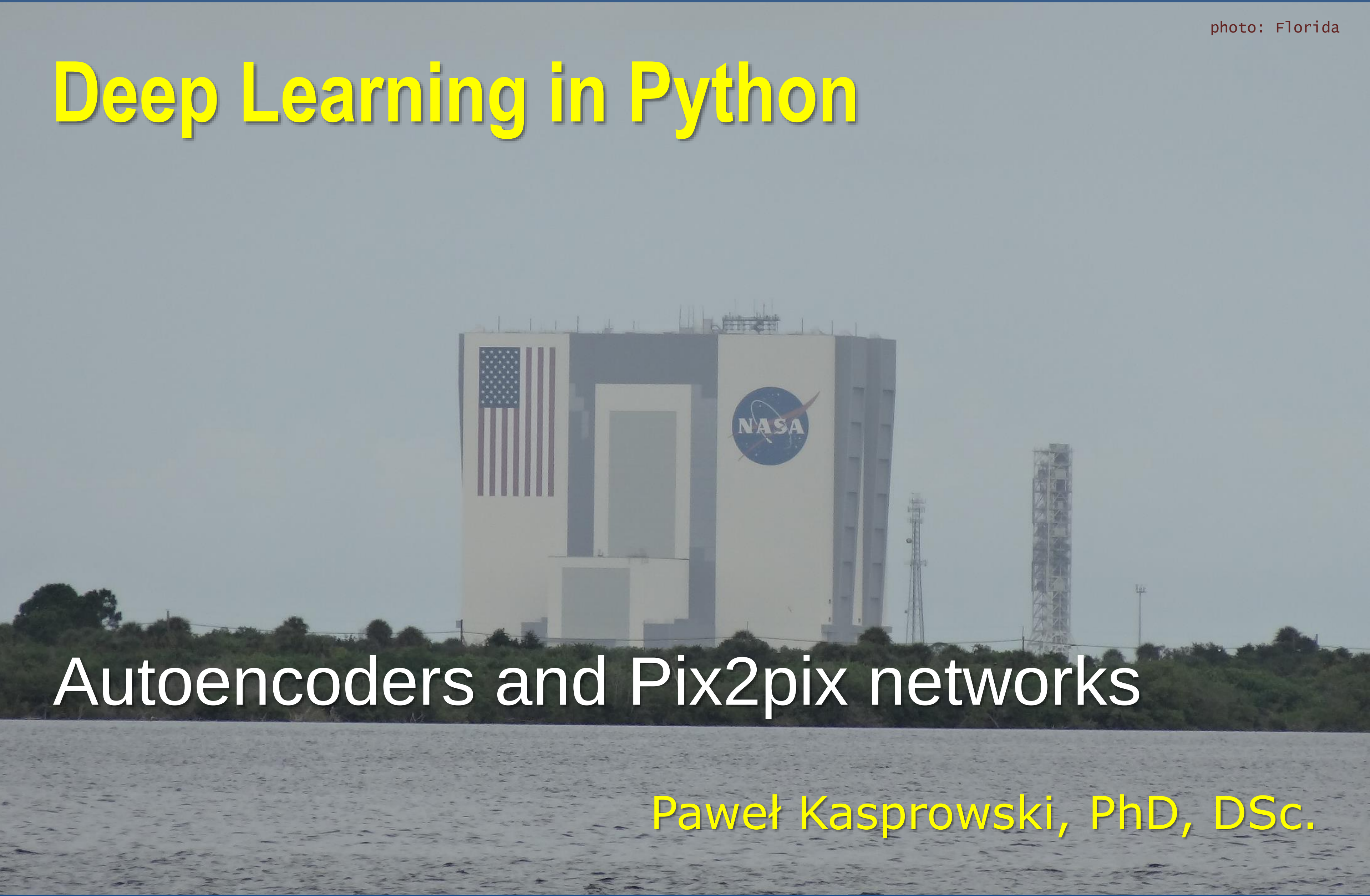
photo: Florida

Deep Learning in Python



Autoencoders and Pix2pix networks

Paweł Kasprowski, PhD, DSc.

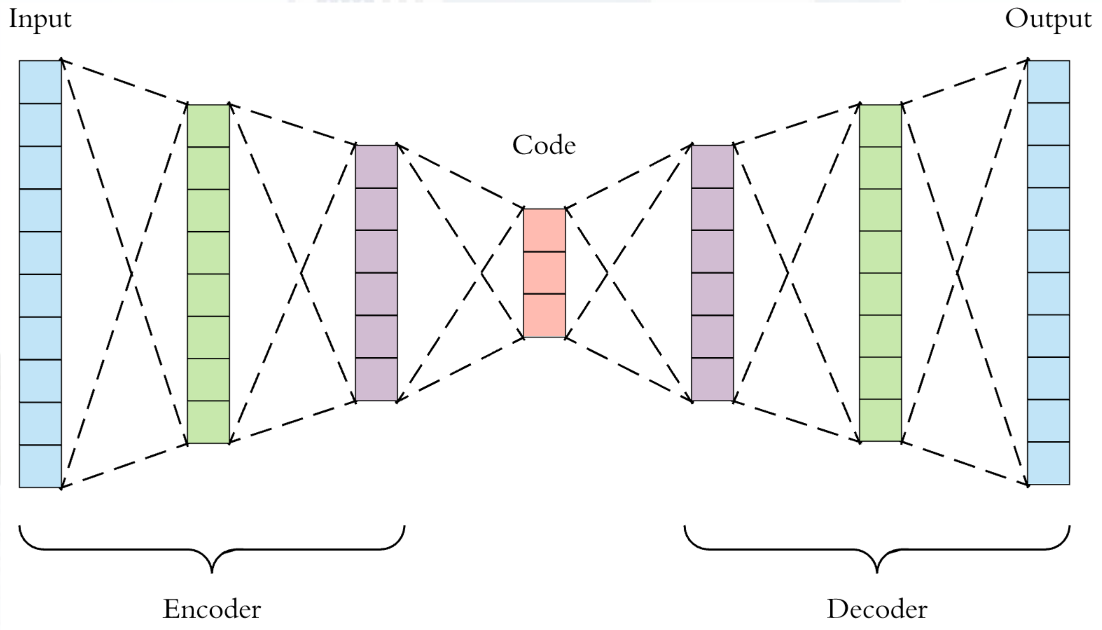
pawel.kasprowski@polsl.pl

Picture to picture

- The network that generates an image
- Training:
  - input_image -> network -> output_image
- Problem:
  - the network should generalize

Autoencoders

- The network that consists of:
  - Encoder – converts an image into a vector (code)
  - Decoder – converts the code into an image



https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

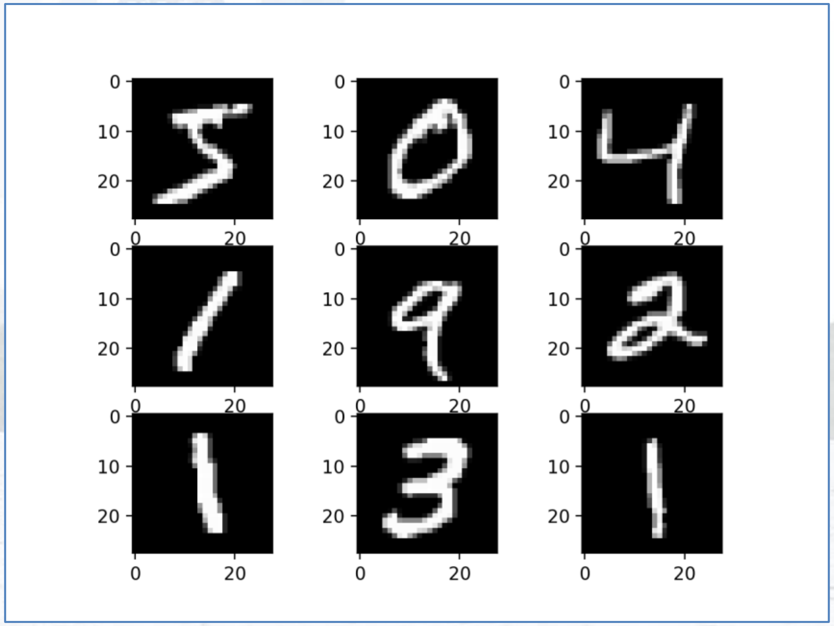
pawel.kasprowski@polsl.pl

Encoder

- The sample is recalculated to lower dimension
- For instance:
  – image (200x200x3) is encoded to the vector (100)
- The idea:
  – this compressed (latent) representation preserves **the most important** properties of the original object
  – it will be possible to reconstruct **the same** object from the latent representation

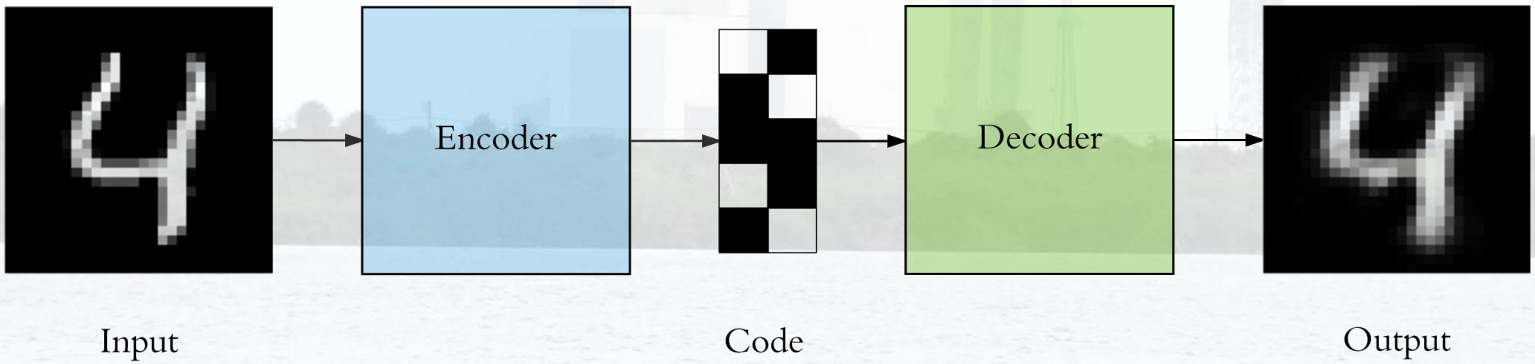
MNIST dataset

- Handwritten digits
- 10 classes
- 60,000 training examples
- 10,000 test examples
- size: 28x28x1



The idea

- Encode to the latent vector of size=code_size
- Decode to the original image
- Training: the same image as input and output!



https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

The simplest autoencoder

autoencoders1.ipynb

- Simple dense network (gets flattened images):

```
code_size = 5
input_img = Input(shape=(28*28,))
code = Dense(code_size, activation='relu')(input_img)
output_img = Dense(28*28, activation='sigmoid')(code)
autoencoder = Model(input_img, output_img)
```

- Training:
  - autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
  - autoencoder.fit(trainSamples, trainSamples, epochs=5)

Testing the network

- For different code_size: 1, 5, 10, 100
- For more sophisticated architecture with two hidden layers:

```
input_img = Input(shape=(input_size,))
hidden_1 = Dense(128, activation='relu')(input_img)
code = Dense(code_size, activation='relu')(hidden_1)
hidden_2 = Dense(128, activation='relu')(code)
output_img = Dense(input_size, activation='sigmoid')(hidden_2)
```

Denoising autoencoder

autoencoders2.ipynb

- Creating noisy samples:

```
noise_factor = 0.4
trainSamples_noisy = trainSamples +
            noise_factor * np.random.normal(size=trainSamples.shape)
trainSamples_noisy = np.clip(trainSamples_noisy, 0.0, 1.0)
```

- Train using noisy samples:
  - autoencoder.fit(trainSamples_noisy, trainSamples, epochs=5)
- Works better for noisy than for sharp!

 pawel.kasprowski@polsl.pl

Analysing codes

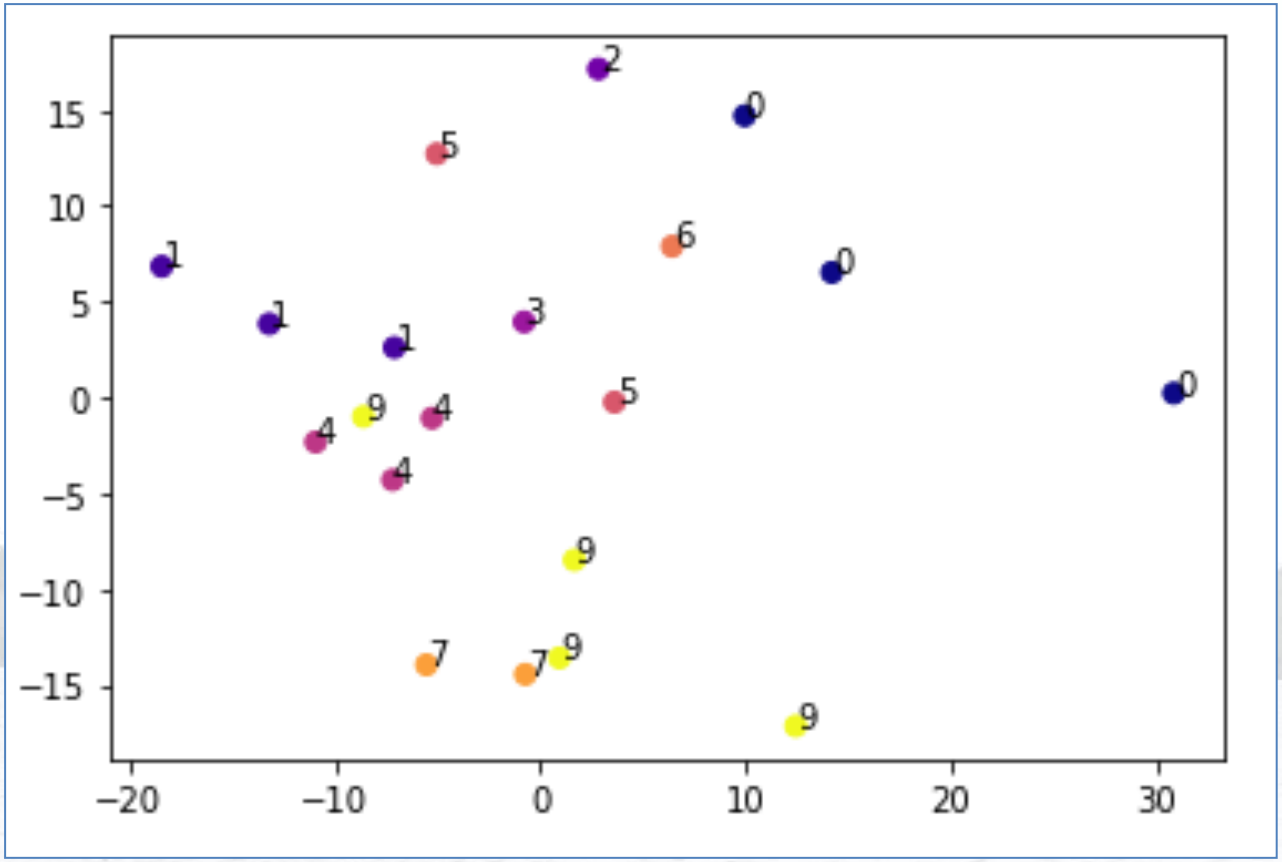
- Codes for the same digits should be similar
- Let's map the codes to 2D and analyse on plot
- We will use Principal Component Analysis (PCA) to map it

```
from sklearn.decomposition import PCA
encoder = Model(input_img, code)
vectors = encoder.predict(testSamples[:20]) # get codes
pca = PCA(n_components=2)
vectors2D = pca.fit_transform(vectors) # transform to 2D
plt.scatter(vectors2D[:,0],vectors2D[:,1], c=testLabels[:points])
for i,w in enumerate(vectors):
    plt.annotate(testLabels[i],(vectors2D[i,0],vectors2D[i,1]))
```

pawel.kasprowski@polsl.pl

Results

- For code_size=32 digits in similar places



Classification of encoded vectors

- Prepare vectors:

```
encoder = Model(input_img, code)
testVectors = encoder.predict(testSamples)
trainVectors = encoder.predict(trainSamples)
```

- Use kNN to classify vectors

```
knn_model = KNeighborsClassifier()
knn_model.fit(trainVectors, trainLabels)
predLabels = knn_model.predict(testVectors)
```

- Results:
  - over 97% for each class!

pawel.kasprowski@polsl.pl

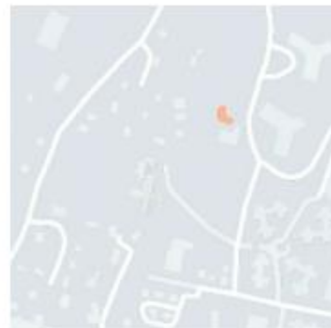
Autoencoder applications

- Compression
  - but only for specific data
  - typically jpeg algorithm is better...
- Denoising
  - requires examples!
- Providing latent space vector for future analysis
  - for instance for classification using kNN

pawel.kasprowski@polsl.pl

Map example

- Input google maps



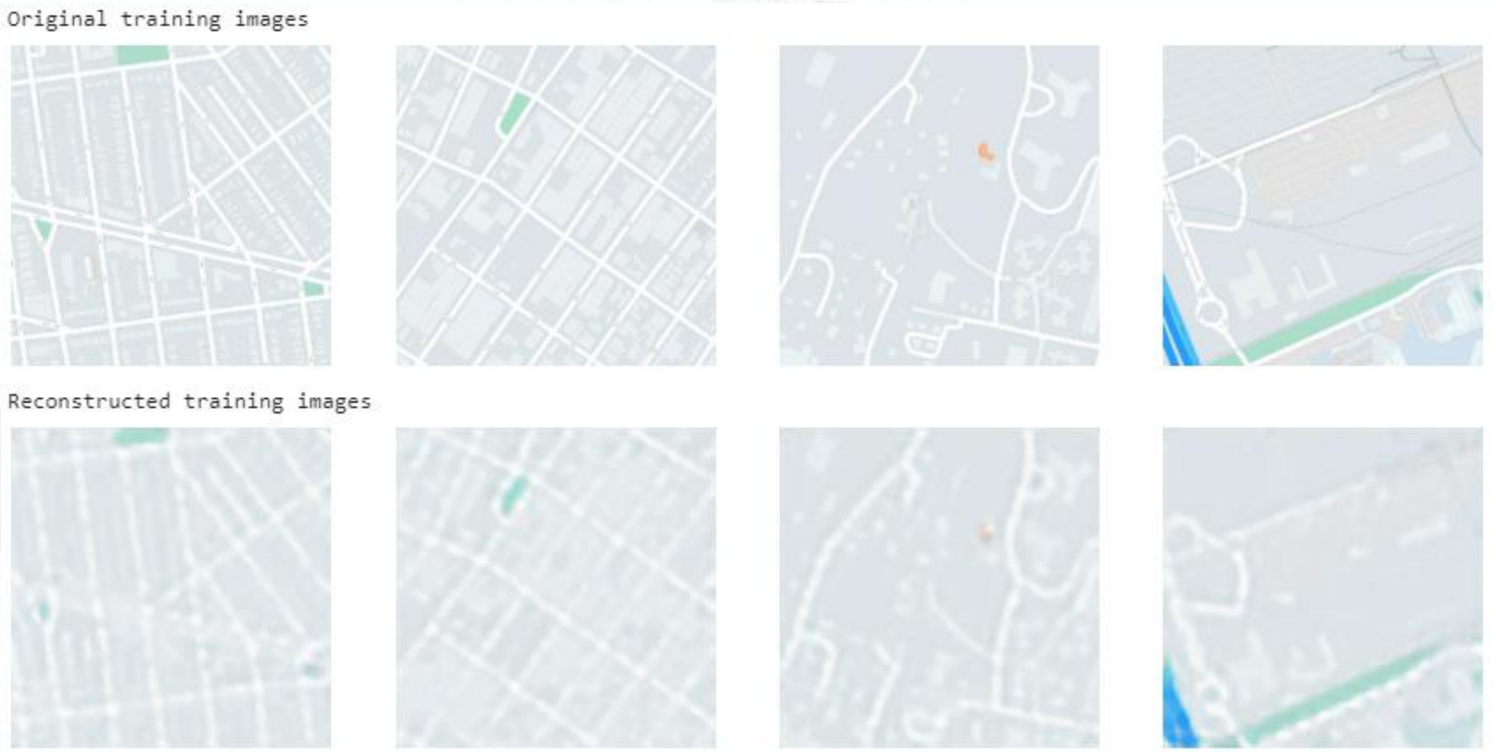

- Create autoencoders
- Three architectures:
  - Conv2D
  - Upsampling
  - UNET

autoencoder_models.py

pawel.kasprowski@polsl.pl

Upsampling

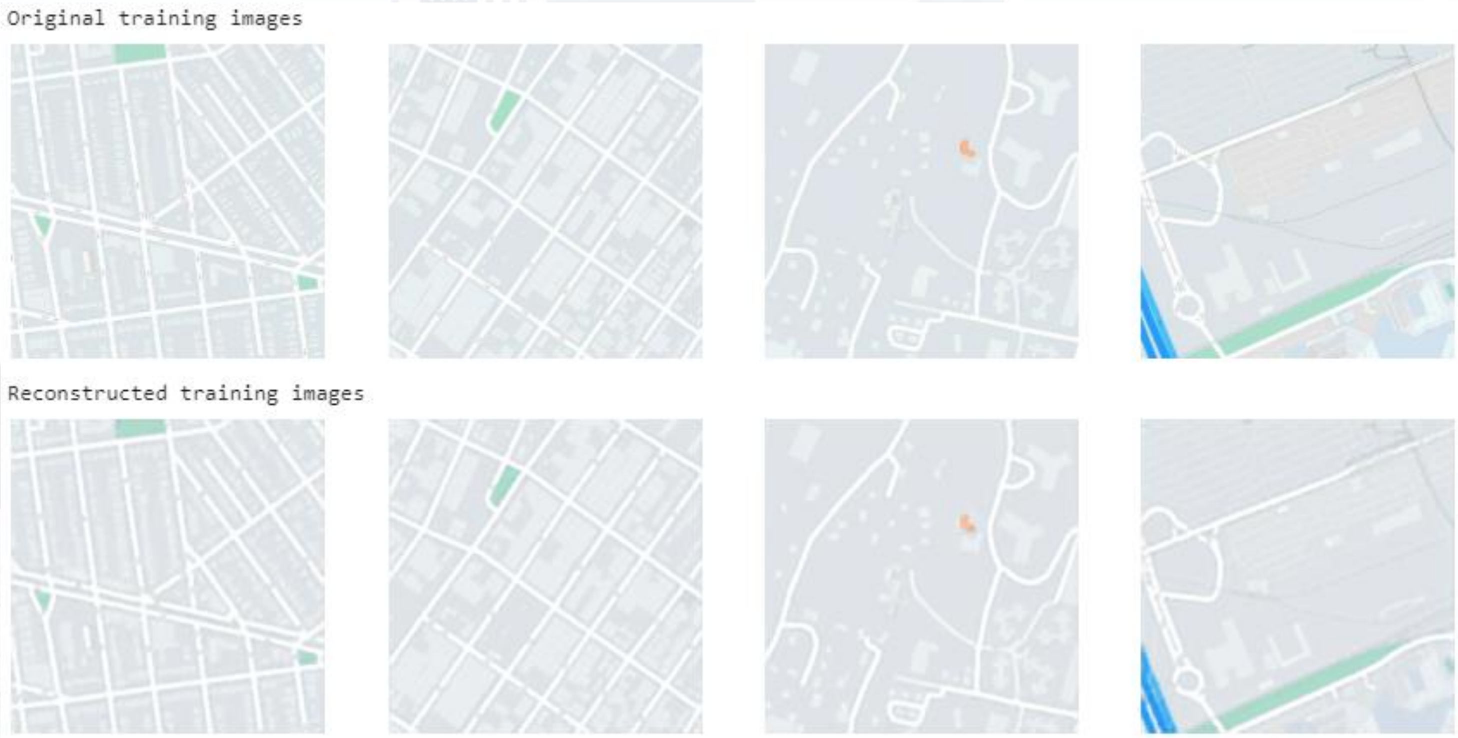
- Uses UpSampling2D to recontruct the image
- Result: blurred (***autoencoders_map_upsampling.ipynb***)



pawel.kasprowski@polsl.pl

Conv2DTranspose

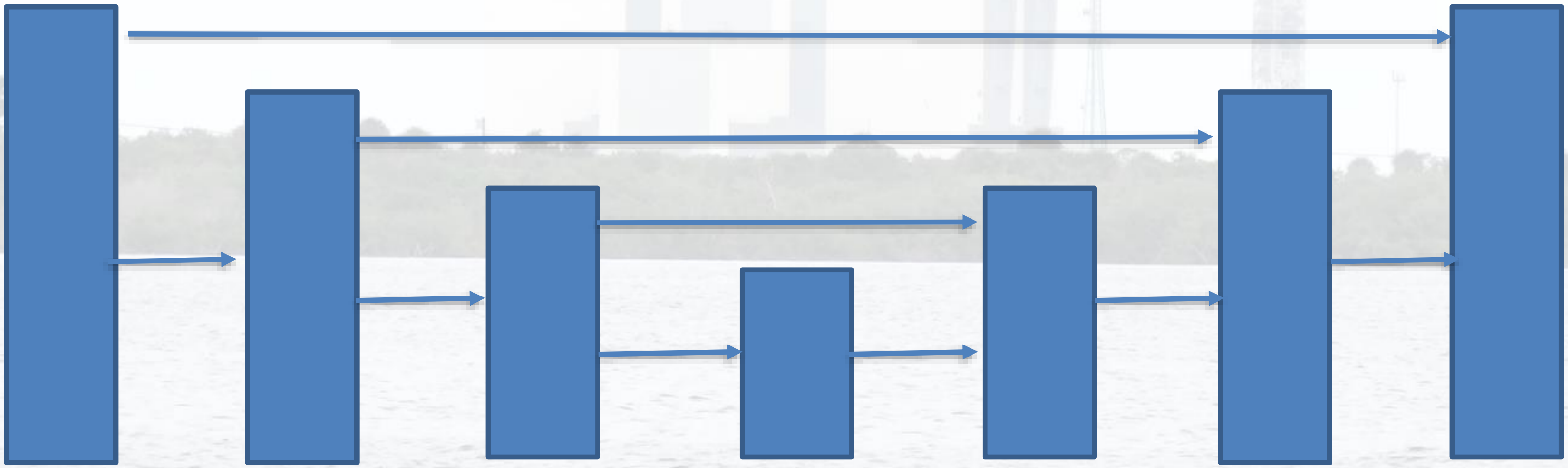
- Used Conv2DTranspose layers to decode
- Result: much better and with less iterations
  autoencoders_map_transpose.ipynb



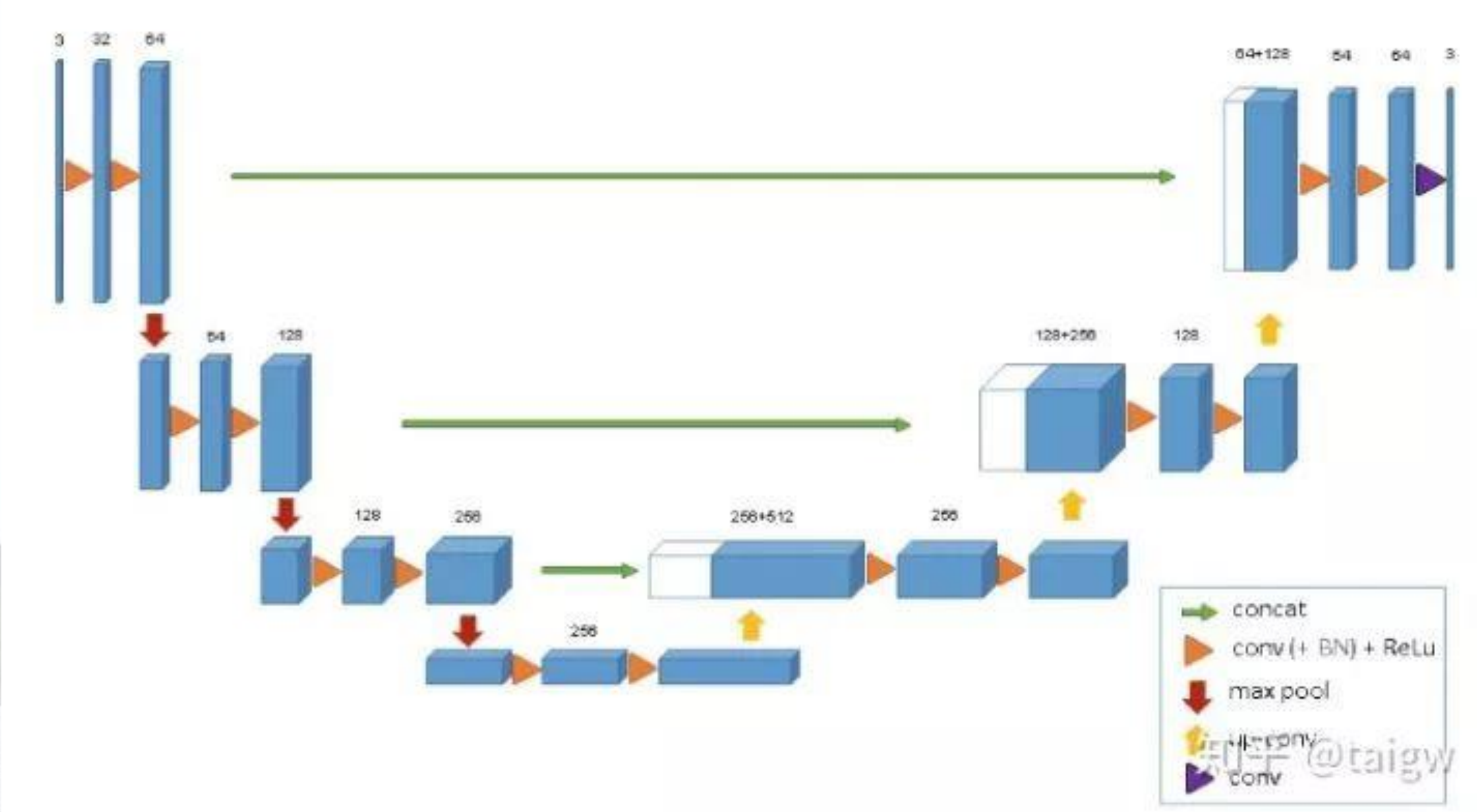
pawel.kasprowski@polsl.pl

U-NET

- A special kind of encoder-decoder network
- N encoder layers
- N decoder layers
- Every i-th encoder layer is connected with (N-i) decoder layer



Why U-NET?



https://programmer.group/unet-network-magic-changes-those-things.html

 pawel.kasprowski@polsl.pl

UNET simplified architecture

```
input = Input(...)
e1 = encoder_block(input,layers, filters,...)
e2 = encoder_block(e1,...)
e3 = encoder_block(e2,...)
b = Conv2D(...)(e3)
d1 = decoder_block(b, e3,...)
d2 = decoder_block(d1, e2,...)
d3 = decoder_block(d2, e1,...)
output = Activation('tanh')(d3)
model = Model(input,output)
```

UNET simplified architecture

```
input = Input(...)
e1 = encoder_block(input,layers, filters,...)
    e2 = encoder_block(e1,...)
        e3 = encoder_block(e2,...)
            b = Conv2D(...)(e3)
        d1 = decoder_block(b, e3,...)
    d2 = decoder_block(d1, e2,...)
d3 = decoder_block(d2, e1,...)
output = Activation('tanh')(d3)
model = Model(input,output)
```

pawel.kasprowski@polsl.pl

encoder

```
def encoder_block(layer_in, n_filters, batchnorm=True):
    g = Conv2D(n_filters, (4,4), strides=(2,2), padding='same',
            kernel_initializer=init)(layer_in)
    if batchnorm:
            g = BatchNormalization()(g, training=True)
    g = LeakyReLU(alpha=0.2)(g)
    return g
```

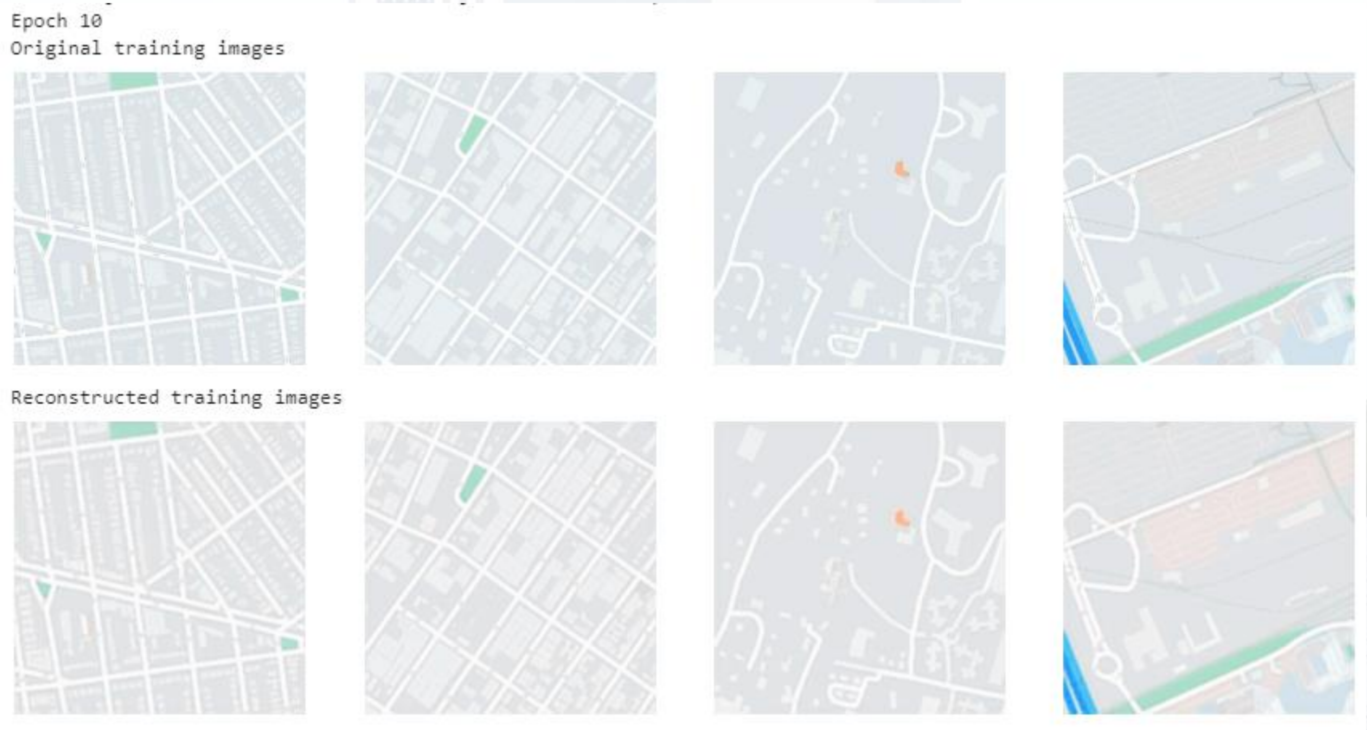
decoder

```
def decoder_block(layer_in, skip_in, n_filters, dropout=True):
    g = Conv2DTranspose(n_filters, (4,4), strides=(2,2),
            padding='same', kernel_initializer=init)(layer_in)
    g = BatchNormalization()(g, training=True)
    if dropout:
            g = Dropout(0.5)(g, training=True)
    g = Concatenate()([g, skip_in]) # merge with skip connection
    g = Activation('relu')(g)
    return g
```

pawel.kasprowski@polsl.pl

Maps with U-NET

- Good results in few epochs
- Not surprising – there are direct connections!
  autoencoder_map_unet.ipynb



Real applications

Some examples:

- Colorization
  - BW image -> color image
- Super-resolution
  - image 64x64 > image 256x256
- Image segmentation:
  - https://keras.io/examples/vision/oxford_pets_image_segmentation/
- Creating analogy
  - satellite image -> map

Colorization

- Simple CNN network (***colorize.ipynb***)
  - N x N x 1 -> N x N x 3
- Architecture:

```
input_img = Input(shape=image_shape)
x = Conv2D(filters = 16, kernel_size = (3, 3), activation='relu',
    padding='same')(input_img)
x = Conv2D(filters = 32, kernel_size = (3, 3), activation='relu',
    padding='same')(x)
x = Conv2D(filters = 64, kernel_size = (3, 3), activation='relu',
    padding='same')(x)
output_img = Conv2D(3, (3, 3),  padding='same')(x)
model = Model(input_img, output_img)
```

Colorization

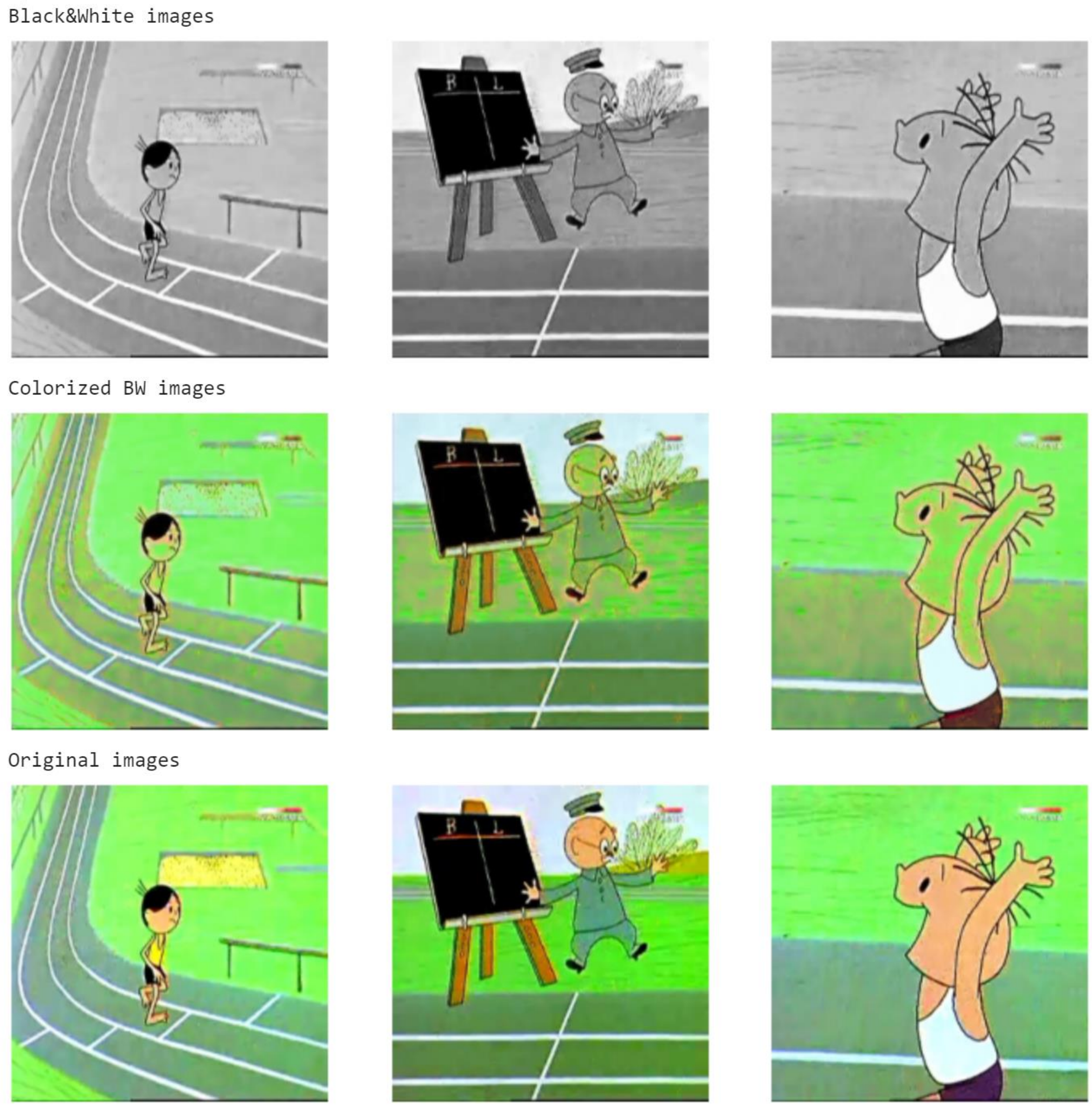
- Simple CNN network
  - N x N x 1 -> N x N x 3
- Architecture (simplified notation):

```
args = {"activation": "relu","padding": "same", "kernel_size": (3,3)}
input_img = Input(shape=image_shape)
x = Conv2D(filters = 16, **args)(input_img)
x = Conv2D(filters = 32, **args)(x)
x = Conv2D(filters = 64, **args)(x)
output_img = Conv2D(3, (3, 3),  padding='same')(x)
model = Model(input_img, output_img)
```

pawel.kasprowski@polsl.pl

Results

- Not very good...
- ...but not very bad as well!



Important property of CNN

- The number of weights for CNN is independent of the image resolution!
- Conv2D(filters = 16, kernel_size = (3, 3)) always has
  - 16*3*3 + 16 = 160 weights
  - regardless of an image size!
- The next layer Conv2D(32,(3,3)) always has
  - 16* 32*3*3+32 = 4640 weights
  - regardless of an image size!
- Pure CNN models work for images with any size!

pawel.kasprowski@polsl.pl

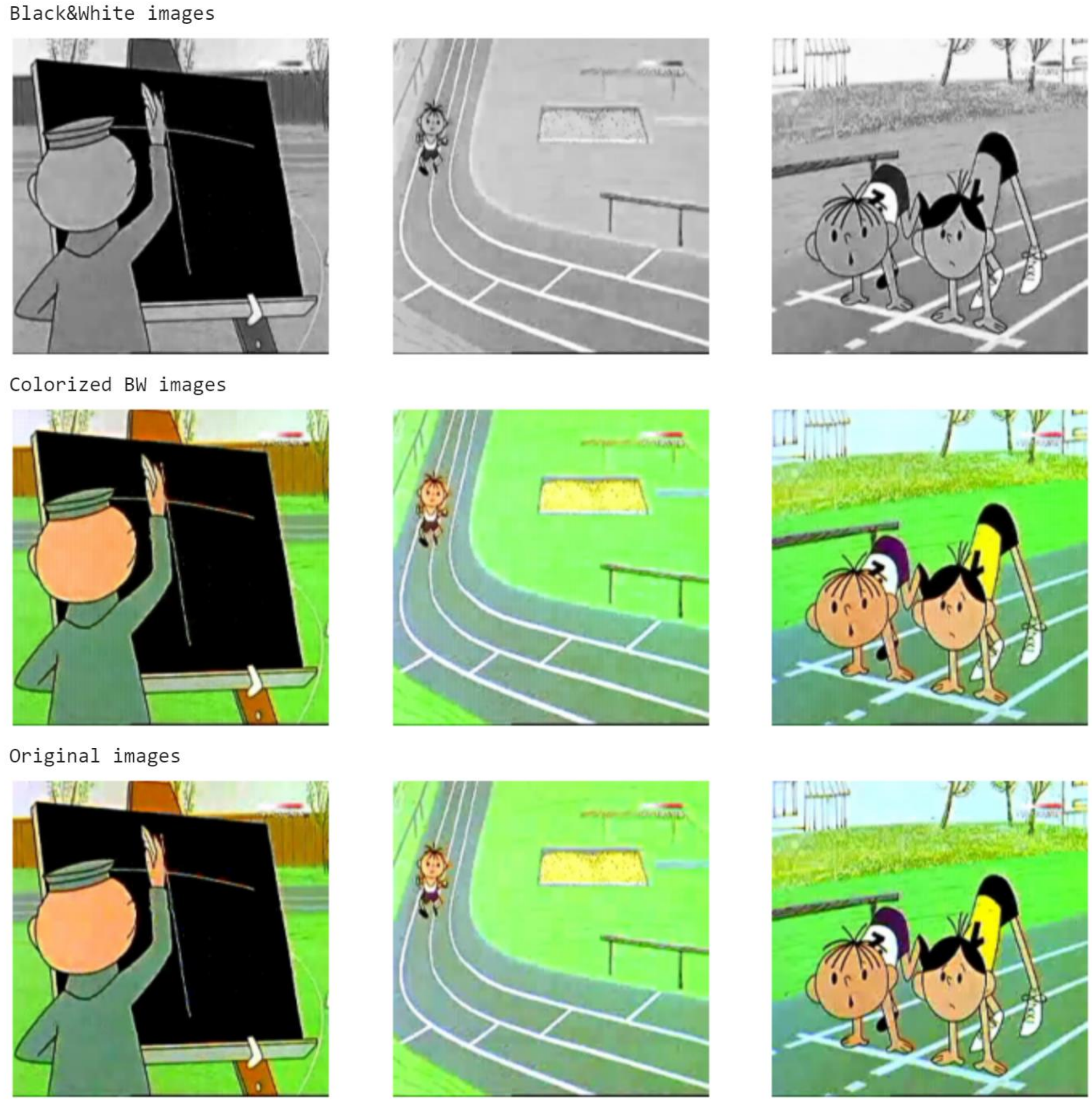
Using UNET architecture

- Necessary to add two channels to BW images:
  - bwImages = np.concatenate((bwImages,bwImages,bwImages),axis=3)
- Results: much better just after few epochs
- ***colorize_unet.ipynb***

pawel.kasprowski@polsl.pl

UNET Results

- Much better!



 pawel.kasprowski@polsl.pl

Super-resolution

- Turn an image with low resolution into the image with high resolution
- The state of the art established during:
  - New Trends in Image Restoration and Enhancement (NTIRE) workshop and challenge on image super-resolution
  - part of the CVPR conference
  - several editions: 2017-2021
- Different possible architectures

The simplest example

- ***supersampling_bolek.ipynb***
  - 64x64 -> 256x256
- The model:

```
conv_args = {"activation": "relu","padding": "same", }
inputs = Input(shape=image_shape)
x = Conv2D(64, 5, **conv_args)(inputs)
x = Conv2D(64, 3, **conv_args)(x)
x = Conv2D(32, 3, **conv_args)(x)
x = Conv2D(channels * (up_factor ** 2), 3, **conv_args)(x)
outputs = tf.nn.depth_to_space(x, up_factor)
model = Model(inputs, outputs)
```

pawel.kasprowski@polsl.pl

depth_to_space

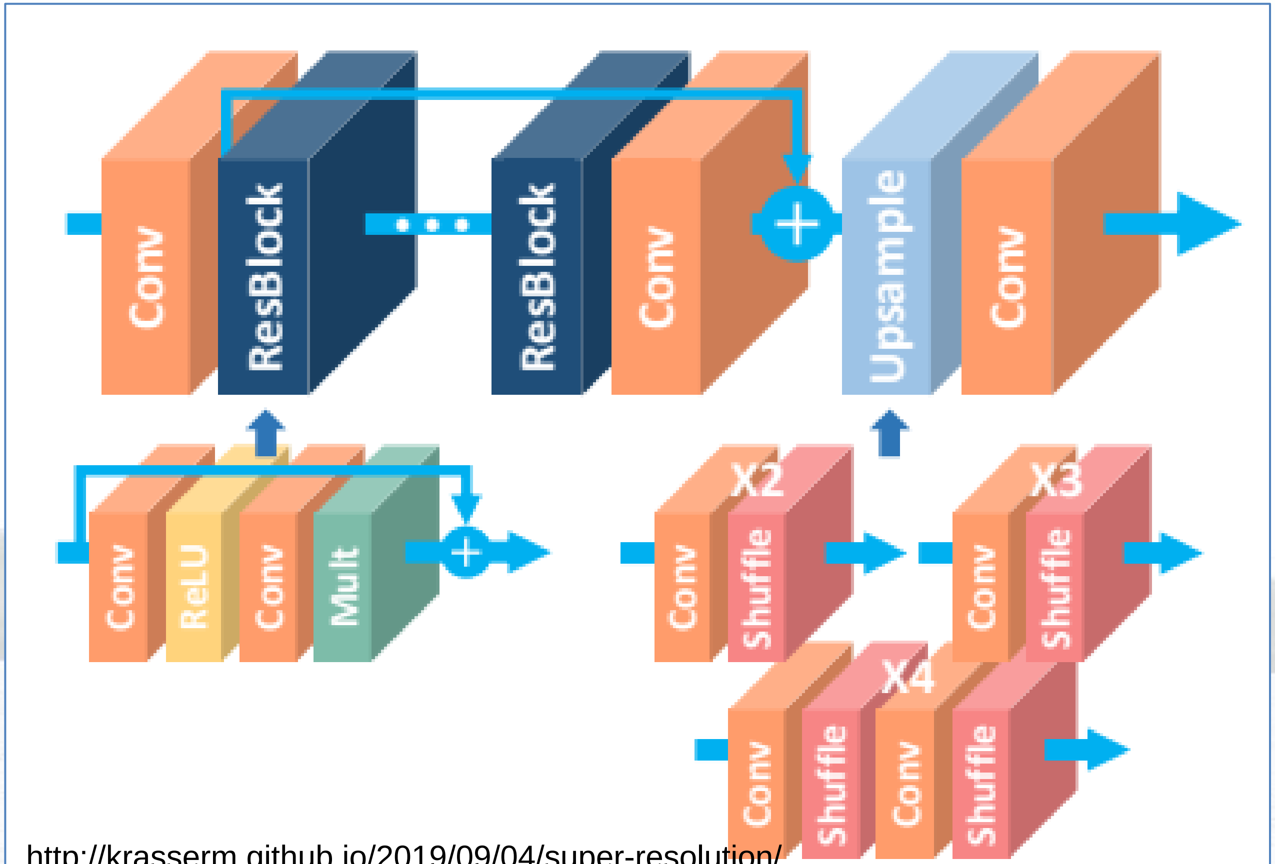
- Conversion with scale factor: s
- General task:
  - (W, H, C) > (s*W, s*H, C)
- Depth to space layer:
  - (W, H, C*s^2) > (s*W, s*H, C)
- Example:
  - (32, 32, 3)
  - ...
  - (32, 32, 3*4^2)
  - depth_to_space layer
  - (32*4, 32*4, 3)

More sophisticated architectures

- Enhanced Deep Residual Networks for Single Image Super-Resolution (EDSR)
  - winner of NTIRE 2017
- Wide Activation for Efficient and Accurate Image Super-Resolution (WDSR)
  - winner of NTIRE 2018
- Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network (SRGAN)
  - GAN network

EDSR

- Residual network with Conv2D-RELU-Conv2D-Mult blocks

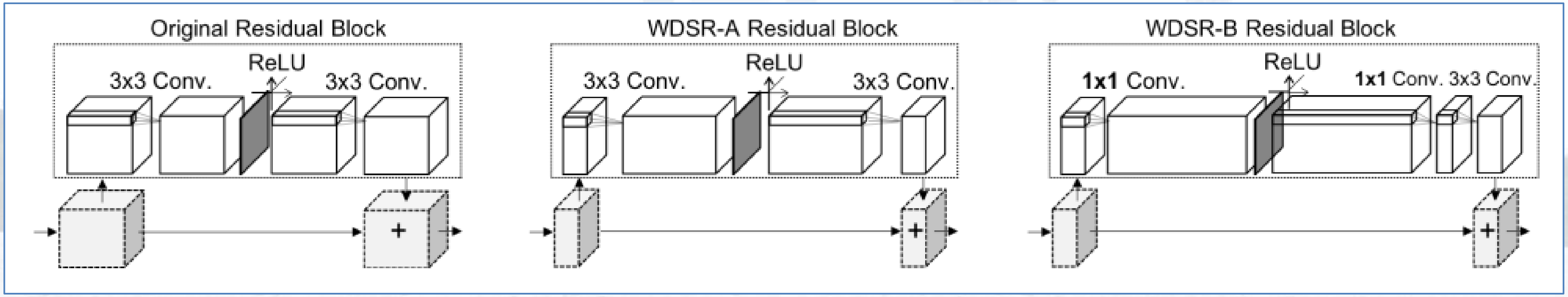


http://krasserm.github.io/2019/09/04/super-resolution/

pawel.kasprowski@polsl.pl

WDSR

- Extension of EDSR:
  - increases the number of channels in residual blocks
  - reduces the mumber of channels in mapping path
  - the same number of weights



http://krasserm.github.io/2019/09/04/super-resolution/

pawel.kasprowski@polsl.pl

Evaluation

- How to evaluate the correctness of superscaling?
- The obvious idea: calculate the difference between the generated image and the real image:
  - L2 norm
  - L1 norm
  - Binary crossentropy
- Problem: images percieved as blurred have typically good results
- A step forward:
  - use the additional network (discriminator!) to judge the correctness!
- SRGAN

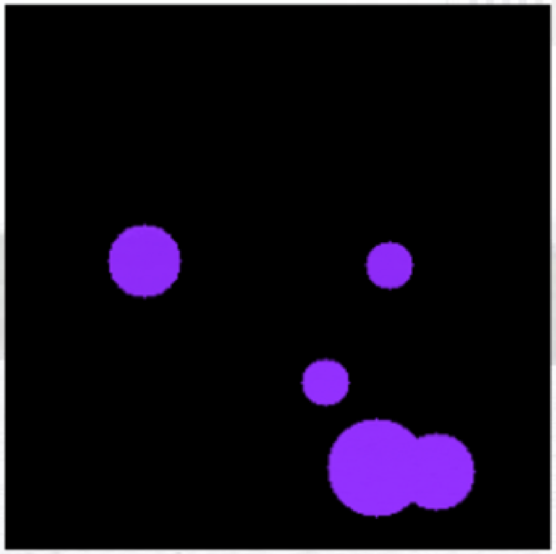
 pawel.kasprowski@polsl.pl

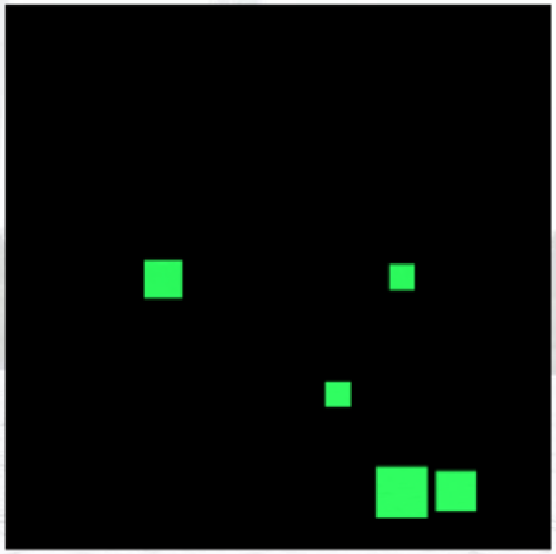
Ready to use

- More information:
  - http://krasserm.github.io/2019/09/04/super-resolution/
- Library with code:
  - https://github.com/krasserm/super-resolution
- Execution examples
  - article.ipynb
  - example-esdr.ipynb
  - example-wdsr.ipynb
  - example-srgan.ipynb

Image to image (pix2pix)

- GAN 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



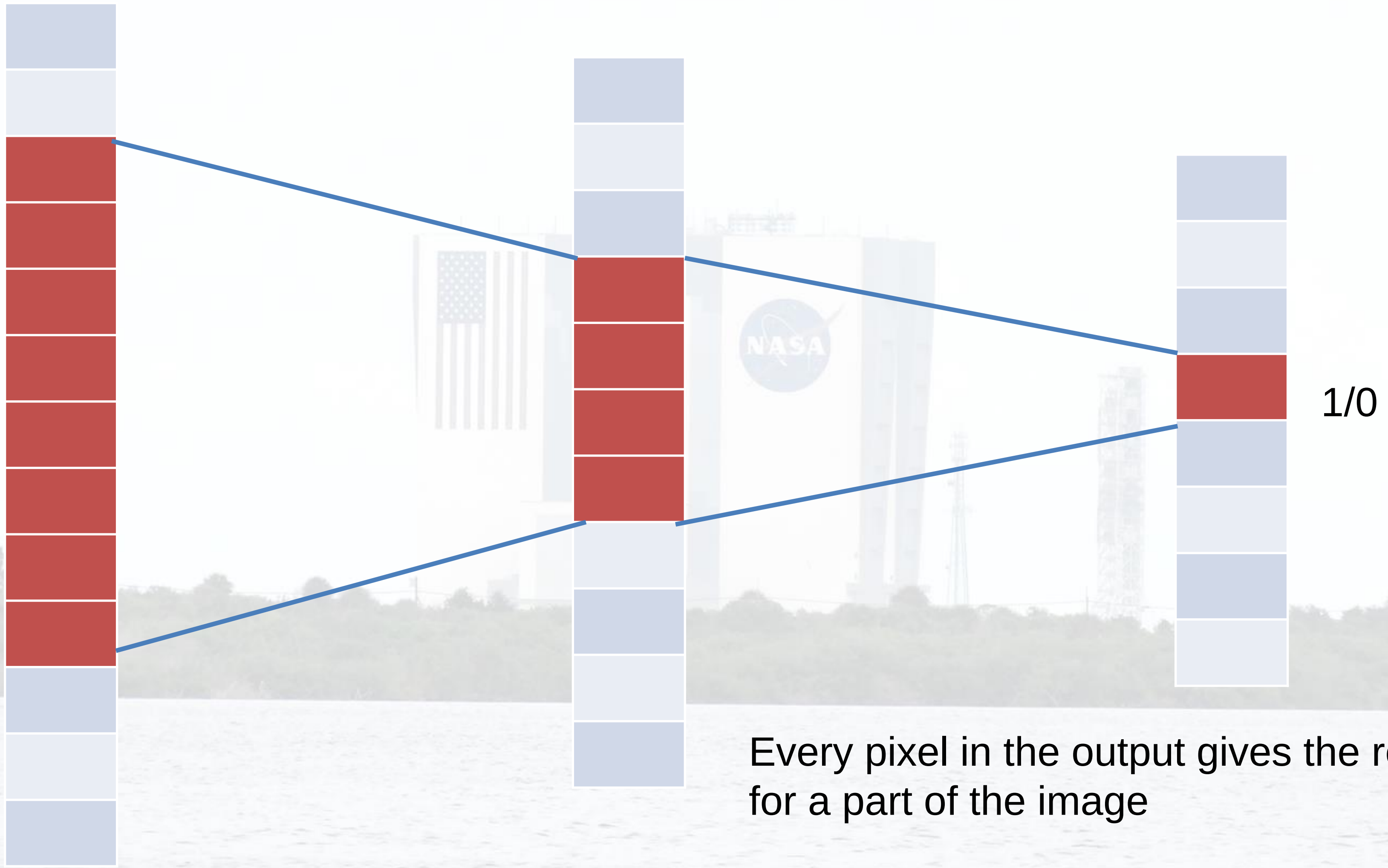


Working example

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!

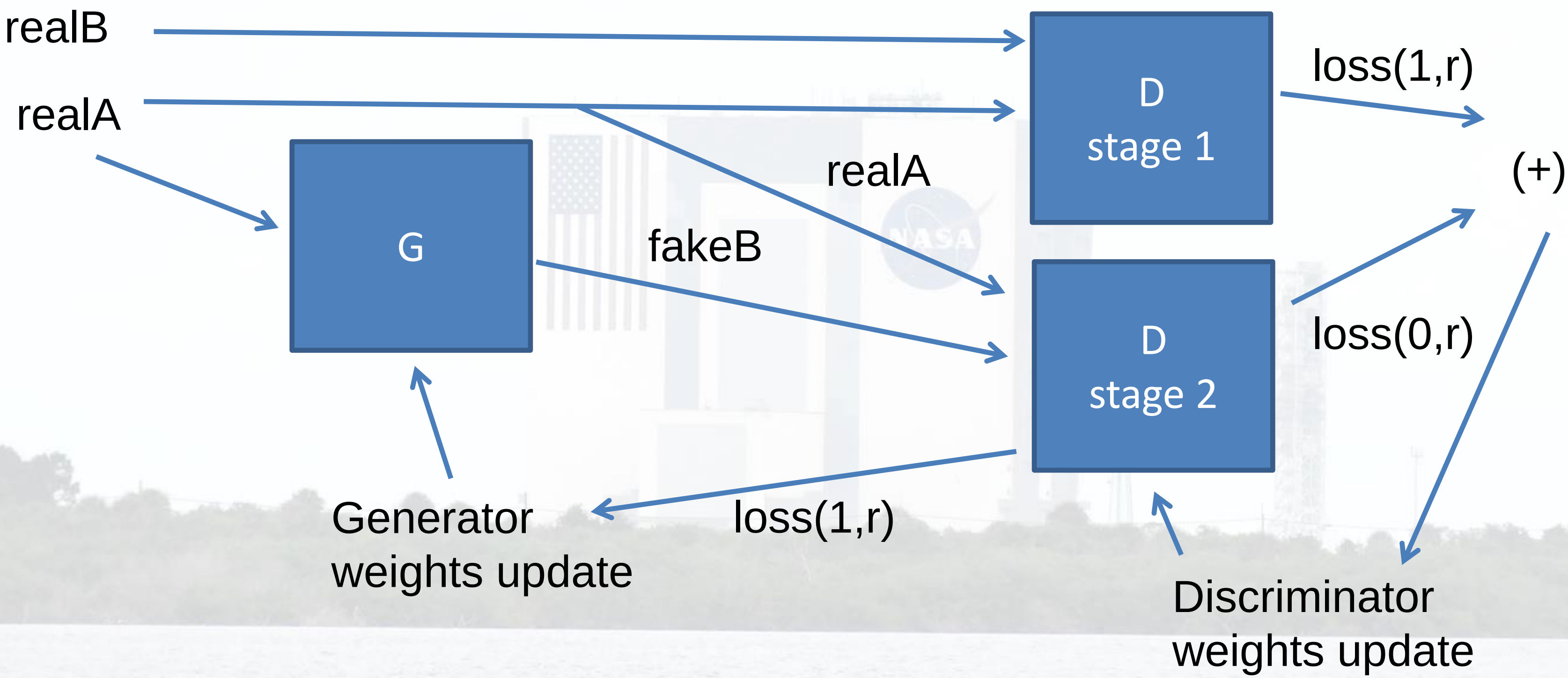
PatchGAN



Every pixel in the output gives the result for a part of the image

pawel.kasprowski@polsl.pl

Pix2Pix GAN architecture



A bit different GAN creation

- This time we don't use the GradientTape!
  - discriminator_model – will be trained by itself
  - generator_model – will be trained through the "gan_model"
- Creation of gan_model (the network used to train generator)

```
def gan_model (generator_model,discriminator_model):
	for layer in discriminator_model.layers: # discriminator will not be trained
		layer.trainable = False
	input_src = Input(shape=image_shape)
	gen_out = generator_model(input_src)
	disc_out = discriminator_model([input_src, gen_out])
	model = Model(input_src, [disc_out, gen_out])
	return model
```

pawel.kasprowski@polsl.pl

Training GAN

- Preparation:

```
patch = d_model.output_shape[1] # output of discriminator
steps = int(len(trainImgs) / batch) # steps per epoch
all_ones = np.ones((batch, patch, patch, 1)) # expected output for real
all_zeros = np.zeros((batch, patch, patch, 1)) # expected output for fake
```

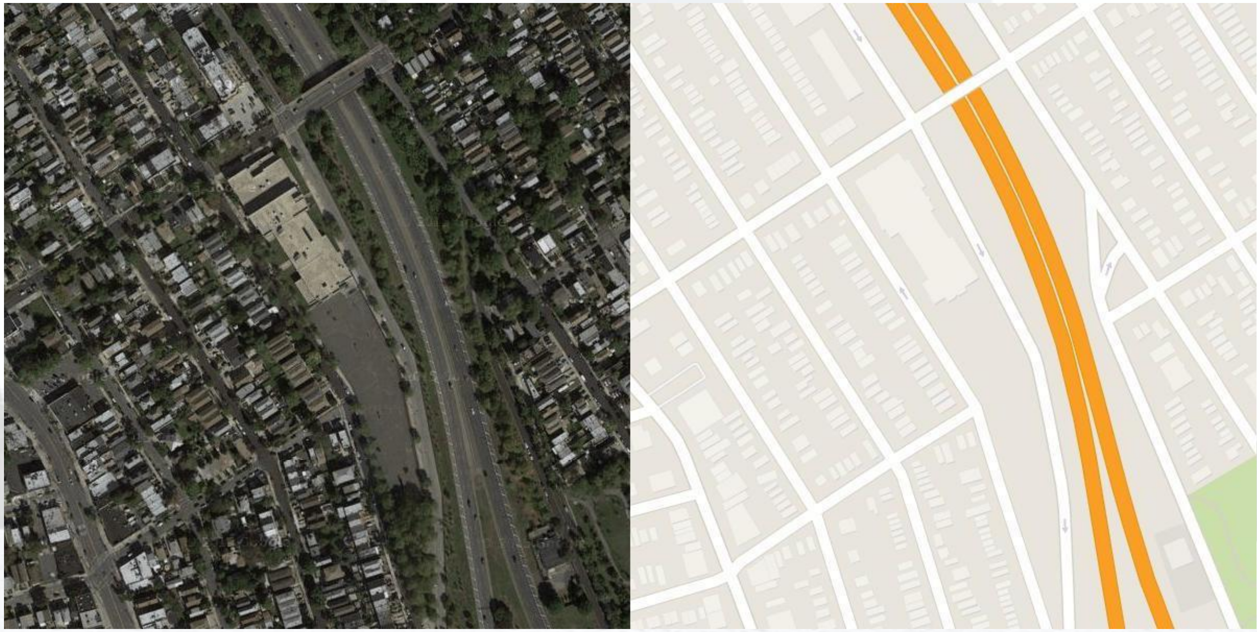
- One learning step:

```
for epoch in range(epochs):
    for i in range(steps):
        realA, realB = generate_real_samples(batch)
        fakeB = g_model.predict(realA)
        d_loss1 = d_model.train_on_batch([realA, realB], all_ones )
        d_loss2 = d_model.train_on_batch([realA, fakeB], all_zeros)
        g_loss, _, _ = gan_model.train_on_batch(realA, [all_ones, realB])
```

pawel.kasprowski@polsl.pl

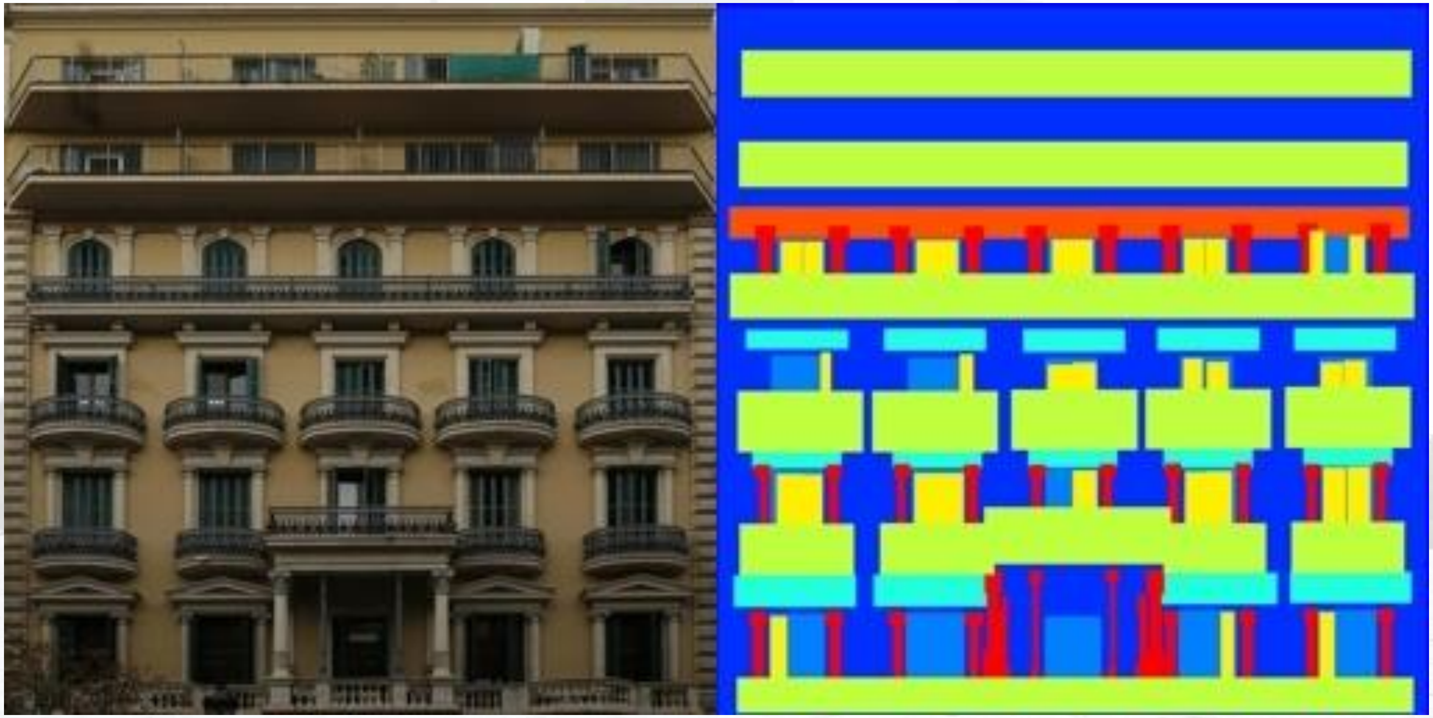
map2image example

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



Ready-to-use solution

- https://github.com/affinelayer/pix2pix-tensorflow
- Dataset preparation: set of images side by side:



pawel.kasprowski@polsl.pl

Using pix2pix

- Training on facades:
  - python pix2pix.py --mode train --output_dir facades_train --max_epochs 200 --input_dir facades/train --which_direction BtoA
- Testing facades:
  - python pix2pix.py --mode test --output_dir facades_test --input_dir facades/val --checkpoint facades_train
- Result: the html file with pairs of images

pawel.kasprowski@polsl.pl

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

pawel.kasprowski@polsl.pl

Using ready-to-use models

- Install tensorflow_examples package
  - pip install git+https://github.com/tensorflow/examples.git
- Import the package:

```
from tensorflow_examples.models.pix2pix import pix2pix
```

- Create the generator and discriminator:

```
generator = pix2pix.unet_generator(....)
discriminator = pix2pix.discriminator(...)
```

Problems with pix2pix

- It requires pairs of analogous images
- It is not always possible
- Simple example: change man face to woman face




- Problem: we don't have many pairs like that
- If we had a software producing such pairs we would not need any GAN!

CycleGAN

- Instead of preparing pairs of images we prepare sets of images
  - without one-to-one relationships!
- For example:
  - set of female images (X)
  - 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!)

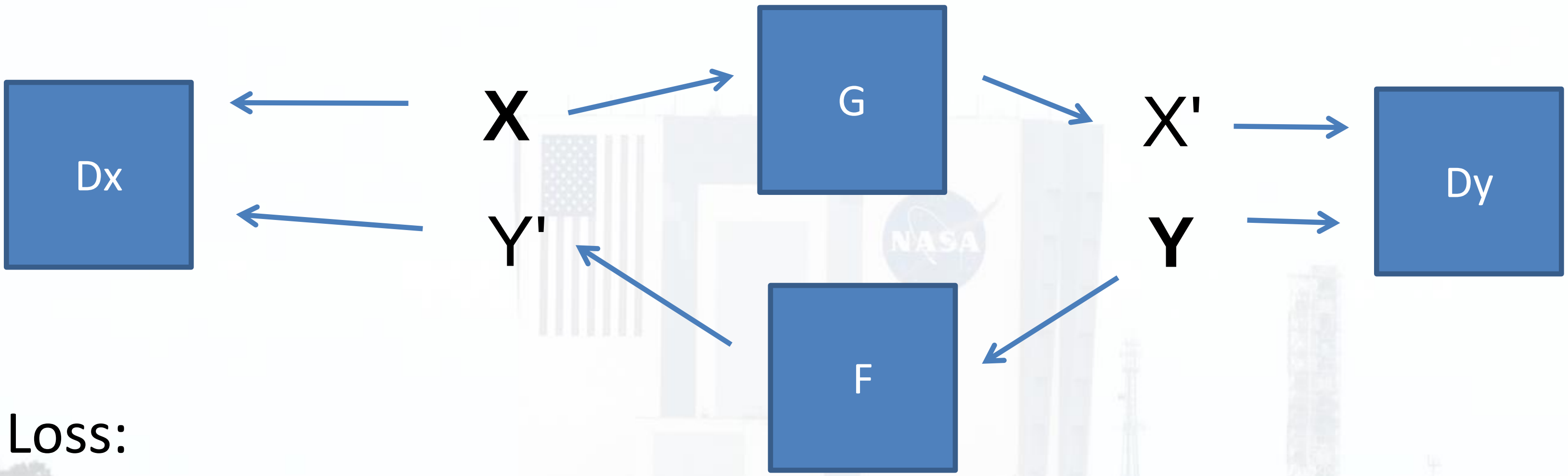
CycleGAN architecture

- 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

pawel.kasprowski@polsl.pl

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)+...

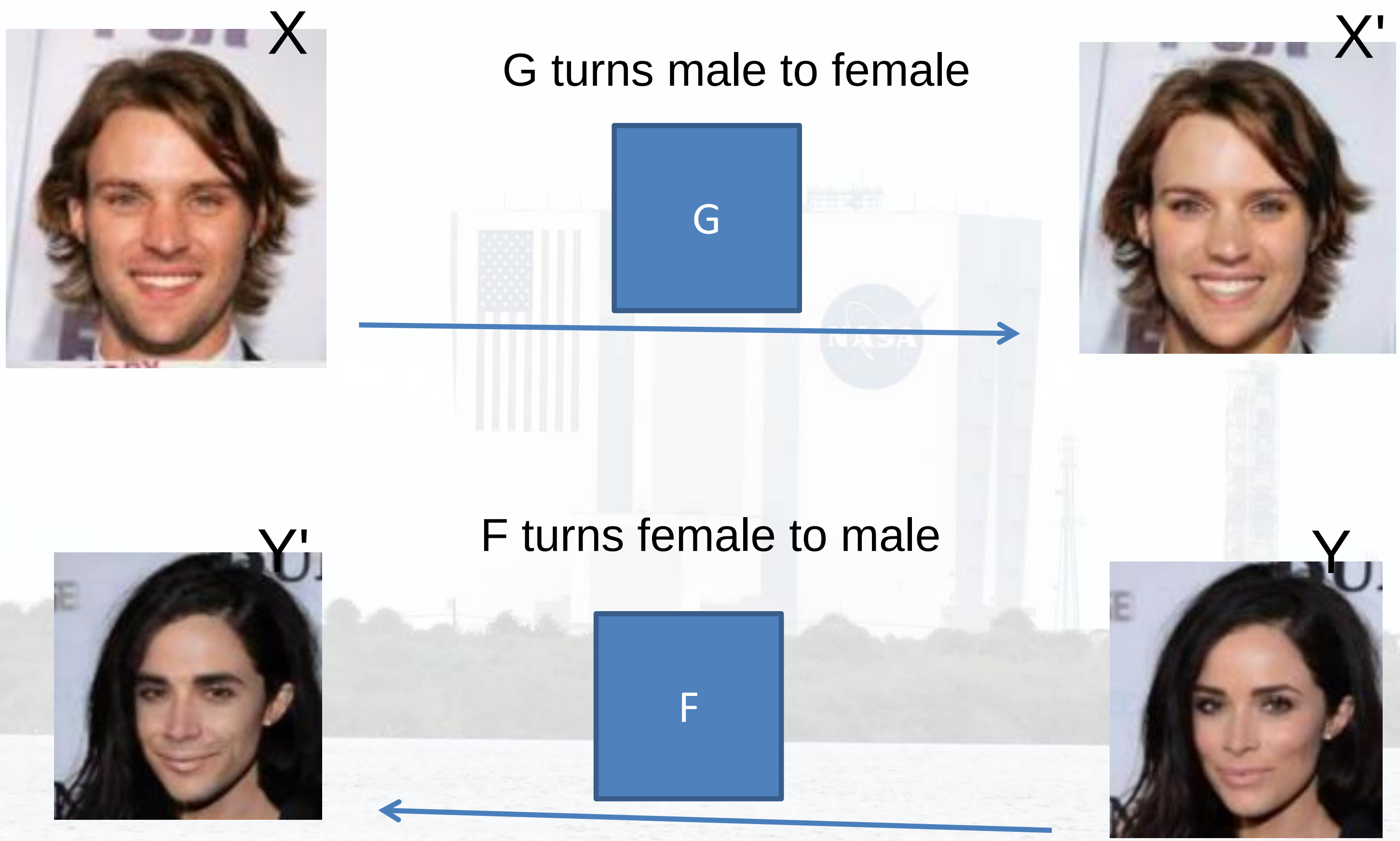
bce = binary crossentropy

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|

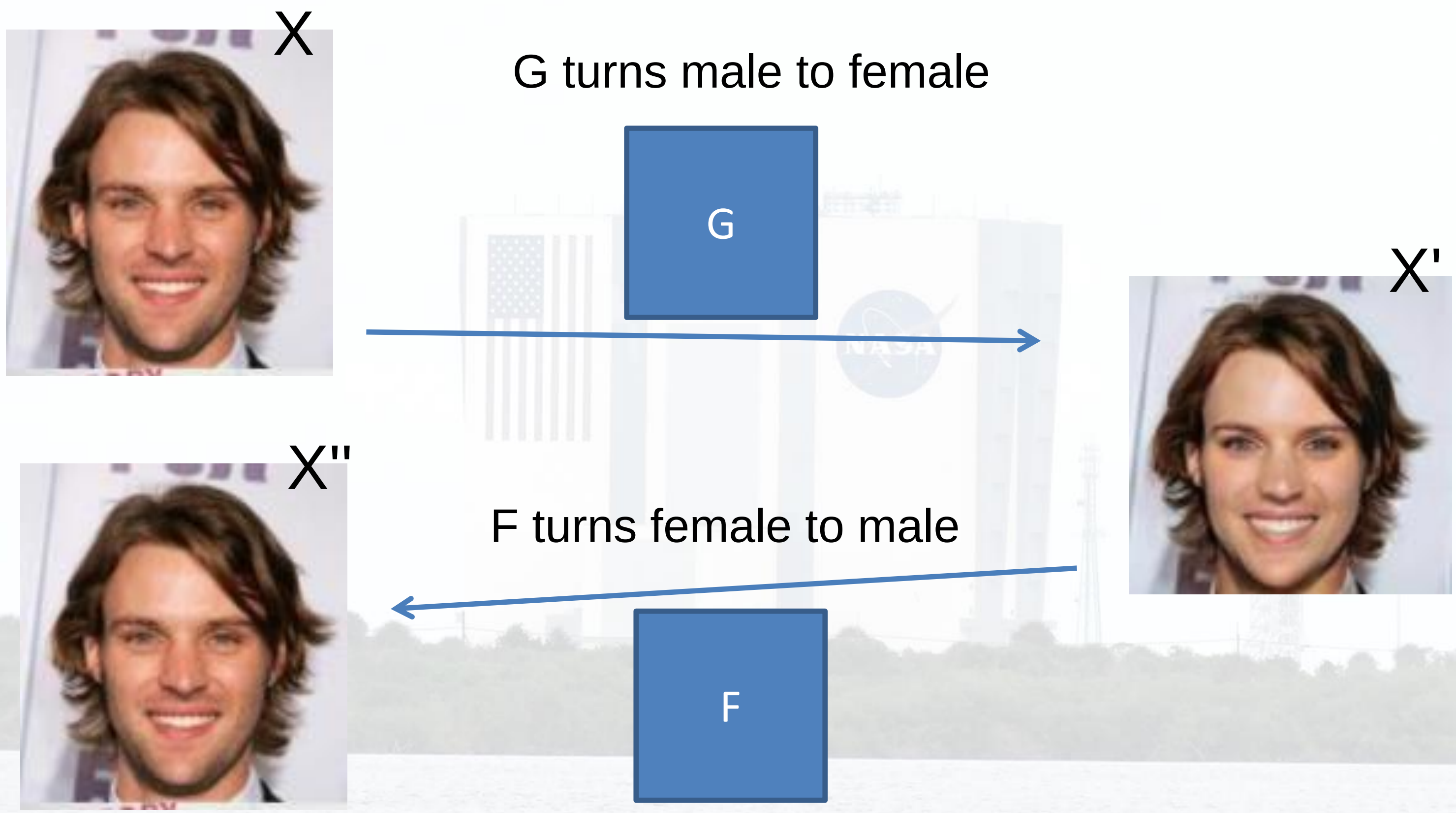
pawel.kasprowski@polsl.pl

For our example



pawel.kasprowski@polsl.pl

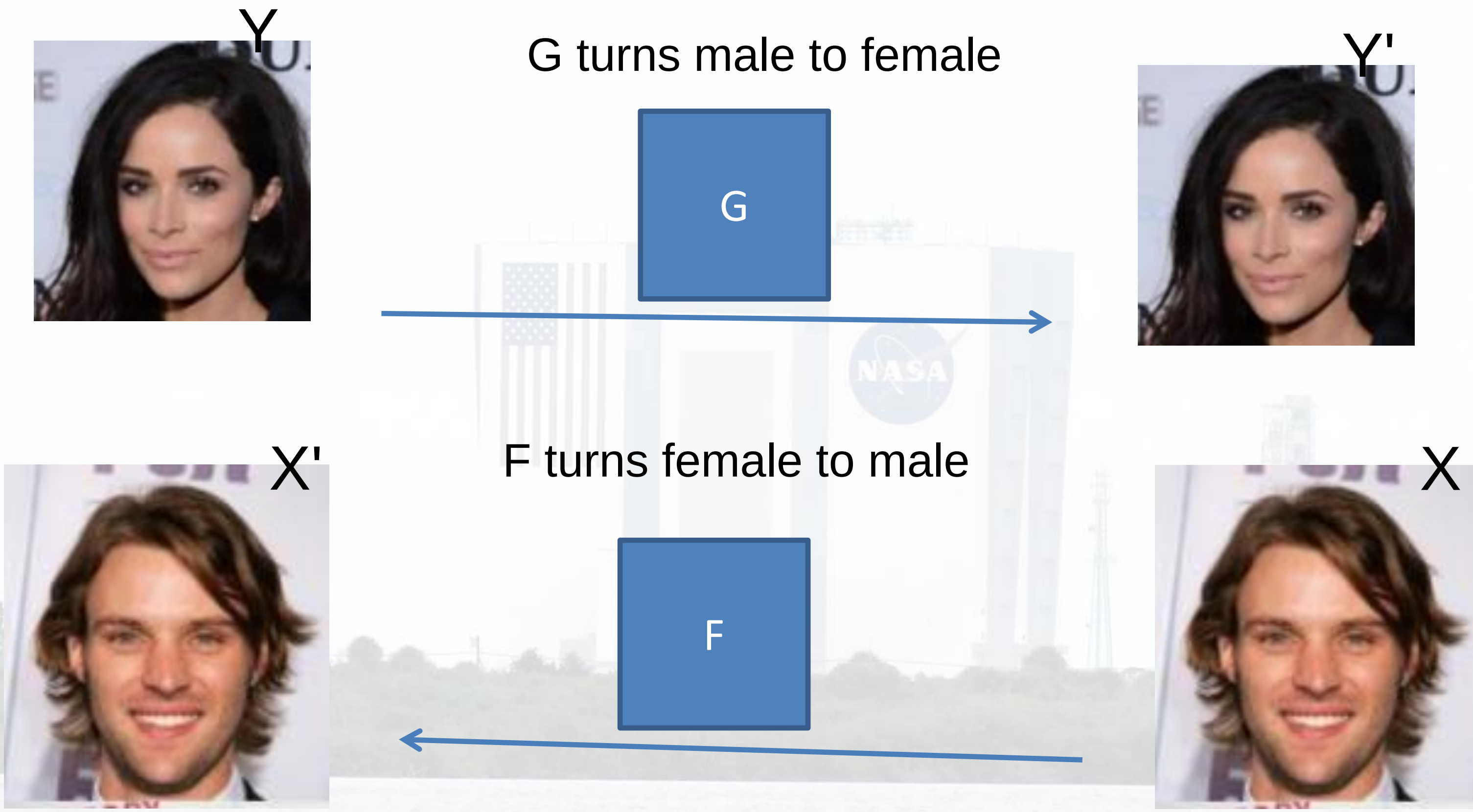
Cycle loss



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

pawel.kasprowski@polsl.pl

Identity loss



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

Rules for loss calculation

- Discriminator X should recognize male faces
- Discriminator Y should recognize female faces
- Cycle:
  - Male face after generator G should turn to female, and this changed image after generator F should turn to male again – the same as at the begining
  - Female face after generator F should turn to male, and this changed image after generator G should turn to female again
- Identity:
  - Male face used as input to generator F should remain male
  - Female face used as input to generator G should remain female

pawel.kasprowski@polsl.pl

One step

input: real_x, real_y

 # generate images

fake_y = generator_g(real_x, training=True)

cycled_x = generator_f(fake_y, training=True)

fake_x = generator_f(real_y, training=True)

cycled_y = generator_g(fake_x, training=True)

same_x = generator_f(real_x, training=True)

same_y = generator_g(real_y, training=True)

 # check results

disc_real_x = discriminator_x(real_x, training=True)

disc_real_y = discriminator_y(real_y, training=True)

disc_fake_x = discriminator_x(fake_x, training=True)

disc_fake_y = discriminator_y(fake_y, training=True)

Calculate loss

bce – binary cross entropy, abs – mean absolute error

 # discriminators losses

disc_x_loss = bce([1],disc_real_x) + bce([0],disc_fake_x)

disc_y_loss = bce([1],disc_real_y) + bce([0], disc_fake_y)

 # generators losses

gen_g_loss = bce([1],disc_fake_y)

gen_f_loss = bce([1],disc_fake_x)

total_cycle_loss = abs(real_x, cycled_x) + abs(real_y, cycled_y)

identity_loss_x = abs(real_x, same_x)

identity_loss_y = abs(real_y, same_y)

 # total generator losses

total_gen_g_loss = gen_g_loss + total_cycle_loss + identity_loss_y

total_gen_f_loss = gen_f_loss + total_cycle_loss + identity_loss_x

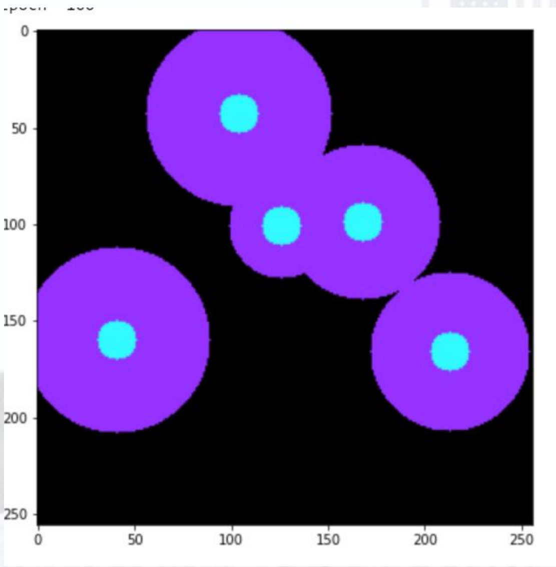
Apply gradients

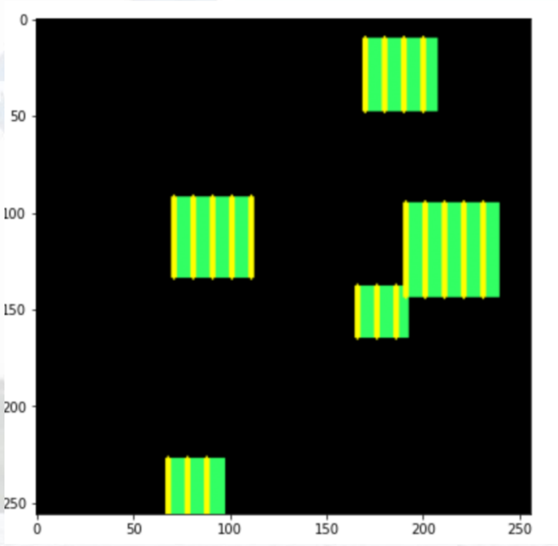
```
# Calculate the gradients for generators and discriminators
g_grads = tape.gradient(total_gen_g_loss, generator_g.trainable_variables)
f_grads = tape.gradient(total_gen_f_loss, generator_f.trainable_variables)
dx_grads = tape.gradient(disc_x_loss, discriminator_x.trainable_variables)
dy_grads = tape.gradient(disc_y_loss, discriminator_y.trainable_variables)

# Apply the gradients to the networks
g_optimizer.apply_gradients(zip(g_grads, generator_g.trainable_variables))
f_optimizer.apply_gradients(zip(f_grads, generator_f.trainable_variables))
dx_opt.apply_gradients(zip(dx_grads, discriminator_x.trainable_variables))
dy_opt.apply_gradients(zip(dy_grads, discriminator_y.trainable_variables))
```

 pawel.kasprowski@polsl.pl

CycleGAN example

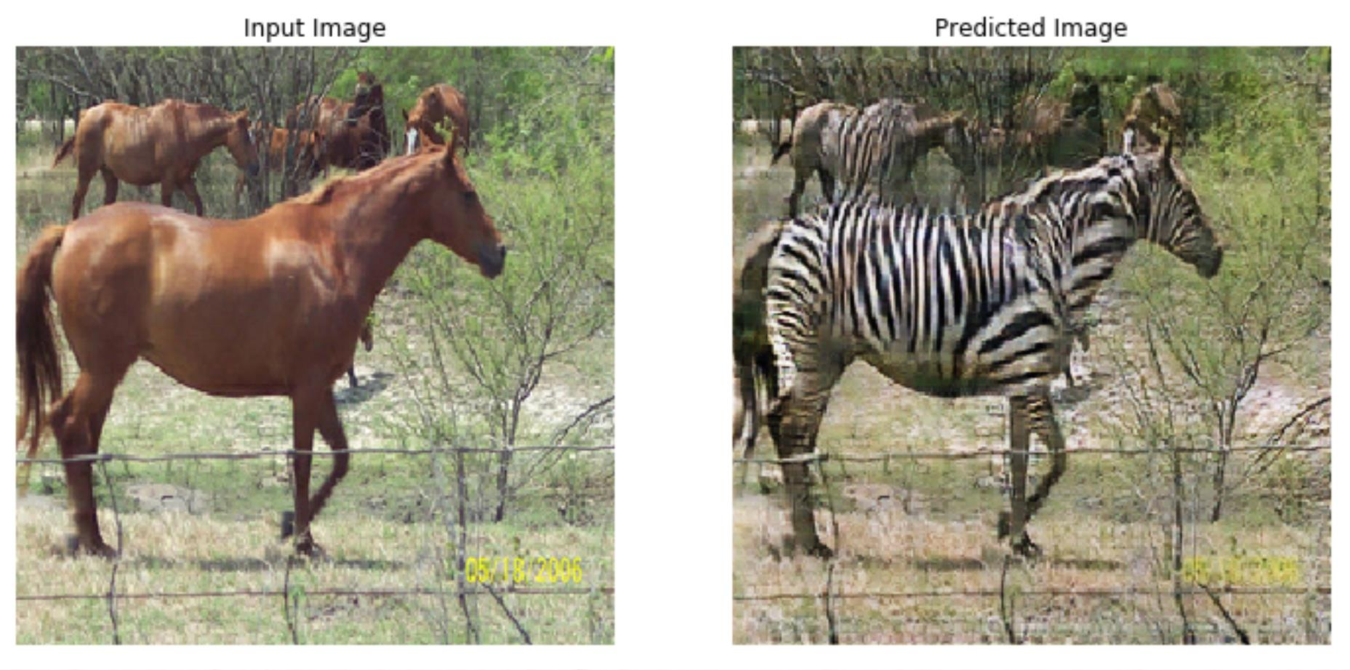
- ***cyclegan.ipynb***
- Changing circles to squares





A classic example

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



pawel.kasprowski@polsl.pl

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
- ...

Summary

- Autoencoders and U-Networks
  - may be used for image conversion (deniosing, colorization, supersampling,...)
- Pix2pix
  - converts one image to another
  - we need pairs of images
- CycleGAN
  - builds generators that convert one type of images into another type
- There are a lot of interesting applications!

pawel.kasprowski@polsl.pl

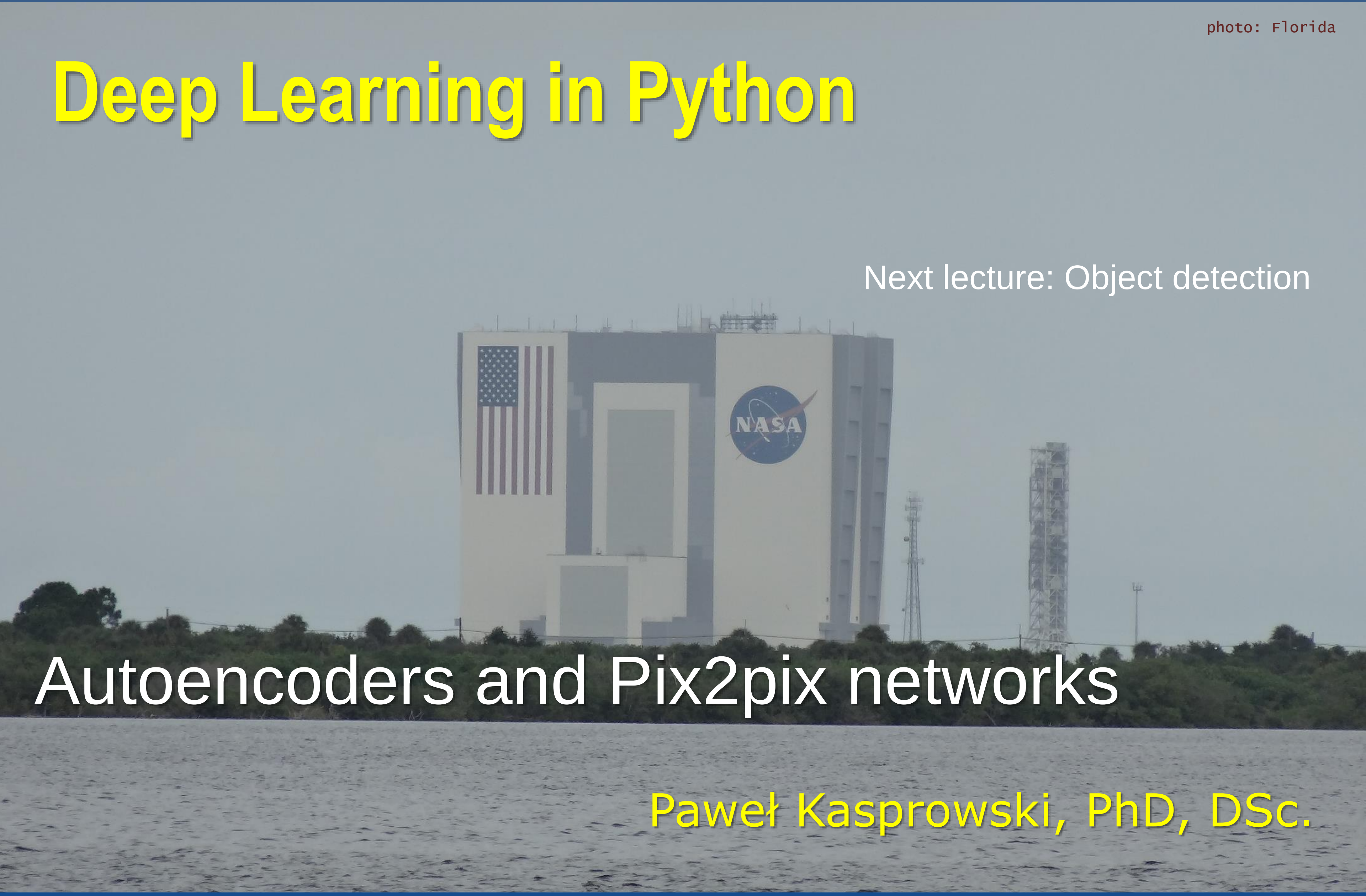


photo: Florida

Deep Learning in Python

Next lecture: Object detection

Autoencoders and Pix2pix networks

Paweł Kasprowski, PhD, DSc.