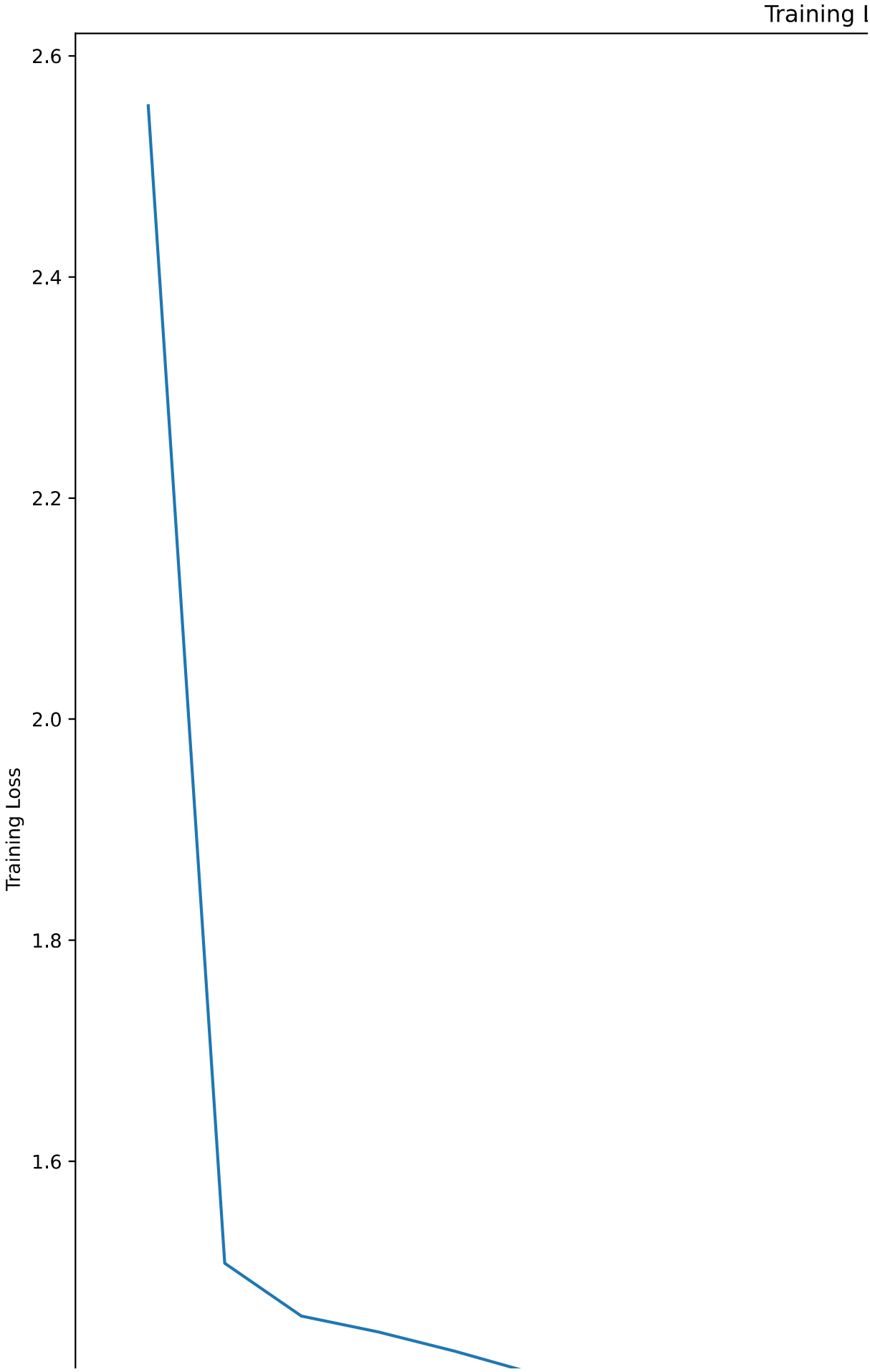
7.2 train loss curve

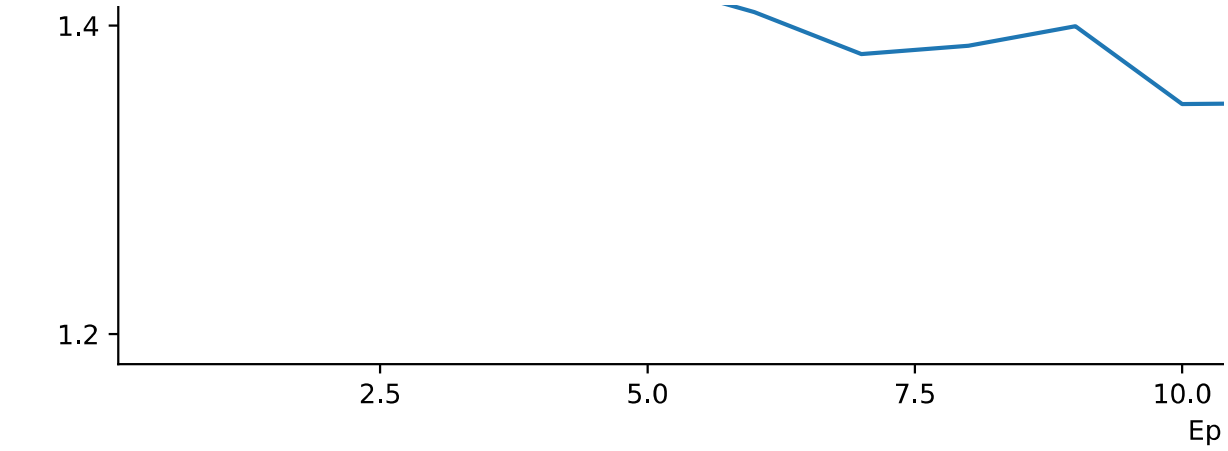
In [ ]:
```
import torch
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np
import torch.nn.functional as F
```

```python
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=5),
    transforms.Resize((32, 32)),  # Resize images to 32x32 for consistency
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])

transform_val = transforms.Compose([
    transforms.Resize((32, 32)),  # Resize images to 32x32 for consistency
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])

# Load training dataset
train_dataset = torchvision.datasets.ImageFolder(root="output_datasets/train", tran
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# Load validation dataset
val_dataset = torchvision.datasets.ImageFolder(root="output_datasets/val", transfor
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)




# Check if GPU is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

model = HW6Net().to(device)  # move model to gpu

criterion = nn.CrossEntropyLoss() # CrossEntropyLoss for multi-class classification


optimizer = torch.optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-3) # use



# ---------train loop------------

# Modify train function to store training loss
def train_model(model, train_loader, criterion, optimizer, epochs):
    train_losses = []  # Store training losses

    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        correct, total = 0, 0

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device) # move data to gp

            optimizer.zero_grad() # zero the parameter gradients to prevent accumul
            outputs = model(images) # forward pass
            loss = criterion(outputs, labels) # calculate loss
```

```
            loss.backward() # backward pass
            optimizer.step() # update weights

            running_loss += loss.item() # accumulate loss
            _, predicted = torch.max(outputs, 1) # get the class label with the hig
            total += labels.size(0) # accumulate total number of labels
            correct += (predicted == labels).sum().item() # accumulate correct prea

        train_losses.append(running_loss / len(train_loader)) # store average loss

        train_acc = 100 * correct / total
        print(f"Epoch [{epoch+1}/{epochs}], Loss: {train_losses[-1]:.4f}, Train Acc

    return train_losses

train_losses = train_model(model, train_loader, criterion, optimizer, epochs=20) #

# ---------end train loop------------

# ----- train loss curve -----
plt.figure(figsize=(15, 15))
plt.plot(range(1, len(train_losses) + 1), train_losses)
plt.xlabel("Epochs")
plt.ylabel("Training Loss")
plt.title("Training Loss Curve")
plt.show()
```

```
Using device: cuda
Epoch [1/20], Loss: 2.5548, Train Acc: 28.43%
Epoch [2/20], Loss: 1.5079, Train Acc: 35.45%
Epoch [3/20], Loss: 1.4601, Train Acc: 38.46%
Epoch [4/20], Loss: 1.4457, Train Acc: 38.84%
Epoch [5/20], Loss: 1.4284, Train Acc: 39.43%
Epoch [6/20], Loss: 1.4088, Train Acc: 40.88%
Epoch [7/20], Loss: 1.3817, Train Acc: 41.74%
Epoch [8/20], Loss: 1.3870, Train Acc: 42.13%
Epoch [9/20], Loss: 1.3996, Train Acc: 42.59%
Epoch [10/20], Loss: 1.3493, Train Acc: 44.06%
Epoch [11/20], Loss: 1.3498, Train Acc: 44.86%
Epoch [12/20], Loss: 1.3263, Train Acc: 45.61%
Epoch [13/20], Loss: 1.3254, Train Acc: 45.49%
Epoch [14/20], Loss: 1.3353, Train Acc: 45.14%
Epoch [15/20], Loss: 1.3232, Train Acc: 45.40%
Epoch [16/20], Loss: 1.3035, Train Acc: 46.54%
Epoch [17/20], Loss: 1.2842, Train Acc: 47.52%
Epoch [18/20], Loss: 1.2644, Train Acc: 48.37%
Epoch [19/20], Loss: 1.2573, Train Acc: 48.90%
Epoch [20/20], Loss: 1.2459, Train Acc: 49.38%
```

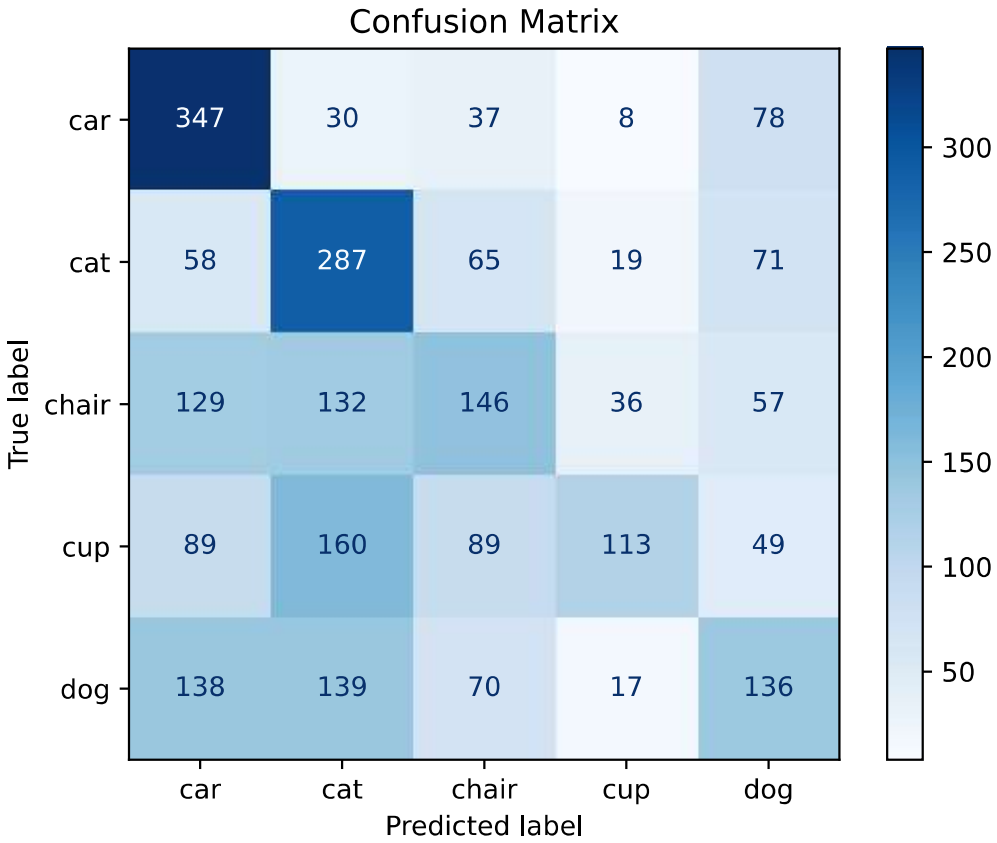

Training L
2.6
2.4
2.2
2.0
1.8
1.6
Training Loss



7.3 conf matrix 5x5

In [ ]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,accuracy_score

def evaluate_model(model, val_loader):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad(): # disable gradient calculation  to save memory for valida
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)

            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    return np.array(all_labels), np.array(all_preds)


# Get true labels and predictions
true_labels, predicted_labels = evaluate_model(model, val_loader)

# Compute overall accuracy by using sklearn
overall_acc = accuracy_score(true_labels, predicted_labels)
print(f"Overall Validation Accuracy: {overall_acc * 100:.2f}%")

# Compute confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=train_dataset.cla

# Plot confusion matrix
plt.figure(figsize=(6, 6))
disp.plot(cmap="Blues", values_format='d')
plt.title("Confusion Matrix")
plt.show()
```

```
Overall Validation Accuracy: 41.16%
<Figure size 600x600 with 0 Axes>
```



7.4 overall accuracy

Overall Validation Accuracy: 41.16%

7.5 per class accuracy 5x1 table

Class	Accuracy(%)
Car	69
Cat	57
Chair	29
Cup	22
Dog	27

7.6 observations

Initially, I expected that using 64×64 images would yield better performance. However, not only did training with 64×64 inputs take significantly longer, but the results were also comparable to 32×32 images.

One major issue I encountered was severe overfitting—the training accuracy reached 90%, while validation accuracy remained around 40%. I believe the primary reason for this is that

I'm using a 22-layer deep convolutional neural network and 2-layer fully connected layers on a relatively simple dataset, which likely causes the model to memorize training samples rather than generalize effectively.

To address this, I experimented with various hyperparameters, such as reducing the batch size, increasing the weight decay, and increasing learning rate. While these adjustments helped mitigate overfitting, I still struggled to push validation accuracy beyond 50%. (41%)

Since the architecture closely resembles the one used for CIFAR-10 (BMENet), I believe the primary issue is not the model itself. Instead, the COCO dataset poses additional challenges due to its complexity and dense object compositions. A model better suited for object detection, such as YOLO, might yield improved performance. Further investigation is required to pinpoint the underlying factors affecting performance.

Source Code

Most of the code is included in every sections above, this section will show the code of generating the COCO dataset.

In [ ]:
```python
# this code is borrow from HW4_spring2025.pdf, my code will be follow by a comment
from PIL import Image
import os
from pycocotools.coco import COCO

# Set COCO dataset paths
data_dir = os.getcwd()
#  mac users
# ann_file = os.path.join(data_dir, "annotations/instances_train2014.json")
# image_dir = os.path.join(data_dir, "train2014/train2014")

# windows users
ann_file = os.path.join(data_dir, r"annotations\instances_train2014.json")
image_dir = os.path.join(data_dir, r"train2014\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, split):
    """Saves the extracted image into a structured output directory for training/va
    img_path = os.path.join(image_dir, img_info['file_name'])
    save_dir = os.path.join(output_dir, dataset_type, split, category_name)
    os.makedirs(save_dir, exist_ok=True)

    # # Check if the file exists before opening
```

```
    # if not os.path.exists(img_path):
    #     print(f"Skipping {img_info['file_name']} - File not found.")
    #     return

    # 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
    """
    Extracts images based on object count conditions for single-instance or multi-i
    - min_instances: Minimum number of object instances required.
    - max_instances: Maximum number of object instances allowed.
    - multiple_categories: If True, selects images with multiple different object t
    """
    cat_ids = coco.getCatIds(catNms=cat_names)
    img_ids = coco.getImgIds(catIds=cat_ids)
    # Counter for extracted images
    extracted = 0


    images_to_process = [] # List to store valid images
    target_classes = ["cup", "car", "cat", "dog", "chair"] # remeber classes we are
    current_category = cat_names[0] ## e.g. current_category: person
    other_categories = [cat for cat in target_classes if cat != current_category] #


    # loop through the images
    for img_id in img_ids:
        img_info = coco.loadImgs(img_id)[0]
        # These are annotation IDs for objects detected in a specific image. (we ar
        ann_ids = coco.getAnnIds(imgIds=img_id, iscrowd=False)
        # anns includes bounding box, category ID, and segmentation, area, imageID.
        anns = coco.loadAnns(ann_ids)


        # Count objects per category
        obj_counts = {}

        obj_categories = set() # set to store object categories in the image. e.g.

        # max_area_category = None # variable to store the category with the maximu
        # max_area = 0

        for ann in anns:

            # return object category name, like obj_category: umbrella, obj_categor
            obj_category = coco.loadCats(ann['category_id'])[0]['name']

            # obj count for each category, like obj_counts: {'umbrella': 1}, obj_co
            obj_counts[obj_category] = obj_counts.get(obj_category, 0) + 1

            obj_categories.add(obj_category) # keep track of all object categories

        #     if ann['area'] > max_area: # Track category with largest area
        #         max_area = ann['area']
```

```
        #         max_area_category = obj_category

        # if max_area_category != current_category: # Skip images with largest area
        #     continue

        if dataset_type in ["single_instance", "multi_instance_same"]: # Skip image
            if any(other_cat in obj_categories for other_cat in other_categories):
                continue

        if multiple_categories:
            # Ensure multiple object categories are present(modified)
            if len(obj_counts) >= 2 and current_category in obj_counts and any(othe
                images_to_process.append(img_info) # instead of saving the image, I
                extracted += 1
        elif dataset_type == "multi_instance_same":
            # Ensure multiple instances of the same object category
            if cat_names[0] in obj_counts and obj_counts[cat_names[0]] >= min_insta
                images_to_process.append(img_info) # instead of saving the image, I
                extracted += 1
        else:
            # Ensure the number of instances falls within the desired range for sin
            if all(obj in obj_counts for obj in cat_names) and min_instances <= obj
                images_to_process.append(img_info) # instead of saving the image, I
                extracted += 1

        # Limit to 500 images per dataset for testing
        if extracted >= 2000:
            break

    train_images = images_to_process[:1500] # Split the images into training and va
    val_images = images_to_process[1500:2000] # Split the images into training and

    for img_info in train_images:   # Save training and validation images
        save_image(img_info, cat_names[0], dataset_type, 'train')

    for img_info in val_images:   # Save training and validation images
        save_image(img_info, cat_names[0], dataset_type, 'val')

# Extract Single-instance Dataset
# not enough images for the following requirements
# extract_images(["cup"], min_instances=1, max_instances=1, dataset_type="single_in
# extract_images(["car"], min_instances=1, max_instances=1, dataset_type="single_in
# extract_images(["cat"], min_instances=1, max_instances=1, dataset_type="single_in
# extract_images(["dog"], min_instances=1, max_instances=1, dataset_type="single_in
# extract_images(["chair"], min_instances=1, max_instances=1, dataset_type="single_


extract_images(["cup"], min_instances=1, dataset_type="single_instance")
extract_images(["car"], min_instances=1, dataset_type="single_instance")
extract_images(["cat"], min_instances=1, dataset_type="single_instance")
extract_images(["dog"], min_instances=1, dataset_type="single_instance")
extract_images(["chair"], min_instances=1, dataset_type="single_instance")
```