A.6

Note that for better visualization and comparison purpose we only plotted 6 set of the hyper-parameters for each models.

A.6.a

Fully-connected output, 0 hidden layers (logistic regression): this network has no hidden layers and linearly maps the input layer to the output layer. This can be written as

$$x^{\text{out}} = W(x^{\text{in}}) + b$$

where $x^{\text{out}} \in \mathbb{R}^{10}$, $x^{\text{in}} \in \mathbb{R}^{32\times32\times3}$, $W \in \mathbb{R}^{10\times3072}$, $b \in \mathbb{R}^{10}$ since $3072 = 32 \cdot 32 \cdot 3$. The hyper-parameters we try to tune in this subsection is batch size and learning rate, the experiments are plotted below:

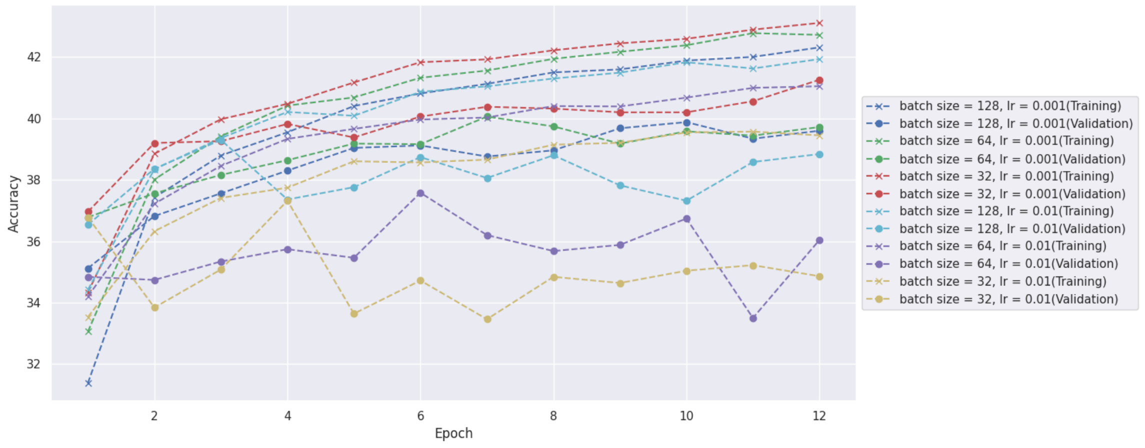

Figure 13: Plot of the training accuracies and validation accuracies versus epoch.

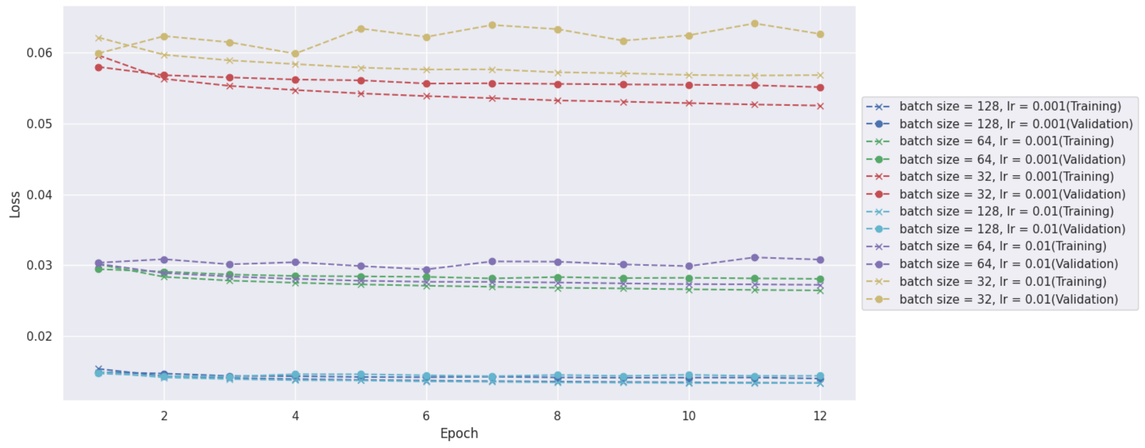

Figure 14: Plot of the training losses and validation losses versus epoch.

The best model from our experiments is the one with batch size of 32 and learning rate of 0.001 reaching a training accuracies of 43.153% and testing accuracies of 39.710% after 12 epochs.

A.6.b

Fully-connected output, 1 fully-connected hidden layer: this network has one hidden layer denoted as $x^{\text{hidden}} \in \mathbb{R}^M$ where M will be a hyperparameter you choose (M could be in the hundreds). The non-linearity applied to the hidden layer will be the `relu` ($\text{relu}(x) = \max\{0, x\}$. This network can be written as

$$x^{\text{out}} = W_2 \text{relu}(W_1(x^{\text{in}}) + b_1) + b_2$$

where $W_1 \in \mathbb{R}^{M \times 3072}$, $b_1 \in \mathbb{R}^M$, $W_2 \in \mathbb{R}^{10 \times M}$, $b_2 \in \mathbb{R}^{10}$. The hyper-parameters we try to tune in this subsection is batch size, learning rate and hidden size, the experiments are plotted below:

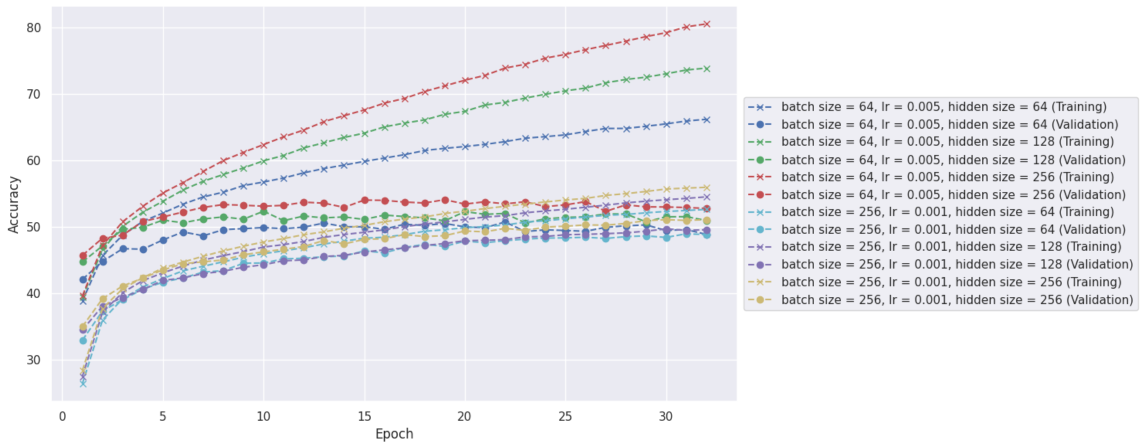

Figure 15: Plot of the training accuracies and validation accuracies versus epoch.

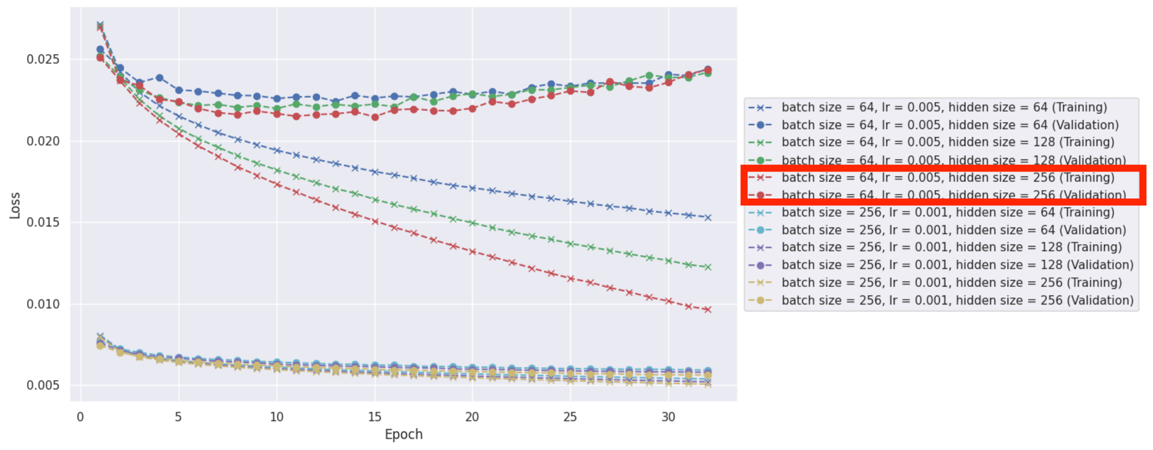

Figure 16: Plot of the training losses and validation losses versus epoch.

The best model from our experiments is the one with batch size of 64, learning rate of 0.005 and hidden size of 256 reaching a training accuracies of 80.787% and testing accuracies of 52.000% after 32 epochs.

A.6.c

Convolutional layer with max-pool and fully-connected output: for a convolutional layer W_1 with filters of size $k \times k \times 3$, and M filters (reasonable choices are $M = 100$, $k = 5$), we have that $\text{Conv2d}(x^{\text{in}}, W_1) \in \mathbb{R}^{(33-k)\times(33-k)\times M}$. This network can be written as

$$x^{\text{output}} = W_2(\text{MaxPool}(\text{relu}(\text{Conv2d}(x^{\text{input}}, W_1) + b_1))) + b_2$$

where $W_2 \in \mathbb{R}^{10\times M(\lfloor \frac{33-k}{N} \rfloor)^2}$, $b_2 \in \mathbb{R}^{10}$. The hyper-parameters we try to tune in this subsection is batch size, learning rat, hidden size and kernel size, the experiments are plotted below:

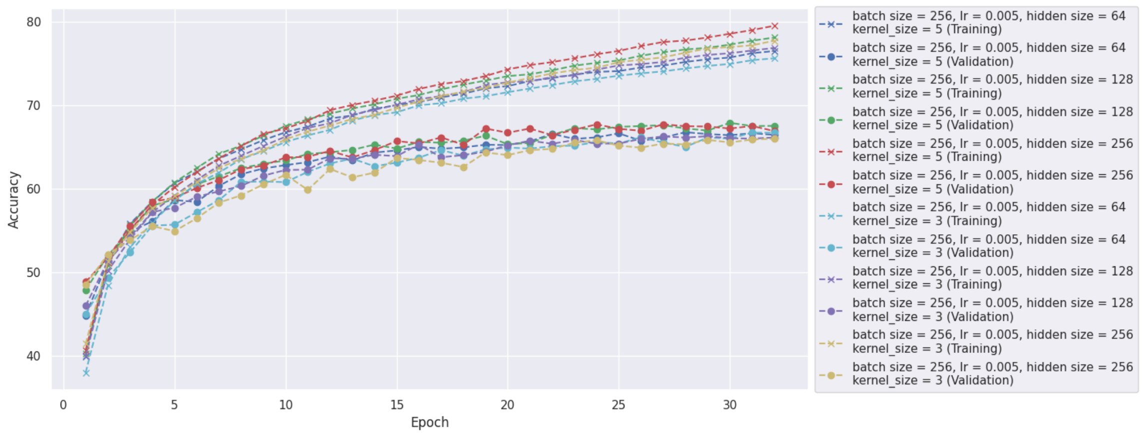

Figure 17: Plot of the training accuracies and validation accuracies versus epoch.

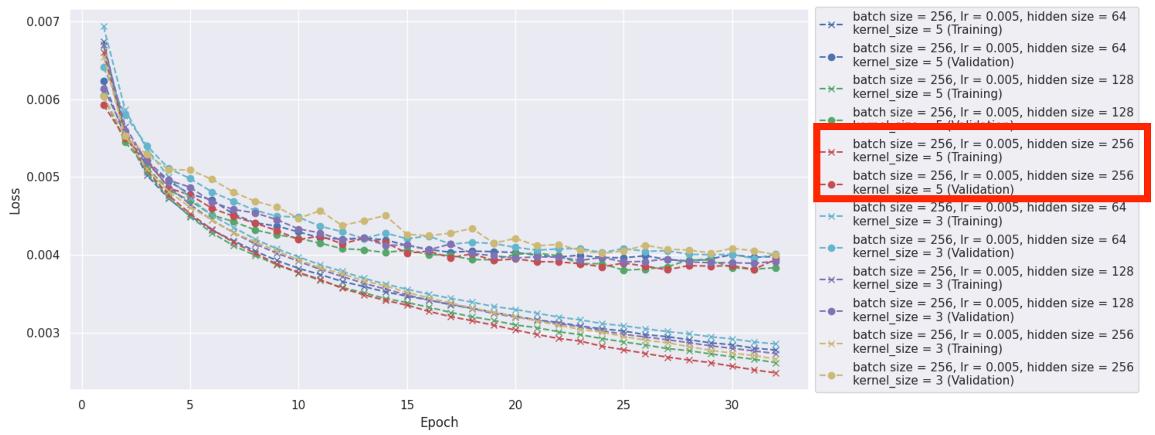

Figure 18: Plot of the training losses and validation losses versus epoch.

The best model from our experiments is the one with batch size of 256, learning rate of 0.005, hidden size of 256 and kernel size of 5 reaching a training accuracies of 79.524% and testing accuracies of 66.170% after 32 epochs.

A.6.d

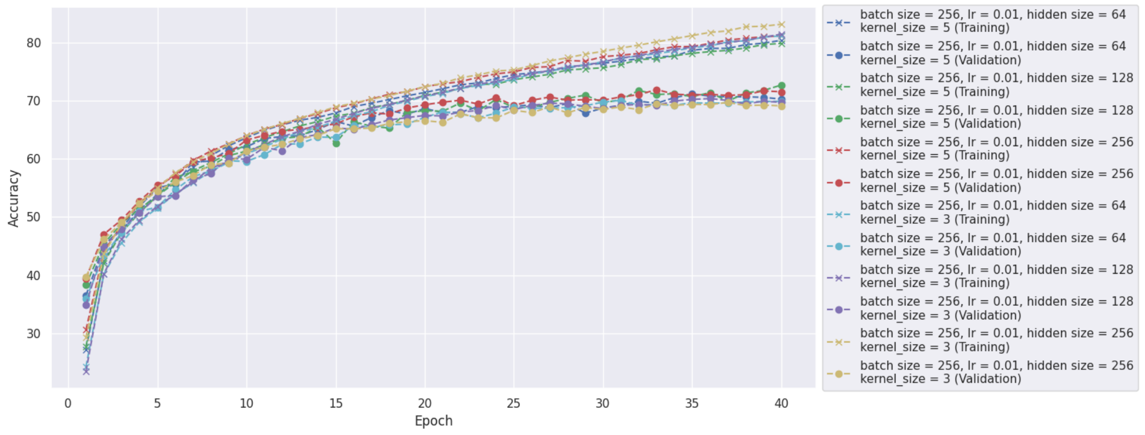

Figure 19: Plot of the training accuracies and validation accuracies versus epoch.

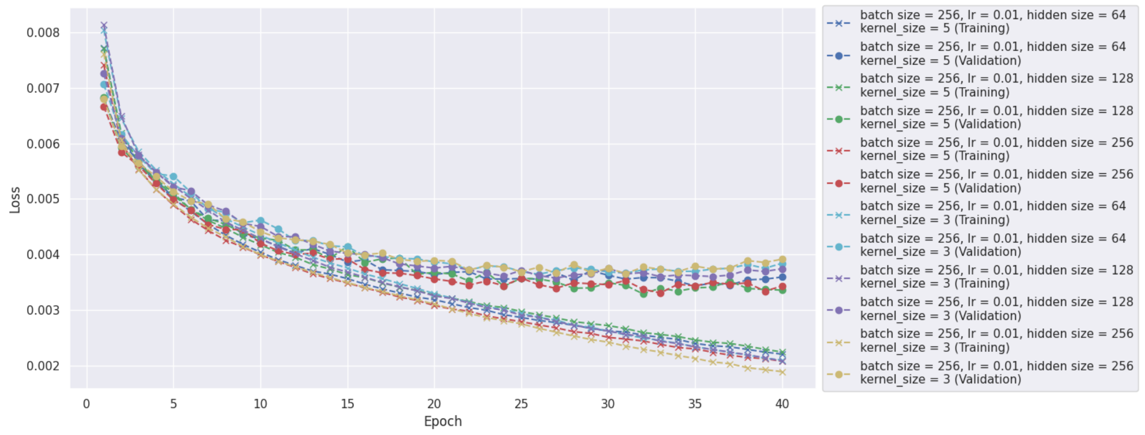

Figure 20: Plot of the training losses and validation losses versus epoch.

The best model from our experiments is the one with batch size of 256, learning rate of 0.01, hidden size of 256 and kernel size of 5 reaching a training accuracies of 79.822% and testing accuracies of 71.220% after 40 epochs.

A.6.model

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


class A6a(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(32 * 32 * 3, 10)

    def forward(self, x):
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = self.fc1(x)
        return F.log_softmax(x, dim=1)


class A6b(nn.Module):
    def __init__(self, hidden_size=64):
        super().__init__()
        self.hidden_size = hidden_size
        self.fc1 = nn.Linear(32 * 32 * 3, self.hidden_size)
        self.fc2 = nn.Linear(self.hidden_size, 10)

    def forward(self, x):
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)


class A6c(nn.Module):
    def __init__(self, hidden_size=64, kernel_size=5):
        super().__init__()
        self.hidden_size = hidden_size
        self.kernel_size = kernel_size
        self.conv1 = nn.Conv2d(3, self.hidden_size, self.kernel_size)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(self.hidden_size * ((33 - self.kernel_size)//2) **
            2, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.pool(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)

        return F.log_softmax(x, dim=1)

class A6d(nn.Module):
    def __init__(self, hidden_size=128, kernel_size=5):
        super().__init__()
        self.hidden_size = hidden_size
        self.kernel_size = kernel_size
        self.conv1 = nn.Conv2d(3, self.hidden_size, self.kernel_size)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(self.hidden_size, 16, self.kernel_size)
        self.fc1 = nn.Linear(16 * ((33 - self.kernel_size)//5) ** 2, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)  # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

A.6.code

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from tqdm import tqdm

from A6_model import A6a, A6b, A6c, A6d

sns.set()


np.random.seed(1968990 + 20210518)
torch.manual_seed(1968990 + 20210518)


def prepare_dataset(batch_size=64, train_val_split_ratio=0.9):

    transform = transforms.Compose([
        # transforms.Resize(256),
        # transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
            0.225]),
    ])

    cifar10_set = datasets.CIFAR10(root='./data', train=True, download=False,
        transform=transform)
    train_size = int(len(cifar10_set) * train_val_split_ratio)
    val_size = len(cifar10_set) - train_size
    cifar10_trainset, cifar10_valset = torch.utils.data.random_split(cifar10_set
        , [train_size, val_size])
    cifar10_testset = datasets.CIFAR10(root='./data', train=False, download=
        False, transform=transform)

    train_loader = torch.utils.data.DataLoader(cifar10_trainset, batch_size=
        batch_size, shuffle=True)
    val_loader = torch.utils.data.DataLoader(cifar10_valset, batch_size=
        batch_size, shuffle=True)
    test_loader = torch.utils.data.DataLoader(cifar10_testset, batch_size=
        batch_size, shuffle=True)

    return train_loader, val_loader, test_loader


def train(epochs, model, train_loader, val_loader, criterion, optimizer,
    batch_size):
    train_losses = []
    val_losses = []
    train_accs = []
    val_accs =[]
    model.train()
    for epoch in range(epochs):
        running_loss = 0.0
        correct = 0
        total = 0
        for i, data in enumerate(train_loader):
            inputs, labels = data[0].to("cuda"), data[1].to("cuda")
            optimizer.zero_grad()

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            torch.nn.utils.clip_grad_norm_(model.parameters(), 1)
            optimizer.step()
            running_loss += loss.item()

            _, predicted = torch.max(outputs.data, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

        train_accs.append(100 * correct / total)
        train_losses.append(running_loss / total)
        print('[%d] Train Accuracy: %.3f %% Train Loss: %.3f' % (epoch + 1, 100
            * correct / total, running_loss / total))
        running_loss = 0.0
        val_acc, val_loss = eval(epoch, model, val_loader, criterion)
        val_accs.append(val_acc)
        val_losses.append(val_loss)

    return train_accs, val_accs, train_losses, val_losses
```

```
def eval(epoch, model, eval_loader, criterion):
    running_loss = 0.0
    correct = 0
    total = 0
    for i, data in enumerate(eval_loader):
        inputs, labels = data[0].to("cuda"), data[1].to("cuda")

        outputs = model(inputs)
        loss = criterion(outputs, labels)
        running_loss += loss.item()

        _, predicted = torch.max(outputs.data, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    print('[%d] Val Accuracy: %.3f %% Val Loss: %.3f' % (epoch + 1, 100 *
       correct / total, running_loss / total))
    return 100 * correct / total, running_loss / total


def plot_acc(train_acc, val_acc, labels, figname):
    epochsx = [int(x) for x in np.arange(1, len(train_acc[0])+1)]
    plt.figure(figsize=(15, 6))
    COLORS = ['b', 'g', 'r', 'c', 'm', 'y']
    for tacc, vacc, label, color in zip(train_acc, val_acc, labels, COLORS):
        plt.plot(epochsx, tacc, '--x', label=label + ' (Training)', color=color)
        plt.plot(epochsx, vacc, '--o', label=label + ' (Validation)', color=
            color)
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.tight_layout()
    plt.savefig(figname)


def plot_loss(train_loss, val_loss, labels, figname):
    epochsx = [int(x) for x in np.arange(1, len(train_loss[0])+1)]
    plt.figure(figsize=(15, 6))
    COLORS = ['b', 'g', 'r', 'c', 'm', 'y']
    for tloss, vloss, label, color in zip(train_loss, val_loss, labels, COLORS):
        plt.plot(epochsx, tloss, '--x', label=label + ' (Training)', color=color
            )
        plt.plot(epochsx, vloss, '--o', label=label + ' (Validation)', color=
            color)
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.tight_layout()
    plt.savefig(figname)


def run_A6a(epochs=12):

    hyper_params = [
        # {'batch_size': 128, 'lr': 1e-3},
        # {'batch_size': 64, 'lr': 1e-3},
        {'batch_size': 32, 'lr': 1e-3},
        {'batch_size': 128, 'lr': 1e-2},
        {'batch_size': 64, 'lr': 1e-2},
        {'batch_size': 32, 'lr': 1e-2}
    ]

    all_train_accuracies = []
    all_val_accuracies = []
    all_train_losses = []
    all_val_losses = []
    labels = []
    for param in hyper_params:
        batch_size = param['batch_size']
        lr = param['lr']

        train_loader, val_loader, test_loader = prepare_dataset(batch_size=
            batch_size)

        model = A6a()
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9)
        model.to("cuda")

        train_accuracies, val_accuracies, train_losses, val_losses = train(
            epochs, model, train_loader, val_loader, criterion, optimizer,
            batch_size)

        all_train_accuracies.append(train_accuracies)
        all_val_accuracies.append(val_accuracies)
        all_train_losses.append(train_losses)
        all_val_losses.append(val_losses)
        labels.append('batch size = {}, lr = {}'.format(batch_size, lr))
        eval(-1, model, test_loader, criterion)

    plot_acc(all_train_accuracies, all_val_accuracies, labels, 'A6a_acc.png')
    plot_loss(all_train_losses, all_val_losses, labels, 'A6a_loss.png')


def run_A6b(epochs=32):

    hyper_params = [
        {'batch_size': 64, 'lr': 5e-3, 'hidden_size': 64},
        {'batch_size': 64, 'lr': 5e-3, 'hidden_size': 128},
        {'batch_size': 64, 'lr': 5e-3, 'hidden_size': 256},
        {'batch_size': 256, 'lr': 1e-3, 'hidden_size': 64},
        {'batch_size': 256, 'lr': 1e-3, 'hidden_size': 128},
        {'batch_size': 256, 'lr': 1e-3, 'hidden_size': 256},
    ]

    all_train_accuracies = []
    all_val_accuracies = []
    all_train_losses = []
    all_val_losses = []
    labels = []
    for param in hyper_params:
        batch_size = param['batch_size']
        lr = param['lr']
        hidden_size = param['hidden_size']

        train_loader, val_loader, test_loader = prepare_dataset(batch_size=
            batch_size)

        model = A6b(hidden_size=hidden_size)
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9)
        model.to("cuda")

        train_accuracies, val_accuracies, train_losses, val_losses = train(
            epochs, model, train_loader, val_loader, criterion, optimizer,
            batch_size)

        all_train_accuracies.append(train_accuracies)
        all_val_accuracies.append(val_accuracies)
        all_train_losses.append(train_losses)
        all_val_losses.append(val_losses)
        labels.append('batch size = {}, lr = {}, hidden size = {}'.format(
            batch_size, lr, hidden_size))
        eval(-1, model, test_loader, criterion)

    plot_acc(all_train_accuracies, all_val_accuracies, labels, 'A6b_acc.png')
    plot_loss(all_train_losses, all_val_losses, labels, 'A6b_loss.png')


def run_A6c(epochs=32):

    hyper_params = [
        {'batch_size': 256, 'lr': 5e-3, 'hidden_size': 64, 'kernel_size': 5},
        {'batch_size': 256, 'lr': 5e-3, 'hidden_size': 128, 'kernel_size': 5},
        {'batch_size': 256, 'lr': 5e-3, 'hidden_size': 256, 'kernel_size': 5},
        {'batch_size': 256, 'lr': 5e-3, 'hidden_size': 64, 'kernel_size': 3},
        {'batch_size': 256, 'lr': 5e-3, 'hidden_size': 128, 'kernel_size': 3},
        {'batch_size': 256, 'lr': 5e-3, 'hidden_size': 256, 'kernel_size': 3},
    ]
    best_param = [
        {'batch_size': 256, 'lr': 5e-3, 'hidden_size': 256, 'kernel_size': 5}
    ]

    all_train_accuracies = []
    all_val_accuracies = []
    all_train_losses = []
    all_val_losses = []
    labels = []
    for param in hyper_params:
        print(param)
        batch_size = param['batch_size']
        lr = param['lr']
        hidden_size = param['hidden_size']
        kernel_size = param['kernel_size']
        train_loader, val_loader, test_loader = prepare_dataset(batch_size=
            batch_size)

        model = A6c(hidden_size=hidden_size, kernel_size=kernel_size)
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9,
            weight_decay=1e-5)
        model.to("cuda")

        train_accuracies, val_accuracies, train_losses, val_losses = train(
            epochs, model, train_loader, val_loader, criterion, optimizer,
            batch_size)

        all_train_accuracies.append(train_accuracies)
        all_val_accuracies.append(val_accuracies)
        all_train_losses.append(train_losses)
        all_val_losses.append(val_losses)
        labels.append('batch size = {}, lr = {}, hidden size = {}\nkernel_size =
             {}'.format(batch_size, lr, hidden_size, kernel_size))
        eval(-1, model, test_loader, criterion)

    plot_acc(all_train_accuracies, all_val_accuracies, labels, 'A6c_acc.png')
    plot_loss(all_train_losses, all_val_losses, labels, 'A6c_loss.png')


def run_A6d(epochs=40, batch_size=128, lr=1e-2, momentum=0.9, hidden_size=256):
    hyper_params = [
        {'batch_size': 256, 'lr': 1e-2, 'hidden_size': 64, 'kernel_size': 5},
        {'batch_size': 256, 'lr': 1e-2, 'hidden_size': 128, 'kernel_size': 5},
        {'batch_size': 256, 'lr': 1e-2, 'hidden_size': 256, 'kernel_size': 5},
        {'batch_size': 256, 'lr': 1e-2, 'hidden_size': 64, 'kernel_size': 3},
        {'batch_size': 256, 'lr': 1e-2, 'hidden_size': 128, 'kernel_size': 3},
        {'batch_size': 256, 'lr': 1e-2, 'hidden_size': 256, 'kernel_size': 3},
    ]
    best_param = [
        {'batch_size': 256, 'lr': 5e-3, 'hidden_size': 256, 'kernel_size': 5}
    ]

    all_train_accuracies = []
    all_val_accuracies = []
    all_train_losses = []
    all_val_losses = []
    labels = []
    for param in hyper_params:
        print(param)
        batch_size = param['batch_size']
        lr = param['lr']
        hidden_size = param['hidden_size']
        kernel_size = param['kernel_size']

        train_loader, val_loader, test_loader = prepare_dataset(batch_size=
            batch_size)

        model = A6d(hidden_size=hidden_size, kernel_size=kernel_size)
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9,
            weight_decay=1e-5)
        model.to("cuda")

        train_accuracies, val_accuracies, train_losses, val_losses = train(
            epochs, model, train_loader, val_loader,
                                                                                        criterion
                                                                                            ,
                                                                                            optimizer
                                                                                            ,
                                                                                            batch_size
                                                                                            )

        all_train_accuracies.append(train_accuracies)
        all_val_accuracies.append(val_accuracies)
        all_train_losses.append(train_losses)
        all_val_losses.append(val_losses)
        labels.append('batch size = {}, lr = {}, hidden size = {}\nkernel_size =
             {}'.format(batch_size, lr, hidden_size,



        eval(-1, model, test_loader, criterion)

    plot_acc(all_train_accuracies, all_val_accuracies, labels, 'A6d_acc.png')
    plot_loss(all_train_losses, all_val_losses, labels, 'A6d_loss.png')



def main():
    # run_A6a()
    # run_A6b()
    run_A6c()
    # run_A6d()

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