**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU**
**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA**

**Sveučilišni studij**

# Generativne suparničke mreže
# (Generative Adversarial Networks)

**Seminarski rad**

**Igor Delić**

**Osijek, 2023.**

# SADRŽAJ

# 1. UVOD

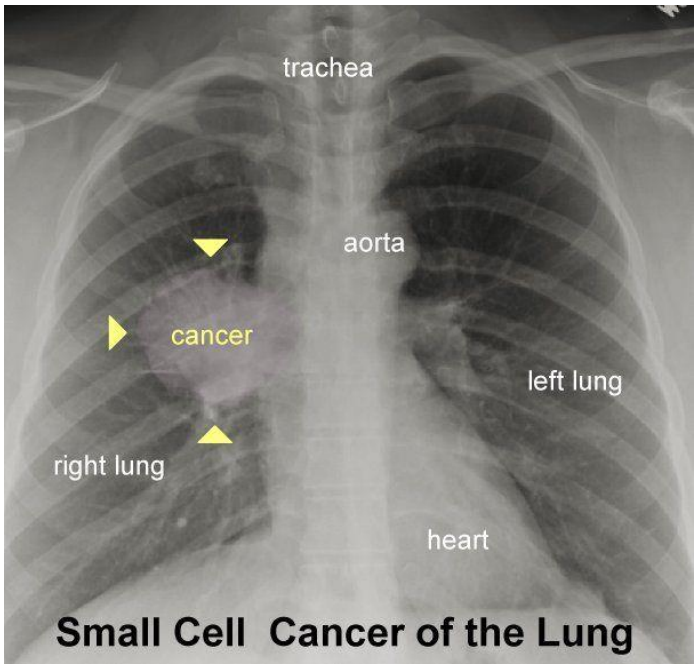
Cilj ovog seminarskog rada iz kolegija Obrada slike i računalni vid je istražiti i opisati teorijske principe rada generativnih suparničkih mreža (engl. generative adversarial networks, GANs) i testirati nekoliko različitih gotovih rješenja.

Obrada slika oduvijek je bila bolna točka umjetne inteligencije. Neuronske mreže i duboko učenje doprinijeli su znatnom poboljšanju u tom području. Neuronska mreža može se odnositi na „pravu" biološku neuronsku mrežu kao što je mreža koja se nalazi u našem mozgu ili na umjetnu neuronsku mrežu simuliranu računalom. Bez obzira na to je li biološka ili umjetna, sastoji se od velikog broja jednostavnih jedinica, neurona, koje primaju signale jedna od druge i prenose ih jedna do druge. Neuroni su jednostavni procesori informacija koji se sastoje od tijela stanice i produžetaka koji neurone međusobno povezuju. Većinu vremena ne rade ništa, odnosno miruju i čekaju da kroz produžetke stignu signali. Osnovni model umjetnog neurona sastoji se od niza prilagodljivih parametara koji se, kao i kod metode linearne i metode logističke regresije, zovu težinske vrijednosti, tj. težine. Baš kao i u metodi regresije, te se težinske vrijednosti upotrebljavaju kao multiplikatori za ulazne varijable neurona koje se zbrajaju, a zbroj težina pomnožen s ulaznim varijablama naziva se linearna kombinacija ulaznih varijabli.

Slika 1.1.

# 2. NEURONSKE MREŽE

Neuronske mreže su vrsta učenja koja se često koristi za obradu slika. One se sastoje od više slojeva neuronskih jedinica (neurona) koji su povezani između sebe, a kojima se podaci unose na ulazu, a na izlazu se dobivaju rezultati. U obradi slika, neuronske mreže se često koriste za razne zadatke, poput klasifikacije slika, detekcije objekata, segmentacije slika, generiranja slika itd. One su u stanju naučiti i izdvojiti važne značajke u slikama bez ručnog programiranja, što ih čini jednostavnijim za korištenje u usporedbi sa drugim algoritmima za obradu slika. Uz to, postoje razne vrste neuronskih mreža, kao što su konvolucijske neuronske mreže (ConvNets), rekurentne neuronske mreže (RNNs), generativne suparničke mreže (GANs) itd., koje se primjenjuju na razne zadatke u obradi slika. Neuronske mreže imaju široku primjenu i u medicini. One se koriste za razne zadatke poput klasifikacije bolesti, detekcije abnormalnosti, prognoziranja ishoda bolesti, segmentacije medicinskih slika, dijagnostike itd. Na primjer, neuronske mreže se koriste za klasifikaciju raka pluća na temelju CT slika, detekciju tuberkuloze na röntgen slikama pluća, prognoziranje razvoja Alzheimerove bolesti na temelju MR slika mozga, segmentaciju tumorskih regija u medicinskim slikama itd. Korištenje neuronskih mreža u medicini omogućuje automatizaciju i preciznost u dijagnozi i prognozi bolesti, što značajno pomaže u boljem razumijevanju i liječenju raznih bolesti.




Slika 2.1. Prikaz stanice raka

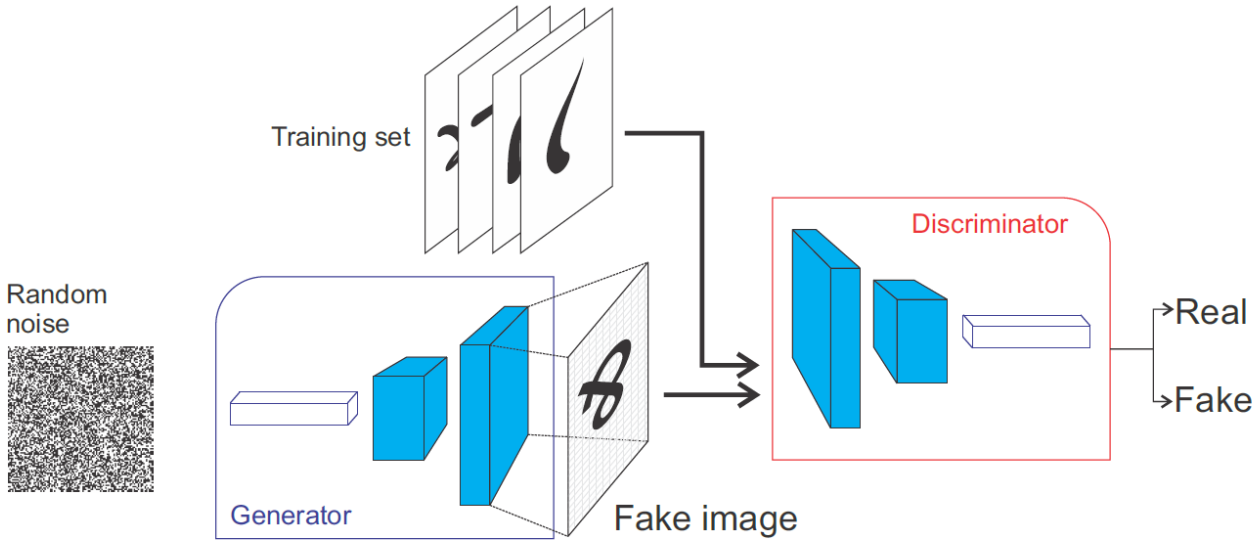
# 3. GENERATIVNE SUPARNIČKE MREŽE (GANs)

Generative Adversarial Networks (GANs) su vrsta umjetne neuronske mreže koja se koristi za generiranje slika, teksta, zvuka i drugih vrsta medija. Funkcioniraju na principu dvije umjetne neuronske mreže koje treniraju jedna protiv druge: generatora i diskriminatora.

Generator se bavi generiranjem lažnih primjera (npr. slika) koji su što realističniji. To čini tako da izvlači nasumične brojeve kao ulaz i obrađuje ih kroz višeslojnu neuronsku mrežu.

Diskriminator se bavi evaluiranjem primjera i pokušava razlikovati između stvarnih i lažnih primjera. To čini tako što obrađuje primjer kroz višeslojnu neuronsku mrežu i daje vrijednost između 0 i 1, pri čemu vrijednost 1 označava stvarni primjer, a vrijednost 0 označava lažni primjer.

Oba modela se treniraju istovremeno i poboljšavaju svoju funkciju tako što se "natječu" jedan protiv drugoga. Generator pokušava generirati slike koje će diskriminator prepoznati kao stvarne, a diskriminator pokušava razlikovati između stvarnih i lažnih primjera. Kada se GAN trenira, generator se s vremenom poboljšava u generiranju realističnih slika, a diskriminator u razlikovanju između stvarnih i lažnih slika. Konačno, generator postaje u stanju generirati vrlo realistične primjere.




Slika 3.1.

Diskriminator u GAN-u obučava se na stvarnim slikama, a generator pokušava generirati slike koje su što sličnije stvarnim slikama i koje će ih diskriminator prepoznati kao stvarne. Ovaj proces treniranja omogućuje GAN-u da uči o karakteristikama stvarnih slika i da koristi te znanja za generiranje novih, realističnih slika. GAN se često koristi za obradu slika u aplikacijama kao što su generiranje fotorealističnih slika, superrezolucija slika, poboljšanje kvalitete slika i sl.

Osim slika, GAN može se koristiti i za druge vrste podataka, kao što su tekst, zvuk i drugo, te u različite svrhe, kao što su generiranje novih slika, prekrivanje jezika, poboljšanje kvalitete slika i videozapisa i drugo.

Međutim, vrijedi napomenuti da GAN-ovi zahtijevaju visoku računalnu moć i vrijeme za treniranje, te da često zahtijevaju velike količine podataka za treniranje. Stoga, testiranje različitih GAN arhitektura i postupaka obrade slika može biti kompleksno i zahtijevati stručno znanje.

Slika 3.2.

Gore prikazane slike generirane su s pomoću generativne suparničke mreže koju je izradilo društvo NVIDIA i radi se o lažnim slikama.

Postoje nekoliko vrsta Generativnih Suparničkih Mreža (GANs), neke od njih su:

- DCGAN (Deep Convolutional Generative Adversarial Network)
- WGAN (Wasserstein Generative Adversarial Network)
- LSGAN (Least Squares Generative Adversarial Network)
- BEGAN (Boundary Equilibrium Generative Adversarial Network)
- SNGAN (Spectral Normalization Generative Adversarial Network)
- ProGAN (Progressive Growing Generative Adversarial Network)
- StyleGAN (Style-Based Generative Adversarial Network)
- CycleGAN (Cycle-Consistent Adversarial Network)

Svaka od ovih vrsta ima svoje specifične karakteristike i primjene.

DCGAN (Deep Convolutional Generative Adversarial Network) je jedna od najpopularnijih arhitektura za generativne suparničke mreže. DCGAN koristi konvulzione slojeve u generatoru i diskriminatoru umjesto potpuno povezanih slojeva, što povećava njihovu efikasnost i sposobnost generiranja visokokvalitetnih slika. DCGAN se često koristi u raznim aplikacijama uključujući generiranje slika, sintezu slika, video generiranje, i sl. To je popularno zbog svoje jednostavne arhitekture, visokokvalitetnih rezultata i efikasnosti u procesu učenja.

WGAN (Wasserstein Generative Adversarial Network) je jedna od najnovijih arhitektura za generativne suparničke mreže. WGAN razlikuje se od drugih GAN arhitektura po tome što koristi Wassersteinovu distribuciju kao funkciju gubitka, umjesto tradicionalne funkcije gubitka kao što je binary cross-entropy. Wassersteinova distribucija koristi pojam Wassersteinove udaljenosti između dvije distribucije, što omogućava precizniji praćenje razlike između stvarne i generirane distribucije. To rezultira u boljim rezultatima generiranja, posebno kada su slike složene i sa šumom. WGAN također ima i neke tehničke prednosti, uključujući smanjenje problema mode collapse (kada generator generira iste slike), i smanjenje problema "trenutka nagle stabilnosti" (kada diskriminator postane prejak, što onemogućava generatoru da se dalje razvija).

LSGAN (Least Squares Generative Adversarial Network) se razlikuje od drugih GAN arhitektura po tome što koristi funkciju gubitka zasnovanu na najmanjim kvadratima, umjesto tradicionalne funkcije gubitka kao što je binary cross-entropy. Funkcija gubitka zasnovana na najmanjim kvadratima koristi se za praćenje razlike između stvarne i generirane distribucije, što omogućava bolje praćenje razlike u regiji nule, što često rezultira u boljim rezultatima generiranja. Međutim, zbog specifičnosti funkcije gubitka zasnovane na najmanjim kvadratima, LSGAN može imati neke tehničke nedostatke, kao što su nedostatak balansa između generatora i diskriminatora i teškoće u konvergenciji u nekim slučajevima.

BEGAN (Boundary Equilibrium Generative Adversarial Network) koristi koncept ravnoteže granice kako bi automatski prilagodio brzinu učenja i kontrolirao konvergenciju mreže. BEGAN koristi funkciju gubitka koja se temelji na korelaciji između ulazne slike i rekonstruirane slike, kako bi se osigurala ravnoteža između generatora i diskriminatora. Ova funkcija gubitka omogućuje automatsko prilagodavanje brzine učenja kako bi se osiguralo da se generator i diskriminator ne "odbijaju" jedan od drugoga, što često rezultira u boljim rezultatima generiranja.. Međutim, BEGAN ima neke tehničke izazove kao što su složenost implementacije i zahtjev za velikim brojem podataka za treniranje mreže. Također, BEGAN može imati neke teškoće s konvergencijom u nekim slučajevima, zbog čega su potrebne dodatne mjere kako bi se osiguralo da mreža radi pravilno.

SNGAN (Spectral Normalization Generative Adversarial Network) je jedna od varijanti GAN-a koja koristi tehniku normalizacije spektralne radijacije kako bi izbjegla problem nestabilnosti u učenju. Ova normalizacija radijacije se primjenjuje na konvolutivne slojeve u generatoru i diskriminatoru, što rezultira konvergencijom bržom i stabilnijom u odnosu na druge varijante GAN-a. SNGAN je također poznat po visokoj kvaliteti generiranih slika.

ProGAN (Progressive Growing Generative Adversarial Network) je GAN arhitektura koja koristi napredno koncept učenja pomoću progresivnog rasta. Ideja ovog pristupa je da se trenira mreža postepeno, počevši od jednostavnijih verzija generatora i diskriminatora, a kasnije dodavanjem slojeva i povećanjem njihove veličine. Ovo postepeno povećavanje složenosti omogućava brži i stabilniji razvoj modele. ProGAN se često koristi za generiranje visoko rezolucijskih slika i njegove primjene uključuju stvaranje fotorealističnih slika, animiranih likova itd.

StyleGAN (Style-Based Generative Adversarial Network) je generativna suparnička mreža koju je razvio Timo Süvegec i njegovi kolege u OpenAI. Ovaj model je posebno dizajniran za generiranje realističnih slika lica. StyleGAN se razlikuje od drugih generativnih mreža tako što koristi koncept "stilova" kako bi se razlikovali elementi slike koji određuju osobine slike poput oblika, boje, teksture i sl. StyleGAN također uključuje složeniju arhitekturu koja omogućuje izvođenje stilova slika iz složenijih slika. Ovaj model je dobio veliku pažnju zbog svoje sposobnosti generiranja visoko kvalitetnih slika lica s velikom varijabilnošću, što ga čini popularnim u mnogim različitim primjenama poput računalne grafike, video igara, kreativne umjetnosti i sl.

CycleGAN (Cycle-Consistent Adversarial Network) je generativna suparnička mreža koja se koristi za rješavanje problema prebacivanja stilova među različitim skupovima podataka. Na primjer, CycleGAN može pretvoriti slike mačaka u slike pasa i obratno. CycleGAN koristi dvije mreže: generator i diskriminator, kako bi se izvršila pretvorba. Generator stvara novu sliku iz ulazne slike, a diskriminator pokušava utvrditi je li nova slika prirodna ili ne. CycleGAN također uključuje koncept "ciklus-konzistentnosti" kako bi se osiguralo da nova slika može biti ponovno pretvorena u izvornu sliku bez gubitka kvalitete. CycleGAN je široko korišten u mnogim primjenama poput slikovne obrade, kreativne umjetnosti, strojnog učenja i sl, zbog svoje sposobnosti pretvaranja stilova bez potrebe za velikim skupovima podataka s označenjima.

# 4. PRIMJER

U ovom primjeru implementiramo Deep Convolutional GAN(DCGAN) i koristimo slike (bez oznaka razreda) iz skupa podataka CIFAR_10. Ove slike zajedno s lažnim slikama bit će predane diskriminatoru u paketima. Pogledajmo korake koje će naš GAN slijediti:

```
# Load CIFAR10 data
(X_train, y_train), (_, _) = keras.datasets.cifar10.load_data()

# Select a single class images (birds)
X_train = X_train[y_train.flatten() == 2]
```

```
# Input shape
img_rows = 32
img_cols = 32
channels = 3

img_shape = (img_rows, img_cols, channels)
latent_dim = 100
```

Slika 4.1. Prikaz koda

Navedeni kod služi za učitavanje skupa podataka CIFAR10 i odabir samo jedne klase slika (ptica) za treniranje. Potom se postavljaju varijable za oblik slika (širina, visina i broj kanala) i za dimenziju skrivenog prostora (latent_dim).

```
def build_generator():

        model = Sequential()

        model.add(Dense(128 * 8 * 8, activation="relu", input_dim=latent_dim))
        model.add(Reshape((8, 8, 128)))

        model.add(UpSampling2D())#upsamples to 16*16*128

        model.add(Conv2D(128, kernel_size=3, padding="same"))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Activation("relu"))

        model.add(UpSampling2D()) #upsamples to 32*32*128

        model.add(Conv2D(64, kernel_size=3, padding="same"))
        model.add(BatchNormalization(momentum=0.8))
        model.add(Activation("relu"))

        model.add(Conv2D(channels, kernel_size=3, padding="same"))
        model.add(Activation("tanh"))

        #outputs an image of 32*32*3

        noise = Input(shape=(latent_dim,))
        img = model(noise)

        return Model(noise, img)
```

Slika 4.2. Prikaz koda

Ova funkcija gradi generator u Deep Convolutional GAN (DCGAN) arhitekturi. Generator se sastoji od nekoliko slojeva koji povećavaju rezoluciju izlazne slike smanjivanjem dimenzija skrivenih slojeva. Na kraju se koristi tangens hiperbolni (tanh) aktivacijski sloj kako bi se dobila slika u rasponu od -1 do 1. Funkcija vraća model koji prima ulazni šum latent_dim dimenzija i vraća generiranu sliku.

```
def build_discriminator():

        model = Sequential()

        model.add(Conv2D(32, kernel_size=3, strides=2, input_shape=img_shape, padding="same"))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dropout(0.25))
        #no normalization for the first layer

        model.add(Conv2D(64, kernel_size=3, strides=2, padding="same"))
        model.add(ZeroPadding2D(padding=((0,1),(0,1))))
        model.add(BatchNormalization(momentum=0.8))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dropout(0.25))

        model.add(Conv2D(128, kernel_size=3, strides=2, padding="same"))
        model.add(BatchNormalization(momentum=0.8))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dropout(0.25))

        model.add(Conv2D(256, kernel_size=3, strides=1, padding="same"))
        model.add(BatchNormalization(momentum=0.8))
        model.add(LeakyReLU(alpha=0.2))
        model.add(Dropout(0.25))

        model.add(Flatten())
```

```
        model.add(Flatten())
        model.add(Dense(1, activation='sigmoid'))



        img = Input(shape=img_shape)
        validity = model(img)

        return Model(img, validity)
```

Slika 4.3. Prikaz koda

Ova funkcija stvara diskriminator, odnosno neuronsku mrežu koja prima sliku kao ulaz i vraća vrijednost između 0 i 1, gdje vrijednost 1 označava da je ulazna slika stvarna, a vrijednost 0 označava da je ulazna slika generirana. Diskriminator se sastoji od konvolucijskih slojeva, slojeva

normalizacije po grupama i gustih slojeva. Slojevi konvolucije uključuju LeakyReLU aktivacijsku funkciju, a slojevi gustine koriste sigmoidnu aktivacijsku funkciju.

```
# Build and compile the discriminator
discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy',
            optimizer=Adam(0.0002,0.5),
            metrics=['accuracy'])

# Build the generator
generator = build_generator()

# The generator takes noise as input and generates imgs
z = Input(shape=(latent_dim,))
img = generator(z)

# For the combined model we will only train the generator
discriminator.trainable = False

# The discriminator takes generated images as input and determines validity
valid = discriminator(img)

# The combined model  (stacked generator and discriminator)
# Trains the generator to fool the discriminator
combined = Model(z, valid)
combined.compile(loss='binary_crossentropy', optimizer=Adam(0.0002,0.5))
```

Slika 4.4. Prikaz koda

U ovom dijelu koda stvara se diskriminator i generator mreža. Nakon toga, spaja se diskriminator i generator tako da generator stvara slike koje diskriminator pokušava klasificirati kao prave ili lažne. Konačno, definira se složeni model koji povezuje generator i diskriminator, ali tijekom učenja, diskriminator nije podložan učenju jer se generiraju slike i provjerava se njihova autentičnost diskriminatorom.

```
def show_imgs(epoch):
        r, c = 4,4
        noise = np.random.normal(0, 1, (r * c,latent_dim))
        gen_imgs = generator.predict(noise)

        # Rescale images 0 - 1
        gen_imgs = 0.5 * gen_imgs + 0.5

        fig, axs = plt.subplots(r, c)
        cnt = 0
        for i in range(r):
            for j in range(c):
                axs[i,j].imshow(gen_imgs[cnt, :,:,])
                axs[i,j].axis('off')
                cnt += 1
        plt.show()
        plt.close()
```

Slika 4.5. Prikaz koda

Ova funkcija prikazuje slike generirane od strane GAN generatora za zadani broj epoha. U svakoj epohi se generira mreža šuma i prosljeđuje generatoru kako bi se stvorile nove slike. Generirane slike se skaliraju na raspon od 0 do 1 i prikazuju u obliku mreže s 4 retka i 4 stupca. Nakon što su slike prikazane, prozor se zatvara.

```
epochs=30000
batch_size=32
display_interval=5000
losses=[]

#normalizing the input
X_train = X_train / 127.5 - 1.


        # Adversarial ground truths
valid = np.ones((batch_size, 1))
        #let's add some noise
valid += 0.05 * np.random.random(valid.shape)
fake = np.zeros((batch_size, 1))
fake += 0.05 * np.random.random(fake.shape)
```

Slika 4.6. Prikaz koda

U ovom kodu definiraju se hiperparametri (broj epoha, veličina grupe, učestalost prikaza, gubitci) i normalizira ulazni skup podataka X_train. Zatim se stvaraju oznake koje će se koristiti u procesu treniranja generatora i diskriminatora. Oznake za istinske primjere (valid) se postavljaju na 1, dok se oznake za lažne primjere (fake) postavljaju na 0. Također, dodaje se šum na oznake kako bi se izbjeglo prenaučivanje i poboljšala stabilnost procesa treniranja.

```
#  Train Discriminator


# Select a random half of images
idx = np.random.randint(0, X_train.shape[0], batch_size)
imgs = X_train[idx]

# Sample noise and generate a batch of new images
noise = np.random.normal(0, 1, (batch_size, latent_dim))
gen_imgs = generator.predict(noise)


# Train the discriminator (real classified as ones and generated as zeros)
d_loss_real = discriminator.train_on_batch(imgs, valid)
d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
```

```
#  Train Generator


# Train the generator (wants discriminator to mistake images as real)
g_loss = combined.train_on_batch(noise, valid)

# Plot the progress
if epoch % 5000==0:
    print ("%d [D loss: %f] [G loss: %f]" % (epoch, d_loss[0], g_loss))
if epoch % 1000==0:
    losses.append((d_loss[0],g_loss))

if epoch % display_interval == 0:
     show_imgs(epoch)
```

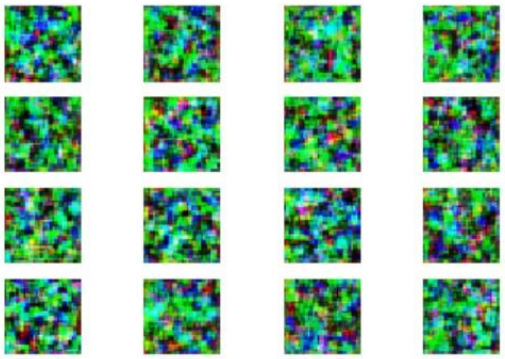
Slika 4.7. Prikaz koda

U koraku treniranja diskriminatora, uzorci stvarnih slika se slučajno odabiru, a za generiranje novih slika koristi se šum koji je generiran iz normalne distribucije. Zatim se diskriminator trenira tako da razlikuje između stvarnih i generiranih slika.

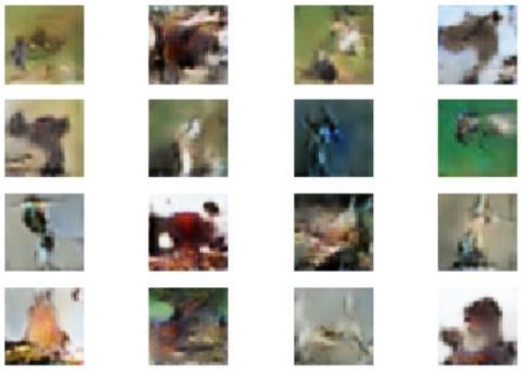
U koraku treniranja generatora, generiraju se nove slike iz šuma i generator se trenira da prevari diskriminator tako da ih klasificira kao stvarne.

Na kraju svake epohe, ispisuju se gubitci i slike koje je generirao generator. Također, gubitci se spremaju u popis za kasniju vizualizaciju.

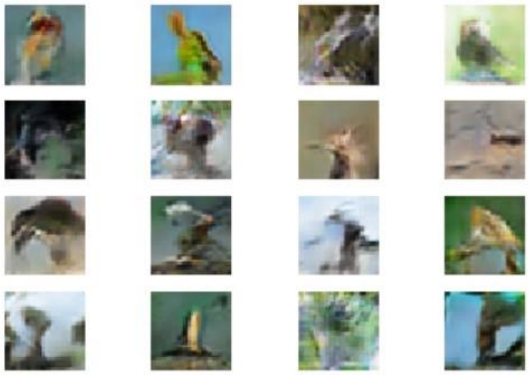
```
0 [D loss: 1.112700] [G loss: 0.501442]
```



```
10000 [D loss: 0.569980] [G loss: 0.891311]
```



```
25000 [D loss: 0.248151] [G loss: 1.693956]
```



Slika 4.8. Prikaz gubitaka i slika koje je generirao generator

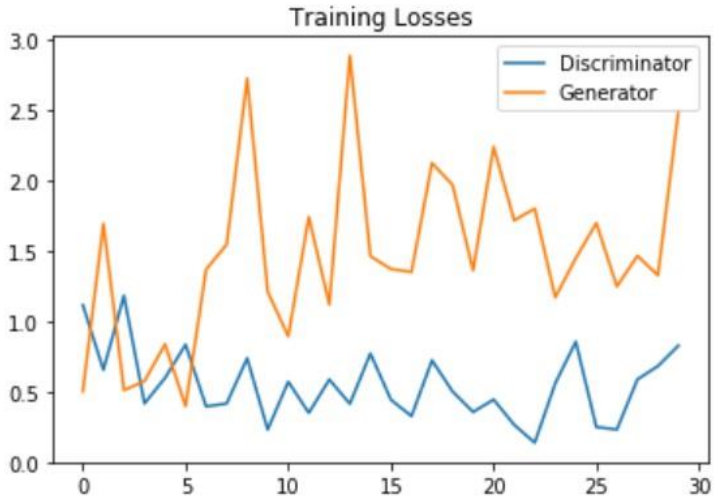
```
def show_losses(losses):
    losses = np.array(losses)

    fig, ax = plt.subplots()
    plt.plot(losses.T[0], label='Discriminator')
    plt.plot(losses.T[1], label='Generator')
    plt.title("Training Losses")
    plt.legend()
    plt.show()
```

Slika Slika 4.9. Prikaz koda

Funkcija "show_losses" prikazuje gubitke (losses) koji se koriste tijekom treniranja GAN mreže. Ulaz u funkciju je niz (array) koji sadrži gubitke po epohama (vremenskim periodima u kojima se obavi cijeli prolazak kroz skup podataka), a zatim se ti gubici prikazuju na grafu.

```
show_losses(losses)
```



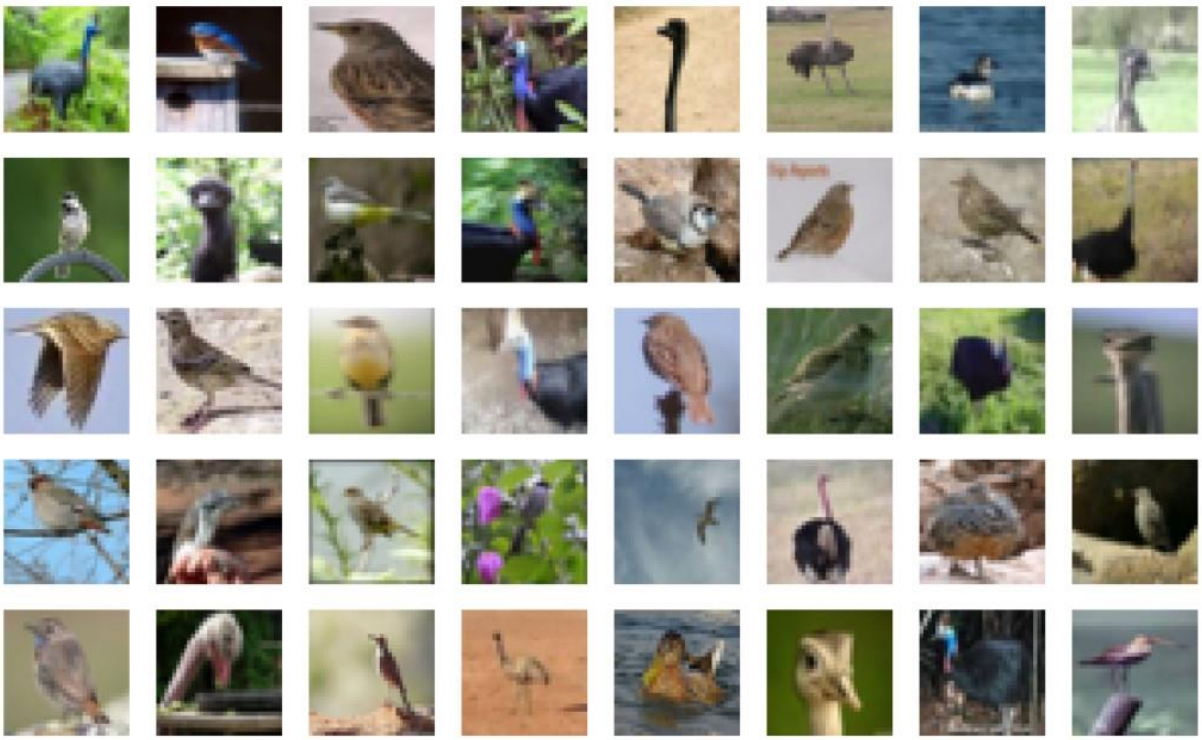

Slika 4.10. Graf funkcije gubitaka

```
s=X_train[:40]
s = 0.5 * s + 0.5
f, ax = plt.subplots(5,8, figsize=(16,10))
for i, img in enumerate(s):
        ax[i//8, i%8].imshow(img)
        ax[i//8, i%8].axis('off')

plt.show()
```

Slika 4.11. Prikaz koda

Ovaj kod prikazuje prvih 40 slika iz skupa za treniranje. Slike su prethodno normalizirane i potrebno ih je vratiti u njihov izvorni raspon vrijednosti skaliranjem.



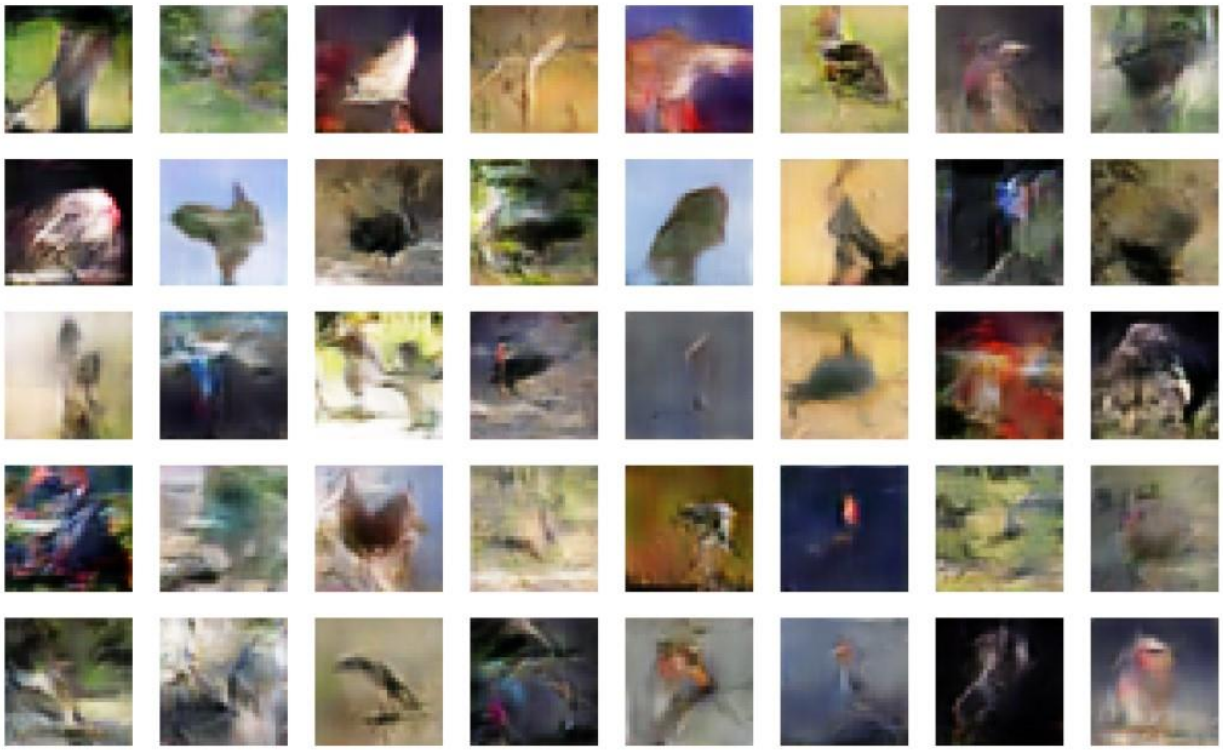
Slika 4.12. Prikaz prvih 40 slika iz skupa za treniranje

```
noise = np.random.normal(size=(40, latent_dim))
generated_images = generator.predict(noise)
generated_images = 0.5 * generated_images + 0.5
f, ax = plt.subplots(5,8, figsize=(16,10))
for i, img in enumerate(generated_images):
        ax[i//8, i%8].imshow(img)
        ax[i//8, i%8].axis('off')

plt.show()
```

Slika 4.13. Prikaz koda

Ovaj kod generira 40 slika koristeći model generatora naučen u GAN (Generative Adversarial Networks) modelu.



Slika 4.14. Prikaz 40 slika koristeći model generatora

# 5. ZAKLJUČAK

Generativne suparničke mreže (GAN) su vrsta neuronskih mreža koja može generirati nove podatke slične onima u skupu podataka kojima je trenirana. GAN-ovi se sastoje od dva dijela: generatora i diskriminatora. Generator generira nove podatke, a diskriminator procjenjuje jesu li ti podaci autentični ili generirani. Oba dijela su trenirana zajedno u procesu učenja. Primjena GAN-ova uključuje generiranje umjetnih slika, videozapisa, zvukova i drugih vrsta podataka. U ovom razgovoru, prikazan je primjer implementacije GAN-a za generiranje slika ptica iz skupa podataka CIFAR-10. Ove slike su daleko od savršenih i mogu se poboljšati dodatnim treniranjem, ali su prilično izvanredne s obzirom na činjenicu da su generirane iz ničega (odnosno slučajnog šuma), ali i dalje lošije od ranije navedenih slika generiranih s pomoću generativne suparničke mreže koju je izradilo društvo NVIDIA.

# LITERATURA


- https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/
- https://developers.google.com/machine-learning/gan/gan_structure
- https://wiki.pathmind.com/generative-adversarial-network-gan
- https://course.elementsofai.com/hr/5/1