RockPaperScissor

June 23, 2022

1 Rock-Paper-Scissor Competition (40%)

For this competition, we will use the Game (https://cloudstor.aarnet.edu.au/plus/s/6QNijohkrfMZ0H7) dataset. This dataset contains images of hand gestures from the Rock-Paper-Scissors game.

The dataset contains a total of 2188 images corresponding to the 'Rock' (726 images), 'Paper' (710 images) and 'Scissors' (752 images) hand gestures of the Rock-Paper-Scissors game. All image are taken on a green background with relatively consistent ligithing and white balance.

All images are RGB images of 300 pixels wide by 200 pixels high in .png format. The images are separated in three sub-folders named 'rock', 'paper' and 'scissors' according to their respective class.

The task is to categorize each hand guesters into one of three categories (Rock/Paper/Scissor).

We provide a baseline by the following steps:

- Loding and Analysing the dataset using torchvision.
- Defining a simple convolutional neural network.
- How to use existing loss function for the model learning.
- Train the network on the training data.
- Test the trained network on the testing data.

1.1 The following trick/tweak(s) could be considered:

1. Change of advanced training parameters: Learning Rate, Optimizer, Batch-size, Number of Max Epochs, and Drop-out.
2. Use of a new loss function.
3. Data augmentation
4. Architectural Changes: Batch Normalization, Residual layers, Attention Block, and other varients.

Your code should be modified from the provided baseline. A pdf report is required to explain the tricks you employed, and the imporvements they achieved. Marking Rules: ——- We will mark the competition based on the final test accuracy on testing images and your report.

Final mark = acc_mark + efficiency mark + report mark + bonus mark ###Acc_mark 15:

We will rank all the submission results based on their test accuracy. The top 30% of the students will get full marks.

Accuracy	Mark
Top 30% in the class	15
30%-50%	11
50%-80%	7
80%-90%	3
90%-100%	1
Not implemented	0

###Efficiency mark 5:

Efficiency is evaluated by the computational costs (flops: https://en.wikipedia.org/wiki/FLOPS). Please report the computational costs for your final model and attach the code/process about how you calculate it.

Efficiency	Mark
Top 30% in the class	5
30%-50%	4
50%-80%	3
80%-90%	2
90%-100%	2
Not implemented	0

###Report mark 20: 1. Introduction and your understanding to the baseline model: 2 points

2. Employed more than three tricks with ablation studies to improve the accuracy: 6 points

Clearly explain the reference, motivation and design choice for each trick/tweak(s). Providing the experimental results in tables. Example table:

Trick1	Trick2	Trick3	Accuracy
N	N	N	60%
Y	N	N	65%
Y	Y	N	77%
Y	Y	Y	82%

Observation and discussion based on the experiment results.

3. Expaination of the methods on reducing the computational cost and/or improve the trade-off between accuracy and efficiency: 4 points
4. Explaination of the code implementation 3 points
5. Visulization results: e.g. training and testing accuracy/loss for each model, case studies: 3 points
6. Open ended: Limitations, conclusions, failure cases analysis...: 2 points

###Bouns mark: 1. Top three results: 2 points 2. Fancy designs: 2 points

[ ]:
```
##########################################################################################
### Subject: Computer Vision
### Year: 2022
### Student Name: MELARN MURPHY, JET HONG LOW
### Student ID: a1771928, a1820924
### Comptetion Name: Rock-Paper-Scissor Classification Competition
### Final Results:{'val_acc': 1.0, 'val_loss': 0.018957555294036865}
### ACC: 100%  FLOPs:0.86G
##########################################################################################
```

[ ]:
```
# Importing libraries.

import os
import random
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from tqdm.notebook import tqdm

# To avoid non-essential warnings
import warnings
warnings.filterwarnings('ignore')

from torchvision import datasets, transforms, models
from torchvision.datasets import ImageFolder
from torchvision.transforms import ToTensor
from torchvision.utils import make_grid
from torch.utils.data import random_split
from torch.utils.data.dataloader import DataLoader
import matplotlib.pyplot as plt
%matplotlib inline
```

[ ]:
```
# # Mounting Melarn's G-Drive to get the game dataset.
# # To access Google Colab GPU; Go To: Edit >>> Network Settings >>> Hardware␣
↪Accelarator: Select GPU.
# # Reference: https://towardsdatascience.com/
↪google-colab-import-and-export-datasets-eccf801e2971
from google.colab import drive
drive.mount('/content/drive')

# # Dataset path.
data_dir = '/content/drive/MyDrive/ColabNotebooks/Dataset/rps-cv-images'
classes = os.listdir(data_dir)
```

```
Mounted at /content/drive
```

[ ]:
```python
# Performing Image Transformations.
## No data augmentation applied
train_transform=transforms.Compose([
        transforms.Resize(40),             # resize shortest side Hints: larger␣
↪input size can lead to higher performance
        transforms.CenterCrop(40),         # crop longest side Hints: crop size␣
↪is usuallt smaller than the resize size
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
])
```

[ ]:
```python
# Checking the dataset training size.
dataset = ImageFolder(data_dir, transform=train_transform)
print('Size of training dataset :', len(dataset))
```

```
Size of training dataset : 2188
```

[ ]:
```python
# Viewing one of images shape.
img, label = dataset[100]
print(img.shape)
```

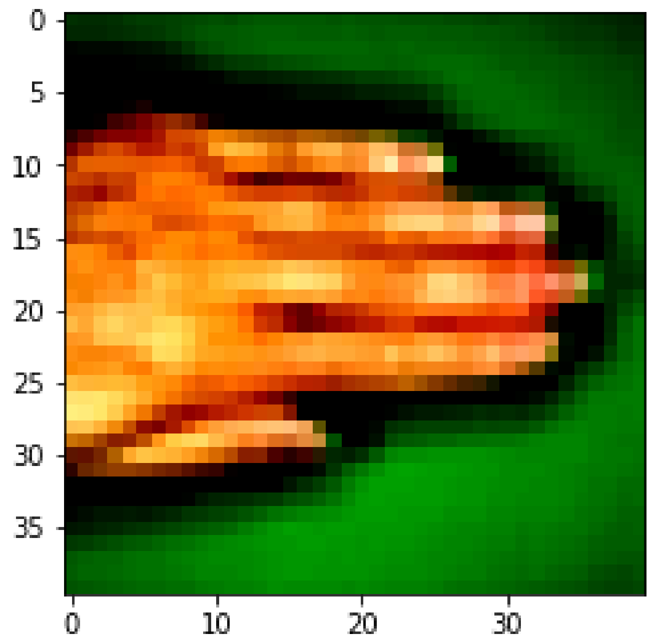
```
torch.Size([3, 40, 40])
```

[ ]:
```python
# Preview one of the images..
def show_image(img, label):
    print('Label: ', dataset.classes[label], "("+str(label)+")")
    plt.imshow(img.permute(1,2,0))
```

[ ]:
```python
show_image(*dataset[200])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

```
Label:  paper (0)
```




```
[ ]: # Setting seed so that value won't change everytime.
     # Splitting the dataset to training, validation, and testing category.
     torch.manual_seed(10)
     val_size = len(dataset)//10
     test_size = len(dataset)//5
     train_size = len(dataset) - val_size - test_size
```

```
[ ]: # Random Splitting.
     train_ds, val_ds, test_ds = random_split(dataset, [train_size, val_size,␣
      ↪test_size])
     len(train_ds), len(val_ds),len(test_ds)
```

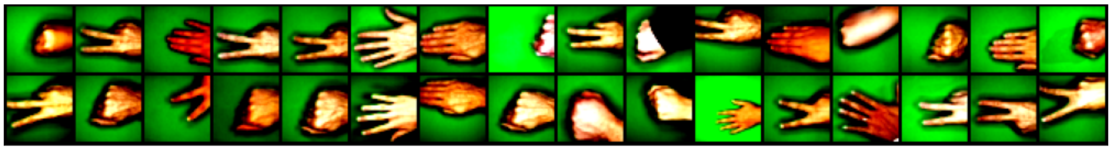
```
[ ]: (1533, 218, 437)
```

```
[ ]: batch_size = 32
     train_loader = DataLoader(train_ds, batch_size, shuffle=True, num_workers=2,␣
      ↪pin_memory=True)
     val_loader = DataLoader(val_ds, batch_size*2, num_workers=2, pin_memory=True)
     test_loader = DataLoader(test_ds, batch_size*2, num_workers=2, pin_memory=True)
```

```
[ ]: # Multiple images preview.
     for images, labels in train_loader:
         fig, ax = plt.subplots(figsize=(18,10))
         ax.set_xticks([])
         ax.set_yticks([])
```

```
    ax.imshow(make_grid(images, nrow=16).permute(1, 2, 0))
    break
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



```
[ ]: # Baseline model class for training and validation purpose. Evaluation metric␣
     ↪function - Accuracy.

def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))

class ImageClassificationBase(nn.Module):
    def training_step(self, batch):
        images, labels = batch
        out = self(images)                  # Generate predictions
        loss = F.cross_entropy(out, labels) # Calculate loss
        return loss

    def validation_step(self, batch):
        images, labels = batch
        out = self(images)                    # Generate predictions
        loss = F.cross_entropy(out, labels)   # Calculate loss
        acc = accuracy(out, labels)           # Calculate accuracy
        return {'val_loss': loss.detach(), 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()   # Combine losses
        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean()      # Combine accuracies
        return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

    def epoch_end(self, epoch, result):
        print("Epoch [{}], train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.
↪4f}".format(
            epoch, result['train_loss'], result['val_loss'], result['val_acc']))
```

[ ]:
```python
# Functions for evaluation and training.
def evaluate(model, val_loader):
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.SGD):
    history = []
    optimizer = opt_func(model.parameters(), lr)
    for epoch in range(epochs):
        # Training Phase
        model.train()
        train_losses = []
        for batch in tqdm(train_loader):
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
        # Validation phase
        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        model.epoch_end(epoch, result)
        history.append(result)
    return history
```

[ ]:
```python
# To check wether Google Colab GPU has been assigned/not.

def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return None

def to_device(data, device):
    """Move tensor(s) to chosen device"""
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():
    """Wrap a dataloader to move data to a device"""
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
```

```
        """Yield a batch of data after moving it to device"""
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        """Number of batches"""
        return len(self.dl)
```

[ ]:
```
device = get_default_device()
device
train_loader = DeviceDataLoader(train_loader, device)
val_loader = DeviceDataLoader(val_loader, device)
test_loader = DeviceDataLoader(test_loader, device)
```

[ ]:
```
input_size = 3*40*40
output_size = 3
```

[ ]:
```
# COMBINATION OF PRELU ACTIVATION FUNCTION MODEL
class CnnModel_batchnorm(ImageClassificationBase):
    def __init__(self, classes):
        super().__init__()
        self.classes = classes
        self.network = nn.Sequential(
            nn.Conv2d(3, 100, kernel_size=3, padding=1),
            nn.BatchNorm2d(100),
            nn.ELU(), # FROM RELU, MR
            nn.Conv2d(100, 150, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(150),
            nn.PReLU(), # from softmax
            #nn.Conv2d(150, 150, kernel_size=3, stride=1, padding=1),
            #nn.BatchNorm2d(150),
            #nn.PReLU(),
            nn.MaxPool2d(2,2),

            nn.Conv2d(150, 200, kernel_size=3, stride=1, padding=1),
            nn.Softmax2d(),
            nn.BatchNorm2d(200),
            #nn.Dropout2d(p=0.6),
            #nn.Conv2d(200, 200, kernel_size=3, stride=1, padding=1),
            #nn.PReLU(),
            nn.MaxPool2d(2, 2),

            nn.Conv2d(200, 250, kernel_size=3, stride=1, padding=1),
            #nn.BatchNorm2d(250),
            nn.Softmax2d(), #
            nn.Conv2d(250, 250, kernel_size=3, stride=1, padding=1),
            #nn.BatchNorm2d(250),
```

```python
            nn.SELU(),
            nn.Dropout2d(p=0.6),
            nn.MaxPool2d(2, 2),

            nn.Flatten(),
            nn.Linear(6250, 32),
            nn.BatchNorm1d(32),
            nn.ReLU(),
            nn.Linear(32, 32),
            nn.BatchNorm1d(32),
            nn.PReLU(),
            nn.Linear(32, 8),
            nn.BatchNorm1d(8),
            nn.ELU(),
            nn.Linear(8, self.classes))

    def forward(self, xb):
        return self.network(xb)
```

[ ]:
```python
# Model print
num_classes = 3
model = CnnModel_batchnorm(num_classes)
model.cuda()
```

[ ]:
```
CnnModel_batchnorm(
  (network): Sequential(
    (0): Conv2d(3, 100, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): Conv2d(100, 150, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(150, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): PReLU(num_parameters=1)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (7): Conv2d(150, 200, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): Softmax2d()
    (9): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (11): Conv2d(200, 250, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): Softmax2d()
    (13): Conv2d(250, 250, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (14): SELU()
    (15): Dropout2d(p=0.6, inplace=False)
```

```
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (17): Flatten(start_dim=1, end_dim=-1)
    (18): Linear(in_features=6250, out_features=32, bias=True)
    (19): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (20): ReLU()
    (21): Linear(in_features=32, out_features=32, bias=True)
    (22): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (23): PReLU(num_parameters=1)
    (24): Linear(in_features=32, out_features=8, bias=True)
    (25): BatchNorm1d(8, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (26): ELU(alpha=1.0)
    (27): Linear(in_features=8, out_features=3, bias=True)
  )
)
```

[ ]:
```
for images, labels in train_loader:
    out = model(images)
    print('images.shape:', images.shape)
    print('out.shape:', out.shape)
    print('out[0]:', out[0])
    break
```

```
images.shape: torch.Size([32, 3, 40, 40])
out.shape: torch.Size([32, 3])
out[0]: tensor([-0.1591,  0.3327, -0.5152], device='cuda:0',
grad_fn=<SelectBackward0>)
```

[ ]:
```
train_dl = DeviceDataLoader(train_loader, device)
val_dl = DeviceDataLoader(val_loader, device)
to_device(model, device)
```

[ ]:
```
CnnModel_batchnorm(
  (network): Sequential(
    (0): Conv2d(3, 100, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): Conv2d(100, 150, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(150, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): PReLU(num_parameters=1)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
```

```
    (7): Conv2d(150, 200, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): Softmax2d()
    (9): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (11): Conv2d(200, 250, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): Softmax2d()
    (13): Conv2d(250, 250, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (14): SELU()
    (15): Dropout2d(p=0.6, inplace=False)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (17): Flatten(start_dim=1, end_dim=-1)
    (18): Linear(in_features=6250, out_features=32, bias=True)
    (19): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (20): ReLU()
    (21): Linear(in_features=32, out_features=32, bias=True)
    (22): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (23): PReLU(num_parameters=1)
    (24): Linear(in_features=32, out_features=8, bias=True)
    (25): BatchNorm1d(8, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (26): ELU(alpha=1.0)
    (27): Linear(in_features=8, out_features=3, bias=True)
  )
)
```

[ ]:
```
@torch.no_grad()
def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

def fit(epochs, lr, model, train_loader, val_loader, opt_func):
    history = []
    optimizer = opt_func(model.parameters(), lr)
    for epoch in range(epochs):
        # Training Phase
        model.train()
        train_losses = []
        for batch in tqdm(train_loader):
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()
```

```
            optimizer.step()
            optimizer.zero_grad()
        # Validation phase
        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        model.epoch_end(epoch, result)
        history.append(result)
    return history
```

[ ]:
```
model = to_device(CnnModel_batchnorm(num_classes), device)
```

[ ]:
```
history=[evaluate(model, val_loader)]
history
```

[ ]:
```
[{'val_acc': 0.3209134638309479, 'val_loss': 1.129584789276123}]
```

[ ]:
```
num_epochs = 12
opt_func = torch.optim.Adam
lr = 0.001
```

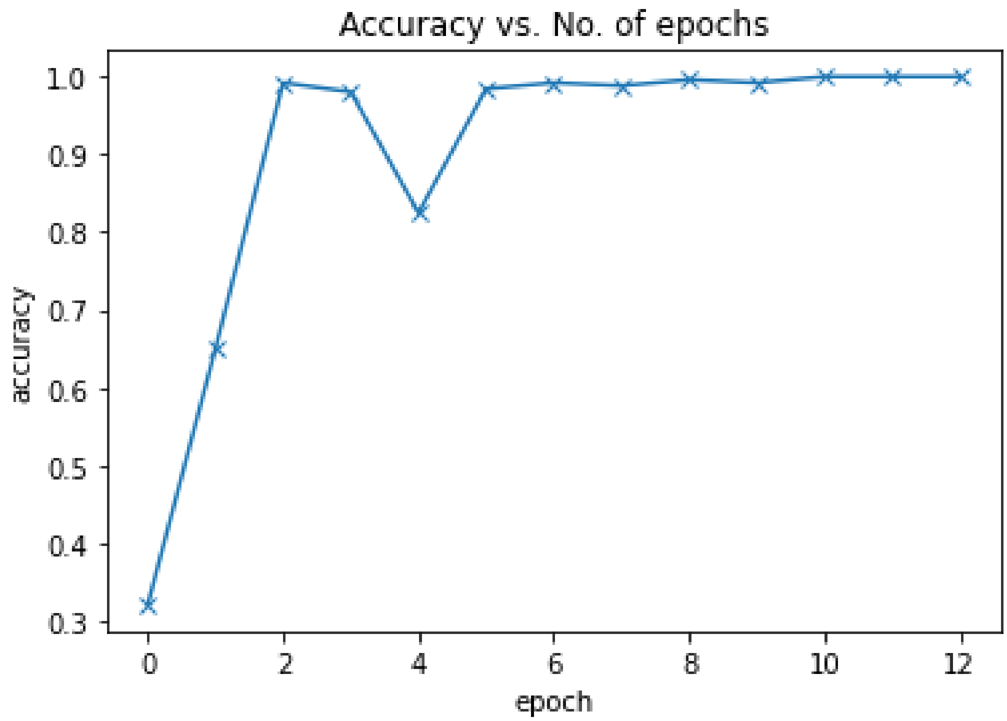
[ ]:
```
history+= fit(num_epochs, lr, model, train_dl, val_dl, opt_func)
```

```
  0%|          | 0/48 [00:00<?, ?it/s]

Epoch [0], train_loss: 0.6000, val_loss: 0.9825, val_acc: 0.6499

  0%|          | 0/48 [00:00<?, ?it/s]

Epoch [1], train_loss: 0.3219, val_loss: 0.2466, val_acc: 0.9922

  0%|          | 0/48 [00:00<?, ?it/s]

Epoch [2], train_loss: 0.2095, val_loss: 0.1866, val_acc: 0.9805

  0%|          | 0/48 [00:00<?, ?it/s]

Epoch [3], train_loss: 0.1509, val_loss: 0.4439, val_acc: 0.8248

  0%|          | 0/48 [00:00<?, ?it/s]

Epoch [4], train_loss: 0.1173, val_loss: 0.1426, val_acc: 0.9844

  0%|          | 0/48 [00:00<?, ?it/s]

Epoch [5], train_loss: 0.1109, val_loss: 0.0864, val_acc: 0.9922

  0%|          | 0/48 [00:00<?, ?it/s]

Epoch [6], train_loss: 0.0900, val_loss: 0.0806, val_acc: 0.9883

  0%|          | 0/48 [00:00<?, ?it/s]

Epoch [7], train_loss: 0.0575, val_loss: 0.0398, val_acc: 0.9961

  0%|          | 0/48 [00:00<?, ?it/s]
```

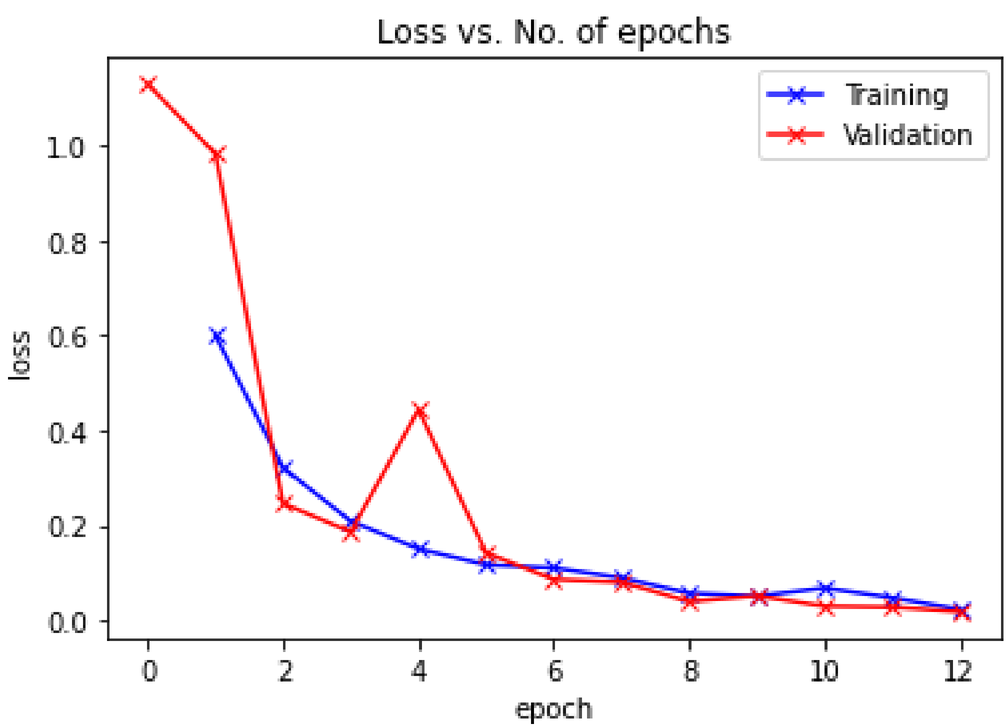
```
Epoch [8], train_loss: 0.0512, val_loss: 0.0519, val_acc: 0.9922

  0%|          | 0/48 [00:00<?, ?it/s]

Epoch [9], train_loss: 0.0685, val_loss: 0.0301, val_acc: 1.0000

  0%|          | 0/48 [00:00<?, ?it/s]

Epoch [10], train_loss: 0.0479, val_loss: 0.0283, val_acc: 1.0000

  0%|          | 0/48 [00:00<?, ?it/s]

Epoch [11], train_loss: 0.0235, val_loss: 0.0186, val_acc: 1.0000
```

[ ]:
```
def plot_accuracies(history):
    accuracies = [x['val_acc'] for x in history]
    plt.plot(accuracies, '-x')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.title('Accuracy vs. No. of epochs')
    plt.show()

def plot_losses(history):
    train_losses = [x.get('train_loss') for x in history]
    val_losses = [x['val_loss'] for x in history]
    plt.plot(train_losses, '-bx')
    plt.plot(val_losses, '-rx')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend(['Training', 'Validation'])
    plt.title('Loss vs. No. of epochs')
    plt.show()
```

[ ]:
```
plot_accuracies(history)
```




```
[ ]: plot_losses(history)
```



[ ]:
```
evaluate(model, test_loader)
```

[ ]:
```
{'val_acc': 1.0, 'val_loss': 0.018957555294036865}
```

##FLOPs

[ ]:
```
#The code from https://cloudstor.aarnet.edu.au/plus/s/PcSc67ZncTSQP0E can be
↪used to count flops
 #Download the code.
!wget -c https://cloudstor.aarnet.edu.au/plus/s/hXo1dK9SZqiEVn9/download
!mv download FLOPs_counter.py
 #!rm -rf download
```

```
--2022-06-23 11:28:11--
https://cloudstor.aarnet.edu.au/plus/s/hXo1dK9SZqiEVn9/download
Resolving cloudstor.aarnet.edu.au (cloudstor.aarnet.edu.au)… 202.158.207.20
Connecting to cloudstor.aarnet.edu.au
(cloudstor.aarnet.edu.au)|202.158.207.20|:443… connected.
HTTP request sent, awaiting response… 200 OK
Syntax error in Set-Cookie: 5230042dc1897=e41ojqooq724eb4bvr24729r8d;
path=/plus;; Secure at position 53.
Syntax error in Set-Cookie: oc_sessionPassphrase=p4KiYpoje7CsA%2BXp81dgjnINWJux2
%2F716zwzRpAZPGVXPECDan7tYVgmon5RIc3iSbTqcxeEcbLcRh1OwV6qnSnn2CkQdyXmG2rkFA8mcD3
tbpkGCh9mxs3etI1uV12r; path=/plus;; Secure at position 166.
Length: 5201 (5.1K) [text/x-python]
Saving to: 'download'

download            100%[===================>]   5.08K  --.-KB/s    in 0s

2022-06-23 11:28:12 (646 MB/s) - 'download' saved [5201/5201]
```

[ ]:
```
from FLOPs_counter import print_model_parm_flops
input = torch.randn(1, 3, 40, 40) # The input size should be the same as the
↪size that you put into your model
#Get the network and its FLOPs
num_classes = 3
model = CnnModel_batchnorm(num_classes)
model.eval()
print_model_parm_flops(model, input, detail=False)
```

```
 + Number of FLOPs: 0.86G
```