```
import numpy as np
import pandas as pd

# Data Visualisation
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.python.client import device_lib

import torch
import torch.optim as optim
import glob
from torchvision import transforms
import torch.nn as nn
import datetime
import torch.nn.functional as F
```

Model for Problem 1

```
class myNet(nn.Module):
  def __init__(self):
    super().__init__()
    self.conv1 = nn.Conv2d(3, 16, kernel_size = 3, padding = 1)
    self.conv2 = nn.Conv2d(16, 8, kernel_size = 3, padding = 1)
    self.fc1 = nn.Linear(8 * 8 * 8, 32)
    self.fc2 = nn.Linear(32, 10)

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

```
from torchvision import datasets
data_path = 'Documents/School/ECGR 4106/dlwpt-code-master/p1ch7'
cifar10 = datasets.CIFAR10(data_path, train=True, download=True, transform = transforms.ToTen
cifar10_val = datasets.CIFAR10(data_path, train=False, download=True, transform = transforms.
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to Documents/School/
                                                                    170499072/? [00:03<00:00, 53144570.98it/s]
Extracting Documents/School/ECGR 4106/dlwpt-code-master/p1ch7/cifar-10-python.tar.gz to
Files already downloaded and verified
```

```
device = (torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu'))
print(f"Training on device {device}.")
```

```
 Training on device cuda.
```

```
def training_loop(n_epochs, optimizer, model, loss_fn, train_loader):
  for epoch in range(1, n_epochs + 1):
    loss_train = 0.0

    for imgs, labels in train_loader:
      imgs = imgs.to(device = device)
      labels = labels.to(device = device)
      outputs = model(imgs)
      loss = loss_fn(outputs, labels)

      optimizer.zero_grad()
      loss.backward()
      optimizer.step()

      loss_train += loss.item()

    if epoch == 1 or epoch % 10 == 0:
      print('{} Epoch {}, Training loss {}'.format(
          datetime.datetime.now(), epoch, loss_train / len(train_loader)))


model = myNet().to(device = device)
optimizer = optim.SGD(model.parameters(), lr = 1e-2)
loss_fn = nn.CrossEntropyLoss()
train_loader = torch.utils.data.DataLoader(cifar10, batch_size = 64, shuffle=True)


training_loop(
    n_epochs = 300,
    optimizer = optimizer,
    model = model,
    loss_fn = loss_fn,
    train_loader = train_loader
)
```

```
 2022-03-31 04:03:17.101799 Epoch 1, Training loss 2.267829884958389
 2022-03-31 04:04:14.484891 Epoch 10, Training loss 1.4064956697661553
 2022-03-31 04:05:18.085778 Epoch 20, Training loss 1.1904813609922025
 2022-03-31 04:06:21.677290 Epoch 30, Training loss 1.064244159042378
 2022-03-31 04:07:24.994342 Epoch 40, Training loss 0.9842125989896867
 2022-03-31 04:08:28.285315 Epoch 50, Training loss 0.9256816506385803
 2022-03-31 04:09:31.635879 Epoch 60, Training loss 0.8820523621557314
 2022-03-31 04:10:34.911315 Epoch 70, Training loss 0.8478115840107584
 2022-03-31 04:11:38.399911 Epoch 80, Training loss 0.8215839589572014
 2022-03-31 04:12:41.679888 Epoch 90, Training loss 0.7996352706342706
 2022-03-31 04:13:44.946630 Epoch 100, Training loss 0.7801369659964691
 2022-03-31 04:14:48.331511 Epoch 110, Training loss 0.7617996210027533
 2022-03-31 04:15:51.591693 Epoch 120, Training loss 0.7459920627610458
```

```
2022-03-31 04:16:54.803146 Epoch 130, Training loss 0.7304977042138424
2022-03-31 04:17:58.324111 Epoch 140, Training loss 0.7178703875416685
2022-03-31 04:19:01.318308 Epoch 150, Training loss 0.7059906645656546
2022-03-31 04:20:04.443644 Epoch 160, Training loss 0.6934841080852177
2022-03-31 04:21:07.727349 Epoch 170, Training loss 0.6825801922232294
2022-03-31 04:22:10.948053 Epoch 180, Training loss 0.6714670094077849
2022-03-31 04:23:14.010460 Epoch 190, Training loss 0.6653707325839631
2022-03-31 04:24:17.214334 Epoch 200, Training loss 0.6548315021769165
2022-03-31 04:25:20.374394 Epoch 210, Training loss 0.646265203804921
2022-03-31 04:26:23.493899 Epoch 220, Training loss 0.6390698861587992
2022-03-31 04:27:26.719583 Epoch 230, Training loss 0.6330312879570305
2022-03-31 04:28:30.031462 Epoch 240, Training loss 0.6269845250241287
2022-03-31 04:29:33.049596 Epoch 250, Training loss 0.6203348417873578
2022-03-31 04:30:36.384986 Epoch 260, Training loss 0.6138431365837527
2022-03-31 04:31:39.762946 Epoch 270, Training loss 0.6063849520119254
2022-03-31 04:32:43.076550 Epoch 280, Training loss 0.6025554522528977
2022-03-31 04:33:46.312269 Epoch 290, Training loss 0.5961232577900752
2022-03-31 04:34:49.693370 Epoch 300, Training loss 0.5921098483950281
```

```
def validate(model, train_loader, val_loader):
  for name, loader in [("train", train_loader), ("val", val_loader)]:
    correct = 0
    total = 0

    with torch.no_grad():
      for imgs, labels in loader:
        imgs = imgs.to(device = device)
        labels = labels.to(device = device)
        outputs = model(imgs)
        _, predicted = torch.max(outputs, dim = 1)
        total += labels.shape[0]
        correct += int((predicted == labels).sum())

  print("Accuracy: {}: {:.2f}".format(name, correct / total))


val_loader = torch.utils.data.DataLoader(cifar10_val, batch_size = 64, shuffle = False)


validate(model, train_loader, val_loader)
```

```
Accuracy: val: 0.59
```

```
class myNet2(nn.Module):
  def __init__(self):
    super().__init__()
    self.conv1 = nn.Conv2d(3, 16, kernel_size = 3, padding = 1)
    self.conv2 = nn.Conv2d(16, 8, kernel_size = 3, padding = 1)
    self.conv3 = nn.Conv2d(8, 4, kernel_size = 3, padding = 1)
    self.fc1 = nn.Linear(4 * 4 * 4, 32)
    self.fc2 = nn.Linear(32, 16)
    self.fc3 = nn.Linear(16, 10)
```

```
  def forward(self, x):
      out = F.max_pool2d(torch.tanh(self.conv1(x)), 2)
      out = F.max_pool2d(torch.tanh(self.conv2(out)), 2)
      out = F.max_pool2d(torch.tanh(self.conv3(out)), 2)
      out = out.view(-1, 4 * 4 * 4)
      out = torch.tanh(self.fc1(out))
      out = torch.tanh(self.fc2(out))
      out = self.fc3(out)
      return out


model2 = myNet2().to(device = device)
optimizer2 = optim.SGD(model2.parameters(), lr = 1e-2)


training_loop(
    n_epochs = 300,
    optimizer = optimizer2,
    model = model2,
    loss_fn = loss_fn,
    train_loader = train_loader
)

     2022-03-31 04:35:03.113621 Epoch 1, Training loss 2.3059863919187387
     2022-03-31 04:36:03.661367 Epoch 10, Training loss 1.7103222492710708
     2022-03-31 04:37:10.581736 Epoch 20, Training loss 1.4185363420135224
     2022-03-31 04:38:17.426355 Epoch 30, Training loss 1.2769807779118227
     2022-03-31 04:39:24.402457 Epoch 40, Training loss 1.200122705124833
     2022-03-31 04:40:31.657922 Epoch 50, Training loss 1.149207271502146
     2022-03-31 04:41:39.023311 Epoch 60, Training loss 1.1085100907194034
     2022-03-31 04:42:46.424742 Epoch 70, Training loss 1.0789862668422787
     2022-03-31 04:43:53.573456 Epoch 80, Training loss 1.051820840372149
     2022-03-31 04:45:01.065694 Epoch 90, Training loss 1.032105919192819
     2022-03-31 04:46:08.476028 Epoch 100, Training loss 1.0153484369635277
     2022-03-31 04:47:15.350908 Epoch 110, Training loss 0.9969144527564573
     2022-03-31 04:48:22.170661 Epoch 120, Training loss 0.9872342742922361
     2022-03-31 04:49:29.022638 Epoch 130, Training loss 0.973143782914447
     2022-03-31 04:50:35.696354 Epoch 140, Training loss 0.9608631995709046
     2022-03-31 04:51:42.354447 Epoch 150, Training loss 0.9525602748022055
     2022-03-31 04:52:49.125404 Epoch 160, Training loss 0.9427786179058387
     2022-03-31 04:53:55.976135 Epoch 170, Training loss 0.9317223448735064
     2022-03-31 04:55:02.992112 Epoch 180, Training loss 0.9247731465817718
     2022-03-31 04:56:09.792377 Epoch 190, Training loss 0.9161280933624644
     2022-03-31 04:57:16.733321 Epoch 200, Training loss 0.911653736210845
     2022-03-31 04:58:23.635629 Epoch 210, Training loss 0.9066353816815349
     2022-03-31 04:59:30.509593 Epoch 220, Training loss 0.8995564425235514
     2022-03-31 05:00:37.230278 Epoch 230, Training loss 0.8945068633160018
     2022-03-31 05:01:43.647080 Epoch 240, Training loss 0.8879972444775769
     2022-03-31 05:02:50.265221 Epoch 250, Training loss 0.8861739274943271
     2022-03-31 05:03:56.737356 Epoch 260, Training loss 0.8845572597382928
     2022-03-31 05:05:03.356754 Epoch 270, Training loss 0.8795914948748811
     2022-03-31 05:06:10.188197 Epoch 280, Training loss 0.8767940401268737
     2022-03-31 05:07:16.739233 Epoch 290, Training loss 0.8723064699517492
     2022-03-31 05:08:23.398534 Epoch 300, Training loss 0.8708225484089474
```

```
validate(model2, train_loader, val_loader)
```

```
     Accuracy: val: 0.57
```

```
class ResBlock(nn.Module):
  def __init__(self, n_chans):
    super(ResBlock, self).__init__()
    self.conv = nn.Conv2d(n_chans, n_chans, kernel_size=3, padding=1, bias=False)
    self.batch_norm = nn.BatchNorm2d(num_features=n_chans)
    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 = self.batch_norm(out)
    out = torch.relu(out)
    return out + x


class ResNet10(nn.Module):
  def __init__(self, n_chans1 = 32, n_blocks = 10):
    super().__init__()
    self.n_chans1 = n_chans1
    self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=2, padding=1)
    self.resblocks = nn.Sequential(
        *(n_blocks * [ResBlock(n_chans = n_chans1)]))
    self.fc1 = nn.Linear(8 * 8 * n_chans1, 32)
    self.fc2 = nn.Linear(32, 10)

  def forward(self, x):
    out = F.max_pool2d(torch.relu(self.conv1(x)), 2)
    out = self.resblocks(out)
    out = F.max_pool2d(out, 2)
    out = out.view(-1, 8 * 8 * self.n_chans1)
    out = torch.relu(self.fc1(out))
    out = self.fc2(out)
    return out


model3 = ResNet10(n_chans1 = 32, n_blocks = 10).to(device = device)
optimizer3 = optim.SGD(model3.parameters(), lr = 1e-2)


training_loop(
    n_epochs = 200,
    optimizer = optimizer3,
    model = model3,
    loss_fn = loss_fn,
```

```
    train_loader = train_loader
)
```

```
2022-03-31 06:47:12.603875 Epoch 1, Training loss 2.303305993604538
2022-03-31 06:48:45.533886 Epoch 10, Training loss 1.2049077692848946
2022-03-31 06:50:28.817255 Epoch 20, Training loss 0.8708393223145429
2022-03-31 06:52:12.010369 Epoch 30, Training loss 0.7135152227204778
2022-03-31 06:53:55.236678 Epoch 40, Training loss 0.6158737372559355
2022-03-31 06:55:38.660663 Epoch 50, Training loss 0.5422364560234577
2022-03-31 06:57:22.085627 Epoch 60, Training loss 0.4836357145586892
2022-03-31 06:59:05.805631 Epoch 70, Training loss 0.4381855211847121
2022-03-31 07:00:49.260279 Epoch 80, Training loss 0.40597007111134126
2022-03-31 07:02:33.944333 Epoch 90, Training loss 0.370671501321256
2022-03-31 07:04:17.626909 Epoch 100, Training loss 0.34514773167345836
2022-03-31 07:06:01.043898 Epoch 110, Training loss 0.31759900256724616
2022-03-31 07:07:44.341636 Epoch 120, Training loss 0.30769834620759007
2022-03-31 07:09:27.680422 Epoch 130, Training loss 0.28341449275517555
2022-03-31 07:11:10.870347 Epoch 140, Training loss 0.27426179337894063
2022-03-31 07:12:53.961959 Epoch 150, Training loss 0.26649320131296395
2022-03-31 07:14:36.887662 Epoch 160, Training loss 0.24591345908334644
2022-03-31 07:16:19.916398 Epoch 170, Training loss 0.2393130737182963
2022-03-31 07:18:02.876060 Epoch 180, Training loss 0.22356515135282598
2022-03-31 07:19:46.226755 Epoch 190, Training loss 0.22998335175787854
2022-03-31 07:21:29.573322 Epoch 200, Training loss 0.21546402045042085
```

```
validate(model3, train_loader, val_loader)
```

```
Accuracy: val: 0.64
```

```
def training_loop_l2reg(n_epochs, optimizer, model, loss_fn,
                        train_loader):
    for epoch in range(1, n_epochs + 1):
        loss_train = 0.0
        for imgs, labels in train_loader:
            imgs = imgs.to(device=device)
            labels = labels.to(device=device)
            outputs = model(imgs)
            loss = loss_fn(outputs, labels)

            l2_lambda = 0.001
            l2_norm = sum(p.pow(2.0).sum()
                          for p in model.parameters())  # <1>
            loss = loss + l2_lambda * l2_norm

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            loss_train += loss.item()
        if epoch == 1 or epoch % 10 == 0:
            print('{} Epoch {}, Training loss {}'.format(
                datetime.datetime.now(), epoch,
                loss_train / len(train_loader)))
```

```
training_loop_l2reg(
    n_epochs = 200,
    optimizer = optimizer3,
    model = model3,
    loss_fn = loss_fn,
    train_loader = train_loader
)
```

```
 2022-03-31 07:21:48.877424 Epoch 1, Training loss 1.0049705963457942
 2022-03-31 07:23:27.447765 Epoch 10, Training loss 0.8480407270933966
 2022-03-31 07:25:17.220620 Epoch 20, Training loss 0.7629982006671788
 2022-03-31 07:27:07.184156 Epoch 30, Training loss 0.6913026767923399
 2022-03-31 07:28:56.955728 Epoch 40, Training loss 0.6568689047146941
 2022-03-31 07:30:46.536811 Epoch 50, Training loss 0.6442430227842477
 2022-03-31 07:32:36.416738 Epoch 60, Training loss 0.6303892508339699
 2022-03-31 07:34:26.273324 Epoch 70, Training loss 0.6245178457568673
 2022-03-31 07:36:15.932038 Epoch 80, Training loss 0.6115621050910267
 2022-03-31 07:38:05.570570 Epoch 90, Training loss 0.6092279070935895
 2022-03-31 07:39:55.375205 Epoch 100, Training loss 0.6067226152971882
 2022-03-31 07:41:45.036945 Epoch 110, Training loss 0.5972748949094806
 2022-03-31 07:43:34.707090 Epoch 120, Training loss 0.5980867899744712
 2022-03-31 07:45:24.190253 Epoch 130, Training loss 0.5988763271237884
 2022-03-31 07:47:13.665074 Epoch 140, Training loss 0.5932472881377505
 2022-03-31 07:49:03.218054 Epoch 150, Training loss 0.577561403326976
 2022-03-31 07:50:52.550493 Epoch 160, Training loss 0.5761745732534876
 2022-03-31 07:52:41.635965 Epoch 170, Training loss 0.576780896548115
 2022-03-31 07:54:31.524253 Epoch 180, Training loss 0.5619603552857934
 2022-03-31 07:56:20.865522 Epoch 190, Training loss 0.5685471096993102
 2022-03-31 07:58:09.983655 Epoch 200, Training loss 0.5643743825766742
```

```
validate(model3, train_loader, val_loader)
```

```
 Accuracy: val: 0.62
```

```
class NetDropout(nn.Module):
    def __init__(self, n_chans1=32):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.conv1_dropout = nn.Dropout2d(p=0.3)
        self.conv2 = nn.Conv2d(n_chans1, n_chans1 // 2, kernel_size=3,
                               padding=1)
        self.conv2_dropout = nn.Dropout2d(p=0.3)
        self.fc1 = nn.Linear(8 * 8 * n_chans1 // 2, 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = F.max_pool2d(torch.tanh(self.conv1(x)), 2)
        out = self.conv1_dropout(out)
        out = F.max_pool2d(torch.tanh(self.conv2(out)), 2)
```

```
        out = self.conv2_dropout(out)
        out = out.view(-1, 8 * 8 * self.n_chans1 // 2)
        out = torch.tanh(self.fc1(out))
        out = self.fc2(out)
        return out

model4 = NetDropout(n_chans1 = 32).to(device = device)
optimizer4 = optim.SGD(model4.parameters(), lr = 1e-2)


training_loop(
    n_epochs = 200,
    optimizer = optimizer4,
    model = model4,
    loss_fn = loss_fn,
    train_loader = train_loader
)
```

```
 2022-03-31 15:44:09.160170 Epoch 1, Training loss 2.1980395323175297
 2022-03-31 15:45:06.244259 Epoch 10, Training loss 1.494749722121012
 2022-03-31 15:46:09.789932 Epoch 20, Training loss 1.3196218495478715
 2022-03-31 15:47:13.489134 Epoch 30, Training loss 1.2320125763830931
 2022-03-31 15:48:17.012613 Epoch 40, Training loss 1.170518615209233
 2022-03-31 15:49:20.885848 Epoch 50, Training loss 1.122844330413872
 2022-03-31 15:50:26.215798 Epoch 60, Training loss 1.0925521270545853
 2022-03-31 15:51:30.430899 Epoch 70, Training loss 1.0583589296511677
 2022-03-31 15:52:34.028897 Epoch 80, Training loss 1.0419079329046752
 2022-03-31 15:53:37.547485 Epoch 90, Training loss 1.0209649172432893
 2022-03-31 15:54:41.063210 Epoch 100, Training loss 1.0068524559135632
 2022-03-31 15:55:44.357262 Epoch 110, Training loss 0.9994589087298459
 2022-03-31 15:56:47.773032 Epoch 120, Training loss 0.9808656894185049
 2022-03-31 15:57:51.179362 Epoch 130, Training loss 0.9716645501306295
 2022-03-31 15:58:54.586126 Epoch 140, Training loss 0.9675744561588063
 2022-03-31 15:59:57.877818 Epoch 150, Training loss 0.9605066857069654
 2022-03-31 16:01:01.229915 Epoch 160, Training loss 0.9497633012359404
 2022-03-31 16:02:04.689083 Epoch 170, Training loss 0.9410074310534445
 2022-03-31 16:03:08.047365 Epoch 180, Training loss 0.9412702220632597
 2022-03-31 16:04:11.351590 Epoch 190, Training loss 0.933208909981391
 2022-03-31 16:05:14.541080 Epoch 200, Training loss 0.9278141051302176
```

```
validate(model4, train_loader, val_loader)
```

```
 Accuracy: val: 0.61
```

```
class NetBatchNorm(nn.Module):
    def __init__(self, n_chans1=32):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.conv1_batchnorm = nn.BatchNorm2d(num_features=n_chans1)
        self.conv2 = nn.Conv2d(n_chans1, n_chans1 // 2, kernel_size=3,
                               padding=1)
        self.conv2_batchnorm = nn.BatchNorm2d(num_features=n_chans1 // 2)
```

```
        self.conv2_batchnorm = nn.BatchNorm2d(num_features=n_chans1 // 2)
        self.fc1 = nn.Linear(8 * 8 * n_chans1 // 2, 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = self.conv1_batchnorm(self.conv1(x))
        out = F.max_pool2d(torch.tanh(out), 2)
        out = self.conv2_batchnorm(self.conv2(out))
        out = F.max_pool2d(torch.tanh(out), 2)
        out = out.view(-1, 8 * 8 * self.n_chans1 // 2)
        out = torch.tanh(self.fc1(out))
        out = self.fc2(out)
        return out


model5 = NetBatchNorm(n_chans1 = 32).to(device = device)
optimizer5 = optim.SGD(model5.parameters(), lr = 1e-2)


training_loop(
    n_epochs = 200,
    optimizer = optimizer5,
    model = model5,
    loss_fn = loss_fn,
    train_loader = train_loader
)
```

```
 2022-03-31 16:09:47.967963 Epoch 1, Training loss 1.7838067279752259
 2022-03-31 16:10:47.051337 Epoch 10, Training loss 0.9624199323032213
 2022-03-31 16:11:52.464671 Epoch 20, Training loss 0.7969129428534252
 2022-03-31 16:12:58.147590 Epoch 30, Training loss 0.7031840597424666
 2022-03-31 16:14:03.656624 Epoch 40, Training loss 0.6305995185661804
 2022-03-31 16:15:09.035426 Epoch 50, Training loss 0.5766544294403032
 2022-03-31 16:16:14.364434 Epoch 60, Training loss 0.5282236191317858
 2022-03-31 16:17:19.712261 Epoch 70, Training loss 0.48431261693653854
 2022-03-31 16:18:25.360500 Epoch 80, Training loss 0.4477320643680175
 2022-03-31 16:19:30.573760 Epoch 90, Training loss 0.414651697954101
 2022-03-31 16:20:35.889676 Epoch 100, Training loss 0.3829454679013518
 2022-03-31 16:21:41.206677 Epoch 110, Training loss 0.3521959857010018
 2022-03-31 16:22:46.786569 Epoch 120, Training loss 0.3273326074490157
 2022-03-31 16:23:52.110456 Epoch 130, Training loss 0.3046942265213603
 2022-03-31 16:24:57.373604 Epoch 140, Training loss 0.2844478160790775
 2022-03-31 16:26:02.606090 Epoch 150, Training loss 0.26248701015854126
 2022-03-31 16:27:07.886430 Epoch 160, Training loss 0.24485927320006864
 2022-03-31 16:28:13.257102 Epoch 170, Training loss 0.22899872874436172
 2022-03-31 16:29:18.518586 Epoch 180, Training loss 0.21485560214923471
 2022-03-31 16:30:23.837559 Epoch 190, Training loss 0.20247267834518268
 2022-03-31 16:31:29.071740 Epoch 200, Training loss 0.18642431348943345
```

```
validate(model5, train_loader, val_loader)
```

```
 Accuracy: val: 0.63
```

✓ 0s completed at 12:31 PM