# 5η Εργασία Deep Learning

```
1.
import pickle as pkl
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import pandas as pd
import tensorboard
from torch.utils.tensorboard import SummaryWriter
import torch
import torch.optim as optim
import torch.nn as nn
import torchvision.transforms.functional as F
from torch.utils.data import Dataset, DataLoader
import torchvision
from torchvision import transforms

"""https://www.kaggle.com/datasets/frabbisw/facial-age?resource=download"""

#-------------NETWORK-------------#

def show(imgs):
    if not isinstance(imgs, list):
        imgs = [imgs]
    fix, axs = plt.subplots(ncols=len(imgs), squeeze=False)
    for i, img in enumerate(imgs):
        img = img.detach()
        img = F.to_pil_image(img)
        axs[0, i].imshow(np.asarray(img))
        axs[0, i].set(xticklabels=[], yticklabels=[], xticks=[], yticks=[])

class Encoder(nn.Module):
    def __init__(self,in_channels,hidden):
        super(Encoder,self).__init__()

        self.enc = nn.Sequential(
                nn.Conv2d(3, hidden//16, 4,2,1),# 32x32
                nn.BatchNorm2d(hidden//16),
                nn.Conv2d(hidden//16, hidden//8, 4,2,1, bias=False), #16x16
                nn.BatchNorm2d(hidden//8),
                nn.Conv2d(hidden//8, hidden//4, 4,2,1, bias=False), #8x8
                nn.BatchNorm2d(hidden//4),
                nn.Conv2d(hidden//4, hidden//2, 4,2,1, bias=False), #4x4
                nn.BatchNorm2d(hidden//2),
                nn.Conv2d(hidden//2, hidden, 4,2,1, bias=False), #2x2
                nn.BatchNorm2d(hidden),
                nn.Conv2d(hidden, hidden, 4,2,1, bias=False), #1x1
                nn.BatchNorm2d(hidden),
            )

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


class Discriminator(nn.Module):
    def __init__(self,img_channels,features_d,num_classes,img_size):
        super(Discriminator,self).__init__()
        self.img_size = img_size
        #Input Batches X img_channels X 64 X 64
        self.disc = nn.Sequential(
            nn.Conv2d(
```

```
                img_channels+1, features_d, kernel_size=4,stride=2,padding=1
                ), #32x32  extra channel for the label from embedding
            nn.LeakyReLU(0.2),
            self._block(features_d, features_d*2, 4, 2, 1), #16x16
            self._block(features_d*2, features_d*4, 4, 2, 1),#8x8
            self._block(features_d*4, features_d*8, 4, 2, 1),#4x4

            nn.Conv2d(features_d*8, 1, 4,2,0), #1x1
            nn.Sigmoid()
            )
        self.embed = nn.Embedding(num_classes, img_size*img_size)

    def _block(self,in_channels,out_channels,kernel_size,stride,padding):
        return nn.Sequential(
            nn.Conv2d(
                in_channels,
                out_channels,
                kernel_size,
                stride,
                padding,
                bias = False
                ),
                nn.BatchNorm2d(out_channels),
                nn.LeakyReLU(0.2)
            )

    def forward(self,x,labels):
        embedding = self.embed(labels).view(labels.shape[0],1,self.img_size,self.img_size)
        x = torch.cat([x,embedding],dim = 1)
        return self.disc(x)


class Generator(nn.Module):
    def __init__(self,z_dim,img_channels,features_g,num_classes,img_size,embed_size):
        super(Generator,self).__init__()
        #input: 1x1
        self.img_size = img_size
        self.gen = nn.Sequential(
            self._block(z_dim+embed_size, features_g*16,4,1,0),#4x4
            self._block(features_g*16, features_g*8,4,2,1), #8x8
            self._block(features_g*8, features_g*4,4,2,1),  #16x16
            self._block(features_g*4, features_g*2,4,2,1),  #32x32

            nn.ConvTranspose2d(
                features_g*2, img_channels, kernel_size=4,stride =2,padding=1   #64x64
            ),
            nn.Tanh()
        )
        self.embed = nn.Embedding(num_classes,embed_size)

    def _block(self,in_channels,out_channels,kernel_size,stride,padding):
        return nn.Sequential(
            nn.ConvTranspose2d(in_channels,
                out_channels,
                kernel_size,
                stride,
                padding,
                bias=False
            ),
            nn.BatchNorm2d(out_channels),
            nn.ReLU()
        )

    def forward(self,x,label):
        #N x Noise x 1 x 1
```

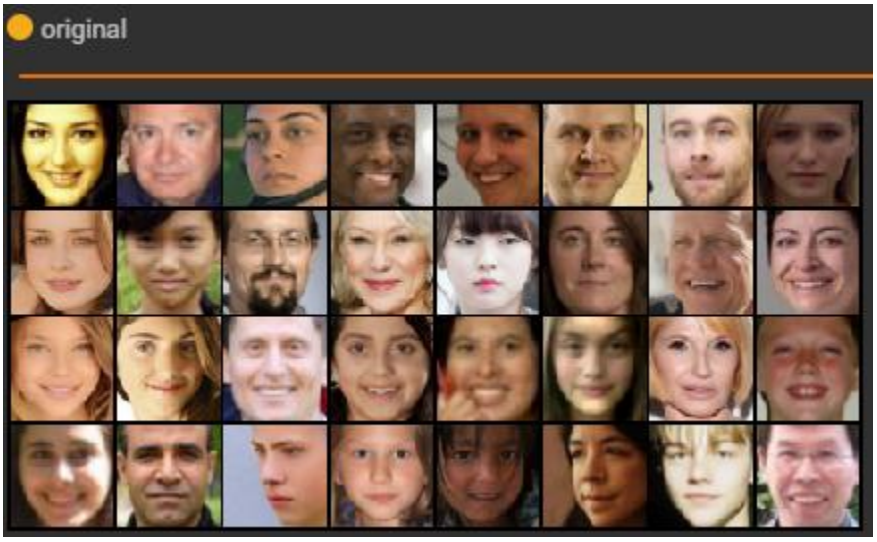
```
        embedding = self.embed(label).unsqueeze(2).unsqueeze(3)
        x = torch.cat([x,embedding],dim=1)
        return self.gen(x)

def initialize_weights(model):
    for m in model.modules():
        if isinstance(m,(nn.Conv2d,nn.ConvTranspose2d,nn.BatchNorm2d)):
            nn.init.normal_(m.weight.data,0.0,0.02)

#-------------NETWORK-------------#


#-------------DATASET-------------#


class CustomDataset(Dataset):
    def __init__(self,csv,transform = None):
        self.csv = pd.read_csv(csv)
        self.transform = transform


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

    def __getitem__(self, index):

        image = Image.open(self.csv.iloc[index].iloc[0])
        label = self.csv.iloc[index].iloc[1]
        if self.transform:
            image = self.transform(image)

        label = torch.tensor(label)

        return image,label


#-------------DATASET-------------#
BATCH_SIZE = 64
LEARNING_RATE = 2e-4
IMG_SIZE = 64
IMG_CHANNELS = 3
Z_DIM = 100
EPOCHS = 10
FEATURES_DISC = 64
FEATURES_GEN = 64
NUM_CLASSES = 6
GEN_EMBEDDING = 100
TRAIN = False
HIDDENS = 128

class_map = {0:19,
             1:29,
             2:39,
             3:49,
             4:59,
             5:69}

transform = transforms.Compose(
    [
        transforms.Resize(IMG_SIZE),
        transforms.ToTensor(),
        transforms.Normalize(
            [0.5 for _ in range(IMG_CHANNELS)], [0.5 for _ in range(IMG_CHANNELS)])
    ])    #normalize to [-1,1]
```

```
dataset = CustomDataset('image_labels2.csv',transform = transform)

loader = DataLoader(dataset,batch_size= BATCH_SIZE, shuffle=True)
gen = Generator(HIDDENS, IMG_CHANNELS, FEATURES_GEN,NUM_CLASSES,IMG_SIZE,GEN_EMBEDDING)
disc = Discriminator(IMG_CHANNELS,FEATURES_DISC,NUM_CLASSES,IMG_SIZE)
encoder = Encoder(IMG_CHANNELS,HIDDENS)

initialize_weights(gen)
initialize_weights(disc)

optimizer_gen = optim.Adam(gen.parameters(),lr = LEARNING_RATE,betas = (0.5,0.999))

optimizer_disc = optim.Adam(disc.parameters(), lr = LEARNING_RATE, betas = (0.5,0.999))

optimizer_enc = optim.Adam(encoder.parameters(),lr = LEARNING_RATE)

criterion = nn.BCELoss()
recon_loss = nn.L1Loss()
writer_real = SummaryWriter(f"logs/real")
writer_fake = SummaryWriter(f"logs/fake")
writer_enc = SummaryWriter(f"logs/recons")
writer_enc_real = SummaryWriter(f"logs/original")
writer_test = SummaryWriter(f"logs/test")
step = 0


gen.train()
disc.train()
if TRAIN:
    for epoch in range(EPOCHS):
        for batch , (real,label) in enumerate(loader):
            z = encoder(real)
            fake = gen(z,label)

            loss = recon_loss(fake,real)

            optimizer_enc.zero_grad()
            optimizer_gen.zero_grad()
            loss.backward()
            optimizer_enc.step()
            optimizer_gen.step()

            if batch % 40 == 0:
                print(
                    f"Epoch: [{epoch}/{EPOCHS}] Batch {batch}/{len(loader)}\
                        Loss G : {loss:.4f}")
                with torch.no_grad():
                    z = encoder(real)
                    fake = gen(z,label)

                    img_grid_real = torchvision.utils.make_grid(real[:32],normalize = True)

                    img_grid_fake = torchvision.utils.make_grid(fake[:32],normalize = True)


                    writer_enc_real.add_image("original",img_grid_real,global_step=step)
                    writer_enc.add_image("recons",img_grid_fake,global_step=step)

                step += 1

    for epoch in range(EPOCHS):
        for batch , (real,label) in enumerate(loader):
            recon = encoder(real)
            fake = gen(recon,label)
```

```
            #Train discriminator

            disc_real = disc(real,label).reshape(-1)
            loss_disc_real = criterion(disc_real,torch.ones_like(disc_real))

            disc_fake = disc(fake,label).reshape(-1)
            loss_disc_fake = criterion(disc_fake,torch.zeros_like(disc_fake))

            loss_disc = (loss_disc_fake + loss_disc_real) / 2

            disc.zero_grad()
            loss_disc.backward(retain_graph = True)
            optimizer_disc.step()

            #Train generator

            output = disc(fake,label).reshape(-1)
            loss_gen = criterion(output,torch.ones_like(output))
            gen.zero_grad()
            loss_gen.backward()
            optimizer_gen.step()

            if batch % 40 == 0:
                print(
                    f"Epoch: [{epoch}/{EPOCHS}] Batch {batch}/{len(loader)}\
                        loss D: {loss_disc:.4f}, Loss G : {loss_gen:.4f}")
                with torch.no_grad():
                    z = encoder(real)
                    fake = gen(z,label)

                    img_grid_real = torchvision.utils.make_grid(real[:32],normalize = True)

                    img_grid_fake = torchvision.utils.make_grid(fake[:32],normalize = True)


                    writer_real.add_image("Real",img_grid_real,global_step=step)
                    writer_fake.add_image("Fake",img_grid_fake,global_step=step)

                step += 1
    torch.save(disc.state_dict(), 'disc.pth')
    torch.save(gen.state_dict(), 'gen.pth')
    torch.save(encoder.state_dict(),'enc.pth')
else:
    encoder.load_state_dict(torch.load('enc.pth'))
    disc.load_state_dict(torch.load('disc.pth'))
    gen.load_state_dict(torch.load('gen.pth'))

    encoder.eval()
    disc.eval()
    gen.eval()

    with torch.no_grad():
        im,label = next(iter(loader))


        z = encoder(im)
        t = torch.empty(64)
        fake19 = gen(z,torch.full_like(t, 0,dtype=torch.long))
        fake29 = gen(z,torch.full_like(t, 1,dtype=torch.long))
        fake39 = gen(z,torch.full_like(t, 2,dtype=torch.long))
        fake49 = gen(z,torch.full_like(t, 3,dtype=torch.long))
        fake59 = gen(z,torch.full_like(t, 4,dtype=torch.long))
        fake69 = gen(z,torch.full_like(t, 5,dtype=torch.long))
```

```
out = torchvision.utils.make_grid(im,normalize=True)
out1 = torchvision.utils.make_grid(fake19,normalize=True)
out2 = torchvision.utils.make_grid(fake29,normalize=True)
out3 = torchvision.utils.make_grid(fake39,normalize=True)
out4 = torchvision.utils.make_grid(fake49,normalize=True)
out5 = torchvision.utils.make_grid(fake59,normalize=True)
out6 = torchvision.utils.make_grid(fake69,normalize=True)

show(out)
show(out1)
show(out2)
show(out3)
show(out4)
show(out5)
show(out6)
```

Encoder outputs:



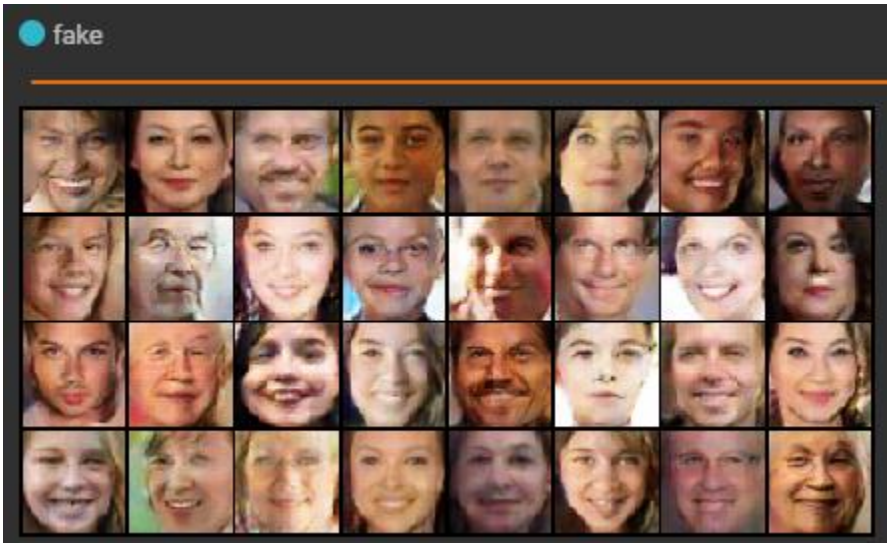
recons

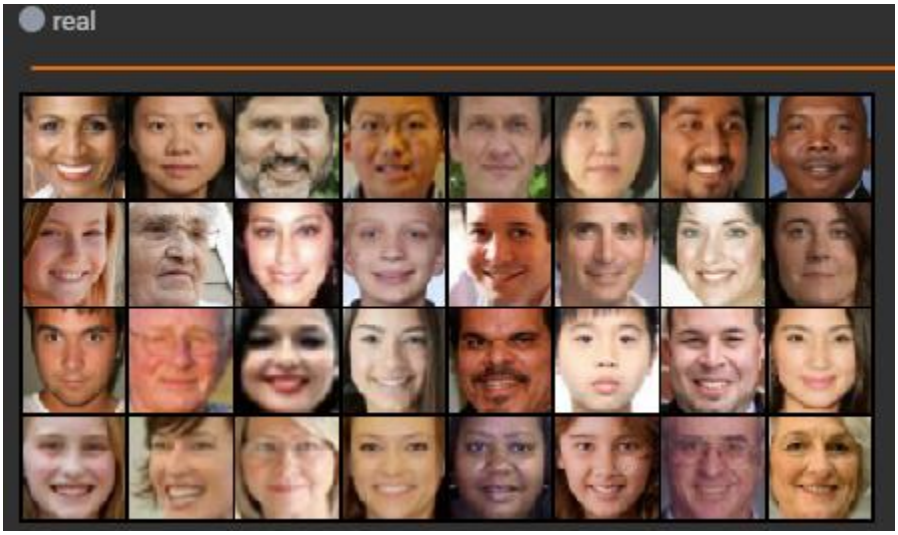
original

original

recons

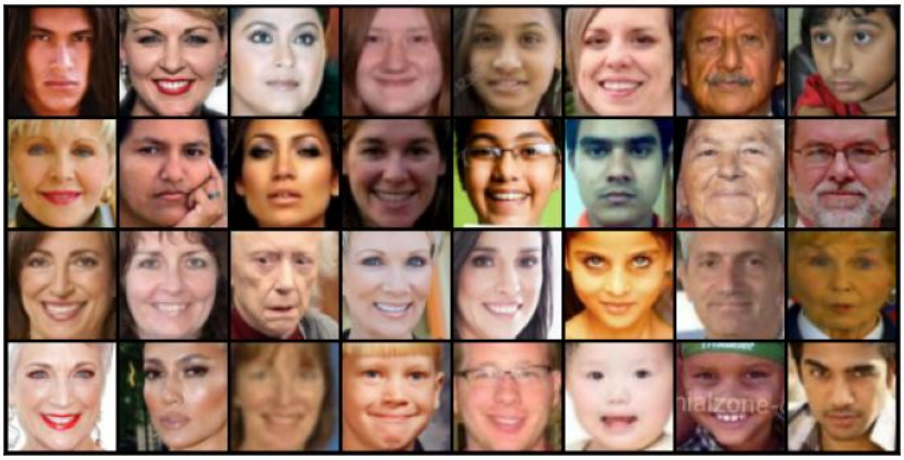
GAN outputs στο training:



# Μετατροπές:

Αρχικές εικόνες:



Μετατροπή σε ηλικία 0-19:

Μετατροπή σε 20-29:

Μετατροπή σε 30-39:

Μετατροπή σε 40-49:

Μετατροπή σε 50-59:

Μετατροπή σε 65+: