pawel.kasprowski@polsl.pl

photo: Florida

Artificial Intelligence

Generative Adversarial Networks

Paweł Kasprowski, PhD, DSc.

How the network works

- Take a sample (input)
- Calculate the output using the weights
- Output may be:
  - one value (class)
  - one number (regression)
  - list of numbers (e.g. coordinates)
  - grid of numbers (e.g. saliency map)
- During training we compare the output with ground-truth
  - and recalculate weights

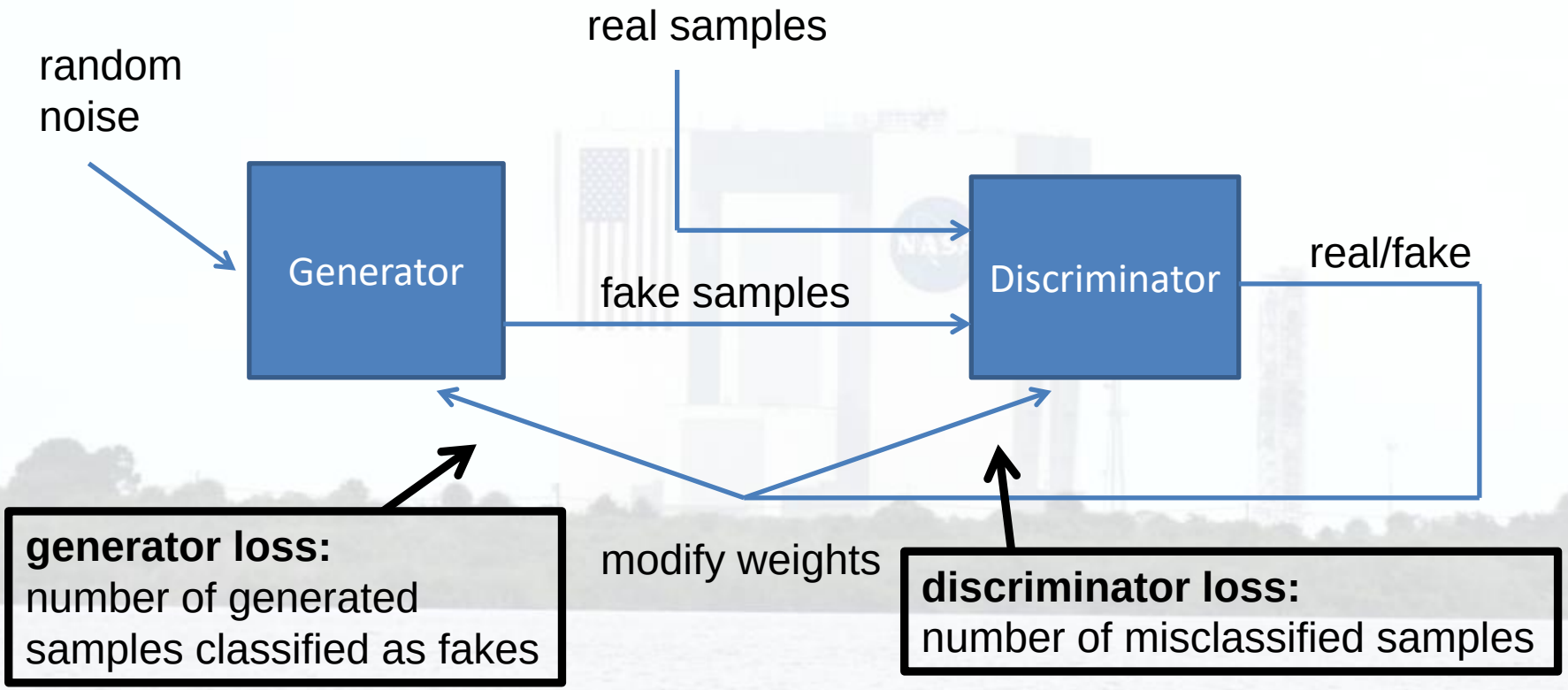
The idea

- We need a network that **generates** samples
  - For example: fake images
- Problem: how to evaluate the fake sample
  - no "hard" conditions
- One idea:
  - hire somebody who evaluates every image
- The better idea:
  - create another network that tries to distinguish between genuine and fake samples!
  - train both networks simultanously

The idea of two networks

- Generator: the network that generates fake samples
  - Input: some random noise
  - Output: sample
- Discriminator: the network that checks if the sample is genuine or fake
  - Input: sample
  - Output: True/False
- Training process:
  - generator tries to "fool" discriminator
    - generate samples that will be classified as genuine

How it works


real samples
random noise
Generator
fake samples
Discriminator
real/fake
modify weights
generator loss:
number of generated samples classified as fakes
discriminator loss:
number of misclassified samples

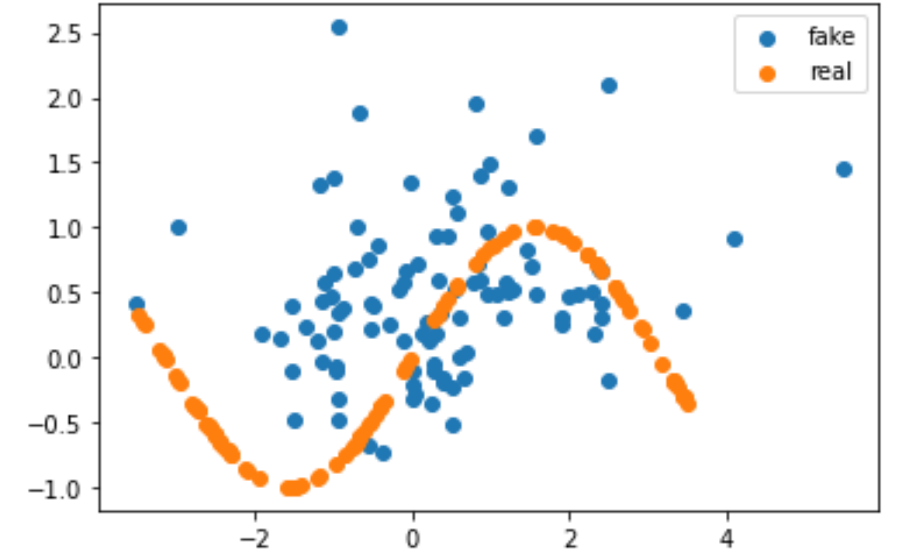
Algorithm

- Produce N random inputs
- Use generator with this N random inputs
- The output from the generator: N fake samples
- Take N real samples and N fake samples
- Use discriminator to classify samples as real or fake
- Apply back propagation to generator and discriminator based on the results
- The next generator will be better – but the next discriminator will be better too!

Simple example

gan0.ipynb

- Sample: 2D point (x,y)
- Real samples: [x,sin(x)]
- Task: train generator to produce correct samples



Generator and discriminator

- Generator
  - gets a random vector (100,) and returns point (x,y)

```
model = Sequential()
model.add(Dense(15, activation='relu', input_dim=100))
model.add(Dense(2, activation='linear'))
```

- Discriminator
  - gets a point (x,y) and returns decision if it is real (0-1)

```
model = Sequential()
model.add(Dense(25, activation='relu',input_dim=2))
model.add(Dense(1, activation='sigmoid'))
```

One training step (1)

- Generate ***batch*** real points

  real_points = generate_real_points(batch)
- Produce a random noise (***batch*** vectors of 100 random values)

  noise = tf.random.normal([batch_size, 100])
- Generate fake points using generator fed with noise

  generated_points = generator(noise, training=True)
- Classify real and fake points using disciminator

  real_output = discriminator(real_points, training=True)

  fake_output = discriminator(generated_points, training=True)

One training step (2)

- Calculate loss for discriminator (based on false classification)

  real_loss = cross_entropy([1...1], real_output)

  fake_loss = cross_entropy([0...0], fake_output)

  disc_loss = (real_loss + fake_loss)/2 # avg of both losses

- Calculate loss for generator (based on recognized fakes)

  gen_loss = cross_entropy([1...1], fake_output)

- Calculate gradients

  gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)

  gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

- Apply gradients (modify weights)

  generator_optimizer.apply_gradients(zip(gradients_of_generator,
  generator.trainable_variables))

  discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
  discriminator.trainable_variables))

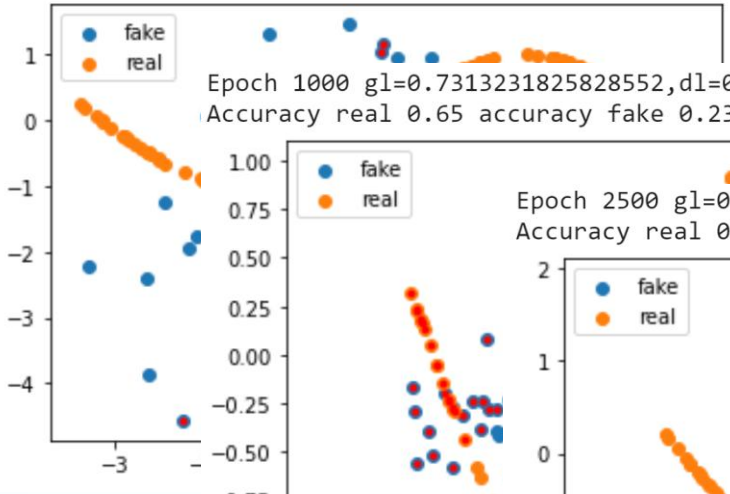
Main loop

- Run 5000 epochs, visualize after each 100 epochs

```
epochs = 5000
for epoch in range(epochs):
    do_step(150)
    if epoch % 100 == 0:
      show_results()
print("Done")
```

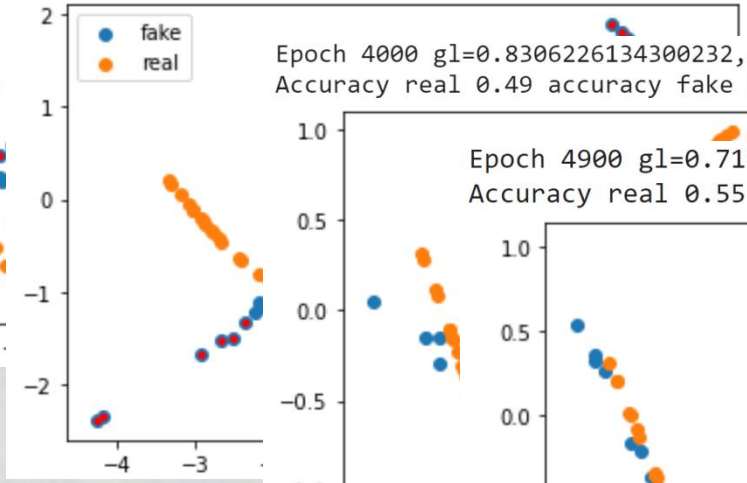
Results

Epoch 100 gl=0.69415682554245,dl=0.6602636575698853
Accuracy real 0.82 accuracy fake 0.4



Epoch 1000 gl=0.7313231825828552,dl=0.6632883548736572
Accuracy real 0.65 accuracy fake 0.23

Epoch 2500 gl=0.7858718633651733,dl=0.6127379536628723
Accuracy real 0.79 accuracy fake 0.31



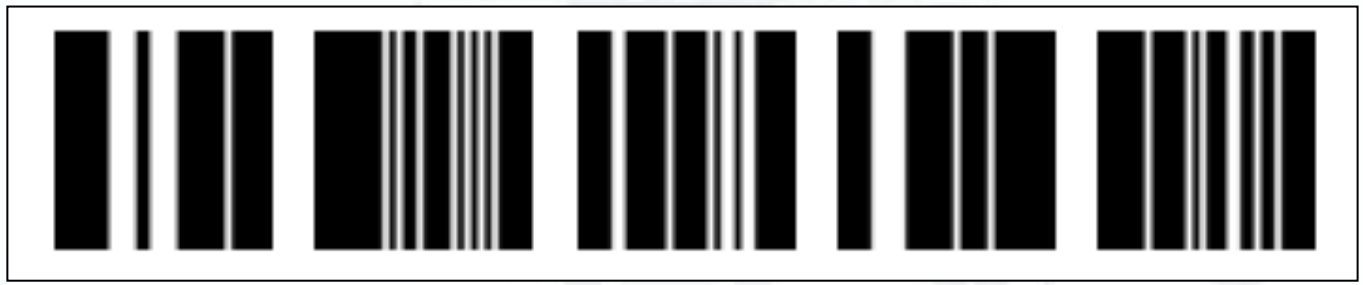
Epoch 4000 gl=0.8306226134300232,dl=0.6306347846984863
Accuracy real 0.49 accuracy fake 0.21

Epoch 4900 gl=0.7193933129310608,dl=0.6836581230163574
Accuracy real 0.55 accuracy fake 0.45

Deep Convolutional GAN (DCGAN)

gan1_vl.ipynb

- Learns how to create images 32x32 with vertical lines



- For every iteration a new set of fake images is created
- It should be better and better!

The code

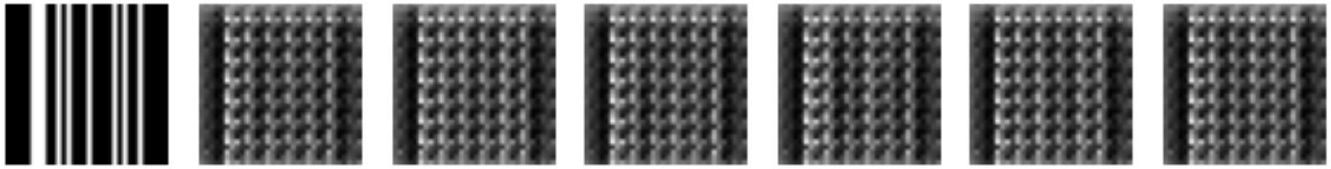
- Generate 1000 images
- Create generator (100,) -> (32,32,1)
- Create discriminator (32,32,1) -> (0/1)
- For 400 epochs:
  - sample 50 real images
  - create 50 fake images
  - execute one learning step
- For every 10 epoch:
  - show one real and 6 generated images

Results

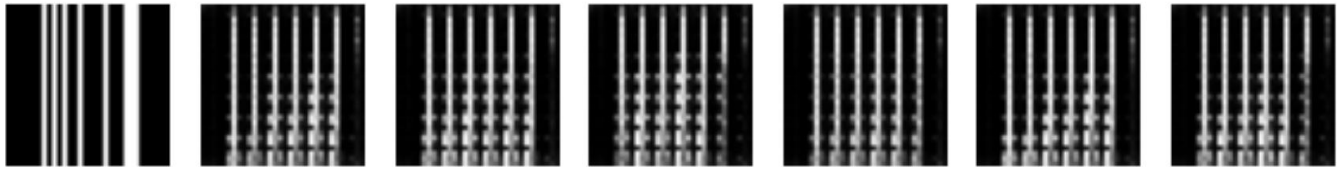
gan1_vl.ipynb

Epoch 0 g_loss= 0.70612067 d_loss= 1.3895196 real_acc= 0.36 fake_acc= 0.64

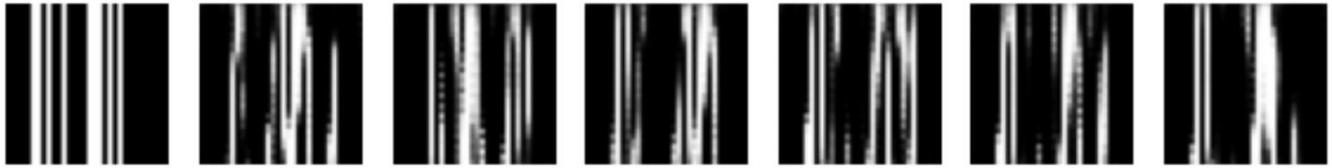
Epoch 100 g_loss= 1.9474657 d_loss= 0.31368467 real_acc= 0.98 fake_acc= 1.0



Epoch 300 g_loss= 2.6416433 d_loss= 0.16897252 real_acc= 0.96 fake_acc= 1.0



Epoch 1000 g_loss= 3.522217 d_loss= 0.26228228 real_acc= 0.96 fake_acc= 0.96



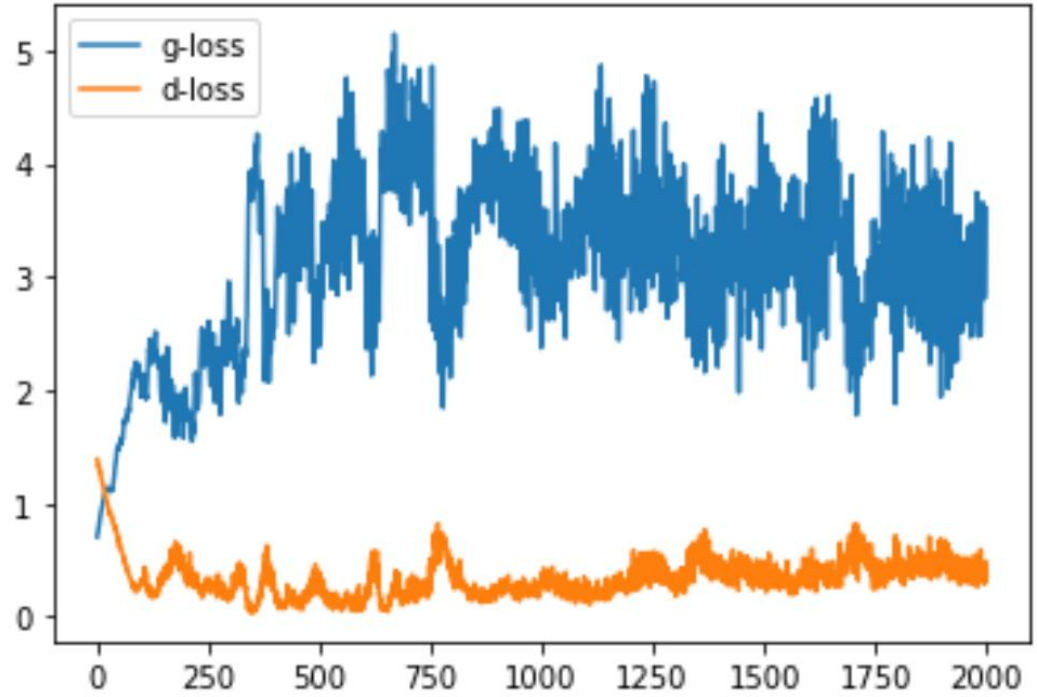
Epoch 1500 g_loss= 3.4661248 d_loss= 0.2947845 real_acc= 0.98 fake_acc= 0.96

How to evaluate GAN?

- Loss of generator and discriminator should not decrease!
- If the discrimintator loss decreases – it means it perfectly distiguishes real and fake images
  - so they are different!
- If the generator loss decreases – it means that it perfectly fools the discrimintator
  - so there is no progress
- Effectively: both losses should be around 0,5-1 and be stable

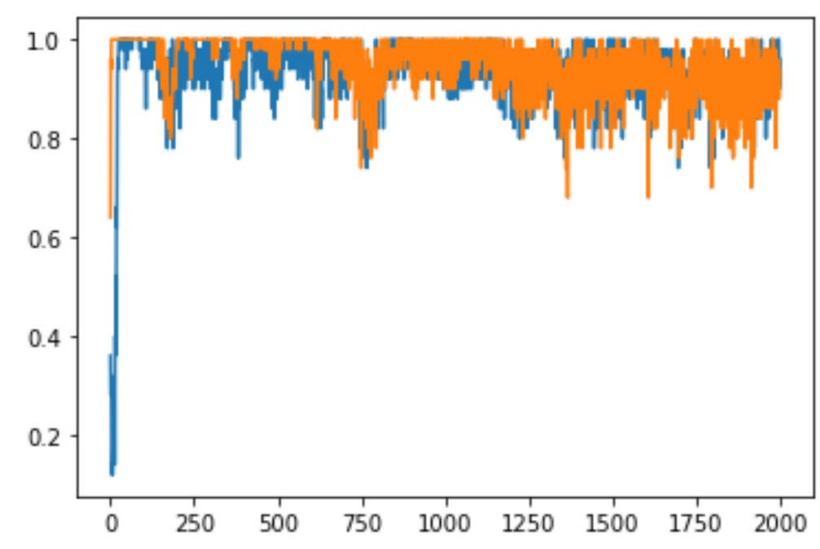
For the last example

- Gen-loss is too high and disc-loss is too low
- There is a big variation in loss values



Accuracy of the discriminator

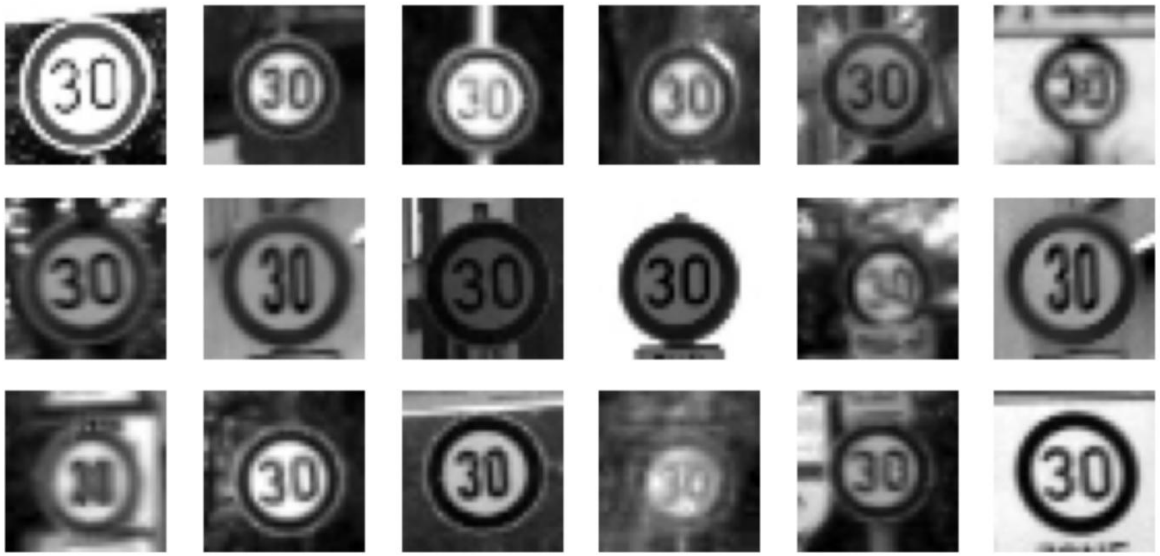
- It should not be 100% and it should not be 0%
- It should be evaluated separately for fake and real images
- The rule of the thumb: it should be about 80% for both
- If accuracy is too high no progress is made!
- In our example:
  - accuracy too high!



Road sign image generation

gan2_signs.ipynb

- Real images: 1500 photos of speed limit 30
- GAN task: generate fake images



Results

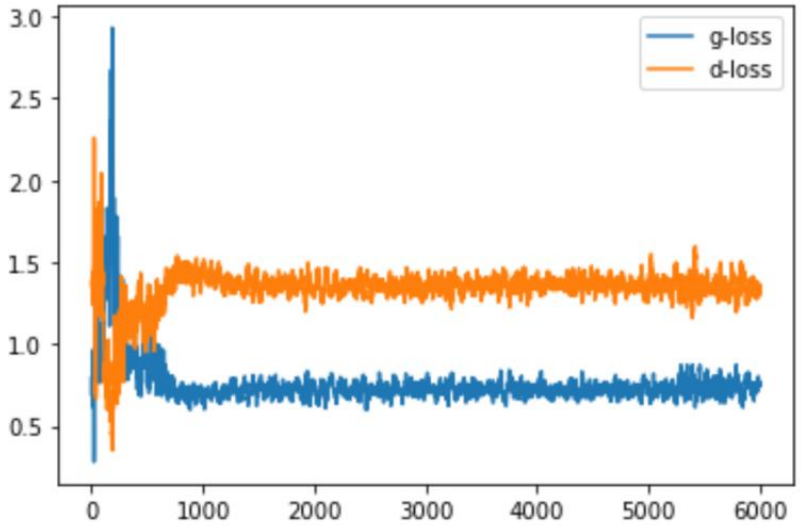
gan2_signs_6000.ipynb

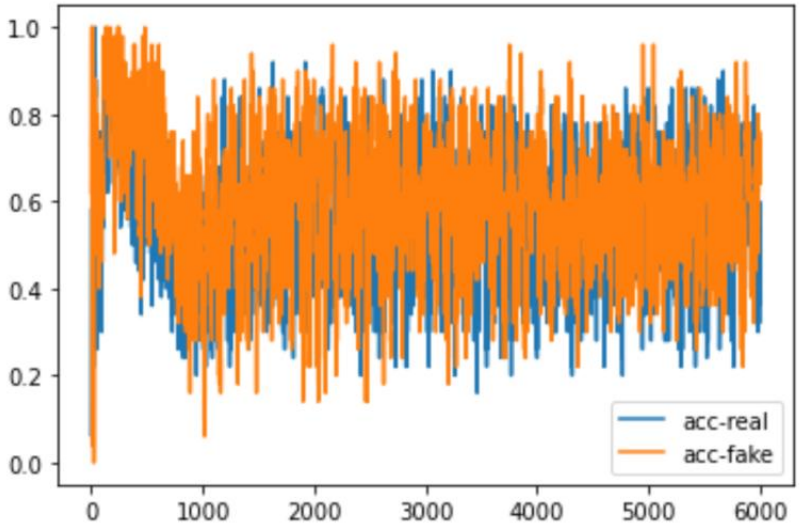
Epoch 10 g_loss= 0.6984158 d_loss= 1.3769169 real_acc= 0.6 fake_acc= 0.62

Epoch 1980 g_loss= 0.759776 d_loss= 1.3104165 real_acc= 0.54 fake_acc= 0.8

Epoch 3960 g_loss= 0.7248465 d_loss= 1.3830683 real_acc= 0.46 fake_acc= 0.7

Loss and accuracy





Very stable!

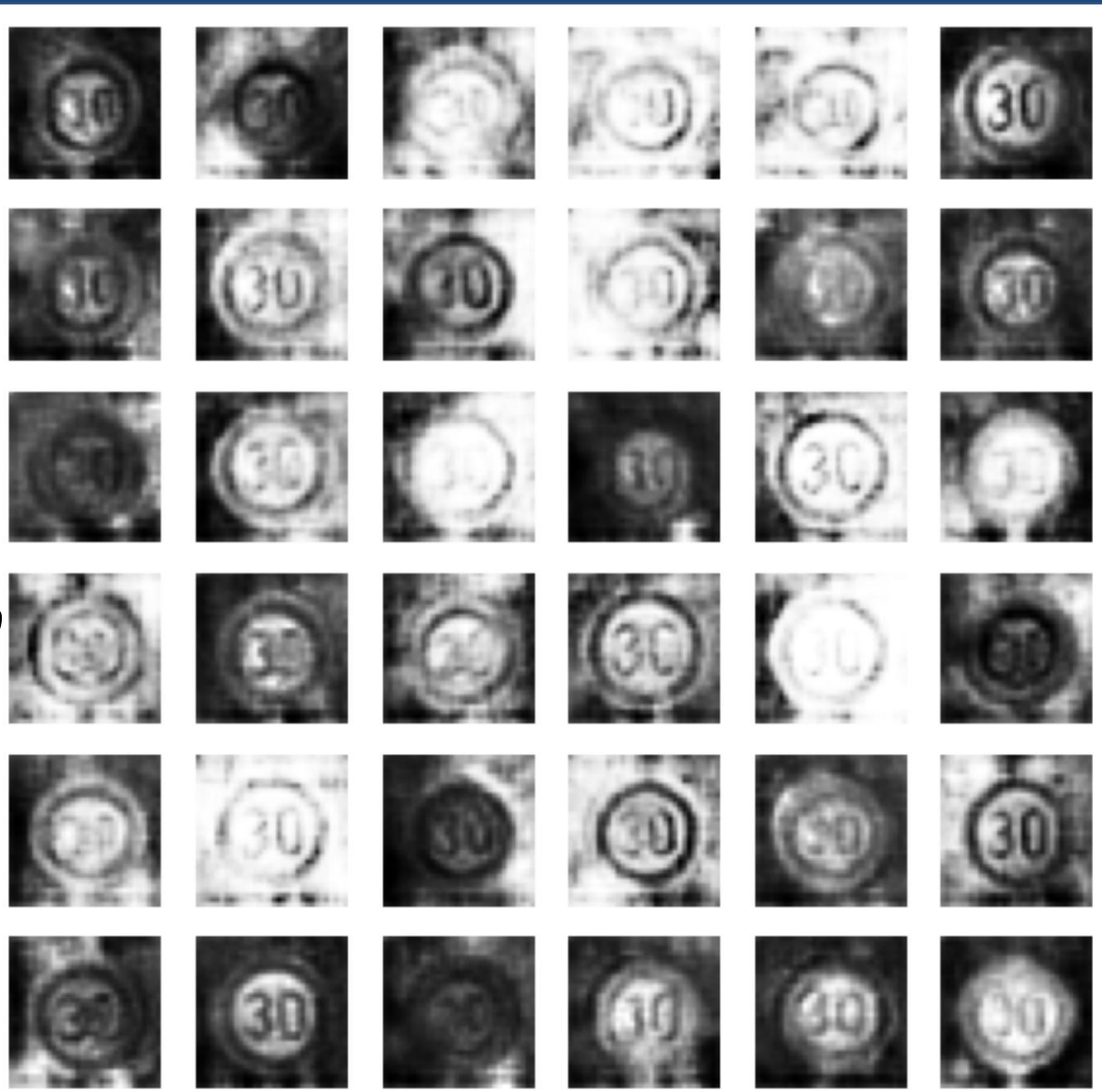
Not too high

```
Epoch 5820 g_loss= 0.72707444 d_loss= 1.3181264 real_acc= 0.62 fake_acc= 0.56
```

pawel.kasprowski@polsl.pl

Fake or real?

check_model.ipynb

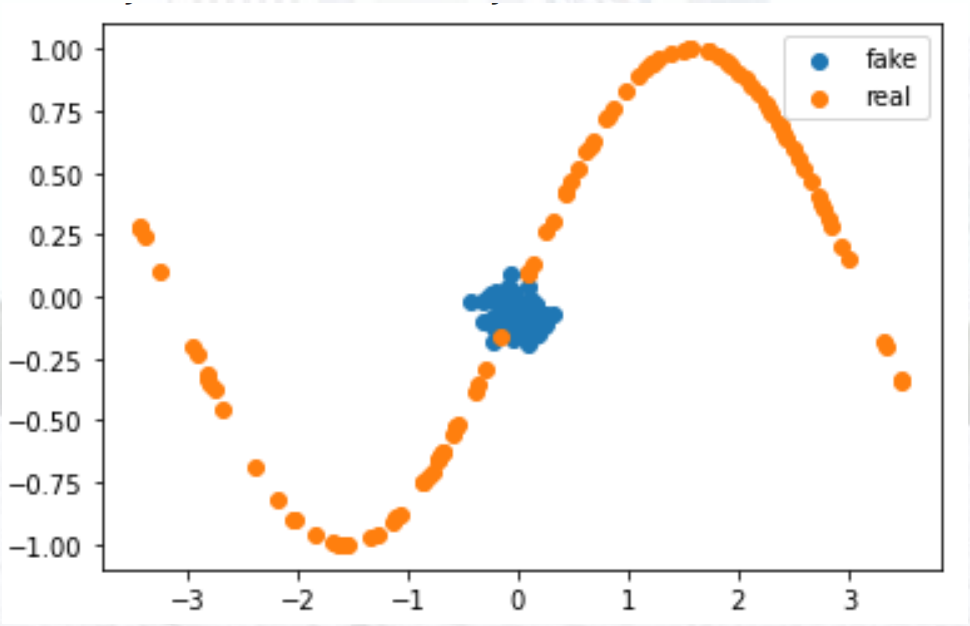


Common problems with GANs

- GANs are dynamic systems!
  - two competing networks evolve simultanously
- There is no "end of training" moment
- Typical problems:
  - mode collapse
  - convergence failure

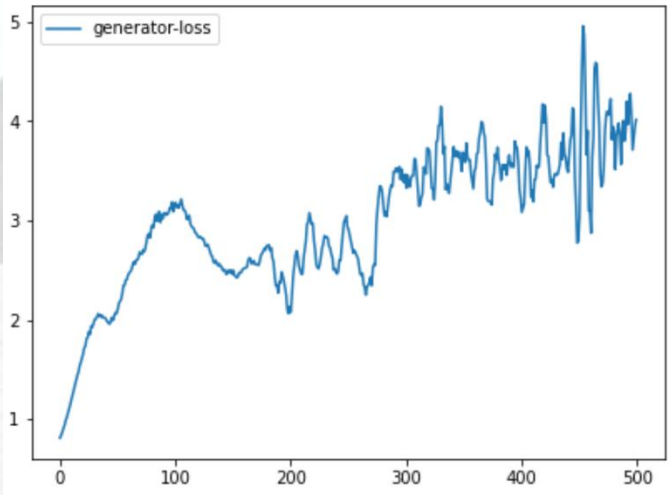
Mode collapse

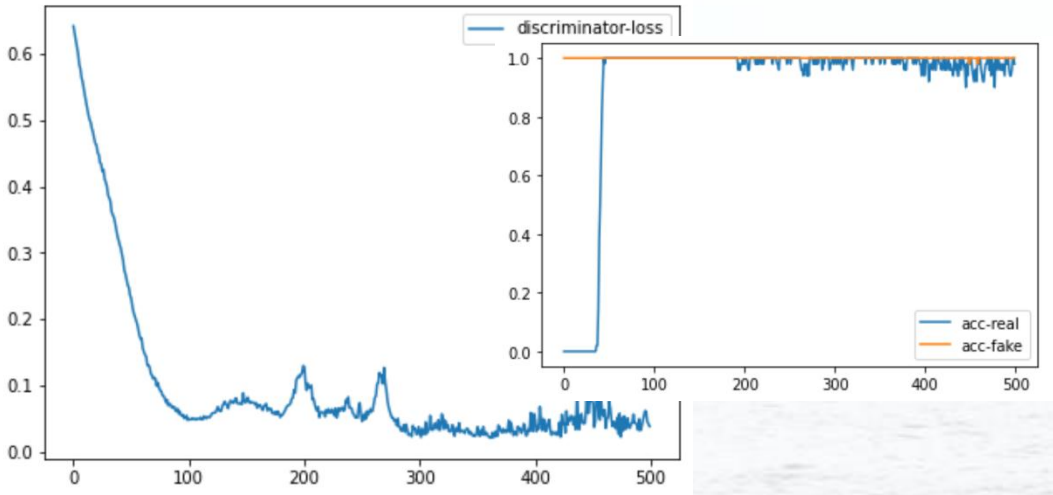
- Generator generates very similar samples
- It does not search for new ones
- The output is very similar regardless of the input



Convergence failure

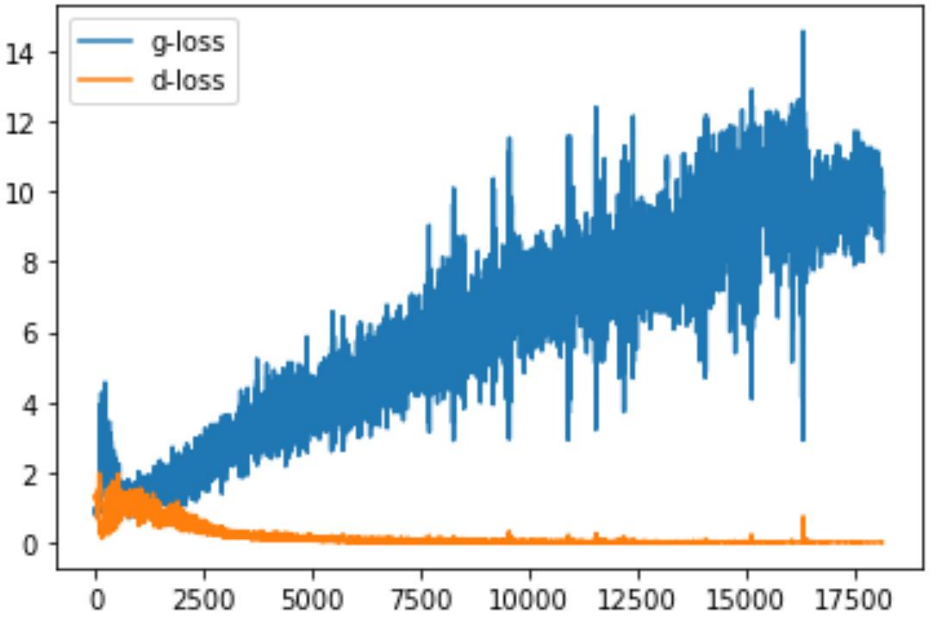
- Generator and discriminator do not converge
- Typically:
  - discriminator loss falls to very low value
  - generator loss grows continously
  - accuracy of the discriminator is 100%





What can we do?

- Change optimization algorithm's parameters (lr?)
- Change size of batches
- Change networks' architecture
  - activations
  - batch normalization
  - dropouts
- Use more data
  - if possible



GAN hacks

- Normalize images to (-1,1)
  - use tanh in generator output
- Use LeakyReLU instead of ReLU
- Use AveragePooling instead of MaxPooling
- Use soft labels
  - e.g. 0.7-1.3 instead of 1 and 0-0.3 instead of 0
- Use Dropouts (even 0.5)
- More hacks:
  - https://github.com/soumith/ganhacks

Conditional GANs

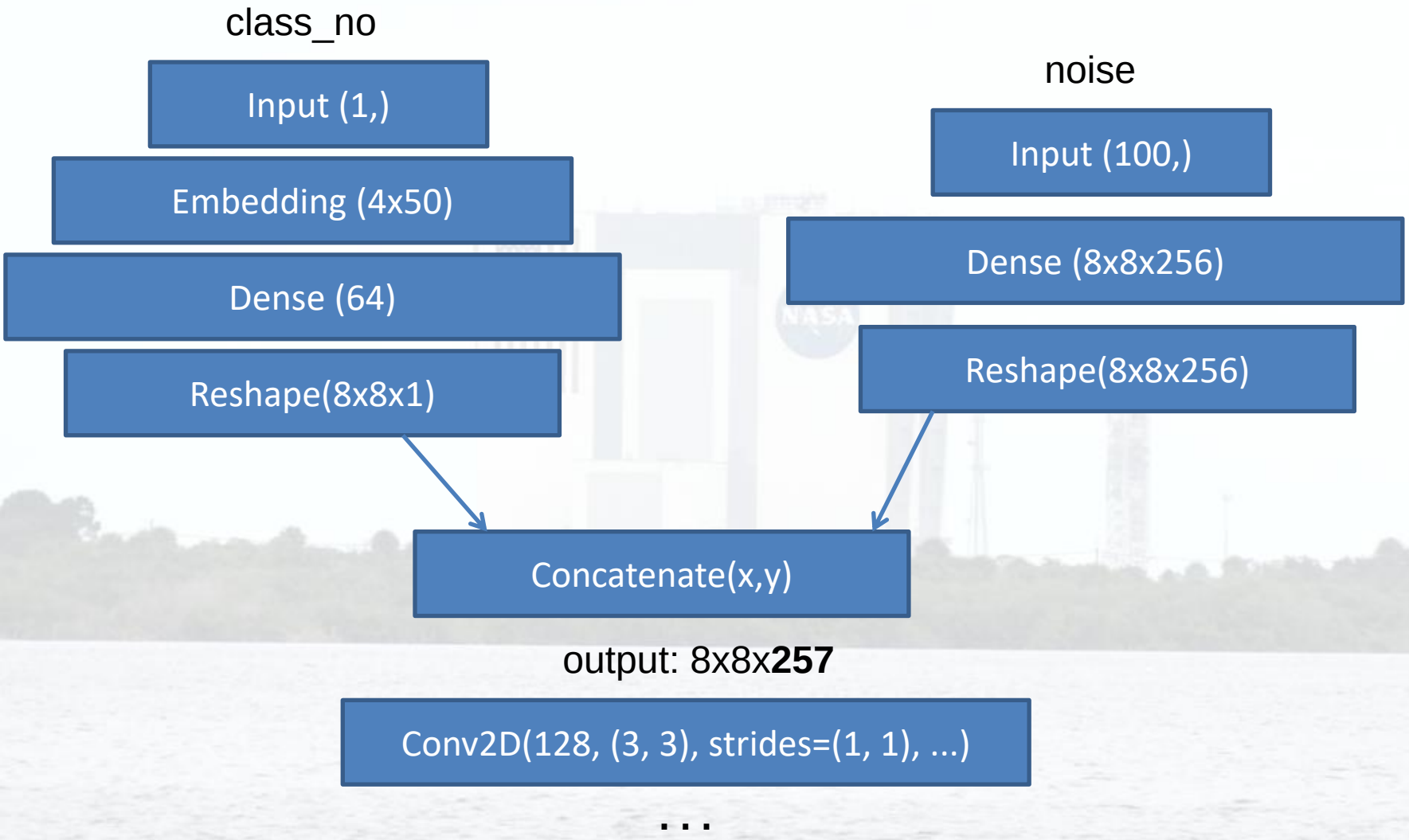
- Generator generates samples of different classes
- Example: generator of different traffic sign images
- Training set four classes:
  - speed limit 30
  - speed limit 100
  - stop
  - no trucks

Examples from the dataset

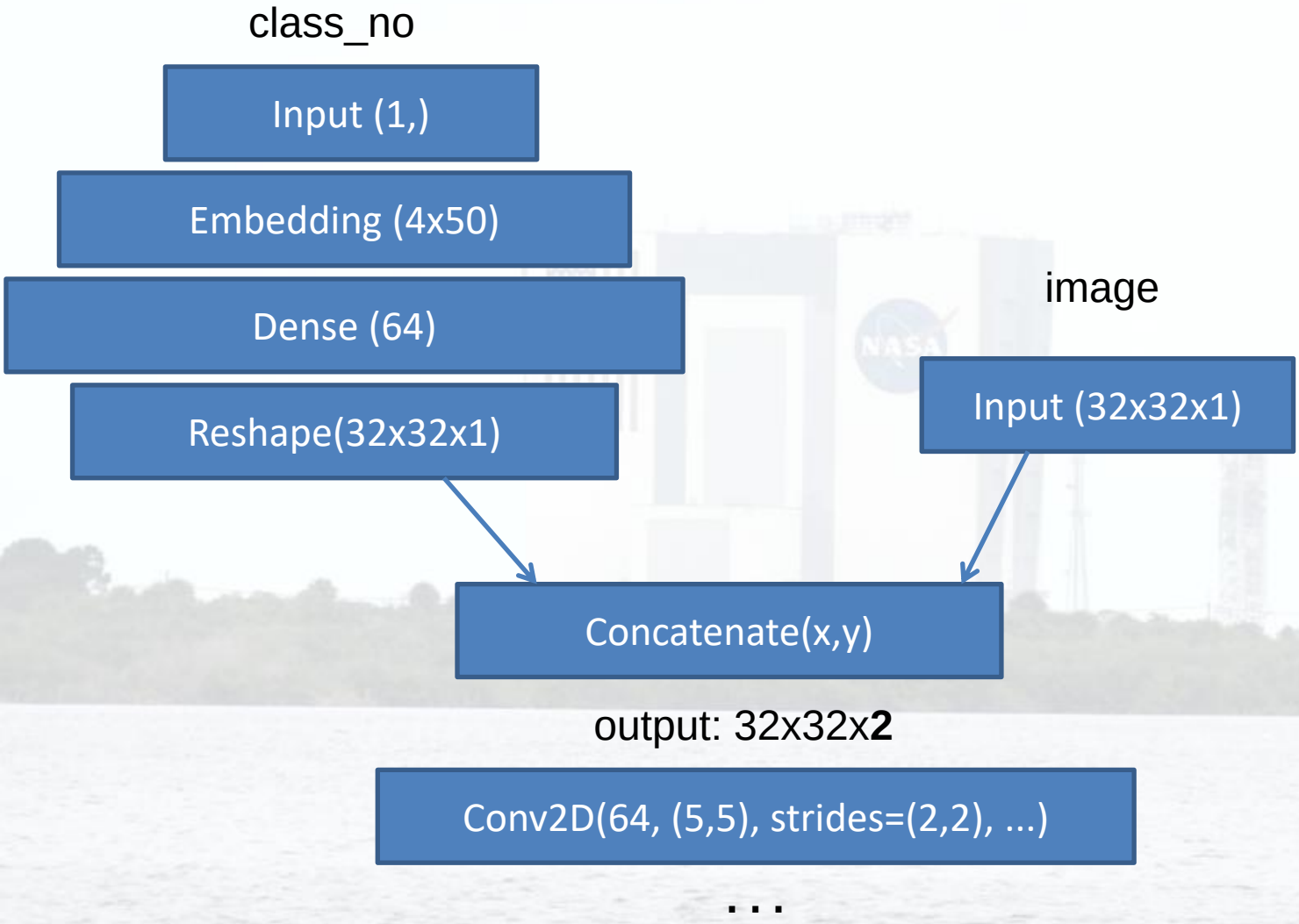
Changes in the architecture

- Generator:
  - [noise, class] -> [image]
- Discriminator
  - [image, class] -> [decision 1/0]

Generator: new input: class_no



Discriminator: new input: class_no



Changes in training

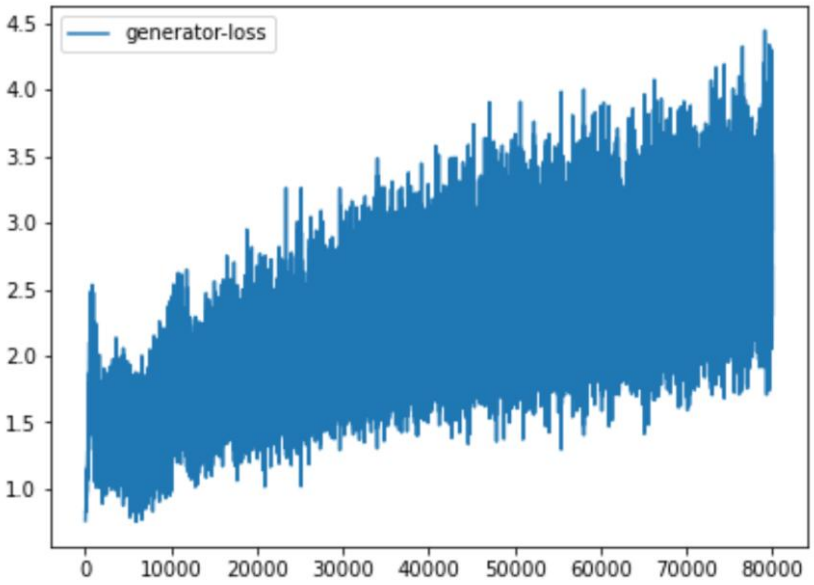
```
def do_step(images, labels):
...
noise = tf.random.normal([batch_size, noise_dim])
random_labels = np.random.randint(4, size=batch_size)
with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      # generate fake image using noise
      generated_images = generator([input_noise,random_labels], training=True)
      # evaluate fake images
      fake_output = discriminator([generated_images,random_labels], training=True)
      ...
     # evaluate real images
     real_output = discriminator([images,labels], training=True)
     ...
```

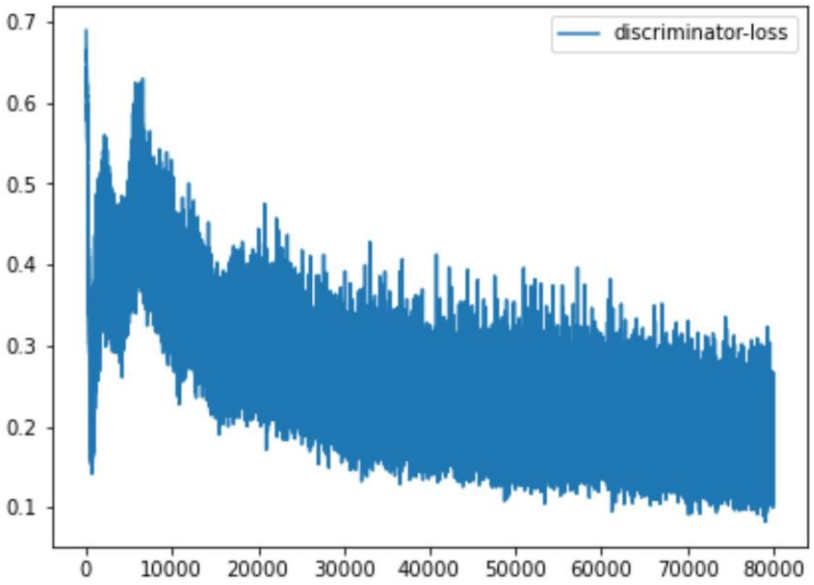
Results after 40k epochs

Results after 80k epochs

Loss changes for 80k epochs

- Not perfect...
- gan3_color.ipynb, check_model_4c.ipynb





Pix2Pix

- The network that converts one image to another
- Training:
  - input_image -> network -> output_image
- Problem:
  - the network should generalize
- Solution:
  - use GAN architecture with the discriminator

Working example

pix2pix.ipynb

- generator:
  - UNET network (encoder-decoder with residuals)
- discriminator:
  - PatchGAN
  - It does not return one value 1/0
  - It returns a matrix of 1/0 values
  - Every pixel in the matrix refers to some part of the image
    - the parts overlap!
  - The architecture works with images of any size!

Pix2Pix GAN architecture

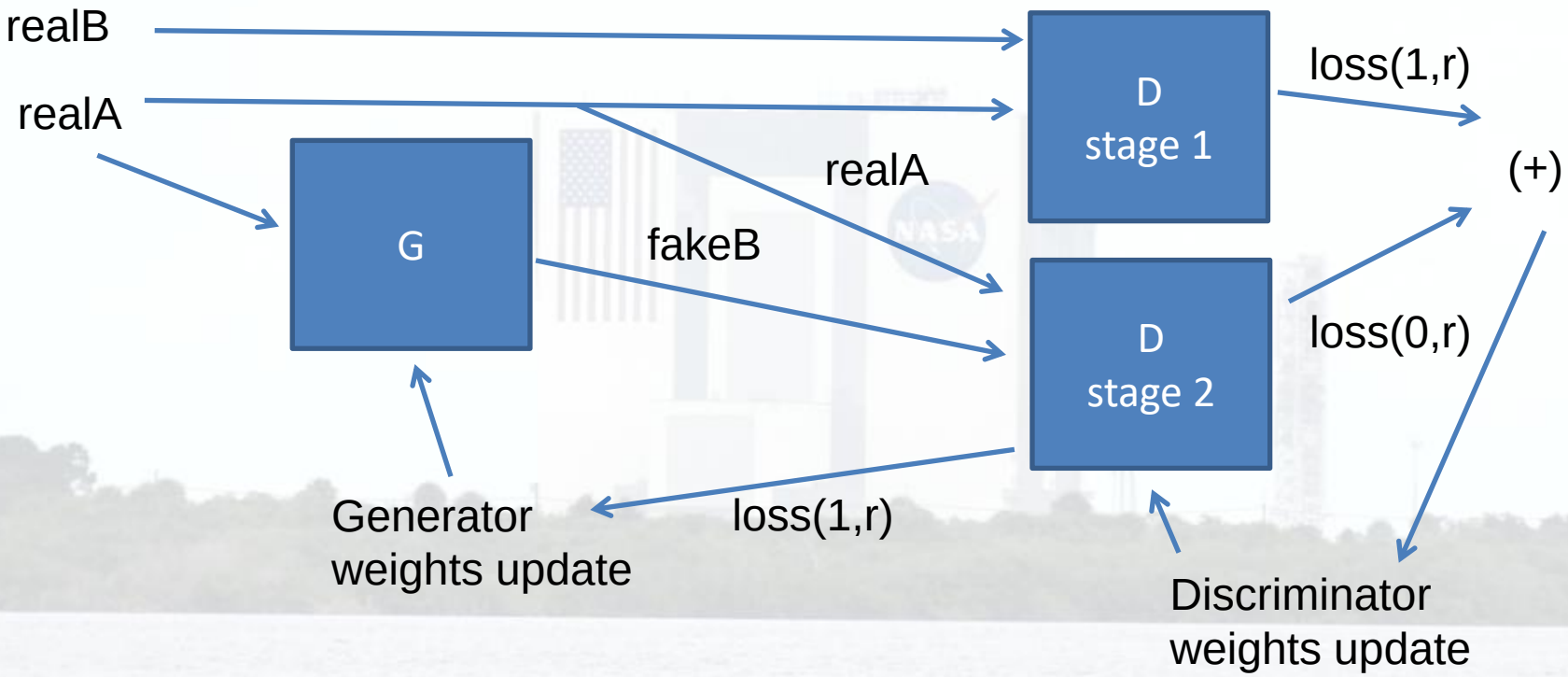
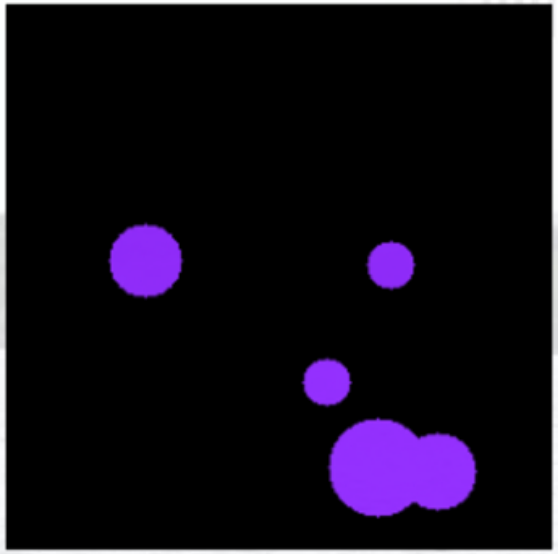
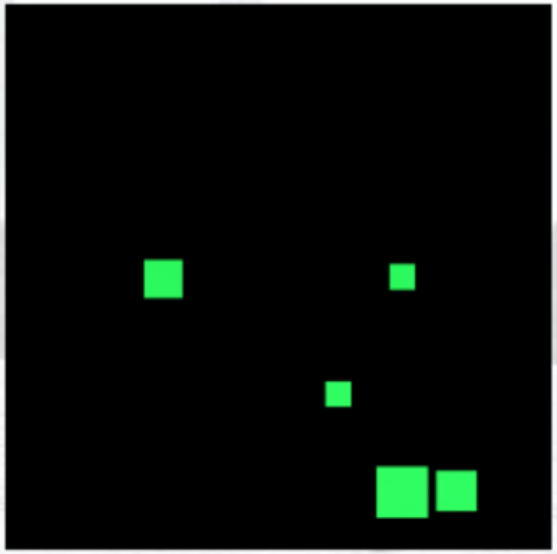


Image to image (pix2pix)

- Networks that converts one image to another
- Input and output images are different but there is **analogy** between them
- A simple example: turn violet circles to green rectangles



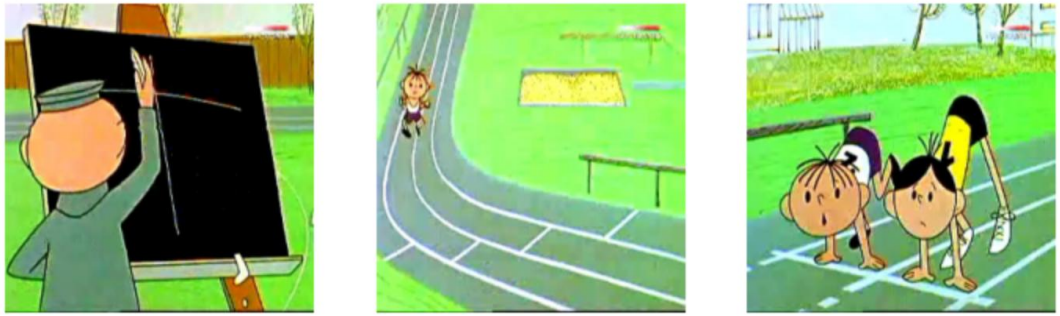


Colorization example

Black&White images

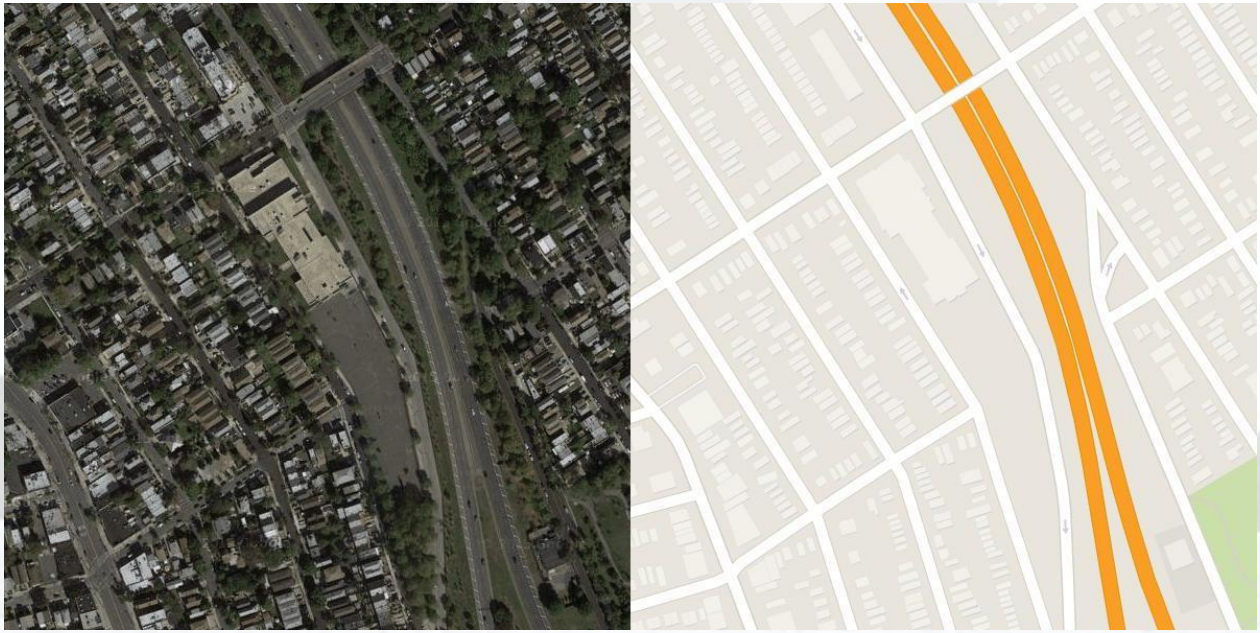
Colorized BW images

Original images



map2image example

- Load pairs: satellite image and google map
- ***pix2pix_map.ipynb***



Available datasets

- https://www.github.com/affinelayer/pix2pix-tensorflow-models.git static/models
  - facades
  - edges2cats
  - edges2shoes
  - edges2handbags
- Online example:
  - https://affinelayer.com/pixsrv/
- It is possible to start your own server:
  - cd server
  - serve.py --port 8001

CycleGAN

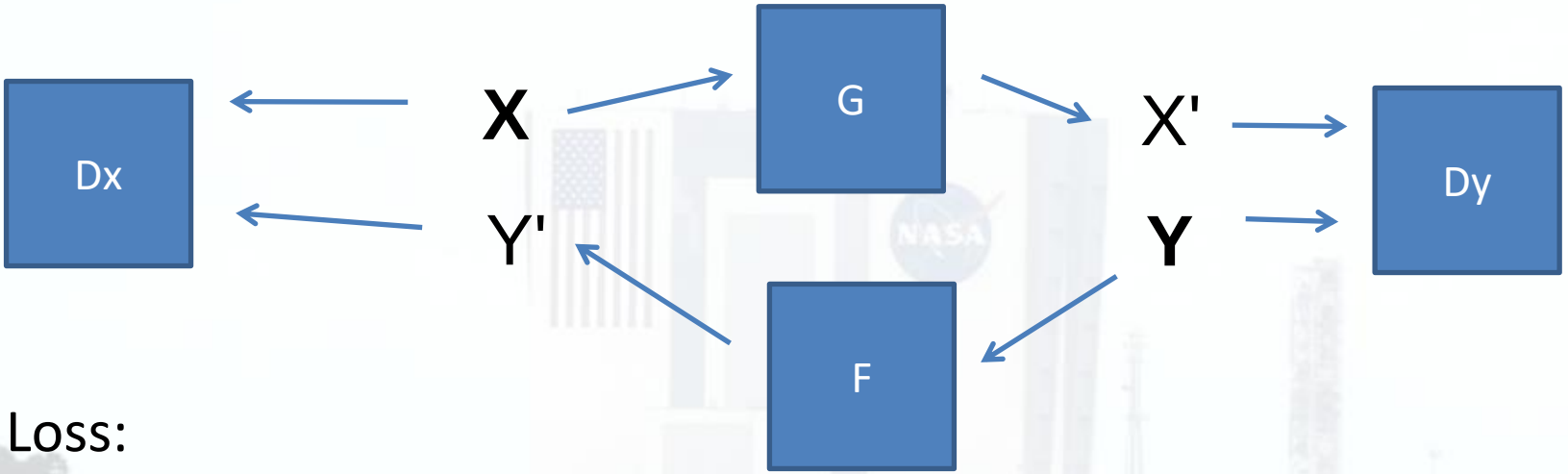
- Instead of preparing pairs of images we prepare sets of images
  - without one-to-one relationships!
- For example:
  - a set of female images (X)
  - a set of male images (Y)
- We train the network to generate images based on X that look like Y
- ...and images based on Y that look like X (a cycle!)

CycleGAN architecture

- Two generators: G and F
  - G translates X to Y
  - F translates Y to X
- Two discriminators Dx and Dy:
  - Dx checks if X is genuine or fake
  - Dy checks if Y is genuine or fake

CycleGAN architecture

Two sets of images: X and Y



Loss:

Dx_loss = bce(X,1)+bce(Y',0)

Dy_loss = bce(Y,1)+bce(X',0)

G_loss = bce(X',1)+...

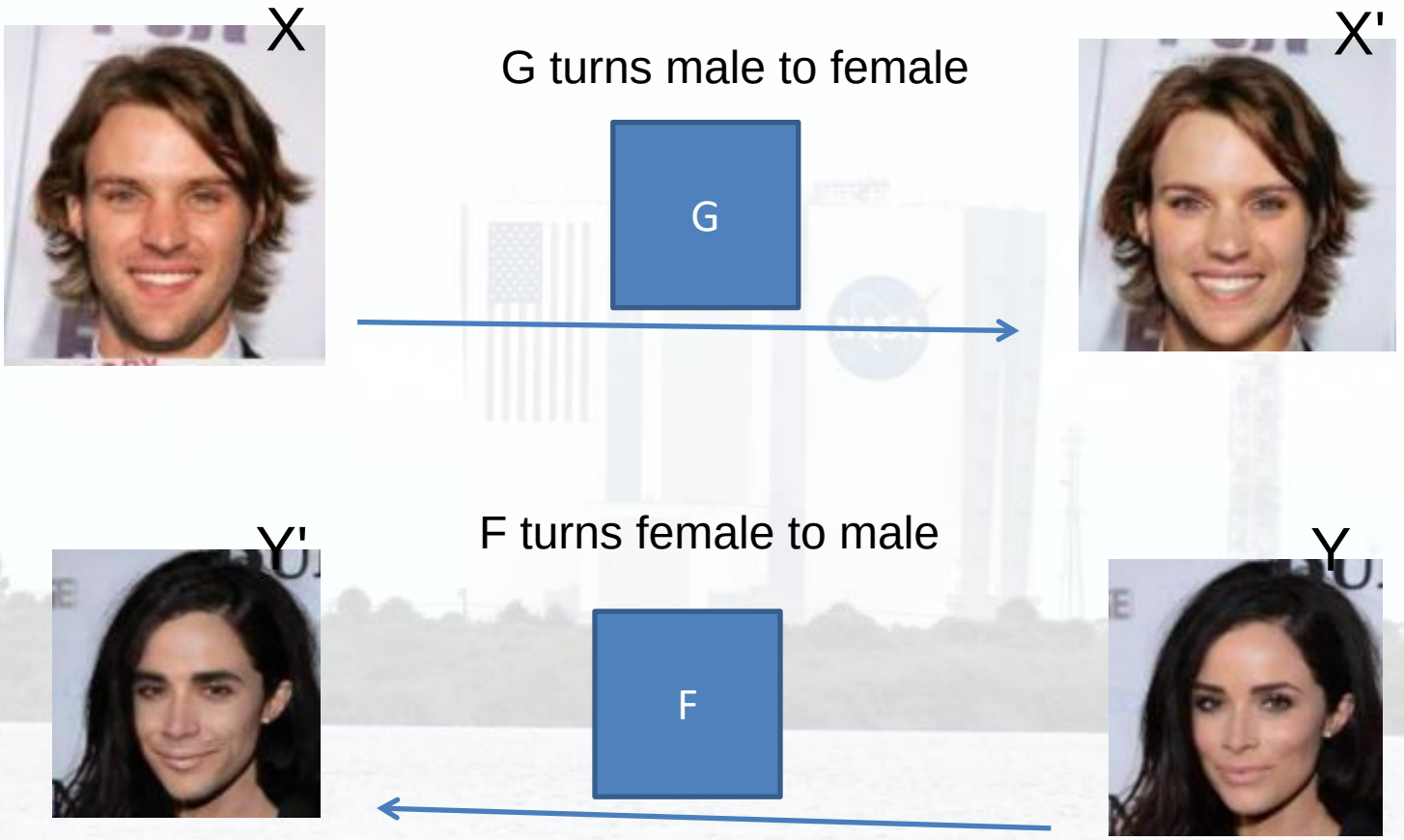
F_loss = bce(Y',1)+...

bce = binary crossentropy

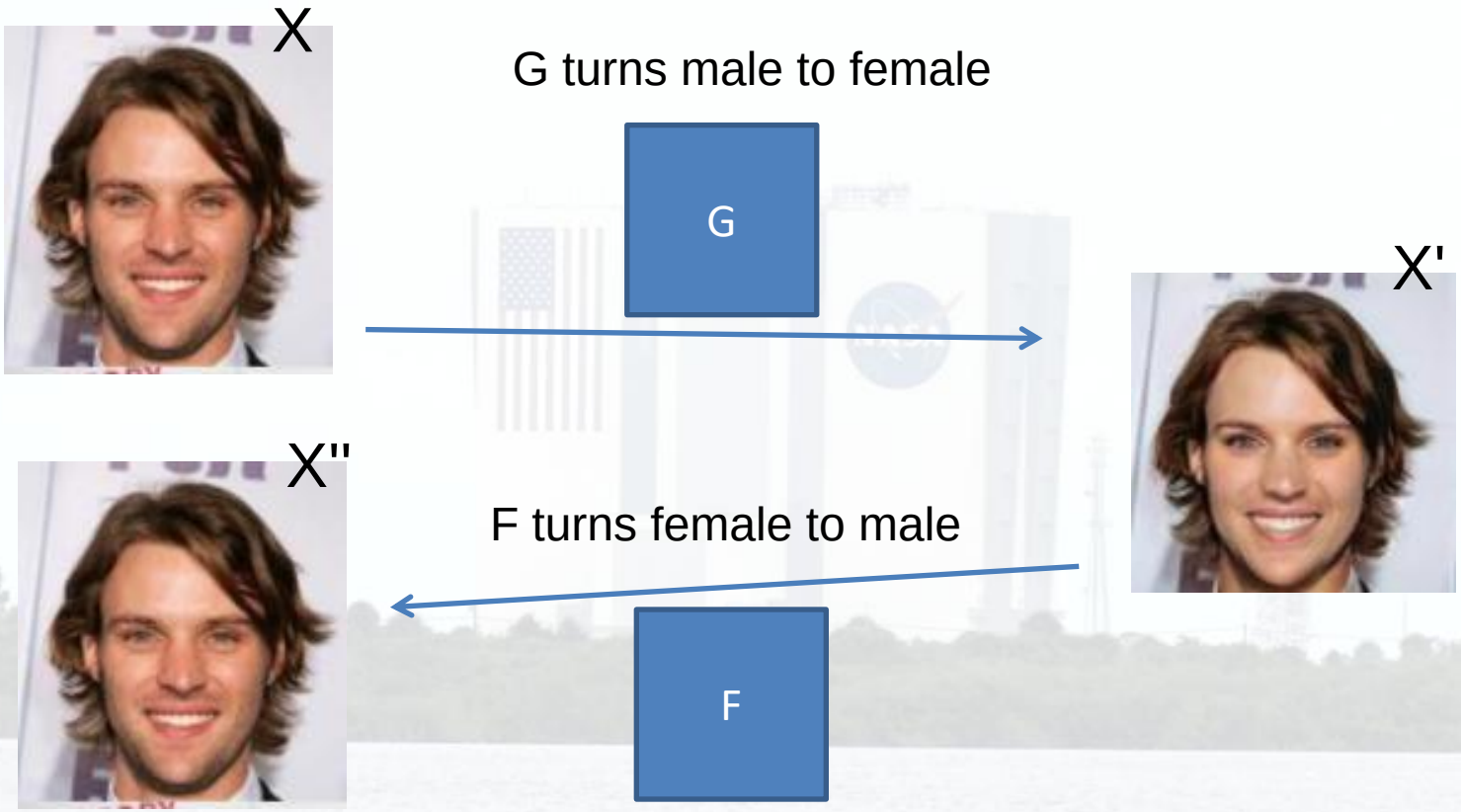
Two additional losses for CycleGAN

- **Cycle loss**: after the cycle the image should look the same
  - X' = G(X)
  - X'' = F(X')
  - cycle_loss_x = |X'' – X| = |F(G(X)) – X|
  - cycle_loss_y = |G(F(Y)) – Y|
  - total_cycle_loss = cycle_loss_x + cycle_loss_y
- **Identity loss**: after the "reverse generation" the image should look the same
  - identity_loss_x = |F(X) – X|
  - identity_loss_y = |G(Y) - Y|

For our example

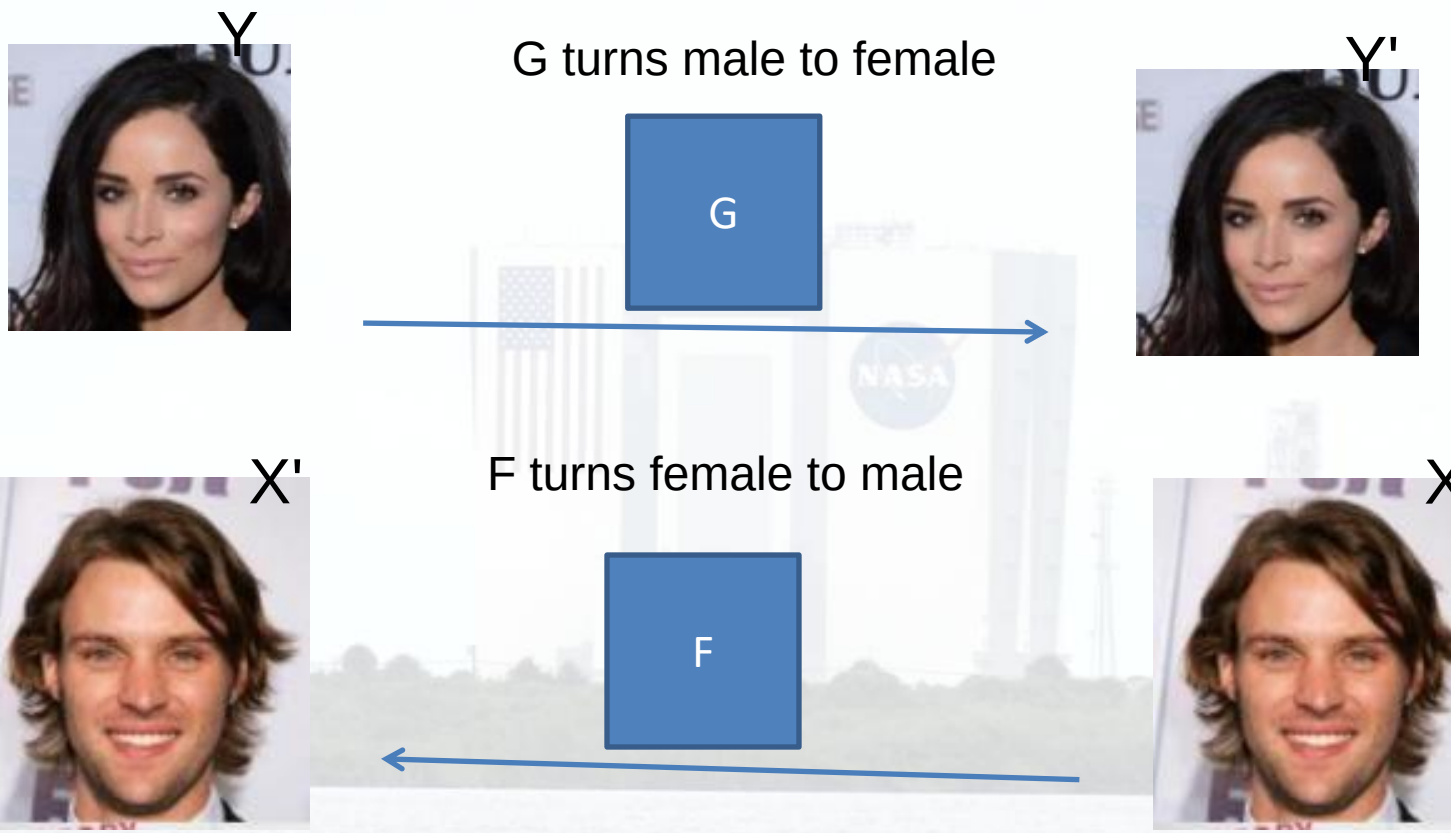


Cycle loss



X'' should be similar do X: cycle_loss_x = |X-X''|=|X-F(G(X)|

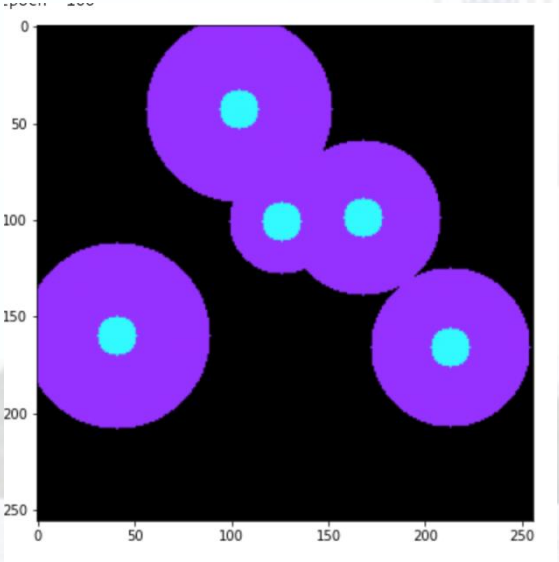
Identity loss

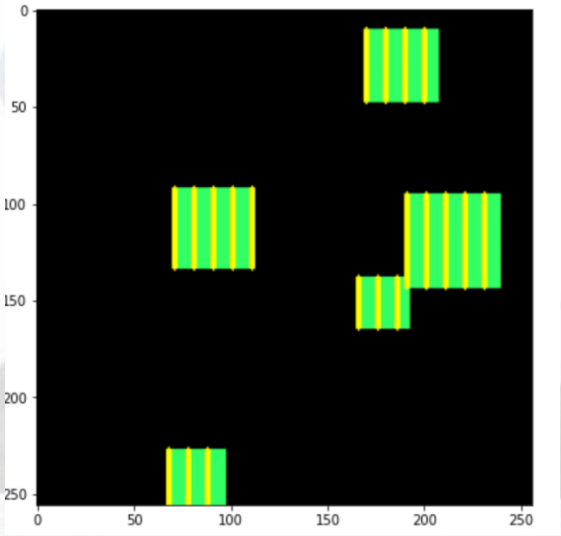


Turning male image to male should have no effect: identity_loss_x=|X-F(X)|

CycleGAN example

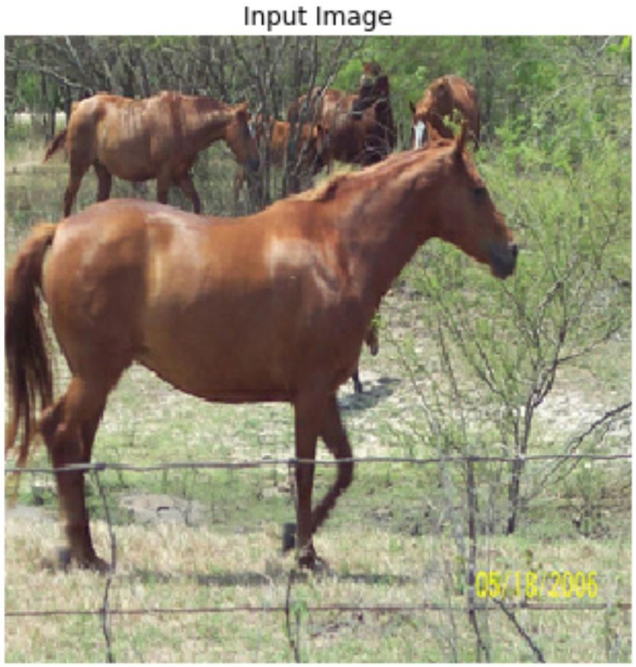
- ***cyclegan.ipynb***
- Changing circles to squares

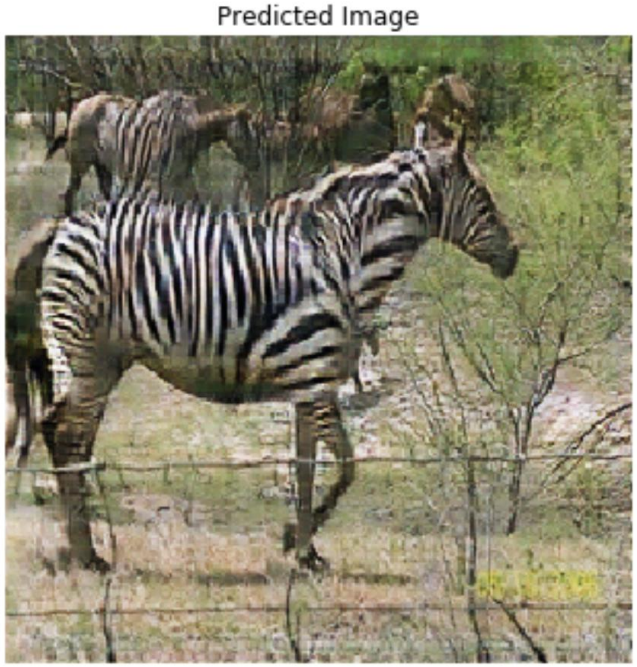




A classic example

- Notebook from the Tensorflow tutorial
  https://www.tensorflow.org/tutorials/generative/cyclegan
- Changing horses to zebras

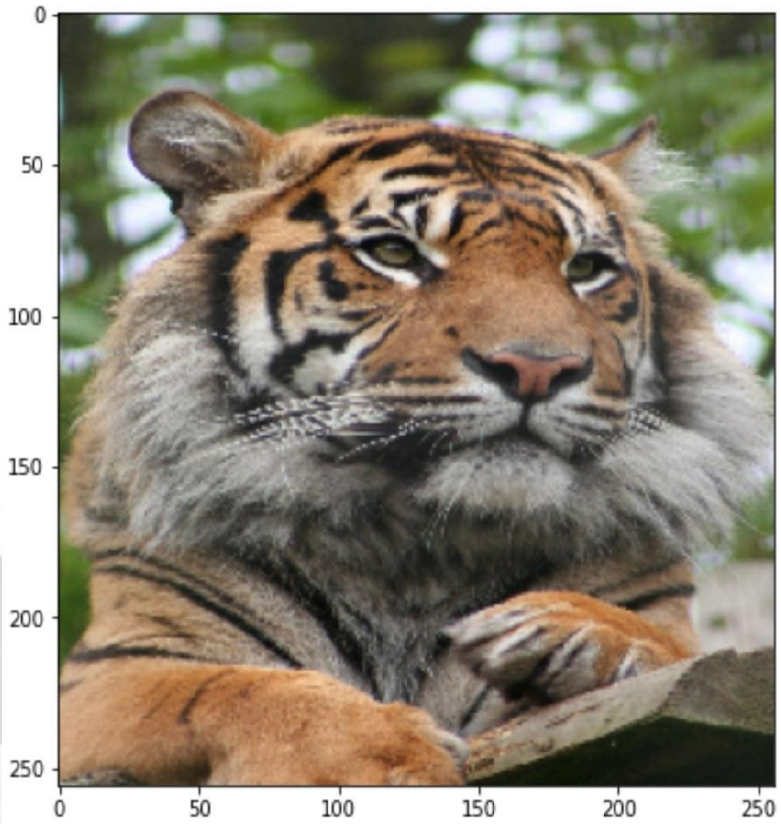


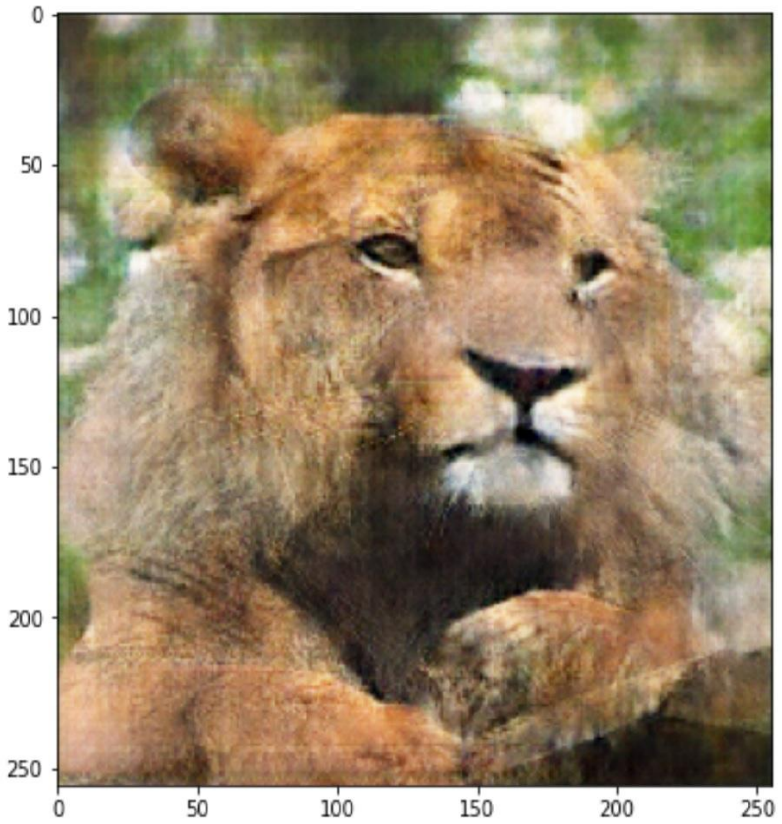


Other datasets

- https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/
- apple2orange.zip
- cezanne2photo.zip
- iphone2dslr_flower.zip
- monet2photo.zip
- summer2winter_yosemite.zip
- vangogh2photo.zip
- ...

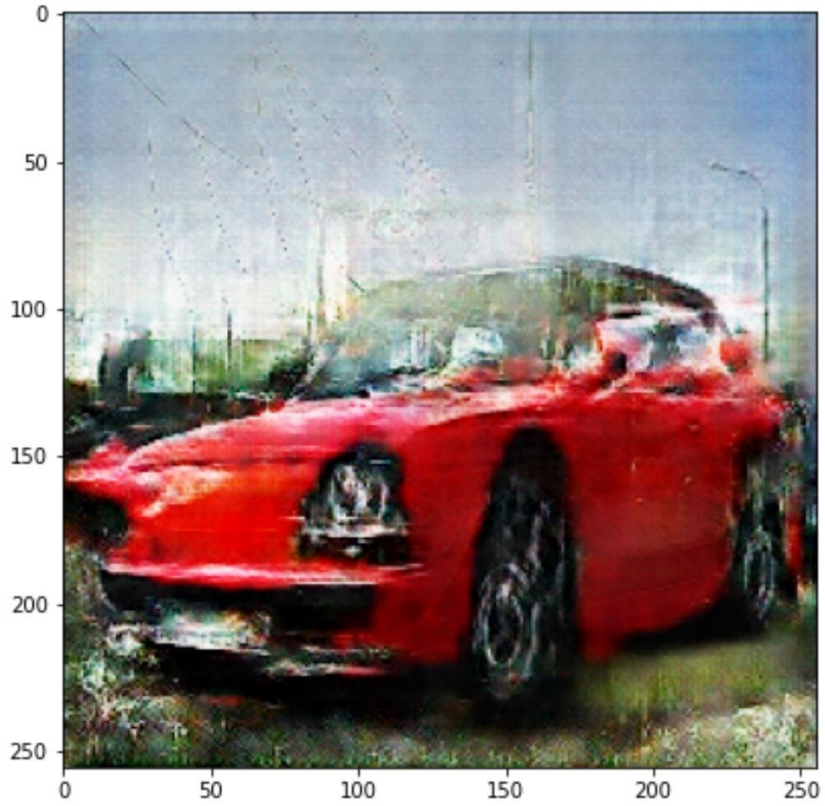
Tiger > Lion





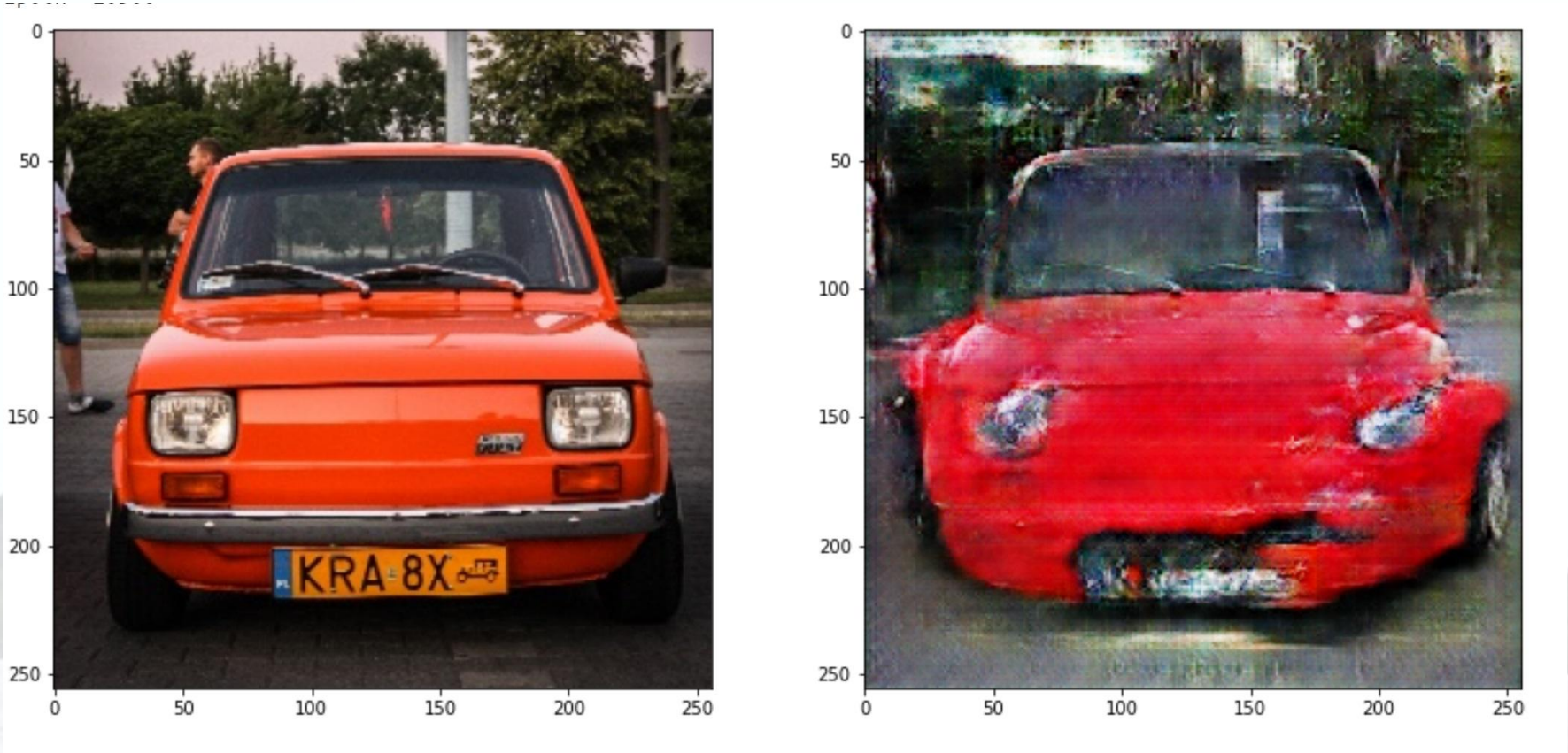
[Zuzanna Lempa]

Maluch > Ferrari



[Dawid Macha, Tomasz Strzoda]

Maluch > Ferrari



[Dawid Macha, Tomasz Strzoda]

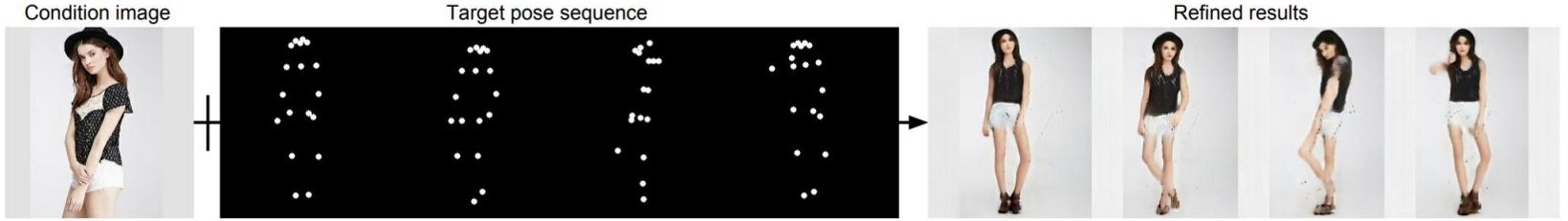
Examples of GANs

- Generator may get noise as input
- But it may also get objects
- There are many cool examples of GANs
- An interesting overview:
  - https://jonathan-hui.medium.com/gan-some-cool-applications-of-gans-4c9ecca35900

Pose Guided Person Image Generation

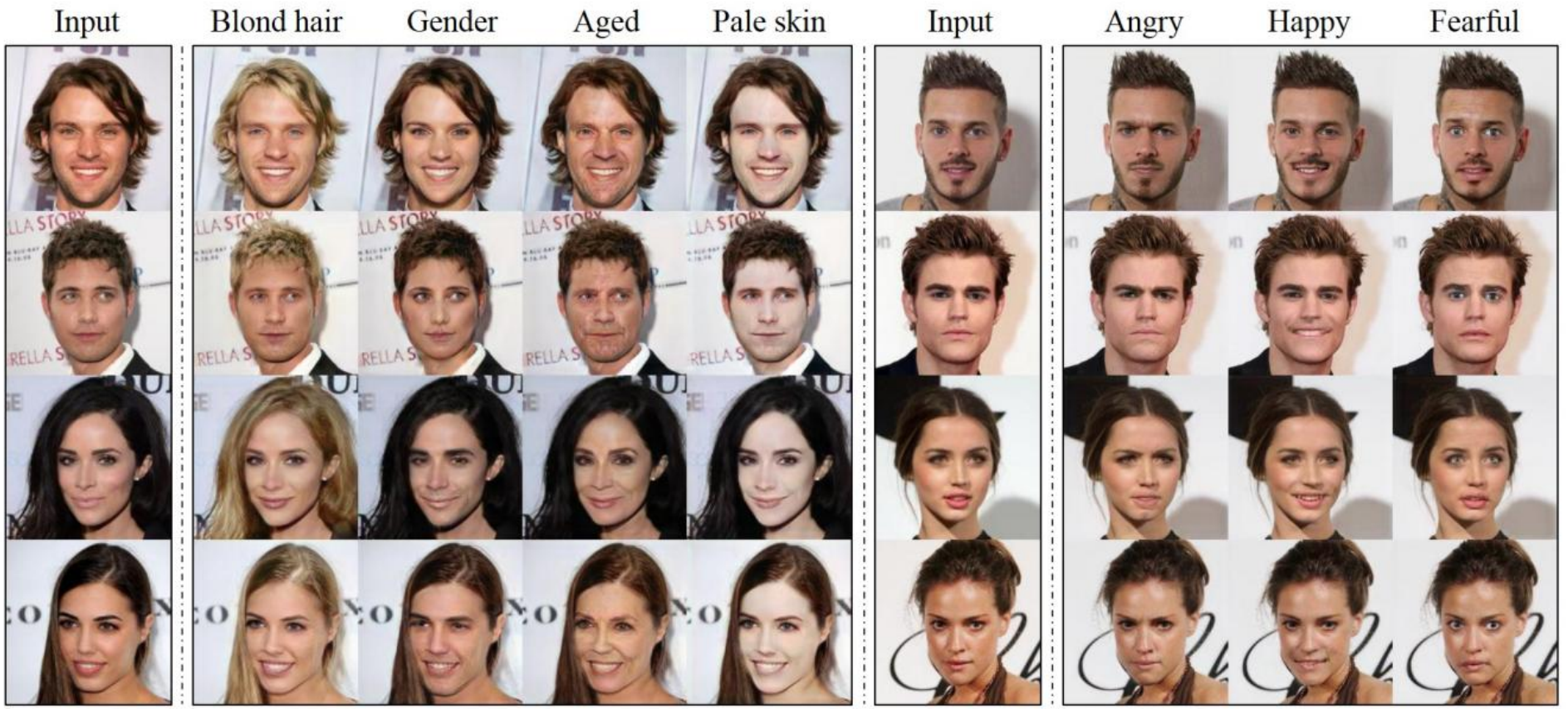
(a) DeepFashion

(b) Market-1501



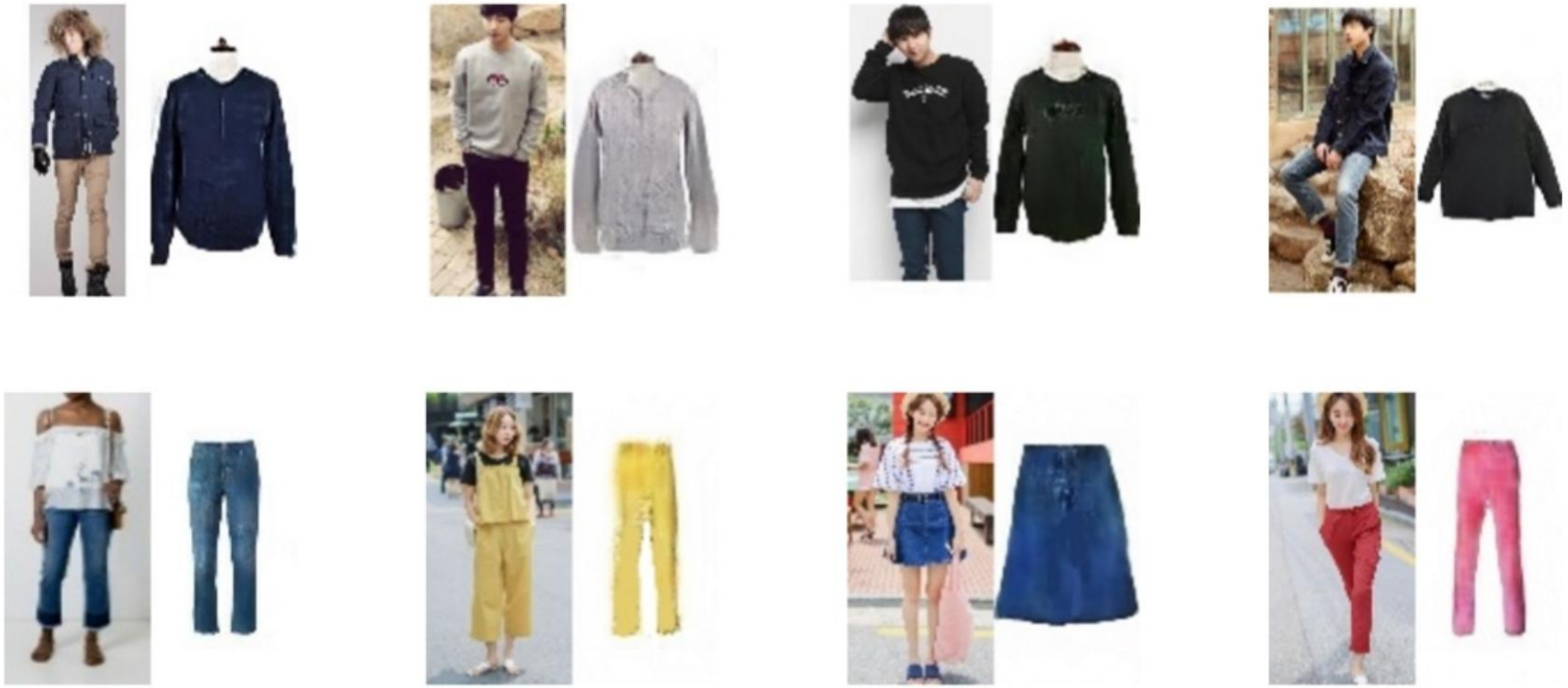
(c) Generating from a sequence of poses

Star GAN


Input
Blond hair
Gender
Aged
Pale skin
Input
Angry
Happy
Fearful

PixelDTGAN

Example results on LOOKBOOK dataset(top), left is input, right is generated clothes. Results on a similar dataset (bottom).
More results will be added soon.



SRGAN

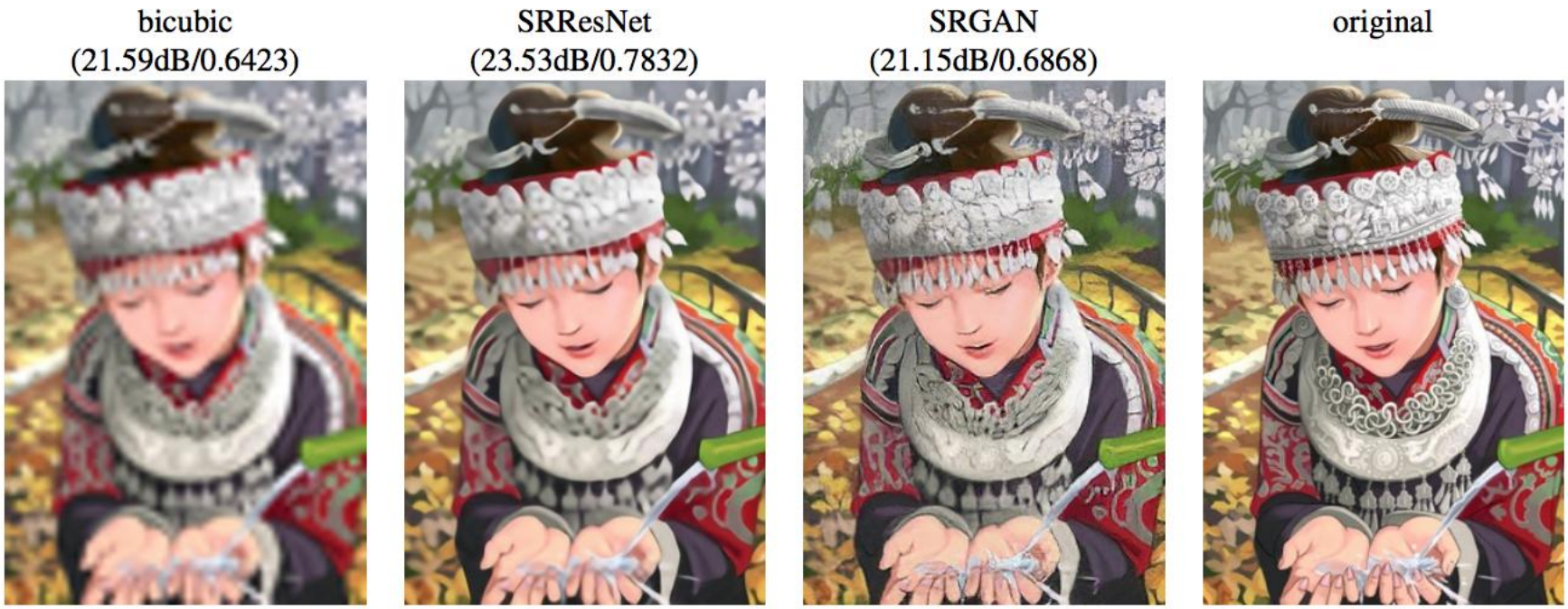


Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

ProgressiveGAN

Figure 5: 1024 × 1024 images generated using the CELEBA-HQ dataset. See Appendix F for a larger set of results, and the accompanying video for latent space interpolations.

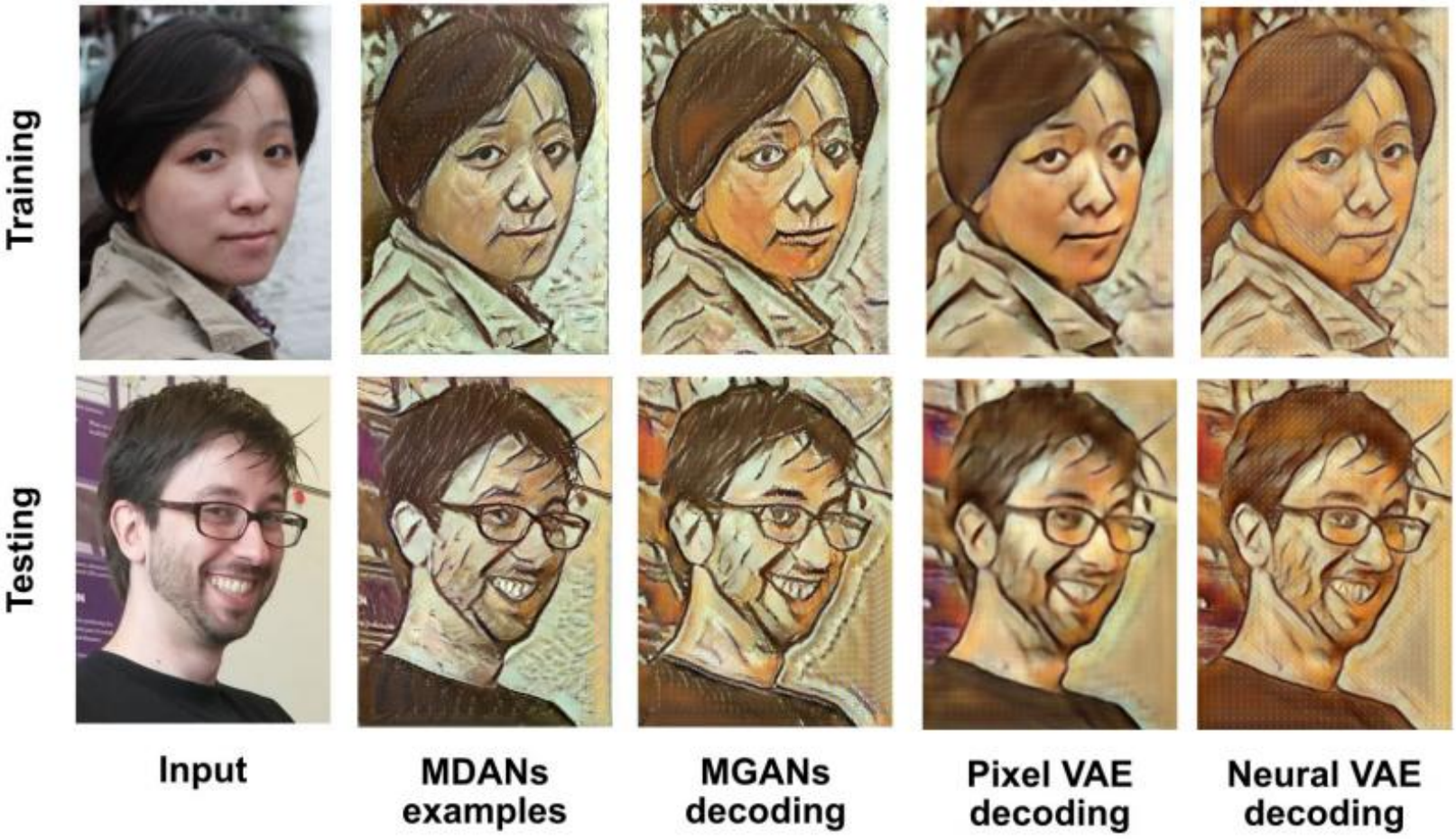
StackGAN

Text to image

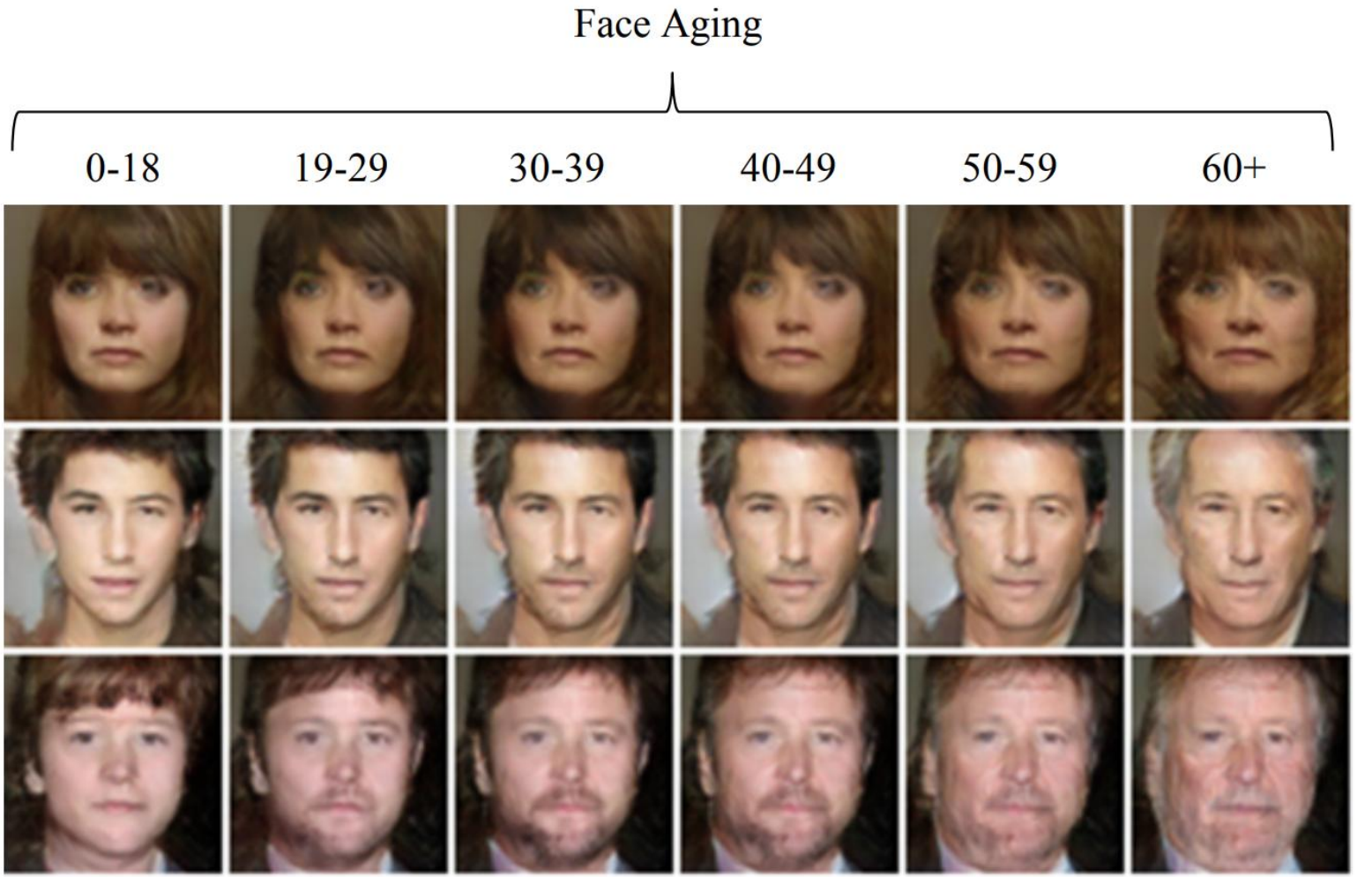
DiscoGAN

(b) Handbag images (input) & **Generated** shoe images (output)

MGAN


Training
Testing
Input
MDANs examples
MGANs decoding
Pixel VAE decoding
Neural VAE decoding

Age-cGAN



AnoGAN

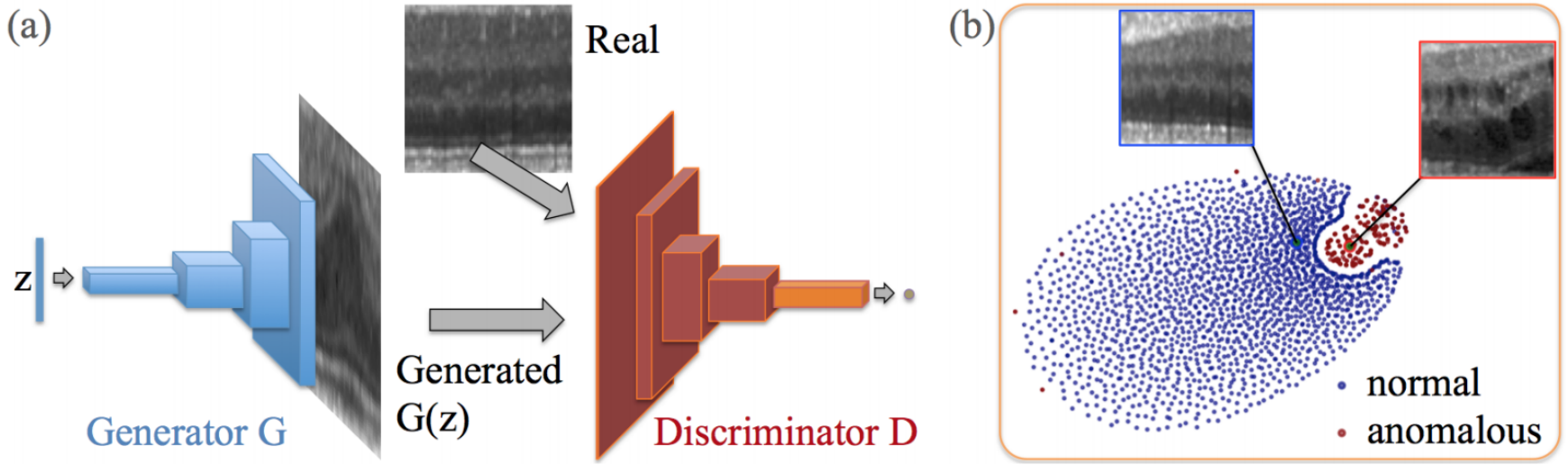


Fig. 2. (a) Deep convolutional generative adversarial network. (b) t-SNE embedding of normal (blue) and anomalous (red) images on the feature representation of the last convolution layer (orange in (a)) of the discriminator.

Other

- 3D model generation from pictures
- Music generation
- Video generation
- All these examples with references and more:
- https://jonathan-hui.medium.com/gan-some-cool-applications-of-gans-4c9ecca35900

Summary

- GANs are very powerful
- The only element they need is a dataset
- GANs may create images, video, music, 3D models, eye movements,...
- The simplest GANs create object based on some random noise
- The more sophisticated GANs need something more (e.g. input picture)