HW 6: ECE 60146 – Deep Learning

-Kaushik Karthikeyan

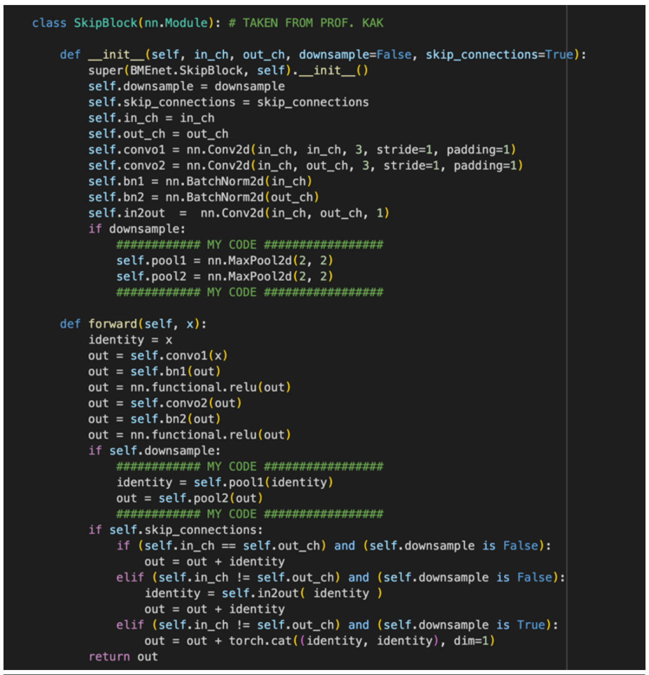
Task 1: Observation on 1-pixel kernel

A 1x1 kernel can be useful in many cases. A 1x1 kernel would essentially look at each pixel one by one without mixing spatial information. Instead, it would apply the convolution formula to each pixel by using values from different channels at that location. It is analogous to a mini-neural network layer applied to each pixel. It can also be used for downsampling, which is an important step while using skip connections.

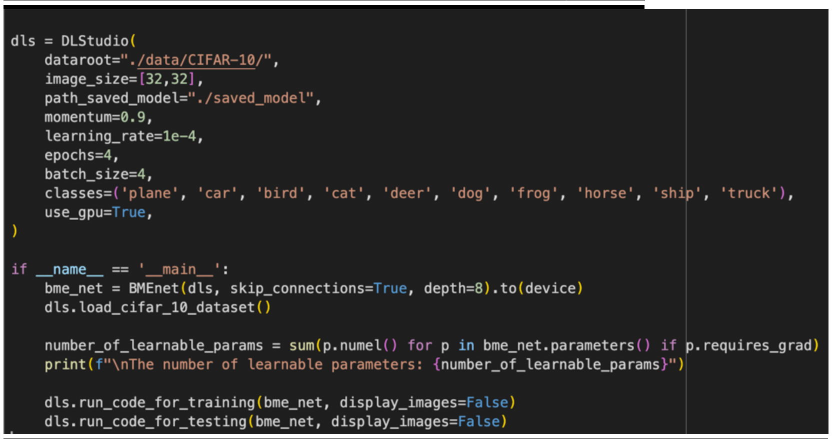
The difference between the 1 x 1 kernel used in Line (I) and the ones used in Lines (J) and (K) is that the kernel used in Line I is a standard 1-pixel kernel used in channel transformation whereas the one in Lines (J) and (K) is a kernel with a stride of 2, meaning that it would reduce the resolution by a factor of 2. The kernel used in Lines (J) and (K) does not change the number of channels as in the one in Line (I).

Task 2: BMEnet with downsampling = {Stride=2, Maxpool} and skip_connections=False

For this task, we are required to use the BMEnet model given to us in the DLStudio.py file. We understand the difference between using skip connections and not using skip connections, and also the difference between down sampling using a stride=2 convolutional layer and down sampling using a Maxpool layer. First, we implement the BMEnet model with Maxpool for down sampling with skip connections on the CIFAR-10 dataset. Below is the code for this implementation:



```
class SkipBlock(nn.Module): # TAKEN FROM PROF. KAK

    def __init__(self, in_ch, out_ch, downsample=False, skip_connections=True):
        super(BMEnet.SkipBlock, self).__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:
            ############ MY CODE #################
            self.pool1 = nn.MaxPool2d(2, 2)
            self.pool2 = nn.MaxPool2d(2, 2)
            ############ MY CODE #################

    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:
            ############ MY CODE #################
            identity = self.pool1(identity)
            out = self.pool2(out)
            ############ MY CODE #################
        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
```



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

if __name__ == '__main__':
    bme_net = BMEnet(dls, skip_connections=True, depth=8).to(device)
    dls.load_cifar_10_dataset()

    number_of_learnable_params = sum(p.numel() for p in bme_net.parameters() if p.requires_grad)
    print(f"\nThe number of learnable parameters: {number_of_learnable_params}")

    dls.run_code_for_training(bme_net, display_images=False)
    dls.run_code_for_testing(bme_net, display_images=False)
```

Image 2.1: Screenshot of code snippet for maxpool for downsampling

For this task, upon implementing the above code on a Colab GPU for the CIFAR-10 dataset, we obtain the following results for the loss curve, confusion matrices, and per-class accuracies:

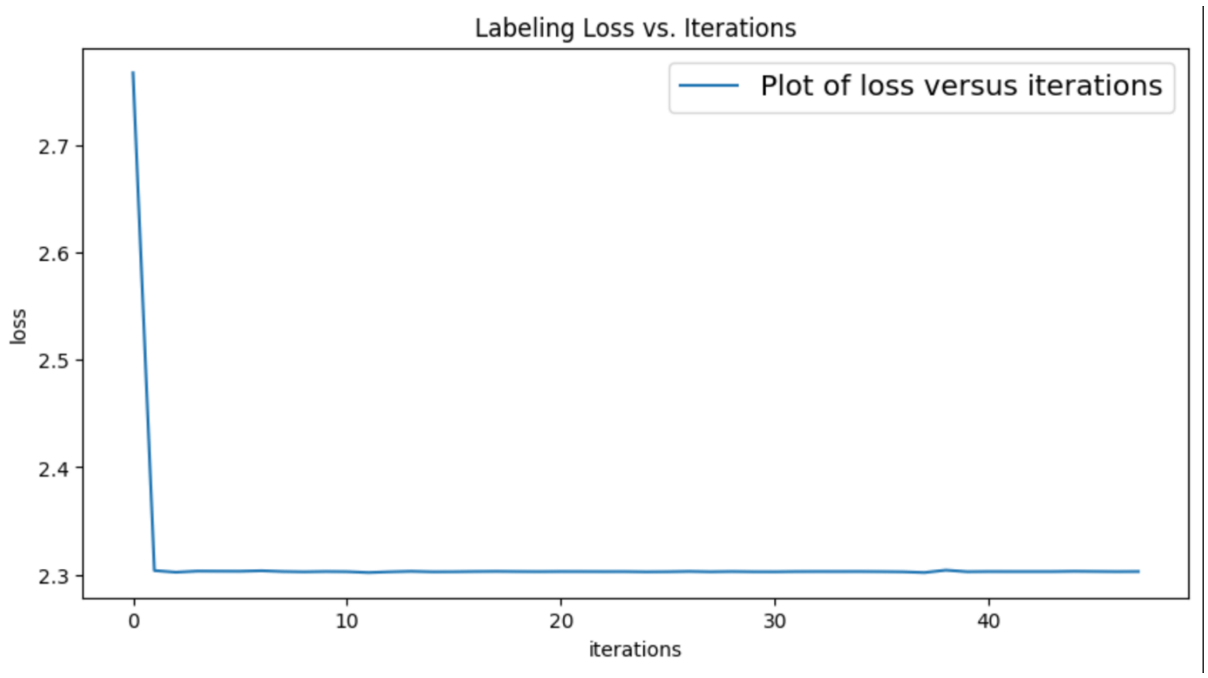
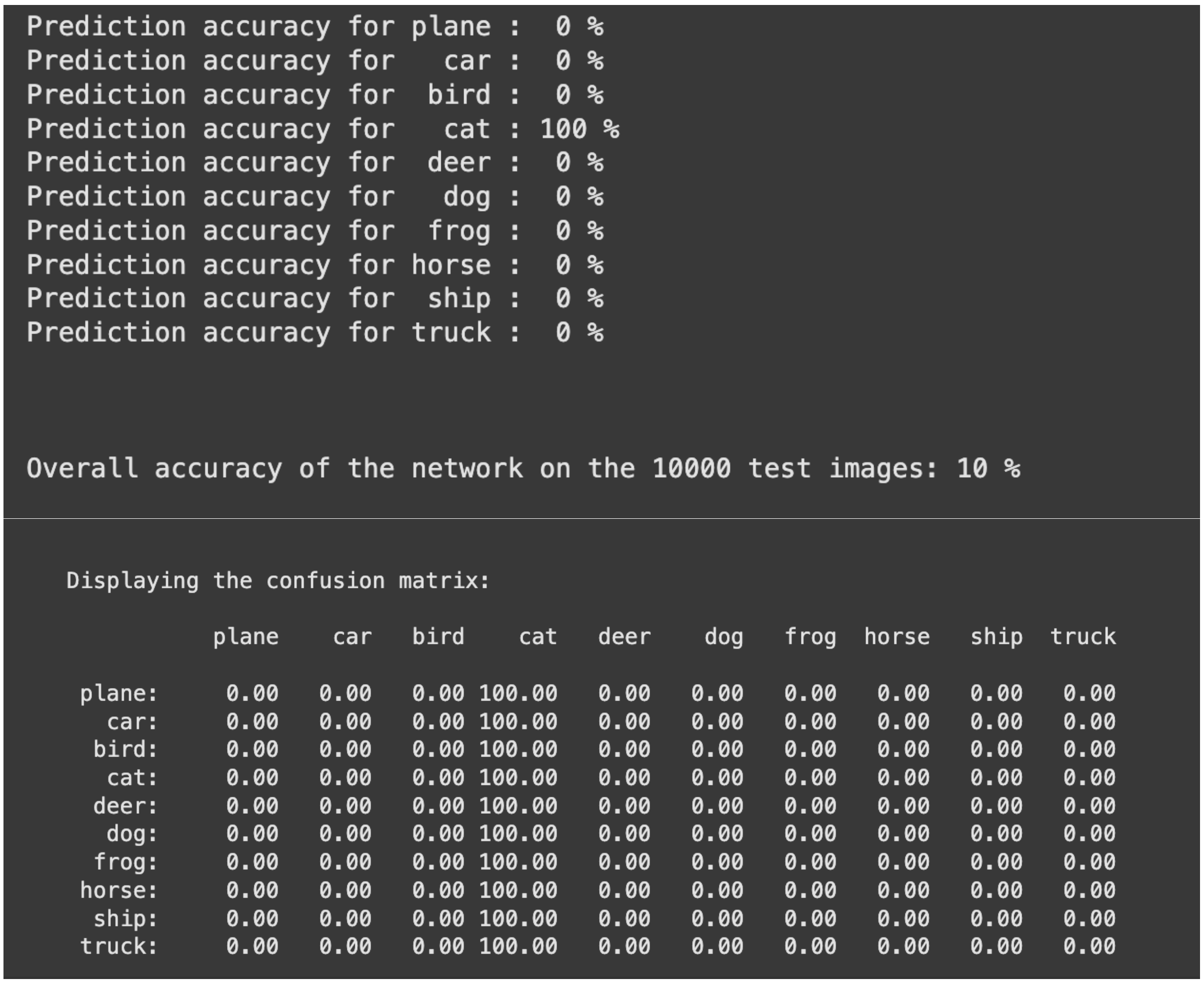
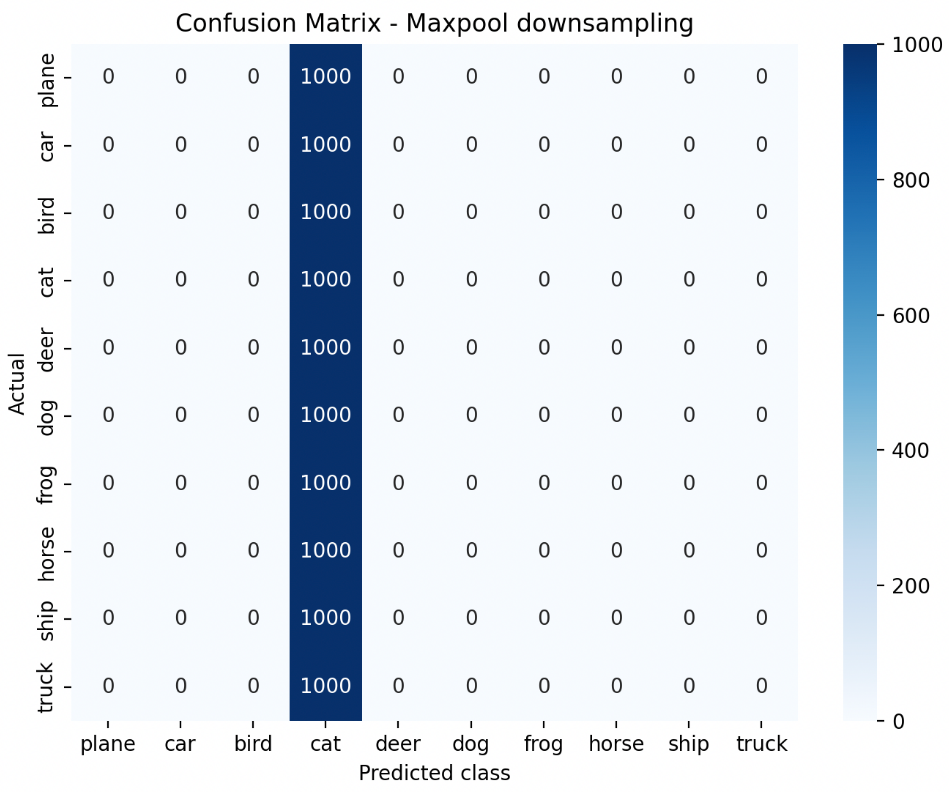


Image 2.2: Loss curve for maxpool downsampling, skip_connections=True



```
Prediction accuracy for plane :  0 %
Prediction accuracy for   car :  0 %
Prediction accuracy for  bird :  0 %
Prediction accuracy for   cat : 100 %
Prediction accuracy for  deer :  0 %
Prediction accuracy for   dog :  0 %
Prediction accuracy for  frog :  0 %
Prediction accuracy for horse :  0 %
Prediction accuracy for  ship :  0 %
Prediction accuracy for truck :  0 %



Overall accuracy of the network on the 10000 test images: 10 %
```

```
   Displaying the confusion matrix:

               plane    car   bird    cat   deer    dog   frog  horse   ship  truck

    plane:      0.00   0.00   0.00 100.00   0.00   0.00   0.00   0.00   0.00   0.00
      car:      0.00   0.00   0.00 100.00   0.00   0.00   0.00   0.00   0.00   0.00
     bird:      0.00   0.00   0.00 100.00   0.00   0.00   0.00   0.00   0.00   0.00
      cat:      0.00   0.00   0.00 100.00   0.00   0.00   0.00   0.00   0.00   0.00
     deer:      0.00   0.00   0.00 100.00   0.00   0.00   0.00   0.00   0.00   0.00
      dog:      0.00   0.00   0.00 100.00   0.00   0.00   0.00   0.00   0.00   0.00
     frog:      0.00   0.00   0.00 100.00   0.00   0.00   0.00   0.00   0.00   0.00
    horse:      0.00   0.00   0.00 100.00   0.00   0.00   0.00   0.00   0.00   0.00
     ship:      0.00   0.00   0.00 100.00   0.00   0.00   0.00   0.00   0.00   0.00
    truck:      0.00   0.00   0.00 100.00   0.00   0.00   0.00   0.00   0.00   0.00
```
Image 2.3: Terminal output for maxpool downsampling, skip_connections=True

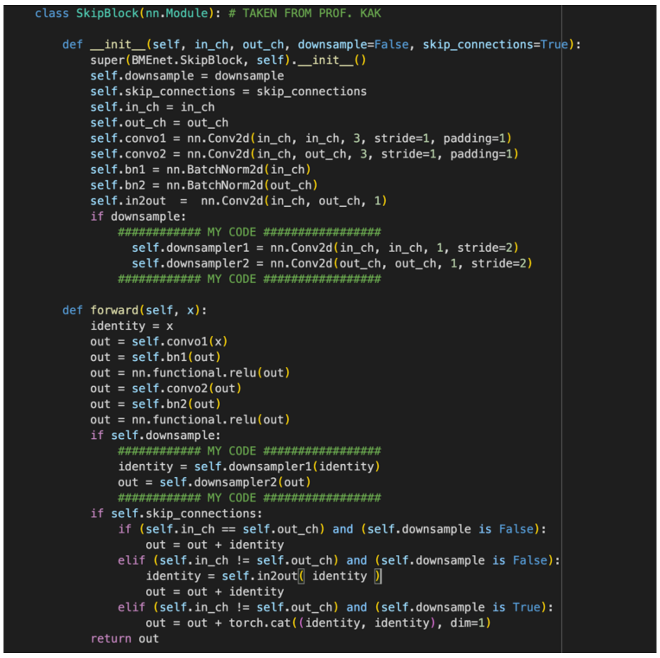


	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	0	0	0	1000	0	0	0	0	0	0
car	0	0	0	1000	0	0	0	0	0	0
bird	0	0	0	1000	0	0	0	0	0	0
cat	0	0	0	1000	0	0	0	0	0	0
deer	0	0	0	1000	0	0	0	0	0	0
dog	0	0	0	1000	0	0	0	0	0	0
frog	0	0	0	1000	0	0	0	0	0	0
horse	0	0	0	1000	0	0	0	0	0	0
ship	0	0	0	1000	0	0	0	0	0	0
truck	0	0	0	1000	0	0	0	0	0	0

Image 2.4: Confusion Matrix for maxpool downsampling, skip_connections=True

As we notice, the predictions made by this model are evidently terrible. The model guesses a single class for all of the images. I will discuss the detailed observations in the next section once I display the plots for each of the models tested.

The next model I test is the original method used for downsampling, which is using a convolutional layer with stride = 2. The code for this model looks like this:



```
class SkipBlock(nn.Module): # TAKEN FROM PROF. KAK

    def __init__(self, in_ch, out_ch, downsample=False, skip_connections=True):
        super(BMEnet.SkipBlock, self).__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:
            ############ MY CODE #################
              self.downsampler1 = nn.Conv2d(in_ch, in_ch, 1, stride=2)
              self.downsampler2 = nn.Conv2d(out_ch, out_ch, 1, stride=2)
            ############ MY CODE #################

    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:
            ############ MY CODE #################
            identity = self.downsampler1(identity)
            out = self.downsampler2(out)
            ############ MY CODE #################
        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
```

Image 2.5: Code block for downsampling using stride=2 layer

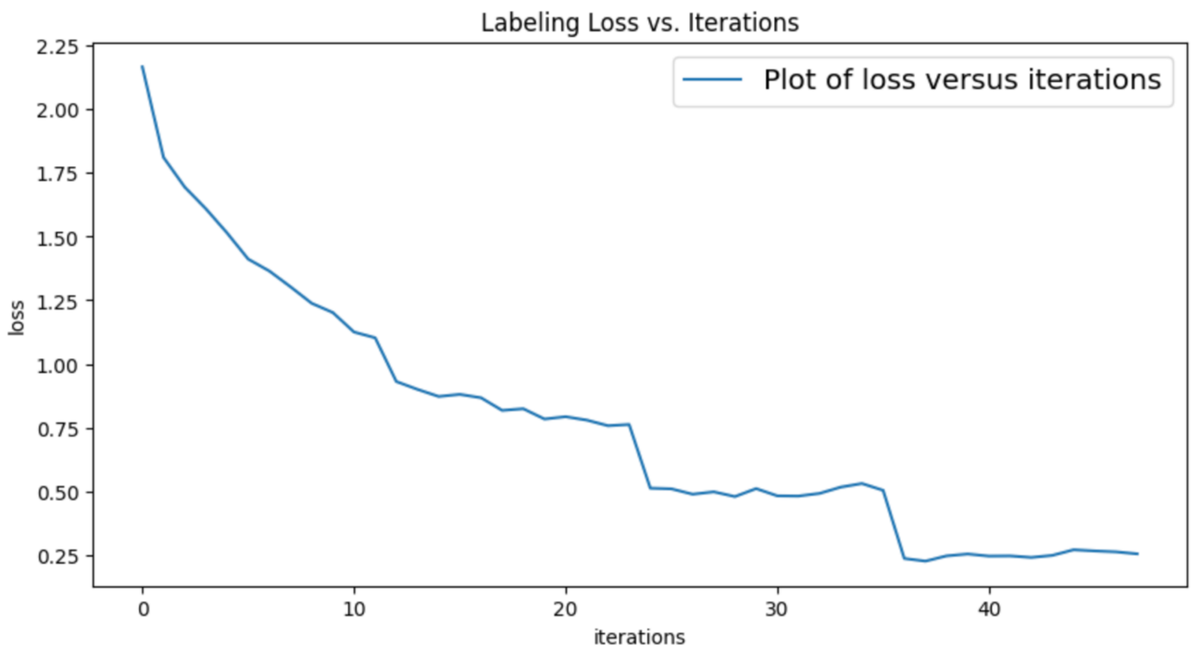
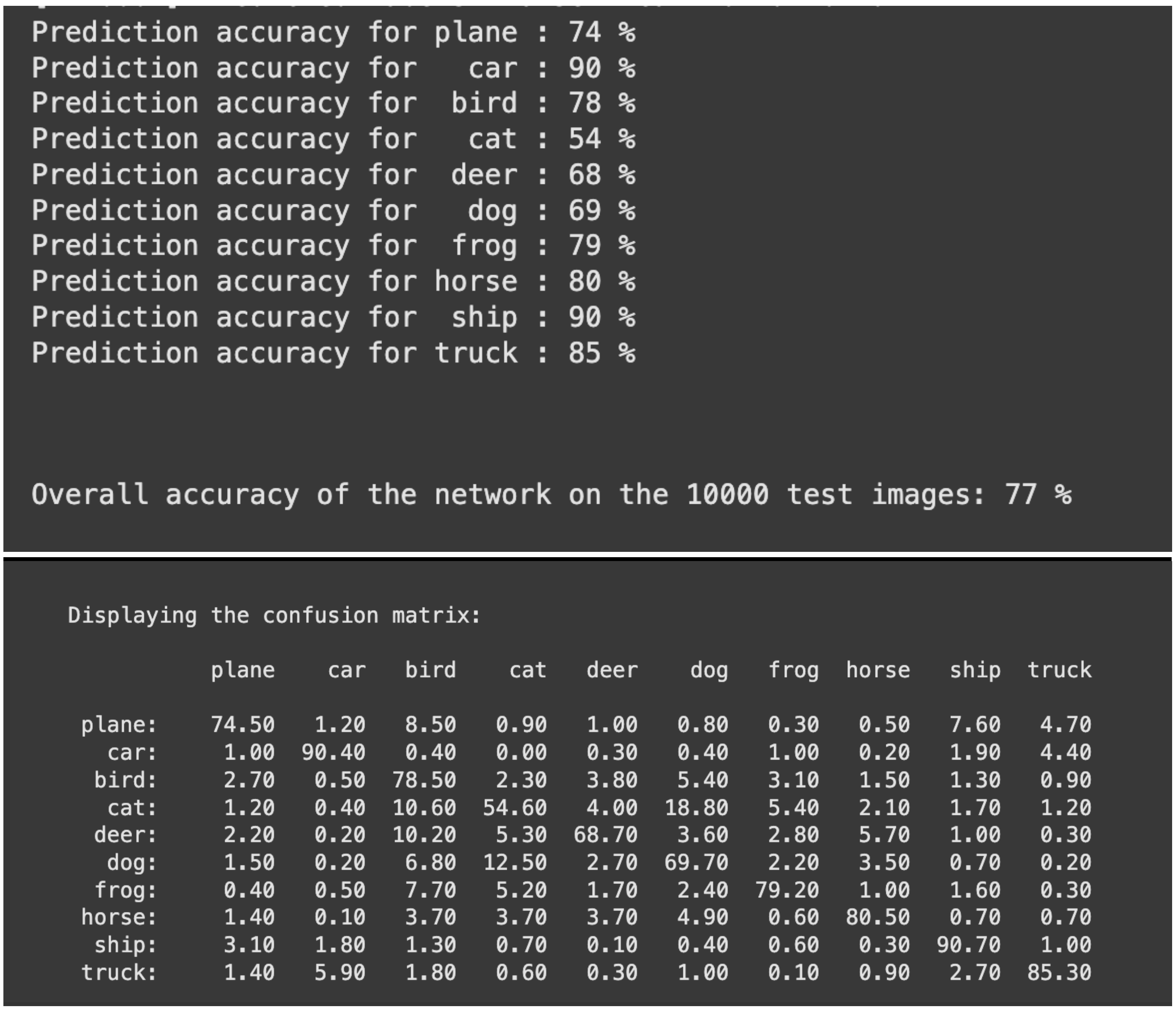


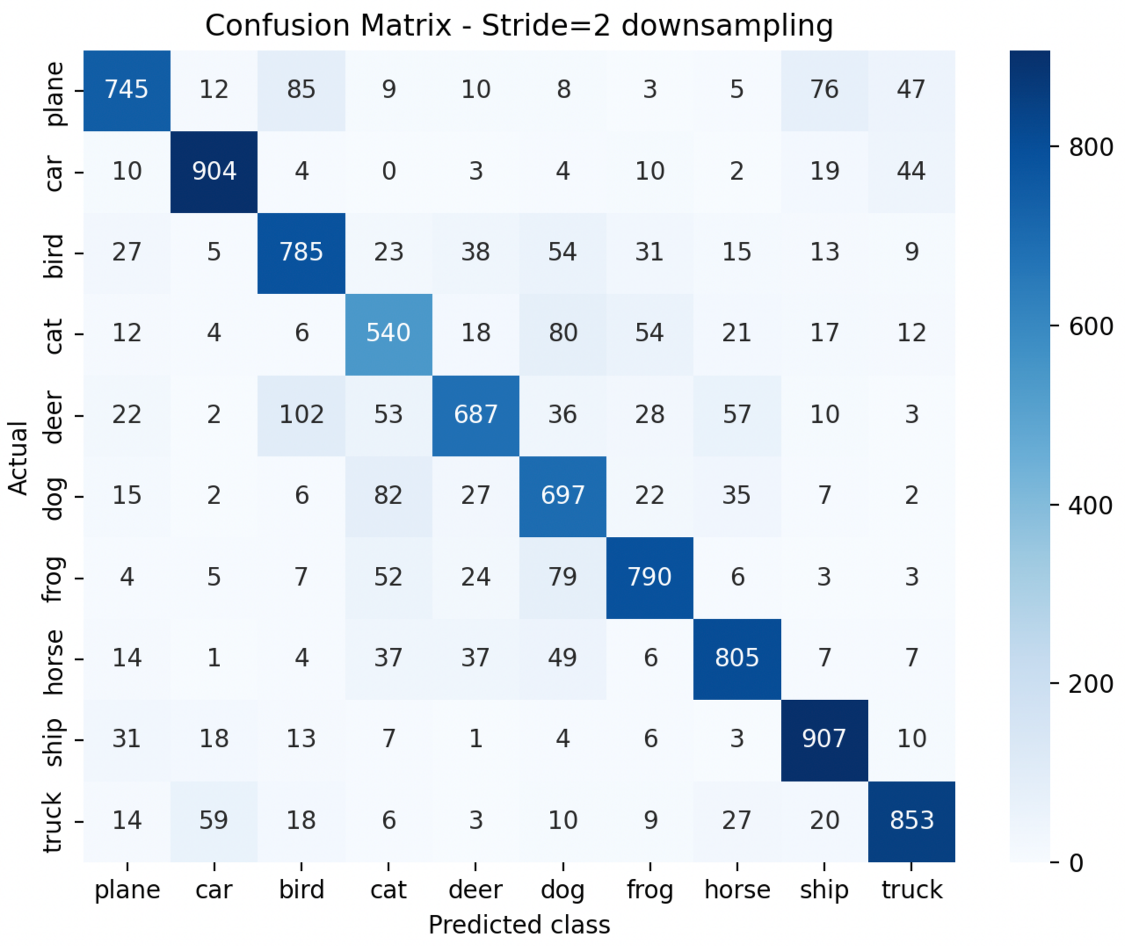
Image 2.6: Loss curve for stride=2 downsampling, skip_connections=True



```
Prediction accuracy for plane : 74 %
Prediction accuracy for   car : 90 %
Prediction accuracy for  bird : 78 %
Prediction accuracy for   cat : 54 %
Prediction accuracy for  deer : 68 %
Prediction accuracy for   dog : 69 %
Prediction accuracy for  frog : 79 %
Prediction accuracy for horse : 80 %
Prediction accuracy for  ship : 90 %
Prediction accuracy for truck : 85 %



Overall accuracy of the network on the 10000 test images: 77 %
```

```
  Displaying the confusion matrix:

             plane    car   bird    cat   deer    dog   frog  horse   ship  truck

    plane:   74.50   1.20   8.50   0.90   1.00   0.80   0.30   0.50   7.60   4.70
      car:    1.00  90.40   0.40   0.00   0.30   0.40   1.00   0.20   1.90   4.40
     bird:    2.70   0.50  78.50   2.30   3.80   5.40   3.10   1.50   1.30   0.90
      cat:    1.20   0.40  10.60  54.60   4.00  18.80   5.40   2.10   1.70   1.20
     deer:    2.20   0.20  10.20   5.30  68.70   3.60   2.80   5.70   1.00   0.30
      dog:    1.50   0.20   6.80  12.50   2.70  69.70   2.20   3.50   0.70   0.20
     frog:    0.40   0.50   7.70   5.20   1.70   2.40  79.20   1.00   1.60   0.30
    horse:    1.40   0.10   3.70   3.70   3.70   4.90   0.60  80.50   0.70   0.70
     ship:    3.10   1.80   1.30   0.70   0.10   0.40   0.60   0.30  90.70   1.00
    truck:    1.40   5.90   1.80   0.60   0.30   1.00   0.10   0.90   2.70  85.30
```

Image 2.7: Terminal output for stride=2 downsampling, skip_connections=True



Actual \ Predicted	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	745	12	85	9	10	8	3	5	76	47
car	10	904	4	0	3	4	10	2	19	44
bird	27	5	785	23	38	54	31	15	13	9
cat	12	4	6	540	18	80	54	21	17	12
deer	22	2	102	53	687	36	28	57	10	3
dog	15	2	6	82	27	697	22	35	7	2
frog	4	5	7	52	24	79	790	6	3	3
horse	14	1	4	37	37	49	6	805	7	7
ship	31	18	13	7	1	4	6	3	907	10
truck	14	59	18	6	3	10	9	27	20	853

Image 2.8: Confusion Matrix for stride=2 downsampling, skip_connections=True

As we notice, this model performs great, with a testing accuracy of 77%. In order to highlight the importance of skip connections, below is the same model run once again, with the only difference being in the line:

```
bme_net = dls.BMEnet(dls, skip_connections=True, depth=8)
```

which we now change to:

```
bme_net = dls.BMEnet(dls, skip_connections=False, depth=8)
```

Upon running this code, we obtain the following results:

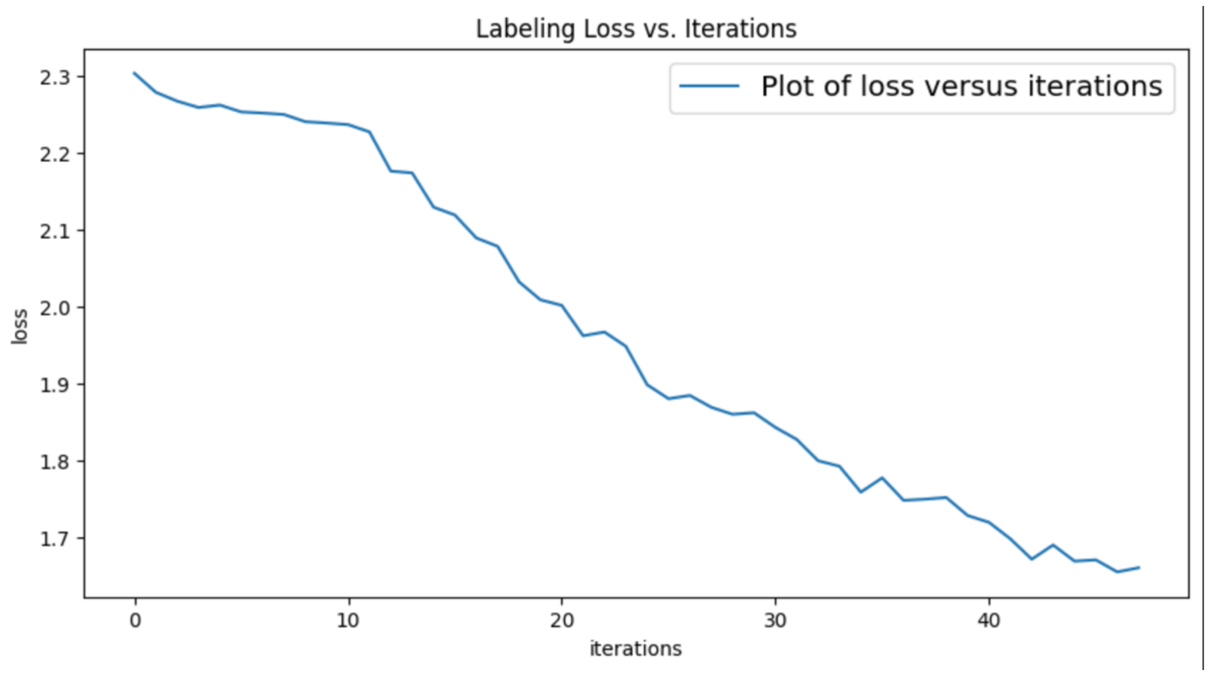
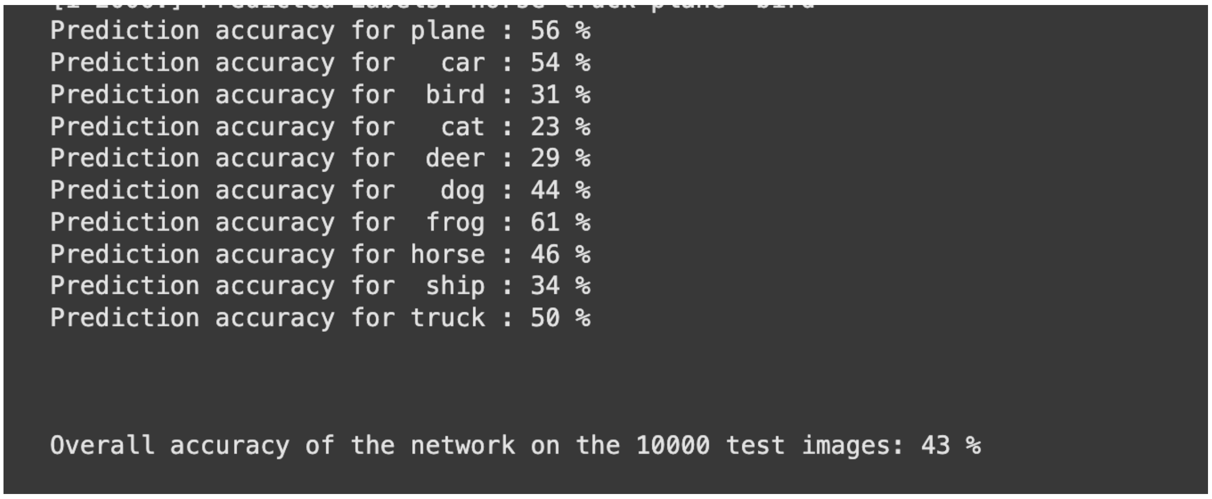
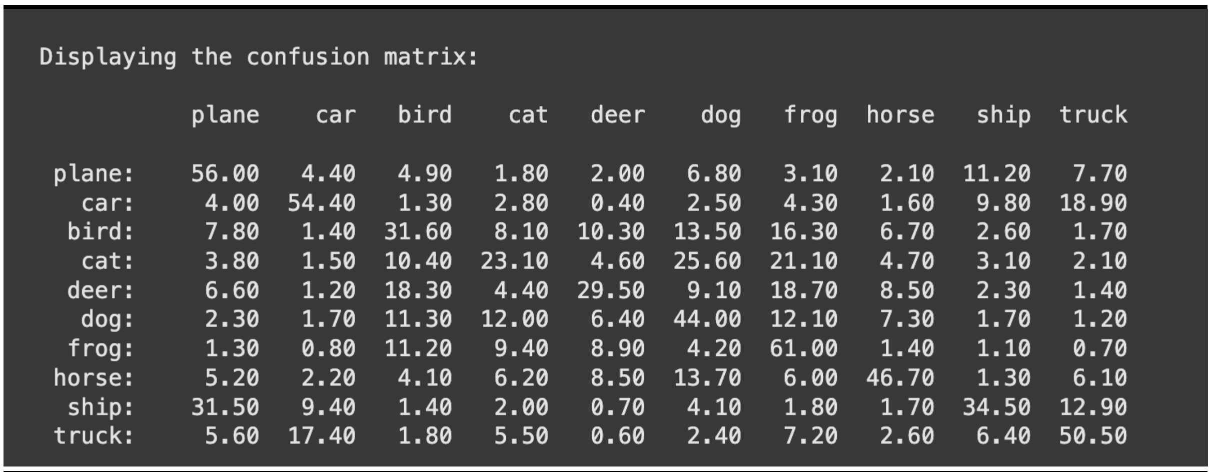


Image 2.9: Loss curve for stride=2 downsampling, skip_connections=False

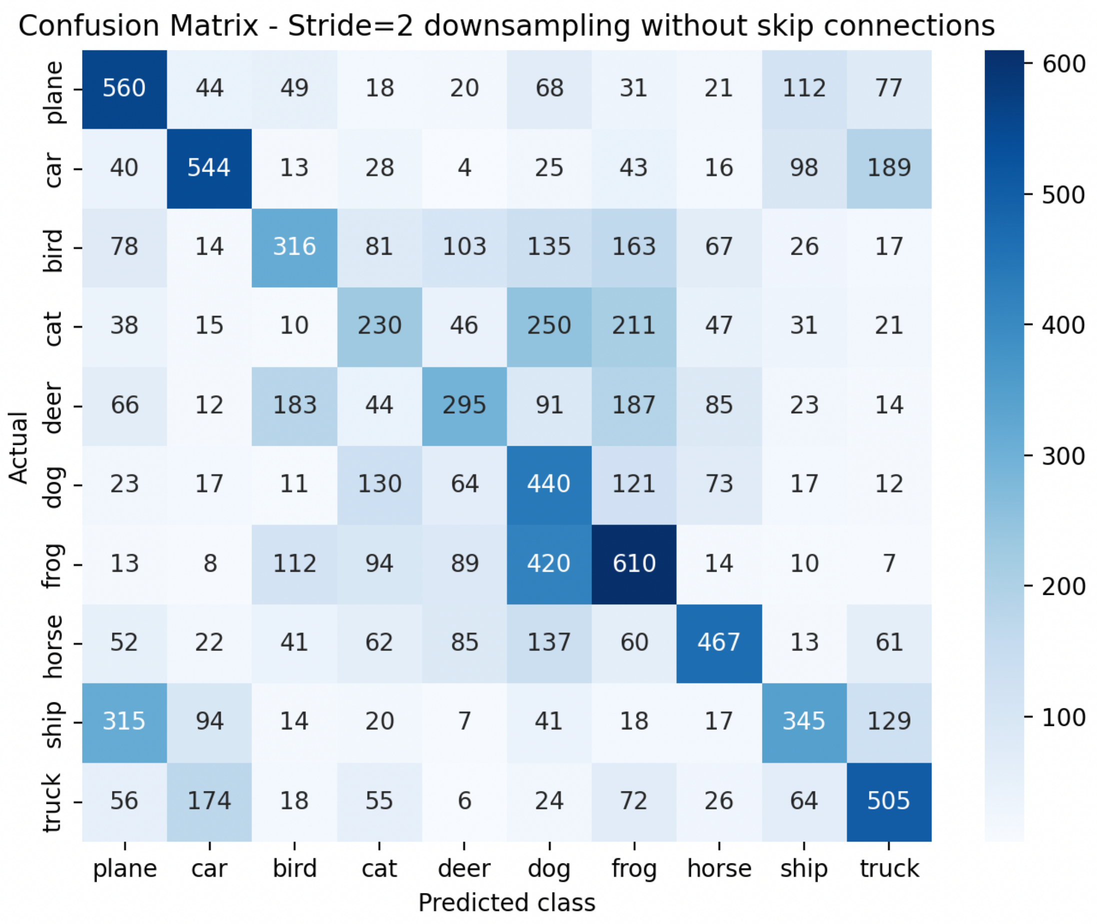


```
Prediction accuracy for plane : 56 %
Prediction accuracy for   car : 54 %
Prediction accuracy for  bird : 31 %
Prediction accuracy for   cat : 23 %
Prediction accuracy for  deer : 29 %
Prediction accuracy for   dog : 44 %
Prediction accuracy for  frog : 61 %
Prediction accuracy for horse : 46 %
Prediction accuracy for  ship : 34 %
Prediction accuracy for truck : 50 %



Overall accuracy of the network on the 10000 test images: 43 %
```



```
Displaying the confusion matrix:

          plane    car   bird    cat   deer    dog   frog  horse   ship  truck

 plane:   56.00   4.40   4.90   1.80   2.00   6.80   3.10   2.10  11.20   7.70
   car:    4.00  54.40   1.30   2.80   0.40   2.50   4.30   1.60   9.80  18.90
  bird:    7.80   1.40  31.60   8.10  10.30  13.50  16.30   6.70   2.60   1.70
   cat:    3.80   1.50  10.40  23.10   4.60  25.60  21.10   4.70   3.10   2.10
  deer:    6.60   1.20  18.30   4.40  29.50   9.10  18.70   8.50   2.30   1.40
   dog:    2.30   1.70  11.30  12.00   6.40  44.00  12.10   7.30   1.70   1.20
  frog:    1.30   0.80  11.20   9.40   8.90   4.20  61.00   1.40   1.10   0.70
 horse:    5.20   2.20   4.10   6.20   8.50  13.70   6.00  46.70   1.30   6.10
  ship:   31.50   9.40   1.40   2.00   0.70   4.10   1.80   1.70  34.50  12.90
 truck:    5.60  17.40   1.80   5.50   0.60   2.40   7.20   2.60   6.40  50.50
```

Image 2.10: Terminal output for stride=2 downsampling, skip_connections=False



Actual \ Predicted	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	560	44	49	18	20	68	31	21	112	77
car	40	544	13	28	4	25	43	16	98	189
bird	78	14	316	81	103	135	163	67	26	17
cat	38	15	10	230	46	250	211	47	31	21
deer	66	12	183	44	295	91	187	85	23	14
dog	23	17	11	130	64	440	121	73	17	12
frog	13	8	112	94	89	420	610	14	10	7
horse	52	22	41	62	85	137	60	467	13	61
ship	315	94	14	20	7	41	18	17	345	129
truck	56	174	18	55	6	24	72	26	64	505

Image 2.11: Confusion Matrix for stride=2 downsampling, skip_connections=False

The accuracy of this model being a lot lower than the one with skip_connections highlights the importance of the skip block. Skip connections help in solving the vanishing gradient problem. When gradients diminish during the backpropagation step, the first few layers learn extremely slowly. The presence of skip connections allows gradients to flow more freely. However, as this model (model 3) had the parameter skip_connections set to ‘False’, the gradients must have diminished faster, causing the poor accuracy.

Below are two summary tables that help us compare the three models in terms of their overall accuracies and per-class accuracies:

Model	**Accuracy**
BMEnet() with maxpool downsampling, skip_connections=True (MODEL 1)	10%
BMEnet() with stride=2 downsampling, skip_connections=True (MODEL 2)	77%
BMEnet() with stride=2 downsampling, skip_connections=False (MODEL 3)	43%

Table 1: Table showing accuracies of all 3 models

Class	**Model 1**	**Model 2**	**Model 3**
Plane	0%	74%	56%
Car	0%	90%	54%
Bird	0%	78%	31%
Cat	100%	54%	23%
Deer	0%	68%	29%
Dog	0%	69%	44%
Frog	0%	79%	61%
Horse	0%	80%	46%
Ship	0%	90%	34%
Truck	0%	85%	50%

Table 2: Table showing per-class accuracies of all 3 models

In this table, Model 1 refers to the model with Maxpool downsampling with skip connections. Model 2 refers to the model with Stride=2 downsampling with skip connections. Model 3 refers to the model with Stride = 2 downsampling without skip connections.

Observation on Task 2

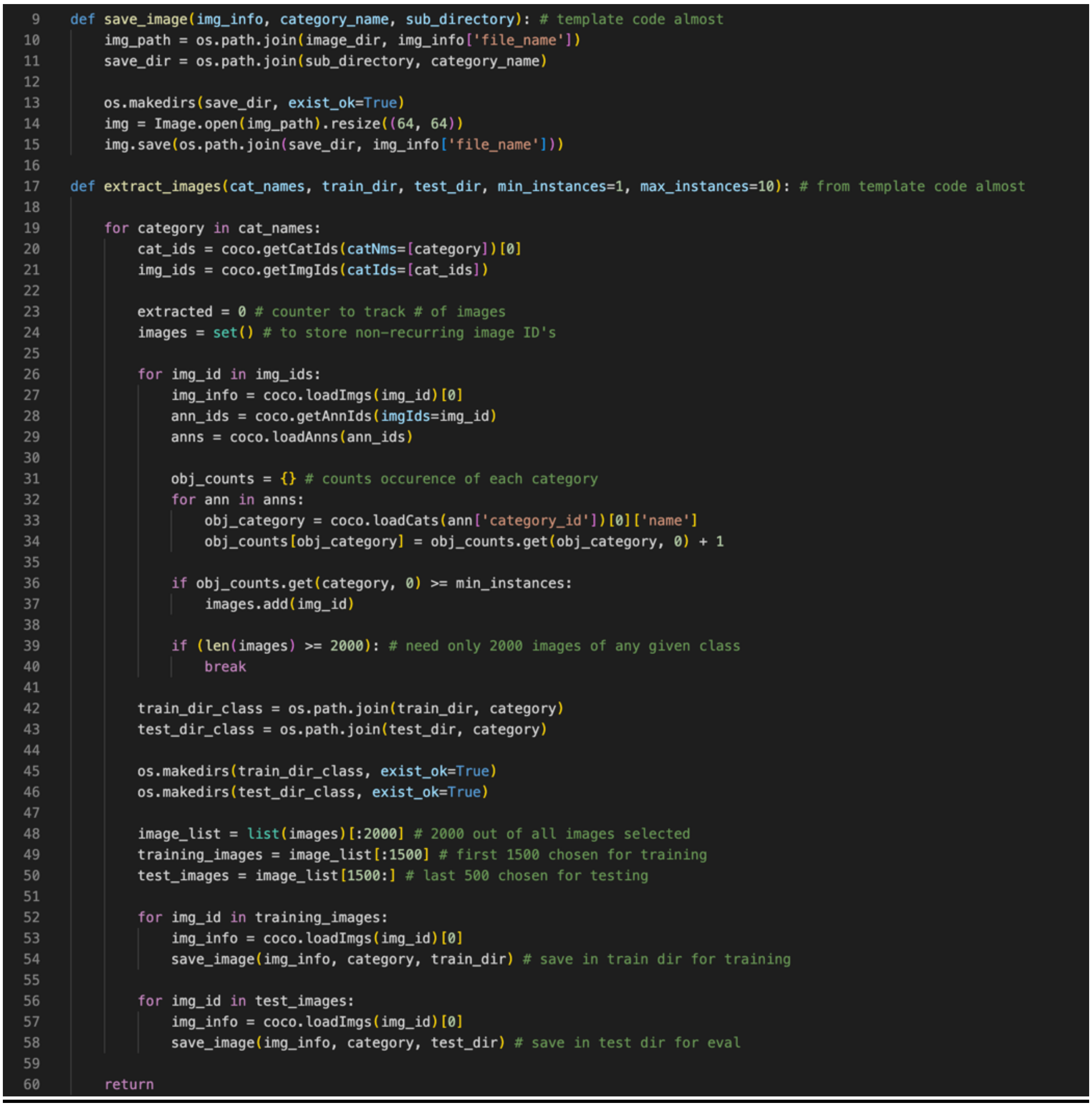
We observe from Tables 1 and 2, and all the other intermediate results in this section, that the model that works best for the given dataset is the Model 2, which performs skip connections and downsampling using Stride = 2. The reason for superior performance of this model is that when we use stride = 2, the feature maps from the identity path and convolutional path have matching dimensions and channels. It downsamples while transforming whereas the axpool layer used in Model 2 blindly chooses the highest pixel values and removes important parts of the image while downsampling. Hence, this reduces the spatial resolution and ends up guessing the same class for all of the images.

We can conclude that the Stride =2 convolution allows network to optimally downsample whereas using Maxpool, which is not a learnable layer, results in disruption of skip connections, causing poor performance and terrible accuracy. We also note that as mentioned earlier, the third model (skip_connections = False) is able to learn better than the Maxpool layer as it still learns the good representations because of the convolutions and activations as in a normal convolutional network. However, due to the absence of skip connections, and thereby the presence of the vanishing gradient problem, the model 3 fails to perform as well as Model 1.

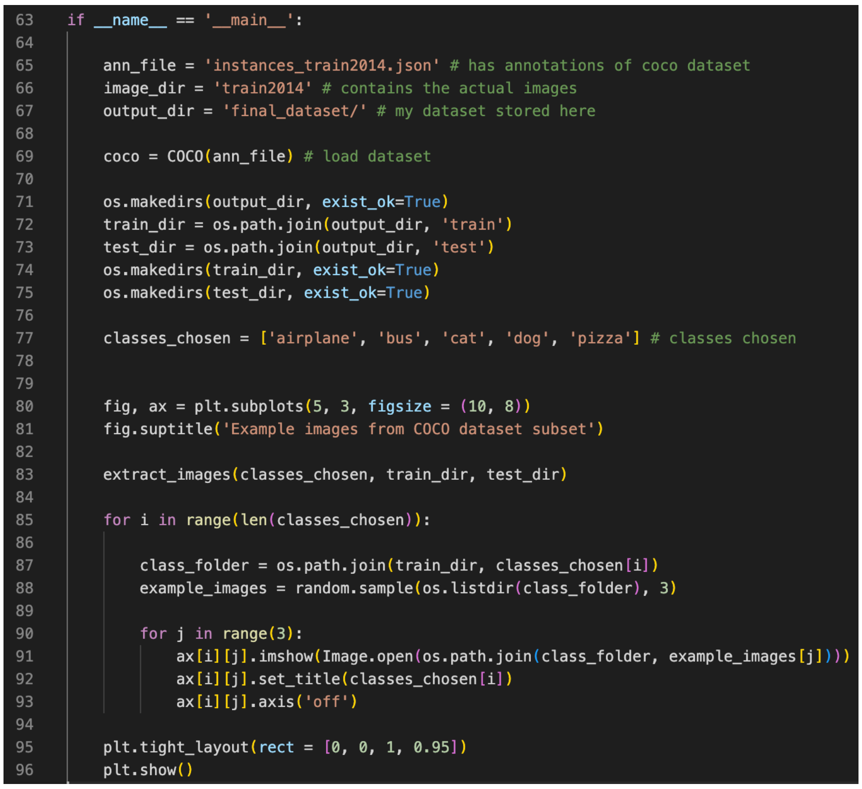
Task 3: Skip Connections with MSCOCO dataset

The first step of this task requires us to create the subset of the dataset with 2000 images per class split into 1500 training and 500 testing images. We extract 2000 such images for each of the five classes :
[‘airpane’, ‘bus’, ‘cat’, ‘dog’, ‘pizza’].

We do so using the code given below:



```python
def save_image(img_info, category_name, sub_directory): # template code almost
    img_path = os.path.join(image_dir, img_info['file_name'])
    save_dir = os.path.join(sub_directory, category_name)

    os.makedirs(save_dir, exist_ok=True)
    img = Image.open(img_path).resize((64, 64))
    img.save(os.path.join(save_dir, img_info['file_name']))

def extract_images(cat_names, train_dir, test_dir, min_instances=1, max_instances=10): # from template code almost

    for category in cat_names:
        cat_ids = coco.getCatIds(catNms=[category])[0]
        img_ids = coco.getImgIds(catIds=[cat_ids])

        extracted = 0 # counter to track # of images
        images = set() # to store non-recurring image ID's

        for img_id in img_ids:
            img_info = coco.loadImgs(img_id)[0]
            ann_ids = coco.getAnnIds(imgIds=img_id)
            anns = coco.loadAnns(ann_ids)

            obj_counts = {} # counts occurence of each category
            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 obj_counts.get(category, 0) >= min_instances:
                images.add(img_id)

            if (len(images) >= 2000): # need only 2000 images of any given class
                break

        train_dir_class = os.path.join(train_dir, category)
        test_dir_class = os.path.join(test_dir, category)

        os.makedirs(train_dir_class, exist_ok=True)
        os.makedirs(test_dir_class, exist_ok=True)

        image_list = list(images)[:2000] # 2000 out of all images selected
        training_images = image_list[:1500] # first 1500 chosen for training
        test_images = image_list[1500:] # last 500 chosen for testing

        for img_id in training_images:
            img_info = coco.loadImgs(img_id)[0]
            save_image(img_info, category, train_dir) # save in train dir for training

        for img_id in test_images:
            img_info = coco.loadImgs(img_id)[0]
            save_image(img_info, category, test_dir) # save in test dir for eval

    return
```



```python
if __name__ == '__main__':

    ann_file = 'instances_train2014.json' # has annotations of coco dataset
    image_dir = 'train2014' # contains the actual images
    output_dir = 'final_dataset/' # my dataset stored here

    coco = COCO(ann_file) # load dataset

    os.makedirs(output_dir, exist_ok=True)
    train_dir = os.path.join(output_dir, 'train')
    test_dir = os.path.join(output_dir, 'test')
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(test_dir, exist_ok=True)

    classes_chosen = ['airplane', 'bus', 'cat', 'dog', 'pizza'] # classes chosen


    fig, ax = plt.subplots(5, 3, figsize = (10, 8))
    fig.suptitle('Example images from COCO dataset subset')

    extract_images(classes_chosen, train_dir, test_dir)

    for i in range(len(classes_chosen)):

        class_folder = os.path.join(train_dir, classes_chosen[i])
        example_images = random.sample(os.listdir(class_folder), 3)

        for j in range(3):
            ax[i][j].imshow(Image.open(os.path.join(class_folder, example_images[j])))
            ax[i][j].set_title(classes_chosen[i])
            ax[i][j].axis('off')

    plt.tight_layout(rect = [0, 0, 1, 0.95])
    plt.show()
```

Image 3.1: Code block for creation of dataset

This code also gives us the 5 x 3 table of example images from the subset, showing 3 images for each of the five classes as shown below:

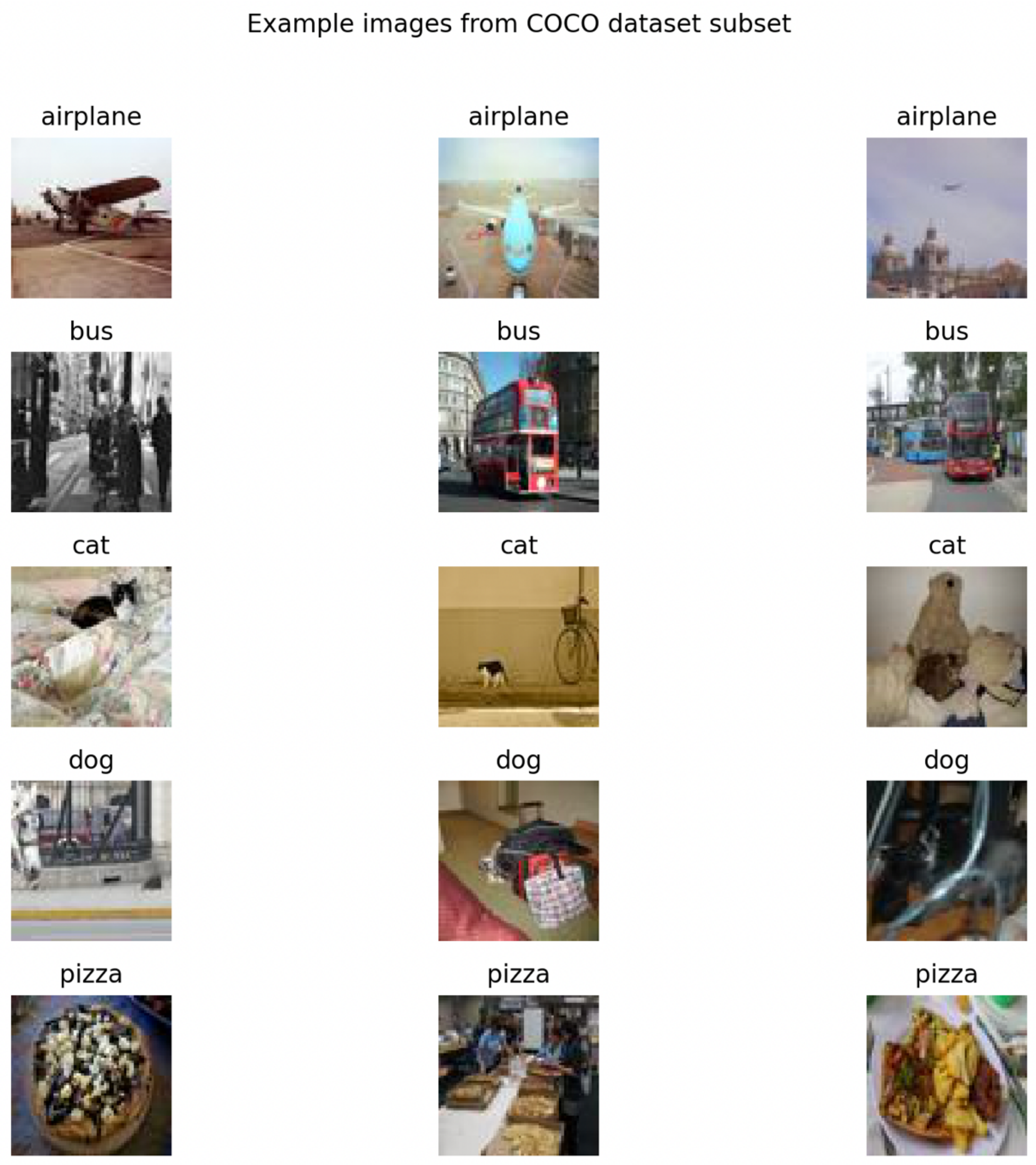
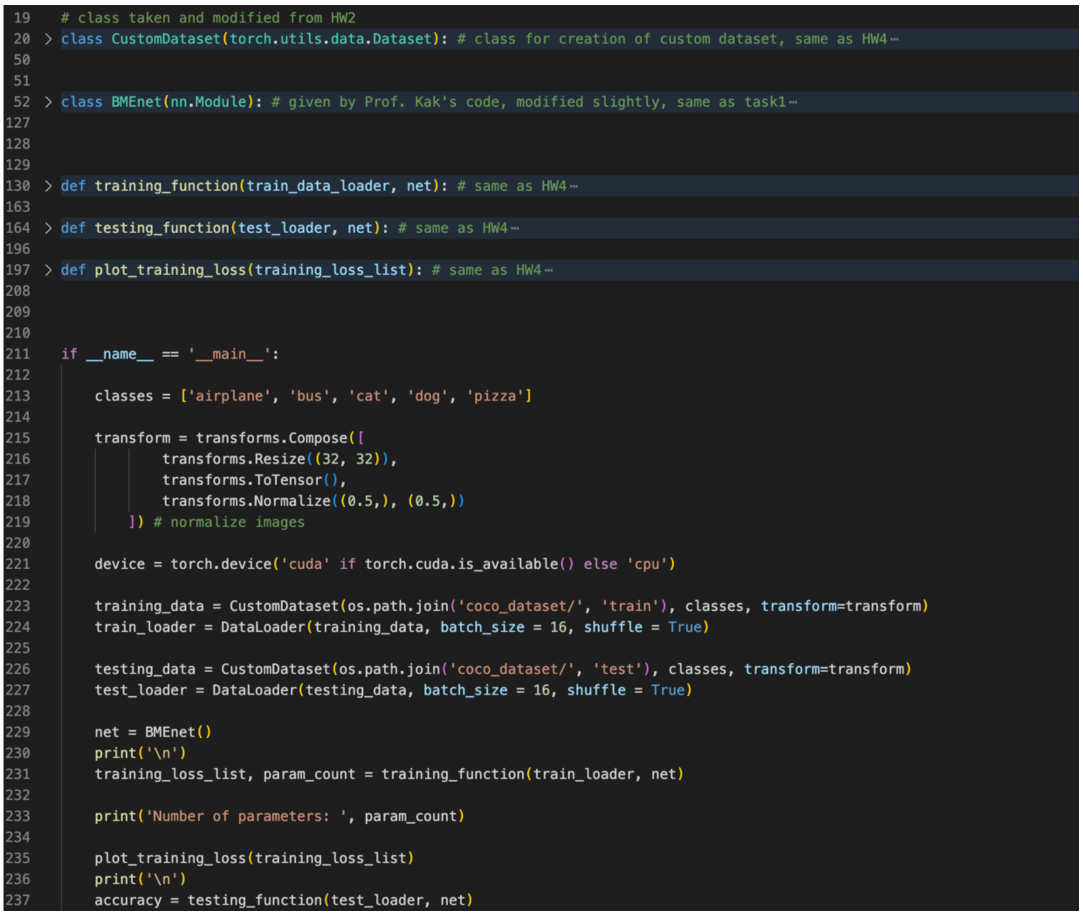


Image 3.2: 5 x 3 table showing example images from dataset

Once we have the dataset is created, we modify the code we had for BMENet from earlier with the SkipBlock and add a training and evaluation function as in HW4. Below is the code for this section:



```
# class taken and modified from HW2
> class CustomDataset(torch.utils.data.Dataset): # class for creation of custom dataset, same as HW4…


> class BMEnet(nn.Module): # given by Prof. Kak's code, modified slightly, same as task1…



> def training_function(train_data_loader, net): # same as HW4…

> def testing_function(test_loader, net): # same as HW4…

> def plot_training_loss(training_loss_list): # same as HW4…



if __name__ == '__main__':

    classes = ['airplane', 'bus', 'cat', 'dog', 'pizza']

    transform = transforms.Compose([
            transforms.Resize((32, 32)),
            transforms.ToTensor(),
            transforms.Normalize((0.5,), (0.5,))
        ]) # normalize images

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    training_data = CustomDataset(os.path.join('coco_dataset/', 'train'), classes, transform=transform)
    train_loader = DataLoader(training_data, batch_size = 16, shuffle = True)

    testing_data = CustomDataset(os.path.join('coco_dataset/', 'test'), classes, transform=transform)
    test_loader = DataLoader(testing_data, batch_size = 16, shuffle = True)

    net = BMEnet()
    print('\n')
    training_loss_list, param_count = training_function(train_loader, net)

    print('Number of parameters: ', param_count)

    plot_training_loss(training_loss_list)
    print('\n')
    accuracy = testing_function(test_loader, net)
```

Image 3.3: Code block for training and evaluation of skipBlock on COCO

Upon implementing this code, we obtain the following plots:

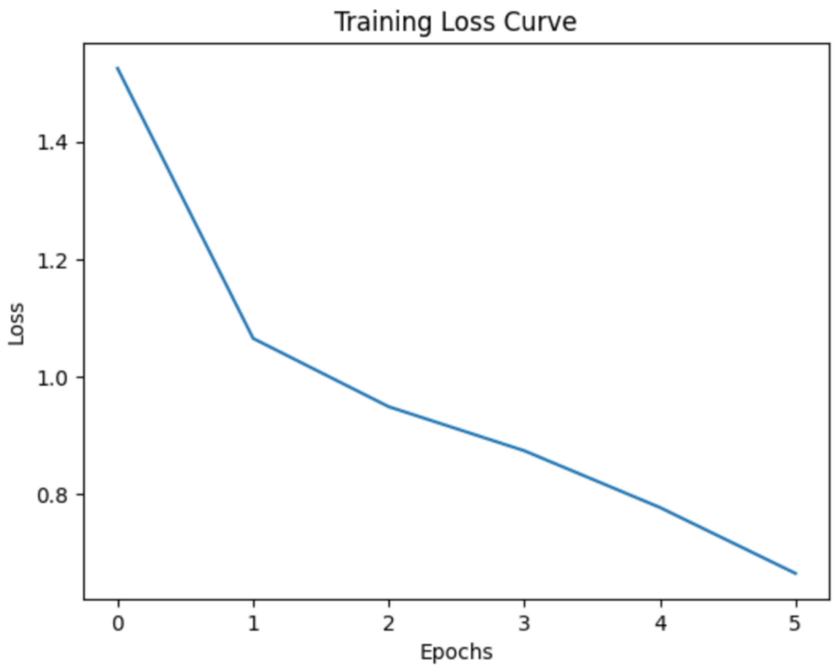
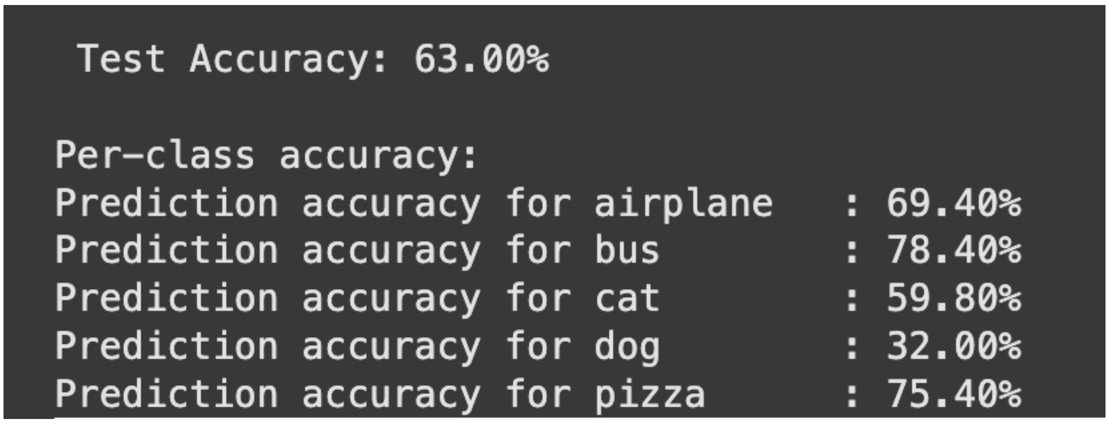
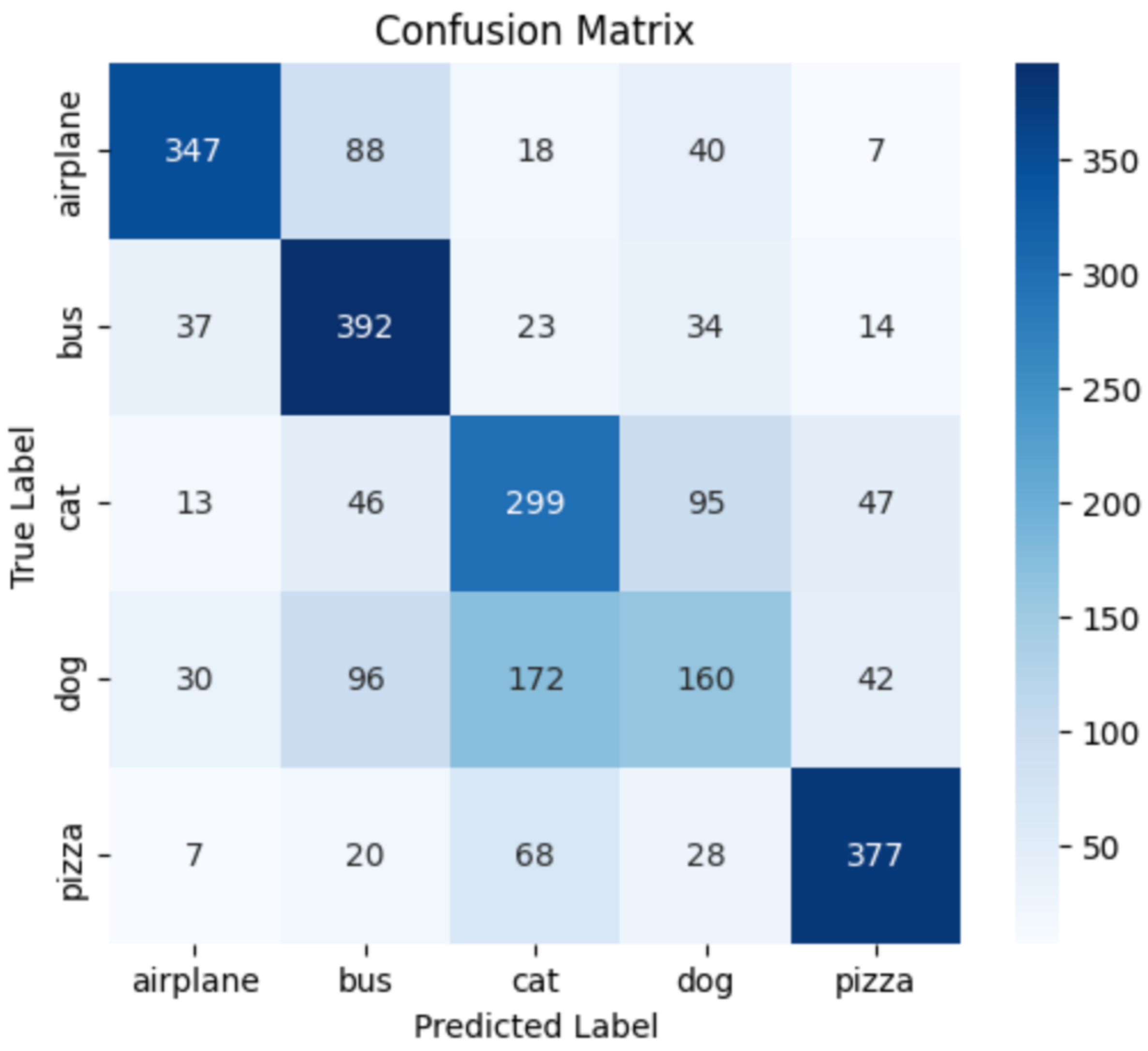


Image 3.3: Loss curve of training on MS-COCO subset



```
 Test Accuracy: 63.00%

Per-class accuracy:
Prediction accuracy for airplane   : 69.40%
Prediction accuracy for bus        : 78.40%
Prediction accuracy for cat        : 59.80%
Prediction accuracy for dog        : 32.00%
Prediction accuracy for pizza      : 75.40%
```

Image 3.4: Terminal output of evaluation on MS-COCO subset



	airplane	bus	cat	dog	pizza
airplane	347	88	18	40	7
bus	37	392	23	34	14
cat	13	46	299	95	47
dog	30	96	172	160	42
pizza	7	20	68	28	377

Image 3.5: Confusion Matrix of testing on MS-COCO subset

Overall Accuracy: 63%

Class	**Accuracy**
Airplane	69.40%
Bus	78.40%
Cat	59.80%
Dog	32.00%
Pizza	75.40%

Table 3: 5x1 table showing per-class accuracies

<u>Observations</u>

As we observe from the table of accuracies, overall accuracy, loss curve, and confusion matrix, the BMENet model performs fairly well on the COCO dataset. Although I end up running this model only for 6 epochs due to computational bottlenecks, it still has an overall testing accuracy of 63% on 2500 unseen images.

This relatively high accuracy could be attributed to the presence of a deep neural network with skip connections. As mentioned multiple times so far in this document, the presence of skip connections allow gradients to flow through the layers a lot more easily. The vanishing gradient problem is solved, which results in deeper layers suffering from unstable learning and low accuracy. Skip connections also help in retaining spatial and textural information from earlier layers. This is also why it has a classification accuracy of over ~70% for three out of the five classes.

Upon looking deeply at the confusion matrix, we also notice that a common misclassification occurs between dog and cat. This is easy to understand as both dogs and cats share several common features (e.g.: eyes, nose, tail, etc.) This means that the model would have perhaps performed better had we given it classes that were not as similar to one another. The model's loss curve also shows that the loss was continually decreasing and had not stagnated yet. This means that if we had more computational power and had run the model for more number of epochs, this would have also resulted in a better accuracy. I believe even data augmentation to increase size and robustness of dataset would have helped in solving the {dog, cat} misclassification problem.

Finally, we conclude that stride=2 downsampling and skip connections aided the BMENet model in preserving spatial information and avoiding the vanishing gradient problem, resulting in a relatively high classification accuracy which could have been further enhanced given a more robust dataset and more computational power.

SOURCE CODE

(1) Task1.py:

```
import random
import numpy
import torch
import os, sys
from DLStudio import *
import torch.nn as nn
import torch.nn.functional as F
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import copy
import torch.optim as optim
import time
import logging



class BMEnet(nn.Module): # given by Prof. Kak's code, modified slightly

    def __init__(self, num_classes = 5, skip_connections = True, depth = 8):
        super(BMEnet, self).__init__()
        self.depth = depth
        self.conv = nn.Conv2d(3, 64, 3, padding=1)
        self.skip64_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip64_arr.append(BMEnet.SkipBlock(64, 64,
skip_connections=skip_connections))
        self.skip64to128ds = BMEnet.SkipBlock(64, 128, downsample=True,
skip_connections=skip_connections )

        self.skip128_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip128_arr.append(BMEnet.SkipBlock(128, 128,
skip_connections=skip_connections))
        self.skip128to256ds = BMEnet.SkipBlock(128, 256, downsample=True,
skip_connections=skip_connections )
        self.skip256_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip256_arr.append(BMEnet.SkipBlock(256, 256,
skip_connections=skip_connections))

        num_ds = 2 # we downsample two times ()
        self.fc1 =  nn.Linear( (32 // (2 ** num_ds))  *  (32 //(2 ** num_ds))  *
256, 1000)
        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

    class SkipBlock(nn.Module): # given by Prof. Kak's code

        def __init__(self, in_ch, out_ch, downsample=False, skip_connections=True):
            super(BMEnet.SkipBlock, self).__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:
                #################### MY CODE ##########################
                self.downsampler1 = nn.Conv2d(in_ch, in_ch, 1, stride=2)
                self.downsampler2 = nn.Conv2d(out_ch, out_ch, 1, stride=2) # CHANGE
TO MAXPOOL FOR MODEL 1
                #################### MY CODE ##########################

        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:
                    identity = self.in2out(identity)

                out = out + identity

            return out




device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

dls = DLStudio(
                  dataroot = "./data/CIFAR-10/",
                  image_size = [32,32],
                  path_saved_model = "./saved_model",
                  momentum = 0.9,
                  learning_rate = 1e-4,
                  epochs = 4,
                  batch_size = 16,
                  classes =
('plane','car','bird','cat','deer','dog','frog','horse','ship','truck'),
                  use_gpu = True,
              )


if __name__ == '__main__':
    bme_net = BMEnet(dls, skip_connections=True, depth=8).to(device) # for model 3,
make skip_connections=False
    dls.load_cifar_10_dataset()

    number_of_learnable_params = sum(p.numel() for p in bme_net.parameters() if
p.requires_grad)
    print("\n\nThe number of learnable parameters in the model: %d" %
number_of_learnable_params)

    dls.run_code_for_training(bme_net, display_images=False)

    dls.run_code_for_testing(bme_net, display_images=False)
```

(2) Create_dataset.py (for task2)’

```
from pycocotools.coco import COCO
import os
from PIL import Image
import torchvision
import random
import matplotlib.pyplot as plt
```

```python
def save_image(img_info, category_name, sub_directory): # template code almost
    img_path = os.path.join(image_dir, img_info['file_name'])
    save_dir = os.path.join(sub_directory, category_name)

    os.makedirs(save_dir, exist_ok=True)
    img = Image.open(img_path).resize((64, 64))
    img.save(os.path.join(save_dir, img_info['file_name']))

def extract_images(cat_names, train_dir, test_dir, min_instances=1,
max_instances=10): # from template code almost

    for category in cat_names:
        cat_ids = coco.getCatIds(catNms=[category])[0]
        img_ids = coco.getImgIds(catIds=[cat_ids])

        extracted = 0 # counter to track # of images
        images = set() # to store non-recurring image ID's

        for img_id in img_ids:
            img_info = coco.loadImgs(img_id)[0]
            ann_ids = coco.getAnnIds(imgIds=img_id)
            anns = coco.loadAnns(ann_ids)

            obj_counts = {} # counts occurence of each category
            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 obj_counts.get(category, 0) >= min_instances:
                images.add(img_id)

            if (len(images) >= 2000): # need only 2000 images of any given class
                break

        train_dir_class = os.path.join(train_dir, category)
        test_dir_class = os.path.join(test_dir, category)

        os.makedirs(train_dir_class, exist_ok=True)
        os.makedirs(test_dir_class, exist_ok=True)

        image_list = list(images)[:2000] # 2000 out of all images selected
        training_images = image_list[:1500] # first 1500 chosen for training
        test_images = image_list[1500:] # last 500 chosen for testing

        for img_id in training_images:
            img_info = coco.loadImgs(img_id)[0]
            save_image(img_info, category, train_dir) # save in train dir for
training
```

```
        for img_id in test_images:
            img_info = coco.loadImgs(img_id)[0]
            save_image(img_info, category, test_dir) # save in test dir for eval

    return


if __name__ == '__main__':

    ann_file = 'instances_train2014.json' # has annotations of coco dataset
    image_dir = 'train2014' # contains the actual images
    output_dir = 'final_dataset/' # my dataset stored here

    coco = COCO(ann_file) # load dataset

    os.makedirs(output_dir, exist_ok=True)
    train_dir = os.path.join(output_dir, 'train')
    test_dir = os.path.join(output_dir, 'test')
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(test_dir, exist_ok=True)

    classes_chosen = ['airplane', 'bus', 'cat', 'dog', 'pizza'] # classes chosen


    fig, ax = plt.subplots(5, 3, figsize = (10, 8))
    fig.suptitle('Example images from COCO dataset subset')

    extract_images(classes_chosen, train_dir, test_dir)

    for i in range(len(classes_chosen)): # plots 5 x 3 table

        class_folder = os.path.join(train_dir, classes_chosen[i])
        example_images = random.sample(os.listdir(class_folder), 3)

        for j in range(3):
            ax[i][j].imshow(Image.open(os.path.join(class_folder,
example_images[j])))
            ax[i][j].set_title(classes_chosen[i])
            ax[i][j].axis('off')

    plt.tight_layout(rect = [0, 0, 1, 0.95])
    plt.show()
```

(3) Task2.py (for training and eval on COCO)

```
from pycocotools.coco import COCO
import os
import numpy
```

```python
from PIL import Image
import random
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, Subset
import matplotlib.pyplot as plt
import time
from sklearn.metrics import confusion_matrix, accuracy_score
import torch.nn as nn
import torch.nn.functional as F
import seaborn as sns
import pandas as pd
from DLStudio import *


# class taken and modified from HW2
class CustomDataset(torch.utils.data.Dataset): # class for creation of custom
dataset, same as HW4
    def __init__(self, root, class_names, transform=None):
        self.root = root
        self.class_names = class_names
        self.transform = transform
        self.class_to_index = {} # dict stores mapping from class to index
        self.image_paths = []
        self.labels = []

        index = 0
        for class_ in class_names:
            self.class_to_index[class_] = index

            class_dir = os.path.join(root, class_)
            if (os.path.exists(class_dir)):
                for image in os.listdir(class_dir):
                    self.image_paths.append(os.path.join(class_dir, image))
                    self.labels.append(index) # stores corresponding label
            index += 1

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

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


class BMEnet(nn.Module): # given by Prof. Kak's code, modified slightly, same as
task1
```

```
    def __init__(self, num_classes = 5, skip_connections = True, depth = 8):
        super(BMEnet, self).__init__()
        self.depth = depth
        self.conv = nn.Conv2d(3, 64, 3, padding=1)
        self.skip64_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip64_arr.append(BMEnet.SkipBlock(64, 64,
skip_connections=skip_connections))
        self.skip64to128ds = BMEnet.SkipBlock(64, 128, downsample=True,
skip_connections=skip_connections )

        self.skip128_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip128_arr.append(BMEnet.SkipBlock(128, 128,
skip_connections=skip_connections))
        self.skip128to256ds = BMEnet.SkipBlock(128, 256, downsample=True,
skip_connections=skip_connections )
        self.skip256_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip256_arr.append(BMEnet.SkipBlock(256, 256,
skip_connections=skip_connections))

        num_ds = 2 # we downsample two times ()
        self.fc1 =  nn.Linear( (32 // (2 ** num_ds))  *  (32 //(2 ** num_ds))  *
256, 1000)
        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

    class SkipBlock(nn.Module): # given by Prof. Kak's code

        def __init__(self, in_ch, out_ch, downsample=False, skip_connections=True):
            super(BMEnet.SkipBlock, self).__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:
                #################### MY CODE ##########################
                self.downsampler1 = nn.Conv2d(in_ch, in_ch, 1, stride=2)
                self.downsampler2 = nn.Conv2d(out_ch, out_ch, 1, stride=2) # CHANGE
TO MAXPOOL FOR MODEL 1
                #################### MY CODE ##########################

        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:
                    identity = self.in2out(identity)

                out = out + identity

            return out



def training_function(train_data_loader, net): # same as HW4
    net = net.to(device)
    param_count = 0
    for param in net.parameters():
        if (param.requires_grad):
            param_count += param.numel() # for num-param table

    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(net.parameters(), lr = 1e-4)

    epochs = 1
    training_loss_list = []
    for epoch in range(epochs):
        running_loss = 0.0
        for i, data in enumerate(train_data_loader): # training as per template
            inputs, labels = data
            inputs = inputs.to(device)
            labels = labels.to(device)
            optimizer.zero_grad()
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        epoch_loss = running_loss / len(train_data_loader)
        training_loss_list.append(epoch_loss)
        print(f'epoch: {epoch + 1}, loss: {epoch_loss:.4f}')
    print('\n')

    print('Training complete')

    return training_loss_list, param_count

def testing_function(test_loader, net): # same as HW4
    net.eval()
    predictions_list = []
    labels_list = []
    misclass_list = []
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = net(inputs)
            prediction = outputs.argmax(dim = 1)
            fin_pred = prediction.cpu().tolist()
            fin_label = labels.cpu().tolist()
            predictions_list += fin_pred
            labels_list += fin_label

    accuracy = accuracy_score(labels_list, predictions_list) * 100

    print('\n')
    print(f'Validation accuracy: {accuracy:.2f}%')
    print('\n')

    conf_mat = confusion_matrix(labels_list, predictions_list)

    plt.figure(figsize = (8, 6))
    sns.heatmap(conf_mat, annot = True, fmt = 'd', cmap = 'Blues', xticklabels =
classes, yticklabels = classes)
    plt.xlabel('Predicted class')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()

    return accuracy

def plot_training_loss(training_loss_list): # same as HW4
```

```python
    plt.figure(figsize = (8, 6))
    x = range(0, 6)
    y = training_loss_list
    plt.plot(x, y)
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Training loss curve')
    plt.show()
    return



if __name__ == '__main__':

    classes = ['airplane', 'bus', 'cat', 'dog', 'pizza']

    transform = transforms.Compose([
            transforms.Resize((32, 32)),
            transforms.ToTensor(),
            transforms.Normalize((0.5,), (0.5,))
        ]) # normalize images

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    training_data = CustomDataset(os.path.join('coco_dataset/', 'train'), classes,
transform=transform)
    train_loader = DataLoader(training_data, batch_size = 16, shuffle = True)

    testing_data = CustomDataset(os.path.join('coco_dataset/', 'test'), classes,
transform=transform)
    test_loader = DataLoader(testing_data, batch_size = 16, shuffle = True)

    net = BMEnet()
    print('\n')
    training_loss_list, param_count = training_function(train_loader, net)

    print('Number of parameters: ', param_count)

    plot_training_loss(training_loss_list)
    print('\n')
    accuracy = testing_function(test_loader, net)
```

END