Implementing the baseline paper

Loading Image Paths

We load in our image data.

In [ ]:
```
import os
import glob
import time
import numpy as np
from PIL import Image
from pathlib import Path
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
from skimage.color import rgb2lab, lab2rgb

import torch
from torch import nn, optim
from torchvision import transforms
from torchvision.utils import make_grid
from torch.utils.data import Dataset, DataLoader
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
use_colab = None
```

Mounting Google drive path. Comment this section if not used.

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

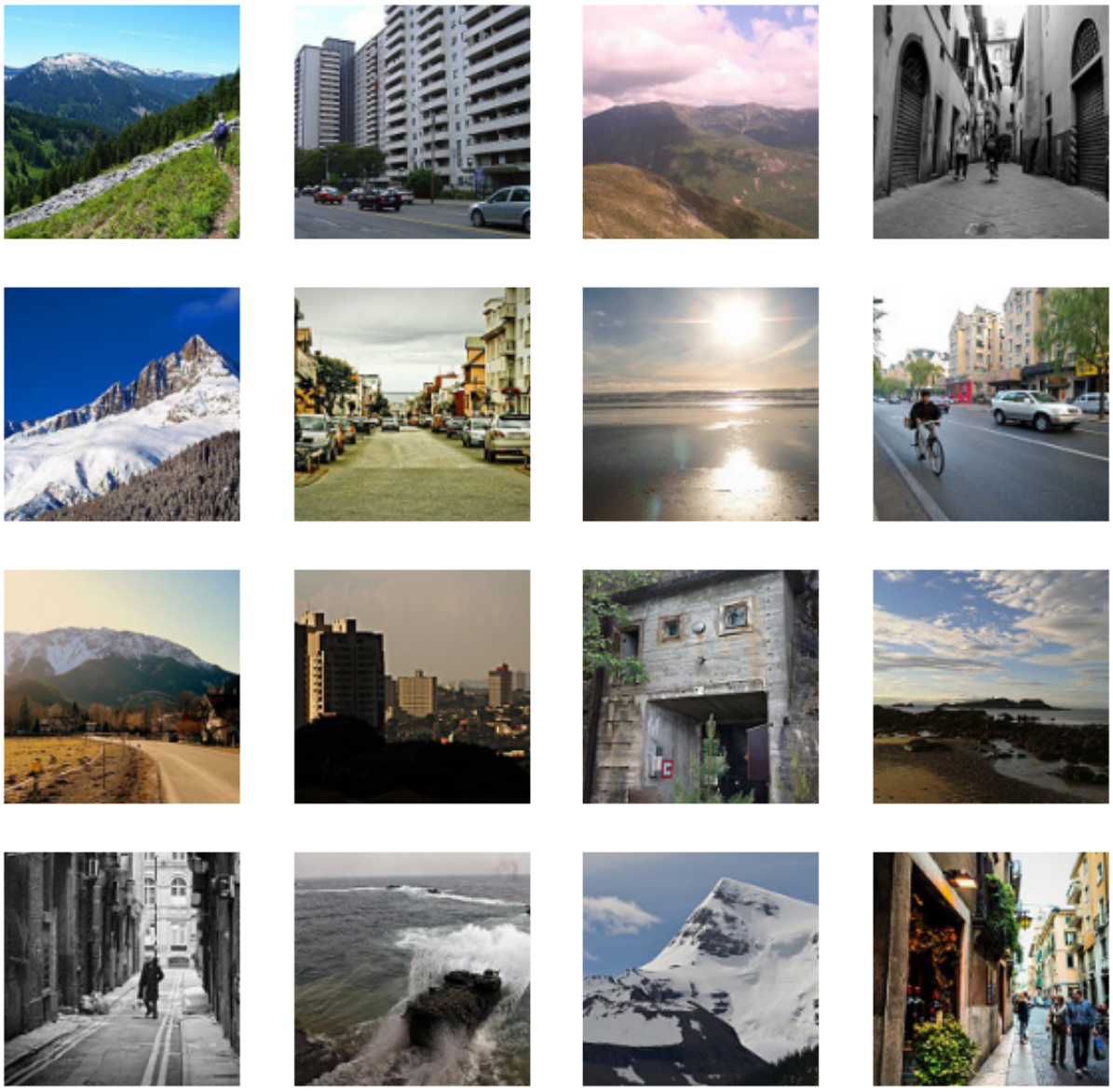
```
Mounted at /content/drive
```

In [ ]:
```
#path = "path to the dataset"
path = '/content/drive/MyDrive/'

paths = glob.glob(path + "Deep Learning/*.jpg") # Fetching image files
np.random.seed(123)
print(len(paths))
paths_subset = np.random.choice(paths, 2_000, replace=False) # choosin
rand_idxs = np.random.permutation(2_000)
train_idxs = rand_idxs[:1500] # choosing the first 1500 for training
val_idxs = rand_idxs[1500:] # choosing last 500 for validation
train_paths = paths_subset[train_idxs]
val_paths = paths_subset[val_idxs]
print(len(train_paths), len(val_paths))
```

```
3000
1500 500
```

In [ ]:
```
_, axes = plt.subplots(4, 4, figsize=(10, 10))
for ax, img_path in zip(axes.flatten(), train_paths):
    ax.imshow(Image.open(img_path))
    ax.axis("off")
```



In [ ]:
```
SIZE = 256
class ColorizationDataset(Dataset):
    def __init__(self, paths, split='train'):
        if split == 'train':
            self.transforms = transforms.Compose([
                transforms.Resize((SIZE, SIZE),  Image.BICUBIC),
                transforms.RandomHorizontalFlip(), # A little data aug
            ])
        elif split == 'val':
            self.transforms = transforms.Resize((SIZE, SIZE),  Image.B

        self.split = split
        self.size = SIZE
        self.paths = paths

    def __getitem__(self, idx):
        img = Image.open(self.paths[idx]).convert("RGB")
        img = self.transforms(img)
        img = np.array(img)
        img_lab = rgb2lab(img).astype("float32") # Converting RGB to L
        img_lab = transforms.ToTensor()(img_lab)
        L = img_lab[[0], ...] / 50. - 1. # Between -1 and 1
        ab = img_lab[[1, 2], ...] / 110. # Between -1 and 1

        return {'L': L, 'ab': ab}

    def __len__(self):
        return len(self.paths)

def make_dataloaders(batch_size=16, n_workers=4, pin_memory=True, **kw
    dataset = ColorizationDataset(**kwargs)
    dataloader = DataLoader(dataset, batch_size=batch_size, num_worker
                            pin_memory=pin_memory)
    return dataloader
```

In [ ]:
```
train_dl = make_dataloaders(paths=train_paths, split='train')
val_dl = make_dataloaders(paths=val_paths, split='val')

data = next(iter(train_dl))
Ls, abs_ = data['L'], data['ab']
print(Ls.shape, abs_.shape)
print(len(train_dl), len(val_dl))
```

```
/usr/local/lib/python3.7/dist-packages/torchvision/transforms/transfo
rms.py:281: UserWarning: Argument interpolation should be of type Int
erpolationMode instead of int. Please, use InterpolationMode enum.
  "Argument interpolation should be of type InterpolationMode instead
of int. "
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py
:481: UserWarning: This DataLoader will create 4 worker processes in
total. Our suggested max number of worker in current system is 2, whi
ch is smaller than what this DataLoader is going to create. Please be
aware that excessive worker creation might get DataLoader running slo
w or even freeze, lower the worker number to avoid potential slowness
/freeze if necessary.
  cpuset_checked))

torch.Size([16, 1, 256, 256]) torch.Size([16, 2, 256, 256])
94 32
```

In [ ]:
```
class UnetBlock(nn.Module):
    def __init__(self, nf, ni, submodule=None, input_c=None, dropout=F
                 innermost=False, outermost=False):
        super().__init__()
        self.outermost = outermost
        if input_c is None: input_c = nf
        downconv = nn.Conv2d(input_c, ni, kernel_size=4,
                             stride=2, padding=1, bias=False)
        downrelu = nn.LeakyReLU(0.2, True)
        downnorm = nn.BatchNorm2d(ni)
        uprelu = nn.ReLU(True)
        upnorm = nn.BatchNorm2d(nf)

        if outermost:
            upconv = nn.ConvTranspose2d(ni * 2, nf, kernel_size=4,
                                        stride=2, padding=1)
            down = [downconv]
            up = [uprelu, upconv, nn.Tanh()]
            model = down + [submodule] + up
        elif innermost:
            upconv = nn.ConvTranspose2d(ni, nf, kernel_size=4,
                                        stride=2, padding=1, bias=Fals
            down = [downrelu, downconv]
            up = [uprelu, upconv, upnorm]
            model = down + up
```

```
            model = down + up
        else:
            upconv = nn.ConvTranspose2d(ni * 2, nf, kernel_size=4,
                                        stride=2, padding=1, bias=Fals
            down = [downrelu, downconv, downnorm]
            up = [uprelu, upconv, upnorm]
            if dropout: up += [nn.Dropout(0.5)]
            model = down + [submodule] + up
        self.model = nn.Sequential(*model)

    def forward(self, x):
        if self.outermost:
            return self.model(x)
        else:
            return torch.cat([x, self.model(x)], 1)

class Unet(nn.Module):
    def __init__(self, input_c=1, output_c=2, n_down=8, num_filters=64
        super().__init__()
        unet_block = UnetBlock(num_filters * 8, num_filters * 8, inner
        for _ in range(n_down - 5):
            unet_block = UnetBlock(num_filters * 8, num_filters * 8, s
        out_filters = num_filters * 8
        for _ in range(3):
            unet_block = UnetBlock(out_filters // 2, out_filters, subm
            out_filters //= 2
        self.model = UnetBlock(output_c, out_filters, input_c=input_c,

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

In [ ]:
```
class PatchDiscriminator(nn.Module):
    def __init__(self, input_c, num_filters=64, n_down=3):
        super().__init__()
        model = [self.get_layers(input_c, num_filters, norm=False)]
        model += [self.get_layers(num_filters * 2 ** i, num_filters *
                                  for i in range(n_down)] # the 'if' statement
                                                          # stride of 2 for th
        model += [self.get_layers(num_filters * 2 ** n_down, 1, s=1, n

        self.model = nn.Sequential(*model)

    def get_layers(self, ni, nf, k=4, s=2, p=1, norm=True, act=True):
        layers = [nn.Conv2d(ni, nf, k, s, p, bias=not norm)]
        if norm: layers += [nn.BatchNorm2d(nf)]
        if act: layers += [nn.LeakyReLU(0.2, True)]
        return nn.Sequential(*layers)

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

In [ ]:
```
PatchDiscriminator(3)
```

Out[19]:
```
PatchDiscriminator(
  (model): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(
1, 1))
      (1): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (1): Sequential(
      (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding
=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (2): Sequential(
      (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), paddin
g=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (3): Sequential(
      (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(1, 1), paddin
g=(1, 1), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, tra
ck_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (4): Sequential(
      (0): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), padding=
(1, 1))
    )
  )
)
```

In [ ]:
```
discriminator = PatchDiscriminator(3)
dummy_input = torch.randn(16, 3, 256, 256) # batch_size, channels, siz
out = discriminator(dummy_input)
out.shape
```

Out[20]:
```
torch.Size([16, 1, 30, 30])
```

In [ ]:
```
class GANLoss(nn.Module):
    def __init__(self, gan_mode='vanilla', real_label=1.0, fake_label=
        super().__init__()
        self.register_buffer('real_label', torch.tensor(real_label))
        self.register_buffer('fake_label', torch.tensor(fake_label))
        if gan_mode == 'vanilla':
            self.loss = nn.BCEWithLogitsLoss()
        elif gan_mode == 'lsgan':
            self.loss = nn.MSELoss()

    def get_labels(self, preds, target_is_real):
        if target_is_real:
            labels = self.real_label
        else:
            labels = self.fake_label
        return labels.expand_as(preds)

    def __call__(self, preds, target_is_real):
        labels = self.get_labels(preds, target_is_real)
        loss = self.loss(preds, labels)
        return loss
```

In [ ]:
```
class MainModel(nn.Module):
    def __init__(self, net_G=None, lr_G=2e-4, lr_D=2e-4,
                 beta1=0.5, beta2=0.999, lambda_L1=100.):
        super().__init__()

        self.device = torch.device("cuda" if torch.cuda.is_available()
        self.lambda_L1 = lambda_L1

        if net_G is None:
            self.net_G = init_model(Unet(input_c=1, output_c=2, n_down
        else:
            self.net_G = net_G.to(self.device)
        self.net_D = init_model(PatchDiscriminator(input_c=3, n_down=3
        self.GANcriterion = GANLoss(gan_mode='vanilla').to(self.device
        self.L1criterion = nn.L1Loss()
        self.opt_G = optim.Adam(self.net_G.parameters(), lr=lr_G, beta
        self.opt_D = optim.Adam(self.net_D.parameters(), lr=lr_D, beta

    def set_requires_grad(self, model, requires_grad=True):
        for p in model.parameters():
            p.requires_grad = requires_grad

    def setup_input(self, data):
        self.L = data['L'].to(self.device)
        self.ab = data['ab'].to(self.device)

    def forward(self):
```

```
        self.fake_color = self.net_G(self.L)

    def backward_D(self):
        fake_image = torch.cat([self.L, self.fake_color], dim=1)
        fake_preds = self.net_D(fake_image.detach())
        self.loss_D_fake = self.GANcriterion(fake_preds, False)
        real_image = torch.cat([self.L, self.ab], dim=1)
        real_preds = self.net_D(real_image)
        self.loss_D_real = self.GANcriterion(real_preds, True)
        self.loss_D = (self.loss_D_fake + self.loss_D_real) * 0.5
        self.loss_D.backward()

    def backward_G(self):
        fake_image = torch.cat([self.L, self.fake_color], dim=1)
        fake_preds = self.net_D(fake_image)
        self.loss_G_GAN = self.GANcriterion(fake_preds, True)
        self.loss_G_L1 = self.L1criterion(self.fake_color, self.ab) *
        self.loss_G = self.loss_G_GAN + self.loss_G_L1
        self.loss_G.backward()

    def optimize(self):
        self.forward()
        self.net_D.train()
        self.set_requires_grad(self.net_D, True)
        self.opt_D.zero_grad()
        self.backward_D()
        self.opt_D.step()

        self.net_G.train()
        self.set_requires_grad(self.net_D, False)
        self.opt_G.zero_grad()
        self.backward_G()
        self.opt_G.step()
```

In [ ]:

```
class AverageMeter:
    def __init__(self):
        self.reset()

    def reset(self):
        self.count, self.avg, self.sum = [0.] * 3

    def update(self, val, count=1):
        self.count += count
        self.sum += count * val
        self.avg = self.sum / self.count

def create_loss_meters():
    loss_D_fake = AverageMeter()
    loss_D_real = AverageMeter()
    loss_D = AverageMeter()
```

```
    loss_G_GAN = AverageMeter()
    loss_G_L1 = AverageMeter()
    loss_G = AverageMeter()

    return {'loss_D_fake': loss_D_fake,
            'loss_D_real': loss_D_real,
            'loss_D': loss_D,
            'loss_G_GAN': loss_G_GAN,
            'loss_G_L1': loss_G_L1,
            'loss_G': loss_G}

def update_losses(model, loss_meter_dict, count):
    for loss_name, loss_meter in loss_meter_dict.items():
        loss = getattr(model, loss_name)
        loss_meter.update(loss.item(), count=count)

def lab_to_rgb(L, ab):
    """
    Takes a batch of images
    """

    L = (L + 1.) * 50.
    ab = ab * 110.
    Lab = torch.cat([L, ab], dim=1).permute(0, 2, 3, 1).cpu().numpy()
    rgb_imgs = []
    for img in Lab:
        img_rgb = lab2rgb(img)
        rgb_imgs.append(img_rgb)
    return np.stack(rgb_imgs, axis=0)

def visualize(model, data, save=True):
    model.net_G.eval()
    with torch.no_grad():
        model.setup_input(data)
        model.forward()
    model.net_G.train()
    fake_color = model.fake_color.detach()
    real_color = model.ab
    L = model.L
    fake_imgs = lab_to_rgb(L, fake_color)
    real_imgs = lab_to_rgb(L, real_color)
    fig = plt.figure(figsize=(15, 8))
    for i in range(5):
        ax = plt.subplot(3, 5, i + 1)
        ax.imshow(L[i][0].cpu(), cmap='gray')
        ax.axis("off")
        ax = plt.subplot(3, 5, i + 1 + 5)
        ax.imshow(fake_imgs[i])
        ax.axis("off")
        ax = plt.subplot(3, 5, i + 1 + 10)
        ax.imshow(real_imgs[i])
```

```
        ax.imshow(real_imgs[i])
        ax.axis("off")
    plt.show()
    if save:
        fig.savefig(f"colorization_{time.time()}.png")

def log_results(loss_meter_dict):
    for loss_name, loss_meter in loss_meter_dict.items():
        print(f"{loss_name}: {loss_meter.avg:.5f}")
```

In [ ]:
```
def train_model(model, train_dl, epochs, display_every=10, save_every=
    data = next(iter(val_dl)) # getting a batch for visualizing the mo
    for e in range(epochs):
        loss_meter_dict = create_loss_meters() # function returing a d
        i = 0                                  # log the losses of the
        for data in tqdm(train_dl):
            model.setup_input(data)
            model.optimize()
            update_losses(model, loss_meter_dict, count=data['L'].size
            i += 1
            if i % display_every == 0:
                print(f"\nEpoch {e+1}/{epochs}")
                print(f"Iteration {i}/{len(train_dl)}")
                log_results(loss_meter_dict) # function to print out t
                visualize(model, data, save=False) # function displayi
        if e % save_every == 0:
          #save checkpoint every epoch
          torch.save(model.state_dict(), path+'U-Net/checkpoint_'+str(

model = MainModel()
#load a checkpoint changed from every 10 iterations to every epoch, wh
model.load_state_dict(torch.load(path+'U-Net/checkpoint_0'))
train_model(model, train_dl, 1)
```

Trying Transfer Learning

In [ ]:
```
!pip install fastai==2.4
from fastai.vision.learner import create_body
from torchvision.models.resnet import resnet18
from fastai.vision.models.unet import DynamicUnet
```

```
Requirement already satisfied: fastai==2.4 in /usr/local/lib/python3.
7/dist-packages (2.4)
Requirement already satisfied: fastprogress>=0.2.4 in /usr/local/lib/
python3.7/dist-packages (from fastai==2.4) (1.0.2)
Requirement already satisfied: fastcore<1.4,>=1.3.8 in /usr/local/lib
/python3.7/dist-packages (from fastai==2.4) (1.3.29)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3
.7/dist-packages (from fastai==2.4) (1.0.2)
```

```
Requirement already satisfied: torchvision>=0.8.2 in /usr/local/lib/p
ython3.7/dist-packages (from fastai==2.4) (0.10.1)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist
-packages (from fastai==2.4) (1.4.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/
dist-packages (from fastai==2.4) (21.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7
/dist-packages (from fastai==2.4) (3.2.2)
Requirement already satisfied: pillow>6.0.0 in /usr/local/lib/python3
.7/dist-packages (from fastai==2.4) (7.1.2)
Requirement already satisfied: spacy<4 in /usr/local/lib/python3.7/di
st-packages (from fastai==2.4) (2.2.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dis
t-packages (from fastai==2.4) (1.3.5)
Requirement already satisfied: pip in /usr/local/lib/python3.7/dist-p
ackages (from fastai==2.4) (21.1.3)
Requirement already satisfied: torch<1.10,>=1.7.0 in /usr/local/lib/p
ython3.7/dist-packages (from fastai==2.4) (1.9.1)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dis
t-packages (from fastai==2.4) (3.13)
Requirement already satisfied: requests in /usr/local/lib/python3.7/d
ist-packages (from fastai==2.4) (2.23.0)
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in /usr/local/lib/
python3.7/dist-packages (from spacy<4->fastai==2.4) (1.0.5)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7
/dist-packages (from spacy<4->fastai==2.4) (57.4.0)
Requirement already satisfied: blis<0.5.0,>=0.4.0 in /usr/local/lib/p
ython3.7/dist-packages (from spacy<4->fastai==2.4) (0.4.1)
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in /usr/local/
lib/python3.7/dist-packages (from spacy<4->fastai==2.4) (1.0.0)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/
python3.7/dist-packages (from spacy<4->fastai==2.4) (4.64.0)
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in /usr/local/lib
/python3.7/dist-packages (from spacy<4->fastai==2.4) (0.9.1)
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python
3.7/dist-packages (from spacy<4->fastai==2.4) (1.21.6)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/loca
l/lib/python3.7/dist-packages (from spacy<4->fastai==2.4) (1.0.6)
Requirement already satisfied: thinc==7.4.0 in /usr/local/lib/python3
.7/dist-packages (from spacy<4->fastai==2.4) (7.4.0)
Requirement already satisfied: plac<1.2.0,>=0.9.6 in /usr/local/lib/p
ython3.7/dist-packages (from spacy<4->fastai==2.4) (1.1.3)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/li
b/python3.7/dist-packages (from spacy<4->fastai==2.4) (3.0.6)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/
python3.7/dist-packages (from spacy<4->fastai==2.4) (2.0.6)
Requirement already satisfied: importlib-metadata>=0.20 in /usr/local
/lib/python3.7/dist-packages (from catalogue<1.1.0,>=0.0.7->spacy<4->
fastai==2.4) (4.11.3)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/
```

```
dist-packages (from importlib-metadata>=0.20->catalogue<1.1.0,>=0.0.7
->spacy<4->fastai==2.4) (3.8.0)
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local
/lib/python3.7/dist-packages (from importlib-metadata>=0.20->catalogu
e<1.1.0,>=0.0.7->spacy<4->fastai==2.4) (4.2.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/p
ython3.7/dist-packages (from requests->fastai==2.4) (2021.10.8)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/py
thon3.7/dist-packages (from requests->fastai==2.4) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.
1 in /usr/local/lib/python3.7/dist-packages (from requests->fastai==2
.4) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3
.7/dist-packages (from requests->fastai==2.4) (2.10)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/py
thon3.7/dist-packages (from matplotlib->fastai==2.4) (1.4.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0
.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->fastai
==2.4) (3.0.8)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib
/python3.7/dist-packages (from matplotlib->fastai==2.4) (2.8.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3
.7/dist-packages (from matplotlib->fastai==2.4) (0.11.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/d
ist-packages (from python-dateutil>=2.1->matplotlib->fastai==2.4) (1.
15.0)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3
.7/dist-packages (from pandas->fastai==2.4) (2022.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib
/python3.7/dist-packages (from scikit-learn->fastai==2.4) (3.1.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3
.7/dist-packages (from scikit-learn->fastai==2.4) (1.1.0)
```

In [ ]:
```
def build_res_unet(n_input=1, n_output=2, size=256):
    device = torch.device("cuda" if torch.cuda.is_available() else "cp
    body = create_body(resnet18, pretrained=True, n_in=n_input, cut=-2
    net_G = DynamicUnet(body, n_output, (size, size)).to(device)
    return net_G
```

Pretraining the generator for colorization task

In [ ]:
```
def pretrain_generator(net_G, train_dl, opt, criterion, epochs):
    for e in range(epochs):
        loss_meter = AverageMeter()
        for data in tqdm(train_dl):
            L, ab = data['L'].to(device), data['ab'].to(device)
            preds = net_G(L)
            loss = criterion(preds, ab)
            opt.zero_grad()
            loss.backward()
            opt.step()

            loss_meter.update(loss.item(), L.size(0))

        print(f"Epoch {e + 1}/{epochs}")
        print(f"L1 Loss: {loss_meter.avg:.5f}")

net_G = build_res_unet(n_input=1, n_output=2, size=256)
opt = optim.Adam(net_G.parameters(), lr=1e-4)
criterion = nn.L1Loss()
pretrain_generator(net_G, train_dl, opt, criterion, 1)
torch.save(net_G.state_dict(), path+"U-Net/res18-unet.pt")
```

```
  0%|          | 0/94 [00:00<?, ?it/s]

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py
:481: UserWarning: This DataLoader will create 4 worker processes in
total. Our suggested max number of worker in current system is 2, whi
ch is smaller than what this DataLoader is going to create. Please be
aware that excessive worker creation might get DataLoader running slo
w or even freeze, lower the worker number to avoid potential slowness
/freeze if necessary.
  cpuset_checked))

Epoch 1/1
L1 Loss: 0.08196
```

We load in pre-trained weights from

In [ ]:
```
!gdown --id 1lR6DcS4m5InSbZ5y59zkH2mHt_4RQ2KV
```

```
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarnin
g: Option `--id` was deprecated in version 4.3.1 and will be removed
in 5.0. You don't need to pass it anymore to use a file ID.
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1lR6DcS4m5InSbZ5y59zkH2mHt_4RQ2K
V (https://drive.google.com/uc?id=1lR6DcS4m5InSbZ5y59zkH2mHt_4RQ2KV)
To: /content/final_model_weights.pt
100% 136M/136M [00:00<00:00, 151MB/s]
```

In [ ]:
```
net_G = build_res_unet(n_input=1, n_output=2, size=256)
net_G.load_state_dict(torch.load(path+"U-Net/res18-unet.pt", map_locat
model = MainModel(net_G=net_G)
model.load_state_dict(torch.load("final_model_weights.pt", map_locatio
```

```
model initialized with norm initialization
```

Out[25]:
```
<All keys matched successfully>
```