BME646 and ECE60146: Homework 6

Manas V Shetty

Question 1: observations on 1 pixel kernel

Using a 1×1 kernel for convolution, as seen in Line (I) in slide 16, is a common practice in deep learning, especially for changing the number of channels without affecting the spatial dimensions of the image. This is useful for adjusting feature maps to match the expected input size for subsequent layers. However, in Lines (K) and (L) in slide 16, the 1×1 convolution is used for downsampling by setting the stride to 2, effectively reducing the spatial resolution by half. Unlike standard convolutions that extract spatial features, these 1×1 convolutions with stride 2 act as simple sampling layers, selecting alternate pixels and reducing resolution while adjusting channels.

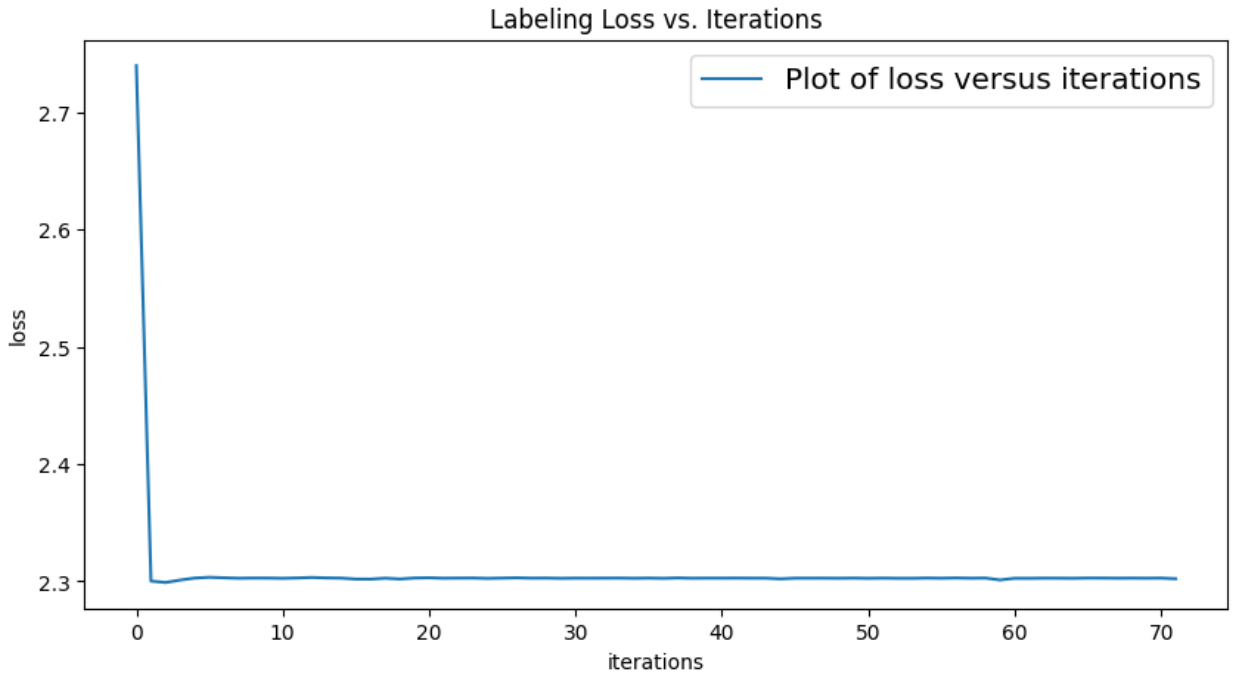
Instead of using 1×1 convolutions with stride 2 for downsampling, an alternative approach is to use MaxPooling (torch.nn.MaxPool2d(2,2)), which also reduces spatial size by half but selects the most prominent feature in each 2×2 patch, potentially improving feature retention. Another option is using a standard convolution with a kernel size of 3×3 and a stride of 2, which can provide additional feature extraction while reducing resolution. A comparison between these approaches could involve evaluating accuracy and loss on a test dataset

Question 2: BMENet with maxpool

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd


def main():
    dls = DLStudio(
        dataroot="./data/CIFAR-10/",
        image_size=[32, 32],
        path_saved_model="./saved_model_model.pt",
        momentum=0.9,
        learning_rate=1e-4,
        epochs=6,
        batch_size=4,
        classes=('plane','car','bird','cat','deer','dog','frog','horse','ship','truck'),
        use_gpu=True
    )

    print("Experiment 1: Original BMEnet with skip connections (convolution-based downsampling)")
    model1 = dls.BMEnet(dls, skip_connections=True, depth=8)
    model1.load_cifar_10_dataset()
    params1 = sum(p.numel() for p in model1.parameters() if p.requires_grad)
```

```
    print("Learnable parameters (Original BMEnet):", params1)

    model1.run_code_for_training(model1, display_images=False)
    model1.run_code_for_testing(model1, display_images=False)

    print("Experiment 2: BMEnet variant with maxpool downsampling")
    model2 = BMEnetMaxPool(dls, skip_connections=True, depth=8)
    model2.load_cifar_10_dataset()
    params2 = sum(p.numel() for p in model2.parameters() if p.requires_grad)
    print("Learnable parameters (BMEnet with MaxPool downsampling):", params2)

    model2.run_code_for_training(model2, display_images=False)
    model2.run_code_for_testing(model2, display_images=False)
if __name__ == '__main__':
    main()
```

2.1 Train loss curve

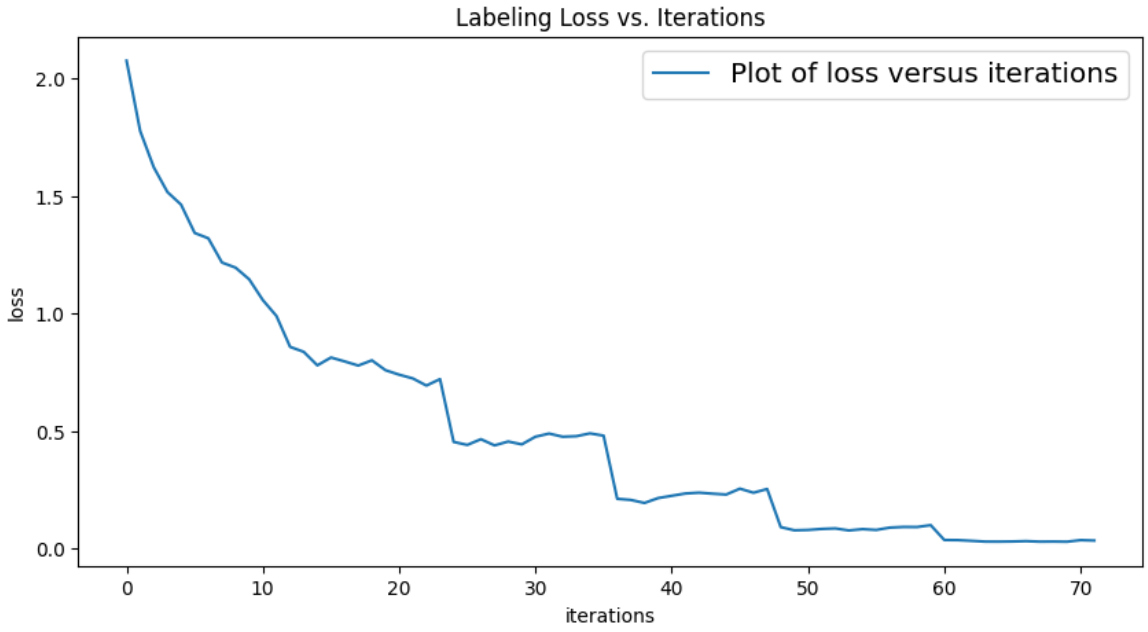


2.2 Confusion Matrix

```
Displaying the confusion matrix:

          plane    car   bird    cat   deer    dog   frog  horse   ship  truck
 plane:    0.00   0.00   0.00   0.00   0.00   0.00   0.00 100.00   0.00   0.00
   car:    0.00   0.00   0.00   0.00   0.00   0.00   0.00 100.00   0.00   0.00
  bird:    0.00   0.00   0.00   0.00   0.00   0.00   0.00 100.00   0.00   0.00
   cat:    0.00   0.00   0.00   0.10   0.00   0.00   0.10  99.80   0.00   0.00
  deer:    0.00   0.00   0.00   0.00   0.00   0.00   0.00 100.00   0.00   0.00
   dog:    0.00   0.00   0.00   0.10   0.00   0.00   0.00  99.90   0.00   0.00
  frog:    0.00   0.00   0.00   0.00   0.00   0.00   0.00 100.00   0.00   0.00
 horse:    0.00   0.00   0.00   0.00   0.00   0.00   0.00 100.00   0.00   0.00
  ship:    0.00   0.00   0.00   0.00   0.00   0.00   0.00 100.00   0.00   0.00
 truck:    0.00   0.00   0.00   0.00   0.00   0.00   0.00 100.00   0.00   0.00
```

Question 3: BMENet with stride

3.1 Train loss curve



3.2 Confusion Matrix

```
Displaying the confusion matrix:

          plane    car   bird    cat   deer    dog   frog  horse   ship  truck

 plane:   84.10   0.60   3.40   0.50   1.20   0.60   0.50   1.20   5.30   2.60
   car:    1.10  87.30   0.70   0.40   0.40   0.20   0.70   0.20   2.90   6.10
  bird:    5.30   0.10  75.40   2.00   6.10   4.70   3.20   1.90   0.60   0.70
   cat:    2.80   0.50   8.90  53.60   4.70  17.40   6.00   3.30   1.70   1.10
  deer:    1.70   0.10   5.90   4.20  75.60   2.60   3.10   5.80   0.90   0.10
   dog:    1.40   0.30   5.30  10.10   3.50  72.40   1.00   4.70   1.00   0.30
  frog:    0.70   0.30   5.00   3.20   1.40   1.70  84.90   1.00   1.30   0.50
 horse:    0.80   0.30   3.70   1.10   3.80   3.40   0.30  85.80   0.40   0.40
  ship:    3.70   0.60   1.10   0.40   0.10   0.40   0.30   0.60  91.90   0.90
 truck:    3.00   3.70   0.60   0.60   0.60   0.50   0.30   0.70   2.80  87.20
```

Question 4 Table1: Overall accuracy of 2 models

a. BMENet with maxpool

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

b. BMENet with stride

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

Question 5 Table2: Per class accuracy of 2 models

a. BMENet with maxpool

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

b. BMENet with stride

```
Prediction accuracy for plane : 84 %
Prediction accuracy for   car : 87 %
Prediction accuracy for  bird : 75 %
Prediction accuracy for   cat : 53 %
Prediction accuracy for  deer : 75 %
Prediction accuracy for   dog : 72 %
Prediction accuracy for  frog : 84 %
Prediction accuracy for horse : 85 %
Prediction accuracy for  ship : 91 %
Prediction accuracy for truck : 87 %
```

Question 6: Observations of Maxpool vs Stride

The comparison between BMEnet with MaxPool and BMEnet with Stride-based downsampling reveals significant differences in performance. The loss behavior in the MaxPool-based model shows a high initial loss that quickly drops but then plateaus around 2.3, indicating that the model struggles to learn meaningful features. In contrast, the Stride-based BMEnet exhibits a smooth and steady decline in loss, reaching values close to zero, which suggests that it is effectively learning from the dataset.

In terms of overall accuracy, the MaxPool-based BMEnet achieves only 10% accuracy, which is equivalent to random guessing on the 10-class CIFAR-10 dataset. On the other hand, the Stride-based BMEnet reaches 79% accuracy, demonstrating that it successfully learns to differentiate between different classes. This vast difference in performance indicates that the choice of downsampling method plays a crucial role in network effectiveness.

A closer look at the confusion matrices further supports this conclusion. In the MaxPool-based BMEnet, the model misclassifies almost everything as "horse" (100% accuracy for horse, 0% for all other classes), indicating a severe failure in generalization. In contrast, the Stride-based BMEnet produces a more balanced distribution of predictions, with per-class accuracy ranging between 53% and 91%, signifying much better feature extraction and class separation.

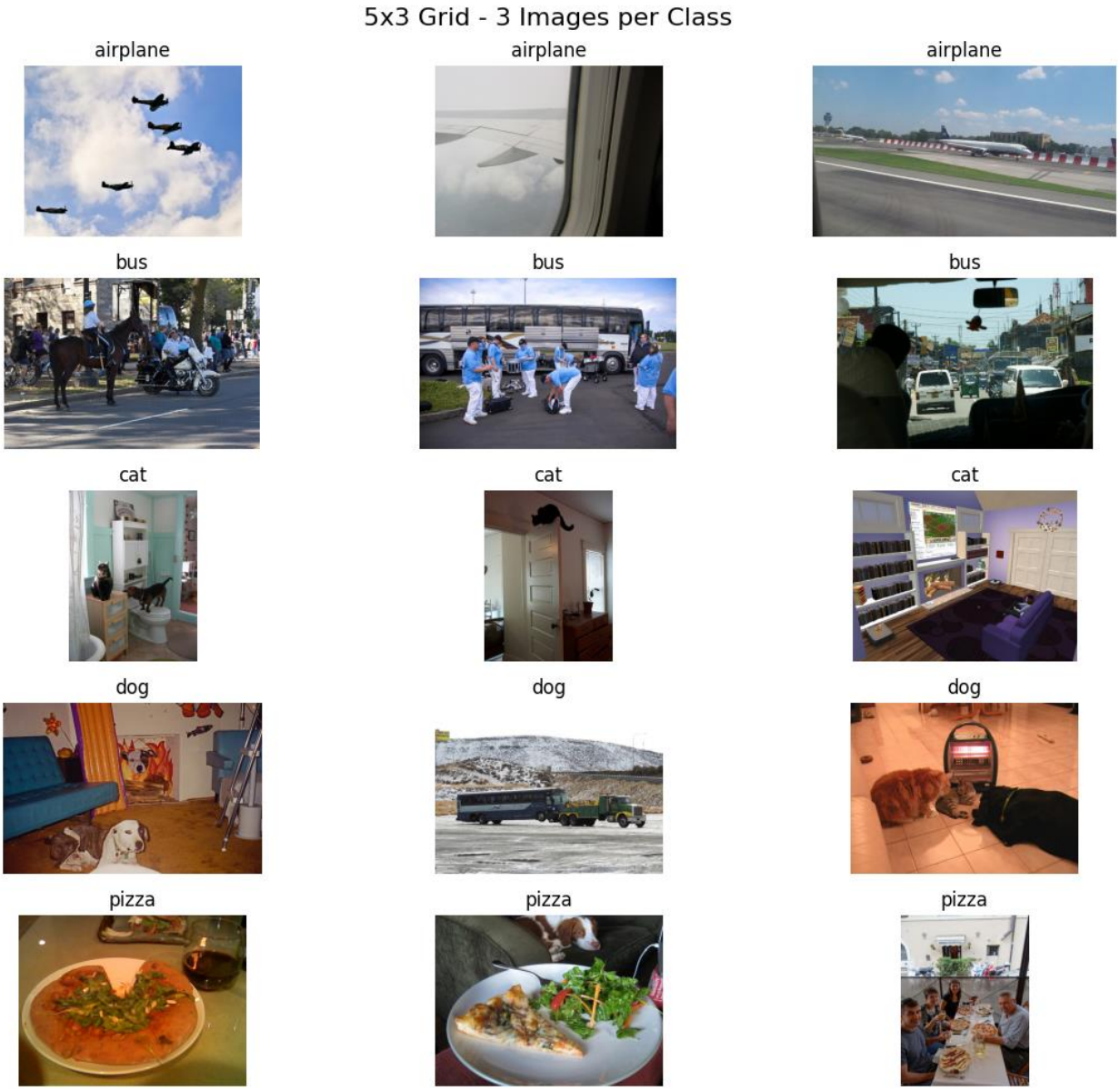
The key reason behind this difference lies in the effectiveness of downsampling methods. MaxPooling retains dominant features but often loses fine-grained spatial details, which are essential for differentiating visually similar objects. Stride-based convolutions, however, preserve spatial structure better while learning optimal feature representations, leading to more meaningful activations in deeper layers.

Stride-based downsampling is clearly the superior approach for BMEnet in this case. The MaxPool-based model fails to learn effectively, resulting in almost random predictions. In contrast, Stride-based downsampling improves gradient flow, maintains spatial information, and significantly boosts accuracy. Based on these findings, stride-based downsampling should be preferred over max pooling in BMEnet for CIFAR-10 classification.

Question 7: Skip Connections with MSCOCO

7.1 5x3 images

```
import random
import matplotlib.pyplot as plt
import torchvision.transforms as transforms
from PIL import Image

def plot_images_from_coco(root_dir, class_names, transform=None):
    plt.figure(figsize=(12, 10))
    image_count = 0

    for class_name in class_names:
        class_folder = os.path.join(root_dir, class_name)
        image_paths = glob.glob(f"{class_folder}/*.jpg")
        sampled_images = random.sample(image_paths, 3)

        for img_path in sampled_images:
            image = Image.open(img_path).convert("RGB")
            image_count += 1
            plt.subplot(5, 3, image_count)
            plt.imshow(image)
            plt.title(class_name)
            plt.axis('off')
    plt.suptitle("5x3 Grid - 3 Images per Class", fontsize=16)
    plt.tight_layout()
    plt.savefig("/content/drive/MyDrive/5x3_coco_images.png")
    plt.show()
dataset_root_dir = "/content/drive/MyDrive/coco_subset/train2017"
plot_images_from_coco(root_dir=dataset_root_dir, class_names=SELECTED_CLASSES, transform=False)
```

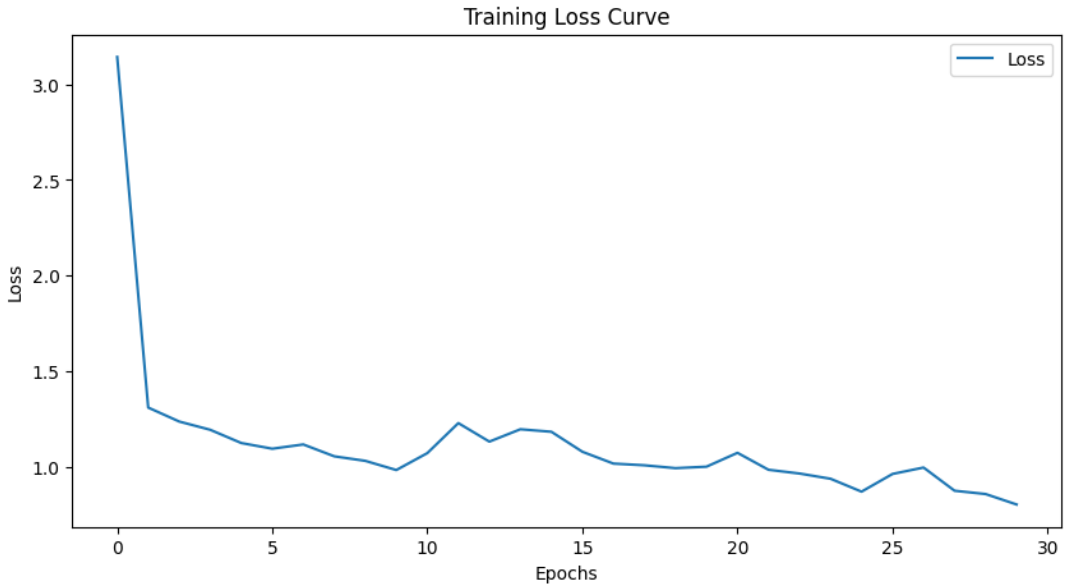


7.2 train loss curve

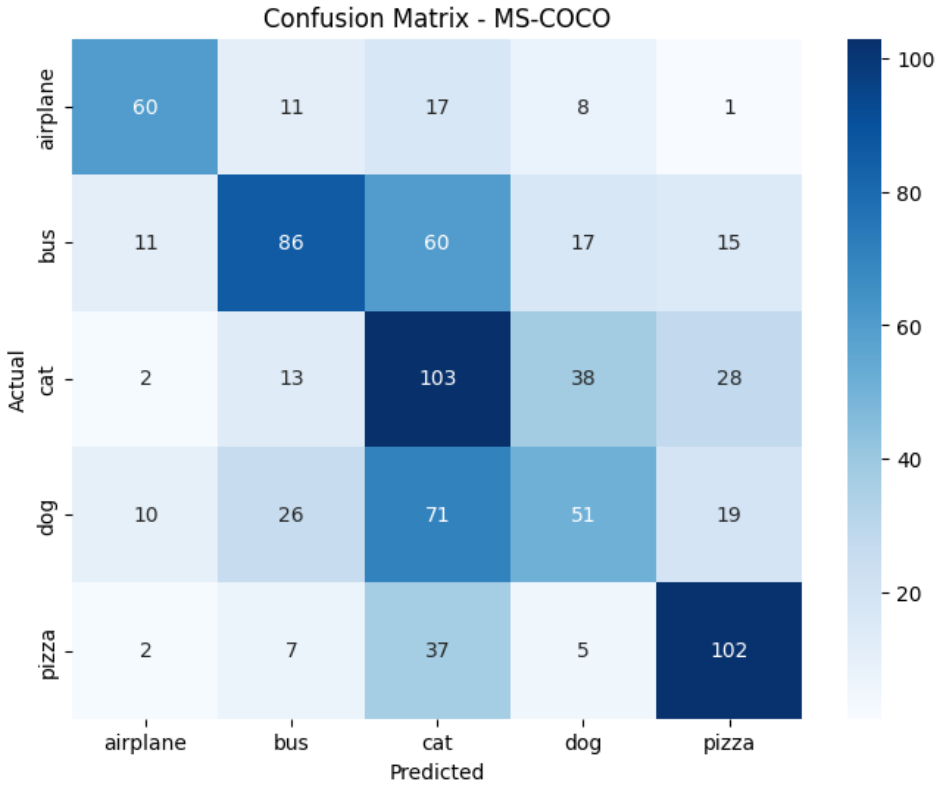
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

def test_model(model, test_loader, class_names):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)
    model.eval()
    all_preds, all_labels = [], []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
```

```
            outputs = model(images)
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
    conf_matrix = confusion_matrix(all_labels, all_preds)
    plt.figure(figsize=(8,6))
    sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=class_names,
yticklabels=class_names)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix - MS-COCO")
    plt.savefig("/content/drive/MyDrive/conf_matrix_coco.png")
    plt.show()

    class_correct = [0] * len(class_names)
    class_total = [0] * len(class_names)

    for i in range(len(all_labels)):
        label = all_labels[i]
        class_correct[label] += (all_preds[i] == label)
        class_total[label] += 1

    print("\nPer-Class Accuracy:")
    for i, class_name in enumerate(class_names):
        accuracy = 100 * class_correct[i] / class_total[i] if class_total[i] > 0 else 0
        print(f"{class_name}: {accuracy:.2f}%")

    overall_accuracy = 100 * np.sum(class_correct) / np.sum(class_total)
    print(f"\nOverall Accuracy: {overall_accuracy:.2f}%")
```

```
model = BMEnetCOCO(num_classes=5, depth=25, skip_connections=True)

loss_tally, accuracy_tally = train_model(model, train_loader, num_epochs=30, learning_rate=1e-3)

plt.figure(figsize=(10,5))
plt.plot(loss_tally, label="Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training Loss Curve")
plt.legend()
plt.savefig("/content/drive/MyDrive/loss_curve_coco.png")
plt.show()
test_model(model, test_loader, SELECTED_CLASSES)
```



7.3 conf matrix 5x5



7.4 overall accuracy

```
Overall Accuracy: 50.25%
```

7.5 per class accuracy 5x1 table

```
Per-Class Accuracy:
airplane: 61.86%
bus: 45.50%
cat: 55.98%
dog: 28.81%
pizza: 66.67%
```

7.6 observations

The results from Skip Connections on the MS-COCO dataset highlight the impact of skip connections on training dynamics and classification performance. The training loss curve shows a sharp drop in the initial epochs, indicating rapid learning at the beginning. However, after about 10 epochs, the loss reduction slows down, and minor fluctuations appear, suggesting that while the model is converging, further optimization—such as tuning the learning rate or applying regularization—could improve stability.

In terms of overall accuracy, the model achieves 50.25%, which is moderate given the complexity of the MS-COCO dataset. This suggests that skip connections are helping preserve information across layers, allowing the network to learn effectively. However, the accuracy indicates that the model still struggles with certain object classes, possibly due to overlapping features or insufficient discriminative power in deeper layers.

The confusion matrix provides more insights into per-class performance. Certain classes, such as "Cat" (103 correct predictions) and "Pizza" (102 correct predictions), are well learned by the model, indicating that their features are distinct and easily recognizable. However, classes like "Bus" and "Dog" suffer from significant misclassification, likely due to visual similarities with other classes or feature extraction limitations.

The per-class accuracy results show variation in performance across different categories. The best-performing class is "Pizza" (66.67%), likely because its distinct shape and color make it easier to classify. On the other hand, "Dog" has the lowest accuracy (28.81%), suggesting that the model struggles to distinguish it from similar animals like cats. Other categories, such as "Airplane" (61.86%), "Bus" (45.50%), and "Cat" (55.98%), show moderate accuracy, indicating that while the model extracts meaningful features, there is still room for improvement in class differentiation.

Skip connections enhance gradient flow and improve feature retention, making training more stable. However, the 50.25% accuracy and class-wise inconsistencies suggest that the model struggles with specific object categories. To improve performance, further optimizations such as hyperparameter tuning, advanced data augmentation, or the integration of attention mechanisms could be explored.

Source Code

a. hw6_1.py

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import os
os.listdir("/content/drive/MyDrive/DLStudio")
!pip install pymsgbox
```

```
import sys
sys.path.append('/content/drive/MyDrive/DLStudio/DLStudio')
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
from DLStudio import DLStudio
class SkipBlockMaxPool(nn.Module):
    def __init__(self, in_ch, out_ch, downsample=False, skip_connections=True):
        super(SkipBlockMaxPool, 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, kernel_size=3, stride=1, padding=1)
        self.convo2 = nn.Conv2d(in_ch, out_ch, kernel_size=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, kernel_size=1)

        if self.downsample:
            self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
        else:
            self.maxpool = None

    def forward(self, x):
        identity = x

        out = self.convo1(x)
        out = self.bn1(out)
        out = F.relu(out)

        out = self.convo2(out)
        out = self.bn2(out)
```

```
        out = F.relu(out)

        if self.downsample and self.maxpool is not None:
            identity = self.maxpool(identity)
            out = self.maxpool(out)

        if self.skip_connections:
            if (self.in_ch == self.out_ch) and (not self.downsample):
                out = out + identity
            elif (self.in_ch != self.out_ch) and (not self.downsample):
                identity = self.in2out(identity)
                out = out + identity
            elif (self.in_ch != self.out_ch) and self.downsample:
                out = out + torch.cat((identity, identity), dim=1)
        return out

class BMEnetMaxPool(DLStudio.BMEnet):
    def __init__(self, dl_studio, skip_connections=True, depth=8):
        super(BMEnetMaxPool, self).__init__(dl_studio)
        self.dl_studio = dl_studio
        self.depth = depth
        image_size = dl_studio.image_size
        num_ds = 0

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

        self.skip64_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip64_arr.append(DLStudio.BMEnet.SkipBlock(64, 64, skip_connections=skip_connections))

        self.skip64to128ds = SkipBlockMaxPool(64, 128, downsample=True,
skip_connections=skip_connections)
        num_ds += 1

        self.skip128_arr = nn.ModuleList()
        for i in range(self.depth):
            self.skip128_arr.append(DLStudio.BMEnet.SkipBlock(128, 128, skip_connections=skip_connections))

        self.skip128to256ds = SkipBlockMaxPool(128, 256, downsample=True,
skip_connections=skip_connections)
        num_ds += 1

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

        fc_in_features = (image_size[0] // (2 ** num_ds)) * (image_size[1] // (2 ** num_ds)) * 256
        self.fc1 = nn.Linear(fc_in_features, 1000)
        self.fc2 = nn.Linear(1000, 10)
```

```
    def forward(self, x):
        x = F.relu(self.conv(x))
        for block in self.skip64_arr:
            x = block(x)
        x = self.skip64to128ds(x)
        for block in self.skip128_arr:
            x = block(x)
        x = self.skip128to256ds(x)
        for block in self.skip256_arr:
            x = block(x)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x


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

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd


def main():
    dls = DLStudio(
        dataroot="./data/CIFAR-10/",
        image_size=[32, 32],
        path_saved_model="./saved_model_model.pt",
        momentum=0.9,
        learning_rate=1e-4,
        epochs=6,
        batch_size=4,
        classes=('plane','car','bird','cat','deer','dog','frog','horse','ship','truck'),
        use_gpu=True
    )

    print("Experiment 1: Original BMEnet with skip connections (convolution-based downsampling)")
    model1 = dls.BMEnet(dls, skip_connections=True, depth=8)
    model1.load_cifar_10_dataset()
    params1 = sum(p.numel() for p in model1.parameters() if p.requires_grad)
    print("Learnable parameters (Original BMEnet):", params1)

    model1.run_code_for_training(model1, display_images=False)
    model1.run_code_for_testing(model1, display_images=False)

    print("Experiment 2: BMEnet variant with maxpool downsampling")
    model2 = BMEnetMaxPool(dls, skip_connections=True, depth=8)
```

```
    model2.load_cifar_10_dataset()
    params2 = sum(p.numel() for p in model2.parameters() if p.requires_grad)
    print("Learnable parameters (BMEnet with MaxPool downsampling):", params2)

    model2.run_code_for_training(model2, display_images=False)
    model2.run_code_for_testing(model2, display_images=False)
```

```
if __name__ == '__main__':
    main()
```

b. hw6_2.py

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import os
os.listdir("/content/drive/MyDrive/DLStudio")
!pip install pymsgbox
```

```
import sys
sys.path.append('/content/drive/MyDrive/DLStudio/DLStudio')
```

```
!pip install pycocotools
!pip install torchvision
```

```
import os
DATA_DIR = "/content/drive/MyDrive/coco_subset"
TRAIN_DIR = os.path.join(DATA_DIR, "train2017")
VAL_DIR = os.path.join(DATA_DIR, "val2017")
ANN_DIR = os.path.join(DATA_DIR, "annotations")
os.makedirs(TRAIN_DIR, exist_ok=True)
os.makedirs(VAL_DIR, exist_ok=True)
os.makedirs(ANN_DIR, exist_ok=True)
print(f"Dataset will be stored in: {DATA_DIR}")
```

```
!wget http://images.cocodataset.org/annotations/annotations_trainval2017.zip -P /content/
!unzip /content/annotations_trainval2017.zip -d /content/
!mv /content/annotations /content/drive/MyDrive/coco_subset/
```

```
from pycocotools.coco import COCO
import requests
import os
import json
```

```
DATA_DIR = "/content/drive/MyDrive/coco_subset"
TRAIN_DIR = os.path.join(DATA_DIR, "train2017")
VAL_DIR = os.path.join(DATA_DIR, "val2017")
ANN_DIR = os.path.join(DATA_DIR, "annotations")

SELECTED_CLASSES = [ 'airplane', 'bus', 'cat', 'dog','pizza']
NUM_TRAIN = 1500
NUM_TEST = 500

train_coco = COCO(os.path.join(ANN_DIR, "instances_train2017.json"))
val_coco = COCO(os.path.join(ANN_DIR, "instances_val2017.json"))

cat_to_id = {cat: train_coco.getCatIds(catNms=[cat])[0] for cat in SELECTED_CLASSES}

def download_coco_images(coco, save_dir, num_images_per_class):
    os.makedirs(save_dir, exist_ok=True)

    for class_name, class_id in cat_to_id.items():
        class_folder = os.path.join(save_dir, class_name)
        os.makedirs(class_folder, exist_ok=True)

        img_ids = coco.getImgIds(catIds=class_id)[:num_images_per_class]

        for img_id in img_ids:
            img_info = coco.loadImgs(img_id)[0]
            img_url = img_info['coco_url']
            img_path = os.path.join(class_folder, img_info['file_name'])

            if not os.path.exists(img_path):
                img_data = requests.get(img_url).content
                with open(img_path, 'wb') as f:
                    f.write(img_data)

        print(f"Downloaded {num_images_per_class} images for {class_name} into {class_folder}")

download_coco_images(train_coco, TRAIN_DIR, NUM_TRAIN)
download_coco_images(val_coco, VAL_DIR, NUM_TEST)
```

```
import random
import matplotlib.pyplot as plt
import torchvision.transforms as transforms
from PIL import Image
import glob
def plot_images_from_coco(root_dir, class_names, transform=None):
    plt.figure(figsize=(12, 10))
    image_count = 0

    for class_name in class_names:
```

```
        class_folder = os.path.join(root_dir, class_name)
        image_paths = glob.glob(f"{class_folder}/*.jpg")
        sampled_images = random.sample(image_paths, 3)

        for img_path in sampled_images:
            image = Image.open(img_path).convert("RGB")
            image_count += 1
            plt.subplot(5, 3, image_count)
            plt.imshow(image)
            plt.title(class_name)
            plt.axis('off')
    plt.suptitle("5x3 Grid - 3 Images per Class", fontsize=16)
    plt.tight_layout()
    plt.savefig("/content/drive/MyDrive/5x3_coco_images.png")
    plt.show()
dataset_root_dir = "/content/drive/MyDrive/coco_subset/train2017"
plot_images_from_coco(root_dir=dataset_root_dir, class_names=SELECTED_CLASSES, transform=False)
```

```
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.utils.data import Dataset, DataLoader
import PIL.Image as Image
import glob
from DLStudio import DLStudio
SELECTED_CLASSES = [ 'airplane', 'bus', 'cat', 'dog', 'pizza']
transform = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

class CocoSubset(Dataset):
    def __init__(self, root_dir, class_names, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.class_names = class_names
        self.image_paths = []
        self.labels = []

        for label, class_name in enumerate(class_names):
            class_images = glob.glob(f"{root_dir}/{class_name}/*.jpg")
            self.image_paths.extend(class_images)
            self.labels.extend([label] * len(class_images))

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

    def __getitem__(self, idx):
```

```
        img_path = self.image_paths[idx]
        image = Image.open(img_path).convert("RGB")
        label = self.labels[idx]

        if self.transform:
            image = self.transform(image)

        return image, label

train_dataset = CocoSubset(root_dir=TRAIN_DIR, class_names=SELECTED_CLASSES,
transform=transform)
test_dataset = CocoSubset(root_dir=VAL_DIR, class_names=SELECTED_CLASSES, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=2)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False, num_workers=2)

print("COCO Dataset Loaded Successfully!")
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from DLStudio import DLStudio
#I have used Dr. Kaks Code as the base for this part of the assignment
class BMEnetCOCO(nn.Module):

    def __init__(self, num_classes=5, depth=30, skip_connections=True):
        super(BMEnetCOCO, self).__init__()
        self.depth = depth

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)

        self.skip64_blocks = nn.ModuleList()
        for _ in range(15):
            self.skip64_blocks.append(DLStudio.BMEnet.SkipBlock(64, 64, skip_connections=skip_connections))
        self.skip64to128 = DLStudio.BMEnet.SkipBlock(64, 128, downsample=True,
skip_connections=skip_connections)
        self.skip128_blocks = nn.ModuleList()
        for _ in range(8):
            self.skip128_blocks.append(DLStudio.BMEnet.SkipBlock(128, 128,
skip_connections=skip_connections))

        self.skip128to256 = DLStudio.BMEnet.SkipBlock(128, 256, downsample=True,
skip_connections=skip_connections)

        self.skip256_blocks = nn.ModuleList()
        for _ in range(7):
            self.skip256_blocks.append(DLStudio.BMEnet.SkipBlock(256, 256,
skip_connections=skip_connections))
```

```
        self.fc1 = nn.Linear(256 * 8 * 8, 1024)
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(1024, num_classes)

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))

        for skip in self.skip64_blocks:
            x = skip(x)

        x = self.skip64to128(x)

        for skip in self.skip128_blocks:
            x = skip(x)

        x = self.skip128to256(x)

        for skip in self.skip256_blocks:
            x = skip(x)

        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

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

def train_model(model, train_loader, num_epochs=10, learning_rate=1e-2):

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    loss_tally = []
    accuracy_tally = []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct_predictions = 0
        total_predictions = 0
```

```
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

            _, preds = torch.max(outputs, 1)
            correct_predictions += (preds == labels).sum().item()
            total_predictions += labels.size(0)

        epoch_loss = running_loss / len(train_loader)
        epoch_accuracy = 100 * correct_predictions / total_predictions

        loss_tally.append(epoch_loss)
        accuracy_tally.append(epoch_accuracy)
        print(f"Epoch {epoch+1}/{num_epochs} | Loss: {epoch_loss:.4f} | Accuracy: {epoch_accuracy:.2f}%")

    print(" Training complete!")
    return loss_tally, accuracy_tally
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

def test_model(model, test_loader, class_names):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)
    model.eval()
    all_preds, all_labels = [], []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
    conf_matrix = confusion_matrix(all_labels, all_preds)
    plt.figure(figsize=(8,6))
    sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=class_names,
yticklabels=class_names)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
```

```
    plt.title("Confusion Matrix - MS-COCO")
    plt.savefig("/content/drive/MyDrive/conf_matrix_coco.png")
    plt.show()

    class_correct = [0] * len(class_names)
    class_total = [0] * len(class_names)

    for i in range(len(all_labels)):
        label = all_labels[i]
        class_correct[label] += (all_preds[i] == label)
        class_total[label] += 1

    print("\nPer-Class Accuracy:")
    for i, class_name in enumerate(class_names):
        accuracy = 100 * class_correct[i] / class_total[i] if class_total[i] > 0 else 0
        print(f"{class_name}: {accuracy:.2f}%")

    overall_accuracy = 100 * np.sum(class_correct) / np.sum(class_total)
    print(f"\nOverall Accuracy: {overall_accuracy:.2f}%")
```

```
model = BMEnetCOCO(num_classes=5, depth=25, skip_connections=True)

loss_tally, accuracy_tally = train_model(model, train_loader, num_epochs=30, learning_rate=1e-3)

plt.figure(figsize=(10,5))
plt.plot(loss_tally, label="Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training Loss Curve")
plt.legend()
plt.savefig("/content/drive/MyDrive/loss_curve_coco.png")
plt.show()

test_model(model, test_loader, SELECTED_CLASSES)
```