Modeli za predstavljanje slike i videa

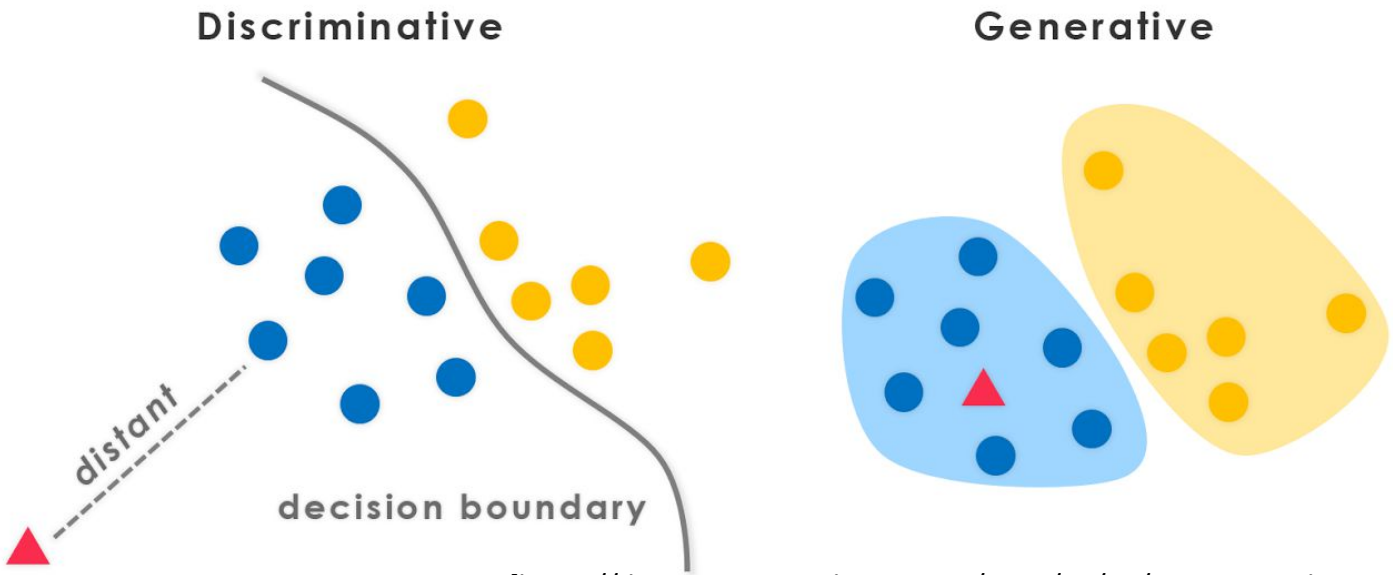
Generativni suparnički modeli

Generative Adversarial Networks – GANs

Tomislav Hrkać

Generativni vs. diskriminativni modeli

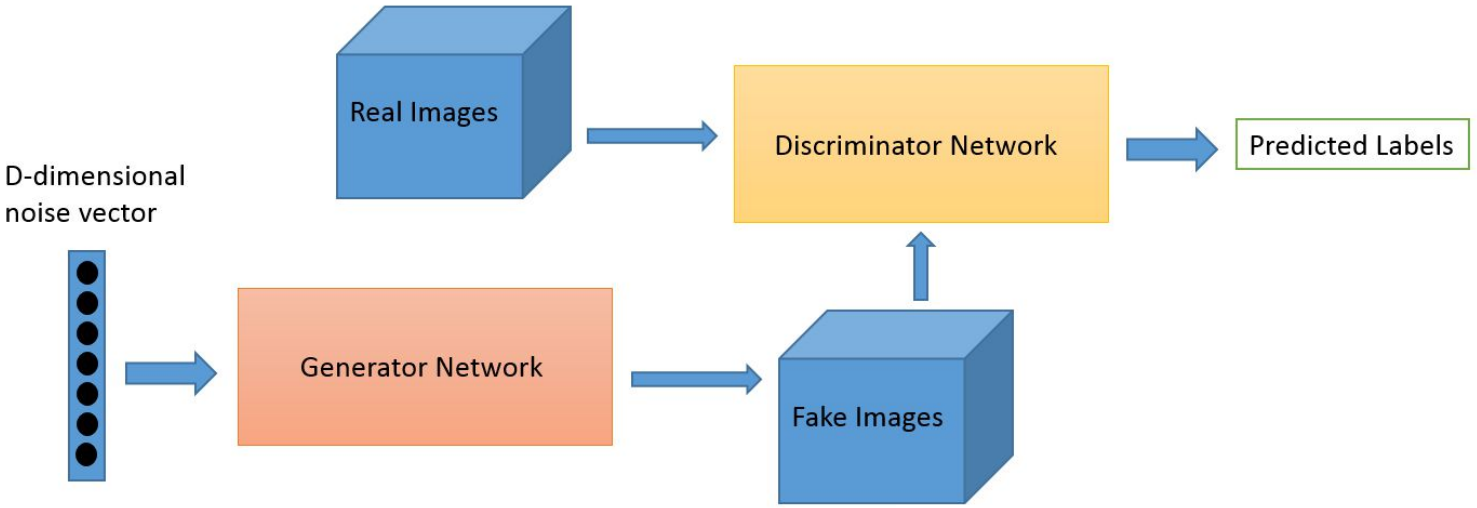
- Diskriminativni model
  - P(Y|X), fokus na decizijsku granicu
  - Nadzirano učenje, tipična primjena - klasifikator
- Generativni model
  - P(Y,X) ili P(X|Y)P(Y), fokus na distribuciju podataka
  - Nenadzirano učenje, može se također koristiti za klasifikaciju ,ali i za generiranje novih primjera sličnih stvarnima



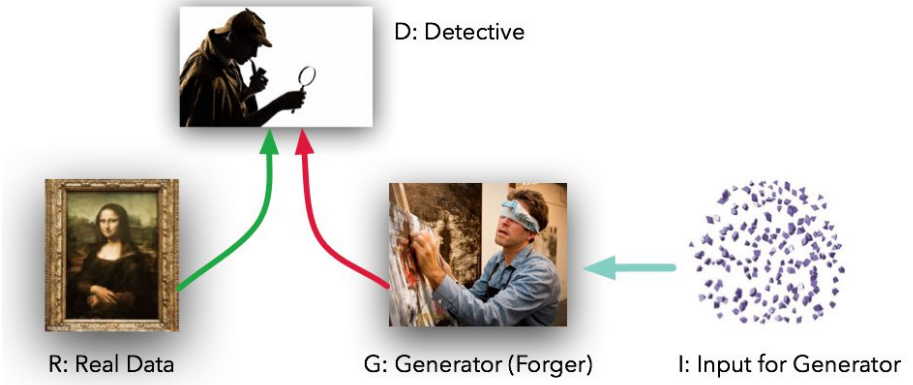
[https://datawarrior.wordpress.com/2016/05/08/generative-discriminative-pairs/]

Generativni suparnički modeli

- Goodfellow et al., Generative Adversarial Nets, NIPS 2014
- Osnovna ideja:

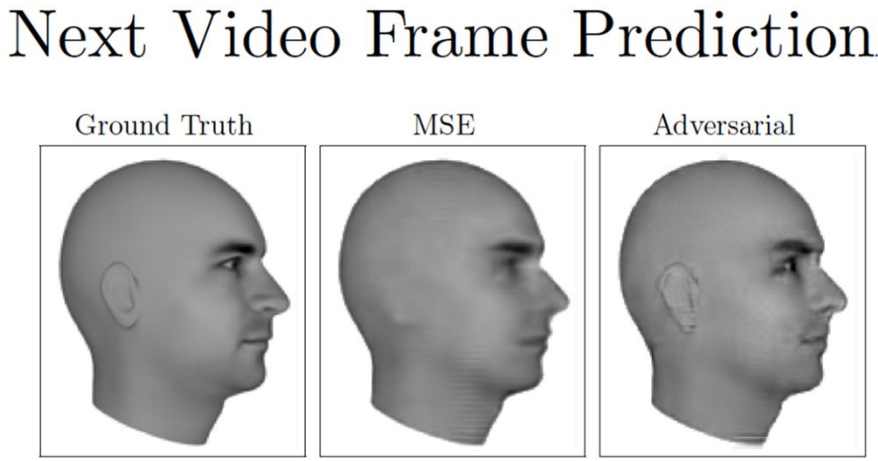


[https://deeplearning4j.org/generative-adversarial-network]



Generativni suparnički modeli

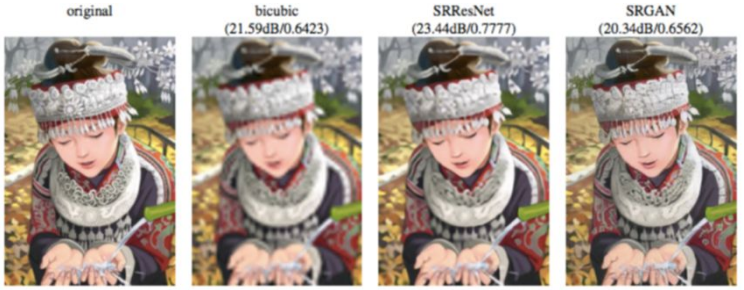
- Zašto proučavati generativne modele? [1/2]
  - Izvrstan test naše sposobnosti korištenja visokodimenzionalnih i složenih distribucija vjerojatnosti (važno u primijenjenoj matematici i inženjerstvu)
  - Simuliranje mogućih budućih stanja za planiranje i potporno učenje
  - Mogućnost treniranja uz nedostajuće podatke (polunadzirano učenje)
  - Mogućnost rada s višemodalnim izlazima



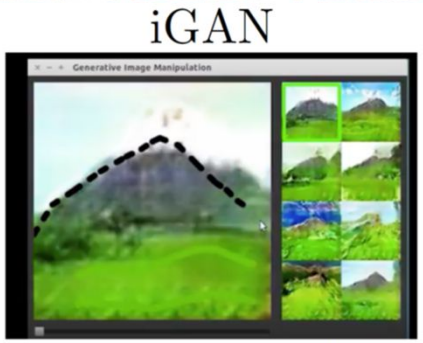
Generativni suparnički modeli

- Zašto proučavati generativne modele? [2/2]
  - Zadatci koji zahtijevaju generiranje realističnih uzoraka

- Generiranje slike u višoj rezoluciji (single image superresolution)

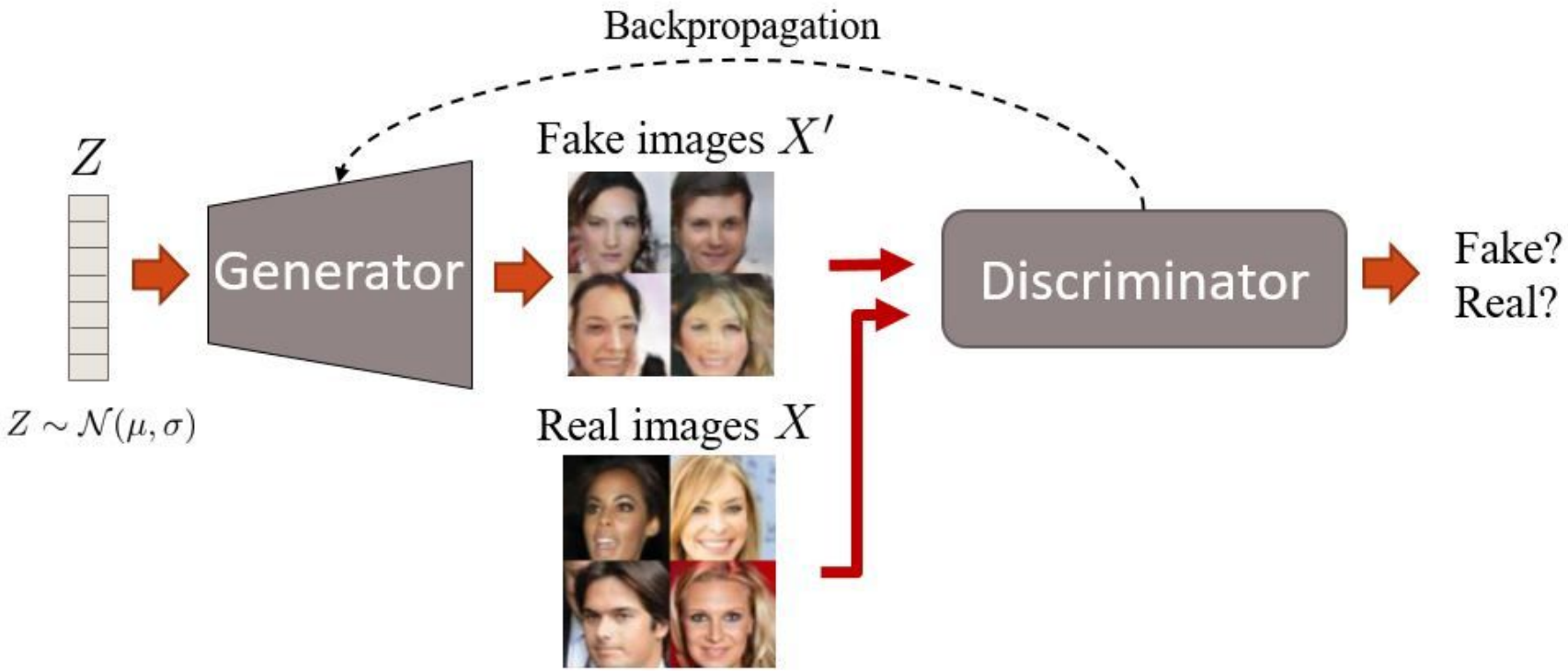


- Pomoć u generiranju umjetničkih crteža/slika



- Prevođenje između slika različitih tipova (npr. zračna snimka u mapu) ("Image to image translation")

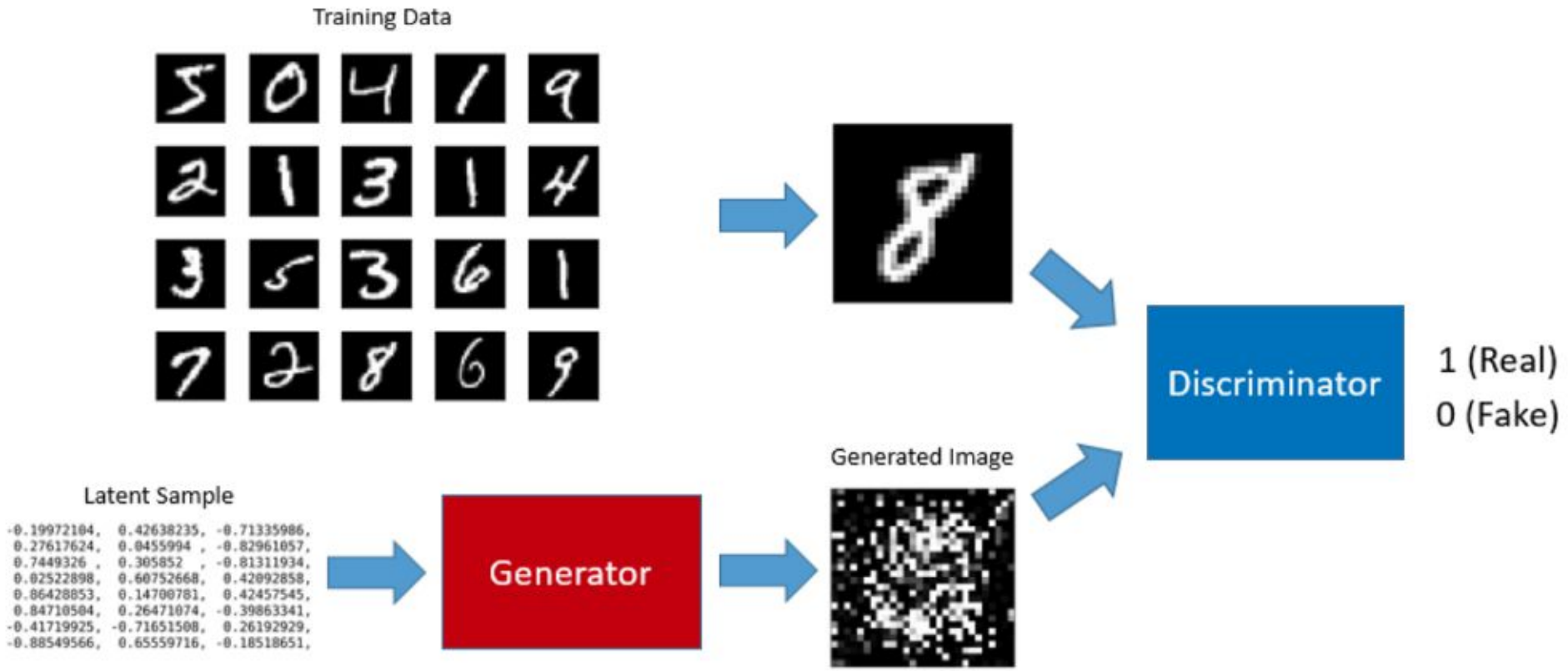
Generativni suparnički modeli



[http://guimperarnau.com/blog/2017/03/Fantastic-GANs-and-where-to-find-them]

Generativni suparnički modeli

- Generiranje slika



[https://towardsdatascience.com/understanding-generative-adversarial-networks-4dafc963f2ef]

Generativni suparnički modeli

– Formalno, generator i diskriminator igraju min-max igru za dva igrača sa sljedećom funkcijom vrijednosti V(G,D):

Objective

$$\min(g)\ \max(d)\ \mathbf{E}_{x\sim P_{data}}[\log(D(x;\theta_d))] + \mathbf{E}_{z\sim P_z}[\log(1 - D(G(z;\theta_g);\theta_d))]$$

P_z -> The data distribution of the noise

P_{data} -> The original data distribution as in the dataset

$x \sim P_{data}$ -> Data sampled from P_{data}

$z \sim P_{data}$ -> Data sampled from P_z

θ_g -> The parameters of the generator network

θ_d -> The parameters of the discriminator network

Generativni suparnički modeli

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

- Prvi element jednadžbe: $\mathbf{E}_{\mathtt{x}\sim \mathtt{P}_{\mathtt{data}}}[\log(D(x;\theta_{\mathtt{d}}))]$

  $D(x;\theta_{\mathtt{d}})$ -> The likelihood of the image 'x' being real

  $\log(D(x;\theta_{\mathtt{d}}))$ -> The log likelihood of the image 'x' being real

  $\mathbf{E}_{\mathtt{x}\sim \mathtt{P}_{\mathtt{data}}}[\log(D(x;\theta_{\mathtt{d}}))]$ -> The expected log likelihood with input samples from real data

- Iz perspektive diskriminatora: maksimizirati vjerojatnost (logaritam vjerojatnosti) ispravne klasifikacije slike kao stvarne ili lažne [D(x) je vjerojatnost da je ulazna slika prava; dakle treba maksimizirati D(x), odnosno log D(x)].

Generativni suparnički modeli

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

- Drugi element jednadžbe: $\mathbf{E}_{\mathrm{z}\sim \mathrm{P}_z}[\log(1 - \mathrm{D}(G(\mathrm{z};\theta_g);\theta_d))]$

  $G(z;\theta_g)$ -> The generated image from noise sample 'z'

  $\mathrm{D}(G(z;\theta_g);\theta_d)$-> The likelihood of the image from Generator being real

  $\log(\mathrm{D}(G(z;\theta_g);\theta_d))$ -> The log likelihood of the image from Generator being real

  $\log(1 - \mathrm{D}(G(z;\theta_g);\theta_d))$ -> The log likelihood of the image from Generator being fake

- Iz perspektive generatora: maksimizirati vjerojatnost (-> logaritam vjerojatnosti) da će diskriminator biti zavaran – maksimizirati D(G(z)), odnosno minimizirati 1-D(G(z)), odnosno minimizirati log(1-D(G(z))).

Generativni suparnički modeli

- Funkcije gubitka
  - Osnovna verzija: igra nulte sume (minimax):

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$

$$J^{(G)} = -J^{(D)}$$

    - Omogućuje jednostavnu teoretsku analizu; lošiji rezultati u praksi (kad je discriminator uspješan, nestaju gradijenti generatora)
  - Heuristička funkcija:

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D\left(G(\boldsymbol{z})\right)$$

    - Bolji rezultati u praksi (svaki igrač koji “gubi” dobiva dostatne gradijente)

Generativni suparnički modeli

- Algoritam učenja

for number of training iterations **do**
  for k steps **do**
    - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

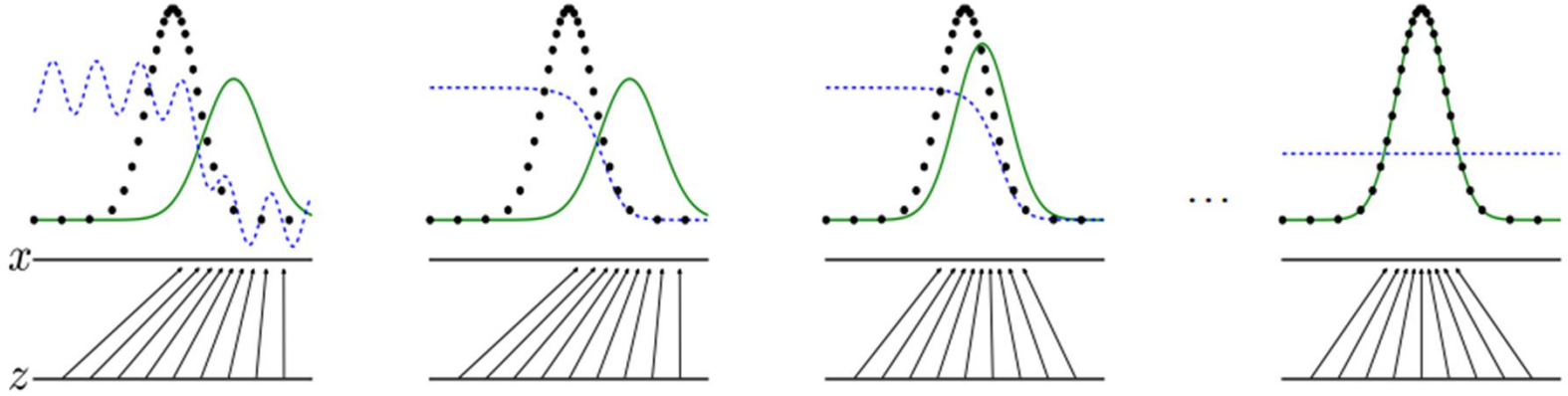
  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

Generativni suparnički modeli

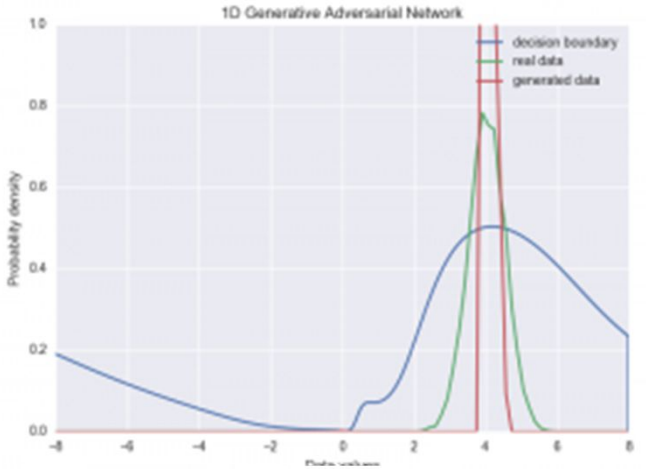
- Ilustracija na 1D primjeru
  - Crna točkana linija: distribucija stvarnih podataka
  - Zelena puna linija: distribucija generiranih podataka
  - Plava crtkana linija: funkcija odluke diskriminatora (procjena vjerojatnosti stvarnog podatka na ulazu)

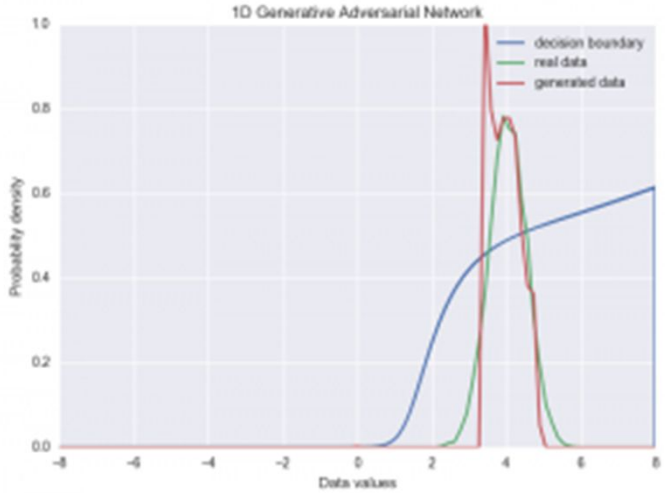


  - (a) suparnički par (G,D) blizu konvergencije – razdiobe stvarnih i generiranih podataka relativno blizu; D relativno točan; (b) ažuriranje D, konvergira prema optimumu; (c) gradijenti D-a vode G prema području koje se vjerojatnije klasificira kao stvarni podaci; (d) nakon niza koraka učenja, ako je kapacitet G I D dovoljan, dosežu točku iz koje ne mogu napredovati jer je distribucija generiranih I stvarnih podataka jednaka; discriminator ih ne može razlikovati te je D(x)=0.5

Generativni suparnički modeli

- Ilustracija na 1D primjeru

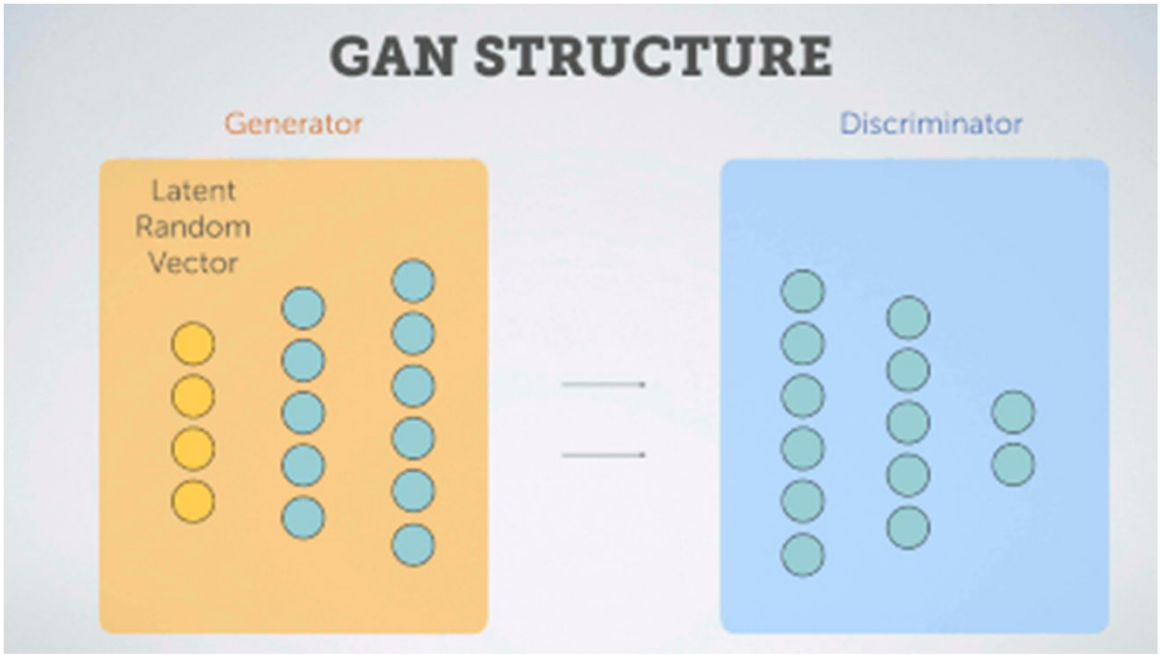




[http://blog.aylien.com/introduction-generative-adversarial-networks-code-tensorflow/]

Osnovni GAN (“Vanilla GAN”)

- Generator i diskriminator – višeslojni perceptroni (potpuno povezani slojevi)



Osnovni GAN (“Vanilla GAN”) – kôd (Python + Keras) [1/3]

```
from keras.datasets import mnist

from keras.layers import Input, Dense, Reshape, Flatten, Dropout
from keras.layers import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU
from keras.models import Sequential, Model
from keras.optimizers import Adam, RMSprop

def generator(self):
    model = Sequential()
    model.add(Dense(256, input_shape=(100,)))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))
   ( ….)
    model.add(Dense(1024))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(self.WIDTH * self.HEIGHT * self.CHANNELS, activation='tanh’))
    model.add(Reshape((self.WIDTH, self.HEIGHT, self.CHANNELS)))
    return model
```

Osnovni GAN (“Vanilla GAN”) – kôd (Python + Keras) [2/3]

```
def discriminator(self):
    model = Sequential()
    model.add(Flatten(input_shape=self.SHAPE))
    model.add(Dense((self.WIDTH * self.HEIGHT * self.CHANNELS), input_shape=self.SHAPE))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense((self.WIDTH * self.HEIGHT * self.CHANNELS)/2))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(1, activation='sigmoid’))
    return model

def __init__(self, width = 28, height= 28, channels = 1):
    (….)
    self.G = self.generator()
    self.G.compile(loss='binary_crossentropy', optimizer=self.OPTIMIZER)
    self.D = self.discriminator()
    self.D.compile(loss='binary_crossentropy', optimizer=self.OPTIMIZER, metrics=['accuracy’]

def stacked_G_D(self):
    self.D.trainable = False ## this freezes the weight, keep reading to understand why this is necessary
    model = Sequential()
    model.add(self.G)
    model.add(self.D)
    return model
```

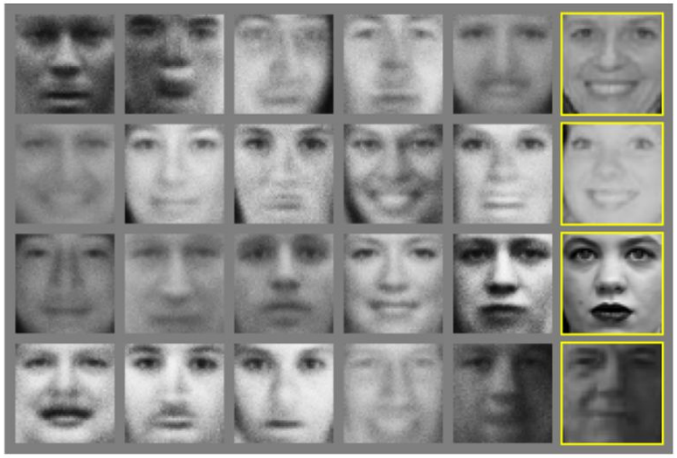
Osnovni GAN (“Vanilla GAN”) – kôd (Python + Keras) [3/3]

```
### Compile the stacked model, so that a new instance
### of the discriminator is created with the frozen parameters.
self.stacked_G_D = self.stacked_G_D()
self.stacked_G_D.compile(loss='binary_crossentropy', optimizer=self.OPTIMIZER)

def train(self, X_train, epochs=20000, batch = 32, save_interval = 200):
    for cnt in range(epochs):
        # train discriminator
        random_index = np.random.randint(0, len(X_train) - batch/2)
        legit_images = X_train[random_index : random_index + batch/2] \
          .reshape(batch/2, self.WIDTH, self.HEIGHT, self.CHANNELS)
        gen_noise = np.random.normal(0, 1, (batch/2,100))
        syntetic_images = self.G.predict(gen_noise)
        x_combined_batch = np.concatenate((legit_images, syntetic_images))
        y_combined_batch = np.concatenate((np.ones((batch/2, 1)), np.zeros((batch/2, 1))))
        d_loss = self.D.train_on_batch(x_combined_batch, y_combined_batch)

        # train generator
        noise = np.random.normal(0, 1, (batch,100))
        y_mislabled = np.ones((batch, 1))
        g_loss = self.stacked_G_D.train_on_batch(noise, y_mislabled)
```

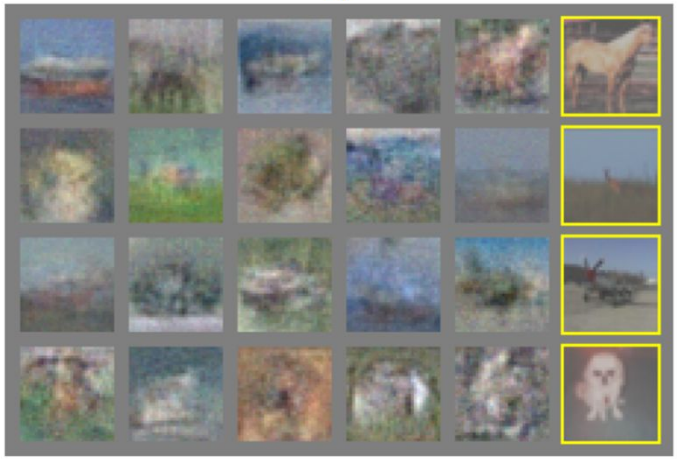
Osnovni GAN

- Osnovni GAN - rezultati

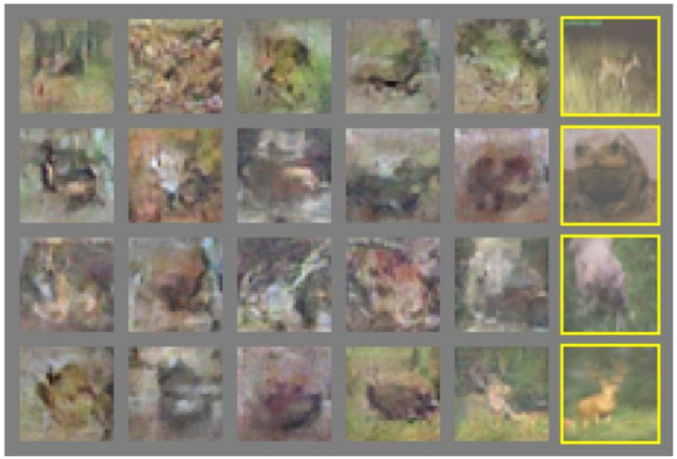
a)



b)



c)



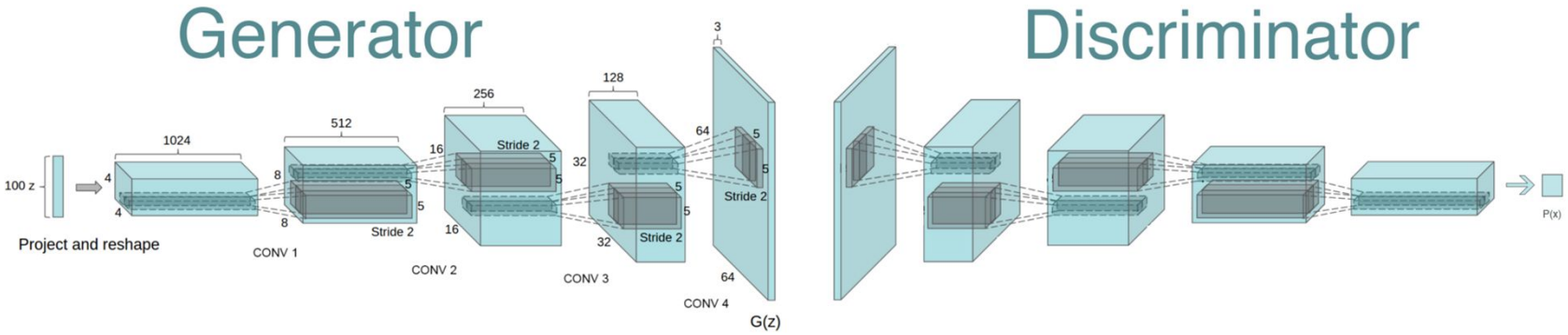
d)

Osnovni GAN

- Osnovni GAN – problem
  - Teško treniranje
    - Rješenje: DCGAN
  - Nedostatak smislene metrike (funkcija gubitka D i G u pravilu nije padajuća tijekom procesa učenja)
    - Rješenje: Wasserstein GAN
  - Kolaps modova (“mode collapse”)
    - Rješenje: Razni trikovi
  - Neki praktični savjeti
    - “How to Train a GAN? Tips and tricks to make GANs work”: https://github.com/soumith/ganhacks

DCGAN (Duboki konvolucijski GAN)

- Radford et al., Unsupervised Representational Learning with Deep Convolutional Generative Adversarial Networks, ICLR 2016

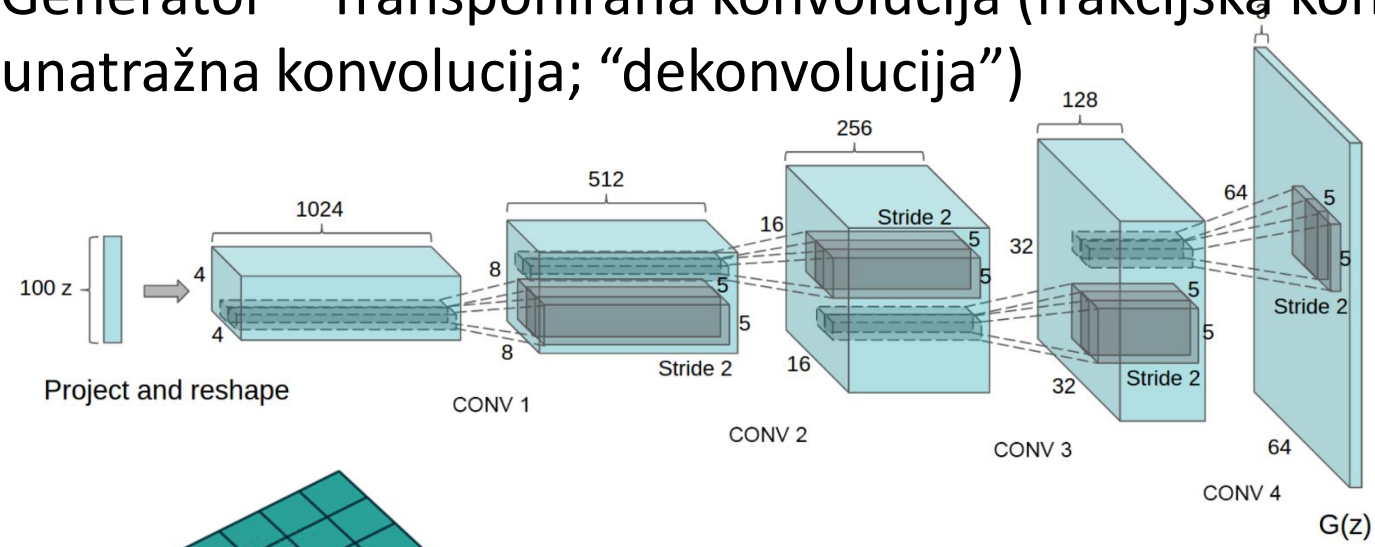


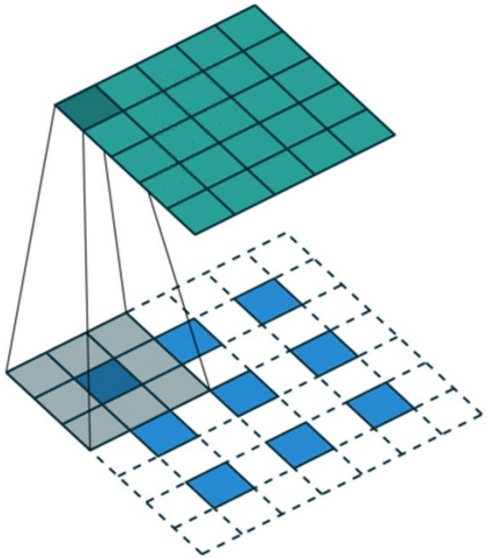
Architecture guidelines for stable Deep Convolutional GANs

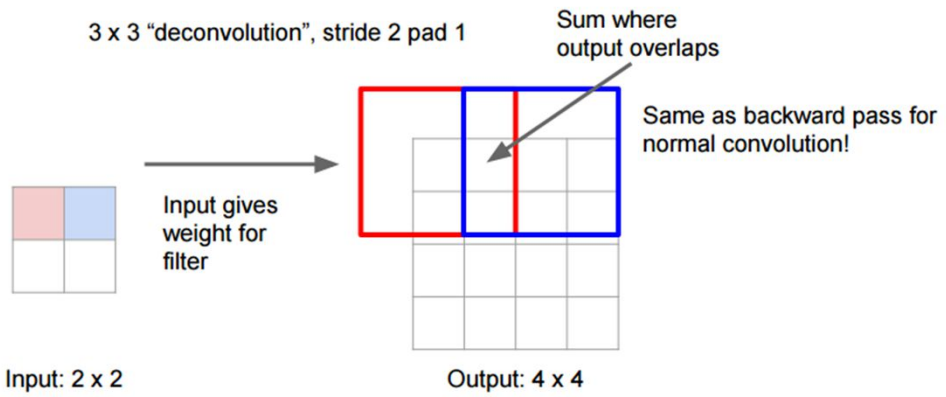
- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

DCGAN

- Generator – Transponirana konvolucija (frakcijska konvolucija, unatražna konvolucija; “dekonvolucija”)





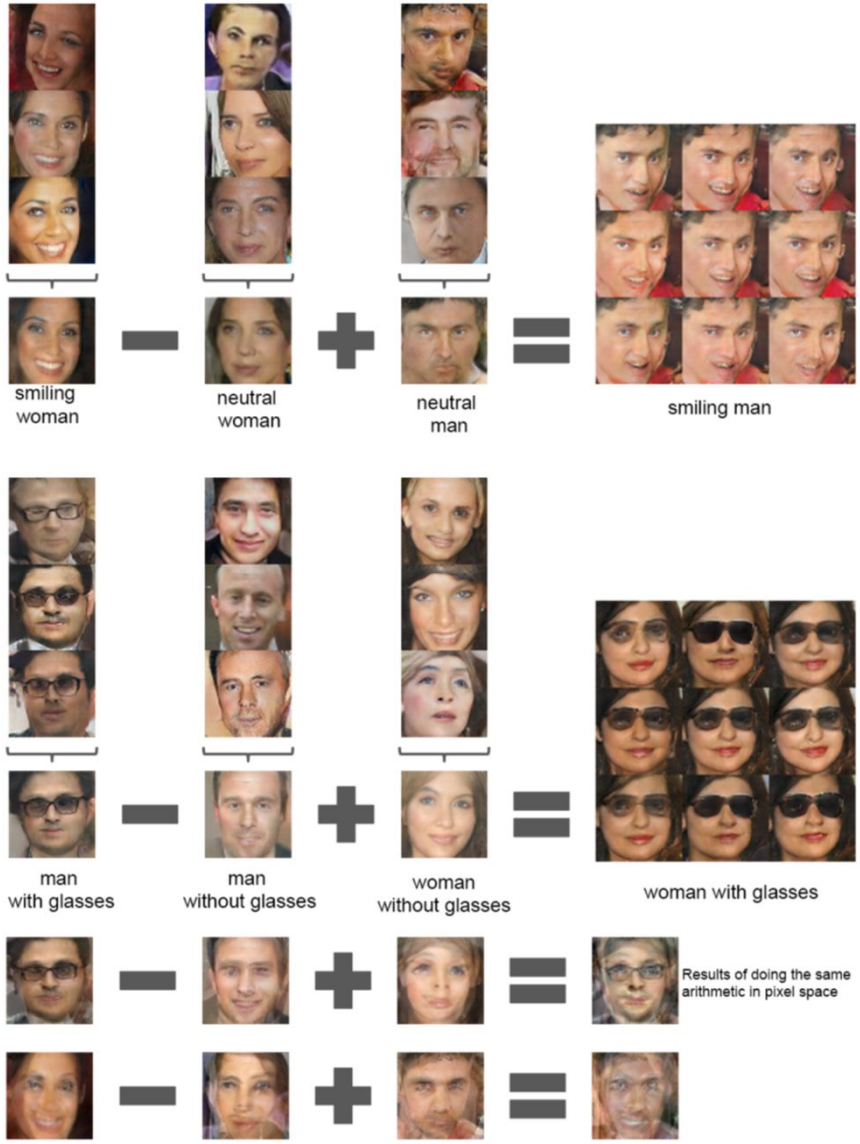


- Transp. Konv. Vs. Upsampling + Konv: https://distill.pub/2016/deconv-checkerboard/

DCGAN

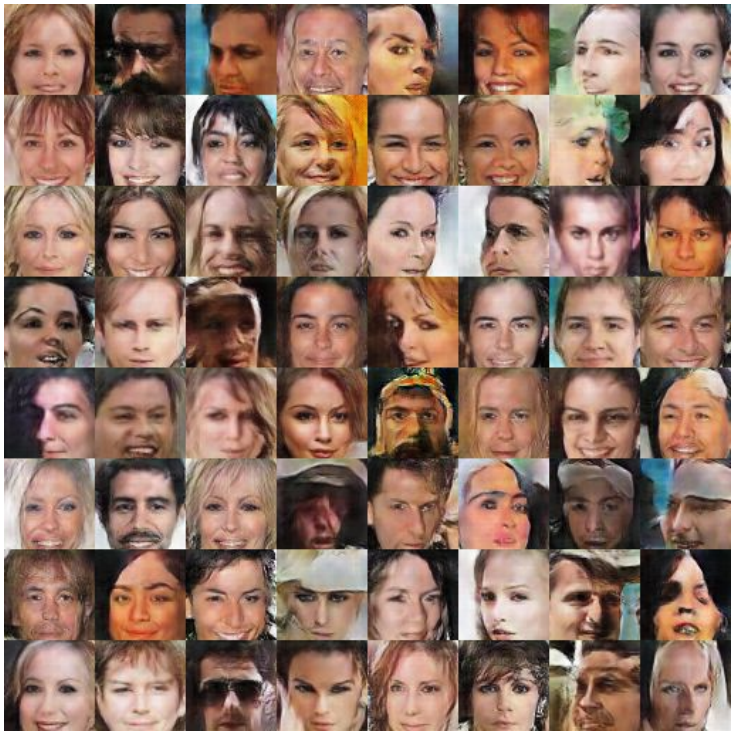
- Rezultati:
  - Primjer generiranja slika spavaćih soba;
    aritmetika u latentnom prostoru

(Radford et al 2015)



DCGAN

- Primjeri:
  - Generiranje MNIST znamenaka
  - Generiranje lica



  - Online demo: https://carpedm20.github.io/faces/

DCGAN – programski kod (Python + Tensorflow + Keras)

```
def discriminator():

    net = Sequential()
    input_shape = (28, 28, 1)
    dropout_prob = 0.4

    net.add(Conv2D(64, 5, strides=2, input_shape=input_shape, padding='same'))
    net.add(LeakyReLU())

    net.add(Conv2D(128, 5, strides=2, padding='same'))
    net.add(LeakyReLU())
    net.add(Dropout(dropout_prob))

    net.add(Conv2D(256, 5, strides=2, padding='same'))
    net.add(LeakyReLU())
    net.add(Dropout(dropout_prob))

    net.add(Conv2D(512, 5, strides=1, padding='same'))
    net.add(LeakyReLU())
    net.add(Dropout(dropout_prob))

    net.add(Flatten())
    net.add(Dense(1))
    net.add(Activation('sigmoid'))

    return net
```

DCGAN – programski kod (Python + Tensorflow + Keras)

```
def generator():

    net = Sequential()
    dropout_prob = 0.4

    net.add(Dense(7*7*256, input_dim=100))
    net.add(BatchNormalization(momentum=0.9))
    net.add(LeakyReLU())
    net.add(Reshape((7,7,256)))
    net.add(Dropout(dropout_prob))

    net.add(UpSampling2D())
    net.add(Conv2D(128, 5, padding='same'))
    net.add(BatchNormalization(momentum=0.9))
    net.add(LeakyReLU())

    net.add(UpSampling2D())
    net.add(Conv2D(64, 5, padding='same'))
    net.add(BatchNormalization(momentum=0.9))
    net.add(LeakyReLU())

    net.add(Conv2D(32, 5, padding='same'))
    net.add(BatchNormalization(momentum=0.9))
    net.add(LeakyReLU())

    net.add(Conv2D(1, 5, padding='same'))
    net.add(Activation('sigmoid'))

    return net
```

DCGAN – programski kod (Python + Tensorflow + Keras)

```
optim_discriminator = RMSprop(lr=0.0008, clipvalue=1.0, decay=1e-10)
model_discriminator = Sequential()
model_discriminator.add(net_discriminator)
model_discriminator.compile(loss='binary_crossentropy', optimizer=optim_discriminator, metrics=['accuracy'])
```

```
optim_adversarial = Adam(lr=0.0004, clipvalue=1.0, decay=1e-10)
model_adversarial = Sequential()
model_adversarial.add(net_generator)

# Disable layers in discriminator
for layer in net_discriminator.layers:
    layer.trainable = False

model_adversarial.add(net_discriminator)
model_adversarial.compile(loss='binary_crossentropy', optimizer=optim_adversarial, metrics=['accuracy'])
```

DCGAN – programski kod (Python + Tensorflow + Keras)

```
# Read MNIST data
x_train = input_data.read_data_sets("mnist", one_hot=True).train.images
x_train = x_train.reshape(-1, 28, 28, 1).astype(np.float32)
```

```
batch_size = 256

for i in range(3001):

    # Select a random set of training images from the mnist dataset
    images_train = x_train[np.random.randint(0, x_train.shape[0], size=batch_size), :, :, :]
    # Generate a random noise vector
    noise = np.random.uniform(-1.0, 1.0, size=[batch_size, 100])
    # Use the generator to create fake images from the noise vector
    images_fake = net_generator.predict(noise)

    # Create a dataset with fake and real images
    x = np.concatenate((images_train, images_fake))
    y = np.ones([2*batch_size, 1])
    y[batch_size:, :] = 0

    # Train discriminator for one batch
    d_stats = model_discriminator.train_on_batch(x, y)

    # Train the generator
    # The input of th adversarial model is a list of noise vectors. The generator is 'good' if the discriminator classifies
    # all the generated images as real. Therefore, the desired output is a list of all ones.
    y = np.ones([batch_size, 1])
    noise = np.random.uniform(-1.0, 1.0, size=[batch_size, 100])
    a_stats = model_adversarial.train_on_batch(noise, y)
```

Stacked GAN, Progressive GAN

- Originalni GAN – niska izlazna rezolucija (do 64x64)
- Progressive growing of GANs (Karras et al, ICLR 2018)

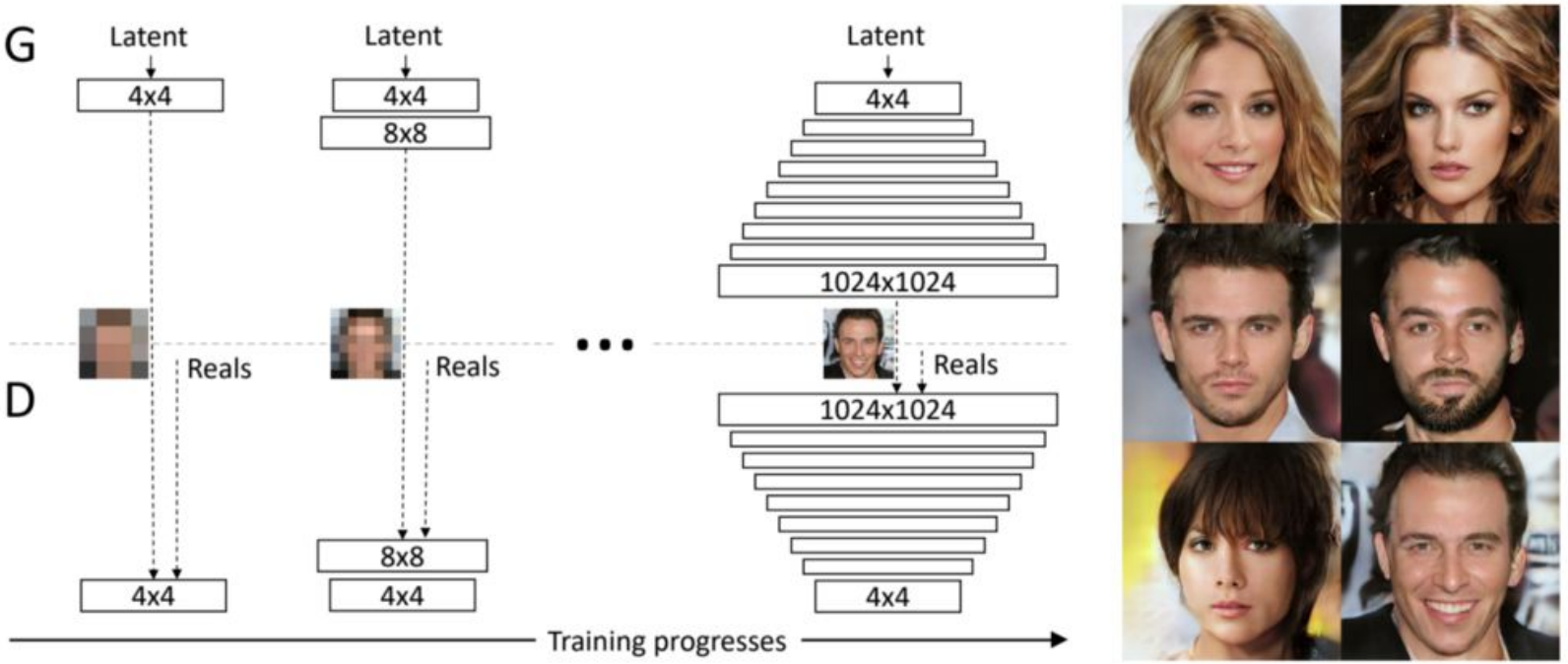
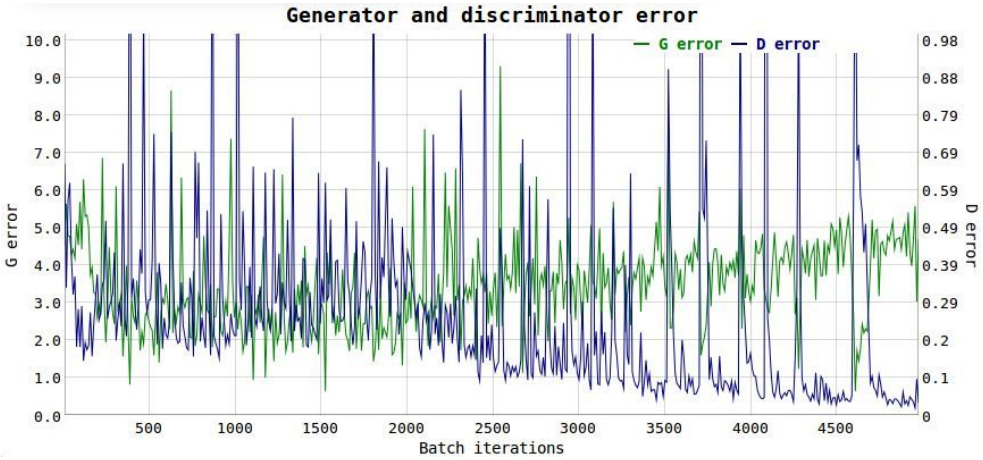
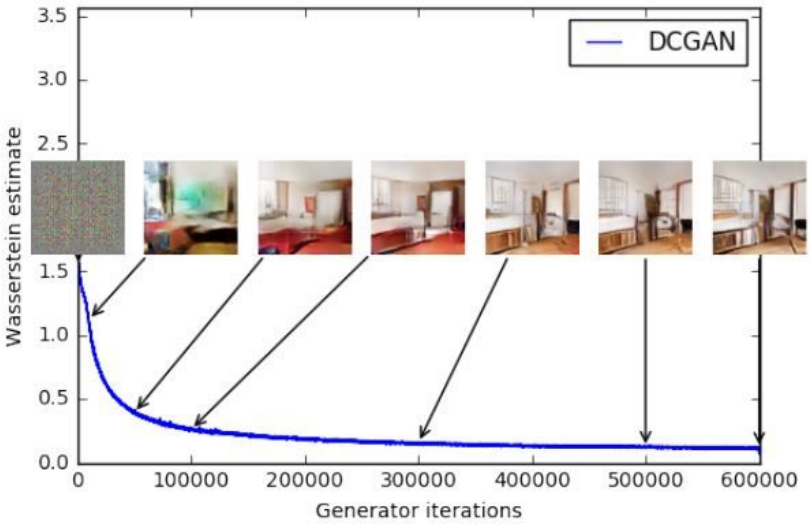


Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $N \times N$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at 1024×1024.

Wasserstein GAN

- Arjovsky et al., Wasserstein GAN, ICML 2017





- Ponašanje gubitka za: Klasični GAN (gore lijevo); Wasserstainov GAN (gore desno)
- Funkcija gubitka postaje korelirana s kvalitetom generiranih slika; stabilnije učenje, manja ovisnost o arhitekturi
- Implementaciia:

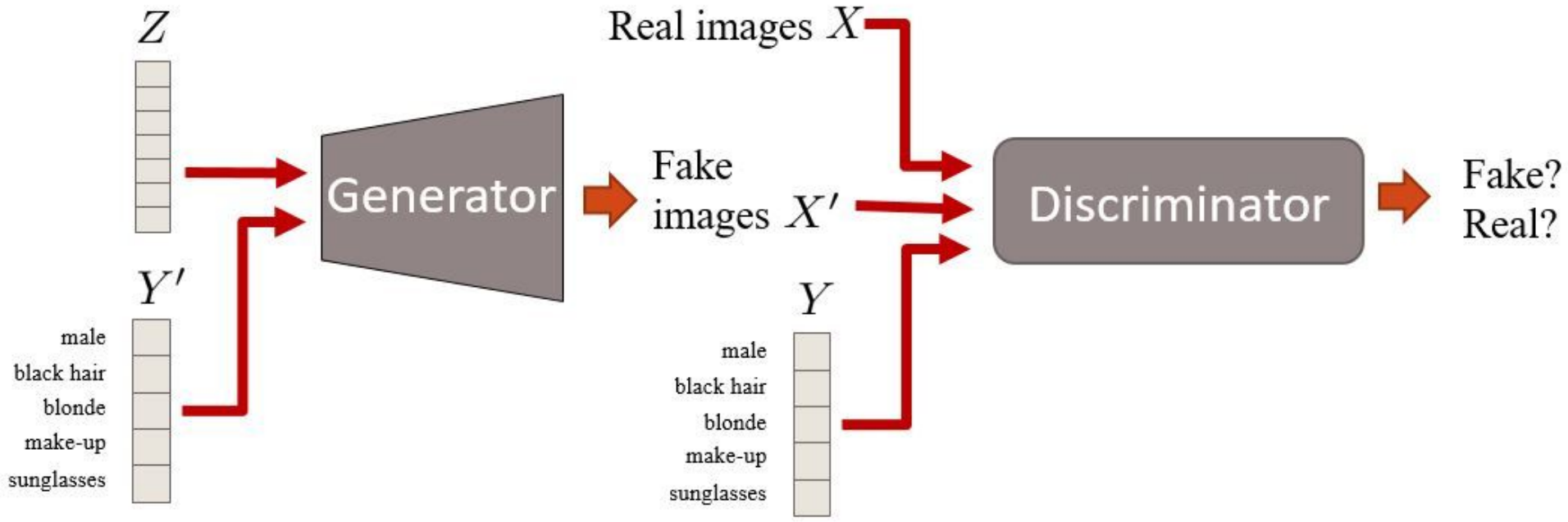
	Discriminator/Critic	Generator
GAN	$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right]$	$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} -\log\left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$
WGAN	$\nabla_w \frac{1}{m} \sum_{i=1}^{m} \left[f(x^{(i)}) - f(G(z^{(i)})) \right]$	$\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -f(G(z^{(i)}))$

$$w \leftarrow w + \alpha \cdot \mathrm{RMSProp}(w, g_w)$$
$$w \leftarrow \mathrm{clip}(w, -c, c)$$

Uvjetni GAN (conditional GAN – cGAN)

- Uvjetni generativni suparnički model
  (Mirza, Conditional Generative adversarial Networks, CoRR Abs/1411.1784, 2014)

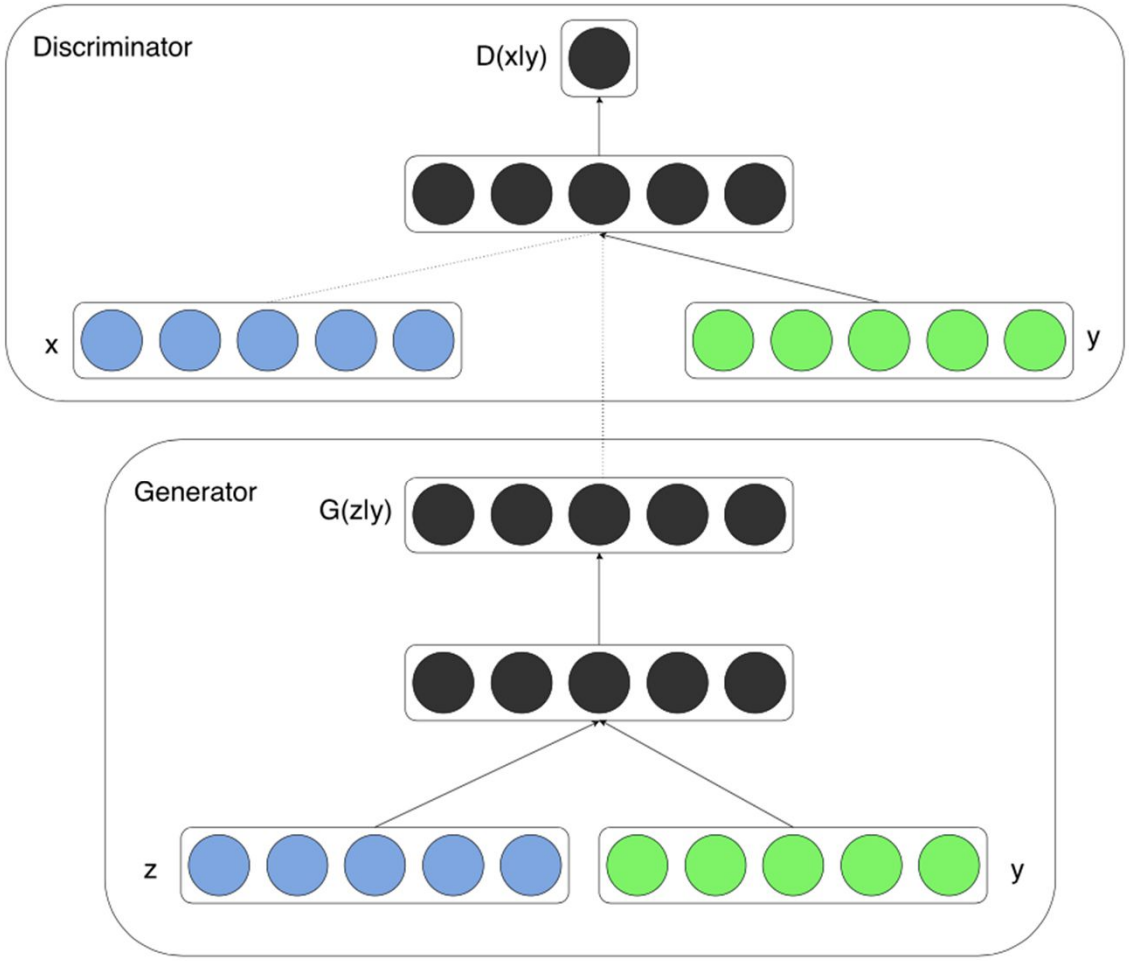
$$\min_G \max_D V(D,G) = \mathop{\mathbb{E}}_{x \sim p_{data}(x)} [\log D(x,y)] + \mathop{\mathbb{E}}_{z \sim p_z(z)} [\log(1 - D(G(z,y),y))]$$



[http://guimperarnau.com/blog/2017/03/Fantastic-GANs-and-where-to-find-them]

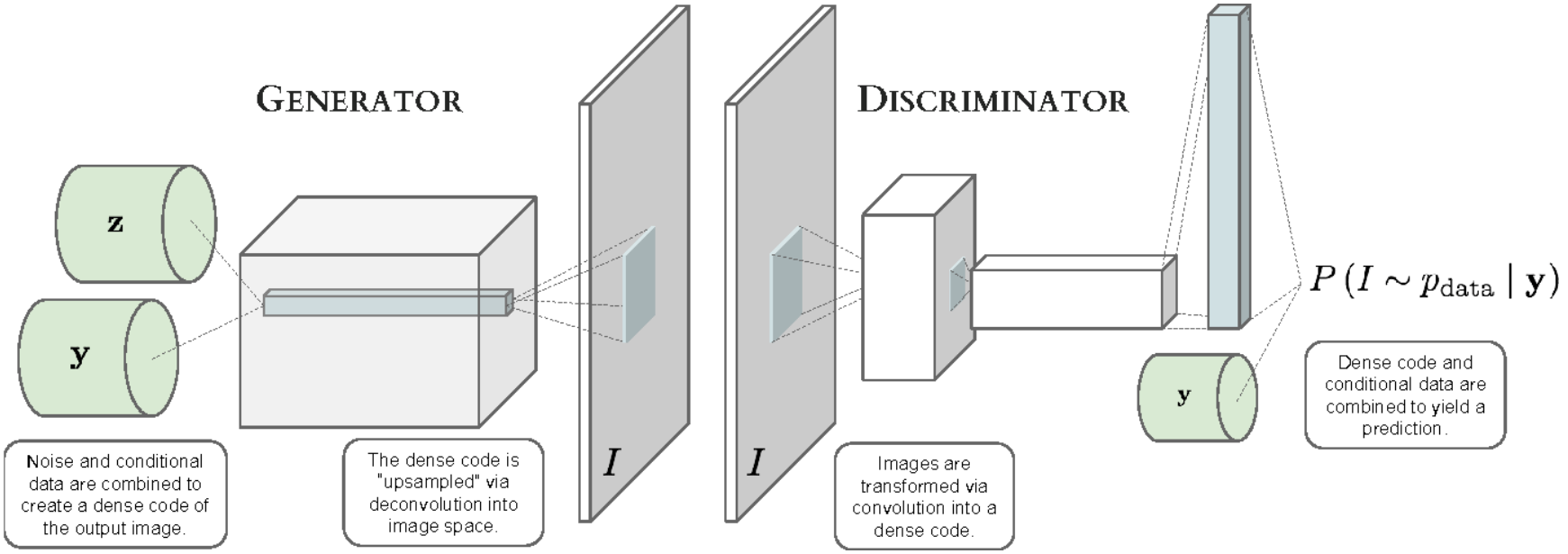
Uvjetni GAN (conditional GAN – cGAN)

- Osnovni (“vanilla”) uvjetni generativni suparnički model



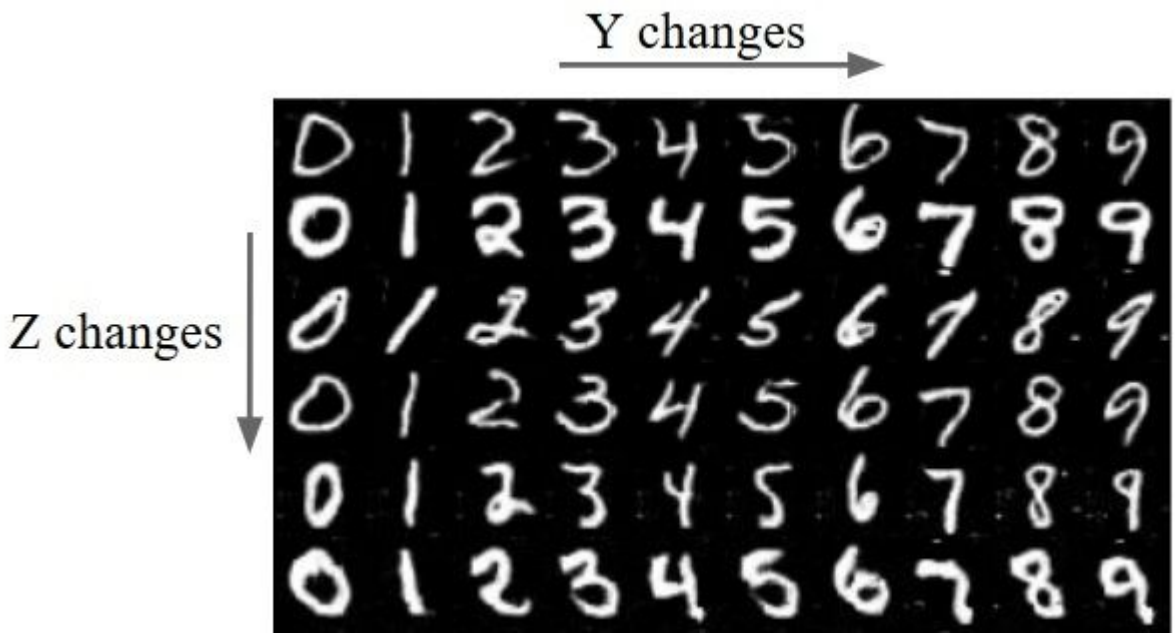
Uvjetni GAN (conditional GAN – cGAN)

- Uvjetni DCGAN (cDCGAN)



Uvjetni GAN

- Primjer: MNIST
  - Dodatno, zadajemo klasu (znamenku koju želimo generirati)



Uvjetni GAN – kôd (Python + Keras) [1/2]

```
#g_inputs is the input for generator
#auxiliary_input is the condition
#d_inputs is the input for discriminator
g_inputs = (Input(shape=(100,), dtype='float32'))
auxiliary_input = (Input(shape=(y_dim,), dtype='float32'))
d_inputs = (Input(shape=(1,28,28), dtype='float32'))

def generator_model():
    T = concatenate([g_inputs,auxiliary_input])
    T = (Dense(1024))(T)
    T = (Dense(128*7*7))(T)
    T = (BatchNormalization())(T)
    T = (Activation('tanh'))(T)
    T = (Reshape((128, 7, 7), input_shape=(128*7*7,)))(T)
    T = (UpSampling2D(size=(2, 2)))(T)
    T = (Convolution2D(64, 5, 5, border_mode='same'))(T)
    T = (BatchNormalization())(T)
    T = (Activation('tanh'))(T)
    T = (UpSampling2D(size=(2, 2)))(T)
    T = (Convolution2D(1, 5, 5, border_mode='same'))(T)
    T = (BatchNormalization())(T)
    T = (Activation('tanh'))(T)
    model = Model(input=[g_inputs,auxiliary_input], output=T)
    return model
```

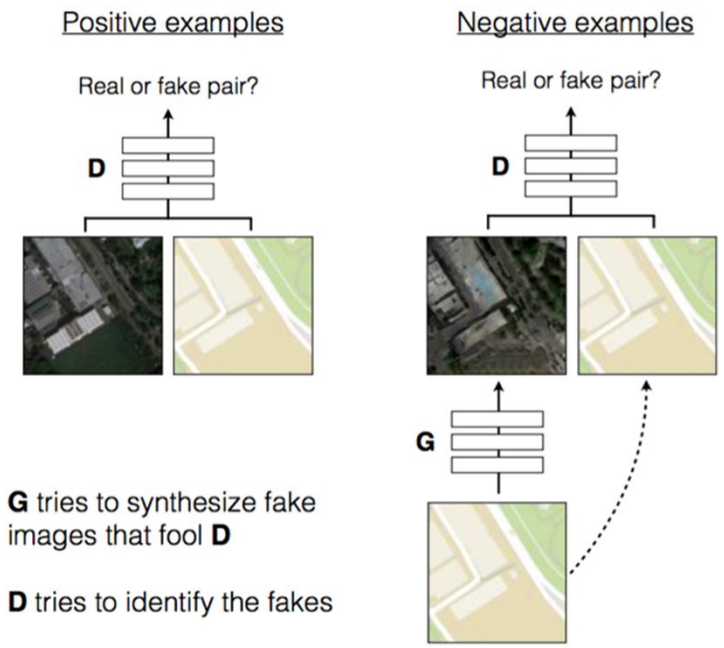
Uvjetni GAN – kôd (Python + Keras) [2/2]

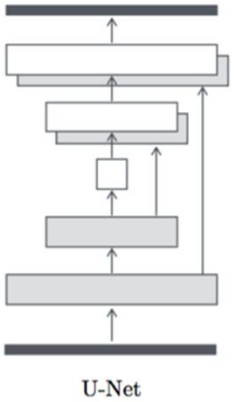
```
def discriminator_model():
    T = (Convolution2D(filters= 64, kernel_size= (5,5), padding='same'))(d_inputs)
    T = (BatchNormalization())(T)
    T = (Activation('tanh'))(T)
    T = (MaxPooling2D(pool_size=(2, 2)))(T)
    T = (Convolution2D(128, 5, 5))(T)
    T = (BatchNormalization())(T)
    T = (Activation('tanh'))(T)
    T = (MaxPooling2D(pool_size=(2, 2)))(T)
    T = (Flatten())(T)
    T = concatenate([T, auxiliary_input])
    T = (Dense(1024))(T)
    T = (Activation('tanh'))(T)
    T = (Dense(1))(T)
    T = (Activation('sigmoid'))(T)
    model = Model(input=[d_inputs,auxiliary_input], output=T)
    return model

def generator_containing_discriminator(generator, discriminator):
    T1 = generator([g_inputs, auxiliary_input])
    discriminator.trainable = False
    T2 = discriminator([T1,auxiliary_input])
    model = Model(input=[g_inputs, auxiliary_input], output=T2)
    return model
```

Uvjetni GAN

- Primjer: Image-to-image translation (“Pix2pix”)
  - Uvjet zadan u obliku slike
  - Diskriminator: `conv64-conv128-conv256-conv512`, sigmoid akt.
  - Generator – “U-Net” arhitektura

Layer index	Encoder	Decoder
1	conv-64	deconv-dropout-512
2	conv-128	deconv-dropout-1024
3	conv-256	deconv-dropout-1024
4	conv-512	deconv-1024
5	conv-512	deconv-1024
6	conv-512	deconv-512
7	conv-512	deconv-256
8	conv-512	deconv-128

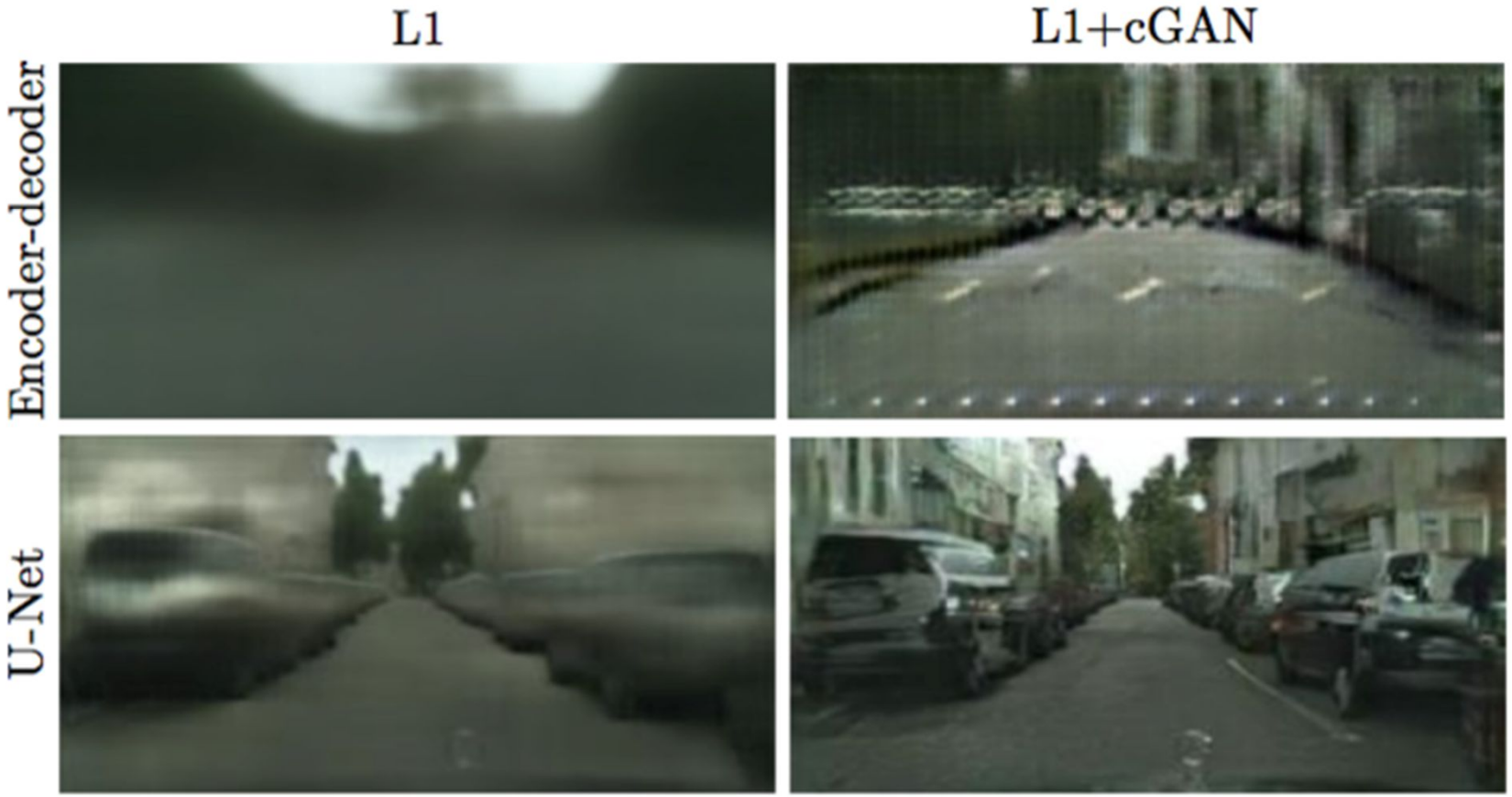




[Isola et al., Image-to-Image translation with Conditional Adversarial Nets, CVPR 2017]

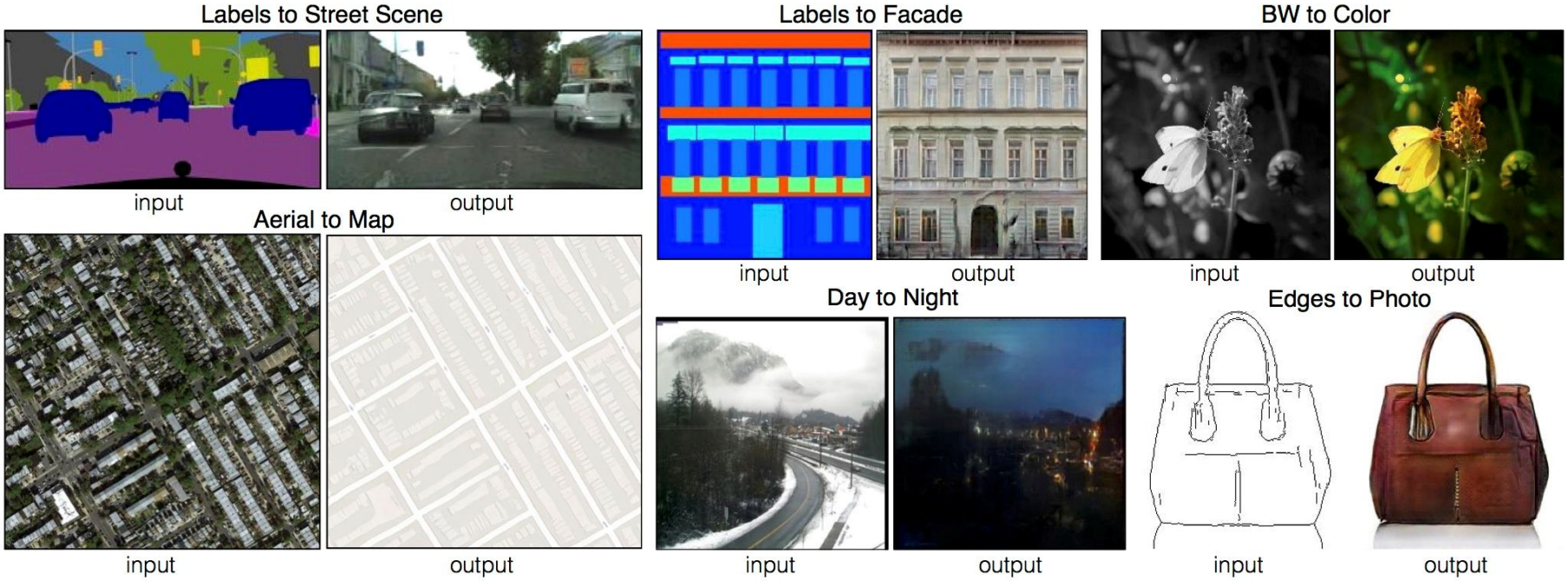
Uvjetni GAN

- Primjer: Image-to-image translation (“Pix2pix”)



Uvjetni GAN

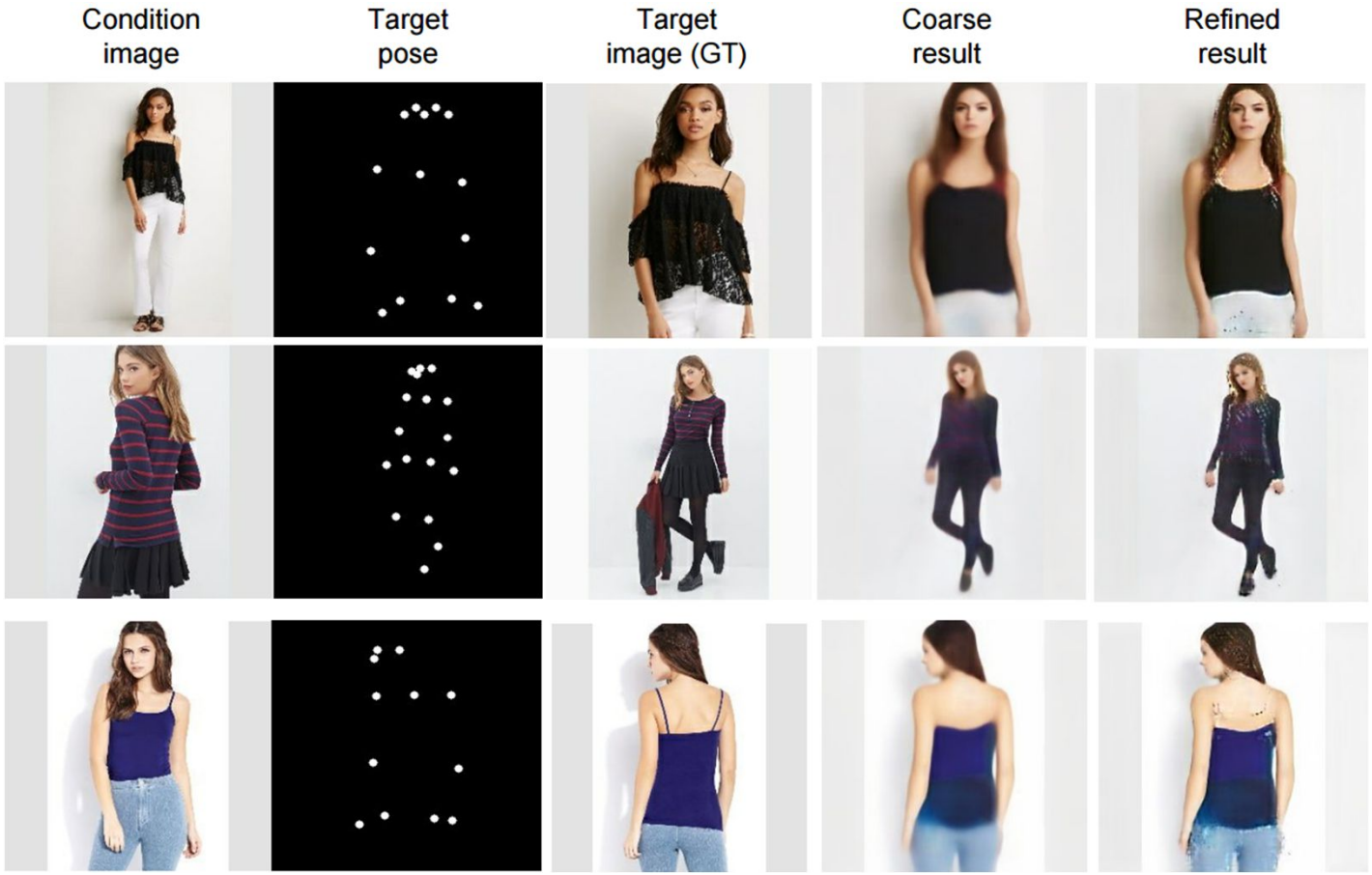
- Primjer: Image-to-image translation (“Pix2pix”)



  - Online demo: https://affinelayer.com/pixsrv/

Uvjetni GAN

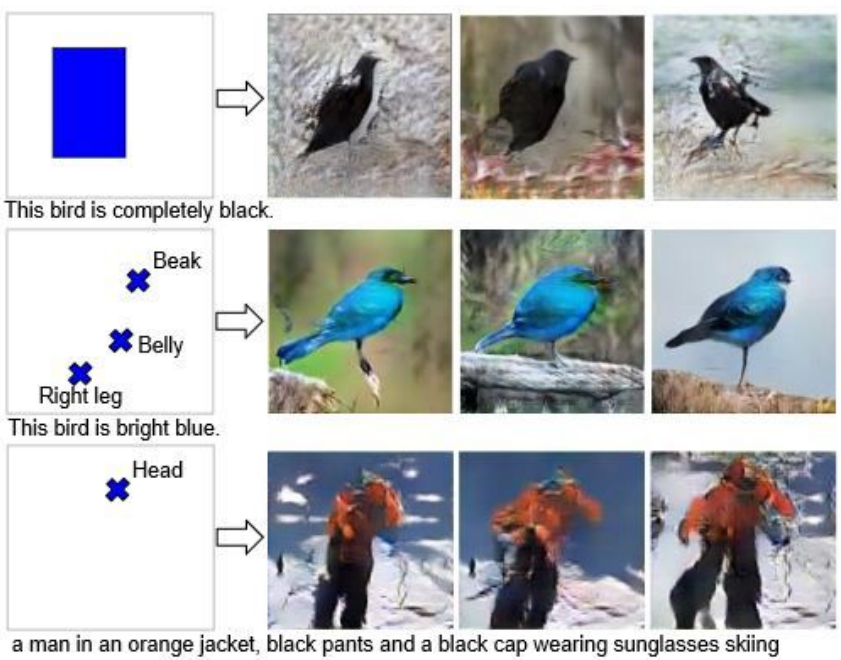
- Primjer: Pose Guided Person Image Generation



[Ma et al., Pose Guided Person Image Generation, NIPS 2017]

Uvjetni GAN

- Primjer: mogućnost zadavanja što i gdje generirati



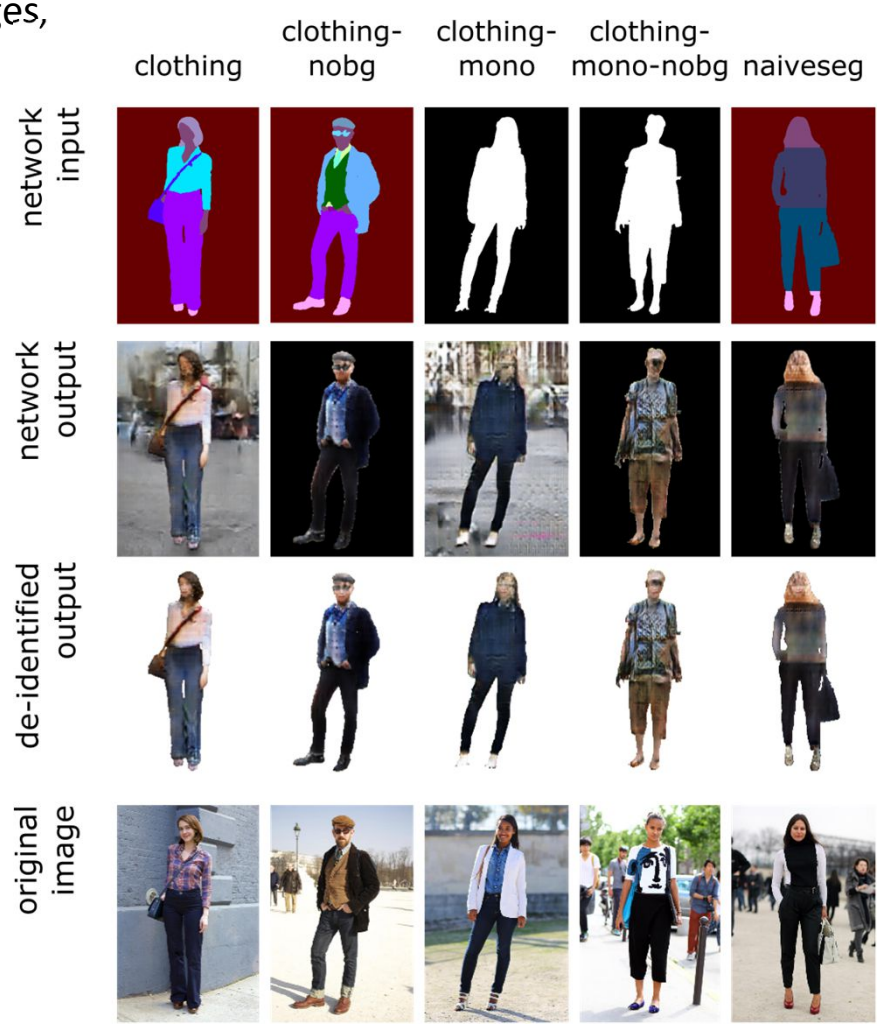
[Reed et al., Learning what and Where to Draw, NIPS 2016]

[Zhang et al., StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, ICCV 2017]

Uvjetni GAN

- Primjena Image-to-image translation – deidentifikacija

  [Brkić et al., I know that person: Generative full body and face de-identification of people in images, CVPR workshop 2017]



GAN – primjeri i primjene

- https://github.com/nashory/gans-awesome-applications
- http://guimperarnau.com/blog/2017/03/Fantastic-GANs-and-where-to-find-them

- https://towardsdatascience.com/do-gans-really-model-the-true-data-distribution-or-are-they-just-cleverly-fooling-us-d08df69f25eb