Image Completion based on Super-resolution

Project mentor: Guanghui Qin

Yishun Zhou yzhou165@jh.edu, Weina Dai wdai11@jh.edu, Lei Shi lshi23@jh.edu, Lingyun Di ldi4@jh.edu

Code Repo:
https://drive.google.com/drive/folders/1J4SWRebHpBVmIgy_GHFSty7wXzq7fTTR?usp=sharing

Outline and Deliverables

Uncompleted Deliverables

1. To bring color to a black and white photography: We found out that this is essetially a different goal and task than super resolution and requires different models.
2. To remove blurs and cracks present in a scan of a photograph: This also requires a model than ours.
3. create an new algorithm for image completion: Image completion is a much more challenging problem than image super resolution.

Completed Deliverables

1. To bring low resolution images to high resolution on train images: We were able to perform super resolution on x4 and x8 downscaled images.
2. To bring low resolution images to high resolution on any images: Based on our model and testing result, we were able to perform super resolution on any x4 and x8 downscaled images.
3. To improve the SRCNN algorithm: We are able to find a faster and better algorithm and SRCNN called VDSR.

Preliminaries

What problem were you trying to solve or understand?

Our project aims to implement an machine learning algorithm that can output a high-resolution image after inputting a low-resolution image. In other words, we wish to come up with a model Y = F(X), where X is the low resolution image and Y is the predicted high resolution image.

How is this problem similar to others we've seen in lectures, breakouts, and homeworks?

This problem is a great example of supervised learning algorithm that is very similar with the homework 2's programming part, the algorithm takes many images as an input to train a model to label them, using stachostic gradient descent to minimize the MSE loss function. Instead of output labels, this project we will output an single image after processed by super-resolution algorithm based on an significant part introduced on the course: the Convolutional Neural Network

What makes this problem unique?

From medical image analysis to old photo restoration, many domains are seeking a higher resolution of images while HR images are not always available caused of inherent limitations or expensive devices. Therefore, the desire of converting a Low-Resolution(LR) image into a High-Resolution(HR) image has long been an attractive area in the Computer Vision field.

What ethical implications does this problem have?

The super resolution based on CNN is no more than predicting the color values of different pixels and making the image look sharper by minimizing the objective function. The images we use in our dataset is publicly available. Since super resolution model generates new information. Then the super resolution result that our model predicts should be probably noted that they are brought to a higher resolution by machine learning algorithm in case of any ethical problems.

Dataset(s)

We are using DIV2K, which is a dataset consisting of 1000 sets of RGB images with various scenarios. Each set contains a high-resolution image and many low-resolution images with different downscale factors 2,3,4 and 8. The dataset is divided into three data sections: 800 for training, 100 for validating, and 100 for testing. We chose it since it is specifically collected for NTIRE2017 and NTIRE2018 Super-Resolution Challenges, which is consistent with the goal of our project: Image completion based on Super-resolution.

However, due to time constraint and size constraint, we have to upload all the data to google drive and run code using google colab GPU. So we reduced the training set to 50 images, the validating set to 10 images and the testing set to 10 images.

Below we plotted 2 image examples from training, validation and test set each. Each image has different scales: the original size, the one downscaled by 4 and the one downscaled by 8. Note that they are of different sizes. The width and height in number of pixels are shown in the axes.

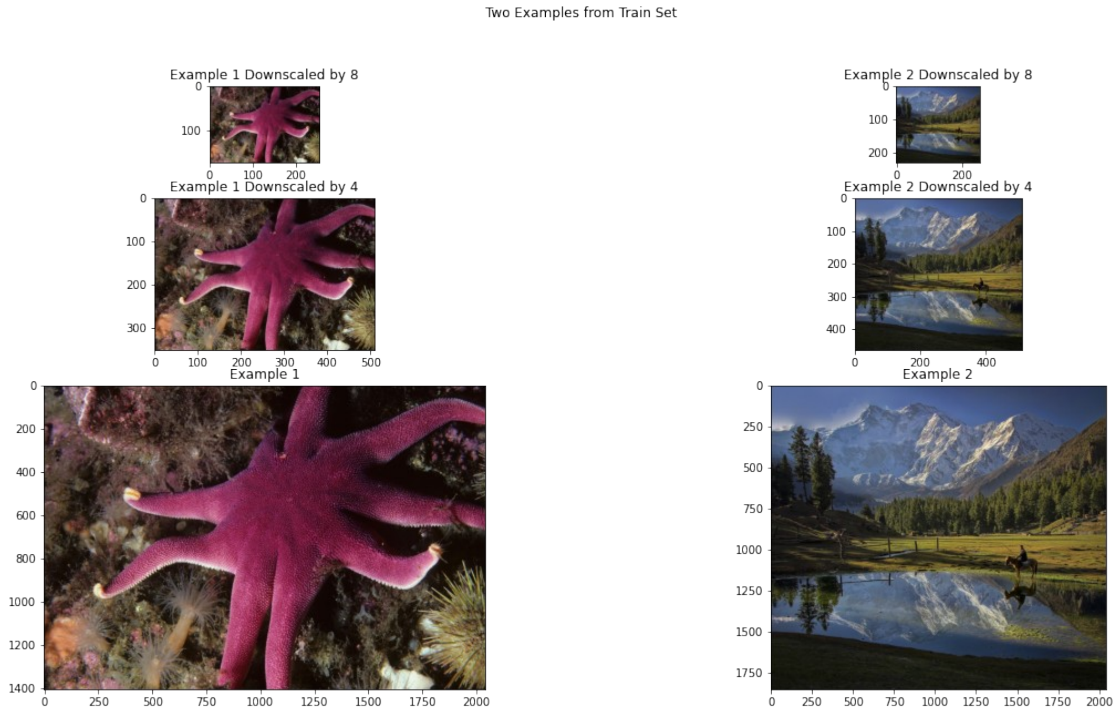
```
from google.colab import drive
drive.mount('/content/drive/')
```

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

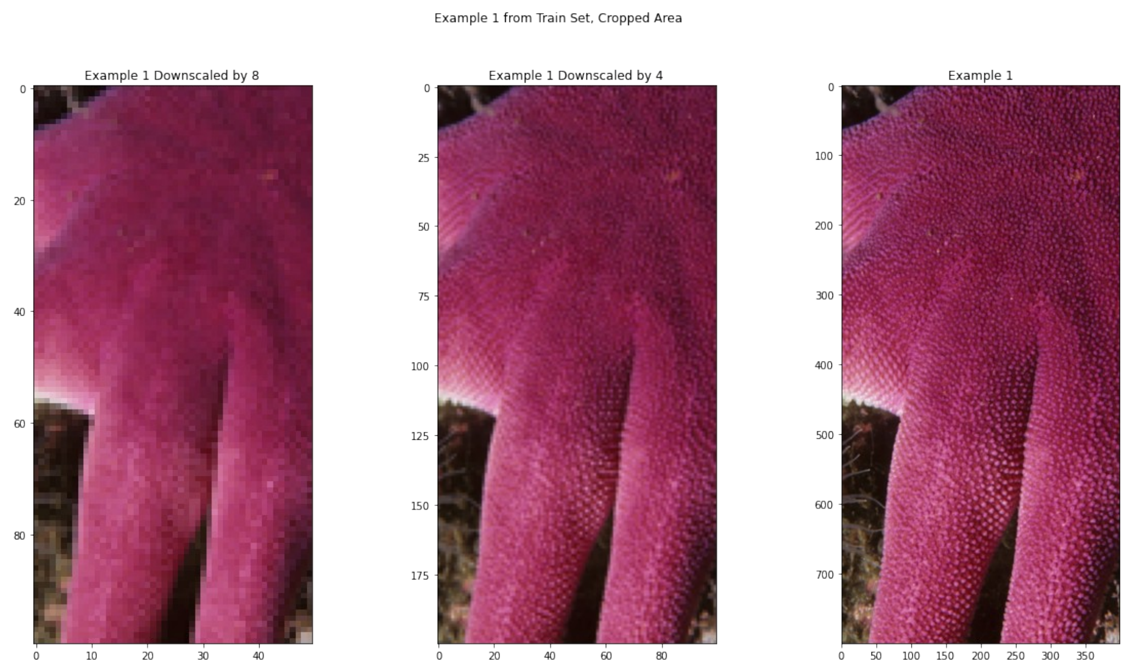
```
# import modules
import os
os.chdir('/content/drive/My Drive/machine-learning-VDSR/')
```

```
import sys
import csv
import numpy as np
import datetime
import torch
import torch.nn.functional as F
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from torch.autograd import Variable
from PIL import Image
from itertools import product
import copy
from math import *
import time
import random
import math

DATA_DIR = "dataset"

# training dataset
HR_IMAGES_TRAINING = DATA_DIR+ "/DIV2K_train_HR"
LR_IMAGES_TRAINING = DATA_DIR+ "/DIV2K_train_LR_bicubic"
images_hr = os.listdir(HR_IMAGES_TRAINING)
img1_hr = Image.open(HR_IMAGES_TRAINING+'/0001.png').convert('RGB')
img2_hr = Image.open(HR_IMAGES_TRAINING+'/0002.png').convert('RGB')
images_lr = os.listdir(LR_IMAGES_TRAINING)
img1_lr_x8 =
Image.open(LR_IMAGES_TRAINING+'/X8/0001x8.png').convert('RGB')
img2_lr_x8 =
Image.open(LR_IMAGES_TRAINING+'/X8/0002x8.png').convert('RGB')
img1_lr_x4 =
Image.open(LR_IMAGES_TRAINING+'/X4/0001x4.png').convert('RGB')
img2_lr_x4 =
Image.open(LR_IMAGES_TRAINING+'/X4/0002x4.png').convert('RGB')
# 2 examples from training
f, axarr = plt.subplots(3,2,gridspec_kw={'height_ratios': [1, 2, 4]})
axarr[0, 0].imshow(img1_lr_x8)
axarr[0, 0].set_title('Example 1 Downscaled by 8')
axarr[0, 1].imshow(img2_lr_x8)
axarr[0, 1].set_title('Example 2 Downscaled by 8')
axarr[1, 0].imshow(img1_lr_x4)
axarr[1, 0].set_title('Example 1 Downscaled by 4')
axarr[1, 1].imshow(img2_lr_x4)
axarr[1, 1].set_title('Example 2 Downscaled by 4')
axarr[2, 0].imshow(img1_hr)
axarr[2, 0].set_title('Example 1')
axarr[2, 1].imshow(img2_hr)
axarr[2, 1].set_title('Example 2')
```

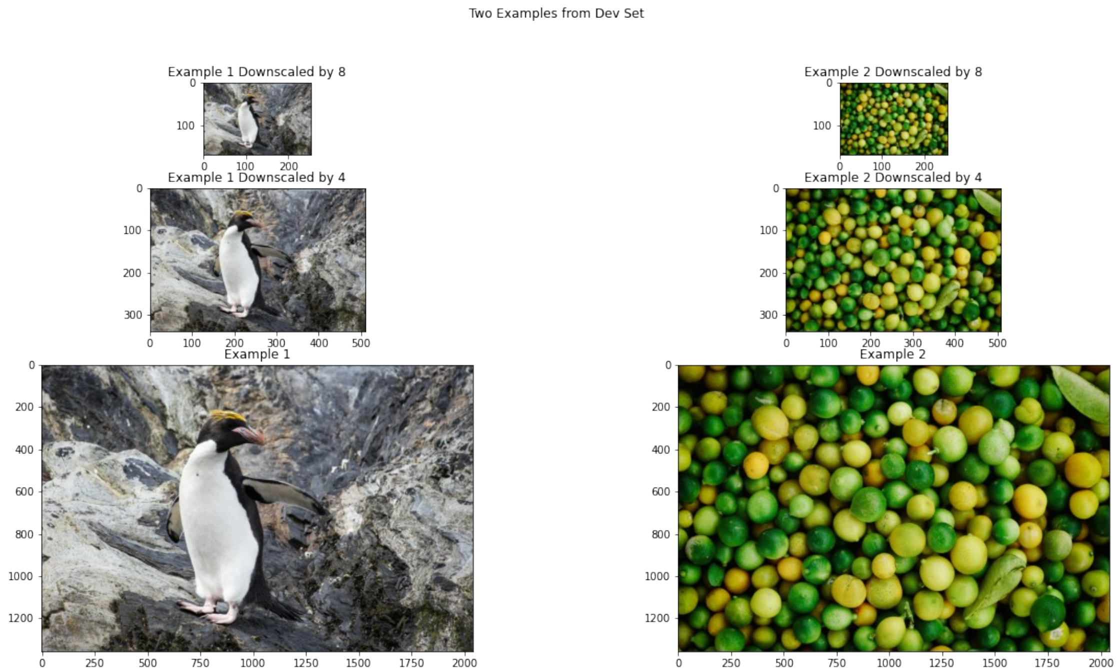
```
f.suptitle('Two Examples from Train Set')

plt.show()
plt.rcParams['figure.figsize'] = [20, 10]
```



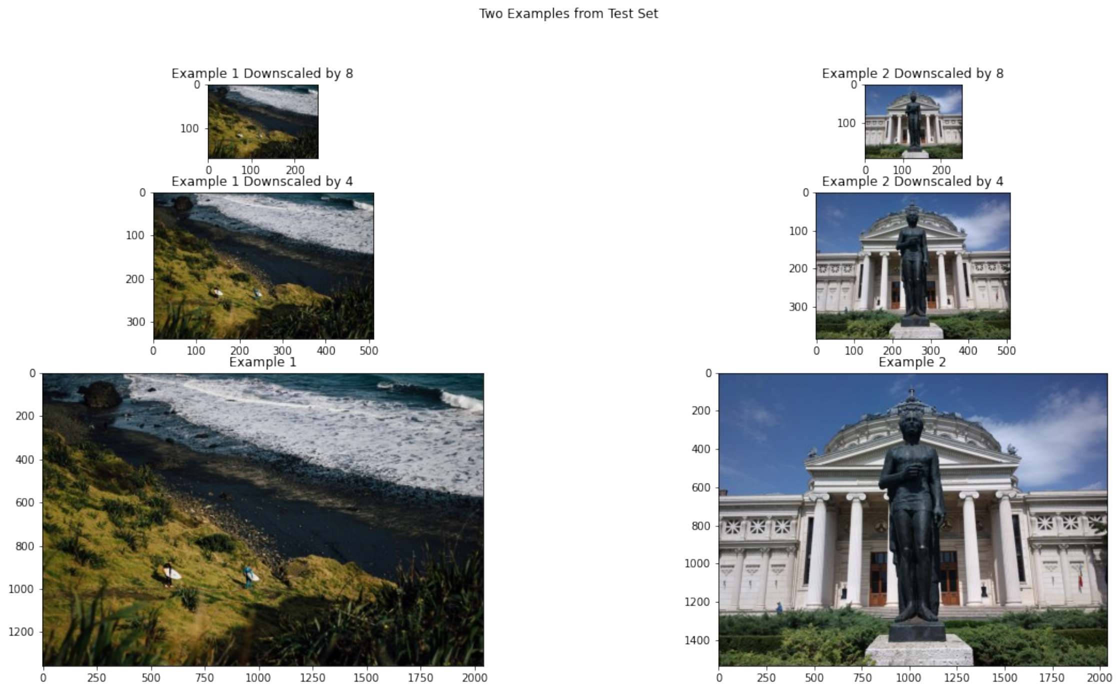

```
f, axarr = plt.subplots(1,3)
axarr[0].imshow(img1_lr_x8.crop((100,50,150,150)))
axarr[0].set_title('Example 1 Downscaled by 8')
axarr[1].imshow(img1_lr_x4.crop((200,100,300,300)))
axarr[1].set_title('Example 1 Downscaled by 4')
axarr[2].imshow(img1_hr.crop((800,400,1200,1200)))
axarr[2].set_title('Example 1')
f.suptitle('Example 1 from Train Set, Cropped Area')

plt.show()
plt.rcParams['figure.figsize'] = [20, 10]
```




```python
# dev dataset
HR_IMAGES_DEV = DATA_DIR+ "/DIV2K_dev_HR"
LR_IMAGES_DEV = DATA_DIR+ "/DIV2K_dev_LR_bicubic"
# 2 examples from dev
images_hr = os.listdir(HR_IMAGES_DEV)
img1_hr = Image.open(HR_IMAGES_DEV+'/0801.png').convert('RGB')
img2_hr = Image.open(HR_IMAGES_DEV+'/0802.png').convert('RGB')
images_lr = os.listdir(LR_IMAGES_DEV)
img1_lr_x8 = Image.open(LR_IMAGES_DEV+'/X8/0801x8.png').convert('RGB')
img2_lr_x8 = Image.open(LR_IMAGES_DEV+'/X8/0802x8.png').convert('RGB')
img1_lr_x4 = Image.open(LR_IMAGES_DEV+'/X4/0801x4.png').convert('RGB')
img2_lr_x4 = Image.open(LR_IMAGES_DEV+'/X4/0802x4.png').convert('RGB')

f, axarr = plt.subplots(3,2,gridspec_kw={'height_ratios': [1, 2, 4]})
axarr[0, 0].imshow(img1_lr_x8)
axarr[0, 0].set_title('Example 1 Downscaled by 8')
axarr[0, 1].imshow(img2_lr_x8)
axarr[0, 1].set_title('Example 2 Downscaled by 8')
axarr[1, 0].imshow(img1_lr_x4)
axarr[1, 0].set_title('Example 1 Downscaled by 4')
axarr[1, 1].imshow(img2_lr_x4)
axarr[1, 1].set_title('Example 2 Downscaled by 4')
axarr[2, 0].imshow(img1_hr)
axarr[2, 0].set_title('Example 1')
axarr[2, 1].imshow(img2_hr)
axarr[2, 1].set_title('Example 2')
f.suptitle('Two Examples from Dev Set')
plt.show()
plt.rcParams['figure.figsize'] = [20, 10]
```




```python
# test dataset
HR_IMAGES_DEV = DATA_DIR+ "/DIV2K_test_HR"
LR_IMAGES_DEV = DATA_DIR+ "/DIV2K_test_LR_bicubic"
# 2 examples from test
images_hr = os.listdir(HR_IMAGES_DEV)
img1_hr = Image.open(HR_IMAGES_DEV+'/0811.png').convert('RGB')
img2_hr = Image.open(HR_IMAGES_DEV+'/0812.png').convert('RGB')
images_lr = os.listdir(LR_IMAGES_DEV)
img1_lr_x8 = Image.open(LR_IMAGES_DEV+'/X8/0811x8.png').convert('RGB')
img2_lr_x8 = Image.open(LR_IMAGES_DEV+'/X8/0812x8.png').convert('RGB')
img1_lr_x4 = Image.open(LR_IMAGES_DEV+'/X4/0811x4.png').convert('RGB')
img2_lr_x4 = Image.open(LR_IMAGES_DEV+'/X4/0812x4.png').convert('RGB')

f, axarr = plt.subplots(3,2,gridspec_kw={'height_ratios': [1, 2, 4]})
axarr[0, 0].imshow(img1_lr_x8)
axarr[0, 0].set_title('Example 1 Downscaled by 8')
axarr[0, 1].imshow(img2_lr_x8)
axarr[0, 1].set_title('Example 2 Downscaled by 8')
axarr[1, 0].imshow(img1_lr_x4)
axarr[1, 0].set_title('Example 1 Downscaled by 4')
axarr[1, 1].imshow(img2_lr_x4)
axarr[1, 1].set_title('Example 2 Downscaled by 4')
axarr[2, 0].imshow(img1_hr)
axarr[2, 0].set_title('Example 1')
axarr[2, 1].imshow(img2_hr)
axarr[2, 1].set_title('Example 2')
f.suptitle('Two Examples from Test Set')
plt.show()
plt.rcParams['figure.figsize'] = [20, 10]
```



Pre-processing

We pre-processed our images in Matlab for a shorter runtime. We also wrote the image files to hdf5 to compress size of the data. The low resolution images were resized using bicubic interpolation method within Matlab so that they are the same size as the high resolution images. Then we break each image into patches of 41x41 pixels. The border left after patching is discarded from the pre-processed data. So the input data for training is composed of an array of low resolution image patches with bicubic interpolation and an corresponding array of high resolution image patches of the same size.

In terms of selecting the features, the RGB image was first converted to YCbCr format. And we found out that the Cb and Cr channels of a high resolution image and its correpsonding low resolution image are similar, which agrees with the implementation of VDSR from the paper we referenced. So we only need to use the Y channel for training the model. We then normalized all the channels so that the maximum of the image array is 1 and the minimum of the image array is 0.

We didn't do image data augmentation such as rescaling or rotating the image patches since we have a large dataset. It would also take a long time for us to upload a large augemented dataset to google drive and to train the model.

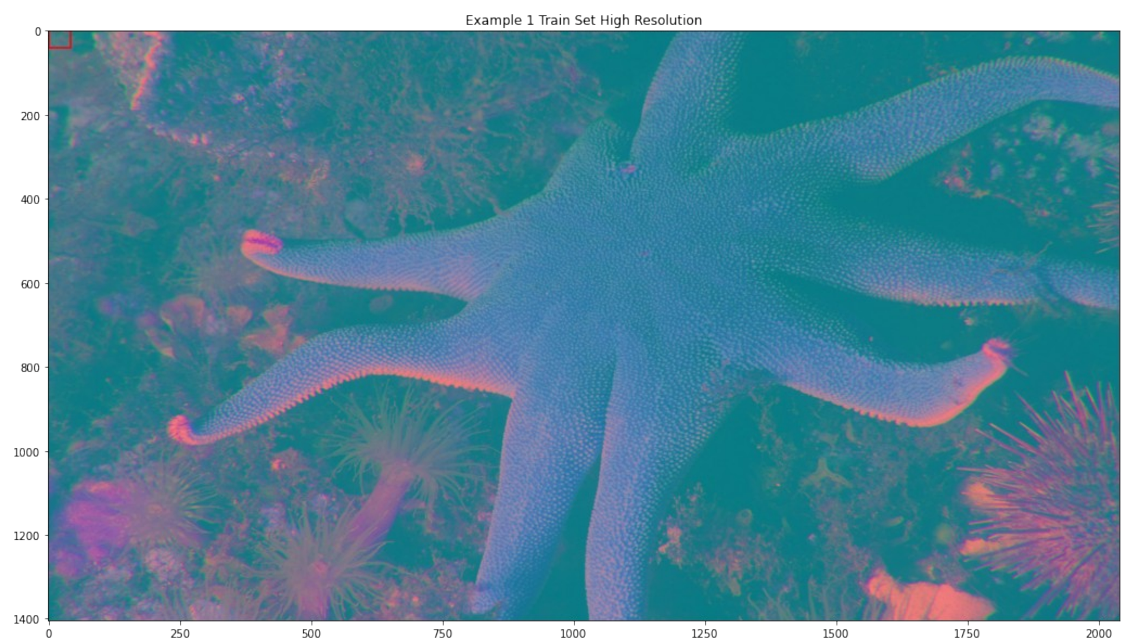
```
# loading pre-processed data
# downscaled by 8
import h5py

hf_x8 = h5py.File("dataset/train_x8.h5")
hf_x8_dev = h5py.File("dataset/dev_x8.h5")
```

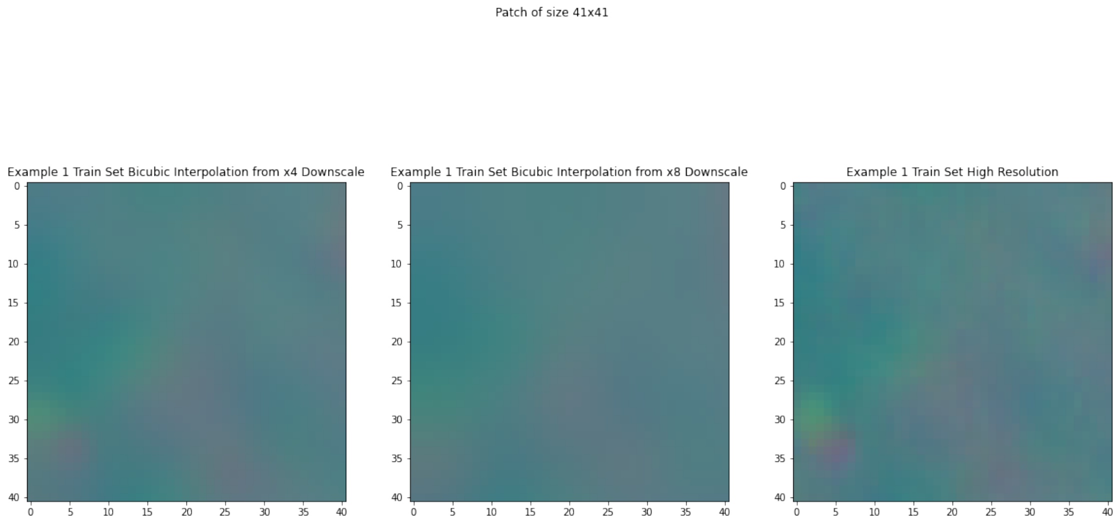
```
LR_x8 = hf_x8.get('lr')
HR_x8 = hf_x8.get('hr')
LR_x8 = torch.from_numpy(LR_x8[:]).float()
HR_x8 = torch.from_numpy(HR_x8[:]).float()

LR_x8_dev = hf_x8.get('lr')
HR_x8_dev = hf_x8.get('hr')
LR_x8_dev = torch.from_numpy(LR_x8_dev[:]).float()
HR_x8_dev = torch.from_numpy(HR_x8_dev[:]).float()

# downscaled by 4
import h5py
hf_x4 = h5py.File("dataset/train_x4.h5")
hf_x4_dev = h5py.File("dataset/dev_x4.h5")

LR_x4 = hf_x4.get('lr')
HR_x4 = hf_x4.get('hr')
LR_x4 = torch.from_numpy(LR_x4[:]).float()
HR_x4 = torch.from_numpy(HR_x4[:]).float()

LR_x4_dev = hf_x4.get('lr')
HR_x4_dev = hf_x4.get('hr')
LR_x4_dev = torch.from_numpy(LR_x4_dev[:]).float()
HR_x4_dev = torch.from_numpy(HR_x4_dev[:]).float()
```

In the following cell, an example from the train set is shown in the YCbCr form. The red box on the top left corner is one of the patch among the patches that we broke the image into. The patches corresponding to the red cropped area are shown below also. These patches are cropped from x8, x4 downscaled images and the original high resolution image respectively and bicubic interpolated to the same size as the original high resolution image patch (41x41 pixles).

```
# Visualize the distribution of your data before and after pre-
processing.
# You may borrow from how we visualized data in the Lab homeworks.

import matplotlib.patches as patches
# conver to RGB
x_img1_x4=LR_x4[0,:,:,:]
x_img1_x4=x_img1_x4.cpu().detach().numpy()
y_img1_x4=HR_x4[0,:,:,:]
y_img1_x4=y_img1_x4.cpu().detach().numpy()
x_img1_x4 = np.moveaxis(x_img1_x4, 0, -1)
y_img1_x4 = np.moveaxis(y_img1_x4, 0, -1)

x_img1_x8=LR_x8[0,:,:,:]
x_img1_x8=x_img1_x8.cpu().detach().numpy()
y_img1_x8=HR_x8[0,:,:,:]
y_img1_x8=y_img1_x8.cpu().detach().numpy()
x_img1_x8 = np.moveaxis(x_img1_x8, 0, -1)
```

```
y_img1_x8 = np.moveaxis(y_img1_x8, 0, -1)

# example 1 train
f, axarr = plt.subplots(1,1)
# Create a Rectangle patch
rect = patches.Rectangle((0, 0), 41, 41, linewidth=1.5, edgecolor='r',
facecolor='none')
axarr.add_patch(rect)
axarr.imshow(np.array(img1_hr.convert("YCbCr")),aspect=0.8)
axarr.set_title('Example 1 Train Set High Resolution')

f, axarr = plt.subplots(1,3)
axarr[0].imshow(x_img1_x4,aspect=1)
axarr[0].set_title('Example 1 Train Set Bicubic Interpolation from x4
Downscale')
axarr[1].imshow(x_img1_x8,aspect=1)
axarr[1].set_title('Example 1 Train Set Bicubic Interpolation from x8
Downscale')
axarr[2].imshow(y_img1_x4,aspect=1)
axarr[2].set_title('Example 1 Train Set High Resolution')
f.suptitle('Patch of size 41x41')


plt.show()
# plt.rcParams['figure.figsize'] = [30, 10]
```





Models and Evaluation

Experimental Setup

We evaluated our methods using two metrics, the mean squared error (MSE) between high resolution image and the predicted high resolution image and the peak signal to noise ratio (PSNR). The closer the predicted image is to the ground truth high resolution image, the samller the MSE and the better the prediction. PSNR is a common measure of the quality of reconstruction of compression, which approximates human perceptionof the reconstructed image. The better the reconstruction, the higher the PSNR. The formula for PSNR is shown below.

$$\begin{aligned} PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\ &= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\ &= 20 \cdot \log_{10}\left(MAX_I\right) - 10 \cdot \log_{10}\left(MSE\right) \end{aligned}$$

We used MSE for loss function to train our model. We didn't try other loss functions since MSE is representative of the difference between predicted high resolution image and the ground truth high resolution image. PSNR also has MSE in its equation.

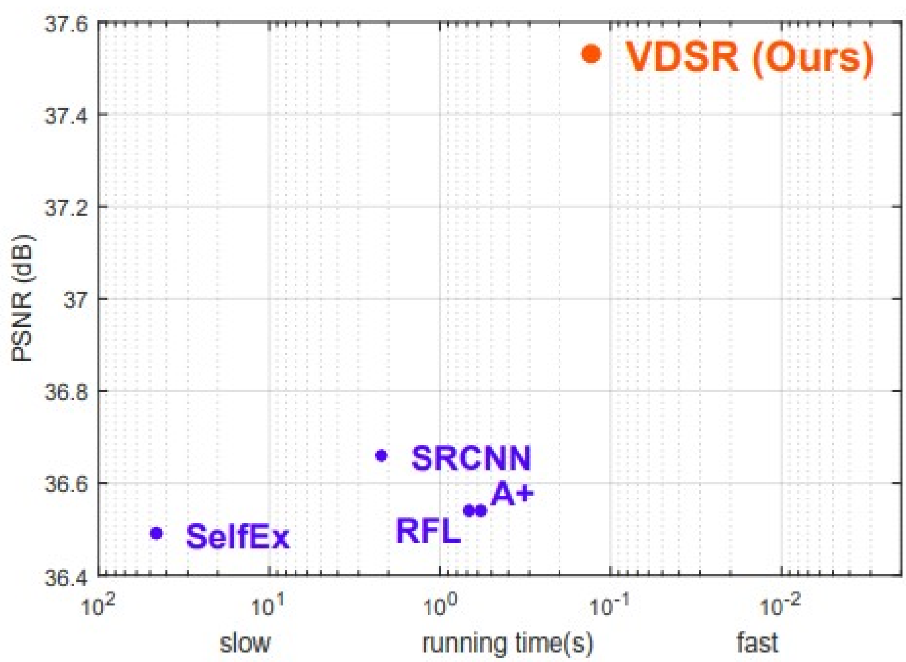
```
# MSE
loss_func = torch.nn.MSELoss(reduction ='sum')

# PSNR
def PSNR(prediction, hr):
    mse = math.sqrt(np.mean((prediction - hr) ** 2))
    if mse == 0:
```

```
        return np.Inf
    return 20 * math.log10(255.0 / mse)
```

Baselines

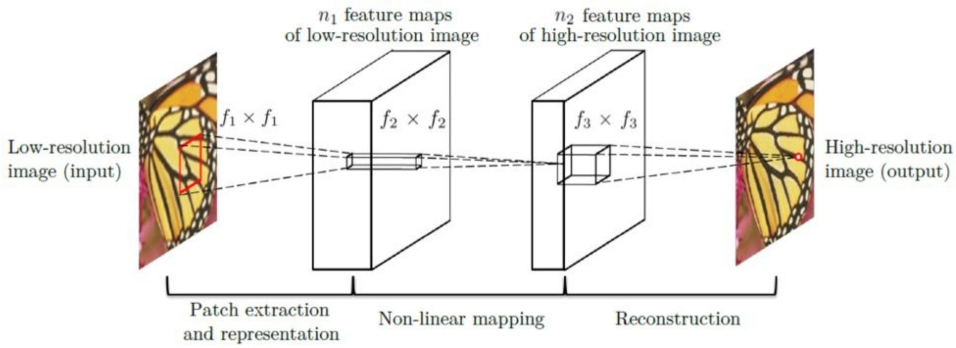
We used bicubic interpoltation as the baseline. It takes a low resolution image and resample the image to a larger size by filling in the added pixels with value interpolated from a 4x4 grid around it. This smooths the resampled image and increase the image resolution by approximating the best pixel values through interpolation. We were able to measure how this baseline method performs on out dataset.

There are a few other image super resolution methods based on machine learning. According to Lee et al., these methods are slower and less accurate than the method that we are using. We didn't use those methods on our dataset due to constraint on training time. The following comparison was performed by Lee's group.



Methods

We looked at SRCNN first. It takes a low resolution image and predict its corresponding high resolution image. It only has three layers. The structure is shown below. Its learning rate is small (1e-5), which makes it hard to converge within resonable time we have for this project.

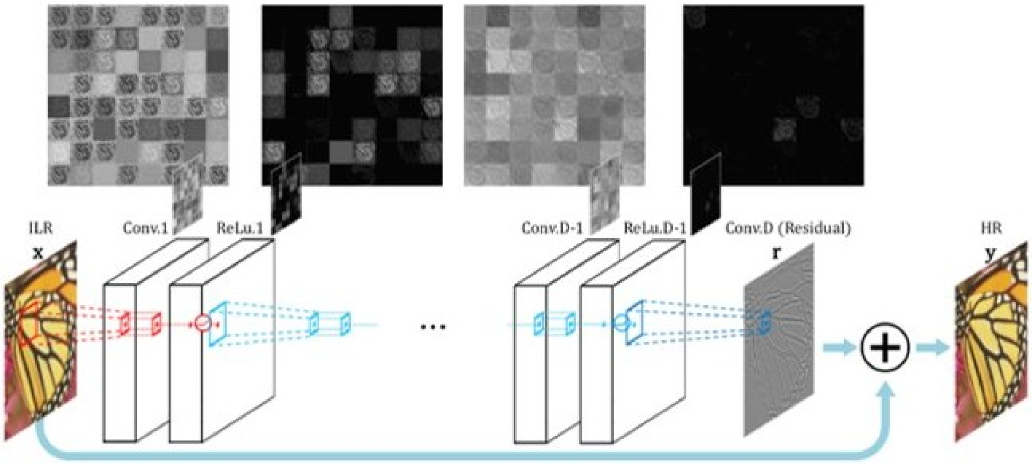


Layer	# of layers	Filter size	Channels	Activation
Input Layer	1	9 x 9	(1, 64)	ReLU
Hidden Layer	1	5 x 5	(64, 32)	ReLU
Output Layer	1	5 x 5	(32, 1)	-

We chose VDSR (Very Deep Super Resolution) instead to perform the task. It has a relatively short training time and good result compared to SRCNN. The main benefits of VDSR are listed below:

- Larger learning rate at 0.1 compared to 10e-5 of SRCNN, which allows the training to converge faster.
- Adjustable gradient clipping (MIN<gradient<MAX) to prevent the model from blowing up.
- Residual learning allows the model to learn the difference between high resolution and low resolution images(r = Y – X) instead of learning the high resolution image directly. Since the residual are small values, this prevents vanishing gradient.

The structure of VDSR is shown below.



Layer	# of layers	Filter size	Channels	Activation
Input Layer	1	3 x 3	(1, 64)	ReLU
Hidden Layer	18	3 x 3	(64, 64)	ReLU
Output Layer	1	3 x 3	(64, 1)	-

In the following cells, our VDSR model is implmented and trained. We see that the loss of the train and dev dataset both decreasing and that the psnr of the train and dev dateset both increasing with more epochs.

```
# Set up GPU
if torch.cuda.is_available():
    cuda = True
else:
    cuda = False
# os.environ["CUDA_VISIBLE_DEVICES"] = opt.gpus
device = 'cuda:0' if torch.cuda.is_available() else 'cpu'

# Check Dev accracy and loss
def approx_dev_acc_and_loss(model,x_original,y_original):
    x_original = Variable(x_original.cuda(), requires_grad=False)
    y_original = Variable(y_original.cuda(), requires_grad=False)
    x_Y = x_original
    y_Y = y_original
    residual_Y = y_Y-x_Y #batchx1x
    # print(x_Y.size())
    residual_Y_hat = model(x_Y) #
    loss_func = torch.nn.MSELoss(reduction ='sum')

    loss=loss_func(model(x_Y),residual_Y)
    psnr = PSNR(residual_Y_hat.cpu().detach().numpy()*255,
residual_Y.cpu().detach().numpy()*255)
    return psnr, loss
```

```
# Network
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.convInput = 
torch.nn.Sequential(torch.nn.Conv2d(1,64,3,1,1, bias=False), 
torch.nn.ReLU(inplace=True))
        layers = []
        for i in range(18):
             layers.append(torch.nn.Conv2d(64,64,3,1, 1, bias=False))
             layers.append(torch.nn.ReLU(inplace=True))
        self.hiddenLayer = torch.nn.Sequential(*layers)
        self.convOutput = torch.nn.Conv2d(64, 1, 3, 1, 1, bias=False)

        for m in self.modules():
            if isinstance(m, torch.nn.Conv2d):
                n = m.kernel_size[0] * m.kernel_size[1] * 
m.out_channels
                m.weight.data.normal_(0, sqrt(2. / n))

    def forward(self, x):
        # residual = x
        out = self.convInput(x)
        out = self.hiddenLayer(out)
        out = self.convOutput(out)
        # out = torch.add(out,residual)
        return out

# Set up Model parameters
MODEL_SAVE_DIR = "model_files/"
LEARNING_RATE = 0.1
BATCH_SIZE = 100
EPOCHS = 50
MOMENTUNM = 0.9
WEIGHT_DECAY = 0.0001

num_patches_x8 = 82467
num_patches_x4 = 53257
'''change the num_patches to corresponding scale'''
iteration_per_epoch = num_patches_x4//BATCH_SIZE

num_patches_dev= 15000
iteration_per_epoch_dev = num_patches_dev//BATCH_SIZE


model = Net()
steps = range(EPOCHS)
accuracies = np.zeros((EPOCHS,4))
optimizer = torch.optim.SGD(model.parameters(), lr=LEARNING_RATE, 
momentum=MOMENTUNM, weight_decay=WEIGHT_DECAY)
```

```
# resume from a pretrained data
# weights = torch.load('model_files/VDSR_x8/VDSR_epoch19_lr0.1.pt')
# model.load_state_dict(weights.state_dict())

loss_func = torch.nn.MSELoss(reduction ='sum')

if cuda:
    model = model.cuda()
    loss_func = loss_func.cuda()

# set up log file
LOG_DIR = "log/"
LOGFILE = open(os.path.join(LOG_DIR, f"vdsr"+"x4"+".log"),'w')
log_fieldnames = ['step', 'train_loss', 'train_psnr', 'dev_loss',
'dev_psnr']
logger = csv.DictWriter(LOGFILE, log_fieldnames)
logger.writeheader()

from psutil import virtual_memory
ram_gb = virtual_memory().total / 1e9
print('Your runtime has {:.1f} gigabytes of available RAM\
n'.format(ram_gb))

if ram_gb < 20:
  print('Not using a high-RAM runtime')
else:
  print('You are using a high-RAM runtime!')
```

```
Your runtime has 54.8 gigabytes of available RAM

You are using a high-RAM runtime!
```

```
# Learning
import gc
torch.cuda.empty_cache()

for step in range(EPOCHS):
    # step = step

    print('epoch: '+str(step))
    train_loss = 0
    train_psnr = 0
    for iteration in range(iteration_per_epoch):
        gc.collect()
        torch.cuda.empty_cache()
        '''load data'''
        '''change the loaded data to corresponding scale'''
        random_num = random.sample(range(num_patches_x4), BATCH_SIZE)
        x_original=LR_x4[random_num,0,:,:]
```

```
        y_original=HR_x4[random_num,0,:,:]
        x_original = x_original[:,None,:,:]
        y_original = y_original[:,None,:,:]

        x_original = Variable(x_original.cuda(), requires_grad=False)
        y_original = Variable(y_original.cuda(), requires_grad=False)
        x_Y = x_original
        y_Y = y_original

        '''residual learning'''
        # x_original = None
        # y_original = None

        residual_Y = y_Y-x_Y #batchx1x
        residual_Y_hat = model(x_Y) #
        loss=loss_func(model(x_Y),residual_Y)
        # Zero gradients, perform a backward pass, and update the
weights.
        optimizer.zero_grad()
        ## gradient clipping
        loss.backward()
        grad_clip = 0.4
        learning_rate = optimizer.param_groups[0]['lr']

        '''update learning rate'''
        for g in optimizer.param_groups:
            g['lr'] = LEARNING_RATE/(10**(step // 10))
        torch.nn.utils.clip_grad_norm_(model.parameters(),0.4)#
grad_clip/learning_rate)
        optimizer.step()


        ### TODO log accuracies and run validations
        if iteration%100 ==0:
            print(loss)
            print(learning_rate)
        train_loss = train_loss+loss.item()
        train_psnr =
train_psnr+PSNR(residual_Y_hat.cpu().detach().numpy()*255,residual_Y.c
pu().detach().numpy()*255)

    model_savepath =
os.path.join(MODEL_SAVE_DIR,f"VDSR_epoch{step}_lr{learning_rate}.pt")
    torch.save(model, model_savepath)

    # log accuracies
    train_loss = train_loss/iteration
    train_psnr = train_psnr/iteration

    random_num = random.sample(range(num_patches_dev), BATCH_SIZE)
```

```
    x_dev = LR_x4_dev[random_num,0,:,:]
    y_dev = HR_x4_dev[random_num,0,:,:]
    x_dev = x_dev[:,None,:,:]
    y_dev = y_dev[:,None,:,:]
    dev_psnr, dev_loss = approx_dev_acc_and_loss(model,x_dev,y_dev)

    accuracies[step][0] = train_loss
    accuracies[step][1] = train_psnr
    accuracies[step][2] = dev_loss
    accuracies[step][3] = dev_psnr
    step_metrics = {
                    'step': step,
                    'train_loss': train_loss,
                    'train_psnr': train_psnr,
                    'dev_loss': dev_loss,
                    'dev_psnr': dev_psnr
                }

    print(f"On step {step}:\tTrain psnr is {train_psnr}\t|\tDev psnr is {dev_psnr}")
    logger.writerow(step_metrics)


LOGFILE.close()

# save model
model_savepath = os.path.join(MODEL_SAVE_DIR,f"VDSR.pt")
print(model_savepath)
print("Training completed, saving model at {model_savepath}")
torch.save(model, model_savepath)
```

```
epoch: 0
tensor(347.6141, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(361.0608, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(387.1207, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(378.2018, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(404.1746, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(289.6548, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
On step 0: Train psnr is 27.324273362802735 |     Dev psnr is
27.54609005386439
epoch: 1
tensor(459.0286, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(351.7468, device='cuda:0', grad_fn=<MseLossBackward0>)
```

```
0.1
tensor(329.9837, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(292.3926, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(252.3450, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(272.0183, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
On step 1: Train psnr is 27.38463292031978  |     Dev psnr is
27.097845398348063
epoch: 2
tensor(264.6421, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(197.7473, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(297.3276, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(322.0832, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(254.0270, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(250.4630, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
On step 2: Train psnr is 27.450832541195524 |     Dev psnr is
25.2671464743879
epoch: 3
tensor(394.1964, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(339.8530, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(315.5304, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(276.9407, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(272.2747, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(490.5503, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
On step 3: Train psnr is 27.385243436907402 |     Dev psnr is
28.119329881549252
epoch: 4
tensor(269.3464, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(231.8857, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(296.8943, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(248.8809, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
```

```
tensor(334.6339, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(297.4858, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
On step 4: Train psnr is 27.48113690121922  |     Dev psnr is
26.7841674257986
epoch: 5
tensor(275.0550, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(321.4150, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(231.8809, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(304.3112, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(388.4303, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(305.7134, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
On step 5: Train psnr is 27.492697789681834 |     Dev psnr is
27.97956324078369
epoch: 6
tensor(357.8086, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(385.6545, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(332.6471, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(314.1372, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(276.7232, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(213.3384, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
On step 6: Train psnr is 27.484880645953297 |     Dev psnr is
26.620459182362847
epoch: 7
tensor(324.2387, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(278.3504, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(296.2817, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(260.9796, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(318.0877, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(235.4145, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
On step 7: Train psnr is 27.60236047215544  |     Dev psnr is
```

```
26.859180021059093
epoch: 8
tensor(333.1791, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(317.0591, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(199.0314, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(347.7928, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(264.5004, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(245.9287, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
On step 8: Train psnr is 27.502358366432137 |     Dev psnr is
28.61844620714768
epoch: 9
tensor(334.3035, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(248.7795, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(382.1470, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(360.6227, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(347.8069, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(259.6762, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
On step 9: Train psnr is 27.526052676124344 |     Dev psnr is
27.74287305784966
epoch: 10
tensor(328.0627, device='cuda:0', grad_fn=<MseLossBackward0>)
0.1
tensor(283.8655, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(277.3497, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(351.3776, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(284.5415, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(290.3948, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
On step 10:      Train psnr is 27.6718020965385   |     Dev psnr is
27.13309693000923
epoch: 11
tensor(286.7036, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(185.1712, device='cuda:0', grad_fn=<MseLossBackward0>)
```

```
0.01
tensor(252.9147, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(317.6248, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(209.6454, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(433.6430, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
On step 11:      Train psnr is 27.735677920146934 |     Dev psnr is
28.402550488556216
epoch: 12
tensor(417.9154, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(319.6056, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(316.9468, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(259.8520, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(355.1770, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(229.8378, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
On step 12:      Train psnr is 27.697535866751412 |     Dev psnr is
28.349505827389365
epoch: 13
tensor(477.2399, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(302.3488, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(265.7835, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(293.7851, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(313.8834, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(283.4041, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
On step 13:      Train psnr is 27.772711997914868 |     Dev psnr is
28.912115872294308
epoch: 14
tensor(291.0594, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(238.3547, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(244.7886, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(394.3388, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
```

```
tensor(329.2176, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(194.7014, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
On step 14:     Train psnr is 27.69270672373535  |    Dev psnr is
27.809336199957894
epoch: 15
tensor(371.2318, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(304.9635, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(310.1343, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(246.1262, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(334.2123, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(266.1339, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
On step 15:     Train psnr is 27.737316951706052 |    Dev psnr is
27.08251868718008
epoch: 16
tensor(198.1543, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(360.7124, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(283.7716, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(361.1004, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(257.4504, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(206.3339, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
On step 16:     Train psnr is 27.733388324829352 |    Dev psnr is
27.651881831685138
epoch: 17
tensor(306.5817, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(285.0924, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(374.4418, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(288.8366, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(311.3513, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(316.1406, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
On step 17:     Train psnr is 27.756036703750524 |    Dev psnr is
```

```
27.340259814964128
epoch: 18
tensor(233.1706, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(226.3076, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(350.2948, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(297.1228, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(296.0971, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(270.1392, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
On step 18:     Train psnr is 27.717022169935692 |     Dev psnr is
26.55065224767959
epoch: 19
tensor(253.9969, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(216.0825, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(359.8084, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(365.6469, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(252.3919, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(239.2912, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
On step 19:     Train psnr is 27.798914989653596 |     Dev psnr is
27.907016023229613
epoch: 20
tensor(162.6539, device='cuda:0', grad_fn=<MseLossBackward0>)
0.01
tensor(222.6201, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(226.8509, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(275.2074, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(279.4683, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(342.8154, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
On step 20:     Train psnr is 27.769174700030604 |     Dev psnr is
27.2644574041273
epoch: 21
tensor(366.5626, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(364.5109, device='cuda:0', grad_fn=<MseLossBackward0>)
```

```
0.001
tensor(242.2011, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(314.7084, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(229.5826, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(318.1482, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
On step 21:	Train psnr is 27.764573614323094 |	Dev psnr is
27.87730298293866
epoch: 22
tensor(321.2897, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(275.0717, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(297.9004, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(427.5904, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(432.3963, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(310.2193, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
On step 22:	Train psnr is 27.785643285418892 |	Dev psnr is
27.209041353322764
epoch: 23
tensor(262.9707, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(290.7393, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(231.3060, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(419.3757, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(348.1208, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(295.0320, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
On step 23:	Train psnr is 27.76845429200094  |	Dev psnr is
27.843473027374017
epoch: 24
tensor(306.5670, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(259.4514, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(239.5770, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(315.1597, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
```

```
tensor(361.6018, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(256.9341, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
On step 24:      Train psnr is 27.798565863661555 |     Dev psnr is 
26.934068512029835
epoch: 25
tensor(263.9359, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(206.3984, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(254.0414, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(331.3440, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(313.7055, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(243.5388, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
On step 25:      Train psnr is 27.7697835651079   |     Dev psnr is 
27.421690012220196
epoch: 26
tensor(211.4222, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(269.9775, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(246.7943, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(326.3143, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(306.3583, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(326.9808, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
On step 26:      Train psnr is 27.81076383945202  |     Dev psnr is 
27.683528296823425
epoch: 27
tensor(304.3029, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(270.8540, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(278.2115, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(188.7290, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(252.4995, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(261.0257, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
On step 27:      Train psnr is 27.756980862983614 |     Dev psnr is 
```

```
27.258514688175982
epoch: 28
tensor(248.9451, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(287.3278, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(295.0452, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(385.8092, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(278.9909, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(243.9802, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
On step 28:     Train psnr is 27.706670977795827 |     Dev psnr is
26.237424947868988
epoch: 29
tensor(244.6421, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(389.5529, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(281.1658, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(237.2150, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(258.8379, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(187.6891, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
On step 29:     Train psnr is 27.74569793940569  |     Dev psnr is
28.99835331647013
epoch: 30
tensor(232.2569, device='cuda:0', grad_fn=<MseLossBackward0>)
0.001
tensor(291.5595, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(243.1548, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(232.1934, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(227.8727, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(407.6944, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
On step 30:     Train psnr is 27.845488002883812 |     Dev psnr is
25.949592247540902
epoch: 31
tensor(200.7087, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(310.5175, device='cuda:0', grad_fn=<MseLossBackward0>)
```

```
0.0001
tensor(373.9138, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(346.5282, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(192.4262, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(294.7422, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
On step 31:      Train psnr is 27.754554015171866 |     Dev psnr is
27.05893171469606
epoch: 32
tensor(251.6885, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(265.0058, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(229.1410, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(266.0229, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(317.1397, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(204.7025, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
On step 32:      Train psnr is 27.829205050230303 |     Dev psnr is
26.856164292479527
epoch: 33
tensor(302.1674, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(351.7724, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(242.6647, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(274.2629, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(237.4435, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(376.3542, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
On step 33:      Train psnr is 27.823982886507157 |     Dev psnr is
28.76155429463259
epoch: 34
tensor(227.3174, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(322.5673, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(343.5649, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(342.1089, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
```

```
tensor(492.6958, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(308.2189, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
On step 34:	Train psnr is 27.75367875725845  |	Dev psnr is
27.59287747966317
epoch: 35
tensor(364.8126, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(307.6234, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(226.1742, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(288.7428, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(278.0165, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(358.9467, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
On step 35:	Train psnr is 27.784661093976794 |	Dev psnr is
26.938353284109432
epoch: 36
tensor(248.0714, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(278.5196, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(266.0521, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(337.2539, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(386.6424, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(317.5680, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
On step 36:	Train psnr is 27.711357138118792 |	Dev psnr is
28.191315113655616
epoch: 37
tensor(262.0403, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(267.0296, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(369.3685, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(313.2017, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(224.6406, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(193.4917, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
On step 37:	Train psnr is 27.78474649356977  |	Dev psnr is
```

```
27.757090999717025
epoch: 38
tensor(250.1964, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(249.6889, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(298.9068, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(299.0393, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(306.5362, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(385.0784, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
On step 38:      Train psnr is 27.746725966534925 |     Dev psnr is
27.678103102180035
epoch: 39
tensor(305.5492, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(278.0754, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(215.3063, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(405.2295, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(222.5153, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(340.0468, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
On step 39:      Train psnr is 27.78173007041883  |     Dev psnr is
29.518346083072785
epoch: 40
tensor(417.4430, device='cuda:0', grad_fn=<MseLossBackward0>)
0.0001
tensor(261.6385, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(262.0757, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(288.6707, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(324.0726, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(306.8070, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
On step 40:      Train psnr is 27.782391691483642 |     Dev psnr is
26.8215911840166
epoch: 41
tensor(325.9648, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(382.6058, device='cuda:0', grad_fn=<MseLossBackward0>)
```

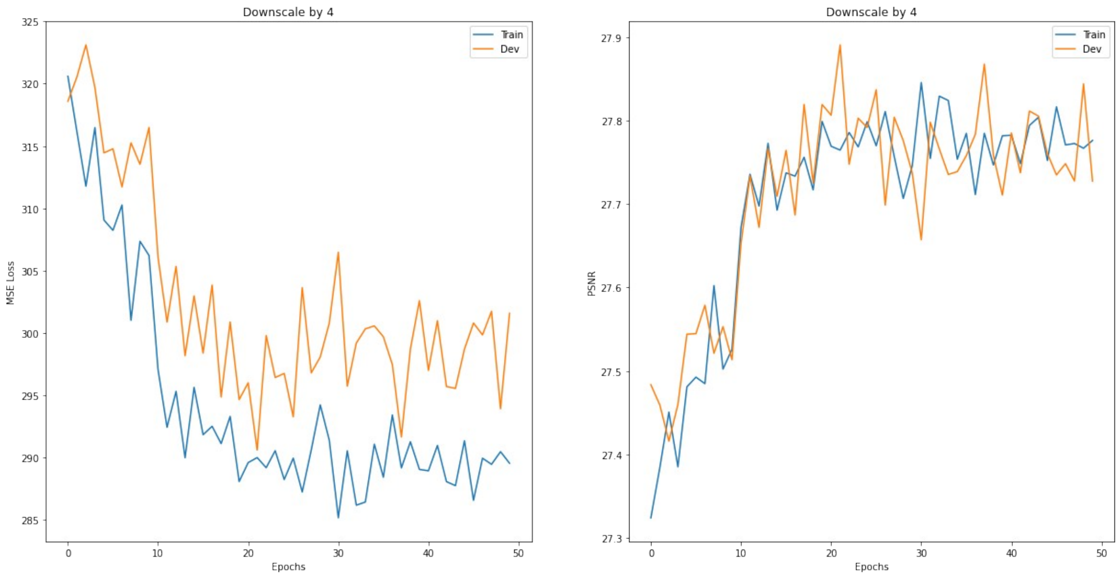
```
1e-05
tensor(227.1507, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(264.1241, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(292.3781, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(289.2066, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
On step 41:      Train psnr is 27.74832863178137  |     Dev psnr is
26.466459646913574
epoch: 42
tensor(206.4367, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(275.5203, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(356.5497, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(276.5396, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(247.1103, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(287.3742, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
On step 42:      Train psnr is 27.794084318543153 |     Dev psnr is
27.379110132055104
epoch: 43
tensor(295.6121, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(288.3493, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(332.5367, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(238.1612, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(259.3045, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(295.9945, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
On step 43:      Train psnr is 27.80391712767781  |     Dev psnr is
26.939618434748724
epoch: 44
tensor(248.2262, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(236.3332, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(351.3100, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(201.1040, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
```

```
tensor(392.6807, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(287.7951, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
On step 44:     Train psnr is 27.7521117785386  |    Dev psnr is
27.770333331539128
epoch: 45
tensor(263.7426, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(384.8681, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(289.0579, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(246.0593, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(219.7635, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(263.3697, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
On step 45:     Train psnr is 27.81643279461911  |    Dev psnr is
27.216750934500897
epoch: 46
tensor(228.0663, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(309.7374, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(258.7872, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(245.5911, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(269.2495, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(271.6288, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
On step 46:     Train psnr is 27.77077718406371  |    Dev psnr is
27.9346949763533
epoch: 47
tensor(304.3947, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(193.1528, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(258.6440, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(273.4726, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(281.1389, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(308.1013, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
On step 47:     Train psnr is 27.772504816410702|    Dev psnr is
```

```
28.67926791058972
epoch: 48
tensor(260.7970, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(249.2240, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(301.1936, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(260.0926, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(290.9763, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(314.0345, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
On step 48:	Train psnr is 27.766807696781445 |	Dev psnr is
26.932202512726274
epoch: 49
tensor(272.9760, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(249.8696, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(188.3887, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(311.5681, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(275.3982, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
tensor(209.5669, device='cuda:0', grad_fn=<MseLossBackward0>)
1e-05
On step 49:	Train psnr is 27.776143602430295 |	Dev psnr is
28.118628358524134
model_files/VDSR.pt
Training completed, saving model at {model_savepath}
```

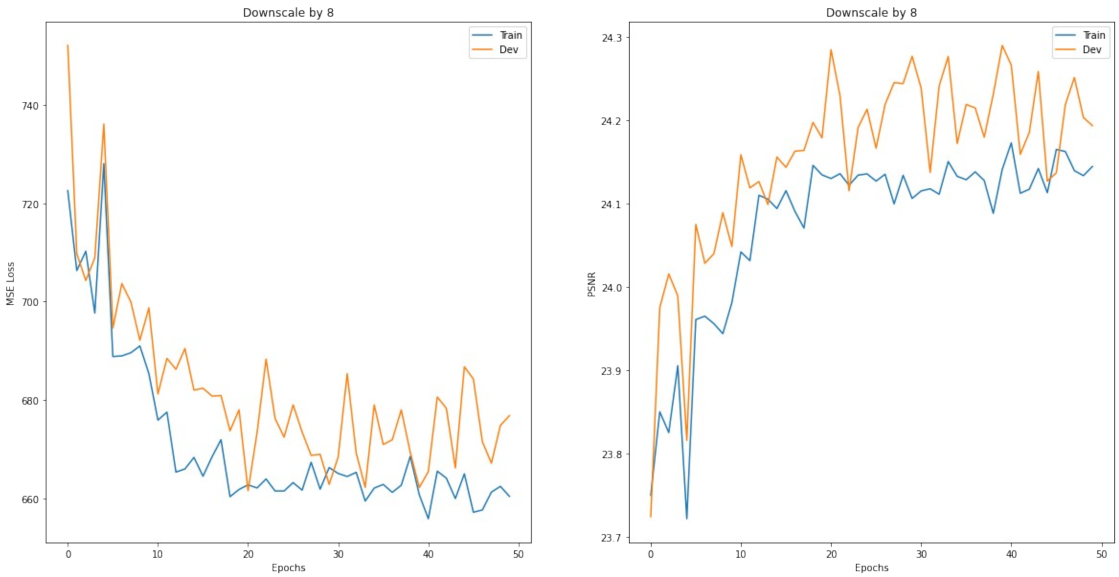
```python
#  Show plots of how these models performed during training.
#  For example, plot train loss and train accuracy (or other
evaluation metric) on the y-axis,
#  with number of iterations or number of examples on the x-axis.
import re

# load model for x4
logfile1 = open('log/vdsrx4_reducing_lr.log', newline='')
logfile2 = open('log/vdsrx4_dev.log', newline='')

reader1 = csv.reader(logfile1, delimiter=',')
reader2 = csv.reader(logfile2, delimiter=',')

train_loss = np.zeros(EPOCHS)
train_psnr = np.zeros(EPOCHS)
dev_loss = np.zeros(EPOCHS)
```

```
dev_psnr = np.zeros(EPOCHS)
steps = range(EPOCHS)
row_id = 0
for row in reader1:
    if row_id > 0:
        train_loss[row_id-1] = float(row[1])
        train_psnr[row_id-1] = float(row[2])
    row_id +=1

row_id = 0
for row in reader2:

    if row_id > 0:
        dev_loss[row_id-1] = float(row[3])
        dev_psnr[row_id-1] = float(row[4])
    row_id +=1

fig1, ax1 = plt.subplots(1,2)
ax1[0].plot(steps, train_loss)
ax1[0].plot(steps, dev_loss)
ax1[0].set_title('Downscale by 4')
ax1[0].set_xlabel('Epochs')
ax1[0].set_ylabel('MSE Loss')
ax1[0].legend(['Train', 'Dev'])

ax1[1].plot(steps, train_psnr)
ax1[1].plot(steps, dev_psnr)
ax1[1].set_title('Downscale by 4')
ax1[1].set_xlabel('Epochs')
ax1[1].set_ylabel('PSNR')
ax1[1].legend(['Train', 'Dev'])

plt.show()
plt.rcParams['figure.figsize'] = [20, 10]
```




```python
import re

# load model for x8
logfile = open('log/vdsrx8_all.log', newline='')
reader = csv.reader(logfile, delimiter=',')
train_loss = np.zeros(EPOCHS)
train_psnr = np.zeros(EPOCHS)
dev_loss = np.zeros(EPOCHS)
dev_psnr = np.zeros(EPOCHS)
steps = range(EPOCHS)
row_id = 0
for row in reader:
    if row_id > 0:
        # print(float(row[1]))
        train_loss[row_id-1] = float(row[1])
        train_psnr[row_id-1] = float(row[2])

        dev_loss[row_id-1] = float(row[3])
        dev_psnr[row_id-1] = float(row[4])
    row_id +=1
fig1, ax1 = plt.subplots(1,2)
ax1[0].plot(steps, train_loss)
ax1[0].plot(steps, dev_loss)
ax1[0].set_title('Downscale by 8')
ax1[0].set_xlabel('Epochs')
ax1[0].set_ylabel('MSE Loss')
ax1[0].legend(['Train', 'Dev'])

ax1[1].plot(steps, train_psnr)
ax1[1].plot(steps, dev_psnr)
ax1[1].set_title('Downscale by 8')
ax1[1].set_xlabel('Epochs')
```

```
ax1[1].set_ylabel('PSNR')
ax1[1].legend(['Train', 'Dev'])
plt.show()
plt.rcParams['figure.figsize'] = [20, 10]
```



Results

Show tables comparing your methods to the baselines.

What about these results surprised you? Why?

Did your models over- or under-fit? How can you tell? What did you do to address these issues?

What does the evaluation of your trained models tell you about your data? How do you expect these models might behave differently on different data?

##Test Result(Scale = 8)

```
# Show plots or visualizations of your evaluation metric(s) on the
train and test sets.
#   What do these plots show about over- or under-fitting?
#   You may borrow from how we visualized results in the Lab
homeworks.
#   Are there aspects of your results that are difficult to visualize?
Why?
torch.cuda.empty_cache()

# Specifies weights files
WEIGHTS_FILE =
"./model_files/VDSR_x8_reducing_lr/VDSR_epoch25_lr0.001.pt"
DATA_DIR = "./dataset"
if WEIGHTS_FILE is None : raise TypeError("for inference, model
```

```
weights must be specified")

# Specify test images
TEST_IMAGES = DATA_DIR+ "/DIV2K_test_LR_bicubic/X8"
GROUND_TRUTH_IMAGES = DATA_DIR+ "/DIV2K_test_HR"

# Specify result directory
RESULT_DIR = "./results/test_result_x8/"
BICUBIC_DIR = "./results/bicubic_x8/"

# Load weights
model = Net()
weights = torch.load(WEIGHTS_FILE)
model.load_state_dict(weights.state_dict())

if cuda:
    model = model.cuda()
#model = torch.load(WEIGHTS_FILE, map_location=torch.device('cuda'))
plt.rcParams['figure.figsize'] = [50, 25]


# Initialize parameters
scale = 8
patch_size = 41

predictions = []
PSNR_Bicubic = {}
PSNR_Model = {}

for lr_filename in os.listdir(TEST_IMAGES):


    hr_img_YCbCr =
np.array(Image.open(GROUND_TRUTH_IMAGES+"/"+lr_filename[0:-
6]+".png").convert('YCbCr'))

    print(np.shape(hr_img_YCbCr)[0:2])

    lr_img_YCbCr_raw =
Image.open(TEST_IMAGES+"/"+lr_filename).convert('YCbCr')
    lr_img_YCbCr = lr_img_YCbCr_raw.resize((np.shape(hr_img_YCbCr)
[1],np.shape(hr_img_YCbCr)[0]),Image.BICUBIC)
    lr_img_YCbCr = np.array(lr_img_YCbCr.convert('YCbCr'))


    height = np.shape(lr_img_YCbCr)[0]
    width = np.shape(lr_img_YCbCr)[1]

    # Run bicubic interpolation on scaled LR images
```

```
    hr_img_YCbCr_modeled = 
lr_img_YCbCr_raw.resize((width,height),Image.BICUBIC)
    hr_img_YCbCr_modeled.convert('RGB').save(BICUBIC_DIR + 
lr_filename[0:-6] + "bicubic.png")
    hr_img_YCbCr_modeled = np.array(hr_img_YCbCr_modeled)


    # Calculate psnr of the baseline
    psnr_bicubic = PSNR(hr_img_YCbCr_modeled[:,:,0], 
hr_img_YCbCr[:,:,0])
    PSNR_Bicubic[lr_filename] = psnr_bicubic

    patch = product(range(0, height-height%patch_size, patch_size), 
        range(0, width-width%patch_size, patch_size))

    patch_id = 0
    row = len(range(0, height-height%patch_size, patch_size))
    col = len(range(0, width-width%patch_size, patch_size))
    lr_batch = np.zeros((row*col,1,patch_size,patch_size));
    hr_batch = np.zeros((row*col,1,patch_size,patch_size));

    for row,col in patch:
        lr_patch_cur = 
lr_img_YCbCr[row:row+patch_size,col:col+patch_size,:]

        if row+patch_size > height and col+patch_size > width:
            pass
        else:
            # pass the current patch into the model
            lr_patch_YCbCr = copy.deepcopy(lr_patch_cur)

            lr_patch_cur = lr_patch_cur[:,:,0]/255
            lr_patch_cur = lr_patch_cur[None,None,:,:]

            lr_patch_cur_ = 
torch.from_numpy((lr_patch_cur).astype(np.float32)).cuda()

            lr_patch_modeled = (model(lr_patch_cur_)
+lr_patch_cur_)*255
            lr_patch_modeled = lr_patch_modeled.cpu()

hr_img_YCbCr_modeled[row:row+patch_size,col:col+patch_size,0] = 
lr_patch_modeled.detach().numpy()[0,0,:,:]

            patch_id += 1

    hr_img_YCbCr_modeled[hr_img_YCbCr_modeled < 0] = 0
    hr_img_YCbCr_modeled[hr_img_YCbCr_modeled > 255] = 255
```

```
    # Calculate PSNR for the model
    psnr_cur = PSNR(hr_img_YCbCr_modeled[:,:,0], hr_img_YCbCr[:,:,0])
    PSNR_Model[lr_filename] = psnr_cur

    predicted_residual = hr_img_YCbCr_modeled - lr_img_YCbCr
    predicted_residual[:,:,1] = predicted_residual[:,:,1]*0
    predicted_residual[:,:,2] = predicted_residual[:,:,2]*0
    predicted_residual =
Image.fromarray(predicted_residual.astype('uint8'), 'YCbCr')
    predicted_residual_RGB = predicted_residual.convert('RGB')

    hr_img_RGB_modeled =
Image.fromarray(hr_img_YCbCr_modeled.astype('uint8'), 'YCbCr')
    lr_img_RGB = Image.fromarray(lr_img_YCbCr.astype('uint8'),
'YCbCr')
    hr_img_RGB_modeled =  hr_img_RGB_modeled.convert('RGB')
    lr_img_RGB =  lr_img_RGB.convert('RGB')

    hr_img_RGB_modeled.save(RESULT_DIR+lr_filename[0:-6]+".png")
    predicted_residual_RGB.save(RESULT_DIR+lr_filename[0:-
6]+"residual.png")
```

```
(1356, 2040)
(1356, 2040)
(1464, 2040)
(1536, 2040)
(1536, 2040)
(1356, 2040)
(1356, 2040)
(1356, 2040)
(1356, 2040)
(1536, 2040)
```

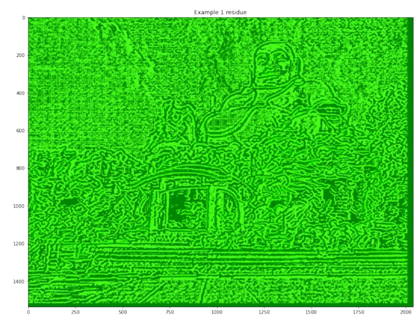
```
# Visualize PSNR
print("Filename     X8 Bicubic PSNR     X8 Model PSNR")
for fn, psnr in PSNR_Bicubic.items():
    print('{}   {}   {}'.format(fn, psnr, PSNR_Model[fn]))
```

```
Filename     X8 Bicubic PSNR     X8 Model PSNR
0811x8.png   30.266403119185405   30.3182482099864
0814x8.png   33.79035465463271   33.81119733854863
0813x8.png   32.37217234459105   32.50950098277856
0818x8.png   31.924474785304362   32.042948508974774
0817x8.png   31.97732041394847   32.12487023044744
0816x8.png   31.67781789553422   31.717738548961627
0819x8.png   30.12370399726379   30.19015936868025
0815x8.png   32.10575307820521   32.133501955875595
```

```
0820x8.png   29.24620326992112   29.359758896741436
0812x8.png   30.947215090398807   31.061310096152006
```

```python
# Plot images

HR_IMAGES_TEST = DATA_DIR+ "/DIV2K_test_HR"
LR_IMAGES_TSET = BICUBIC_DIR
HR_IMAGES_MODELED = RESULT_DIR

images_hr = os.listdir(HR_IMAGES_TEST)
img1_hr = Image.open(HR_IMAGES_TEST +'/0818.png').convert('RGB')

images_lr = os.listdir(LR_IMAGES_TSET)
img1_lr_bicubic = Image.open(LR_IMAGES_TSET +
'/0818bicubic.png').convert('RGB')

img1_hr_modeled = Image.open(HR_IMAGES_MODELED +
'/0818.png').convert('RGB')

img1_hr_residual = Image.open(HR_IMAGES_MODELED +
'/0818residual.png').convert('RGB')


f, axarr = plt.subplots(2,2,gridspec_kw={'height_ratios': [1, 1]})
axarr[0, 0].imshow(img1_hr)
axarr[0, 0].set_title('Example 1 HR')
axarr[0, 1].imshow(img1_lr_bicubic)
axarr[0, 1].set_title('Example 1 Bicubic')
axarr[1, 0].imshow(img1_hr_modeled)
axarr[1, 0].set_title('Example 1 modeled')
axarr[1, 1].imshow(img1_hr_residual)
axarr[1, 1].set_title('Example 1 residue')
```

```
Text(0.5, 1.0, 'Example 1 residue')
```



What about these results surprised you? Why?

We are not particularly suprised by any of the results, but we did observe that our models do not perform as well as the model in the paper. The authors of the paper reported an increase in the PSNR by 2.0, but our maximum increase is less than 1.

Did your models over- or under-fit? How can you tell? What did you do to address these issues?

Our models did not overfit as the dev loss was not significantly higher than the train loss during the 50 epoches for which we trained our models.

No direct evidence shows that our model underfit but we cannot rule out the possibility. The train and dev loss decreased very slowly after the first few epoches. It is possible that we reached a local minimum and our model might perform better if we ran the training process longer without decreasing the learning rate.

What does the evaluation of your trained models tell you about your data? How do you expect these models might behave differently on different data? The evaluation of our trained models shows that our data is well-formed and unbiased as the trained models performed equally well on images with people, animals, plants, and inorganic matters. We would expect the models trained on a larg scale factor(e.g. X8) to perform better on test data of a smaller scale factor(e.g. X4), and models trained on a small scale factor to perform worse n test data of a larger scale factor.

##Test Result(Scale = 4)

```
# Show plots or visualizations of your evaluation metric(s) on the
train and test sets.
#   What do these plots show about over- or under-fitting?
```

```
#   You may borrow from how we visualized results in the Lab
homeworks.
#   Are there aspects of your results that are difficult to visualize?
Why?
torch.cuda.empty_cache()

# Specifies weights files
WEIGHTS_FILE =
"./model_files/VDSR_x4_reducing_lr/VDSR_epoch25_lr0.001.pt"
DATA_DIR = "./dataset"
if WEIGHTS_FILE is None : raise TypeError("for inference, model
weights must be specified")

# Specify test images
TEST_IMAGES = DATA_DIR+ "/DIV2K_test_LR_bicubic/X4"
GROUND_TRUTH_IMAGES = DATA_DIR+ "/DIV2K_test_HR"

# Specify result directory
RESULT_DIR = "./results/test_result_x4/"
BICUBIC_DIR = "./results/bicubic_x4/"

# Load weights
model = Net()
weights = torch.load(WEIGHTS_FILE)
model.load_state_dict(weights.state_dict())

if cuda:
    model = model.cuda()
#model = torch.load(WEIGHTS_FILE, map_location=torch.device('cuda'))
plt.rcParams['figure.figsize'] = [50, 25]


# Initialize parameters
scale = 8
patch_size = 41

predictions = []
PSNR_Bicubic = {}
PSNR_Model = {}

for lr_filename in os.listdir(TEST_IMAGES):


    hr_img_YCbCr =
np.array(Image.open(GROUND_TRUTH_IMAGES+"/"+lr_filename[0:-
6]+".png").convert('YCbCr'))

    print(np.shape(hr_img_YCbCr)[0:2])
```

```
    lr_img_YCbCr_raw = 
Image.open(TEST_IMAGES+"/"+lr_filename).convert('YCbCr')
    lr_img_YCbCr = lr_img_YCbCr_raw.resize((np.shape(hr_img_YCbCr)
[1],np.shape(hr_img_YCbCr)[0]),Image.BICUBIC)
    lr_img_YCbCr = np.array(lr_img_YCbCr.convert('YCbCr'))


    height = np.shape(lr_img_YCbCr)[0]
    width = np.shape(lr_img_YCbCr)[1]

    # Run bicubic interpolation on scaled LR images
    hr_img_YCbCr_modeled = 
lr_img_YCbCr_raw.resize((width,height),Image.BICUBIC)
    hr_img_YCbCr_modeled.convert('RGB').save(BICUBIC_DIR + 
lr_filename[0:-6] + "bicubic.png")
    hr_img_YCbCr_modeled = np.array(hr_img_YCbCr_modeled)


    # Calculate psnr of the baseline
    psnr_bicubic = PSNR(hr_img_YCbCr_modeled[:,:,0], 
hr_img_YCbCr[:,:,0])
    PSNR_Bicubic[lr_filename] = psnr_bicubic

    patch = product(range(0, height-height%patch_size, patch_size),
        range(0, width-width%patch_size, patch_size))

    patch_id = 0
    row = len(range(0, height-height%patch_size, patch_size))
    col = len(range(0, width-width%patch_size, patch_size))
    lr_batch = np.zeros((row*col,1,patch_size,patch_size));
    hr_batch = np.zeros((row*col,1,patch_size,patch_size));

    for row,col in patch:
        lr_patch_cur = 
lr_img_YCbCr[row:row+patch_size,col:col+patch_size,:]

        if row+patch_size > height and col+patch_size > width:
            pass
        else:
            # pass the current patch into the model
            lr_patch_YCbCr = copy.deepcopy(lr_patch_cur)

            lr_patch_cur = lr_patch_cur[:,:,0]/255
            lr_patch_cur = lr_patch_cur[None,None,:,:]

            lr_patch_cur_ = 
torch.from_numpy((lr_patch_cur).astype(np.float32)).cuda()

            lr_patch_modeled = (model(lr_patch_cur_)
+lr_patch_cur_)*255
```

```
            lr_patch_modeled = lr_patch_modeled.cpu()

hr_img_YCbCr_modeled[row:row+patch_size,col:col+patch_size,0] =
lr_patch_modeled.detach().numpy()[0,0,:,:]

            patch_id += 1

    hr_img_YCbCr_modeled[hr_img_YCbCr_modeled < 0] = 0
    hr_img_YCbCr_modeled[hr_img_YCbCr_modeled > 255] = 255

    # Calculate PSNR for the model
    psnr_cur = PSNR(hr_img_YCbCr_modeled[:,:,0], hr_img_YCbCr[:,:,0])
    PSNR_Model[lr_filename] = psnr_cur

    predicted_residual = hr_img_YCbCr_modeled - lr_img_YCbCr
    predicted_residual[:,:,1] = predicted_residual[:,:,1]*0
    predicted_residual[:,:,2] = predicted_residual[:,:,2]*0
    predicted_residual =
Image.fromarray(predicted_residual.astype('uint8'), 'YCbCr')
    predicted_residual_RGB = predicted_residual.convert('RGB')

    hr_img_RGB_modeled =
Image.fromarray(hr_img_YCbCr_modeled.astype('uint8'), 'YCbCr')
    lr_img_RGB = Image.fromarray(lr_img_YCbCr.astype('uint8'),
'YCbCr')
    hr_img_RGB_modeled =  hr_img_RGB_modeled.convert('RGB')
    lr_img_RGB =  lr_img_RGB.convert('RGB')

    hr_img_RGB_modeled.save(RESULT_DIR+lr_filename[0:-6]+".png")
    predicted_residual_RGB.save(RESULT_DIR+lr_filename[0:-
6]+"residual.png")
```

```
(1356, 2040)
(1356, 2040)
(1356, 2040)
(1536, 2040)
(1536, 2040)
(1356, 2040)
(1356, 2040)
(1536, 2040)
(1356, 2040)
(1464, 2040)
```

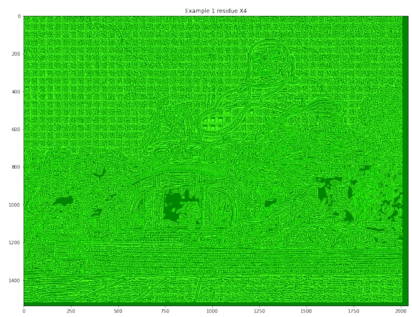
```
# Visualize PSNR
print("Filename     X8 Bicubic PSNR     X8 Model PSNR")
for fn, psnr in PSNR_Bicubic.items():
    print('{}   {}   {}'.format(fn, psnr, PSNR_Model[fn]))
```

```
Filename     X8 Bicubic PSNR     X8 Model PSNR
0814x4.png   34.93042484596772   35.172679600175826
```

```
0820x4.png   30.488614222528057   30.953786037321937
0819x4.png   31.793683234794496   32.27042547305936
0812x4.png   31.906946865726972   32.23992451057481
0817x4.png   34.142967991765985   34.69891835230119
0815x4.png   35.917929375586795   36.47941698853296
0816x4.png   33.320342325248006   33.72260367308189
0818x4.png   33.55037360764831   34.09697313324899
0811x4.png   31.85080724162149   32.14451758438129
0813x4.png   34.08783871660636   34.723195930182406
```

```
# Plot images

HR_IMAGES_TEST = DATA_DIR+ "/DIV2K_test_HR"
LR_IMAGES_TSET = BICUBIC_DIR
HR_IMAGES_MODELED = RESULT_DIR

images_hr = os.listdir(HR_IMAGES_TEST)
img1_hr = Image.open(HR_IMAGES_TEST +'/0818.png').convert('RGB')

images_lr = os.listdir(LR_IMAGES_TSET)
img1_lr_bicubic = Image.open(LR_IMAGES_TSET +
'/0818bicubic.png').convert('RGB')

img1_hr_modeled = Image.open(HR_IMAGES_MODELED +
'/0818.png').convert('RGB')

img1_hr_residual = Image.open(HR_IMAGES_MODELED +
'/0818residual.png').convert('RGB')


f, axarr = plt.subplots(2,2,gridspec_kw={'height_ratios': [1, 1]})
axarr[0, 0].imshow(img1_hr)
axarr[0, 0].set_title('Example 1 HR X4')
axarr[0, 1].imshow(img1_lr_bicubic)
axarr[0, 1].set_title('Example 1 Bicubic X4')
axarr[1, 0].imshow(img1_hr_modeled)
axarr[1, 0].set_title('Example 1 modeled X4')
axarr[1, 1].imshow(img1_hr_residual)
axarr[1, 1].set_title('Example 1 residue X4')
```

```
Text(0.5, 1.0, 'Example 1 residue X4')
```



Discussion

Our project focuses on bringing low resolution images to high resolution using VDSR model (very deep convolutional network). So, in this project we use deep learning especially convolution networks and ReLu activation function to construct our model.

What mostly surprises us is the transformation between RGB and YCbCr. According to medical research, human eyes are more sensitive to brightness than color. In YCbCr, Y is the luminance while Cb and Cr are about color. So,we can save a lot of bandwidth if we transmit the intensity in high resolution and colour in lower resolutionby by converting RGB to YCbCr and leaving Y channel only for learning.

In our project, the most challenging problem is that the convergence velocity is relatively slow because of the very deep convolutional networks. So we adjust the learning rate adaptively and apply residual learning. So for very large dataset or very conplex model such as ours, residual learning is a really pratical and efficient method.

Through the final project, our TA GuangHui Qin suggested us to input into the model the low resolution image processed by baseline method fisrt, which saves significant convergence time.

If we have more time to polish our model, we'd spend more time on adjustment our model to further increase the convergence velocity. We would like to efficiently load a larger dataset to increase the model performance. We would like to tune some hyperparameters such as depth of network and learning rate to get a better understanding of the model.

Reference

1. Kim, Jiwon, et al. "Accurate Image Super-Resolution Using Very Deep Convolutional Networks." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, https://doi.org/10.1109/cvpr.2016.182.
2. C. Dong, C. C. Loy, K. He, and X. Tang. Image superresolution using deep convolutional networks. TPAMI, 2015.