```
import torch
import torchvision
import os
from PIL import Image
import matplotlib.pyplot as plt
%matplotlib inline
from torch.autograd import Variable
from torch.nn import Linear, ReLU, CrossEntropyLoss, Sequential, Conv2d, MaxPool2d, Module, Softmax, BatchNorm2d, Dr
import torch.nn as nn
from torch.optim import Adam
import numpy as np
from tqdm import tqdm
from google.colab import drive
from torchvision.datasets import CIFAR10
```

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

```
!pip install --upgrade --no-cache-dir gdown
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (4.6.6)
Collecting gdown
  Downloading gdown-4.7.1-py3-none-any.whl (15 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.12.2)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.27.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from gdown) (1.16.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.65.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.11.2)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.4.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2023.5.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.0.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.4)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)
Installing collected packages: gdown
  Attempting uninstall: gdown
    Found existing installation: gdown 4.6.6
    Uninstalling gdown-4.6.6:
      Successfully uninstalled gdown-4.6.6
Successfully installed gdown-4.7.1
```

Data Loading

Download and unzip the dataset

```
# !gdown https://drive.google.com/file/d/1y75TPaomSp2QlXeWWv-WMRbHVCKC5VB_/view?usp=sharing
```

```
/usr/local/lib/python3.10/dist-packages/gdown/parse_url.py:44: UserWarning: You specified a Google Drive link that is not the correct l
  warnings.warn(
Downloading...
From: https://drive.google.com/file/d/1y75TPaomSp2QlXeWWv-WMRbHVCKC5VB_/view?usp=sharing
To: /content/view?usp=sharing
80.2kB [00:00, 522MB/s]
```

```
# !unzip /content/CIFAR-10.zip;
```

```
unzip:  cannot find or open /content/CIFAR-10.zip, /content/CIFAR-10.zip.zip or /content/CIFAR-10.zip.ZIP.
```

Loading and dividing the dataset into train, val and test sets

Initializing ImageFolder Instance

```
transform = torchvision.transforms.Compose([torchvision.transforms.ToTensor(), torchvision.transforms.Resize((90,90)
# data_path = '/content'
dataset = CIFAR10(root='data/train/', download=True, train=True, transform=transform)
```

```
dataset = CIFAR10(root='data/train/',download=True, train=True, transform=transform)
test_set = CIFAR10(root='data/test/',download=True, train=False, transform=transform)
#transform/load data, indexes and loads up the set of sub-dirs
```

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

Attributes of the dataset object

```
len(dataset) # number of samples
```

```
50000
```

```
dataset
```

```
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: data/train/
    Split: Train
    StandardTransform
Transform: Compose(
               ToTensor()
               Resize(size=(90, 90), interpolation=bilinear, max_size=None, antialias=warn)
           )
```

```
dataset.classes # classes(build in?????, figure this out later)
```

```
['airplane',
 'automobile',
 'bird',
 'cat',
 'deer',
 'dog',
 'frog',
 'horse',
 'ship',
 'truck']
```

```
dataset.class_to_idx # indices of corresponding labels
```

```
{'airplane': 0,
 'automobile': 1,
 'bird': 2,
 'cat': 3,
 'deer': 4,
 'dog': 5,
 'frog': 6,
 'horse': 7,
 'ship': 8,
 'truck': 9}
```

```
dataset.transform
```

```
Compose(
    ToTensor()
    Resize(size=(90, 90), interpolation=bilinear, max_size=None, antialias=warn)
)
```

```
class_counts = np.zeros(len(dataset.class_to_idx))
```

```
for image, label in dataset:
    class_counts[label] += 1
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default value of the antialias para
  warnings.warn(
```

```
for i in range(len(class_counts)):
  print("class:%s, instances: %d"%([k for k,v in dataset.class_to_idx.items() if v == i], class_counts[i]))
```

```
class:['airplane'], instances: 5000
class:['automobile'], instances: 5000
class:['bird'], instances: 5000
```

```
class:['cat'], instances: 5000
class:['deer'], instances: 5000
class:['dog'], instances: 5000
class:['frog'], instances: 5000
class:['horse'], instances: 5000
class:['ship'], instances: 5000
class:['truck'], instances: 5000
```

```
def display_img(img,label):
    #print(f"Label : {dataset.classes[label]}")
    plt.figure()
    plt.title(f"Label : {dataset.classes[label]}")
    plt.imshow(img.permute(1,2,0))

#display the first image in the dataset
for i in range(1,4):
  display_img(*dataset[i])
```

Split into train, validation and test sets

```
# test train split, utils package, hard coded??? takes fractions??? check later,,,,, manual seed >> random seeding!!
# train_set, val_set, test_set = torch.utils.data.random_split(dataset, [11924, 2555, 2555], generator=torch.Generat
```

```
train_set = dataset
# print(train_set)
```

```
# train_set.indices
```

```
# train_set.dataset # dataset means train set is a part/subset of the dataset!,
```

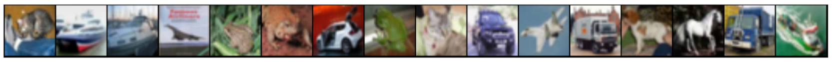
```
def display_img(img,label):
    print(f"Label : {dataset.classes[label]}")
    plt.imshow(img.permute(1,2,0))

#display the first image in the dataset
display_img(*train_set[1000])
```

Initializing the pytorch dataloaders

```
train_loader = torch.utils.data.DataLoader(
    train_set,
    batch_size=16,
    shuffle=True
)

val_loader = torch.utils.data.DataLoader(
    test_set,
    batch_size=16,
    shuffle=False
)

test_loader = torch.utils.data.DataLoader(
    test_set,
    batch_size=16,
    shuffle=False
)
```

```
from torchvision.utils import make_grid

def show_batch(loader):
    """Plot images grid of single batch"""
    for images, labels in loader:
```

```
        fig,ax = plt.subplots(figsize = (16,12))
        ax.set_xticks([])
        ax.set_yticks([])
        ax.imshow(make_grid(images,nrow=16).permute(1,2,0))
        break

show_batch(val_loader)
```



Convolutional Neural Networks

nn.Module method of constructing models

```
#TF>> sequential - simple
#  >> modular!! >> complex(flexible)
```

```
#Modular way!!!!
#torch.set_default_tensor_type('torch.cuda.FloatTensor')

class AlexNet(torch.nn.Module):
    def __init__(self):
        super(AlexNet, self).__init__() #init parent class

        # Defining a 2D convolution layer
        self.conv1 = Conv2d(3, 96, kernel_size=11, stride=4, padding=1)#(input channels, output channels,------)
        self.bn1 = BatchNorm2d(96)
        self.relu1 = ReLU(inplace=True) ## note thr inplace, is it important????
        self.maxpool1 = MaxPool2d(kernel_size=3, stride=2)

        # Defining another 2D convolution layer
        self.conv2 = Conv2d(96, 256, kernel_size=5, padding=2)
        self.bn2 = BatchNorm2d(256)
        self.relu2 = ReLU(inplace=True)
        self.maxpool2 = MaxPool2d(kernel_size=3, stride=2)

        self.conv3 = Conv2d(256, 384, kernel_size=3, padding=1)
        self.bn3 = BatchNorm2d(384)
        self.relu3 = ReLU(inplace=True)
        # self.maxpool3 = MaxPool2d(kernel_size=3)

        self.conv4 = Conv2d(384, 384, kernel_size=3, padding=1)
        self.bn4 = BatchNorm2d(384)
        self.relu4 = ReLU(inplace=True)
        # self.maxpool4 = MaxPool2d(kernel_size=3)

        self.conv5 = Conv2d(384, 256, kernel_size=3, padding=1)
        self.bn5 = BatchNorm2d(256)
        self.relu5 = ReLU(inplace=True)
        self.maxpool3 = MaxPool2d(kernel_size=3, stride=2)
        #linear(input,output)
        self.linear_layers = Linear(256, 10)  ## flatten the image here. i.e linear layer!!

    # Defining the forward pass
    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
```

```
        x = self.relu1(x)
        x = self.maxpool1(x)

        # Apply conv2, bn2, relu2 and maxpool2
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.relu2(x)
        x = self.maxpool2(x)

        # Apply conv3, bn3, relu3 and maxpool3
        x = self.conv3(x)
        x = self.bn3(x)
        x = self.relu3(x)
        # x = self.maxpool3(x)

        # Apply conv4, bn4, relu4 and maxpool4
        x = self.conv4(x)
        x = self.bn4(x)
        x = self.relu4(x)
        # x = self.maxpool4(x)
        x = self.conv5(x)
        x = self.bn5(x)
        x = self.relu5(x)
        x = self.maxpool3(x)

        # Flatten the output from the conv layers
        x = x.view(x.size(0), -1)

        # Apply the linear layer
        x = self.linear_layers(x)
        return x

# Initialize the model
model = AlexNet()
model.to(device)  # to GPU/CPU!!!!!!
```

```
AlexNet(
  (conv1): Conv2d(3, 96, kernel_size=(11, 11), stride=(4, 4), padding=(1, 1))
  (bn1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): ReLU(inplace=True)
  (maxpool1): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(96, 256, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2): ReLU(inplace=True)
  (maxpool2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv2d(256, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn3): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu3): ReLU(inplace=True)
  (conv4): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn4): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu4): ReLU(inplace=True)
  (conv5): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu5): ReLU(inplace=True)
  (maxpool3): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  (linear_layers): Linear(in_features=256, out_features=10, bias=True)
)
```

sequential method of constructing models

```
# sequential way

model = Sequential(
            # Defining a 2D convolution layer
            Conv2d(3, 4, kernel_size=3, stride=1, padding=1),
            BatchNorm2d(4),
            ReLU(inplace=True),
            MaxPool2d(kernel_size=2, stride=2),
            # Defining another 2D convolution layer
```

```
        Conv2d(4, 8, kernel_size=3, stride=1, padding=1),
        BatchNorm2d(8),
        ReLU(inplace=True),
        MaxPool2d(kernel_size=2, stride=2),
        Conv2d(8, 16, kernel_size=3, stride=1, padding=1),
        BatchNorm2d(16),
        ReLU(inplace=True),
        MaxPool2d(kernel_size=2, stride=2),
        Conv2d(16, 32, kernel_size=3, stride=1, padding=1),
        BatchNorm2d(32),
        ReLU(inplace=True),
        MaxPool2d(kernel_size=2, stride=2),
        nn.Flatten(),
        Linear(32 * 5 * 5, 6)
    )
```

Visualization of Model

Visualize through torchsummary

```
from torchsummary import summary
```

```
dummy_model = AlexNet().to(device)
summary(dummy_model, (3, 90, 90))  #-1 correspnds to batch size!, -1>> last or all remaining
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 96, 21, 21]          34,944
       BatchNorm2d-2           [-1, 96, 21, 21]             192
              ReLU-3           [-1, 96, 21, 21]               0
         MaxPool2d-4           [-1, 96, 10, 10]               0
            Conv2d-5          [-1, 256, 10, 10]         614,656
       BatchNorm2d-6          [-1, 256, 10, 10]             512
              ReLU-7          [-1, 256, 10, 10]               0
         MaxPool2d-8            [-1, 256, 4, 4]               0
            Conv2d-9            [-1, 384, 4, 4]         885,120
      BatchNorm2d-10            [-1, 384, 4, 4]             768
             ReLU-11            [-1, 384, 4, 4]               0
           Conv2d-12            [-1, 384, 4, 4]       1,327,488
      BatchNorm2d-13            [-1, 384, 4, 4]             768
             ReLU-14            [-1, 384, 4, 4]               0
           Conv2d-15            [-1, 256, 4, 4]         884,992
      BatchNorm2d-16            [-1, 256, 4, 4]             512
             ReLU-17            [-1, 256, 4, 4]               0
        MaxPool2d-18            [-1, 256, 1, 1]               0
           Linear-19                   [-1, 10]           2,570
================================================================
Total params: 3,752,522
Trainable params: 3,752,522
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.09
Forward/backward pass size (MB): 2.04
Params size (MB): 14.31
Estimated Total Size (MB): 16.44
----------------------------------------------------------------
```

Visualize through TorchViz

```
!pip install torchviz
```

```
Collecting torchviz
  Downloading torchviz-0.0.2.tar.gz (4.9 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from torchviz) (2.0.1+cu118)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from torchviz) (0.20.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->torchviz) (3.12.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch->torchviz) (4.6.3)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->torchviz) (1.11.1)
```

```
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch->torchviz) (3.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->torchviz) (3.1.2)
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packages (from torch->torchviz) (2.0.0)
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch->torchviz) (3.25.2)
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch->torchviz) (16.0.6)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->torchviz) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->torchviz) (1.3.0)
Building wheels for collected packages: torchviz
  Building wheel for torchviz (setup.py) ... done
  Created wheel for torchviz: filename=torchviz-0.0.2-py3-none-any.whl size=4131 sha256=c3e4e516524682a45b19d0e53e529500fc3f90079c6caa0
  Stored in directory: /root/.cache/pip/wheels/4c/97/88/a02973217949e0db0c9f4346d154085f4725f99c4f15a87094
Successfully built torchviz
Installing collected packages: torchviz
Successfully installed torchviz-0.0.2
```

```
from torchviz import make_dot
dummy_image = next(iter(train_loader))[0]
dummy_model = AlexNet()
y_hat = dummy_model(dummy_image)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default value of the antialias para
  warnings.warn(
```

```
make_dot(y_hat.mean(),params=dict(dummy_model.named_parameters())).render("graph2", format="png")
```

```
'graph2.png'
```

```
dummy_model.state_dict() # show the entire as dict!
```

```
[[ 0.0155, 0.0057, 0.0524],
 [-0.0534,  0.0164,  0.0792],
 [-0.0192, -0.0060,  0.0397]]],


[[[-0.0686, -0.0154,  0.0312],
  [-0.0329, -0.0784, -0.0266],
  [ 0.0620,  0.0494, -0.0823]],
```

Model Configuration

```
# Select a loss function
loss_function = torch.nn.CrossEntropyLoss()

# Select an optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=0.00001)
```

Training Loop

Variable Initialization

```
len(train_loader) # number of images per batch
```

```
3125
```

```
def run_1_epoch(model, loss_fn, optimizer, loader, train = False):


  if train:
    model.train()
  else:
    model.eval()

  total_correct_preds = 0

  total_samples_in_loader = len(train_set)
  total_batches_in_loader = len(train_loader)

  total_loss = 0

  for image_batch, labels in tqdm(train_loader): #TPDM gives you proegress updates!

    # Transfer image_batch to GPU if available
    image_batch = image_batch.to(device) #gives you images of the batch
    labels = labels.to(device) #gives you labels of the batch

    # Zeroing out the gradients for parameters
    if train:
      optimizer.zero_grad() # pytorch doesnt reset gradients to zeros after each iteration and keeps adding to the p
      #if this is not done

    # Forward pass on the input batch
    output = model.forward(image_batch)

    # Acquire predicted class indices
    _, predicted = torch.max(output.data, 1) # the dimension 1 corresponds to max along the rows


    # Removing extra last dimension from output tensor
    output.squeeze_(-1)

    # Compute the loss for the minibatch
    loss = loss_function(output, labels)
```

```
    # Backpropagation
    if train:
      loss.backward()

    # Update the parameters using the gradients
    if train:
      optimizer.step()

    # Extra variables for calculating loss and accuracy
    # count total predictions for accuracy calcutuon for this epoch
    total_correct_preds += (predicted == labels).sum().item()

    total_loss += loss.item()

  loss = total_loss / total_batches_in_loader
  accuracy = 100 * total_correct_preds / total_samples_in_loader

  return loss, accuracy


epochs = 30

train_accuracy_list = []
val_accuracy_list = []

train_loss_list = []
val_loss_list = []

val_accuracy_max = -1 # used to store best model based on accuracy!


if torch.cuda.is_available():
    model.cuda()


# Main training and validation loop for n number of epochs
for i in range(epochs):

  # Train model for one epoch
  print("Epoch %d: Train"%(i))
  train_loss, train_accuracy  = run_1_epoch(model, loss_function, optimizer, train_loader, train= True)

  # Lists for train loss and accuracy for plotting
  train_loss_list.append(train_loss)
  train_accuracy_list.append(train_accuracy)

  # Validate the model on validation set
  print("Epoch %d: Validation"%(i))
  with torch.no_grad():
    val_loss, val_accuracy  = run_1_epoch(model, loss_function, optimizer, val_loader, train= False)

  # Lists for val loss and accuracy for plotting
  val_loss_list.append(val_loss)
  val_accuracy_list.append(val_accuracy)

  print('train loss: %.4f'%(train_loss))
  print('val loss: %.4f'%(val_loss))
  print('train_accuracy %.2f' % (train_accuracy))
  print('val_accuracy %.2f' % (val_accuracy))

  # Save model if validation accuracy for current epoch is greater than
  # all the previous epochs
  if val_accuracy > val_accuracy_max:
    val_accuracy_max = val_accuracy
    print("New Max val Accuracy Acheived %.2f. Saving model.\n\n"%(val_accuracy_max))
    torch.save(model,'best_val_acc_model.pth')
  else:
```

```
print("val accuracy did not increase from %.2f\n\n"%(val_accuracy_max))
```

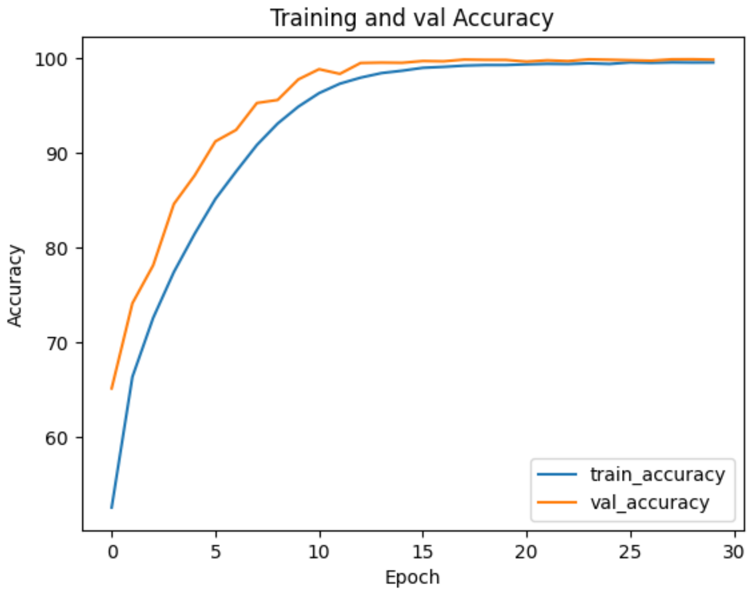
```
Epoch 25: Train
100%|██████████| 3125/3125 [00:48<00:00, 63.97it/s]
Epoch 25: Validation
100%|██████████| 3125/3125 [00:38<00:00, 81.39it/s]
train loss: 0.0219
val loss: 0.0118
train_accuracy 99.61
val_accuracy 99.83
val accuracy did not increase from 99.93


Epoch 26: Train
100%|██████████| 3125/3125 [00:48<00:00, 64.14it/s]
Epoch 26: Validation
100%|██████████| 3125/3125 [00:38<00:00, 81.77it/s]
train loss: 0.0225
val loss: 0.0138
train_accuracy 99.56
val_accuracy 99.79
val accuracy did not increase from 99.93


Epoch 27: Train
100%|██████████| 3125/3125 [00:48<00:00, 64.38it/s]
Epoch 27: Validation
100%|██████████| 3125/3125 [00:37<00:00, 82.81it/s]
train loss: 0.0208
val loss: 0.0082
train_accuracy 99.61
val_accuracy 99.93
New Max val Accuracy Acheived 99.93. Saving model.


Epoch 28: Train
100%|██████████| 3125/3125 [00:49<00:00, 63.46it/s]
Epoch 28: Validation
100%|██████████| 3125/3125 [00:37<00:00, 82.50it/s]
train loss: 0.0209
val loss: 0.0077
train_accuracy 99.59
val_accuracy 99.94
New Max val Accuracy Acheived 99.94. Saving model.


Epoch 29: Train
100%|██████████| 3125/3125 [00:48<00:00, 64.06it/s]
Epoch 29: Validation
100%|██████████| 3125/3125 [00:37<00:00, 82.79it/s]train loss: 0.0197
val loss: 0.0083
train_accuracy 99.60
val_accuracy 99.90
val accuracy did not increase from 99.94
```

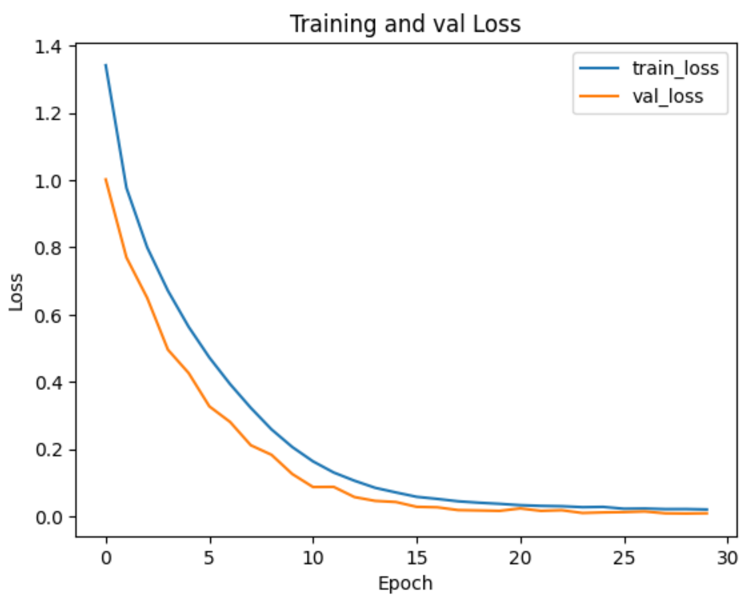
Accuracy and Loss Result Graphs:

```
plt.figure()
plt.plot(train_accuracy_list, label="train_accuracy")
plt.plot(val_accuracy_list, label="val_accuracy")
plt.legend()

plt.xlabel('Epoch')
plt.ylabel('Accuracy')

plt.title('Training and val Accuracy')

plt.figure()
plt.plot(train_loss_list, label="train_loss")
plt.plot(val_loss_list, label="val_loss")
```

```
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and val Loss')
```

```
Text(0.5, 1.0, 'Training and val Loss')
```





Evaluating the Model

Loading the best saved model:

```
best_val_model1 = torch.load('/content/best_val_acc_model.pth')
```

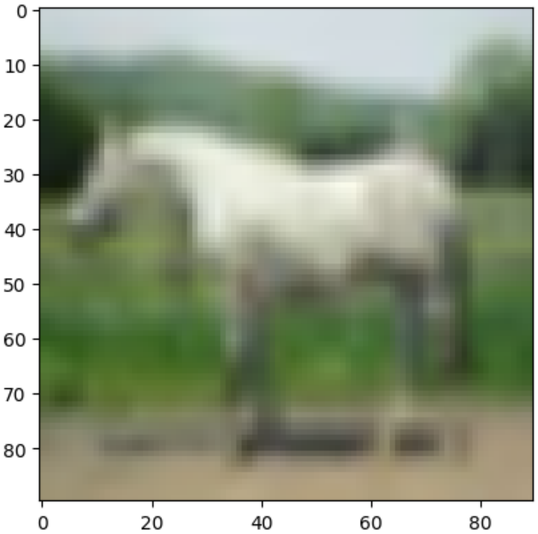
```
torch.save(best_val_model1,'best_val_acc_model.pth')
```

```
with torch.no_grad():
    test_loss, test_accuracy  = run_1_epoch(model, loss_function, optimizer, test_loader, train= False)

print('test loss: %.4f'%(train_loss))
print('test_accuracy %.2f' % (train_accuracy))
```

```
100%|██████████| 3125/3125 [00:38<00:00, 82.02it/s]test loss: 0.0197
test_accuracy 99.60
```

```
def display_img(img,label):
    print(f"Label : {dataset.classes[label]}")
    plt.imshow(img.permute(1,2,0))

#display the first image in the dataset
display_img(*test_set[2340])
```

```
Label : horse
```



Single Sample Example

```
model_test= torch.load('/content/best_val_acc_model.pth')
test_imgset = CIFAR10(root='data/test/',download=True, train=False)
transform1 = torchvision.transforms.Compose([
    torchvision.transforms.Resize((90, 90)),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
input = transform1(test_imgset[213][0])
input = input.unsqueeze(0)
output=model_test(input.to(device)).to(device)
output.max(dim=1), test_imgset[213][1]
# test_imgset[0][1]
```

```
Files already downloaded and verified
(torch.return_types.max(
 values=tensor([4.8348], device='cuda:0', grad_fn=<MaxBackward0>),
 indices=tensor([9], device='cuda:0')),
 9)
```

Predicted→ (indices line)
Original→ (last line)

Credits

Notebook Prepared by Bostan Khan, Team Lead, MachVIS Lab

0s completed at 5:53 PM