Problem4

In this problem we will be achieving large-batch SGD using batch augmentation techniques. In batch augmentation instances of samples within the same batch are generated with different data augmentations. Batch augmentation acts as a regularizer and an accelerator, increasing both generalization and performance scaling. One such augmentation scheme is using Cutout regularization, where additional samples are generated by occluding random portions of an image.

1

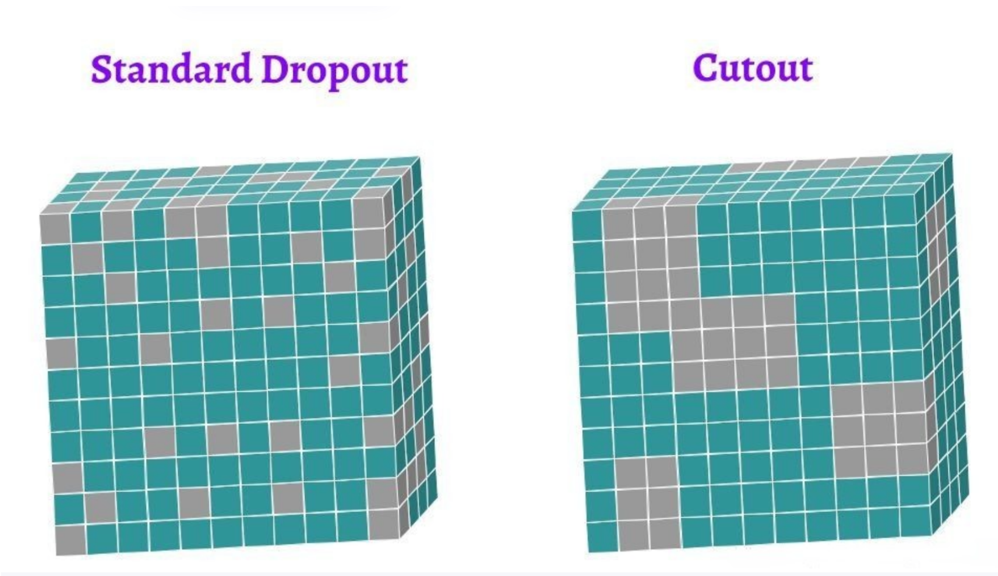
Explain cutout regularization and its advantages compared to simple dropout (as argued in the paper by DeVries et al) in your own words. Select any 2 images from CIFAR10 and show how does these images look after applying cutout. Use a square-shaped fixed size zero-mask to a random location of each image and generate its cutout version. Refer to the paper by DeVries et al (Section 3) and associated github repository. (2+4)

cutout regularization: cutout is an image augmentation and regularization technique that randomly masks out square regions of input during training. cutout regularization can be used to improve the robustness and overall performance of convolutional neural networks.

Intutively, **cutout regulartization** mask out a random region of the image, which can somehow force the model to learn a more global feature rather than rely on the local feature. Such data augmentation can make the model be robust to image noise.

simple dropout: pixel-wise dropout (the pixel can be the feature map pixel), which is more like gaussian noise style data augmentation.

We use a figure to show the difference between simple dropout and cutout regularization.



In [1]:

```
# cifar10
from torchvision.datasets import CIFAR10
import torchvision.transforms as transforms
# transform = transforms.ToTensor()
# cutoff transform

dataset = CIFAR10(root='./cached_datasets/CIFAR10', train=True, download=Tru
```

```
Files already downloaded and verified
```

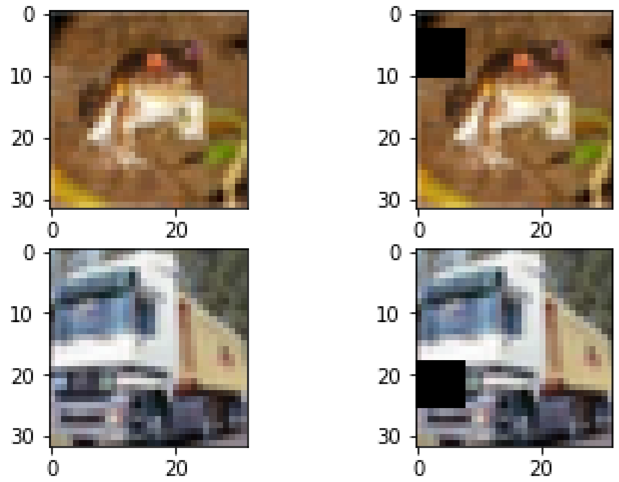
Display

In [7]:

```
import albumentations as A
import PIL
import numpy as np
import cv2

import matplotlib.pyplot as plt
transform = A.Cutout(num_holes=1, max_h_size=8, max_w_size=8, fill_value=0,

# Convert the image back to OpenCV format
# transformed_image = cv2.cvtColor(transformed_image, cv2.COLOR_RGB2BGR)
# Display the image
# plt.subplot(121)
plt.figure(figsize=(20,20))
f, axarr = plt.subplots(2,2)

img = dataset[0][0]
image_array = np.array(img)
# Augment an image
transformed = transform(image=image_array)

transformed_image = transformed["image"]
axarr[0][0].imshow(img)
axarr[0][1].imshow(transformed_image)


img = dataset[1][0]
image_array = np.array(img)
transformed = transform(image=image_array)

transformed_image = transformed["image"]

axarr[1][0].imshow(img)
axarr[1][1].imshow(transformed_image)
```

Out[7]:

```
<matplotlib.image.AxesImage at 0x7f3646bb8e20>

<Figure size 1440x1440 with 0 Axes>
```



2

Using CIFAR10 datasest and Resnet-44 we will first apply simple data augmentation as in He et al.

(look at Section 4.2 of He et al.) and train the model with batch size 64. Note that testing is always done with original images. Plot validation error vs number of training epochs. (4)

In [38]:
```
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import pytorch_lightning as pl
import torch
from models.resnet44 import resnet44
import torchvision.datasets as datasets
import torchmetrics


class ResNetLightningModule(pl.LightningModule):
    def __init__(self, batch_size=32 ,optimizer_name='SGD', aug_method="simp
        super(ResNetLightningModule, self).__init__()
        self.model = resnet44()
        self.loss_fn = nn.CrossEntropyLoss()
        self.optimizer_name = optimizer_name
        self.batch_size = batch_size
        self.aug_method = aug_method
        self.acc_metric = torchmetrics.Accuracy()

    def forward(self, x):
        return self.model(x)

    def training_step(self, batch, batch_idx):
        x, y = batch
        logit = self.model(x)
        loss = self.loss_fn(logit, y)
        train_acc = self.acc_metric(logit.argmax(dim=-1), y)
        self.log('train_acc', train_acc, prog_bar=False, on_epoch=True)
        self.log('train_loss', loss, prog_bar=True, on_epoch=True)
        logs = {'train_loss': loss}

        return {'loss': loss, 'log': logs}

    def train_dataloader(self):
```

```
        if self.aug_method == "simple_aug":
            train_transform = transforms.Compose([# 4 pixels are padded on e
                                                  transforms.Pad(4),
                                                  # a 32×32 crop is randomly s
                                                  # padded image or its horizo
                                                  transforms.RandomHorizontalF
                                                  transforms.RandomCrop(32),
                                                  transforms.ToTensor()
                                              ])
        elif self.aug_method == "cutout":
            train_transform = transforms.Compose([transforms.RandomCrop(32,
                                          transforms.RandomHorizontalFlip(0.5)
                                          transforms.ToTensor(), #convert to t
                                          transforms.Normalize((0.4914, 0.4822
                                          transform])
        else:
            raise ValueError("Invalid augmentation method")
        train_dataset = datasets.CIFAR10(root='./cached_datasets/CIFAR10', t
        return torch.utils.data.DataLoader(train_dataset, batch_size=self.ba

    def test_dataloader(self):
        test_transform = transforms.Compose([
            transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994,
            transforms.ToTensor(),])
        test_dataset = datasets.CIFAR10(root='./cached_datasets/CIFAR10', tr

        return torch.utils.data.DataLoader(test_dataset, batch_size=self.bat

    def validation_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self.model(x)
        loss = self.loss_fn(y_hat, y)
        val_acc = self.acc_metric(y_hat.argmax(dim=-1), y)
        self.log('val_acc', val_acc, prog_bar=False, on_epoch=True)
        val_error = 1 - val_acc
        self.log('val_error', val_error, prog_bar=False, on_epoch=True)

        return {'val_loss': loss}


    def configure_optimizers(self):
        if self.optimizer_name == 'SGD':
            lr = 0.1 # authors cite 0.1
            momentum = 0.9
            weight_decay = 0.0001
            # Use SGD optimizers with 0.1 as the learning rate, momentum 0.9
            optimizer = torch.optim.SGD(self.parameters(), lr=lr, weight_dec
        return optimizer
```

In [39]:
```
lightning_module = ResNetLightningModule(batch_size=64, optimizer_name='SGD'
```

We use our implementation of Resnet-44 to train the model and tried the m=0 in the given code, the result do not have much difference.

python main.py --dataset cifar10 --model resnet --model-config "{'depth': 44}" -b 64 --epochs 100 --save resnet44_simple_aug python main.py --dataset cifar10 --model resnet --model-config "{'depth': 44}" --duplicates 2 --cutout -b 64 --epochs 100 --save resnet44_cutout_m-2 python main.py --dataset cifar10 --model resnet --model-config "{'depth': 44}" --duplicates 4 --cutout -b 64 --epochs 100 --save resnet44_cutout_m-4 python main.py --dataset cifar10 --model resnet --model-config "{'depth': 44}" --duplicates 8 --cutout -b 64 --epochs 100 --save resnet44_cutout_m-8 python main.py --dataset cifar10 --model resnet --model-config "{'depth': 44}" --duplicates 16 --cutout -b 64 --epochs 100 --save resnet44_cutout_m-16 python main.py --dataset cifar10 --model resnet --model-config "{'depth': 44}" --duplicates 32 --cutout -b 64 --epochs 100 --save resnet44_cutout_m-32

Read Results

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
data = pd.read_csv("./problem4/results/resnet44_simple_aug/results.csv")
data
```

Out[2]:

	epoch	steps	training step	training data	training loss	training prec1	training prec5	training error1	tra e
0	1	195	0.056163	0.005818	1.787805	31.957131	84.615385	68.042869	15.38
1	2	390	0.050171	0.004936	1.294457	52.489984	94.330929	47.510016	5.6
2	3	585	0.050089	0.005094	1.017389	63.501603	96.710737	36.498397	3.28
3	4	780	0.049726	0.004855	0.867839	69.124599	97.696314	30.875401	2.30
4	5	975	0.050032	0.004965	0.773381	72.662260	98.106971	27.337740	1.89
...	...	...	...	...	...	...	...	...	
95	96	18720	0.048046	0.004495	0.016185	99.543269	100.000000	0.456731	0.00
96	97	18915	0.048371	0.004690	0.014763	99.581330	100.000000	0.418670	0.00
97	98	19110	0.047950	0.004401	0.013587	99.623397	99.997997	0.376603	0.00
98	99	19305	0.049305	0.005098	0.013661	99.629407	99.997997	0.370593	0.00
99	100	19500	0.048916	0.004973	0.013038	99.641426	100.000000	0.358574	0.00

100 rows × 16 columns

In [11]:
```
import time
import datetime
import re
import pandas as pd
def read_log(name):
    log_file = open(f"./problem4/results/{name}/log.txt", "r")
    first = True
    flag = False
    res_list = []
    # cur_time = None
    for line in log_file.readlines():
        # print(line)
        if "- INFO - TRAINING" in line and first:
            # print(line)
            start_time = line.split("- INFO - TRAINING - ")[0].split(" -")[0
            start_time = datetime.datetime.strptime(start_time, "%Y-%m-%d %H
            # print(start_time)
            first = False
        if "- INFO - TRAINING" in line:
            cur_time = line.split("- INFO - VALIDATION - ")[0].split(" -")[0
            cur_time = datetime.datetime.strptime(cur_time, "%Y-%m-%d %H:%M:
        if "Results - Epoch" in line:
            # print(line)
            flag = True
            cur_epoch = int(re.findall(r"\d+", line)[0])
            # print(cur_epoch)
```

```
            continue
        if flag:
            # print(cur_time, cur_epoch, line)
            res_list.append([cur_time, cur_epoch, line])
            flag = False
    log_file.close()
    res_list = np.array(res_list)
    res_list
    time_list = res_list[:, 0] - start_time
    time_list = [i.total_seconds() for i in time_list]
    return time_list

def get_epoch_from_results(m=0):
    if m == 0:
        name = "resnet44_simple_aug"
    else:
        name = f"resnet44_cutout_m-{m}"
    data = pd.read_csv(f"./problem4/results/{name}/results.csv")
    if len(data[data["validation error1"]<=6]) > 0:
        epoch = data[data["validation error1"]<=6]["epoch"].values[0]
    else:
        epoch = data["epoch"].max()
    epoch_error = data[data["epoch"]==epoch]["validation error1"].values[0]
    # print(epoch)
    time_list = read_log(name)
    # print(time_list, epoch)
    # print(epoch, time_list[epoch-1])
    get_94_time = time_list[epoch-1]
    return get_94_time, epoch, data["validation error1"].to_list(), epoch_er
m_list = [0, 2, 4, 8, 16, 32]
val_err_list = []
df = pd.DataFrame(columns=["m", "wlltime", "epoch", "val_error"])
for m in m_list:
    get_94_time, epoch, val_ser, epoch_error = get_epoch_from_results(m)
    df.loc[len(df)] = [m, get_94_time, epoch, epoch_error]
    val_err_list.append(val_ser)
```

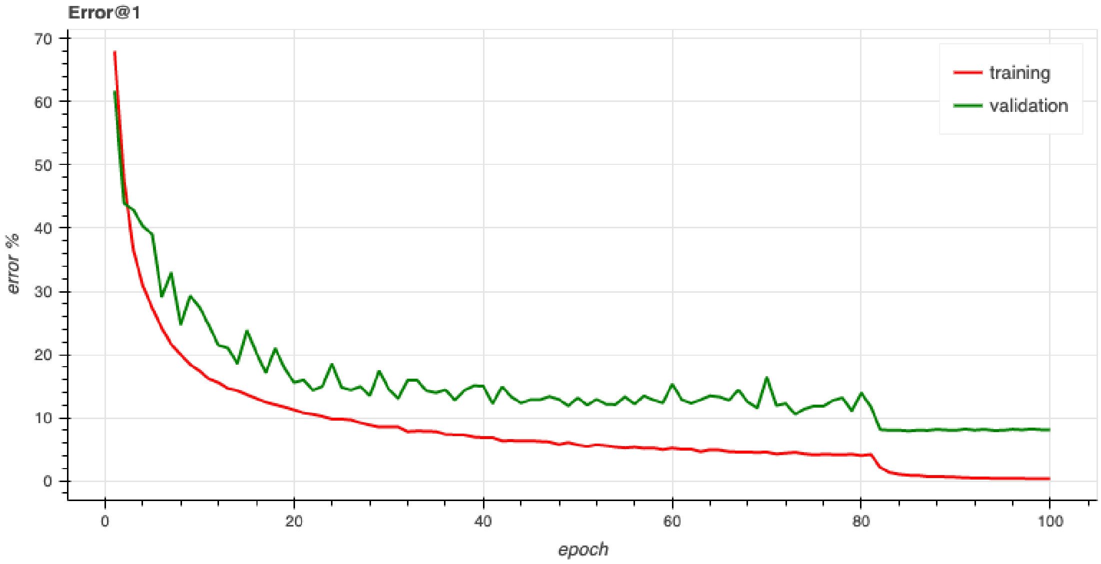
We get the walltime for achieving the 0.94 validation accuracy.

In [12]:
```
df
```

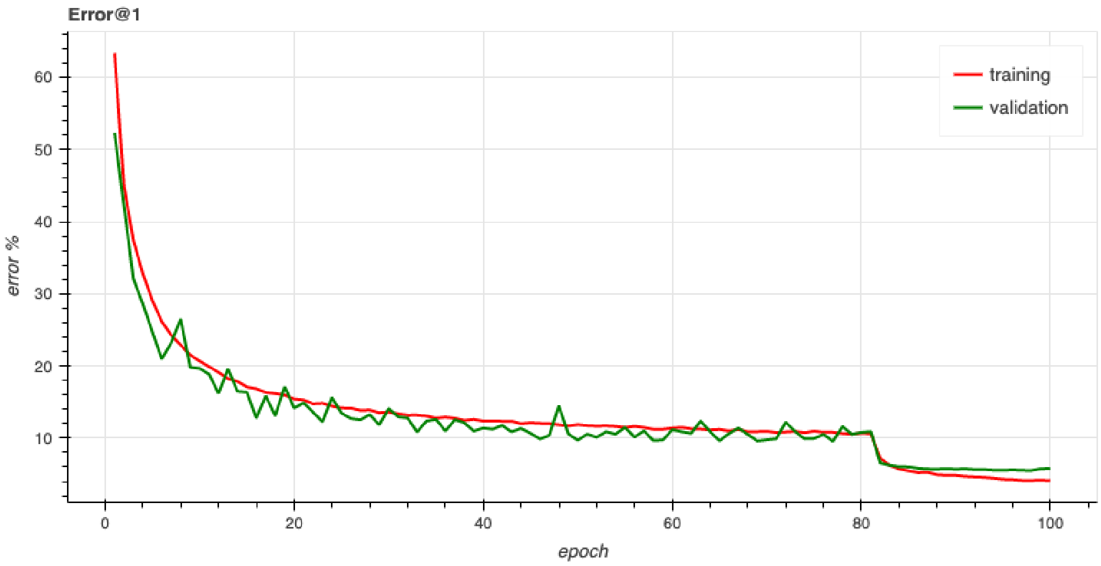
Out[12]:

	m	wlltime	epoch	val_error
0	0.0	1147.0	100.0	8.09
1	2.0	2962.0	86.0	5.82
2	4.0	3406.0	82.0	5.29
3	8.0	5187.0	82.0	5.29
4	16.0	9277.0	82.0	5.17
5	32.0	17874.0	82.0	5.07

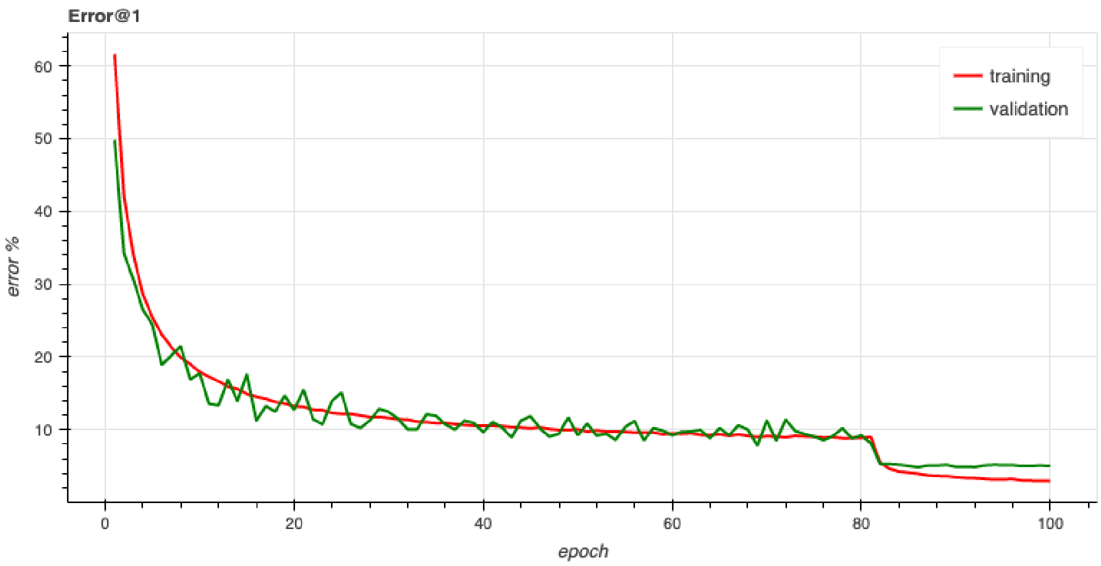
simple_aug (m=0)


Error@1
70
60
50
40
30
20
10
0
error %
0
20
40
60
80
100
epoch
training
validation

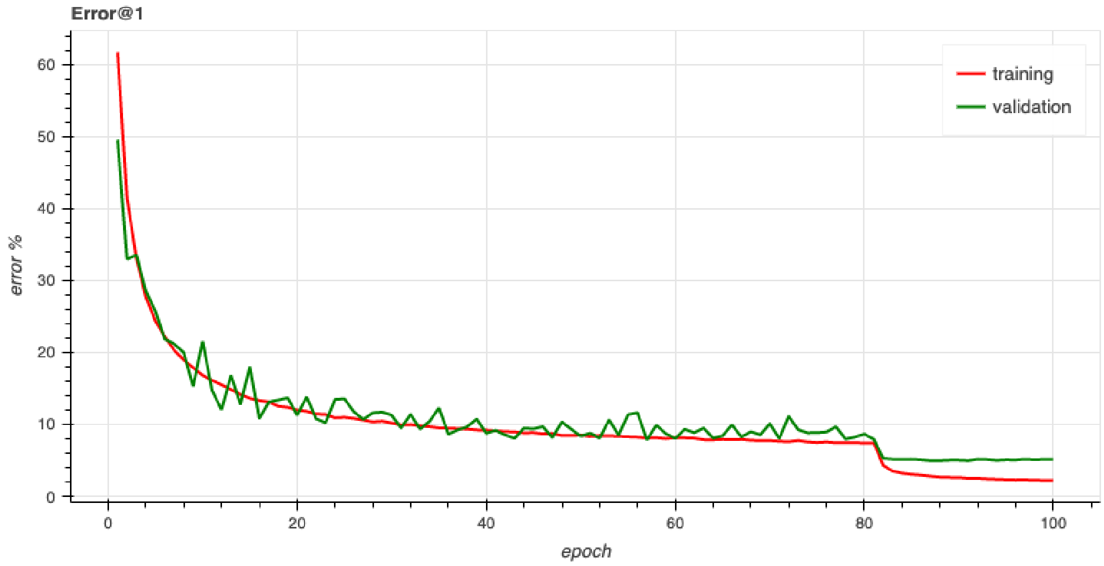
m=2


Error@1
60
50
40
30
20
10
error %
0
20
40
60
80
100
epoch
training
validation

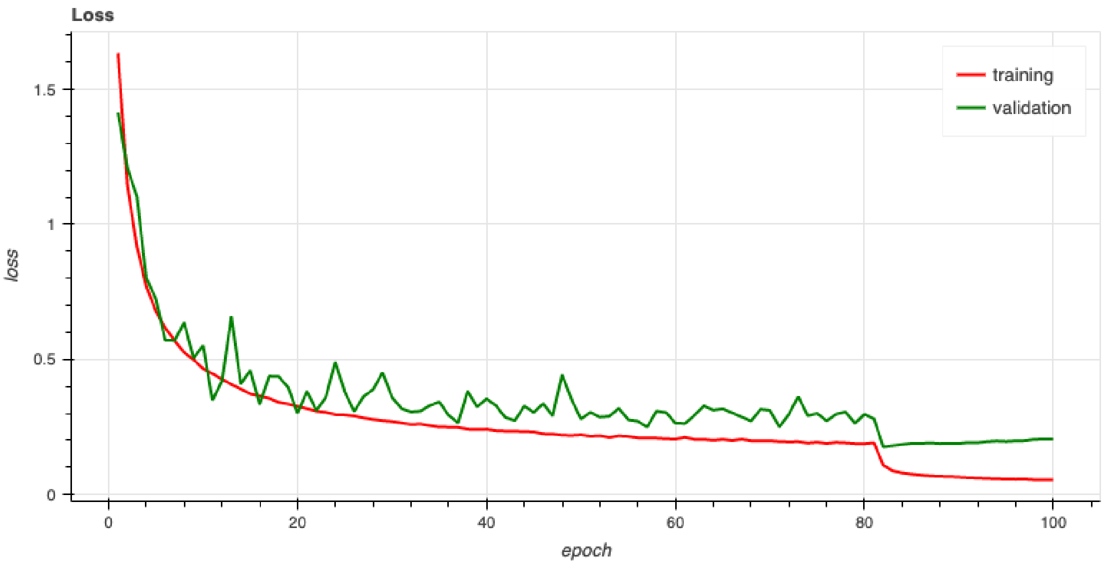
m=4


Error@1
60
50
40
30
20
10
error %
0
20
40
60
80
100
epoch
training
validation

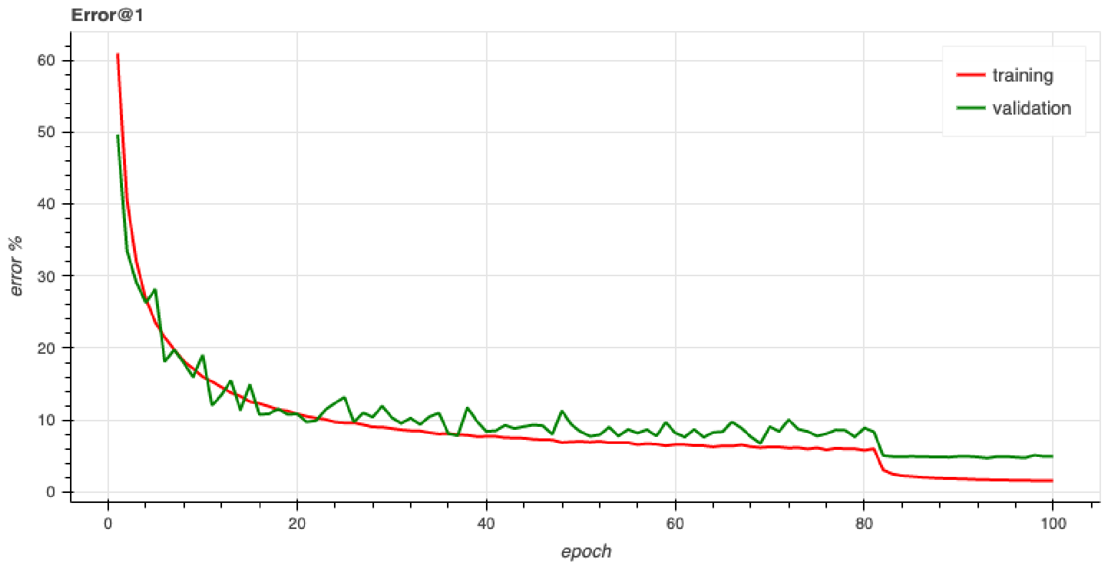
m=8


Error@1
training
validation
error %
epoch
0
10
20
30
40
50
60
0
20
40
60
80
100

m=16


Loss
training
validation
loss
epoch
0
0.5
1
1.5
0
20
40
60
80
100

m=32

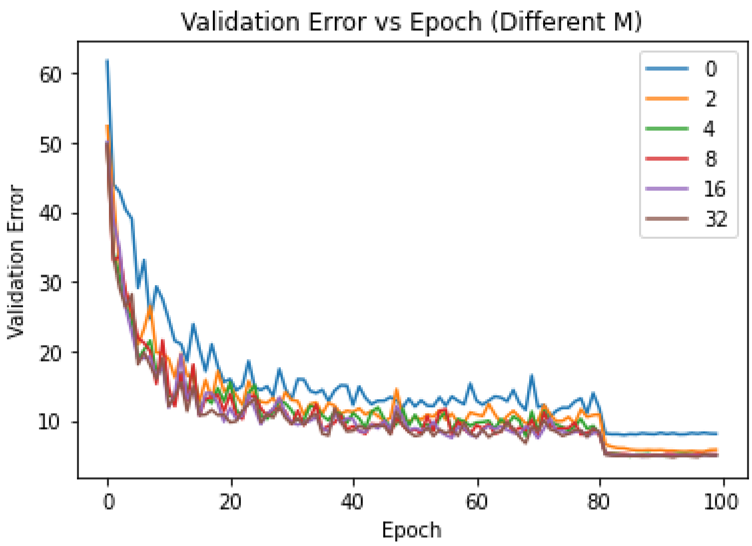

Error@1
training
validation
error %
epoch
0
10
20
30
40
50
60
0
20
40
60
80
100

Plot Together

In [13]:
```
val_err_list  = np.array(val_err_list)
```

In [14]:
```
# val_err_list = np.array(val_err_list)
import matplotlib.pyplot as plt
for i in range(len(m_list)):
    plt.plot(val_err_list[i])
plt.legend(m_list)
plt.title("Validation Error vs Epoch (Different M)")
plt.xlabel("Epoch")
plt.ylabel("Validation Error")
```

Out[14]:
```
Text(0, 0.5, 'Validation Error')
```



m = 0 stands for single augumentation

Comment: m=2 can achieve the 94% validation accuracy with least wall time 2962.0 seconds, if we compare the final accuracy, m=16 can achieve simlar result with m=32, but relatively shorter training time.