Jonathon Nguyen

ID: 801093003

Homework 3

In [1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as f
from torchvision import datasets
import torch.optim as optim

import time
import datetime
import numpy as np
from PIL import Image
import pandas as pd
from matplotlib import pyplot as plt
```

Problem 1a

Build a Convolutional Neural Network, like what we built in lectures (without skip connections), to classify the images across all 10 classes in CIFAR 10.

You need to adjust the fully connected layer at the end properly with respect to the number of output classes. Train your network for 300 epochs.

Report your training time, training loss, and evaluation accuracy after 300 epochs. Analyze your results in your report and compare them against a fully connected network (homework 2) on training time, achieved accuracy, and model size. Make sure to submit your code by providing the GitHub URL of your course repository for this course.

In [2]:
```
class CNN(nn.Module):
    def __init__(self, n_channels1 = 32):
        super().__init__()
        self.n_channels1 = n_channels1
        self.conv1 = nn.Conv2d(3, self.n_channels1, kernel_size = 3, padding = 1)
        self.conv2 = nn.Conv2d(n_channels1, (self.n_channels1 // 2), kernel_size = 3, padd
        self.fc1 = nn.Linear(8 * 8 * (self.n_channels1 // 2), 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = f.max_pool2d(torch.relu(self.conv1(x)), 2)
        out = f.max_pool2d(torch.relu(self.conv2(out)), 2)
        out = out.view(-1, 8 * 8 * (self.n_channels1 // 2))
        out = torch.relu(self.fc1(out))
        out = self.fc2(out)
        return out
```

In [3]:
```
def training_loop(epochs, optimizer, model, loss_fn, train_loader, val_loader):
    training_losses = []
    val_losses = []
    train_accuracies = []
    val_accuracies = []
    for epoch in range(1, epochs + 1):
        # Temp vars for use in finding the accuracy.
        val_correct_labels = 0
        val_count = 0
```

```python
        loss_val_value = 0

        #Set the model to inference mode
        model.eval()
        with torch.no_grad():
            for imgs, labels in val_loader:
                # Move the data to correct device
                imgs = imgs.to(device=device)
                labels = labels.to(device=device)

                # Pass imgs through the model and find the loss.
                output = model(imgs)
                loss_val = loss_fn(output, labels)
                loss_val_value += float(loss_val)

                # Find the accurcey of the model.
                _, predicted = torch.max(output, dim=1)
                val_count += labels.shape[0]
                val_correct_labels += int((predicted == labels).sum())

            # Store the loss and accuracy.
            loss_val_value /= len(val_loader)
            val_losses.append(loss_val_value)
            val_accuracies.append(val_correct_labels/val_count)

        #Set the model to training mode.
        model.train()
        train_correct_labels = 0
        train_count = 0
        loss_train_value = 0
        for imgs, labels in train_loader:
            # Move the data to correct device
            imgs = imgs.to(device=device)
            labels = labels.to(device=device)

            # Pass imgs through the model and find the loss.
            output = model(imgs)
            loss_train = loss_fn(output, labels)
            loss_train_value += float(loss_train)

            # Adject the params
            optimizer.zero_grad()
            loss_train.backward()
            optimizer.step()

            # Find the accurcey of the model.
            _, predicted = torch.max(output, dim=1)
            train_count += labels.shape[0]
            train_correct_labels += int((predicted == labels).sum())

        # Store the loss
        loss_train_value /= len(train_loader)
        training_losses.append(loss_train_value)
        train_accuracies.append(train_correct_labels/train_count)

        # Print out the loss every 10 epoch
        if epoch % 10 == 0 or epoch == 1:
            print(f"Epoch: {epoch}, Training Loss: {loss_train_value}", end="")
            print(f", Validation Loss: {loss_val_value}, Train Accuracy: {(train_correct_l
            print(f"Validation Accuracy: {(val_correct_labels/val_count)*100}%")

    return training_losses, val_losses, train_accuracies, val_accuracies
```

In [4]:

```python
from torchvision import transforms
```

```python
transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4915, 0.4823, 0.4468),
                         (0.2470, 0.2435, 0.2616))
])
```

In [5]:

```python
# Download the cifar10 dataset.
data = '.\cifar10'
cirfar10_train = datasets.CIFAR10(data, train=True, download=True, transform=transforms)
cirfar10_val = datasets.CIFAR10(data, train=False, download=True, transform=transforms)
```

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

In [6]:

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

print(f"Training on {device}")
```

```
Training on cuda
```

In [7]:

```python
NUM_EPOCHS = 300
LEARNING_RATE = 1e-2
BATCH_SIZE = 1024

model = CNN().to(device=device)

loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=LEARNING_RATE)

# Load the data into a dataloaders.
train_loader = torch.utils.data.DataLoader(cirfar10_train,
                                           batch_size=BATCH_SIZE,
                                           shuffle=True,
                                           pin_memory=True,
                                           persistent_workers=True,
                                           num_workers=6)

val_loader = torch.utils.data.DataLoader(cirfar10_val,
                                         batch_size=BATCH_SIZE,
                                         shuffle=False,
                                         pin_memory=True,
                                         persistent_workers=True,
                                         num_workers=2)
```

In [8]:

```python
try:
    # Using time to time the training.
    start_time = time.time()
    training_losses, val_losses, train_accuracies, val_accuracies = training_loop(NUM_EPOC
                                                                                   optimize
                                                                                   model,
                                                                                   loss,
                                                                                   train_lo
                                                                                   val_load
    end_time = time.time()
except Exception as err:
    print(err)
finally:
```

```python
    # Close the threads
    try:
        train_loader._iterator._shutdown_workers()
    except Exception as err:
        print(err)
        print("[Training loader]: Could not shutdown the workers. Might not have spawn yet

    try:
        val_loader._iterator._shutdown_workers()
    except Exception as err:
        print(err)
        print("[Validation loader]: Could not shutdown the workers. Might not have spawn y

# Report the final stats about the training.
print(" ")
print(f"Final Loss: {training_losses[-1]}, Final Training Accuracy: {train_accuracies[-1]
print(f"Final Val Accuracy: {val_accuracies[-1] * 100}%")
print(f"Training Time: {(end_time - start_time):.2f} seconds")
```

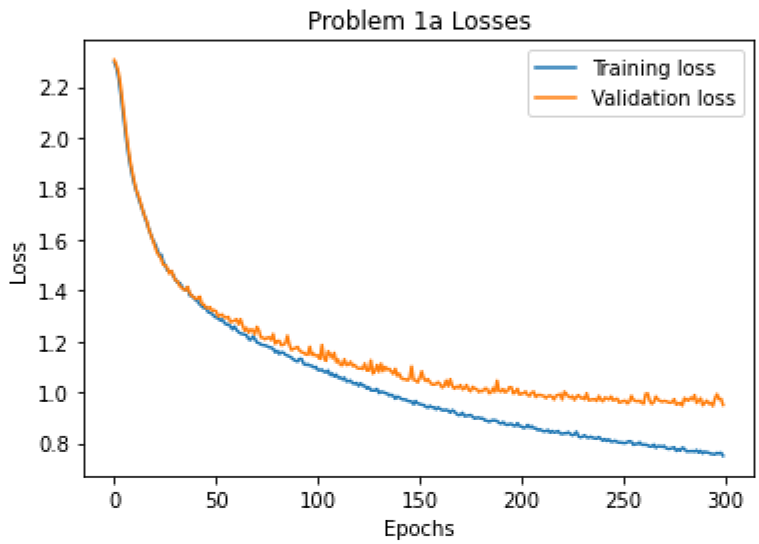
```
Epoch: 1, Training Loss: 2.2955425126211986, Validation Loss: 2.3042561054229735, Train Ac
curacy: 12.952%,
Validation Accuracy: 11.59%
Epoch: 10, Training Loss: 1.8442904462619705, Validation Loss: 1.8580263018608094, Train A
ccuracy: 34.618%,
Validation Accuracy: 34.27%
Epoch: 20, Training Loss: 1.596616151381512, Validation Loss: 1.5941105604171752, Train Ac
curacy: 43.284%,
Validation Accuracy: 43.03%
Epoch: 30, Training Loss: 1.4580550437070885, Validation Loss: 1.4589152812957764, Train A
ccuracy: 47.914%,
Validation Accuracy: 48.04%
Epoch: 40, Training Loss: 1.3700478685145476, Validation Loss: 1.3732717990875245, Train A
ccuracy: 51.06%,
Validation Accuracy: 51.11%
Epoch: 50, Training Loss: 1.299148433062495, Validation Loss: 1.3218871116638184, Train Ac
curacy: 53.93%,
Validation Accuracy: 53.14%
Epoch: 60, Training Loss: 1.2585597962749249, Validation Loss: 1.281577503681183, Train Ac
curacy: 55.47%,
Validation Accuracy: 55.120000000000005%
Epoch: 70, Training Loss: 1.2105555874960763, Validation Loss: 1.227596378326416, Train Ac
curacy: 57.348%,
Validation Accuracy: 56.49999999999999%
Epoch: 80, Training Loss: 1.156944608201786, Validation Loss: 1.1904311299324035, Train Ac
curacy: 59.294000000000004%,
Validation Accuracy: 58.32000000000001%
Epoch: 90, Training Loss: 1.1238568875254418, Validation Loss: 1.1731510996818542, Train A
ccuracy: 60.504000000000005%,
Validation Accuracy: 58.98%
Epoch: 100, Training Loss: 1.0997404079048, Validation Loss: 1.1430832743644714, Train Acc
uracy: 61.466%,
Validation Accuracy: 60.370000000000005%
Epoch: 110, Training Loss: 1.0694901189025567, Validation Loss: 1.1183802962303162, Train
Accuracy: 62.676%,
Validation Accuracy: 61.14000000000001%
Epoch: 120, Training Loss: 1.036156491357453, Validation Loss: 1.0973090887069703, Train A
ccuracy: 63.79%,
Validation Accuracy: 62.23%
Epoch: 130, Training Loss: 1.0129343563196611, Validation Loss: 1.1071391582489014, Train
Accuracy: 64.614%,
Validation Accuracy: 61.970000000000006%
Epoch: 140, Training Loss: 0.983330095300869, Validation Loss: 1.0640487849712372, Train A
ccuracy: 65.816%,
Validation Accuracy: 63.42%
Epoch: 150, Training Loss: 0.9556920929830901, Validation Loss: 1.0387795269489288, Train
```

```
Accuracy: 66.792%,
Validation Accuracy: 64.25999999999999%
Epoch: 160, Training Loss: 0.9400383258352474, Validation Loss: 1.0393112361431123, Train
Accuracy: 67.432%,
Validation Accuracy: 64.44%
Epoch: 170, Training Loss: 0.9099398535125109, Validation Loss: 1.0176909506320952, Train
Accuracy: 68.572%,
Validation Accuracy: 65.14999999999999%
Epoch: 180, Training Loss: 0.9003039209210143, Validation Loss: 1.000786578655243, Train A
ccuracy: 68.818%,
Validation Accuracy: 65.44%
Epoch: 190, Training Loss: 0.8795796292168754, Validation Loss: 1.005182832479477, Train A
ccuracy: 69.422%,
Validation Accuracy: 65.36999999999999%
Epoch: 200, Training Loss: 0.8669570781746689, Validation Loss: 0.9910204708576202, Train
Accuracy: 69.94200000000001%,
Validation Accuracy: 65.93%
Epoch: 210, Training Loss: 0.8542824959268376, Validation Loss: 0.9895675480365753, Train
Accuracy: 70.35%,
Validation Accuracy: 66.10000000000001%
Epoch: 220, Training Loss: 0.8380790121701299, Validation Loss: 0.9922572553157807, Train
Accuracy: 70.954%,
Validation Accuracy: 65.96%
Epoch: 230, Training Loss: 0.8228327194038703, Validation Loss: 0.977374941110611, Train A
ccuracy: 71.63199999999999%,
Validation Accuracy: 66.44%
Epoch: 240, Training Loss: 0.8162860055359042, Validation Loss: 0.971097332239151, Train A
ccuracy: 71.628%,
Validation Accuracy: 67.12%
Epoch: 250, Training Loss: 0.802828561286537, Validation Loss: 0.9624959230422974, Train A
ccuracy: 72.11%,
Validation Accuracy: 67.19000000000001%
Epoch: 260, Training Loss: 0.7943886360343622, Validation Loss: 0.9591034173965454, Train
Accuracy: 72.568%,
Validation Accuracy: 67.32000000000001%
Epoch: 270, Training Loss: 0.7854912475663789, Validation Loss: 0.962449711561203, Train A
ccuracy: 72.83800000000001%,
Validation Accuracy: 67.47%
Epoch: 280, Training Loss: 0.7822473560060773, Validation Loss: 0.9477557480335236, Train
Accuracy: 72.868%,
Validation Accuracy: 67.83%
Epoch: 290, Training Loss: 0.760169694618303, Validation Loss: 0.9485102534294129, Train A
ccuracy: 73.65%,
Validation Accuracy: 67.52%
Epoch: 300, Training Loss: 0.7506052644885316, Validation Loss: 0.9500561714172363, Train
Accuracy: 74.086%,
Validation Accuracy: 68.08%

Final Loss: 0.7506052644885316, Final Training Accuracy: 74.086%,
Final Val Accuracy: 68.08%
Training Time: 919.63 seconds
```

In [9]:

```python
# Plotting the Losses

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
plt.ylabel("Loss")

# Plot the model and the actual values.
plt.plot(training_losses, label='Training loss')
plt.plot(val_losses, label='Validation loss')
plt.legend()
plt.title("Problem 1a Losses")
```

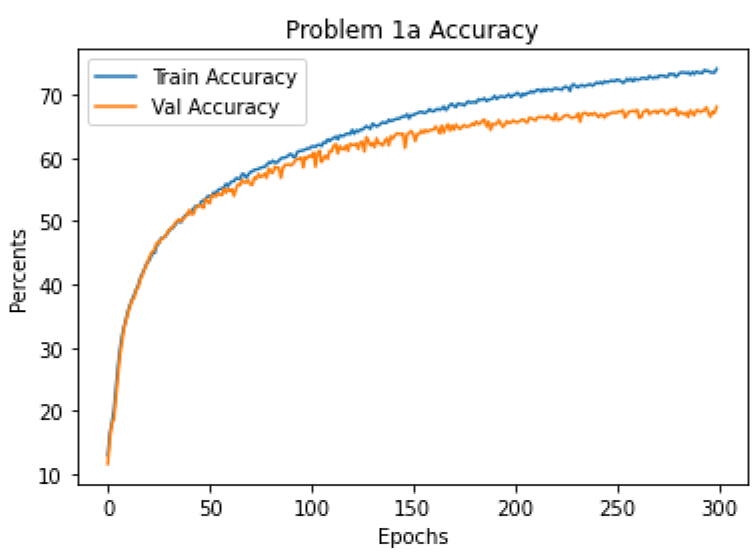
Out[9]: Text(0.5, 1.0, 'Problem 1a Losses')



In [10]:

```
# Plotting the accuracy of the model.

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
plt.ylabel("Percents")

for i, x in enumerate(train_accuracies):
    train_accuracies[i] = x * 100

for i, x in enumerate(val_accuracies):
    val_accuracies[i] = x * 100


plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(val_accuracies, label='Val Accuracy')
plt.legend()
plt.title("Problem 1a Accuracy")
```

Out[10]: Text(0.5, 1.0, 'Problem 1a Accuracy')



Problem 1b

Extend your CNN by adding one more additional convolution layer followed by an activation function and pooling function.

You also need to adjust your fully connected layer properly with respect to intermediate feature dimensions.

Train your network for 300 epochs. Report your training time, loss, and evaluation accuracy after 300 epochs. Analyze your results in your report and compare your model size and accuracy over the baseline implementation in Problem1a. Do you see any over-fitting? Make sure to submit your code by providing the GitHub URL of your course repository for this course.

In [11]:
```
class CNN2(nn.Module):
    def __init__(self, n_channels1 = 32):
        super().__init__()
        self.n_channels1 = n_channels1
        self.conv1 = nn.Conv2d(3, self.n_channels1, kernel_size = 3, padding = 1)
        self.conv2 = nn.Conv2d(n_channels1, (self.n_channels1 // 2), kernel_size = 3, padd
        self.conv3 = nn.Conv2d((self.n_channels1 // 2), (self.n_channels1 // 4), kernel_si
        self.fc1 = nn.Linear(4 * 4 * (self.n_channels1 // 4), 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = f.max_pool2d(torch.relu(self.conv1(x)), 2)
        out = f.max_pool2d(torch.relu(self.conv2(out)), 2)
        out = f.max_pool2d(torch.relu(self.conv3(out)), 2)
        out = out.view(-1, 4 * 4 * (self.n_channels1 // 4))
        out = torch.tanh(self.fc1(out))
        out = self.fc2(out)
        return out
```

In [12]:
```
NUM_EPOCHS = 300
LEARNING_RATE = 1e-2
BATCH_SIZE = 1024

model = CNN2().to(device=device)

loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=LEARNING_RATE)

# Load the data into a dataloaders.
train_loader = torch.utils.data.DataLoader(cirfar10_train,
                                           batch_size=BATCH_SIZE,
                                           shuffle=True,
                                           pin_memory=True,
                                           persistent_workers=True,
                                           num_workers=6)

val_loader = torch.utils.data.DataLoader(cirfar10_val,
                                         batch_size=BATCH_SIZE,
                                         shuffle=False,
                                         pin_memory=True,
                                         persistent_workers=True,
                                         num_workers=2)
```

In [13]:
```
try:
    # Using time to time the training.
    start_time = time.time()
    training_losses, val_losses, train_accuracies, val_accuracies = training_loop(NUM_EPOC
                                                                                 optimize
                                                                                 model,
                                                                                 loss,
                                                                                 train_lo
                                                                                 val_load

    end_time = time.time()
except Exception as err:
    print(err)
finally:
```

```python
    # Close the threads
    try:
        train_loader._iterator._shutdown_workers()
    except Exception as err:
        print(err)
        print("[Training loader]: Could not shutdown the workers. Might not have spawn yet

    try:
        val_loader._iterator._shutdown_workers()
    except Exception as err:
        print(err)
        print("[Validation loader]: Could not shutdown the workers. Might not have spawn y

# Report the final stats about the training.
print(" ")
print(f"Final Loss: {training_losses[-1]}, Final Training Accuracy: {train_accuracies[-1]
print(f"Final Val Accuracy: {val_accuracies[-1] * 100}%")
print(f"Training Time: {(end_time - start_time):.2f} seconds")
```

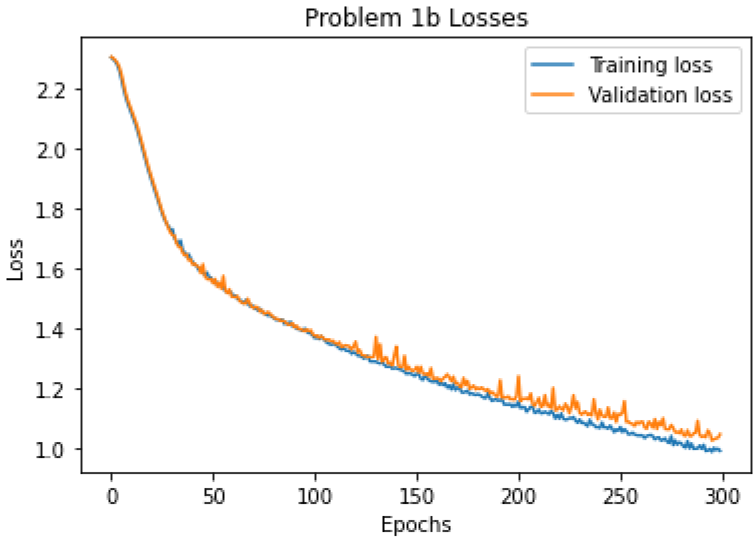
```
Epoch: 1, Training Loss: 2.301606762165926, Validation Loss: 2.30378520488739, Train Accur
acy: 10.158000000000001%,
Validation Accuracy: 9.969999999999999%
Epoch: 10, Training Loss: 2.1319300009279836, Validation Loss: 2.1419822692871096, Train A
ccuracy: 22.142%,
Validation Accuracy: 21.05%
Epoch: 20, Training Loss: 1.9084616777848225, Validation Loss: 1.918773889541626, Train Ac
curacy: 31.8%,
Validation Accuracy: 31.630000000000003%
Epoch: 30, Training Loss: 1.7271657987516753, Validation Loss: 1.7192156076431275, Train A
ccuracy: 37.128%,
Validation Accuracy: 37.269999999999996%
Epoch: 40, Training Loss: 1.6316196894159123, Validation Loss: 1.6280341506004334, Train A
ccuracy: 40.472%,
Validation Accuracy: 40.45%
Epoch: 50, Training Loss: 1.5687541110174996, Validation Loss: 1.5646364450454713, Train A
ccuracy: 42.936%,
Validation Accuracy: 42.809999999999995%
Epoch: 60, Training Loss: 1.5181748137182118, Validation Loss: 1.5059452295303344, Train A
ccuracy: 44.808%,
Validation Accuracy: 44.78%
Epoch: 70, Training Loss: 1.4770508688323352, Validation Loss: 1.4714210629463196, Train A
ccuracy: 46.478%,
Validation Accuracy: 46.21%
Epoch: 80, Training Loss: 1.4374104665250194, Validation Loss: 1.4401942849159242, Train A
ccuracy: 48.098%,
Validation Accuracy: 47.22%
Epoch: 90, Training Loss: 1.406398673446811, Validation Loss: 1.4164637446403503, Train Ac
curacy: 49.328%,
Validation Accuracy: 48.67%
Epoch: 100, Training Loss: 1.3747797328598645, Validation Loss: 1.3795557379722596, Train
Accuracy: 50.51%,
Validation Accuracy: 50.029999999999994%
Epoch: 110, Training Loss: 1.3477652145891774, Validation Loss: 1.3578344702720642, Train
Accuracy: 51.558000000000001%,
Validation Accuracy: 50.519999999999996%
Epoch: 120, Training Loss: 1.3203445016121378, Validation Loss: 1.3394025802612304, Train
Accuracy: 52.769999999999996%,
Validation Accuracy: 51.470000000000006%
Epoch: 130, Training Loss: 1.2892763030772307, Validation Loss: 1.3060084104537963, Train
Accuracy: 53.788000000000004%,
Validation Accuracy: 52.910000000000004%
Epoch: 140, Training Loss: 1.2674669805838137, Validation Loss: 1.31036274433136, Train Ac
curacy: 54.65%,
Validation Accuracy: 52.459999999999994%
Epoch: 150, Training Loss: 1.240694345260153, Validation Loss: 1.2608980536460876, Train A
```

```
ccuracy: 55.788000000000004%,
Validation Accuracy: 54.65%
Epoch: 160, Training Loss: 1.221480172507617, Validation Loss: 1.2328736186027527, Train A
ccuracy: 56.39999999999999%,
Validation Accuracy: 55.989999999999995%
Epoch: 170, Training Loss: 1.1939981640601645, Validation Loss: 1.2161322474479674, Train
Accuracy: 57.408%,
Validation Accuracy: 56.13%
Epoch: 180, Training Loss: 1.1775222457185084, Validation Loss: 1.1958510518074035, Train
Accuracy: 58.10999999999999%,
Validation Accuracy: 57.35%
Epoch: 190, Training Loss: 1.1626266751970564, Validation Loss: 1.1733281493186951, Train
Accuracy: 58.584%,
Validation Accuracy: 58.489999999999995%
Epoch: 200, Training Loss: 1.1394536130282344, Validation Loss: 1.1730970859527587, Train
Accuracy: 59.474000000000004%,
Validation Accuracy: 58.43000000000001%
Epoch: 210, Training Loss: 1.1208114088798056, Validation Loss: 1.1552870988845825, Train
Accuracy: 60.132%,
Validation Accuracy: 59.489999999999995%
Epoch: 220, Training Loss: 1.1112168054191434, Validation Loss: 1.1283663630485534, Train
Accuracy: 60.629999999999995%,
Validation Accuracy: 60.17%
Epoch: 230, Training Loss: 1.0852455168354267, Validation Loss: 1.1539132714271545, Train
Accuracy: 61.45%,
Validation Accuracy: 59.31999999999999%
Epoch: 240, Training Loss: 1.070779029203921, Validation Loss: 1.0957736253738404, Train A
ccuracy: 62.03999999999999%,
Validation Accuracy: 61.61%
Epoch: 250, Training Loss: 1.0585996447777262, Validation Loss: 1.1143147945404053, Train
Accuracy: 62.629999999999995%,
Validation Accuracy: 60.51%
Epoch: 260, Training Loss: 1.042268164303838, Validation Loss: 1.0812086105346679, Train A
ccuracy: 63.214000000000006%,
Validation Accuracy: 61.919999999999995%
Epoch: 270, Training Loss: 1.036059883176064, Validation Loss: 1.087683951854706, Train Ac
curacy: 63.232%,
Validation Accuracy: 61.760000000000005%
Epoch: 280, Training Loss: 1.0194789292861004, Validation Loss: 1.0503705978393554, Train
Accuracy: 64.122%,
Validation Accuracy: 62.949999999999996%
Epoch: 290, Training Loss: 1.0089996019188239, Validation Loss: 1.044353437423706, Train A
ccuracy: 64.338%,
Validation Accuracy: 62.849999999999994%
Epoch: 300, Training Loss: 0.9915532652212649, Validation Loss: 1.0467054963111877, Train
Accuracy: 64.958%,
Validation Accuracy: 63.470000000000006%

Final Loss: 0.9915532652212649, Final Training Accuracy: 64.958%, Final Val Accuracy: 63.4
70000000000006%
Training Time: 892.96 seconds
```

In [14]:

```
# Plotting the Losses

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
plt.ylabel("Loss")

# Plot the model and the actual values.
plt.plot(training_losses, label='Training loss')
plt.plot(val_losses, label='Validation loss')
plt.legend()
plt.title("Problem 1b Losses")
```

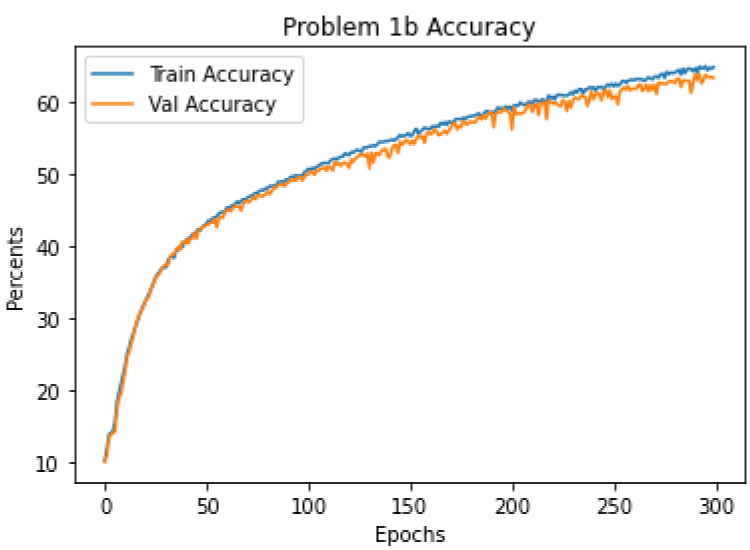
Out[14]: Text(0.5, 1.0, 'Problem 1b Losses')



In [15]:

```
# Plotting the accuracy of the model.

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
plt.ylabel("Percents")

for i, x in enumerate(train_accuracies):
    train_accuracies[i] = x * 100

for i, x in enumerate(val_accuracies):
    val_accuracies[i] = x * 100


plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(val_accuracies, label='Val Accuracy')
plt.legend()
plt.title("Problem 1b Accuracy")
```

Out[15]: Text(0.5, 1.0, 'Problem 1b Accuracy')



In [16]:

```
from ptflops import get_model_complexity_info

model_p1a = CNN()
model_p1b = CNN2()
```

```
macs, params = get_model_complexity_info(model_p1a, (3, 32, 32), as_strings=True,
 print_per_layer_stat=False, verbose=False)
# print out the Model size.
print("Problem 1 Part a")
print("Model size: " + params)

print("")

macs, params = get_model_complexity_info(model_p1b, (3, 32, 32), as_strings=True,
 print_per_layer_stat=False, verbose=False)
# print out the Model size.
print("Problem 1 Part b")
print("Model size: " + params)
```

```
Problem 1 Part a
Model size: 38.65 k

Problem 1 Part b
Model size: 11.14 k
```

Problem 2a

Build a ResNet based Convolutional Neural Network, like what we built in lectures (with skip connections), to classify the images across all 10 classes in CIFAR 10. For this problem, let's use 10 blocks for ResNet and call it ResNet-10.
Use the similar dimensions and channels as we need in lectures.

Train your network for 300 epochs. Report your training time, training loss, and evaluation accuracy after 300 epochs.

Analyze your results in your report and compare them against problem 1.b on training time, achieved accuracy, and model size. Make sure to submit your code by providing the GitHub URL of your course repository for this course.

In [17]:

```
class ResBlock(nn.Module):
    def __init__(self, n_channels1):
        super(ResBlock, self).__init__()
        self.conv = nn.Conv2d(n_channels1, n_channels1, kernel_size = 3, padding = 1)

    def forward(self, x):
        out = self.conv(x)
        out = torch.relu(out)
        return out + x
```

In [18]:

```
class ResNet(nn.Module):
    def __init__(self, n_channels1, n_blocks):
        super().__init__()
        self.n_channels1 = n_channels1
        self.conv1 = nn.Conv2d(3, self.n_channels1, kernel_size = 3, padding = 1)
        self.blocks = nn.Sequential(*(n_blocks * [ResBlock(n_channels1=self.n_channels1)]))
        self.fc1 = nn.Linear(8 * 8 * self.n_channels1, 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = f.max_pool2d(torch.relu(self.conv1(x)), 2)
        out = f.max_pool2d(self.blocks(out), 2)
        out = out.view(-1, 8 * 8 * self.n_channels1)
        out = torch.relu(self.fc1(out))
        out = self.fc2(out)
        return out
```

In [19]:
```
NUM_EPOCHS = 300
LEARNING_RATE = 3e-3
BATCH_SIZE = 1024
NUM_CHANNELS = 32
BLOCKS = 10

model = ResNet(NUM_CHANNELS, BLOCKS).to(device=device)

loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=LEARNING_RATE)

# Load the data into a dataloaders.
train_loader = torch.utils.data.DataLoader(cirfar10_train,
                                           batch_size=BATCH_SIZE,
                                           shuffle=True,
                                           pin_memory=True,
                                           persistent_workers=True,
                                           num_workers=6)

val_loader = torch.utils.data.DataLoader(cirfar10_val,
                                         batch_size=BATCH_SIZE,
                                         shuffle=False,
                                         pin_memory=True,
                                         persistent_workers=True,
                                         num_workers=2)
```

In [20]:
```
try:
    # Using time to time the training.
    start_time = time.time()
    training_losses, val_losses, train_accuracies, val_accuracies = training_loop(NUM_EPOC
                                                                                  optimize
                                                                                  model,
                                                                                  loss,
                                                                                  train_lo
                                                                                  val_loac
    end_time = time.time()
except Exception as err:
    print(err)
finally:
    # Close the threads
    try:
        train_loader._iterator._shutdown_workers()
    except Exception as err:
        print(err)
        print("[Training loader]: Could not shutdown the workers. Might not have spawn yet

    try:
        val_loader._iterator._shutdown_workers()
    except Exception as err:
        print(err)
        print("[Validation loader]: Could not shutdown the workers. Might not have spawn y

# Report the final stats about the training.
print(" ")
print(f"Final Loss: {training_losses[-1]}, Final Training Accuracy: {train_accuracies[-1]
print(f"Final Val Accuracy: {val_accuracies[-1] * 100}%")
print(f"Training Time: {(end_time - start_time):.2f} seconds")
```

```
Epoch: 1, Training Loss: 2.2091207601586165, Validation Loss: 3.409150791168213, Train Acc
uracy: 19.464000000000002%,
Validation Accuracy: 9.87%
Epoch: 10, Training Loss: 1.5232609534750179, Validation Loss: 1.6260629892349243, Train A
ccuracy: 45.564%,
Validation Accuracy: 41.24%
```

```
Epoch: 20, Training Loss: 1.3039579488793198, Validation Loss: 1.3965609073638916, Train A
ccuracy: 53.434000000000005%,
Validation Accuracy: 49.7%
Epoch: 30, Training Loss: 1.1880691197453712, Validation Loss: 1.1879597783088685, Train A
ccuracy: 57.888%,
Validation Accuracy: 57.65%
Epoch: 40, Training Loss: 1.1038207010347016, Validation Loss: 1.1301270127296448, Train A
ccuracy: 60.866%,
Validation Accuracy: 59.91%
Epoch: 50, Training Loss: 1.0277478548945214, Validation Loss: 1.0989624857902527, Train A
ccuracy: 63.944%,
Validation Accuracy: 61.07%
Epoch: 60, Training Loss: 0.9741090134698518, Validation Loss: 1.0422486066818237, Train A
ccuracy: 65.666%,
Validation Accuracy: 63.32%
Epoch: 70, Training Loss: 0.925658178572752, Validation Loss: 1.041062694787979, Train Acc
uracy: 67.514%,
Validation Accuracy: 63.18%
Epoch: 80, Training Loss: 0.8733080491727713, Validation Loss: 1.0066020011901855, Train A
ccuracy: 69.174%,
Validation Accuracy: 64.55%
Epoch: 90, Training Loss: 0.8297736170340557, Validation Loss: 1.0580503582954406, Train A
ccuracy: 70.99%,
Validation Accuracy: 63.859999999999999%
Epoch: 100, Training Loss: 0.8050880578099465, Validation Loss: 0.9575101196765899, Train
Accuracy: 71.592%,
Validation Accuracy: 67.22%
Epoch: 110, Training Loss: 0.7680785376198438, Validation Loss: 0.9932644784450531, Train
Accuracy: 73.074%,
Validation Accuracy: 66.18%
Epoch: 120, Training Loss: 0.7362398541703516, Validation Loss: 0.9578771591186523, Train
Accuracy: 74.068%,
Validation Accuracy: 67.58%
Epoch: 130, Training Loss: 0.7008889864902107, Validation Loss: 1.0174954891204835, Train
Accuracy: 75.342%,
Validation Accuracy: 66.02%
Epoch: 140, Training Loss: 0.6841176286035654, Validation Loss: 0.9404460966587067, Train
Accuracy: 76.102%,
Validation Accuracy: 68.46%
Epoch: 150, Training Loss: 0.6641073141779218, Validation Loss: 0.9659663140773773, Train
Accuracy: 76.86%,
Validation Accuracy: 68.13%
Epoch: 160, Training Loss: 0.6420031615665981, Validation Loss: 1.172751522064209, Train A
ccuracy: 77.446%,
Validation Accuracy: 64.09%
Epoch: 170, Training Loss: 0.5981552783323794, Validation Loss: 1.0293115496635437, Train
Accuracy: 78.85799999999999%,
Validation Accuracy: 67.11%
Epoch: 180, Training Loss: 0.5930914270634554, Validation Loss: 0.9609642207622529, Train
Accuracy: 79.292%,
Validation Accuracy: 68.92%
Epoch: 190, Training Loss: 0.5648371346142828, Validation Loss: 1.0092516183853149, Train
Accuracy: 80.316%,
Validation Accuracy: 67.46%
Epoch: 200, Training Loss: 0.5589074717492474, Validation Loss: 1.0281811714172364, Train
Accuracy: 80.348%,
Validation Accuracy: 67.28%
Epoch: 210, Training Loss: 0.5365018893261345, Validation Loss: 1.0564041078090667, Train
Accuracy: 81.098%,
Validation Accuracy: 66.71000000000001%
Epoch: 220, Training Loss: 0.5275622515045867, Validation Loss: 1.1001995086669922, Train
Accuracy: 81.382%,
Validation Accuracy: 66.53%
Epoch: 230, Training Loss: 0.5058481012071881, Validation Loss: 1.1075478792190552, Train
Accuracy: 82.08800000000001%,
Validation Accuracy: 67.30000000000001%
```

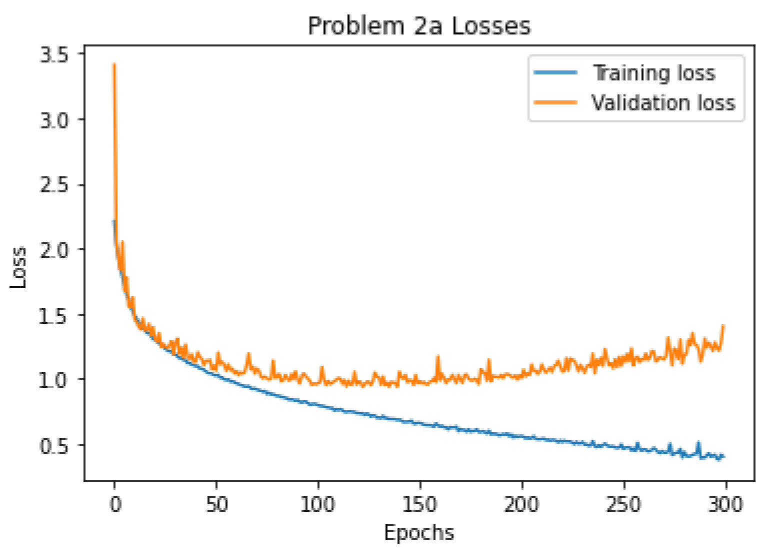
```
Epoch: 240, Training Loss: 0.47735033959758527, Validation Loss: 1.1601692676544189, Train
Accuracy: 83.06%,
Validation Accuracy: 65.32%
Epoch: 250, Training Loss: 0.49791465730083234, Validation Loss: 1.099630731344223, Train
Accuracy: 82.46%,
Validation Accuracy: 68.04%
Epoch: 260, Training Loss: 0.44949058610565806, Validation Loss: 1.1515549540519714, Train
Accuracy: 84.11999999999999%,
Validation Accuracy: 67.14%
Epoch: 270, Training Loss: 0.4470905661582947, Validation Loss: 1.1578999638557435, Train
Accuracy: 84.172%,
Validation Accuracy: 67.80000000000001%
Epoch: 280, Training Loss: 0.39618328943544506, Validation Loss: 1.1172569274902344, Train
Accuracy: 85.99799999999999%,
Validation Accuracy: 68.11%
Epoch: 290, Training Loss: 0.3969786282704801, Validation Loss: 1.1859701156616211, Train
Accuracy: 86.034%,
Validation Accuracy: 67.96%
Epoch: 300, Training Loss: 0.4046458425570507, Validation Loss: 1.4038283228874207, Train
Accuracy: 85.54%,
Validation Accuracy: 64.35%

Final Loss: 0.4046458425570507, Final Training Accuracy: 85.54%, Final Val Accuracy: 64.3
5%
Training Time: 1331.39 seconds
```

In [21]:

```python
# Plotting the Losses

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
plt.ylabel("Loss")

# Plot the model and the actual values.
plt.plot(training_losses, label='Training loss')
plt.plot(val_losses, label='Validation loss')
plt.legend()
plt.title("Problem 2a Losses")
```

Out[21]:

```
Text(0.5, 1.0, 'Problem 2a Losses')
```



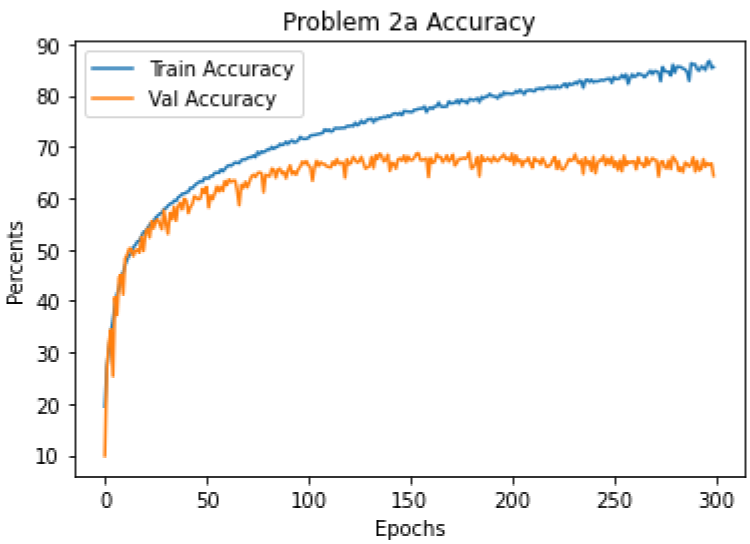
In [22]:

```python
# Plotting the accuracy of the model.

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
```

```
plt.ylabel("Percents")

for i, x in enumerate(train_accuracies):
    train_accuracies[i] = x * 100

for i, x in enumerate(val_accuracies):
    val_accuracies[i] = x * 100


plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(val_accuracies, label='Val Accuracy')
plt.legend()
plt.title("Problem 2a Accuracy")
```

Out[22]:

```
Text(0.5, 1.0, 'Problem 2a Accuracy')
```



Problem 2b

Develop three additional trainings and evaluations for your ResNet-10 to assess the impacts of regularization to your ResNet-10.

- Weight Decay with lambda of 0.001
- Dropout with p=0.3
- Batch Normalization

Report and compare your training time, training loss, and evaluation accuracy after 300 epochs across these three different trainings. Analyze your results in your report and compare them against problem 1.a on training time, achieved accuracy.

In [23]:

```
class ResBlockDropout(nn.Module):
    def __init__(self, n_channels1, p):
        super(ResBlockDropout, self).__init__()
        self.conv = nn.Conv2d(n_channels1, n_channels1, kernel_size = 3, padding = 1)
        self.dropout = nn.Dropout2d(p = p)

    def forward(self, x):
        out = self.conv(x)
        out = self.dropout(out)
        out = torch.relu(out)
        return out + x

class ResNetDropout(nn.Module):
    def __init__(self, n_channels1, n_blocks, p):
        super().__init__()
        self.n_channels1 = n_channels1
```

```
        self.conv1 = nn.Conv2d(3, self.n_channels1, kernel_size = 3, padding = 1)
        self.blocks = nn.Sequential(*(n_blocks * [ResBlockDropout(n_channels1 = self.n_cha
        self.dropout = nn.Dropout2d(p = p)
        self.fc1 = nn.Linear(8 * 8 * self.n_channels1, 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = f.max_pool2d(torch.relu(self.dropout(self.conv1(x))), 2)
        out = f.max_pool2d(self.blocks(out), 2)
        out = out.view(-1, 8 * 8 * self.n_channels1)
        out = torch.relu(self.fc1(out))
        out = self.fc2(out)
        return out
```

In [24]:

```
class ResBlockBatchNorm(nn.Module):
    def __init__(self, n_channels1):
        super(ResBlockBatchNorm, self).__init__()
        self.conv = nn.Conv2d(n_channels1, n_channels1, kernel_size = 3, padding = 1, bias
        self.batch_norm = nn.BatchNorm2d(num_features = n_channels1)
        torch.nn.init.kaiming_normal_(self.conv.weight, nonlinearity = 'relu')
        torch.nn.init.constant_(self.batch_norm.weight, 0.5)
        torch.nn.init.zeros_(self.batch_norm.bias)

    def forward(self, x):
        out = self.conv(x)
        out = torch.relu(out)
        out = self.batch_norm(out)
        return out + x


class ResNetBatchNorm(nn.Module):
    def __init__(self, n_channels1, n_blocks):
        super().__init__()
        self.n_channels1 = n_channels1
        self.conv1 = nn.Conv2d(3, self.n_channels1, kernel_size = 3, padding = 1)
        self.batchnorm1 = nn.BatchNorm2d(num_features = n_channels1)
        self.blocks = nn.Sequential(*(n_blocks * [ResBlockBatchNorm(n_channels1=self.n_cha
        self.fc1 = nn.Linear(8 * 8 * self.n_channels1, 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = f.max_pool2d(self.batchnorm1(torch.relu(self.conv1(x))), 2)
        out = f.max_pool2d(self.blocks(out), 2)
        out = out.view(-1, 8 * 8 * self.n_channels1)
        out = torch.relu(self.fc1(out))
        out = self.fc2(out)
        return out
```

In [25]:

```
def training_loopl2(epochs, optimizer, model, loss_fn, train_loader, val_loader, l2_lambda
    training_losses = []
    val_losses = []
    train_accuracies = []
    val_accuracies = []
    for epoch in range(1, epochs + 1):
        # Temp vars for use in finding the accuracy.
        val_correct_labels = 0
        val_count = 0
        loss_val_value = 0

        #Set the model to inference mode
        model.eval()
        with torch.no_grad():
            for imgs, labels in val_loader:
```

```python
                # Move the data to correct device
                imgs = imgs.to(device=device)
                labels = labels.to(device=device)

                # Pass imgs through the model and find the loss.
                output = model(imgs)
                loss_val = loss_fn(output, labels)
                loss_val_value += float(loss_val)

                # Find the accurcey of the model.
                _, predicted = torch.max(output, dim=1)
                val_count += labels.shape[0]
                val_correct_labels += int((predicted == labels).sum())

            # Store the loss and accuracy.
            loss_val_value /= len(val_loader)
            val_losses.append(loss_val_value)
            val_accuracies.append(val_correct_labels/val_count)

        #Set the model to training mode.
        model.train()
        train_correct_labels = 0
        train_count = 0
        loss_train_value = 0
        for imgs, labels in train_loader:
            # Move the data to correct device
            imgs = imgs.to(device=device)
            labels = labels.to(device=device)

            # Pass imgs through the model and find the loss.
            output = model(imgs)
            loss_train = loss_fn(output, labels)

            # Perform L2 regularization
            l2_norm = sum(p.pow(2.0).sum() for p in model.parameters())
            loss_train = loss_train + l2_lambda * l2_norm
            loss_train_value += float(loss_train)

            # Adject the params
            optimizer.zero_grad()
            loss_train.backward()
            optimizer.step()

            # Find the accurcey of the model.
            _, predicted = torch.max(output, dim=1)
            train_count += labels.shape[0]
            train_correct_labels += int((predicted == labels).sum())

        # Store the loss
        loss_train_value /= len(train_loader)
        training_losses.append(loss_train_value)
        train_accuracies.append(train_correct_labels/train_count)

        # Print out the loss every 10 epoch
        if epoch % 10 == 0 or epoch == 1:
            print(f"Epoch: {epoch}, Training Loss: {loss_train_value}", end="")
            print(f", Validation Loss: {loss_val_value}, Train Accuracy: {(train_correct_
            print(f"Validation Accuracy: {(val_correct_labels/val_count)*100}%")

    return training_losses, val_losses, train_accuracies, val_accuracies
```

L2 regularization

In [26]:

```python
NUM_EPOCHS = 300
```

```python
LEARNING_RATE = 3e-3
BATCH_SIZE = 1024
NUM_CHANNELS = 32
BLOCKS = 10
LAMBDA = 0.001

model = ResNet(NUM_CHANNELS, BLOCKS).to(device=device)

loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=LEARNING_RATE)

# Load the data into a dataloaders.
train_loader = torch.utils.data.DataLoader(cirfar10_train,
                                           batch_size=BATCH_SIZE,
                                           shuffle=True,
                                           pin_memory=True,
                                           persistent_workers=True,
                                           num_workers=6)

val_loader = torch.utils.data.DataLoader(cirfar10_val,
                                         batch_size=BATCH_SIZE,
                                         shuffle=False,
                                         pin_memory=True,
                                         persistent_workers=True,
                                         num_workers=2)
```

In [27]:

```python
try:
    # Using time to time the training.
    start_time = time.time()
    training_losses, val_losses, train_accuracies, val_accuracies = training_loopl2(NUM_EI
                                                                                loss, train_loac
    end_time = time.time()
except Exception as err:
    print(err)
finally:
    # Close the threads
    try:
        train_loader._iterator._shutdown_workers()
    except Exception as err:
        print(err)
        print("[Training loader]: Could not shutdown the workers. Might not have spawn yet

    try:
        val_loader._iterator._shutdown_workers()
    except Exception as err:
        print(err)
        print("[Validation loader]: Could not shutdown the workers. Might not have spawn y

# Report the final stats about the training.
print(" ")
print(f"Final Loss: {training_losses[-1]}, Final Training Accuracy: {train_accuracies[-1]
print(f"Final Val Accuracy: {val_accuracies[-1] * 100}%")
print(f"Training Time: {(end_time - start_time):.2f} seconds")
```

```
Epoch: 1, Training Loss: 2.3142274545163524, Validation Loss: 4.1238309144973755, Train Ac
curacy: 13.919999999999998%,
Validation Accuracy: 9.85%
Epoch: 10, Training Loss: 1.575825394416342, Validation Loss: 1.6705918431282043, Train Ac
curacy: 44.882%,
Validation Accuracy: 41.8%
Epoch: 20, Training Loss: 1.3500861221430254, Validation Loss: 1.3114459753036498, Train A
ccuracy: 53.33%,
Validation Accuracy: 53.63%
Epoch: 30, Training Loss: 1.2328011794966094, Validation Loss: 1.263883078098297, Train Ac
```

```
curacy: 57.838%,
Validation Accuracy: 55.110000000000001%
Epoch: 40, Training Loss: 1.143428019114903, Validation Loss: 1.1672658324241638, Train Ac
curacy: 60.870000000000005%,
Validation Accuracy: 58.34%
Epoch: 50, Training Loss: 1.0749626183996395, Validation Loss: 1.1074361205101013, Train A
ccuracy: 63.528%,
Validation Accuracy: 60.9%
Epoch: 60, Training Loss: 1.0209610316218163, Validation Loss: 1.0571848034858704, Train A
ccuracy: 65.346%,
Validation Accuracy: 62.61%
Epoch: 70, Training Loss: 0.9682030166898455, Validation Loss: 1.064701998233795, Train Ac
curacy: 67.45400000000001%,
Validation Accuracy: 62.45%
Epoch: 80, Training Loss: 0.9278382756272141, Validation Loss: 0.9997530221939087, Train A
ccuracy: 68.94%,
Validation Accuracy: 64.67%
Epoch: 90, Training Loss: 0.8847817717766275, Validation Loss: 1.0017371594905853, Train A
ccuracy: 70.256%,
Validation Accuracy: 64.7%
Epoch: 100, Training Loss: 0.8583354305247871, Validation Loss: 1.023037165403366, Train A
ccuracy: 71.344%,
Validation Accuracy: 64.56%
Epoch: 110, Training Loss: 0.8333709507572408, Validation Loss: 0.9612559080123901, Train
Accuracy: 72.304%,
Validation Accuracy: 66.57%
Epoch: 120, Training Loss: 0.7956217393583181, Validation Loss: 0.936178320646286, Train A
ccuracy: 73.556%,
Validation Accuracy: 67.57%
Epoch: 130, Training Loss: 0.7784959917165795, Validation Loss: 0.9468644380569458, Train
Accuracy: 74.144%,
Validation Accuracy: 67.78999999999999%
Epoch: 140, Training Loss: 0.7455706669359791, Validation Loss: 1.0789949893951416, Train
Accuracy: 75.30799999999999%,
Validation Accuracy: 64.16%
Epoch: 150, Training Loss: 0.7303874249361, Validation Loss: 0.9646796762943268, Train Acc
uracy: 75.94800000000001%,
Validation Accuracy: 67.32000000000001%
Epoch: 160, Training Loss: 0.7034203264178062, Validation Loss: 0.9589688122272492, Train
Accuracy: 77.006%,
Validation Accuracy: 67.86%
Epoch: 170, Training Loss: 0.6908224468328514, Validation Loss: 0.940428364276886, Train A
ccuracy: 77.24600000000001%,
Validation Accuracy: 68.78999999999999%
Epoch: 180, Training Loss: 0.6585673568200092, Validation Loss: 1.0318988144397736, Train
Accuracy: 78.598%,
Validation Accuracy: 66.59%
Epoch: 190, Training Loss: 0.6455984954931298, Validation Loss: 1.076586651802063, Train A
ccuracy: 78.988%,
Validation Accuracy: 65.31%
Epoch: 200, Training Loss: 0.6235856596304445, Validation Loss: 1.082159197330475, Train A
ccuracy: 79.586%,
Validation Accuracy: 65.31%
Epoch: 210, Training Loss: 0.6012028662525878, Validation Loss: 1.1216039180755615, Train
Accuracy: 80.318%,
Validation Accuracy: 64.87%
Epoch: 220, Training Loss: 0.5846772029691812, Validation Loss: 1.02495020031929, Train Ac
curacy: 80.952%,
Validation Accuracy: 67.52%
Epoch: 230, Training Loss: 0.5780847638237233, Validation Loss: 1.0342690408229829, Train
Accuracy: 81.478%,
Validation Accuracy: 67.61%
Epoch: 240, Training Loss: 0.5573226505396317, Validation Loss: 1.0671584606170654, Train
Accuracy: 81.972%,
Validation Accuracy: 67.11%
Epoch: 250, Training Loss: 0.5171754281131589, Validation Loss: 1.023002988100052, Train A
```

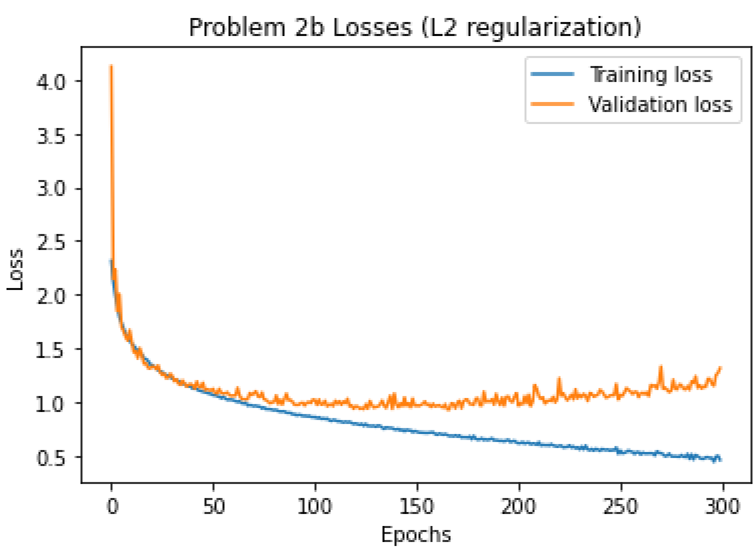
```
ccuracy: 83.74000000000001%,
Validation Accuracy: 68.08%
Epoch: 260, Training Loss: 0.533476430542615, Validation Loss: 1.0454317808151246, Train A
ccuracy: 82.966%,
Validation Accuracy: 68.52000000000001%
Epoch: 270, Training Loss: 0.5258905692976348, Validation Loss: 1.1837811112403869, Train
Accuracy: 83.314%,
Validation Accuracy: 66.60000000000001%
Epoch: 280, Training Loss: 0.4889177211693355, Validation Loss: 1.163470995426178, Train A
ccuracy: 84.52%,
Validation Accuracy: 66.53999999999999%
Epoch: 290, Training Loss: 0.4787518388154555, Validation Loss: 1.1583162307739259, Train
Accuracy: 85.026%,
Validation Accuracy: 66.95%
Epoch: 300, Training Loss: 0.46078920546843083, Validation Loss: 1.3157394528388977, Train
Accuracy: 85.502%,
Validation Accuracy: 64.12%

Final Loss: 0.46078920546843083, Final Training Accuracy: 85.502%, Final Val Accuracy: 64.
12%
Training Time: 1394.76 seconds
```

In [28]:

```python
# Plotting the Losses

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
plt.ylabel("Loss")

# Plot the model and the actual values.
plt.plot(training_losses, label='Training loss')
plt.plot(val_losses, label='Validation loss')
plt.legend()
plt.title("Problem 2b Losses (L2 regularization)")
```

Out[28]:

```
Text(0.5, 1.0, 'Problem 2b Losses (L2 regularization)')
```



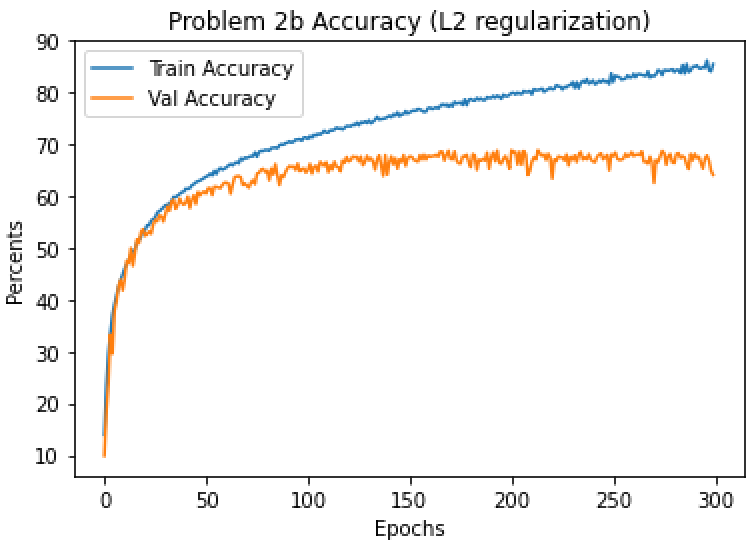
In [29]:

```python
# Plotting the accuracy of the model.

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
plt.ylabel("Percents")

for i, x in enumerate(train_accuracies):
    train_accuracies[i] = x * 100
```

```python
for i, x in enumerate(val_accuracies):
    val_accuracies[i] = x * 100


plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(val_accuracies, label='Val Accuracy')
plt.legend()
plt.title("Problem 2b Accuracy (L2 regularization)")
```

Out[29]:

```
Text(0.5, 1.0, 'Problem 2b Accuracy (L2 regularization)')
```



Dropout

In [30]:

```python
NUM_EPOCHS = 300
LEARNING_RATE = 3e-3
BATCH_SIZE = 1024
NUM_CHANNELS = 32
BLOCKS = 10
DROPOUT_RATE = 0.3

model = ResNetDropout(NUM_CHANNELS, BLOCKS, DROPOUT_RATE).to(device=device)

loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=LEARNING_RATE)

# Load the data into a dataloaders.
train_loader = torch.utils.data.DataLoader(cirfar10_train,
                                           batch_size=BATCH_SIZE,
                                           shuffle=True,
                                           pin_memory=True,
                                           persistent_workers=True,
                                           num_workers=6)

val_loader = torch.utils.data.DataLoader(cirfar10_val,
                                         batch_size=BATCH_SIZE,
                                         shuffle=False,
                                         pin_memory=True,
                                         persistent_workers=True,
                                         num_workers=2)
```

In [31]:

```python
try:
    # Using time to time the training.
    start_time = time.time()
    training_losses, val_losses, train_accuracies, val_accuracies = training_loop(NUM_EPOC
```

```
                                                                                               optimize
                                                                                               model,
                                                                                               loss,
                                                                                               train_lo
                                                                                               val_load
    end_time = time.time()
except Exception as err:
    print(err)
finally:
    # Close the threads
    try:
        train_loader._iterator._shutdown_workers()
    except Exception as err:
        print(err)
        print("[Training loader]: Could not shutdown the workers. Might not have spawn yet

    try:
        val_loader._iterator._shutdown_workers()
    except Exception as err:
        print(err)
        print("[Validation loader]: Could not shutdown the workers. Might not have spawn y

# Report the final stats about the training.
print(" ")
print(f"Final Loss: {training_losses[-1]}, Final Training Accuracy: {train_accuracies[-1]
print(f"Final Val Accuracy: {val_accuracies[-1] * 100}%")
print(f"Training Time: {(end_time - start_time):.2f} seconds")
```

```
Epoch: 1, Training Loss: 2.367990965745887, Validation Loss: 4.822448825836181, Train Accu
racy: 13.522%,
Validation Accuracy: 9.91%
Epoch: 10, Training Loss: 1.7582410938885746, Validation Loss: 1.70624897480011, Train Acc
uracy: 38.318000000000005%,
Validation Accuracy: 40.65%
Epoch: 20, Training Loss: 1.5501156996707528, Validation Loss: 1.4821787476539612, Train A
ccuracy: 45.245999999999995%,
Validation Accuracy: 48.18%
Epoch: 30, Training Loss: 1.4407451906982733, Validation Loss: 1.3800381898880005, Train A
ccuracy: 49.403999999999996%,
Validation Accuracy: 51.33%
Epoch: 40, Training Loss: 1.3726884905172854, Validation Loss: 1.3182076454162597, Train A
ccuracy: 51.803999999999995%,
Validation Accuracy: 53.49%
Epoch: 50, Training Loss: 1.312596819838699, Validation Loss: 1.2776867508888246, Train Ac
curacy: 53.99%,
Validation Accuracy: 54.879999999999995%
Epoch: 60, Training Loss: 1.2667997005034466, Validation Loss: 1.2258456230163575, Train A
ccuracy: 55.65%,
Validation Accuracy: 56.69%
Epoch: 70, Training Loss: 1.2298871376076523, Validation Loss: 1.1914692521095276, Train A
ccuracy: 57.06%,
Validation Accuracy: 57.85%
Epoch: 80, Training Loss: 1.1932431556740586, Validation Loss: 1.168636178970337, Train Ac
curacy: 58.211999999999996%,
Validation Accuracy: 59.019999999999996%
Epoch: 90, Training Loss: 1.15898504305859, Validation Loss: 1.1378766655921937, Train Acc
uracy: 59.67400000000001%,
Validation Accuracy: 60.07%
Epoch: 100, Training Loss: 1.132236833475074, Validation Loss: 1.129099977016449, Train Ac
curacy: 60.626000000000005%,
Validation Accuracy: 60.199999999999996%
Epoch: 110, Training Loss: 1.1028731185562757, Validation Loss: 1.0999318957328796, Train
Accuracy: 61.417999999999999%,
Validation Accuracy: 61.61%
Epoch: 120, Training Loss: 1.0765865335659104, Validation Loss: 1.0925630331039429, Train
```

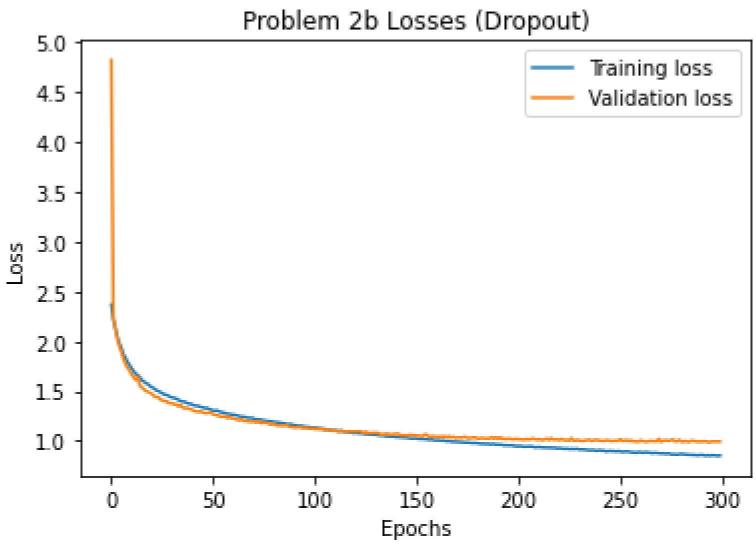
```
Accuracy: 62.342%,
Validation Accuracy: 61.5%
Epoch: 130, Training Loss: 1.060942534281283, Validation Loss: 1.0816362380981446, Train A
ccuracy: 62.978%,
Validation Accuracy: 61.91%
Epoch: 140, Training Loss: 1.0430015228232559, Validation Loss: 1.0669136047363281, Train
Accuracy: 63.452%,
Validation Accuracy: 62.11%
Epoch: 150, Training Loss: 1.0274162365465749, Validation Loss: 1.0559451222419738, Train
Accuracy: 63.968%,
Validation Accuracy: 62.849999999999994%
Epoch: 160, Training Loss: 1.0048301317253892, Validation Loss: 1.0397932827472687, Train
Accuracy: 64.932%,
Validation Accuracy: 63.480000000000004%
Epoch: 170, Training Loss: 0.990006832443938, Validation Loss: 1.0275482296943665, Train A
ccuracy: 65.606%,
Validation Accuracy: 63.77%
Epoch: 180, Training Loss: 0.9793227278456396, Validation Loss: 1.0322331726551055, Train
Accuracy: 65.852%,
Validation Accuracy: 63.59%
Epoch: 190, Training Loss: 0.9603690541520411, Validation Loss: 1.0353974938392638, Train
Accuracy: 66.378%,
Validation Accuracy: 63.56%
Epoch: 200, Training Loss: 0.9484936680112567, Validation Loss: 1.021116954088211, Train A
ccuracy: 66.768%,
Validation Accuracy: 64.11%
Epoch: 210, Training Loss: 0.9347131471244656, Validation Loss: 1.0102991163730621, Train
Accuracy: 67.23400000000001%,
Validation Accuracy: 64.4%
Epoch: 220, Training Loss: 0.9229391424023375, Validation Loss: 1.0135547995567322, Train
Accuracy: 67.684%,
Validation Accuracy: 64.49000000000001%
Epoch: 230, Training Loss: 0.9202546963886339, Validation Loss: 1.0101924300193788, Train
Accuracy: 67.91199999999999%,
Validation Accuracy: 64.31%
Epoch: 240, Training Loss: 0.9021264521443114, Validation Loss: 1.007382720708847, Train A
ccuracy: 68.61%,
Validation Accuracy: 64.79%
Epoch: 250, Training Loss: 0.8976141068400169, Validation Loss: 0.9992087543010711, Train
Accuracy: 68.77%,
Validation Accuracy: 64.85%
Epoch: 260, Training Loss: 0.8869724662936463, Validation Loss: 0.9912411510944367, Train
Accuracy: 69.144%,
Validation Accuracy: 65.28%
Epoch: 270, Training Loss: 0.8743213694922778, Validation Loss: 1.02087544798851, Train Ac
curacy: 69.57799999999999%,
Validation Accuracy: 64.53%
Epoch: 280, Training Loss: 0.8664016151914791, Validation Loss: 1.009973120689392, Train A
ccuracy: 69.702%,
Validation Accuracy: 64.85%
Epoch: 290, Training Loss: 0.8575294978764593, Validation Loss: 0.9940951764583588, Train
Accuracy: 70.216%,
Validation Accuracy: 65.38000000000001%
Epoch: 300, Training Loss: 0.8524240194534769, Validation Loss: 0.9911846339702606, Train
Accuracy: 70.37%,
Validation Accuracy: 65.82000000000001%

Final Loss: 0.8524240194534769, Final Training Accuracy: 70.37%, Final Val Accuracy: 65.82
000000000001%
Training Time: 1071.67 seconds
```

In [32]:

```
# Plotting the Losses

fig = plt.figure()
# Name the x and y axis
```

```
plt.xlabel("Epochs")
plt.ylabel("Loss")

# Plot the model and the actual values.
plt.plot(training_losses, label='Training loss')
plt.plot(val_losses, label='Validation loss')
plt.legend()
plt.title("Problem 2b Losses (Dropout)")
```

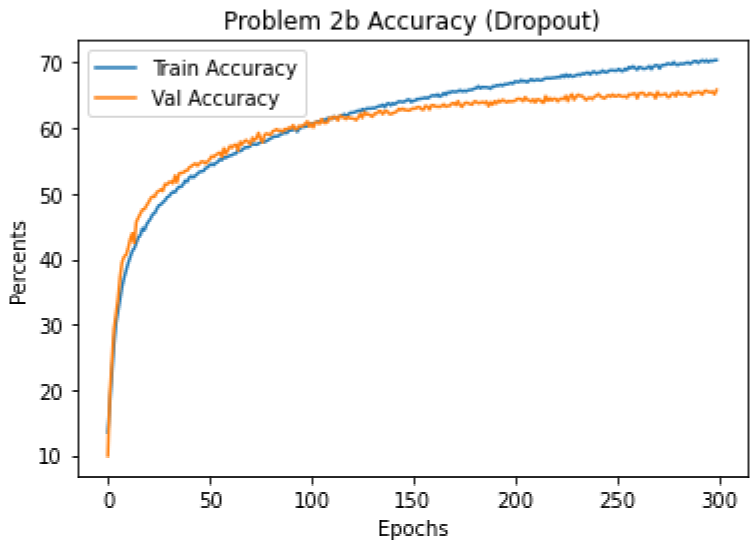
Out[32]: Text(0.5, 1.0, 'Problem 2b Losses (Dropout)')



In [33]:

```
# Plotting the accuracy of the model.

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
plt.ylabel("Percents")

for i, x in enumerate(train_accuracies):
    train_accuracies[i] = x * 100

for i, x in enumerate(val_accuracies):
    val_accuracies[i] = x * 100


plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(val_accuracies, label='Val Accuracy')
plt.legend()
plt.title("Problem 2b Accuracy (Dropout)")
```

Out[33]: Text(0.5, 1.0, 'Problem 2b Accuracy (Dropout)')



Batch Normalization

In [39]:

```
NUM_EPOCHS = 300
LEARNING_RATE = 3e-3
BATCH_SIZE = 1024
NUM_CHANNELS = 32
BLOCKS = 10

model = ResNetBatchNorm(NUM_CHANNELS, BLOCKS).to(device=device)

loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=LEARNING_RATE)

# Load the data into a dataloaders.
train_loader = torch.utils.data.DataLoader(cirfar10_train,
                                           batch_size=BATCH_SIZE,
                                           shuffle=True,
                                           pin_memory=True,
                                           persistent_workers=True,
                                           num_workers=6)

val_loader = torch.utils.data.DataLoader(cirfar10_val,
                                         batch_size=BATCH_SIZE,
                                         shuffle=False,
                                         pin_memory=True,
                                         persistent_workers=True,
                                         num_workers=2)
```

In [40]:

```
try:
    # Using time to time the training.
    start_time = time.time()
    training_losses, val_losses, train_accuracies, val_accuracies = training_loop(NUM_EPOC
                                                                                  optimize
                                                                                  model,
                                                                                  loss,
                                                                                  train_lo
                                                                                  val_load
    end_time = time.time()
except Exception as err:
    print(err)
finally:
    # Close the threads
    try:
        train_loader._iterator._shutdown_workers()
    except Exception as err:
```

```
        print(err)
        print("[Training loader]: Could not shutdown the workers. Might not have spawn yet

    try:
        val_loader._iterator._shutdown_workers()
    except Exception as err:
        print(err)
        print("[Validation loader]: Could not shutdown the workers. Might not have spawn y

# Report the final stats about the training.
print(" ")
print(f"Final Loss: {training_losses[-1]}, Final Training Accuracy: {train_accuracies[-1]
print(f"Final Val Accuracy: {val_accuracies[-1] * 100}%")
print(f"Training Time: {(end_time - start_time):.2f} seconds")
```

```
Epoch: 1, Training Loss: 2.134947188046514, Validation Loss: 6.530372667312622, Train Accu
racy: 23.188%,
Validation Accuracy: 9.030000000000001%
Epoch: 10, Training Loss: 1.2864108912798824, Validation Loss: 1.5457259178161622, Train A
ccuracy: 54.172%,
Validation Accuracy: 44.81%
Epoch: 20, Training Loss: 1.08544721287124, Validation Loss: 1.4156354308128356, Train Acc
uracy: 61.756%,
Validation Accuracy: 49.78%
Epoch: 30, Training Loss: 0.971721483736622, Validation Loss: 1.343389105796814, Train Acc
uracy: 66.12599999999999%,
Validation Accuracy: 52.28%
Epoch: 40, Training Loss: 0.8942666820117405, Validation Loss: 1.2909228205680847, Train A
ccuracy: 68.958%,
Validation Accuracy: 54.6%
Epoch: 50, Training Loss: 0.8348093774853921, Validation Loss: 1.2875849723815918, Train A
ccuracy: 71.084%,
Validation Accuracy: 54.620000000000005%
Epoch: 60, Training Loss: 0.7852898629344239, Validation Loss: 1.2532994747161865, Train A
ccuracy: 72.95%,
Validation Accuracy: 55.589999999999996%
Epoch: 70, Training Loss: 0.7439052280114622, Validation Loss: 1.2634387493133545, Train A
ccuracy: 74.39%,
Validation Accuracy: 55.08%
Epoch: 80, Training Loss: 0.7049869894981384, Validation Loss: 1.2581091284751893, Train A
ccuracy: 75.878%,
Validation Accuracy: 55.489999999999995%
Epoch: 90, Training Loss: 0.6733198019922996, Validation Loss: 1.2819584727287292, Train A
ccuracy: 77.00399999999999%,
Validation Accuracy: 54.37%
Epoch: 100, Training Loss: 0.6414083011296331, Validation Loss: 1.2699986219406127, Train
Accuracy: 78.14%,
Validation Accuracy: 54.96%
Epoch: 110, Training Loss: 0.6174328217701036, Validation Loss: 1.2938064575195312, Train
Accuracy: 79.018%,
Validation Accuracy: 54.43%
Epoch: 120, Training Loss: 0.5867059632223479, Validation Loss: 1.3400663495063783, Train
Accuracy: 80.258%,
Validation Accuracy: 53.010000000000005%
Epoch: 130, Training Loss: 0.5620757992170295, Validation Loss: 1.3434906244277953, Train
Accuracy: 81.03200000000001%,
Validation Accuracy: 53.23%
Epoch: 140, Training Loss: 0.5340441799893672, Validation Loss: 1.334370422363281З, Train
Accuracy: 81.988%,
Validation Accuracy: 53.89000000000001%
Epoch: 150, Training Loss: 0.5114255517112966, Validation Loss: 1.3825472593307495, Train
Accuracy: 82.98%,
Validation Accuracy: 52.59%
Epoch: 160, Training Loss: 0.48648300158734226, Validation Loss: 1.434782087802887, Train
Accuracy: 83.76599999999999%,
```

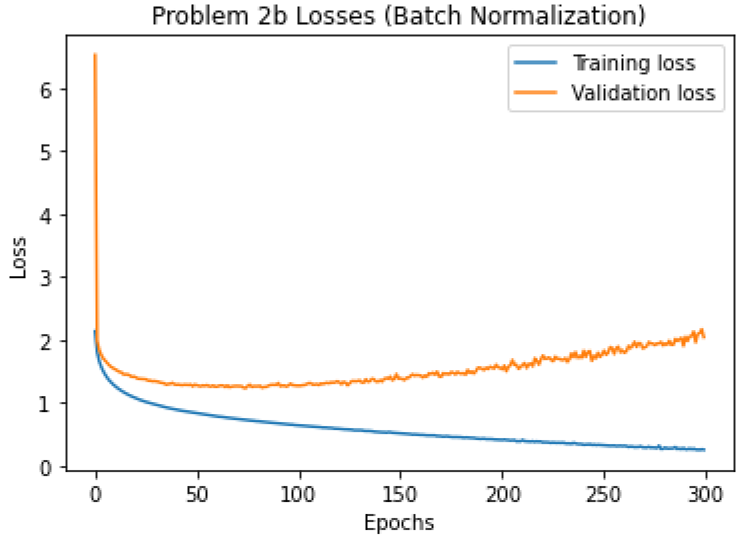
```
Validation Accuracy: 51.38%
Epoch: 170, Training Loss: 0.4695691435920949, Validation Loss: 1.4183242678642274, Train
Accuracy: 84.332%,
Validation Accuracy: 52.23%
Epoch: 180, Training Loss: 0.45027968591573286, Validation Loss: 1.4675779461860656, Train
Accuracy: 85.04599999999999%,
Validation Accuracy: 51.160000000000004%
Epoch: 190, Training Loss: 0.42316525627155693, Validation Loss: 1.5538283824920653, Train
Accuracy: 86.08200000000001%,
Validation Accuracy: 49.36%
Epoch: 200, Training Loss: 0.40919872935937374, Validation Loss: 1.5522674679756165, Train
Accuracy: 86.53999999999999%,
Validation Accuracy: 49.75%
Epoch: 210, Training Loss: 0.384913474929576, Validation Loss: 1.5654204487800598, Train A
ccuracy: 87.488%,
Validation Accuracy: 49.96%
Epoch: 220, Training Loss: 0.38104309354509625, Validation Loss: 1.6905890941619872, Train
Accuracy: 87.464%,
Validation Accuracy: 47.23%
Epoch: 230, Training Loss: 0.3586162900438114, Validation Loss: 1.6812225699424743, Train
Accuracy: 88.446%,
Validation Accuracy: 47.949999999999996%
Epoch: 240, Training Loss: 0.34025941150529043, Validation Loss: 1.7125178694725036, Train
Accuracy: 89.144%,
Validation Accuracy: 47.760000000000005%
Epoch: 250, Training Loss: 0.3203951649519862, Validation Loss: 1.7660947442054749, Train
Accuracy: 89.67399999999999%,
Validation Accuracy: 47.620000000000005%
Epoch: 260, Training Loss: 0.3095585916723524, Validation Loss: 1.8329430937767028, Train
Accuracy: 90.08800000000001%,
Validation Accuracy: 46.650000000000006%
Epoch: 270, Training Loss: 0.2880407568751549, Validation Loss: 1.863990604877472, Train A
ccuracy: 91.014%,
Validation Accuracy: 46.489999999999995%
Epoch: 280, Training Loss: 0.2655640280976587, Validation Loss: 1.9084888696670532, Train
Accuracy: 92.078%,
Validation Accuracy: 46.29%
Epoch: 290, Training Loss: 0.277166161001945, Validation Loss: 1.901686668395996, Train Ac
curacy: 91.036%,
Validation Accuracy: 47.28%
Epoch: 300, Training Loss: 0.24973155406056619, Validation Loss: 2.0405576109886168, Train
Accuracy: 92.374%,
Validation Accuracy: 45.61%

Final Loss: 0.24973155406056619, Final Training Accuracy: 92.374%, Final Val Accuracy: 45.
61%
Training Time: 1085.71 seconds
```

In [41]:

```python
# Plotting the Losses

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
plt.ylabel("Loss")

# Plot the model and the actual values.
plt.plot(training_losses, label='Training loss')
plt.plot(val_losses, label='Validation loss')
plt.legend()
plt.title("Problem 2b Losses (Batch Normalization)")
```

Out[41]:

```
Text(0.5, 1.0, 'Problem 2b Losses (Batch Normalization)')
```

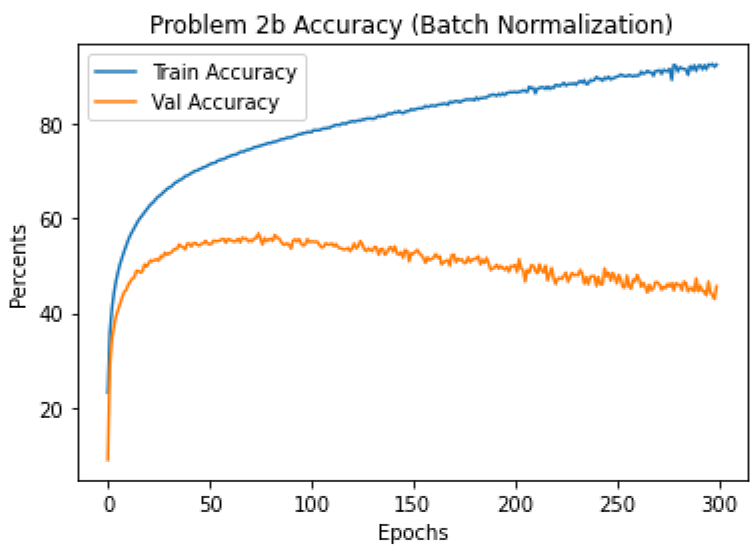


In [42]:

```
# Plotting the accuracy of the model.

fig = plt.figure()
# Name the x and y axis
plt.xlabel("Epochs")
plt.ylabel("Percents")

for i, x in enumerate(train_accuracies):
    train_accuracies[i] = x * 100

for i, x in enumerate(val_accuracies):
    val_accuracies[i] = x * 100


plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(val_accuracies, label='Val Accuracy')
plt.legend()
plt.title("Problem 2b Accuracy (Batch Normalization)")
```

Out[42]:

```
Text(0.5, 1.0, 'Problem 2b Accuracy (Batch Normalization)')
```



In [43]:

```
model_l2 = ResNet(32, 10)
model_dropout = ResNetDropout(32, 10, 0.3)
model_batchnorm = ResNetBatchNorm(32, 10)

macs, params = get_model_complexity_info(model_l2, (3, 32, 32), as_strings=True,
 print_per_layer_stat=False, verbose=False)
```

```python
# print out the Model size.
print("L2 regularization")
print("Model size: " + params)

print("")

macs, params = get_model_complexity_info(model_dropout, (3, 32, 32), as_strings=True,
 print_per_layer_stat=False, verbose=False)
# print out the Model size.
print("Dropout")
print("Model size: " + params)

print("")

macs, params = get_model_complexity_info(model_batchnorm, (3, 32, 32), as_strings=True,
 print_per_layer_stat=False, verbose=False)
# print out the Model size.
print("Batch Normalization")
print("Model size: " + params)
```

```
L2 regularization
Model size: 76.04 k

Dropout
Model size: 76.04 k

Batch Normalization
Model size: 76.14 k
```

In [ ]:

```python

```