Ozyegin University

Fall 2020

CS 554

Homework #4

Submitted by: M. Furkan Oruc **Student ID:** S025464

Submission Date: January 21, 2021

A Deep Encoder-Decoder Network

Parameters	Model 1	Model 2	Model 3	Model 4
Epochs	10	5	15	5
Training Size	182339	182339	182339	50000
Learning Rate	10^-3	10^-4	10^-2	10^-2
Kernel Size	3	5	5	5
Padding	1	2	2	2
Fully Connected Layers	2	4	2	2
Convolutional Layers	2*2	2*2	2*2	2*2

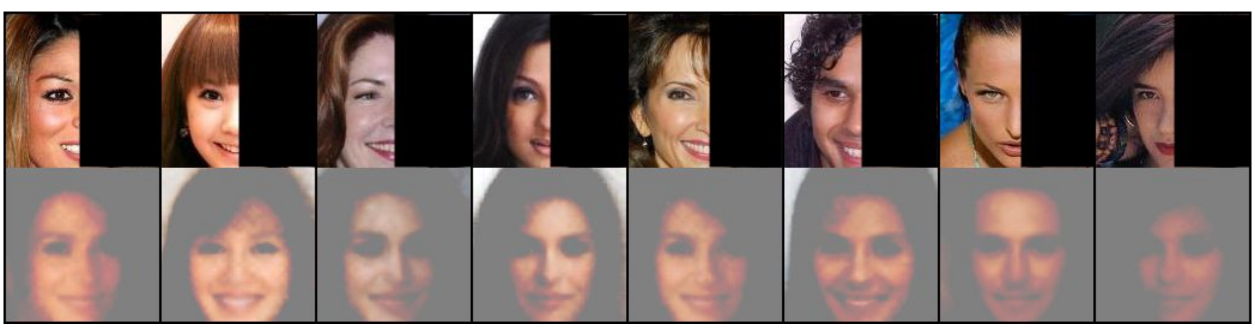
Error System

Error Formula = loss / n_batches

Loss = loss_func(output, img)

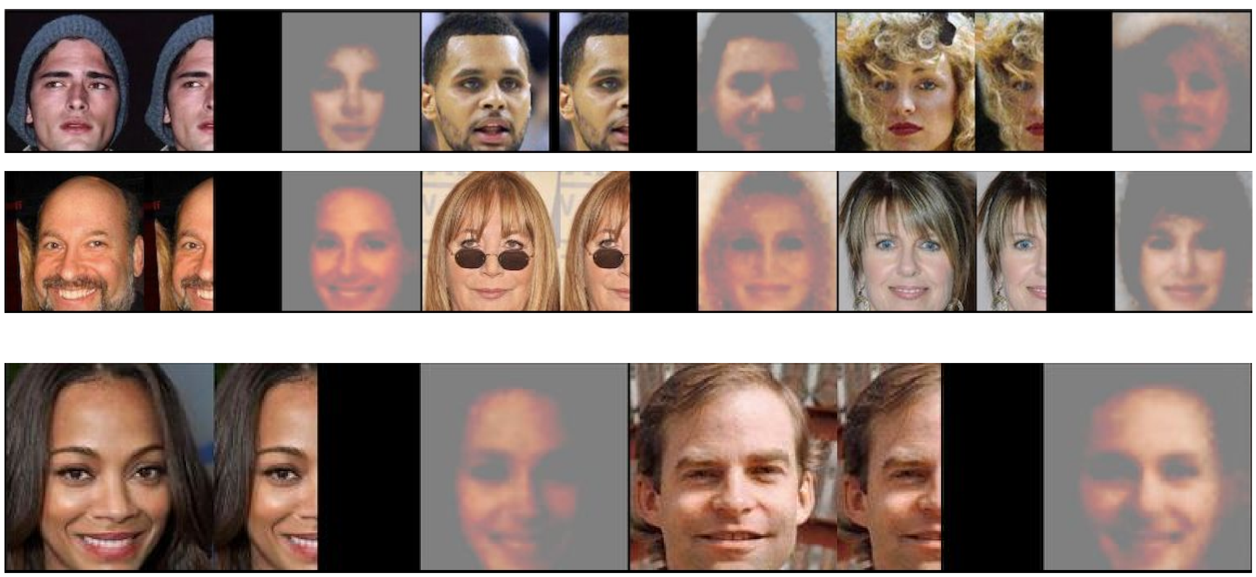
Model 1

Error Rates
0.289
0.277
0.268
0.2676
0.2674
0.2672
0.2626
0.2552
0.2550
0.2550



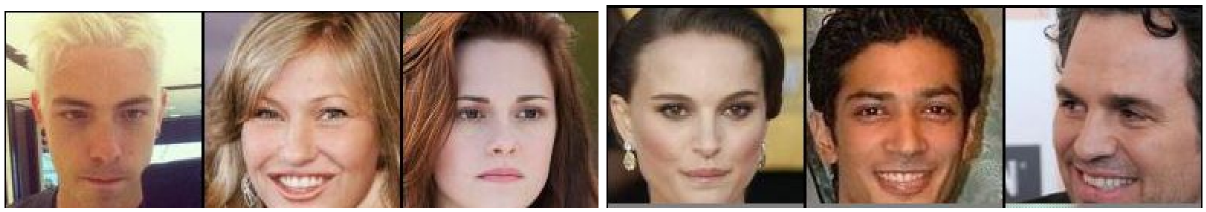
Model 2

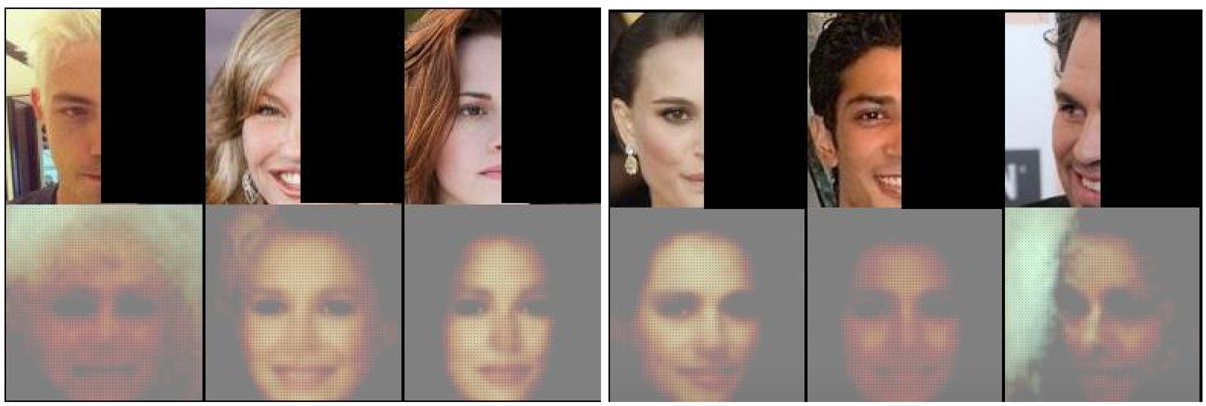
Error Rates
0.134
0.125
0.117
0.113
0.112

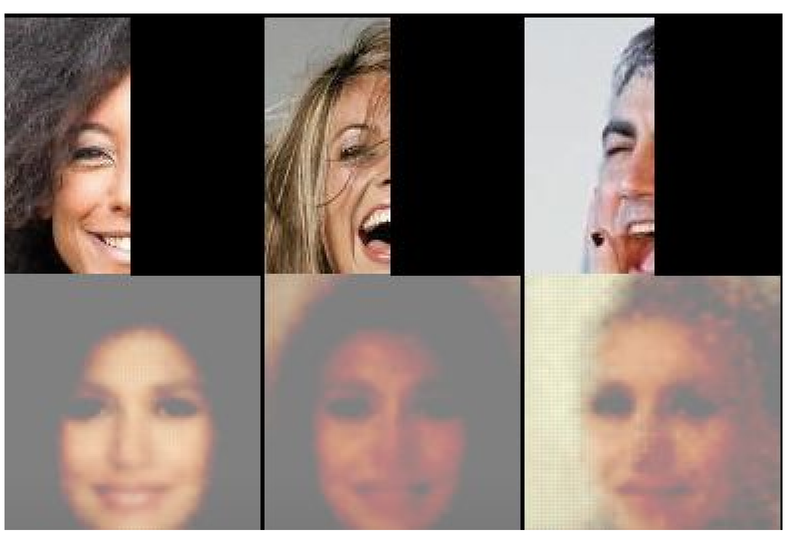


Model 3

Error rates could not be recorded due to an error occurring right after training.

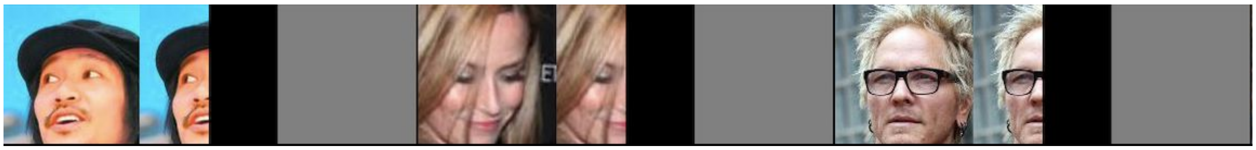


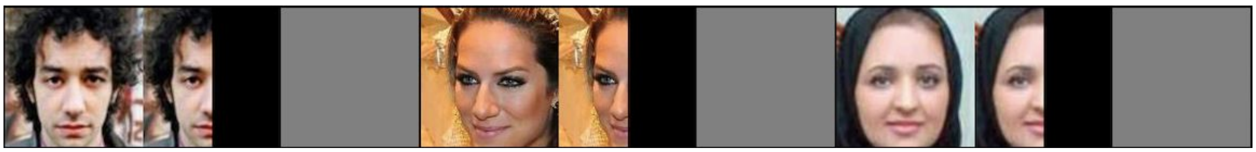


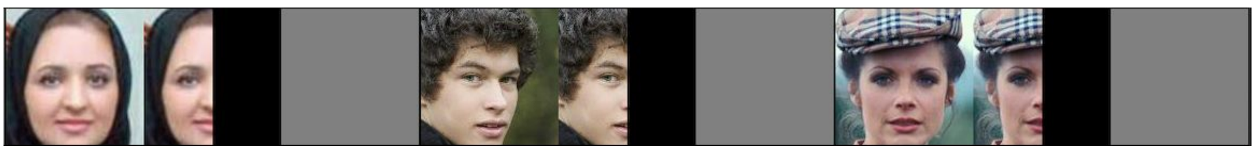


Model 4

Model 4 failed in the test process due to lack of training samples.







In conclusion, the best visible decoding outputs are provided by Model 3. Model 3 utilized 15 epochs, 5 as the kernel size and 2 as the padding. It's also important to note that Model 3 utilized learning rate as 10^-2.

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Jan 16 19:58:17 2021

@author: Furk
"""

#Homework 4 - CS554 - M. Furkan Oruc

import torch
import torchvision.datasets as dset
from torch.utils.data import DataLoader
from torchvision import transforms
from torch import nn
import torch.nn.functional as F
from torchvision.datasets import MNIST
from torchvision.utils import save_image
import matplotlib as plt
import numpy as np
import os.path
import matplotlib.pyplot as plt


import imageio

seed = 60
batch_size = 64
new_image_size = 128

# manual seed to reproduce same results
torch.manual_seed(seed)

# normalize each image and set the pixel values between -1 and 1
img_transform = transforms.Compose([
    transforms.CenterCrop(new_image_size), #Check again to manipulate based on cropped version.
    transforms.ToTensor(),
    transforms.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))
])

# prepare data loader
celebA_folder = './data/celebA/'
dataset = dset.ImageFolder(root=celebA_folder, transform=img_transform)
lengths = [int(len(dataset)*0.9), int(len(dataset)*0.1)+1] #Change Later on
# lengths = [182339, 20260]
train_set, test_set = torch.utils.data.random_split(dataset, lengths)
tr_dataloader = DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=8)
tt_dataloader = DataLoader(test_set, batch_size=batch_size, shuffle=True, num_workers=8)

#Further splitting training dataset for experimental purposes

lengths = [int(len(train_set)*0.5), int(len(train_set)*0.5)+1] #change
train_minisample, train_2_minisample = torch.utils.data.random_split(train_set, lengths)
tr_dataloader = DataLoader(train_minisample, batch_size=batch_size, shuffle=True, num_workers=8)

#print(len(train_minisample))



print(len(train_set))
print(len(test_set))

#Visualize
"""
#plt.figure(figsize=(15,10))

IMG = '/data/celebA/Img/img_align_celeba/'
```

```
for i in range(6):
    plt.subplot(2,3,i+1)
    choose_img = np.random.choice(os.listdir(IMG))
    image_path = os.path.join(IMG,choose_img)
    image = imageio.imread(image_path)
    plt.imshow(image)

"""

class AutoEncoder(nn.Module):
    def __init__(self):
        super(AutoEncoder, self).__init__()
        # ====== ENCODER PART ======
        # MNIST image is 1x28x28 (CxHxW)
        # Pytorch convolution expects input data as BxCxHxW
        # B: Batch size
        # C: number of channels gray scale images have 1 channel
        # W: width of the image
        # H: height of the image

        # use 32 3x3 filters with padding
        # padding is set to 1 so that image W,H is not changed after convolution
        # stride is 2 so filters will move 2 pixels for next calculation
        # W after conv2d  [(W - Kernelw + 2*padding)/stride] + 1
        # after convolution we'll have Bx32 14x14 feature maps (28-3+2)/2 + 1 = 14
        self.conv1 = nn.Conv2d(in_channels=3,      # 3 channels since rgb!
                               out_channels=32,   # apply 32 filters and get a feature map for each filter
                               kernel_size=3,     # filters are 3x3 weights
                               stride=2,          # halves the size of the image
                               padding=1)



        # after convolution we'll have Bx64 7x7 feature maps
        self.conv2= nn.Conv2d(in_channels=32,
                               out_channels=64,
                               kernel_size=3,
                               stride=2,
                               padding=1
                               )


        # first fully connected layer from 64*7*7=3136 input features to 16 hidden units
        self.fc1 = nn.Linear(in_features=64*32*16,
                               out_features=16)
            # first fully connected layer from 64*7*7=3136 input features to 16 hidden units
        # ====== DECODER PART ======
        self.fc2 = nn.Linear(in_features=16,
                               out_features=64*32*32)

        # 32 14x14
        # stride*(Wâ1) + â 2*padding + d*(K-1) + outpadding +1 = 2*(7-1)-2 + 2 +1 +1  =  14
        self.conv_t1 = nn.ConvTranspose2d(in_channels=64,
                                          out_channels=32,
                                          kernel_size=3,
                                          stride=2,
                                          padding=1,
                                          dilation=1,
                                          output_padding=1)



        # 1 28x28
        self.conv_t2 = nn.ConvTranspose2d(in_channels=32,
                                          out_channels=3,
                                          kernel_size=3,
                                          stride=2,
                                          padding=1,
```

```python
                                        dilation=1,
                                        output_padding=1)




    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = torch.flatten(x, start_dim=1) # flatten feature maps, Bx(C*H*W)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = x.view(-1,64,32,32) # reshape back to feature map format
        x = F.relu(self.conv_t1(x))
        x = F.relu(self.conv_t2(x))
        return x


def to_img(x):
    x = 0.5 * (x + 1)   # from [-1, 1] range to [0, 1] range
    x = x.clamp(0, 1)   # assign less than 0 to 0, bigger than 1 to 1
    x = x.view(x.size(0), 3, 128, -1) # B, C, H, W format for MNIST - Adapt to celeba.
    return x

seed = 60
num_epochs = 1 # Change Later!
#batch_size = 512
learning_rate = 1e-3
n_batches = (91000) // batch_size #Based on nuber of training samples!

# manual seed to reproduce same results
torch.manual_seed(seed)

# normalize each image and set the pixel values between -1 and 1
img_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# prepare data loader
#dataset = MNIST('./data', transform=img_transform, download=True)
#dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True, num_workers=8)

# determine where to run the code
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# create an AutoEncoder network instance
net = AutoEncoder().to(device)
# print(net)  # display the architecture
loss_function = nn.MSELoss().to(device)
optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate,
                             weight_decay=1e-5)


def to_img_cropped(x):
    x = 0.5 * (x + 1)   # from [-1, 1] range to [0, 1] range
    x = x.clamp(0, 1)   # assign less than 0 to 0, bigger than 1 to 1
    x = x.view(x.size(0), 3, 128, 64) # B, C, H, W format for celeba
    x_new = torch.zeros(x.size(0), 3, 128, 128)
    x_new[0:x.size(0), 0:3, 0:128, 0:64] = x[0:x.size(0), 0:3, 0:128, 0:64]
    return x_new


def train(net, loader, loss_func, optimizer):
    net.train()                                 # put model in train mode
    total_loss = 0
    for img, _ in loader:                       # next batch
```

```
        img = img.to(device)                         # move to gpu if available
        cropped_img = img[0:img.size(0), 0:3, 0:128, 0:64].to(device)
        noise = torch.randn(*cropped_img.shape).to(device)   # generate random noise
        noised_img = cropped_img.masked_fill(noise > 0.5, 1) # set image values at indices where noise >0.5  to 1
        output = net(cropped_img)                    # feed forward
        loss = loss_func(output, img)                # calculate loss

        optimizer.zero_grad()                        # clear previous gradients
        loss.backward()                              # calculate new gradients
        optimizer.step()                             # update weights
        total_loss += loss.item()                    # accumulate loss
    return img, cropped_img, output, total_loss



output_dir ="conv_auto_encoder_output"
losses=[]
for epoch in range(num_epochs):
    img, cropped_img, output, loss = train(net, tr_dataloader, loss_function, optimizer) #Change later as tr
    # log
    print('epoch [{}/{}], loss:{:.4f}'
          .format(epoch+1, num_epochs, loss/n_batches))
    losses.append(loss/n_batches)
    #if epoch == 1:
    pic_org = to_img(img.cpu().data)
    pic_cropped = to_img_cropped(cropped_img.cpu().data)
    #pic_noised = to_img(noised_img.cpu().data)
    pic_pred = to_img(output.cpu().data)
    res = torch.cat((pic_org,pic_cropped, pic_pred), dim=3)
    save_image(res[:8], f'{output_dir}/res_{epoch}.png')  # save 8 images

# save the model
torch.save(net.state_dict(), f'{output_dir}/conv_autoencoder_4.pth')


# show performance of autoencoder after some epochs
imgs = [plt.imread(f'{output_dir}/res_4_{i}.png') for i in range(3)]

NUM_ROWS = 1
IMGS_IN_ROW = 1
f, ax = plt.subplots(NUM_ROWS, IMGS_IN_ROW, figsize=(5,10))

for i in range(1):
    ax[i].imshow(imgs[i])
    ax[i].set_title(f'Results after {i} epoch') #Change if changed to epoch or mod epoch

plt.tight_layout()
plt.show()

#Change for the 3rd version!!


#Test
PATH_TO_MODEL = "conv_autoencoder_4.pth"
#model = net()  # Initialize model
net.load_state_dict(torch.load(PATH_TO_MODEL, map_location=torch.device('cpu')))  # Load pretrained parameters



def test(net, loader, loss_func, optimizer):
    net.eval()                              #Check for val       # put model in train mode

    total_loss = 0
    for img, _ in loader:                          # next batch
        img = img.to(device)                       # move to gpu if available
        cropped_img = img[0:img.size(0), 0:3, 0:128, 0:64].to(device)
        #noise = torch.randn(*cropped_img.shape).to(device)   # generate random noise
```

```
        #noised_img = cropped_img.masked_fill(noise > 0.5, 1) # set image values at indices where noise >0.5  to 1
        output = net(cropped_img)                    # feed forward
        #loss = loss_func(output, img)               # calculate loss

        #optimizer.zero_grad()                       # clear previous gradients
        #loss.backward()                             # calculate new gradients
        #optimizer.step()                            # update weights
        #total_loss += loss.item()                   # accumulate loss
    return img, cropped_img, output, total_loss


output_dir ="conv_auto_encoder_output"
losses=[]

img, cropped_img, output, loss = test(net, tt_dataloader, loss_function, optimizer) #Change later as tr
    # log
print("Test Results")
#losses.append(loss/n_batches)

pic_org = to_img(img.cpu().data)
pic_cropped = to_img_cropped(cropped_img.cpu().data)
#pic_noised = to_img(noised_img.cpu().data)
pic_pred = to_img(output.cpu().data)
res = torch.cat((pic_org, pic_cropped, pic_pred), dim=3)
res_2 = torch.cat((pic_org, pic_cropped, pic_pred), dim=3)
res_3 = torch.cat((pic_org, pic_cropped, pic_pred), dim=3)
save_image(res[:8], f'{output_dir}/res_yeni4_test.png')  # save 8 images - test version
save_image(res_2[:8], f'{output_dir}/res__yeni4_test_2.png')  # save 8 images - test version
save_image(res_3[:8], f'{output_dir}/res_yeni4_test_3.png')  # save 8 images - test version
```