CEFET/RJ

CBPF

Deep Learning Applications in Astronomy

Clécio R. Bom

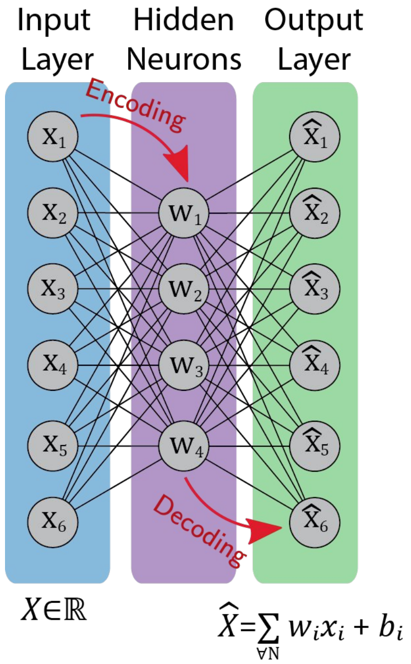
clearnightsrthebest.com

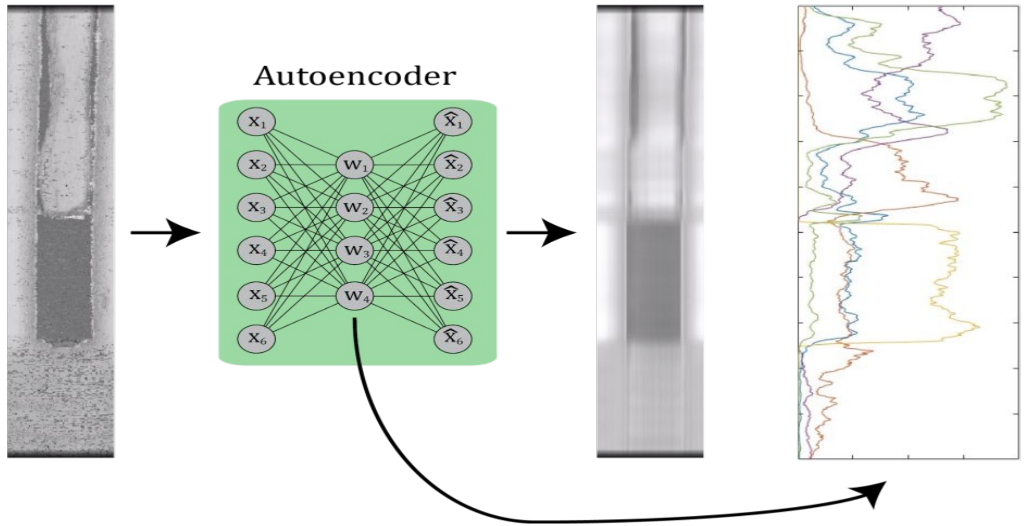
debom@cbpf.br

February 13, 2019

What is an AutoEncoder?

- ANN aim to reproduce input data.
- **Encode/Decode** strategy.
- **IDENTITY FUNCTION → Hidden Features**


Input Layer
Hidden Neurons
Output Layer
Encoding
Decoding
x_1 x_2 x_3 x_4 x_5 x_6
w_1 w_2 w_3 w_4
$\hat{x}_1$ $\hat{x}_2$ $\hat{x}_3$ $\hat{x}_4$ $\hat{x}_5$ $\hat{x}_6$
$X \in \mathbb{R}$
$\hat{X} = \sum_{\forall N} w_i x_i + b_i$



Auto Encoders Example

```
from keras.layers import Input, Dense
from keras.models import Model

# this is the size of our encoded representations
encoding_dim = 32
 # 32 floats -> compression of factor 24.5, assuming the input is 784
floats

# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)

# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
```

Example from: https://blog.keras.io/building-autoencoders-in-keras.html

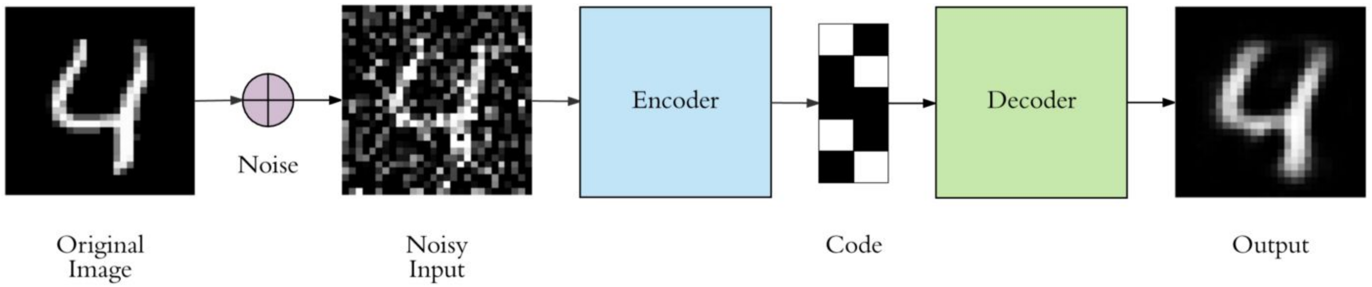
Auto Encoders – A simple example

```
# this model maps an input to its encoded representation

encoder = Model(input_img, encoded)

# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# create the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))
```

Warning: The autoencoders do overfit a lot if they are too complex.

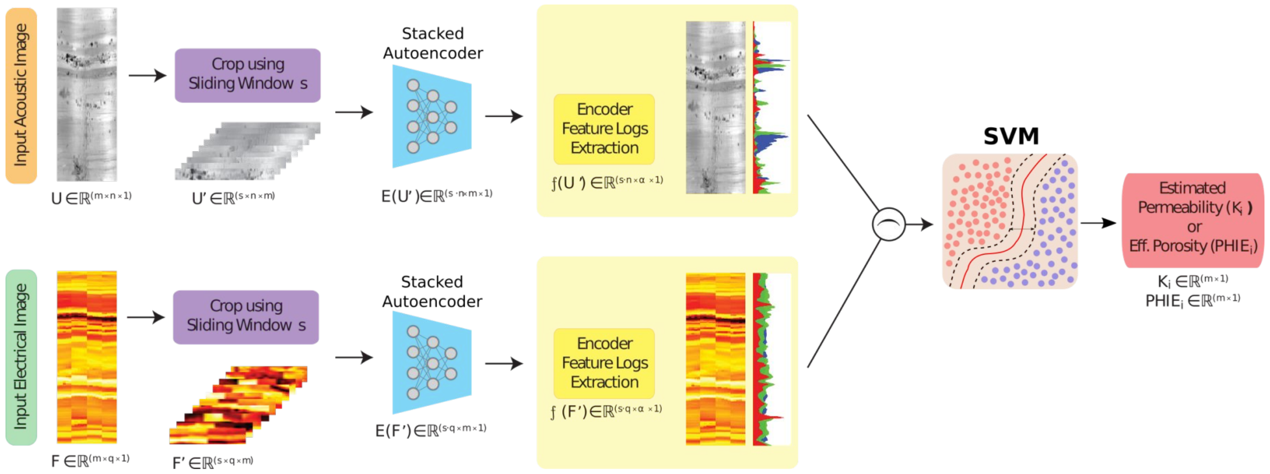
Small number of parameters forces the autoencoder to learn an intelligent representation of the data.

Denoising


Noise
Original Image
Noisy Input
Encoder
Code
Decoder
Output

Regression with AE? But Why?

- Small datasets
- "Empty" datasets



Input Acoustic Image
Crop using Sliding Window s
Stacked Autoencoder
Encoder Feature Logs Extraction
$U \in \mathbb{R}^{(m \times n \times 1)}$
$U' \in \mathbb{R}^{(s \times n \times m)}$
$E(U') \in \mathbb{R}^{(s \cdot n \times m \times 1)}$
$f(U') \in \mathbb{R}^{(s \cdot n \times \alpha \times 1)}$
Input Electrical Image
Crop using Sliding Window s
Stacked Autoencoder
Encoder Feature Logs Extraction
$F \in \mathbb{R}^{(m \times q \times 1)}$
$F' \in \mathbb{R}^{(s \times q \times m)}$
$E(F') \in \mathbb{R}^{(s \cdot q \times m \times 1)}$
$f(F') \in \mathbb{R}^{(s \cdot q \times \alpha \times 1)}$
SVM
Estimated Permeability (K_i) or Eff. Porosity ($PHIE_i$)
$K_i \in \mathbb{R}^{(m \times 1)}$
$PHIE_i \in \mathbb{R}^{(m \times 1)}$

Variational AE

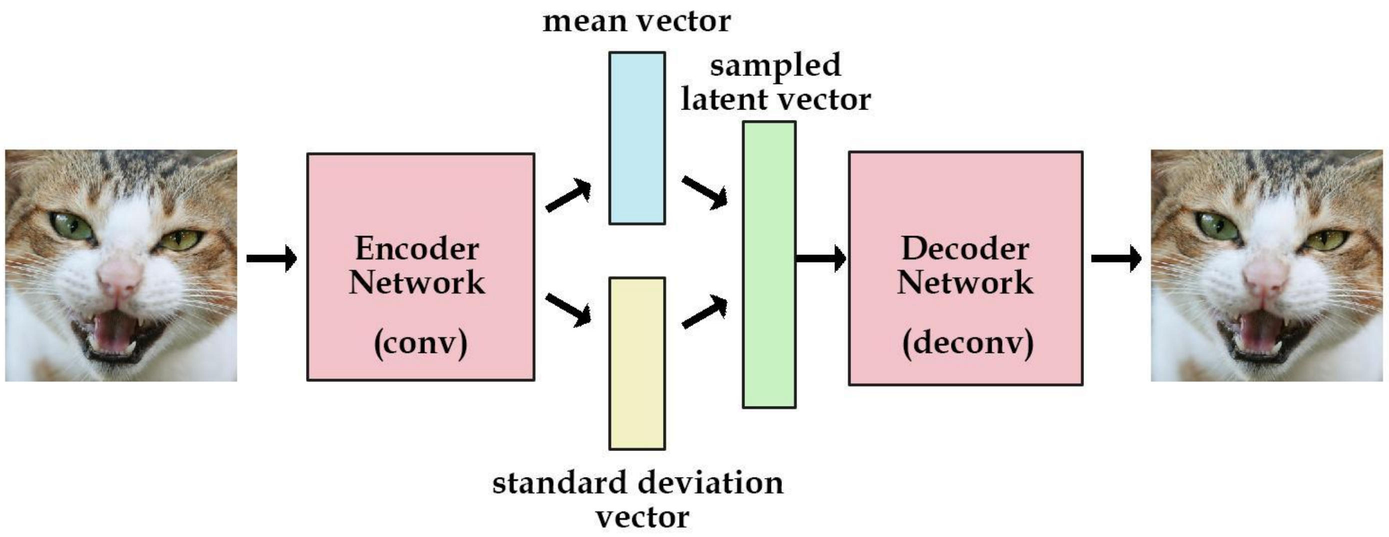


Image Credit http://kvfrans.com/variational-autoencoders-explained/

Variational AE – Generative Model

An autoencoder that learns a latent variable model for its input data. So instead of letting your neural network learn an arbitrary function, you are learning the parameters of a probability distribution modeling your data. If you sample points from this distribution, you can generate new input data samples: a VAE is a "generative model".

$$q(Z|X) \qquad\qquad p(X|Z)$$

Encode

Decode

```
x = Input(batch_shape=(batch_size, original_dim))
h = Dense(intermediate_dim, activation='relu')(x)
z_mean = Dense(latent_dim)(h)
z_log_sigma = Dense(latent_dim)(h)
```

Variational AE – Generative Model

Latent normal distribution that is assumed to generate the data, via z = z_mean + exp(z_log_sigma) * epsilon, where epsilon is a random normal tensor.

A decoder network maps these latent space points back to the original input data.

```
def sampling(args):
    z_mean, z_log_sigma = args
    epsilon = K.random_normal(shape=(batch_size, 
latent_dim),
                              mean=0., 
std=epsilon_std)
    return z_mean + K.exp(z_log_sigma) * epsilon

z = Lambda(sampling, 
output_shape=(latent_dim,))([z_mean, z_log_sigma])
```

$$q_\phi(\mathbf{z}_n|\mathbf{x}_n) = \mathcal{N}(\mathbf{z}_n|\mu_\phi(\mathbf{x}_n), \mathrm{diag}(\sigma^2_\phi(\mathbf{x}_n))),$$

$$\mathbf{z} = g_\phi(\mathbf{x}, \epsilon) = \mu_\phi(\mathbf{x}) + \sigma_\phi(\mathbf{x}) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

Variational AE – Generative Model

Latent normal distribution that is assumed to generate the data, via z = z_mean + exp(z_log_sigma) * epsilon, where epsilon is a random normal tensor.

A decoder network maps these latent space points back to the original input data.

```
decoder_h = Dense(intermediate_dim, activation='relu')
decoder_mean = Dense(original_dim, activation='sigmoid')
h_decoded = decoder_h(z)
x_decoded_mean = decoder_mean(h_decoded)
```

https://blog.keras.io/building-autoencoders-in-keras.h
tml

Variational AE – Generative Model

Latent normal distribution that is assumed to generate the data, via z = z_mean + exp(z_log_sigma) * epsilon, where epsilon is a random normal tensor.

A decoder network maps these latent space points back to the original input data.

```
# end-to-end autoencoder
vae = Model(x, x_decoded_mean)

# encoder, from inputs to latent space
encoder = Model(x, z_mean)

# generator, from latent space to reconstructed inputs
decoder_input = Input(shape=(latent_dim,))
_h_decoded = decoder_h(decoder_input)
_x_decoded_mean = decoder_mean(_h_decoded)
generator = Model(decoder_input, _x_decoded_mean)
```

https://blog.keras.io/building-autoencoders-in-keras.h
tml

Variational AE – Generative Model

We add an extra term to the Loss

$$D_{\mathrm{KL}}(Q\|P) = \sum_i Q(i) \ln\left(\frac{Q(i)}{P(i)}\right)$$

Consider an observed variable that is connected to the latent variable z.
As we are dealing with gaussians we may write

$$\mathrm{KL}[q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})] = -\frac{1}{2}\sum_{k=1}^{K}\{1 + \log\sigma_k^2 - \mu_k^2 - \sigma_k^2\}$$

D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in Proceedings of the 2nd International Conference on Learning Representations (ICLR), 2014.

http://louistiao.me/posts/implementing-variational-autoencoders-in-keras-beyond-the-quickstart-tutorial/#kingma2014

Variational AE – Generative Model

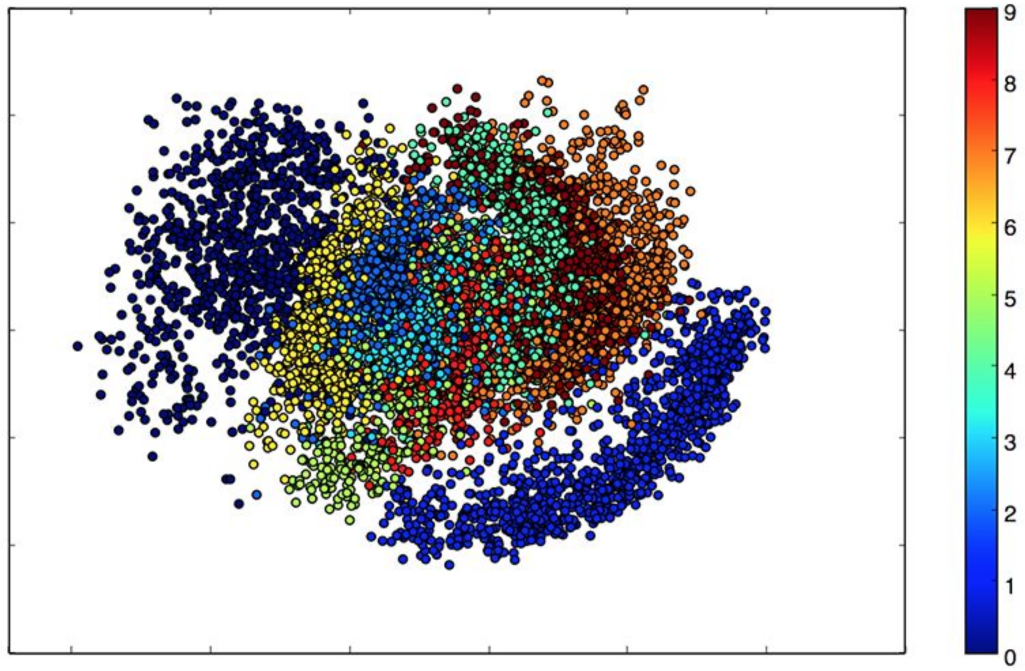
```
def vae_loss(x, x_decoded_mean):
    xent_loss = objectives.binary_crossentropy(x, 
x_decoded_mean)
    kl_loss = - 0.5 * K.mean(1 + z_log_sigma - 
K.square(z_mean) - K.exp(z_log_sigma), axis=-1)
    return xent_loss + kl_loss
```

Example adapted from: https://blog.keras.io/building-autoencoders-in-keras.html

Variational AE – Generative Model

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train),
np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

vae.fit(x_train, x_train,
        shuffle=True,
        epochs=epochs,
        batch_size=batch_size,
        validation_data=(x_test, x_test))
```

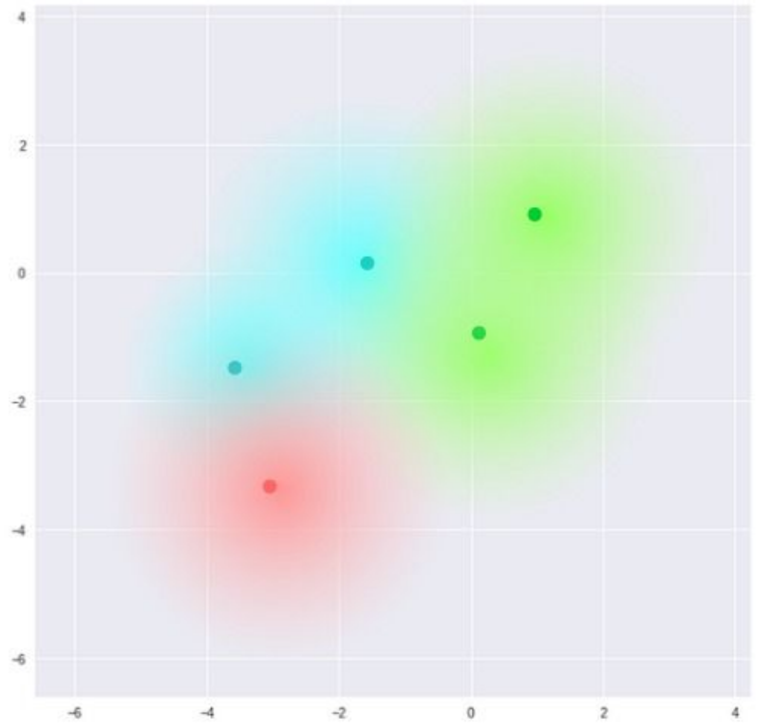
```
x_test_encoded = encoder.predict(x_test, batch_size=batch_size)
plt.figure(figsize=(6, 6))
plt.scatter(x_test_encoded[:, 0], x_test_encoded[:, 1], c=y_test)
plt.colorbar()
plt.show()
```



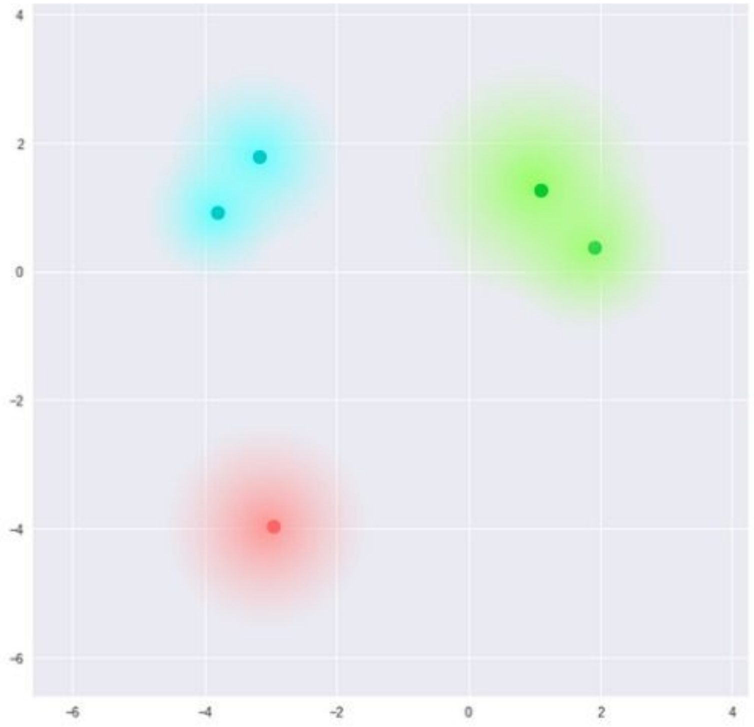
Variational AE – Generative Model

```
n = 15  # figure with 15x15 digits
digit_size = 28
figure = np.zeros((digit_size * n, digit_size * n))
# we will sample n points within [-15, 15] standard deviations
grid_x = np.linspace(-15, 15, n)
grid_y = np.linspace(-15, 15, n)

for i, yi in enumerate(grid_x):
    for j, xi in enumerate(grid_y):
        z_sample = np.array([[xi, yi]]) * epsilon_std
        x_decoded = generator.predict(z_sample)
        digit = x_decoded[0].reshape(digit_size, digit_size)
        figure[i * digit_size: (i + 1) * digit_size,
               j * digit_size: (j + 1) * digit_size] = digit
plt.show()
```

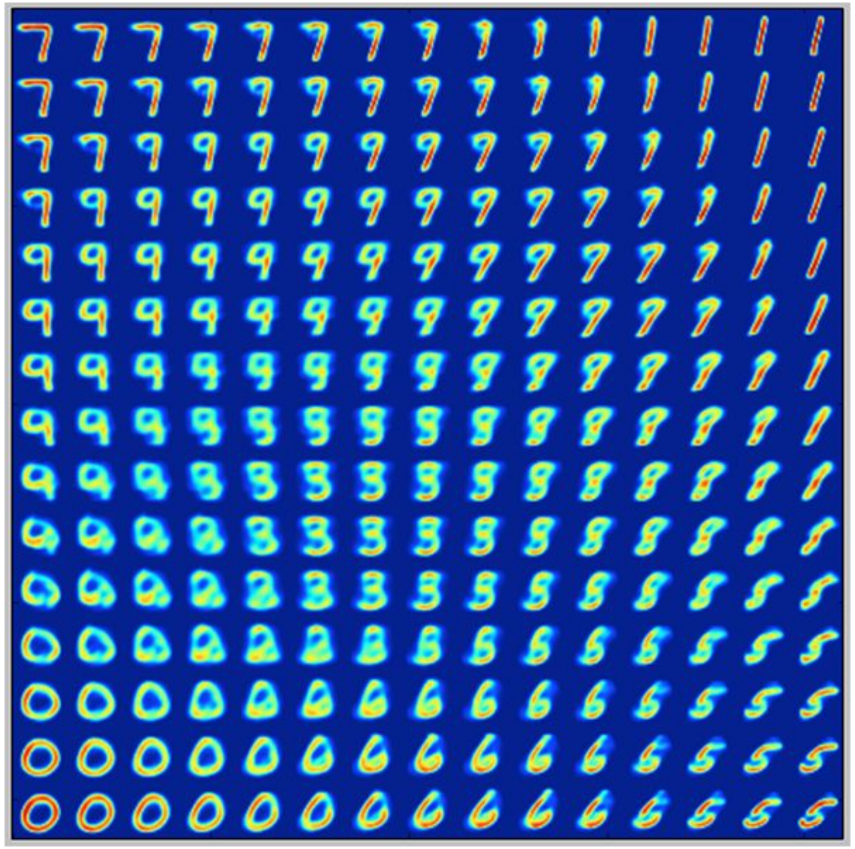
Variational AE – Generative Model



What we require



What we may inadvertently end up with



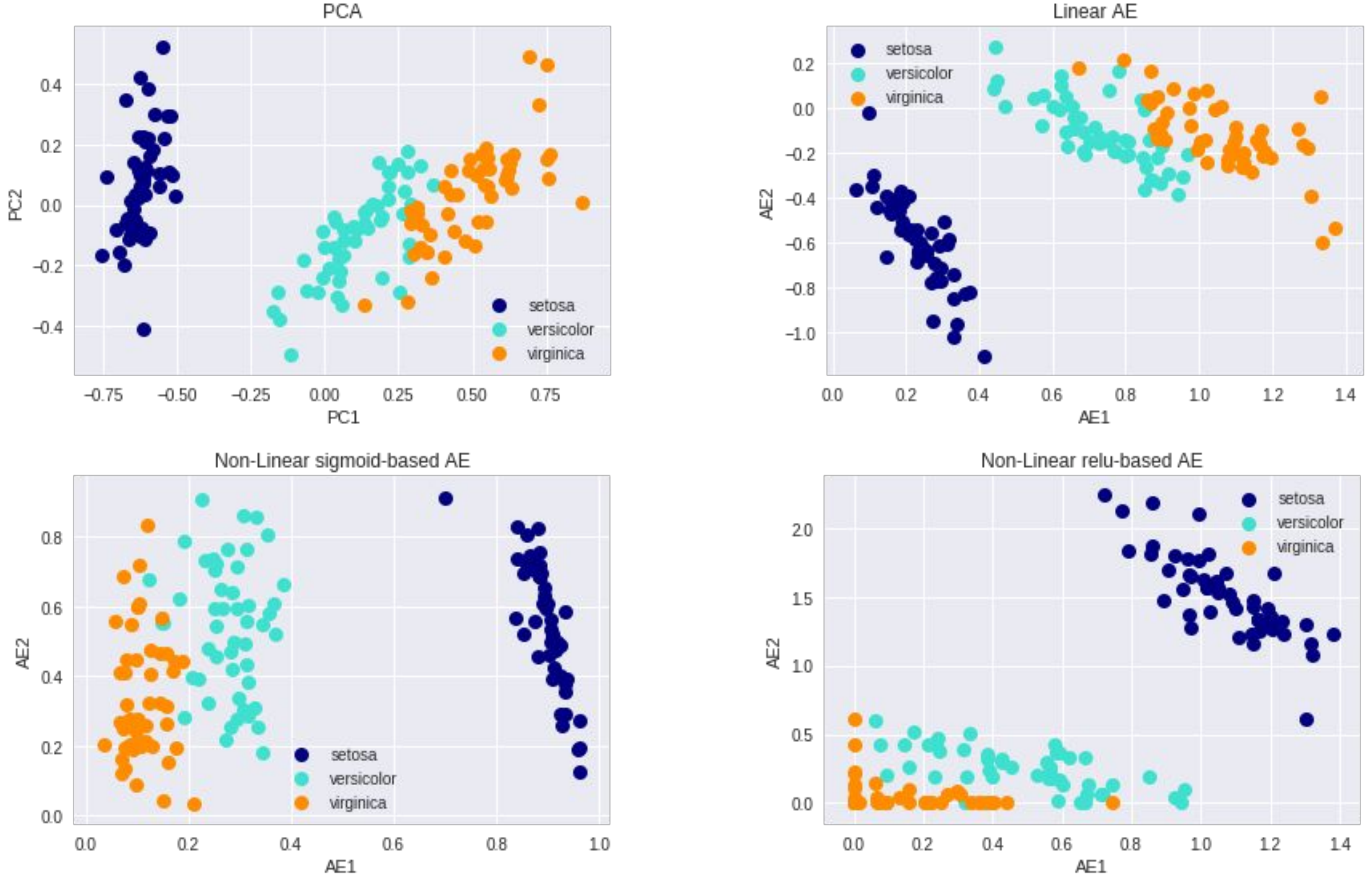
PCA vs AE

Iris Dataset

Attribute Information:
1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

```
iris = datasets.load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names


pca = decomposition.PCA()
pca_transformed = pca.fit_transform(X_scaled)
plot3clusters(pca_transformed[:,:2], 'PCA', 'PC')
```

Example adapted from: https://towardsdatascience.com/pca-vs-autoencoders-1ba08362f450

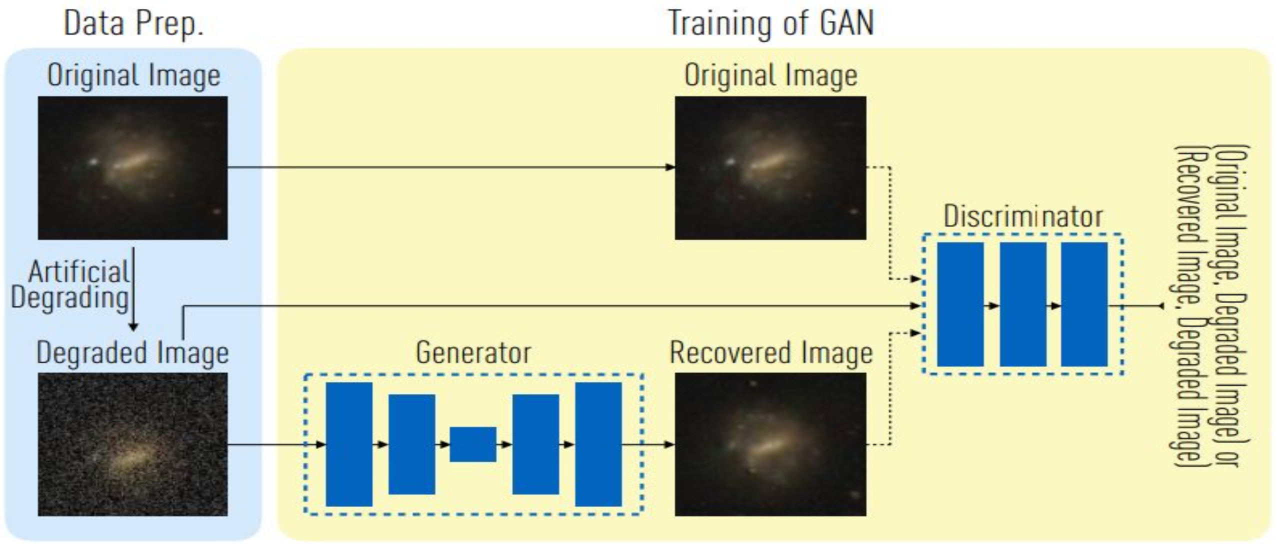

PCA
PC1
PC2
Linear AE
Non-Linear sigmoid-based AE
Non-Linear relu-based AE
AE1
AE2
setosa
versicolor
virginica

Generative Adversarial Neural Networks

First proposed by Goodfellow et al. 2014 arXiv:1406.2661

The Main idea is to make two networks to be adversaries.

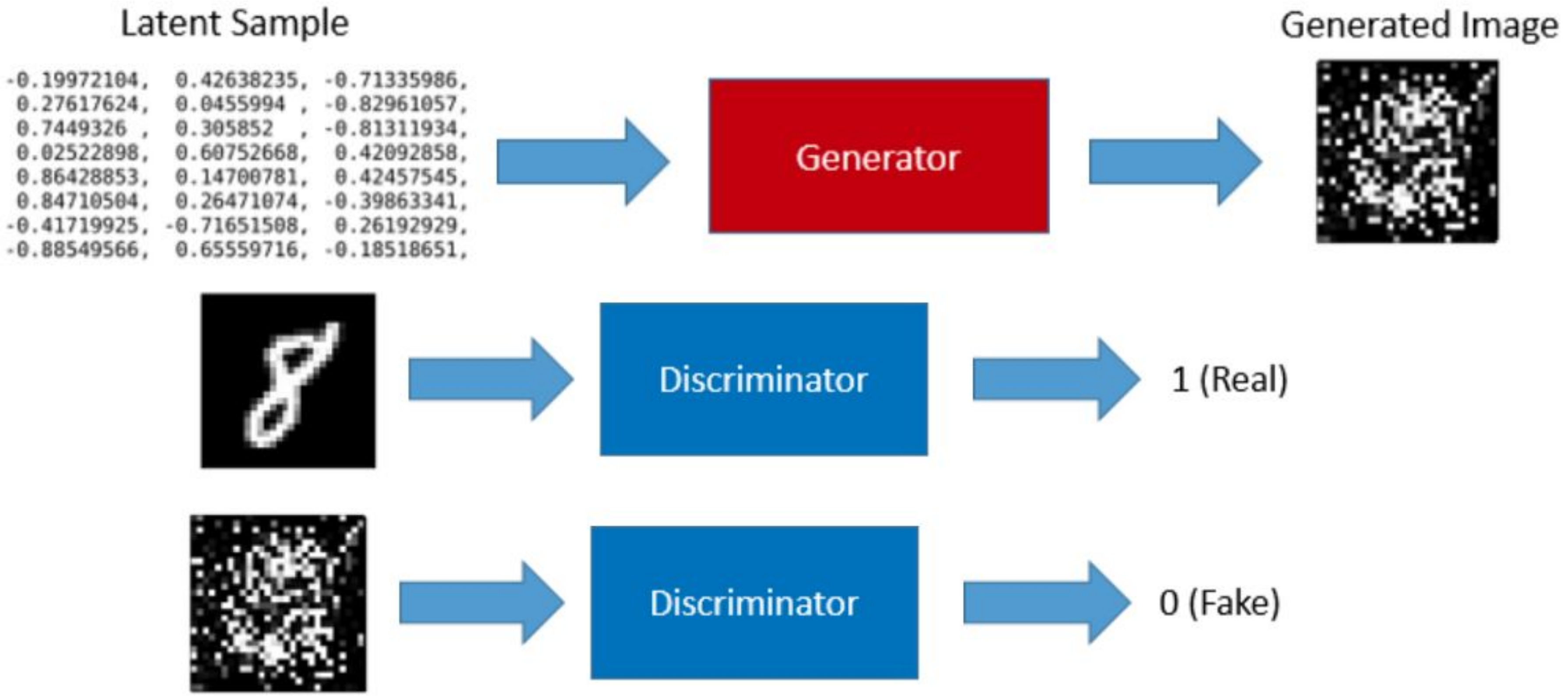
(arXiv:1702.00403v1).



arXiv:1702.00403v1

Generative Adversarial Neural Networks

One need to set two networks: Generator and a Discriminator

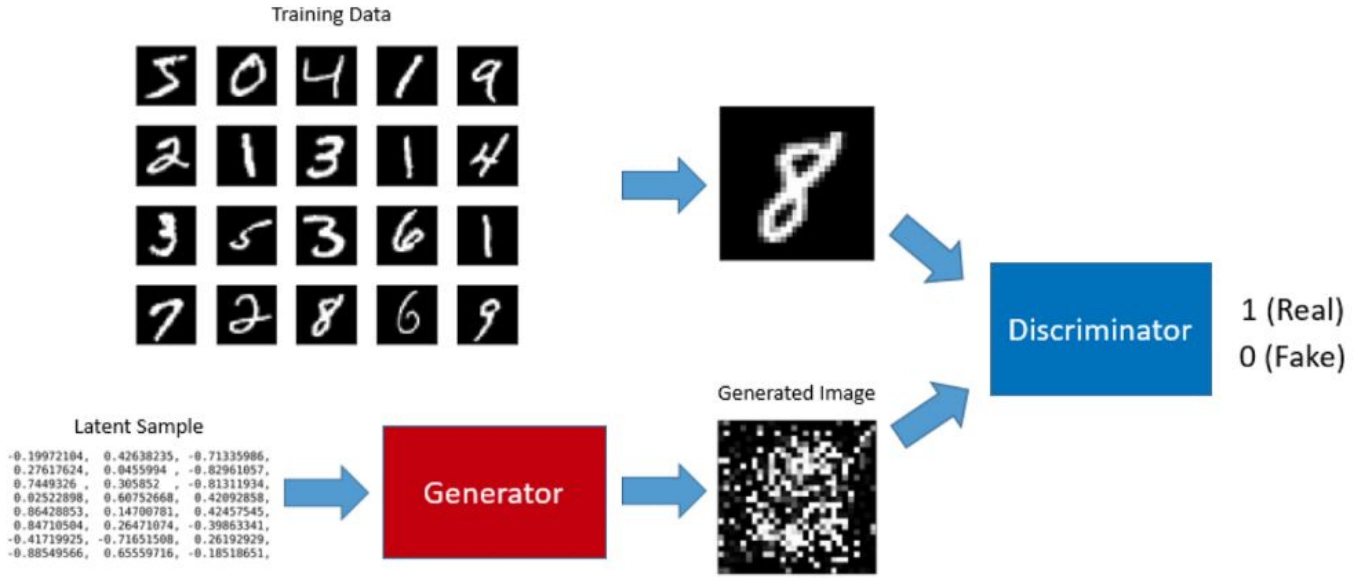


```
-0.19972104,  0.42638235, -0.71335986,
 0.27617624,  0.0455994 , -0.82961057,
 0.7449326 ,  0.305852  , -0.81311934,
 0.02522898,  0.60752668,  0.42092858,
 0.86428853,  0.14700781,  0.42457545,
 0.84710504,  0.26471074, -0.39863341,
-0.41719925, -0.71651508,  0.26192929,
-0.88549566,  0.65559716, -0.18518651,
```

Image credit: https://towardsdatascience.com/understanding-generative-adversarial-networks-4dafc963f2ef

Generative Adversarial Neural Networks

The key here is the training …. There is a loop:
first we train the discriminator. Let the generator create some images out of noise and train **the discriminator only** to define true or false.



```
-0.19972104,  0.42638235, -0.71335986,
 0.27617624,  0.0455994 , -0.82961057,
 0.7449326 ,  0.305852  , -0.81311934,
 0.02522898,  0.60752668,  0.42092858,
 0.86428853,  0.14700781,  0.42457545,
 0.84710504,  0.26471074, -0.39863341,
-0.41719925, -0.71651508,  0.26192929,
-0.88549566,  0.65559716, -0.18518651,
```

Image credit: https://towardsdatascience.com/understanding-generative-adversarial-networks-4dafc963f2ef

Generative Adversarial Neural Networks

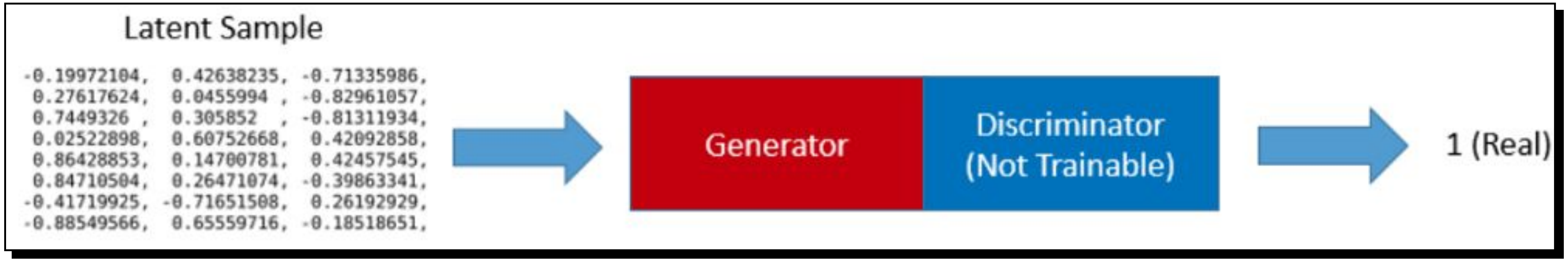
The key here is the training …. There is a loop:
first we train the discriminator. Let the generator create some images out of noise and train **the discriminator only** to define true or false. Consider a Keras model D to be the discriminator , G the Generator and AD the Adversarial Model, i.e., the full model [G,D]

```
generated_images = G.predict(x=random_data, batch_size=batch_size)

d_loss_real = D.train_on_batch(image_real_batch, output_true)
d_loss_fake = D.train_on_batch(generated_images, output_false)
d_loss = 0.5 * np.add(d_loss_fake, d_loss_real)
```

Generative Adversarial Neural Networks

The key here is the training ….
Then we set the Discriminator to be **non-trainable**. After that we optimize the generator. We create some random noise data flows through the Network (Generator and Discriminator). The Generator weights will be updated by backpropagation from the non-trainable Discriminator to the Generator.

```
D.trainable = False
y = np.ones([batch_size, 1])
noise = np.random.uniform(-1.0, 1.0, size=[batch_size, 100])
AD_loss = AD.train_on_batch(random_data, y)
D.trainable = True
```



```
-0.19972104,  0.42638235, -0.71335986,
 0.27617624,  0.0455994 , -0.82961057,
 0.7449326 ,  0.305852  , -0.81311934,
 0.02522898,  0.60752668,  0.42092858,
 0.86428853,  0.14700781,  0.42457545,
 0.84710504,  0.26471074, -0.39863341,
-0.41719925, -0.71651508,  0.26192929,
-0.88549566,  0.65559716, -0.18518651,
```

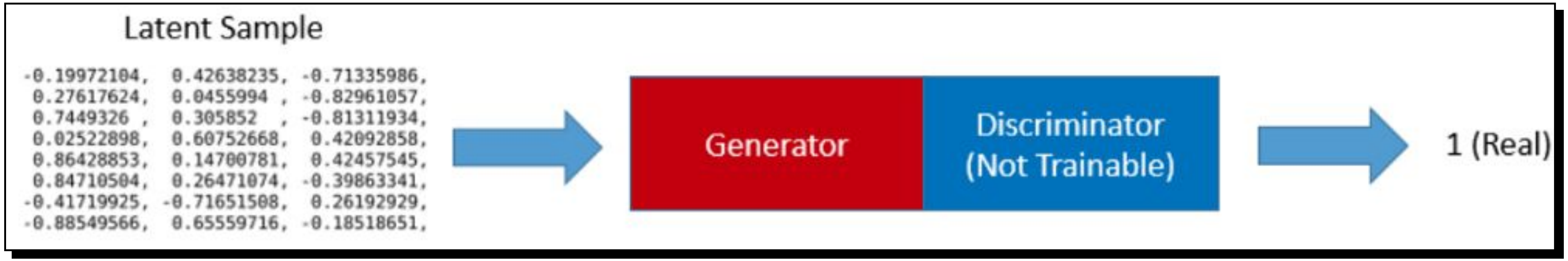
Image credit: https://towardsdatascience.com/understanding-generative-adversarial-networks-4dafc963f2ef

Generative Adversarial Neural Networks are hard to train...

Some general Advice
Gain is particulary sensible to vanishing gradients/ sparse gradients
LeakyReLU is recomend instead of ReLU.

If Discriminator loss converges rapidly and thus prevents the Generator from learning make its learning rate bigger than the Adversarial model learning rate you can also use a different training noise sample for the Generator.

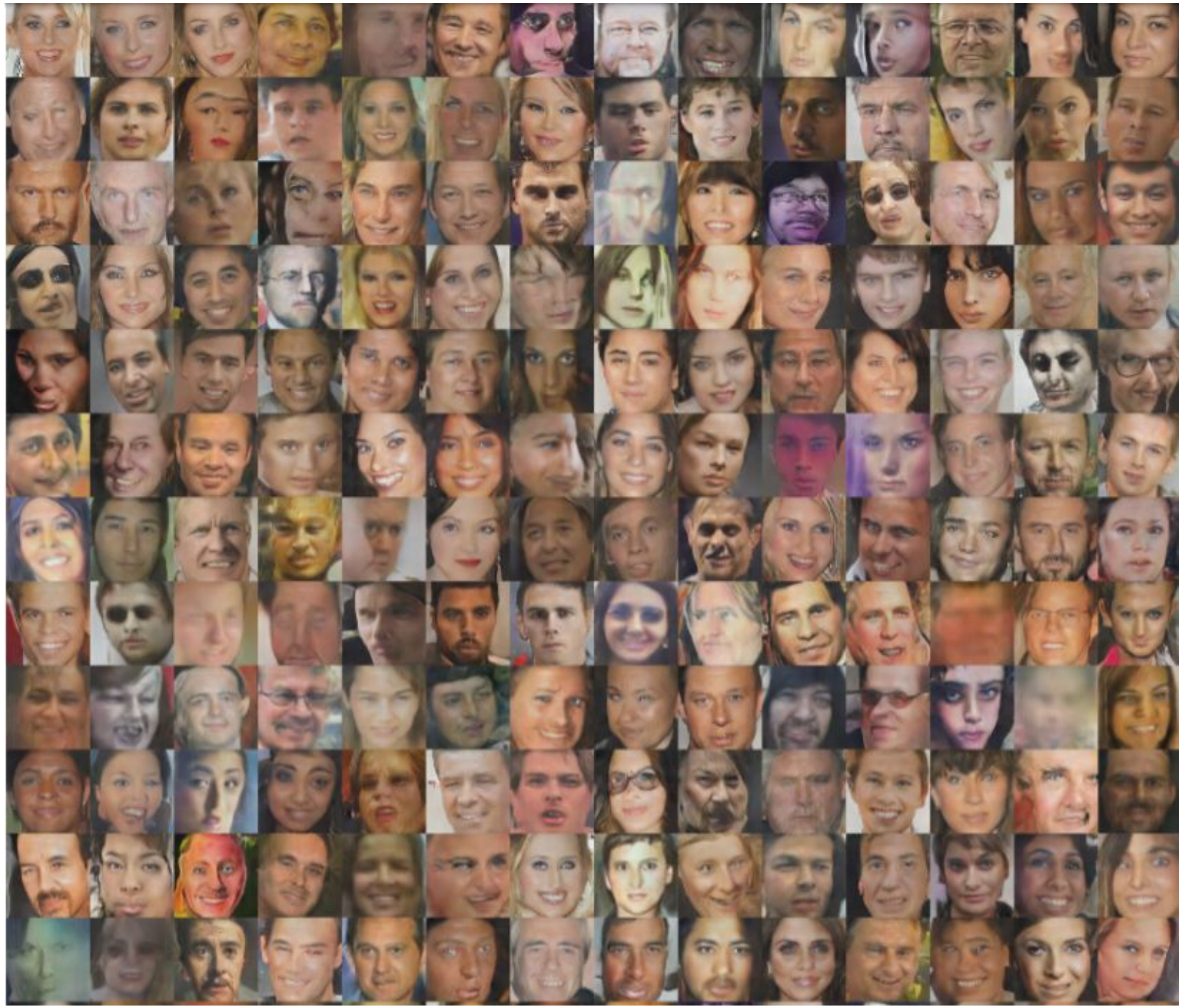
If generated images look like noise. Use dropout on both Discriminator and Generator. Low dropout values generate more realistic images.



```
-0.19972104,  0.42638235, -0.71335986,
 0.27617624,  0.0455994 , -0.82961057,
 0.7449326 ,  0.305852  , -0.81311934,
 0.02522898,  0.60752668,  0.42092858,
 0.86428853,  0.14700781,  0.42457545,
 0.84710504,  0.26471074, -0.39863341,
-0.41719925, -0.71651508,  0.26192929,
-0.88549566,  0.65559716, -0.18518651,
```

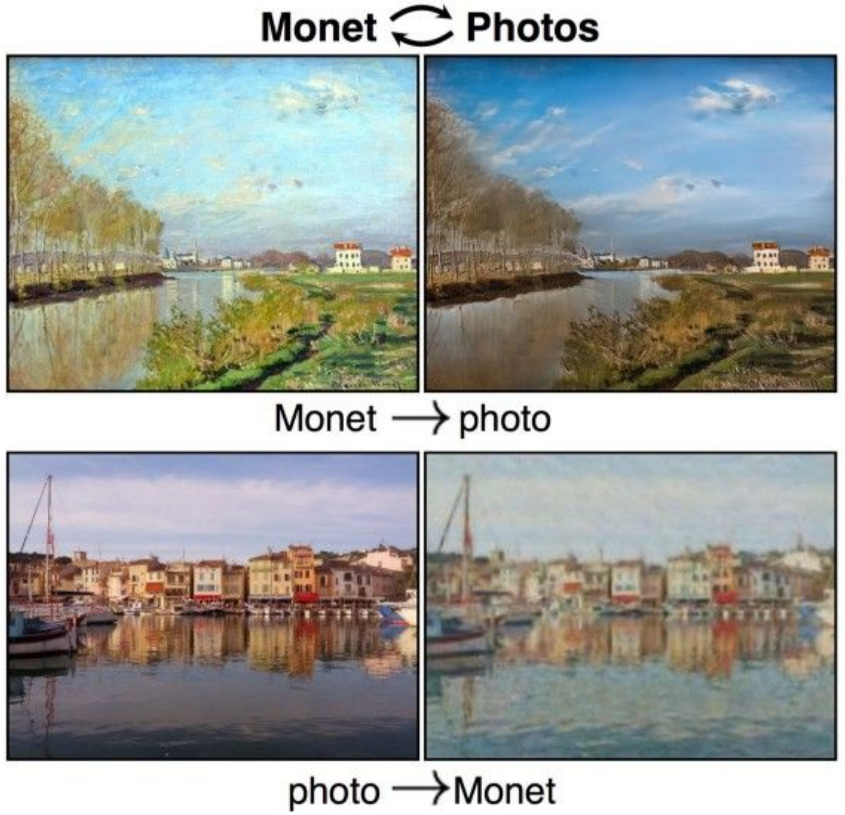
Image credit: https://towardsdatascience.com/understanding-generative-adversarial-networks-4dafc963f2ef

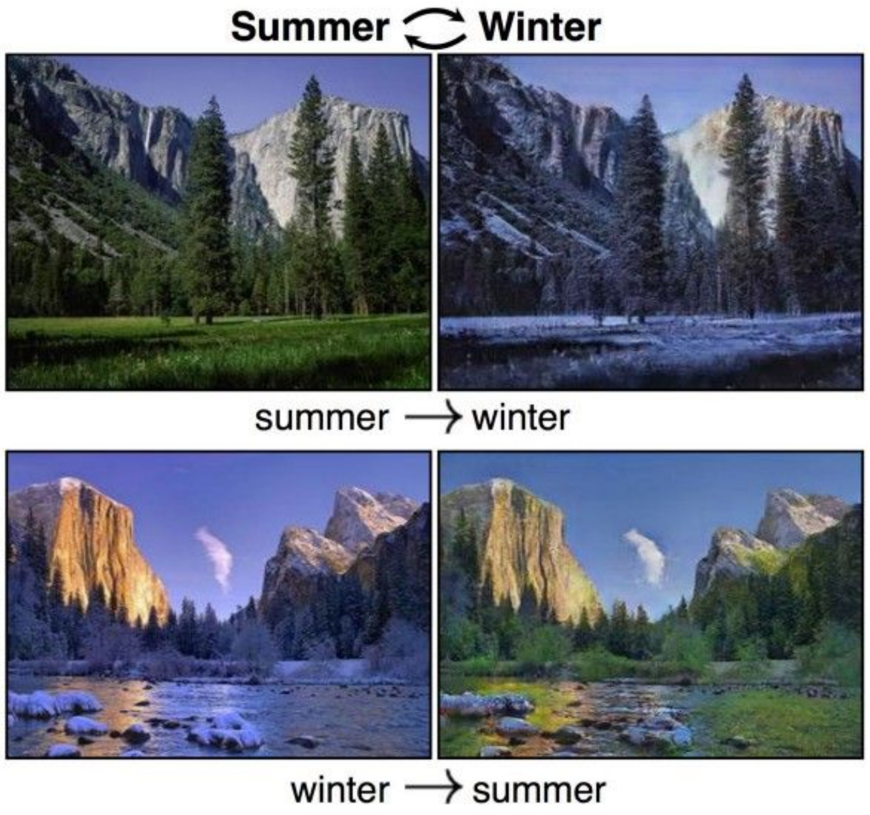
Some Scary smiles from DCGAN



Cycle GAN: Make your own Monet

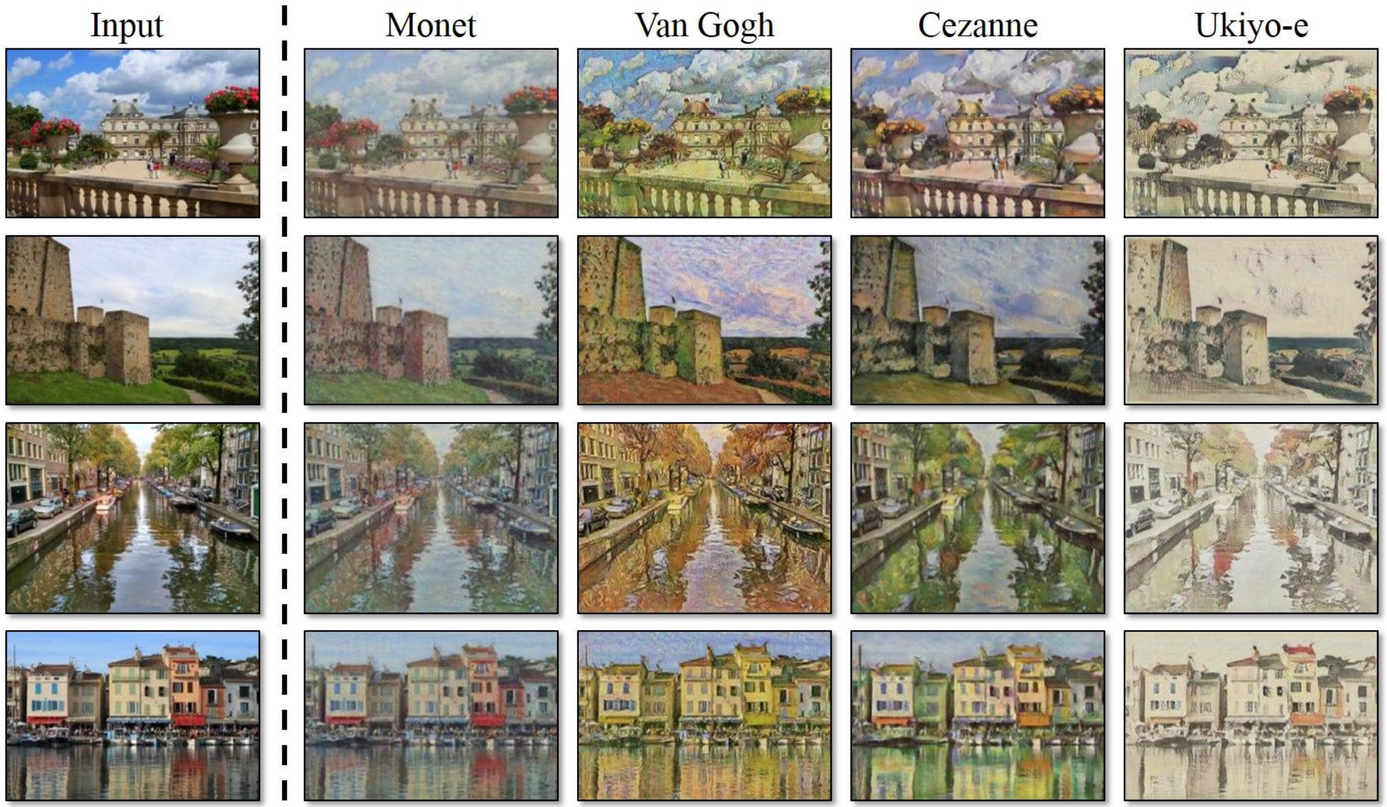
arXiv:1703.10593





Cycle GAN: Make your own Monet

arXiv:1703.10593



What people are doing with this?

DeepCMB: Lensing Reconstruction of the Cosmic Microwave Background with Deep Neural Networks

J. Caldeira[a,1,*], W. L. K. Wu[b], B. Nord[b,c,e], C. Avestruz[a,b], S. Trivedi[d], K. T. Story[f]

arXiv:1810.01483

Deblending galaxy superpositions with branched generative adversarial networks

David M. Reiman[1]★†, Brett E. Göhre[1]‡§
[1]Department of Physics, University of California Santa Cruz, 1156 High Street, Santa Cruz, California 95064, USA

arXiv:1810.10098

Enabling Dark Energy Science with Deep Generative Models of Galaxy Images

Siamak Ravanbakhsh[1], François Lanusse[2], Rachel Mandelbaum[2], Jeff Schneider[1], and Barnabás Póczos[1]

arXiv:1609.05796

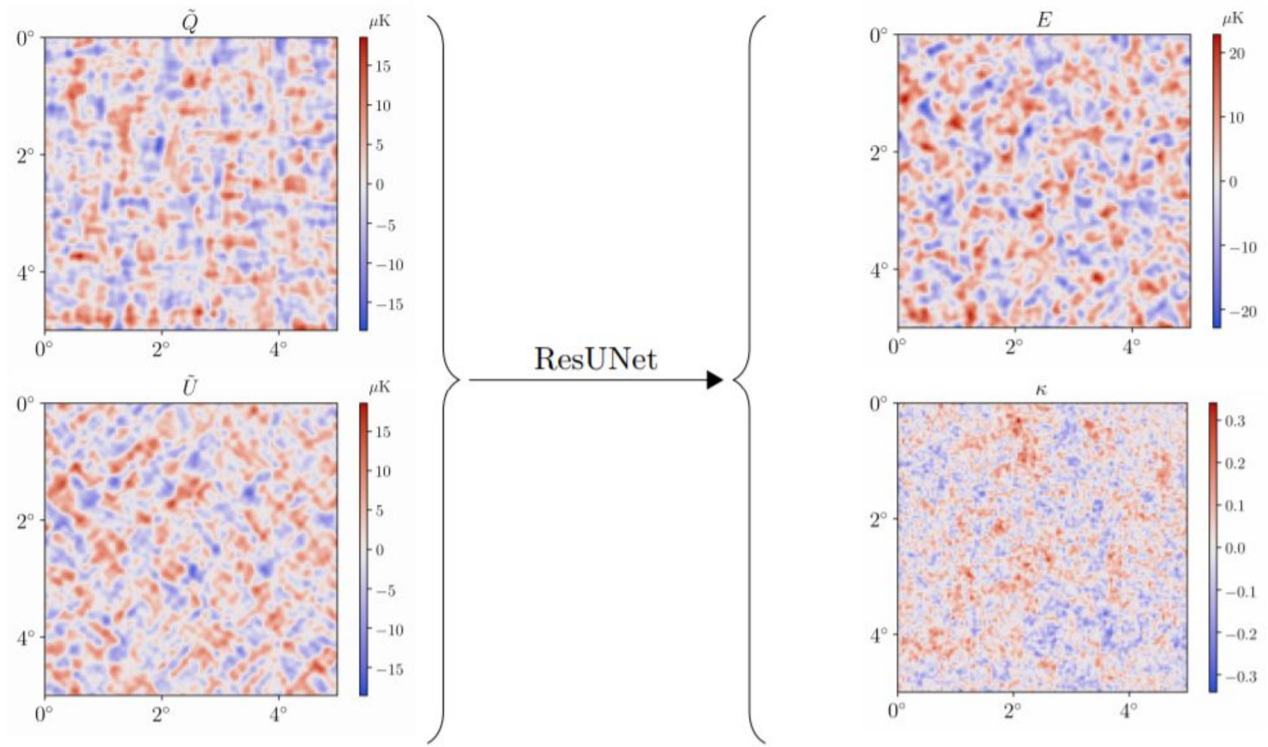
Fast Cosmic Web Simulations with Generative Adversarial Networks

A.C. Rodríguez,[a] T. Kacprzak,[b] A. Lucchi,[c] A. Amara,[b] R. Sgier,[b] J. Fluri,[b] T. Hofmann,[c] and A. Réfrégier[b]

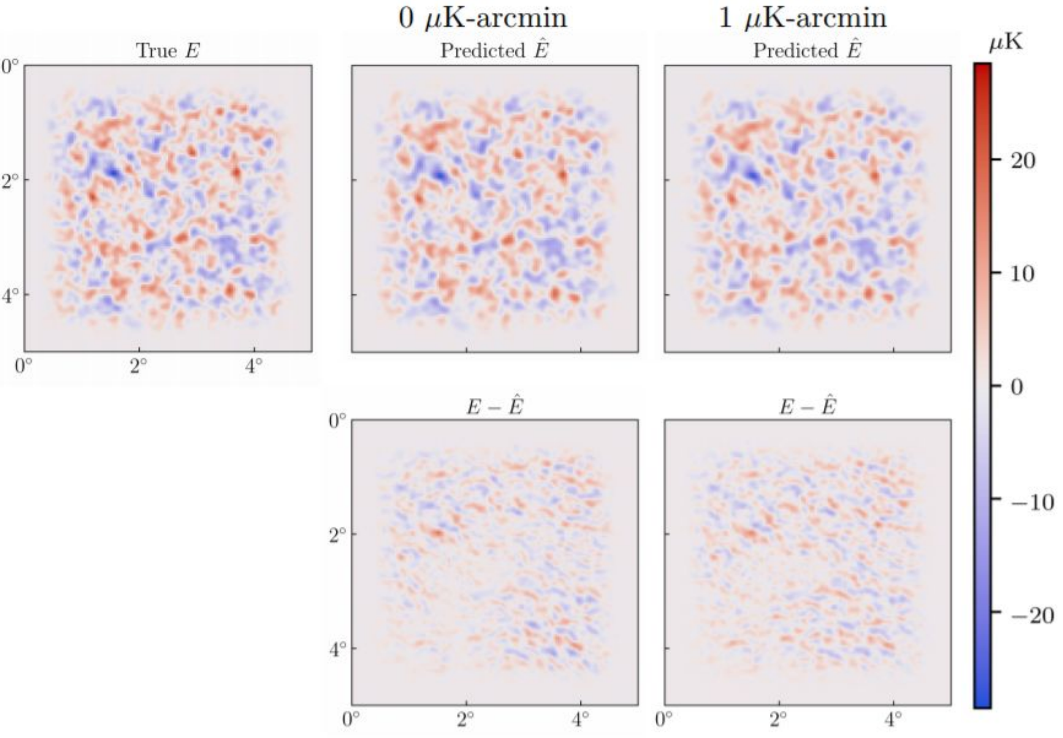
arXiv:1801.09070

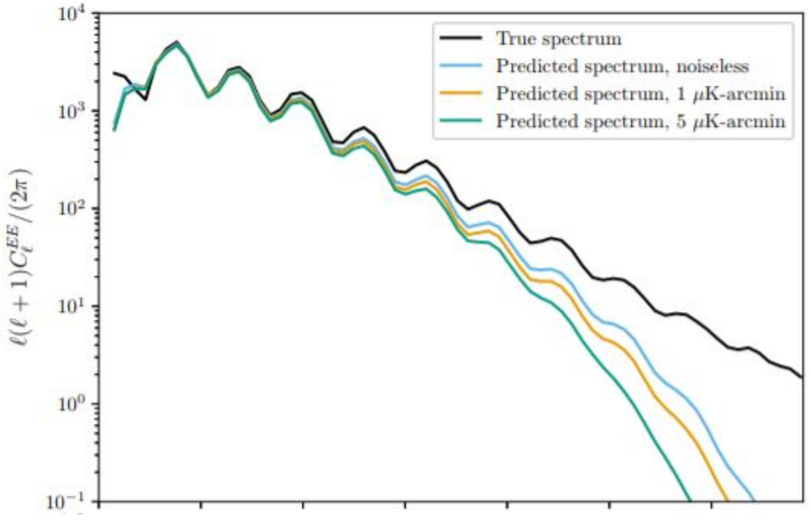
DeepCMB: Lensing Reconstruction of the Cosmic Microwave Background with Deep Neural Networks

CMB polarization maps are usually represented (Q, U) basis corresponds to Stokes parameters and is convenient for mapping onto from the CMB instruments' polarization detector coordinates.



DeepCMB: Lensing Reconstruction of the Cosmic Microwave Background with Deep Neural Networks

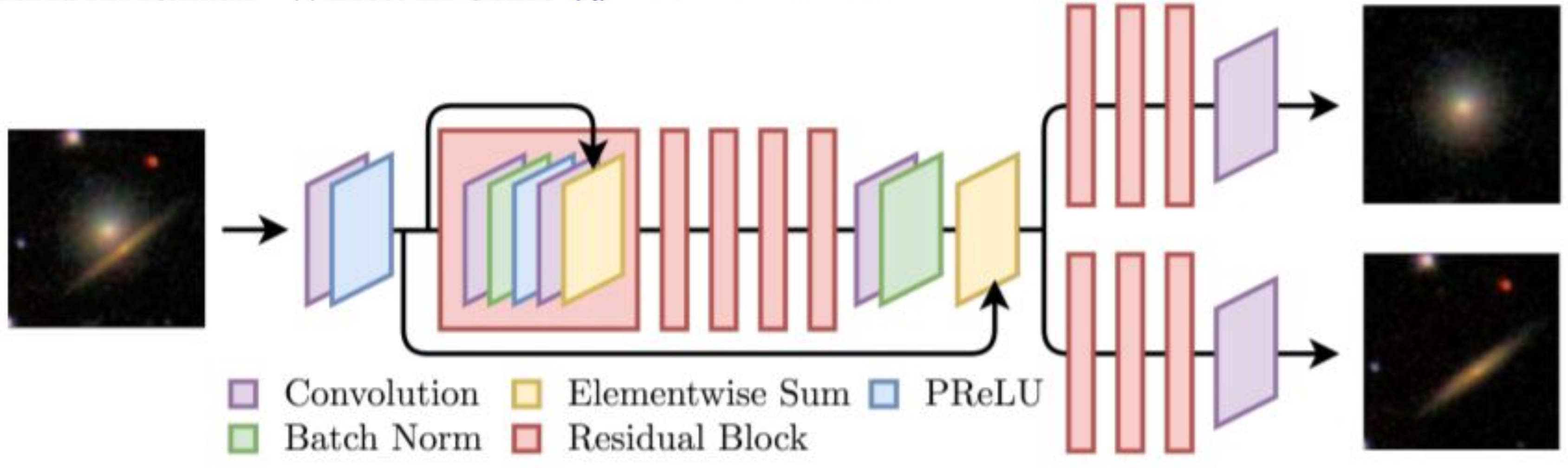




Deblending galaxy superpositions with branched generative adversarial networks

David M. Reiman[1]*†, Brett E. Göhre[1]‡§

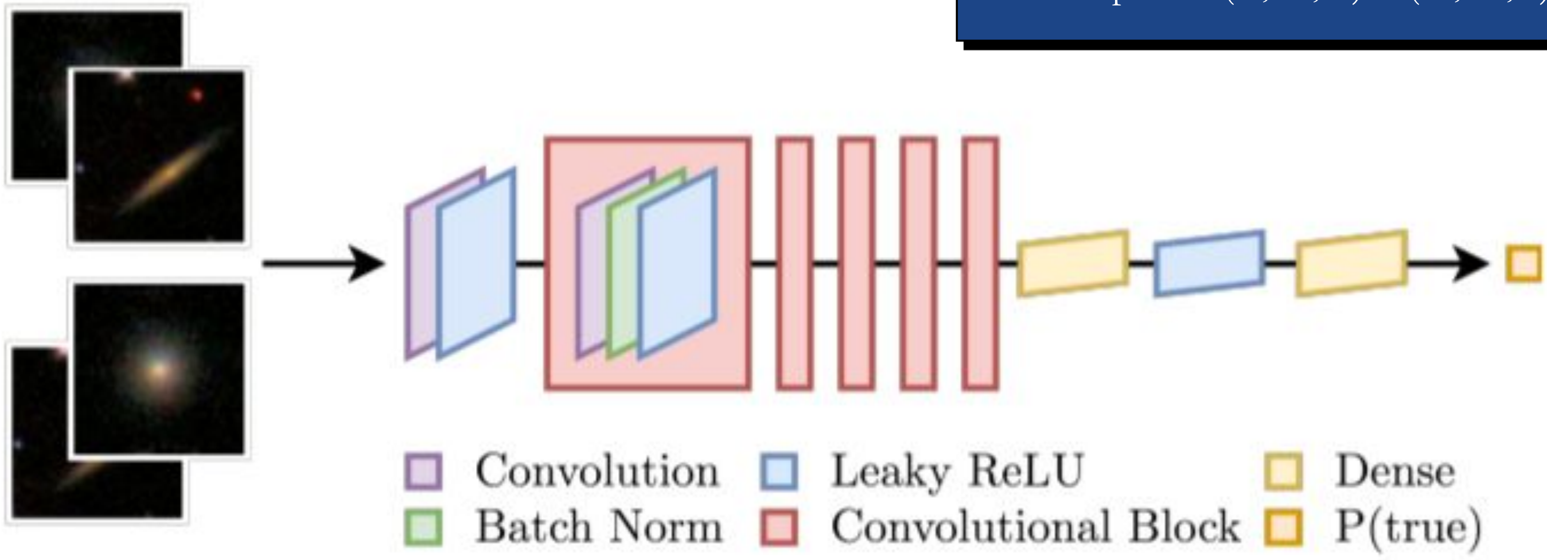
[1]



Deblending galaxy superpositions with branched generative adversarial networks

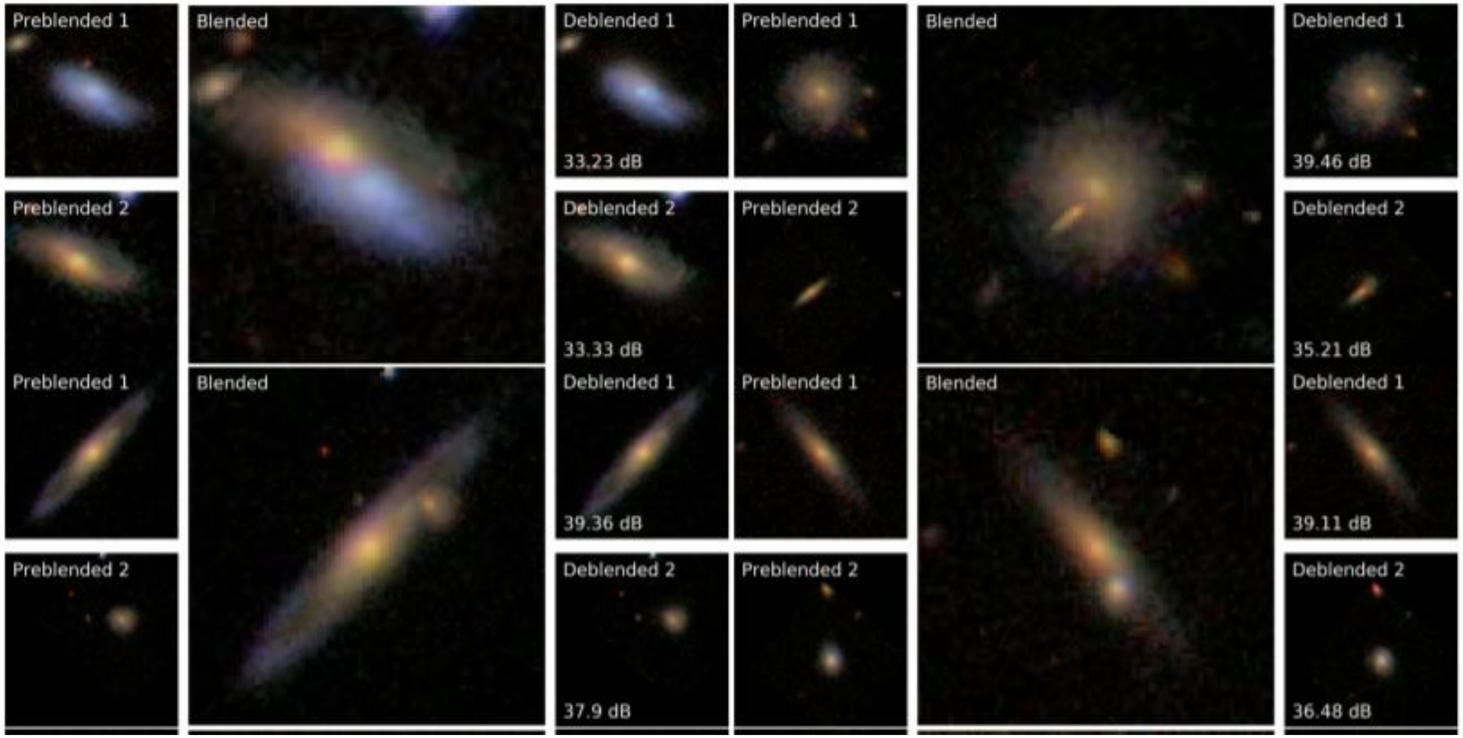
David M. Reiman[1]*†, Brett E. Göhre[1]‡§
[1]Department of Physics, University of California Santa Cruz, 1156 High Street, Santa Cruz, California 95064,

The authors used 141,553 images available from Galaxy Zoo. They used 3 bands in a JPEG format and therefore has a bit depth of 8 bits per color channel with dimension (H, W, C) = (424, 424, 3) where H, W and C are the height, width and channel dimensions, respectively and were cropped and downsampled to (H, W, C) = (80, 80, 3).



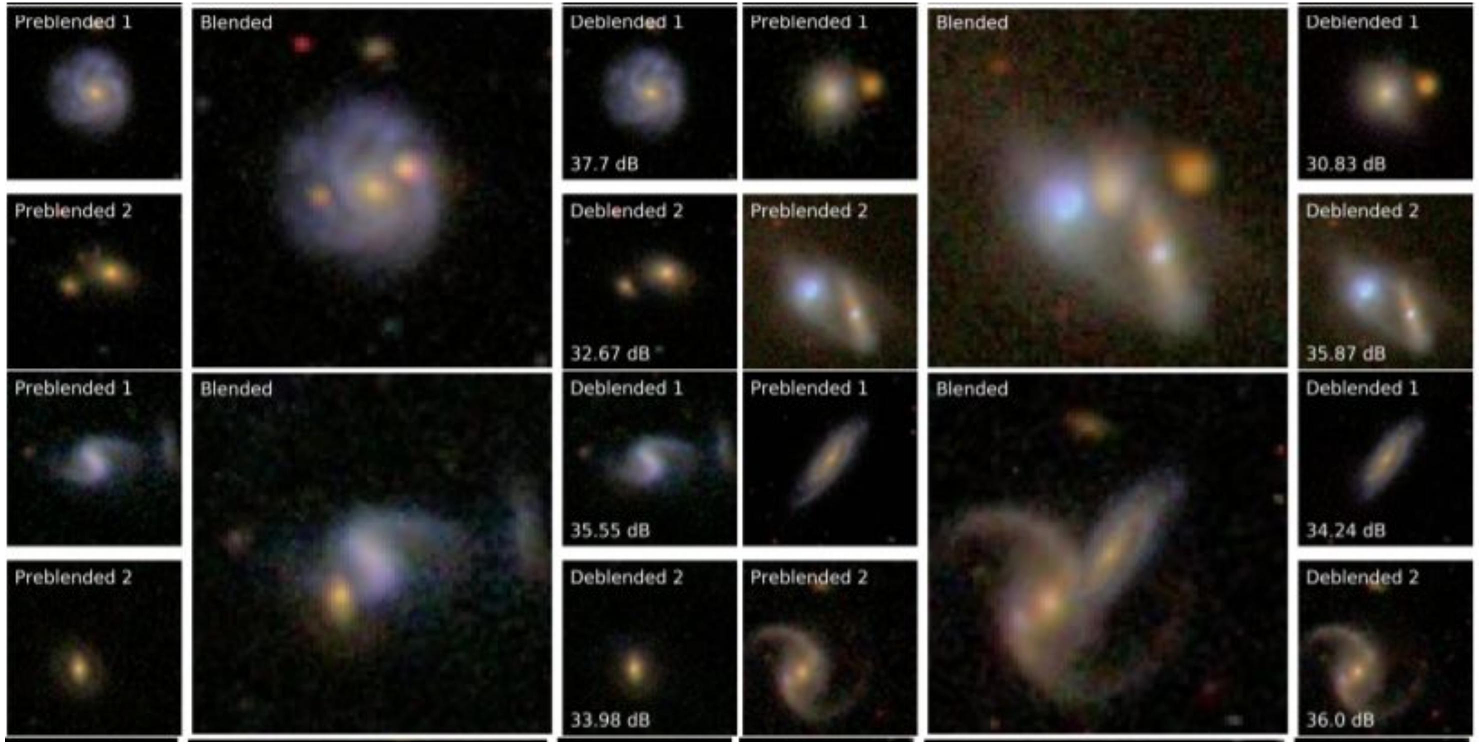
Deblending galaxy superpositions with branched generative adversarial networks

David M. Reiman[1]★†, Brett E. Göhre[1]‡§
[1]Department of Physics, University of California Santa Cruz, 1156 High Street, Santa Cruz, California 95064, USA



Deblending galaxy superpositions with branched generative adversarial networks

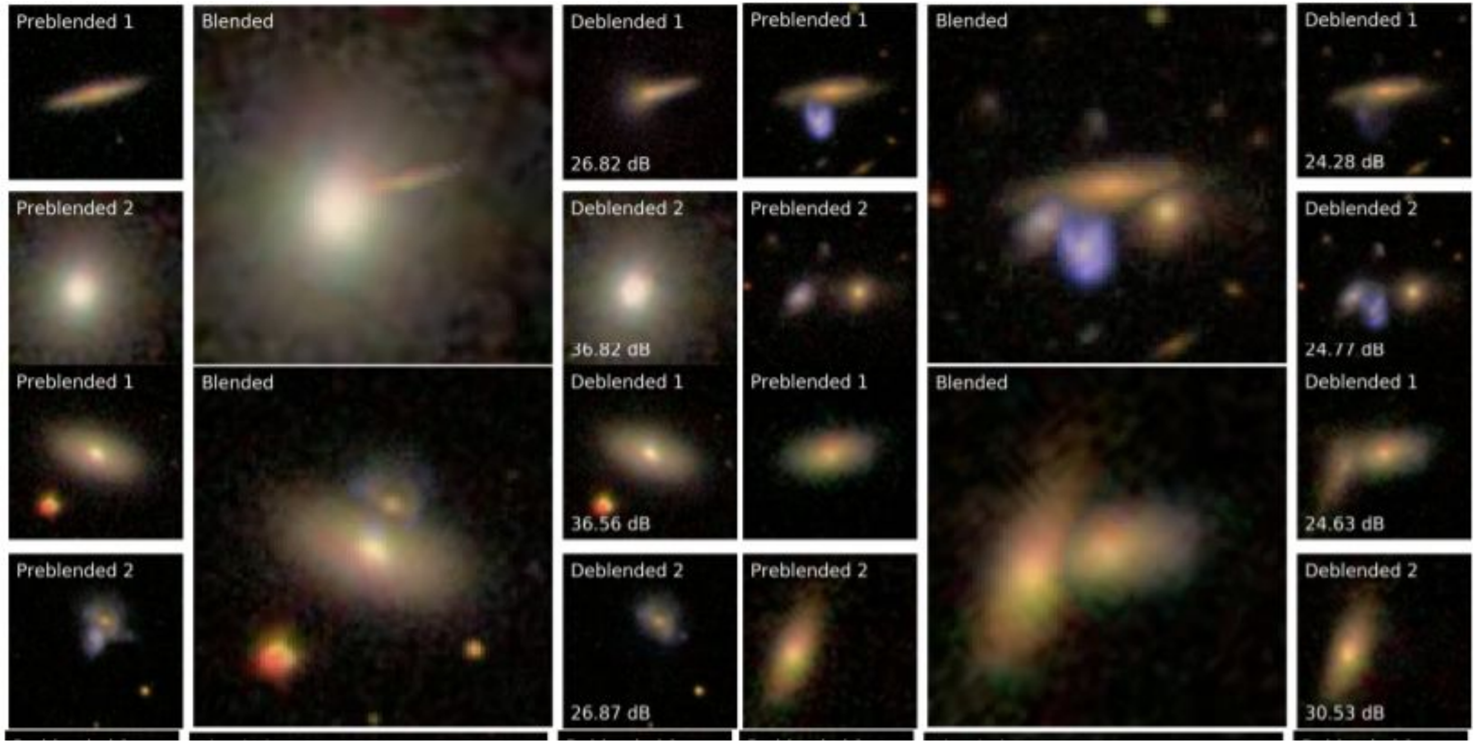
David M. Reiman[1]*†, Brett E. Göhre[1]‡§
[1]Department of Physics, University of California Santa Cruz, 1156 High Street, Santa Cruz, California 95064, USA



Deblending galaxy superpositions with branched generative adversarial networks

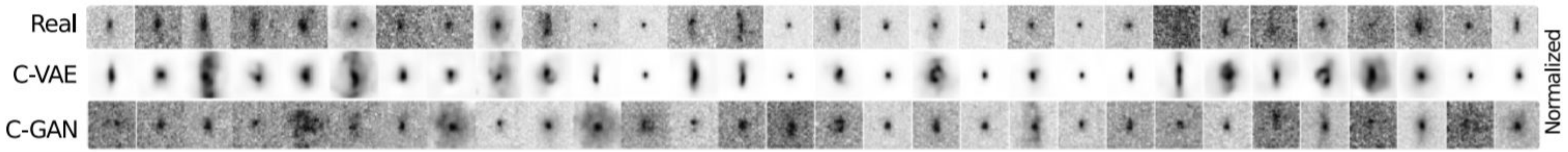
David M. Reiman[1]★†, Brett E. Göhre[1]‡§
[1]Department of Physics, University of California Santa Cruz, 1156 High Street, Santa Cruz, California 95064, USA

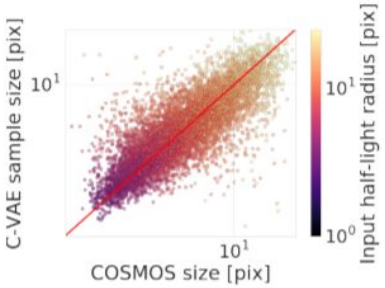
Failures



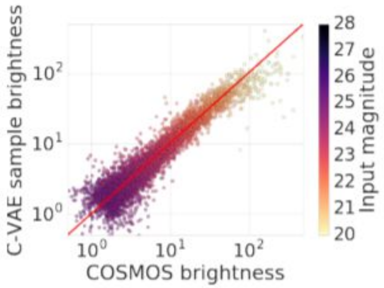
Enabling Dark Energy Science with Deep Generative Models of Galaxy Images

Siamak Ravanbakhsh[1], François Lanusse[2], Rachel Mandelbaum[2], Jeff Schneider[1], and Barnabás Póczos[1]

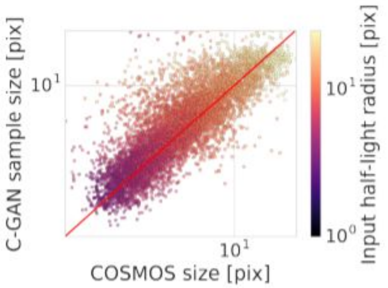




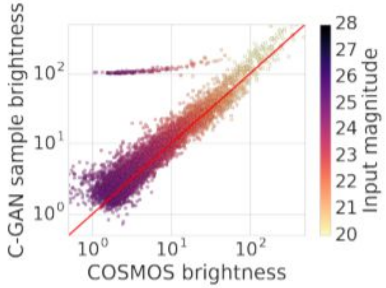
(a) Galaxy sizes



(b) Galaxy brightness



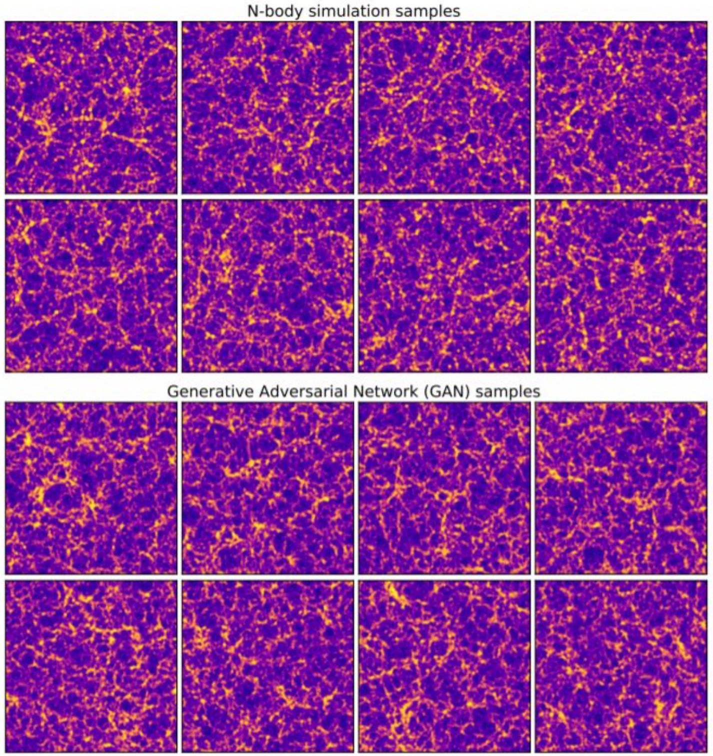
(a) Galaxy sizes



(b) Galaxy brightness

Fast Cosmic Web Simulations with Generative Adversarial Networks

A.C. Rodríguez,[a] T. Kacprzak,[b] A. Lucchi,[c] A. Amara,[b] R. Sgier,[b] J. Fluri,[b] T. Hofmann,[c] and A. Réfrégier[b]


N-body simulation samples
Generative Adversarial Network (GAN) samples

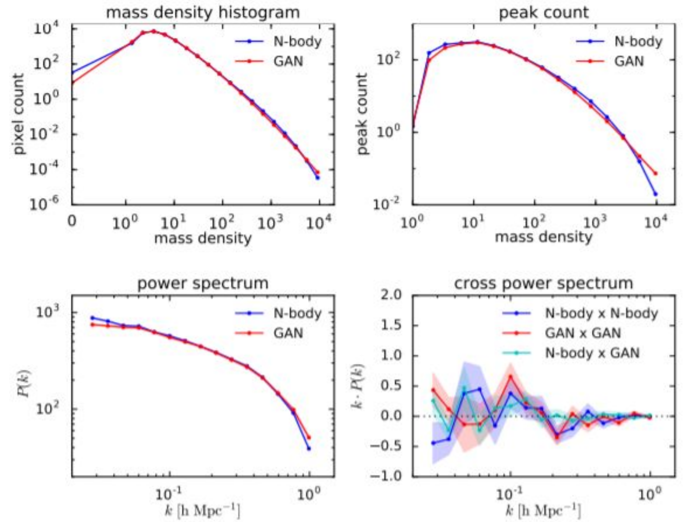
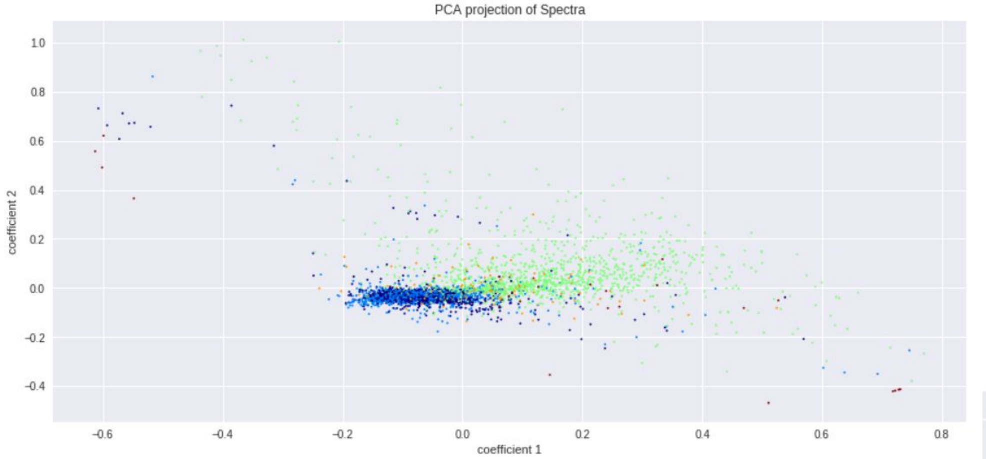
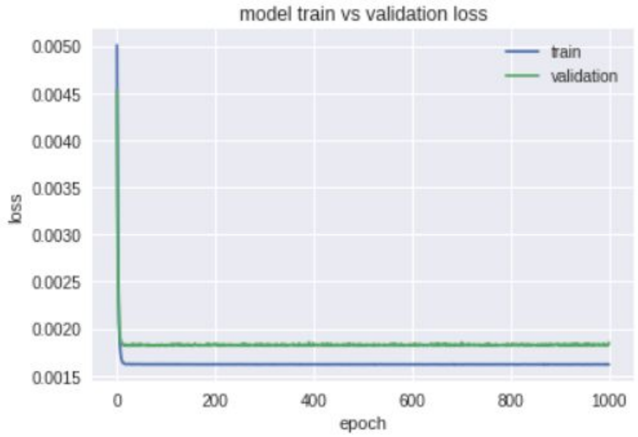
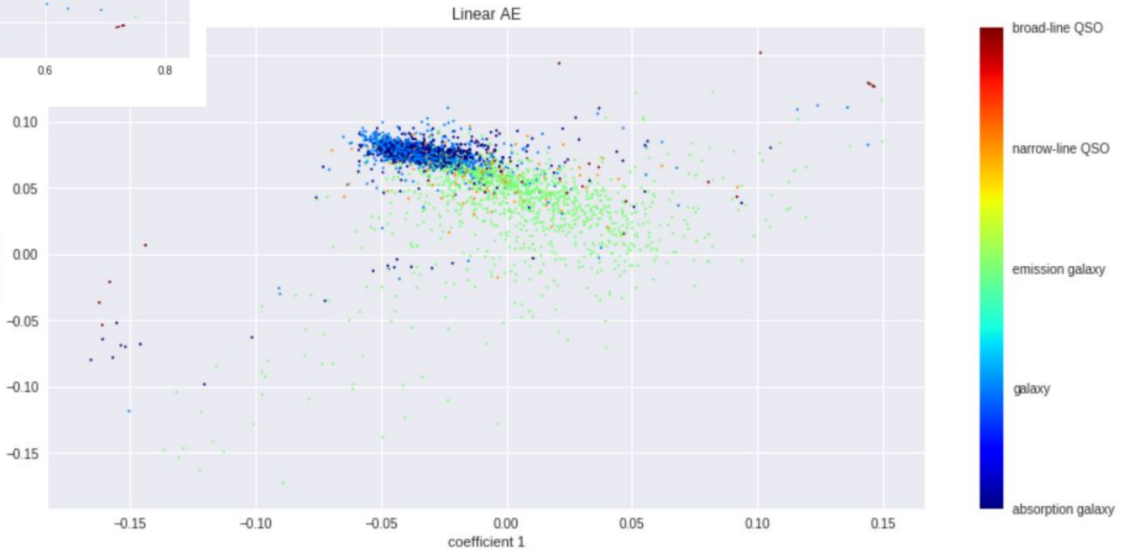


Figure 2: Comparison of summary statistics between N-body and GAN simulations, for box size of 500 Mpc. The statistics are: mass density histogram (upper left), peak count (upper right), power spectrum of 2D images (lower left) and cross power spectrum (lower right). The cross power spectrum is calculated between pairs N-body images (blue points), between pairs of GAN images (red points), and between pairs consisting of one GAN and one N-body image (cyan points). The power spectra are shown in units of h^{-1} Mpc, where $h = \mathrm{H}_0/100$ corresponds to the Hubble parameter. The standard errors on the mean of the shown with a shaded region, and are too small to be seen for the first three panels.

Example 03 - PCA vs AE

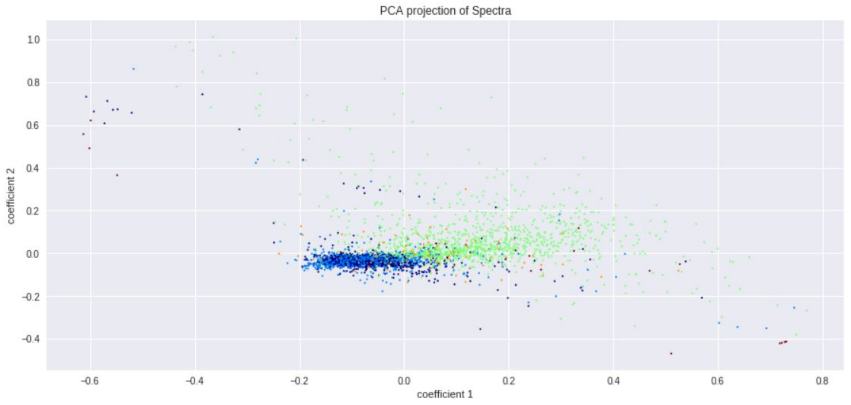


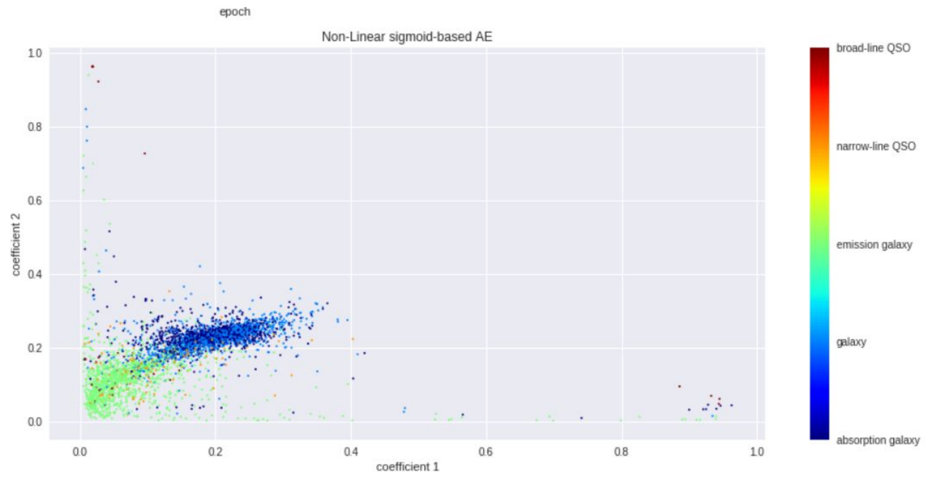


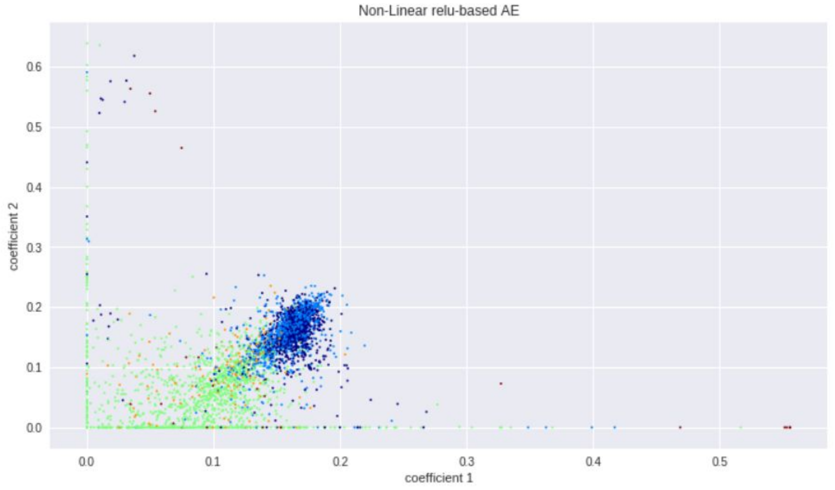


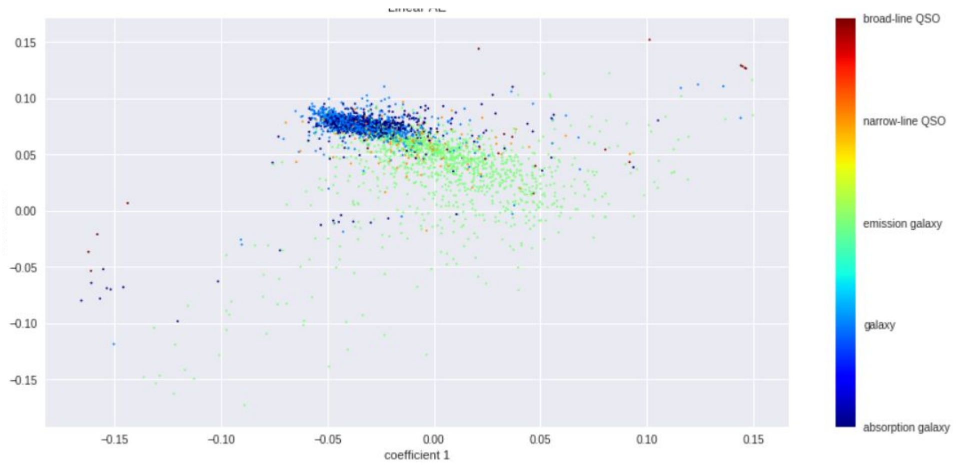
Galaxy Spectra from SDSS available from astroML example. http://www.astroml.org/book_figures/chapter7/fig_PCA_LLE.html

Example 03 - PCA vs AE









CEFET/RJ

CBPF

Deep Learning Applications in Astronomy

Clécio R. Bom

clearnightsrthebest.com

debom@cbpf.br

February 12, 2019