MASARYKOVA UNIVERZITA
Přírodovědecká fakulta
Ústav teoretické fyziky a astrofyziky

BAKALÁŘSKÁ PRÁCE

Brno 2018 Antónia Vojteková

MASARYKOVA UNIVERZITA

Přírodovědecká fakulta

Ústav teoretické fyziky a astrofyziky

STUDIUM GALAXIÍ PROSTŘEDNICTVÍM GAN SITÍ

BAKALÁŘSKÁ PRÁCE

ANTÓNIA VOJTEKOVÁ

VEDOUCÍ BAKALÁŘSKÉ PRÁCE:
Mgr. FILIP HROCH, Ph.D.

BRNO 2018

Bibliografický záznam

Autor:	Antónia Vojteková Přírodovědecká fakulta, Masarykova univerzita Ústav teoretické fyziky a astrofyziky

Název práce:	Studium galaxií prostřednictvím GAN sítí
Studijní program:	Fyzika
Obor:	Astrofyzika
Vedoucí práce:	Mgr. Filip Hroch, Ph.D.
Akademický rok:	2017/2018
Počet stran:	vii + 57
Klíčová slova:	galaxie, průzkum, šum obrázků, gan, neuronová síť

Bibliographic record

Author:	Antónia Vojteková Faculty of Science, Masaryk University Department of Theoretical Physics and Astrophysics

Title of Thesis:	GAN network galaxy probe
Degree Programme:	Physics
Field of study:	Astrophysics
Supervisor:	Mgr. Filip Hroch, Ph.D.
Academic Year:	2017/2018
Number of Pages:	vii + 57
Keywords:	galaxy, survey, image noise, gan, neural network

ABSTRAKT

V současné astronomické éře očekáváme příliv velkého objemu dat, jehož zpracování se vymyká lidským silám. Z těchto důvodů je vhodné vyvinout metody, které efektivně a spolehlivě analyzují data. Práce reaguje na současnou situaci a je inspirována projektem GalaxyGAN. Jejím cílem jest prozkoumání užití neuronových sítí při zpracování astronomických snímků. Pozorování astronomických objektů je limitováno mnoha faktory a jedním z nich je šum. Náš výzkum se zaměřil na odstranění šumu obrazových dat pomocí „generative adversarial network“ – GAN. Výsledky ukázaly, že použití těchto sítí na daný problém je možné. Při porovnání s dalšími dostupnými metodami vykazují srovnatelné, místy i výrazně lepší, výsledky.

ABSTRACT

In the present era of astronomy, we are expecting a large amount of data that can not be fully processed by humans. Because of this, it is convenient to develop a method that analyzes the data efficiently and reliable. Our work reacts to the current situation and it is inspired by the *GalaxyGAN* project we have been exploring the use of neural networks in the processing of astronomical images. Observation of astronomical objects is limited by many factors and one of them is noise. Our research has focused on the removal of noise through generative adversarial networks (GANs). The results have shown that neural networks can be used for a given problem, and in comparison with available methods, the results comparable and sometimes even better.

Poďakovanie

Chcela by som poďakovať najmä môjmu vedúcemu *F.H.* za pomoc a podporu nie len počas písania bakalárskej práce ale aj počas celého štúdia. Za odborné konzultácie ďakujem *M.H.*. Ďalšie poďakovanie patrí kolegom, ktorí ma počas práce podporovali a motivovali v študijnej sfére *N.W., M.K., J.S, T.B.* aj tým, ktorí mi pomohli pri skúškach a udržali ma v dobrej fyzickej a psychickej kondícii *P.K., M.P., P.N., O.L., A.S.*.
Veľká vďaka patrí aj mojej rodine, rodičom a sestre, ktorí pri mne stáli od úplných začiatkov. Špeciálne poďakovanie patrí *V.M.*.
Za využívanie výpočetnej techniky by som sa chcela poďakovať Fakulte informačných technológií VUT v Brne. Za poskytnutie dát ďakujem Sloan Digital Sky Survey (SDSS) a Hubble Space Telescope (HST).

Prohlášení

Prohlašuji, že jsem svoji bakalářskou práci vypracovala samostatně s využitím informačních zdrojů, které jsou v práci citovány.

Brno 24. května 2018

Podpis autora

Contents

Introduction

A noise limits observation of astrophysical objects such as galaxies. In recent years, there has been an interesting breakthrough in inverse tasks, often used to get image data with less noise. In 2017, Kevin Schawinski (Schawinski et al., 2017) creates a method using a generative adversarial network (GAN) trained on galaxy images that can recover features from artificially degraded images with worse seeing and higher noise than the original with a performance that far exceeds any simple deconvolution. The main goal of this work is to reproduce their network, confirm the results.

Nowadays the amount of data is huge and we need to find the efficient way of automatic processing and analyzing the data. The machine learning provides a wide spectrum of the method which helps with the data analysis. Machine learning can helps with different astronomical problems e.g classification of light curves (Janák, 2012), galaxy classification (Kosiba, 2017), discovery and detection of X-ray cavities (Fort, 2017) or cosmic web simulations (Rodriguez et al., 2018), etc.. Machine learning becomes important part of astroinformatics and it is important to explore the different methods of its use in practice.

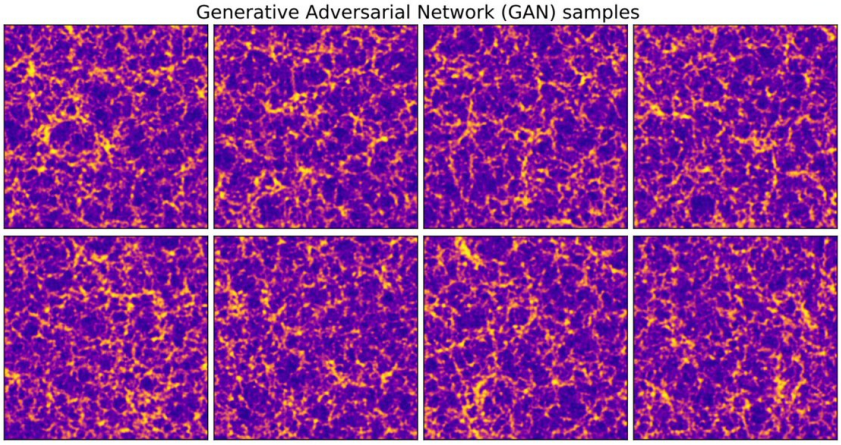


Figure 1.1: Cosmic web generated with GAN (Rodriguez et al., 2018).

Introduction to neural network

2.1 Inspiration for mathematical neuron

The inspiration for neural network comes from nature – the human brain inspiration. The brain is one of the most complicated structures in the known universe and our knowledge about the brain are still incomplete. Cells in the human brain (Figure 2.1), neurons, are main functional units, which receive, process, and transmit chemical and electrical signals. Signals arrive at the dendrites, then travels into the cell body. If the integrated signal in cell body exceeds a certain threshold, it produces an output signal passed on by the axon, which is connected to other neurons by synapses.

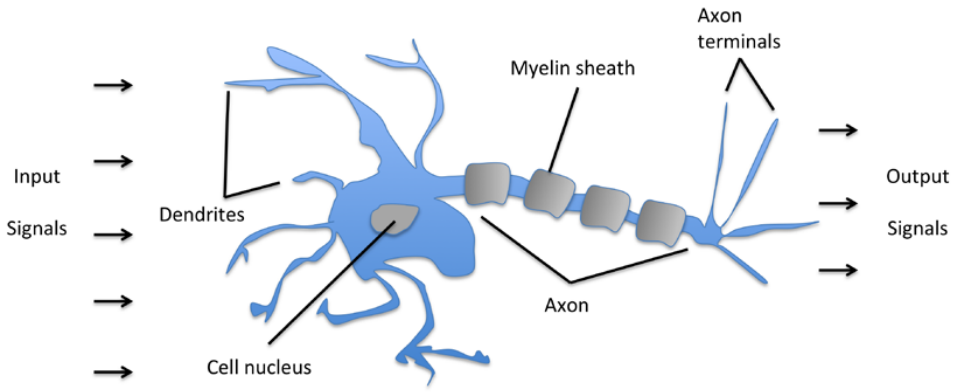


Figure 2.1: Visualization of human neuron (Raschka, 2015)

.

Mathematical neurons, used in neural network, work on the same base. Every n neuron connection has defined weight w_n. Informations flow through connections to the cell body, where it makes sum of input values and corresponding weights,

$$z = \sum_{n=1} w_n x_n, \tag{2.1}$$

then applies an activation function

$$h = f(z). \tag{2.2}$$

If result exceeds threshold, the information is send to a next neuron (Figure 2.2).

In practice, we combine neurons to layers (Figure 2.3). The first layer is called input layer, followed by several hidden layers and finished by output

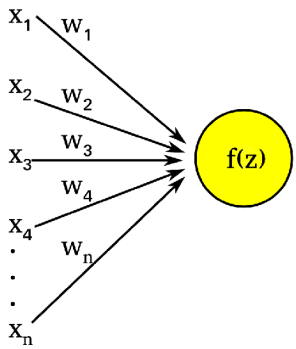


Figure 2.2: Scheme of neuron

layer. A number of hidden layers defines a depth of neural network. Neurons, in the first hidden layer, are fully connected to neuronss in the input layer and neurons in the output layer are fully connected to neurons in the last hidden layers. Every connection between neurons has a weight, which is trainable parameter.

Let's look how information flows through neural network and how we can train it. The first step is to sum signal, which is getting into the neuron. It is done by summing all signal from previous layers multiplied by corresponding weights

$$z = \sum_i w_i x_i + b_i, \tag{2.3}$$

where x_i is an input from previous, w_i is weight, b_i is bias and z is the so-called net input. The bias helps to improve performance and a shift activation function, which may be critical for successful learning. Both weights and biases are matrixes. Thereafter we apply a non-linear activation function $f(z)$ and this value is used as input for the next layer. Output layer is followed by the loss function, which is used to update weights and biases as feed-back.

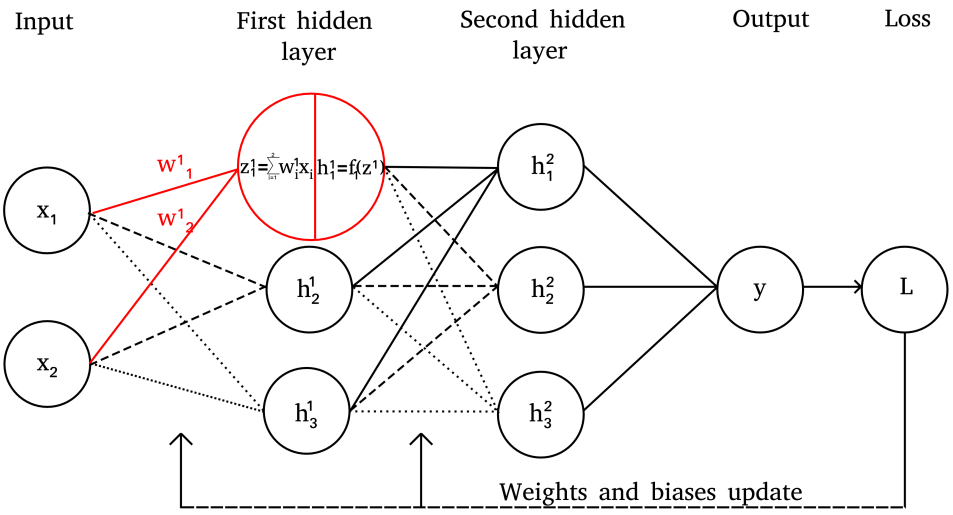


Figure 2.3: Illustration of fully connected neural network consisting the input layer, two hidden layers and the output layer. Information from the output layer is used to calculate loss and update weights through back-propagation.

2.2 The activation function

The activation function is the nonlinear transformation that we do over the signal $f(z)$, transformed output is then sent to the next layer of neurons as input. Without an activation function, a neural network would do a linear transformation. Commonly used activation function are:

Sigmoid has a mathematical form

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.4}$$

The sigmoid function takes real number values as an input and transforms them to values in the range [0,1]. The output of the function is then interpreted as the probability that our sample belongs to the particular class.

Hyperbolic tangent (tanh) the difference between sigmoid and tanh is that tanh tranforms values to the range [-1,1] and it's output is zero centered.

Rectified linear unit or shortly ReLU computes $f(x) = \max(0, x)$, in the other words, ReLU set up values $x < 0$ to zero and for $x > 0$, $f(x) = x$.

Leaky ReLU in contrast with ReLU, values $x < 0$ are not set to zero but to ax, $f(x) = \max(ax, x)$, where a is a small number, in our work 0.02.

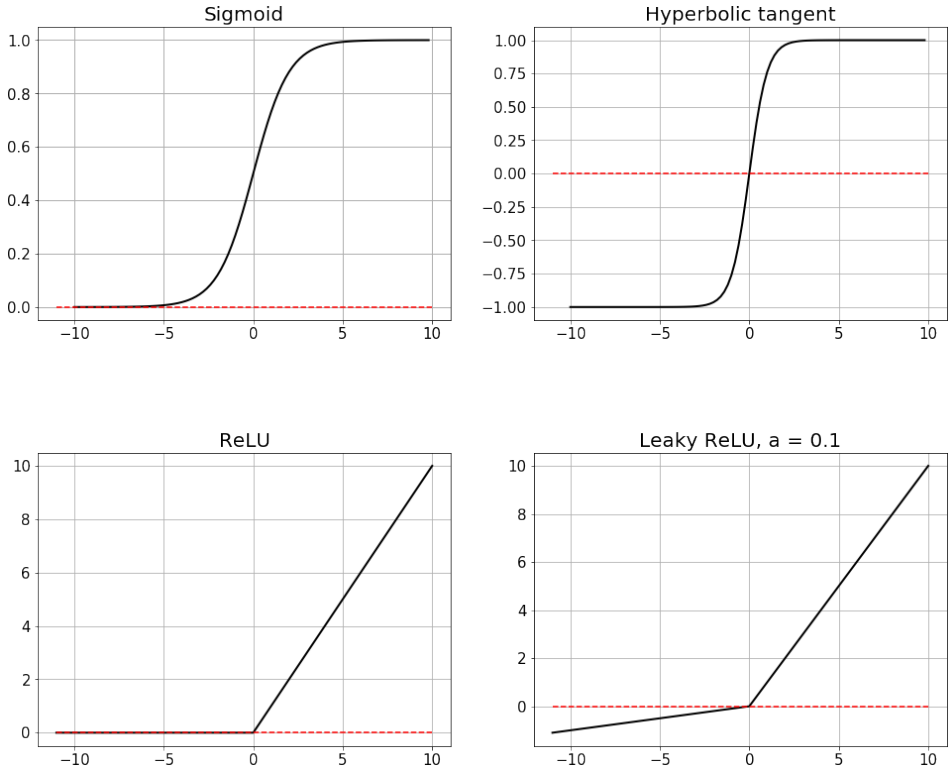


Figure 2.4: Shapes of activation functions

2.3 The loss function

The loss or cost function (L) measures a quality of trainable parameters and how they should be updated. The goal is to minimize loss function by changing weights and biases. There are many types of loss function, in our work we used:

Mean absolute error (MAE) it computes

$$L = \frac{1}{N} \sum_{n=1}^{N} |y_n - y'_n|, \tag{2.5}$$

where y' is predicted value and y is a true value. When y and y' are images, a difference between every pixel is computed and then mean is made.

Sigmoid cross-entropy predict probability that our sample belongs to certain class, it gives us number between zero and one. In our case, we want neural network to predict that the sample is $True/1$ or $False/0$.

$$P = y \log[\sigma(y')] - (1 - y) \log[1 - \sigma(y')], \tag{2.6}$$

where y is true label of the sample, y' is predicted value and $\sigma(y')$ is the sigmoid function.

Binary cross-entropy same as sigmoid cross-entropy, binary cross-entropy predicts probability that our sample belongs in one of two categories

$$P = y \log(y') - (1 - y) \log(1 - y'), \tag{2.7}$$

where y is true label and y' is predicted label of sample.

2.4 Optimization method

The randomly selected optimizing of the loss function is the gradient descent method. The gradient descent is an algorithm which is used to find the minimum of the lost function by updating trainable parameters (weights and biases) via back-propagation. In practice, we first compute gradient of loss function L and then we take step in opposite direction

$$w^{(t+1)} = w^{(t)} + \Delta w^{(t)} \tag{2.8}$$

$$\Delta w^{(t)} = -\alpha \frac{\partial L}{\partial w_i} \tag{2.9}$$

where $w^{(t+1)}$ is updated weight, $\Delta w^{(t)}$ is the rate of weight changing, α is the learning rate and it determines size of step, it used to be small number.

We want to reach the minimum of loss function by taking small steps in opposite direction of the gradient (Figure 2.5). The gradient is computed after every epoch (t). Epoch consists of one full training cycle on the data set. After every sample in the set is seen, new epoch starts $(t + 1)$.

In contrast stochastic gradient descent method (SGD) computes parameter updates via back-propagation after every sample from the dataset.

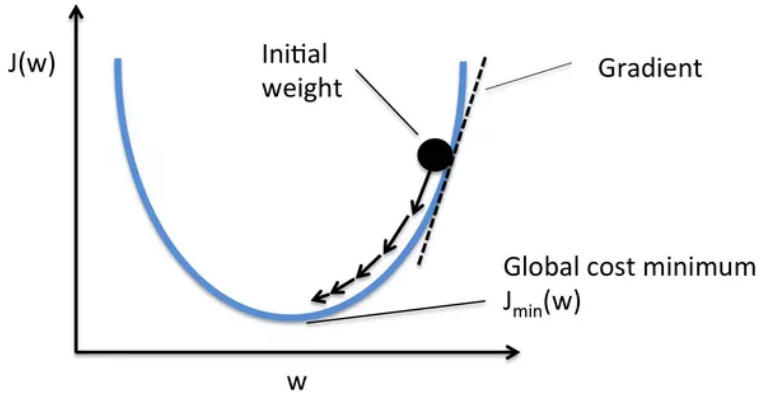


Figure 2.5: Illustration of reaching the minimum of the function by taking small steps in opposite direction of the gradient (Raschka, 2015).

Momentum is a method that accelerate SGD,

$$\Delta w^{(t)} = \gamma \Delta w^{(t-1)} - \alpha \frac{\partial L}{\partial w_i}, \tag{2.10}$$

where $\Delta w^{(t-1)}$ is weight update from previous epoch, γ is the momentum term, usually set up to 0.9. Update of parameter increase when the gradient point in the same direction, otherwise updates is reduced. As a result, we get faster convergence and reduced oscillation (Ruder, 2016).

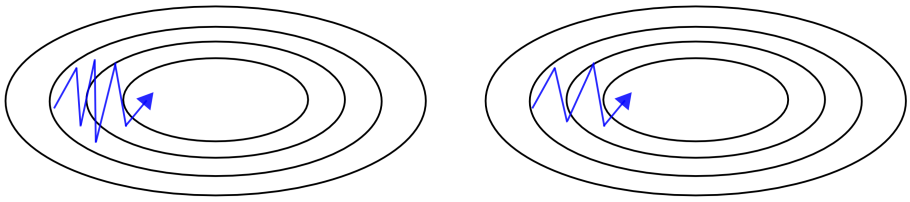


Figure 2.6: Left: SGD without momentum. Right: SGD with momentum (Ruder, 2016)

Adaptive Moment Estimation (adam), a method for efficient stochastic optimization. It was published by Coates et al. (2011). Adam is combination

of two optimization methods AdaGrad (Duchi et al., 2011) and RMSProp (Tieleman and Hinton, 2012). Weight update $w^{(t)}$ is computed as follows:

$$m^{(t)} = b_1 m^{(t-1)} + (1 - b_1) g^{(t)} \tag{2.11}$$

$$v^{(t)} = b_2 v^{(t-1)} + (1 - b_2)(g^{(t)})^2 \tag{2.12}$$

$$g^{(t)} = \frac{\partial L}{\partial w_i} \tag{2.13}$$

$$w^{(t)} = w^{(t-1)} - \frac{\alpha m^{(t)}}{\sqrt{v^{(t)}} + \epsilon} \tag{2.14}$$

where m is a first moment vector, v is second moment vector, b_1 and b_2 are exponential decay rates for the moment estimates, g_t is the gradient of the lost function, α is the learning rate and ϵ is a small constant used to avoid zero division. Indexes (t) and $(t-1)$ correspond to current and previous epoch respectively.

2.5 Back-propagation

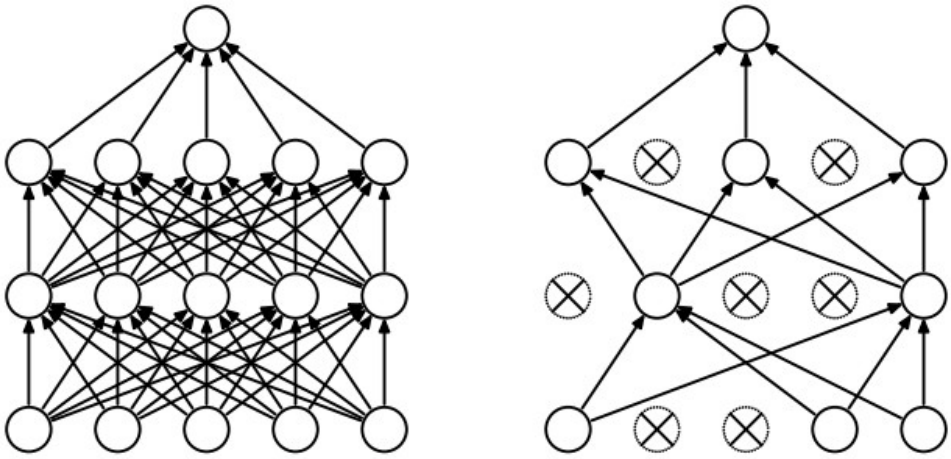
Back-propagation is the learning process, which propagates error through the network. It computes derivatives of a loss function and changes weights of parameters. Sets of weights in every layer are represents by matrices, but the last layer generates a vector. In forward pass, we derive the matrix first and then we multiply it with next matrix. That can be computationally expensive. The algorithm computes gradient in backward order – gradient of the output layer is calculated first. If we derive first output vector and multiply with a matrix in next layer it yields another vector. That is the reason why is the back-propagation computationally efficient approach.

Back-propagation can be summarized into three steps:

1. Output vector is computed.
2. Result is compared with true labels and loss is computed.
3. Weights are updated via back-propagation.

2.6 Dropout

Dropout is a technique to prevents the neural networks from an overfitting. The overfitting means that our model fits the training data well, but it fails when it comes to validation – our network memorizes the training data.
The idea behind dropout is easy, neural networks randomly drops units during training and they are not included in learning process. In other words, units are keeping active with some probability p, which is usually set up to $1/2$. This approach prevents a unit from co-adapting – it creates the unique network for every training case (Figure 2.7).



(a) Standard Neural Net (b) After applying dropout.

Figure 2.7: Illustration of neural network without dropout (a) and with dropout (b) (CS2).

2.7 Convolutional layer

Input for a classical fully-connected neural network is vector – so if our input is an image we have to unroll it into a feature vector, what causes a loss of spatial information. Convolutional networks (LeCun et al., 1989), also known as convolutional neural networks (CNNs), are a specialized kind of neural network for image processing (Figure 2.8). (Goodfellow et al., 2016)

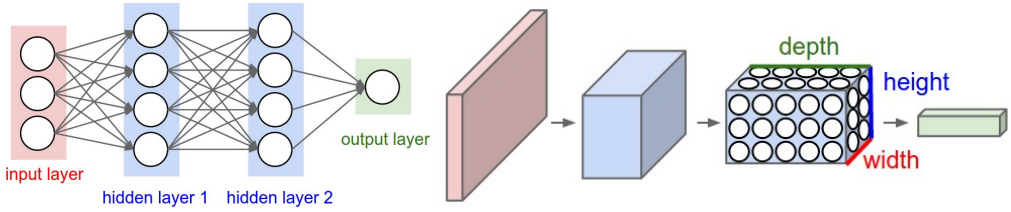


Figure 2.8: Left: fully connected neural network. Right: convolutional neural network (CS2)

We can look at convolution in two different contexts:

1. Mathematical
2. Convolutional network

Deeper description:

1. In the mathematical context, convolution is an operation on two functions of a real-valued argument and is defined for any function for which the below integral (2.16) is defined. Convolution is a specialized kind of linear operation. It is defined as:

$$G = F * H, \tag{2.15}$$

$$G(x') = \int_{-\infty}^{\infty} F(x)H(x' - x)dx. \tag{2.16}$$

2. If we take convolutional network point of view, F is referred to as an input and H as a kernel and output G is a features map. In machine learning application, we can rewrite equation 2.16 into:

$$S(i,j) = I * K(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-m), \tag{2.17}$$

where S in the output, I is the input, K is a two-dimensional kernel (filter) where i, j are over filter's high and width respectively, which are usually smaller than a width and high of the input (e.g. Figure 2.9).

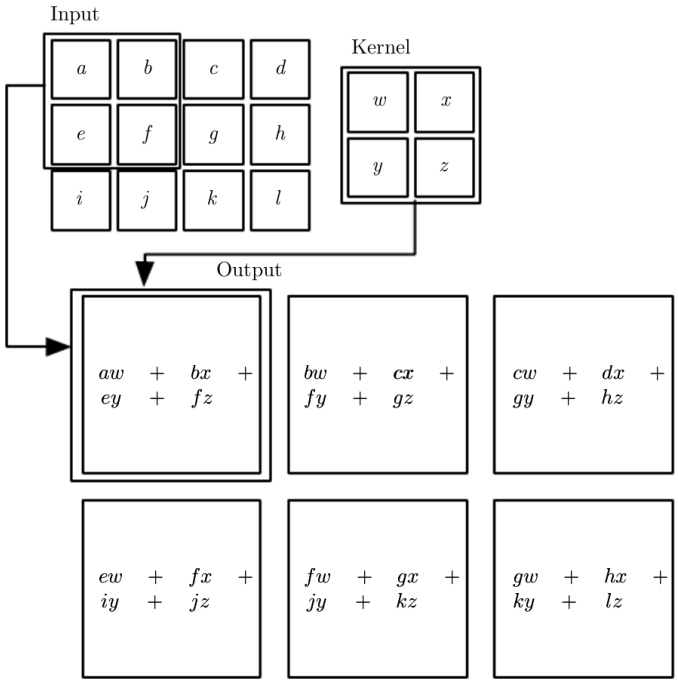


Figure 2.9: An example of 2-D convolution, with kernel (filter) size 2×2, stride 1 and without padding.

With the filter, we can detect meaningful features such as edges of objects at the image. The single filter can extract special kind of feature at many locations and it becomes active when it detects the feature. In practice we want CNN to learn to detect many features, which means that we have to use more filters.

There are more hyperparameters which is necessary to specify – receptive field, stride, and zero-padding.
The receptive field is equivalent to hight (h) and width (w) of the filter.
Stride defines step with which is filter moving along the image. When stride is 2, then filter jump 2 pixels at a time as we slide it around. This will produce smaller output volumes spatially than input.
Zero-padding adds zeros at the border of the spacial dimension of the image. This allows as to control the spatial size of the output (CS2).

Convoutional layer takes set of feature maps as the input (at Figure 2.11 input contains 2 feature maps). The depth of convolutional filter (CONV filter) is equal to the depth of the input volume D_1. Each neuron in the

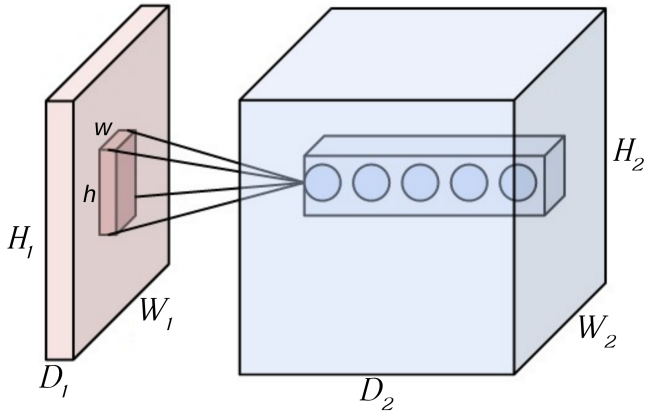


Figure 2.10: We have the input volume of size $W_1 \times H_1 \times D_1$ and we want to apply 5 different CONV filters with receptive field $h \times w$, CONV filter has size $h \times w \times D_1$ - it consists D_1 sets of filters with size $h \times w$. This creates output volume, convolutional layer, with size $W_2 \times H_2 \times D_2$ and $D_2 = 5$ (CS2).

convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (Figure 2.10). After CONV filters are applied we get final output – convolutional layer with another set of feature maps. The number of output's feature maps depends on how many CONV filters did we use.

The output shape of the convolutional layer is affected by the shape of its input as well as the choice of receptive field, strides, and zero-padding. This property of CNN is in contrast to the fully-connected network whose output size is independent of the input size (Dumoulin and Visin, 2016).

In practice, if our input is volume $W_1 = 5$ $H_1 = 5$ $D_1 = 2$ and we want to have two CONV filters (K) with receptive field 3×3 (F) , stride of $S = 2$ and zero-padding $P = 1$. We can compute output volume as:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1, \tag{2.18}$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1, \tag{2.19}$$

$$D_2 = K, \tag{2.20}$$

then the result is a volume with size $3 \times 3 \times 2$ (Figure 2.11).

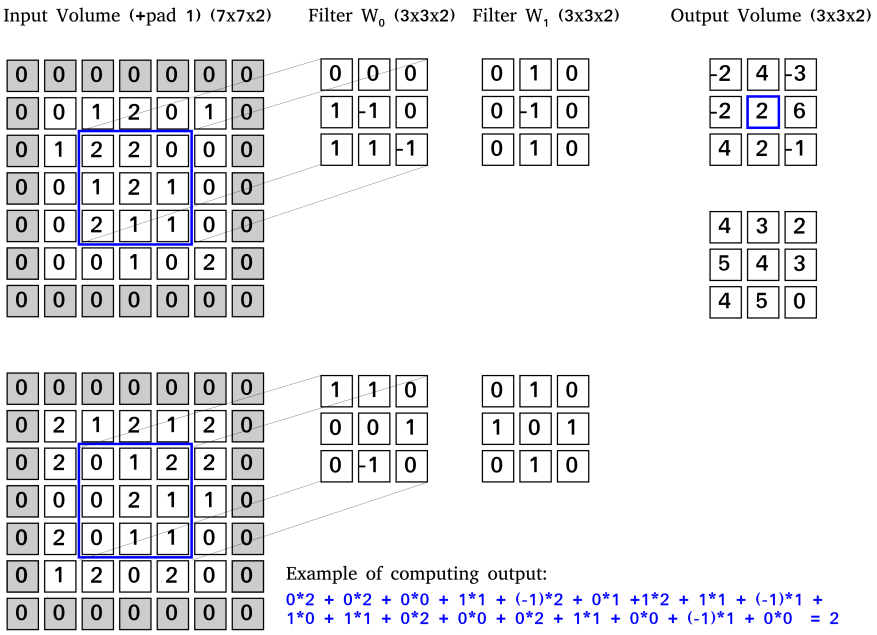


Figure 2.11: Example of the convolutional network with input $5 \times 5 \times 2$ plus 1 padding, 2 sets of CONV filters and output $3 \times 3 \times 2$. Both input and output has two feature maps. At each location, the product between each element of the filter and the input element it overlaps is computed and the results are summed up to obtain the output in the current location. The procedure can be repeated using different CONV filters (in our case 2 filters) to form as many output feature maps as desired (Dumoulin and Visin, 2016).

2.8 Deconvolutional layer

The deconvolution layer or transposed layer is the inverse of the convolution layer. The deconvolutional forward pass is calculated just as is the backward pass of a convolutional layer (Gauthier, 2015). Transposed layer is used to increase the spatial dimension of input (Figure 2.12). Imagine that we

have convolution with given input shape. We can compute output shape by equation (2.18) – (2.20). If the shape of the input layer of transpose convolution is the same as the output shape of convolution, the output of transpose convolution will be the same as input of convolution.

Let's consider that transpose of non-padded convolution is the same as convolving a zero-padded input, then the transpose of zero-padded convolution is equivalent to convolving an input padded with less zeros. If stride of convolution is bigger than one, then zeros are inserted between input units, which makes the filter move slower (Figure 2.13). For more details about transpose convolution read Dumoulin and Visin (2016).

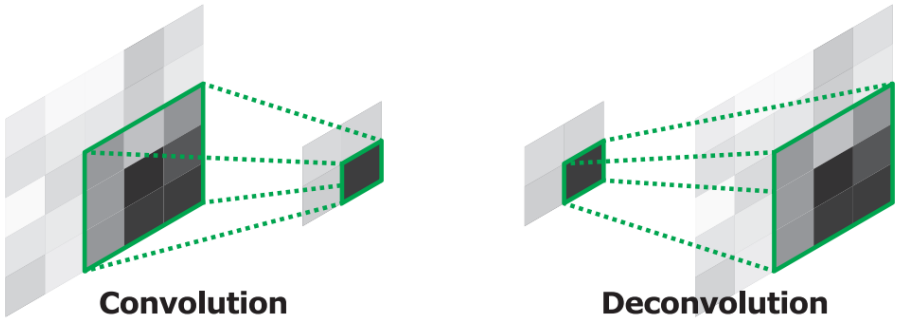


Figure 2.12: Illustration of convolution and deconvolution (Noh et al., 2015).

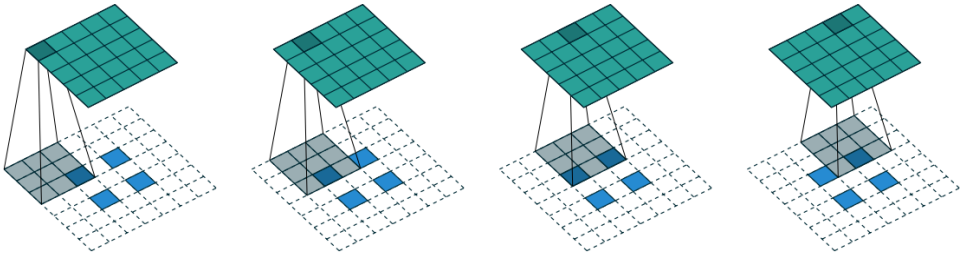


Figure 2.13: The transpose of convolving a 3x3 filter over a 5x5 input using stride two without padding is equivalent to convolving a 3x3 filter over 2x2 input with zero inserted between inputs, padded with 2x2 border of zeros using stride one (Dumoulin and Visin, 2016).

Generative adversarial network

3.1 Introduction

The main goal of the generative networks is to generate new samples follows the same probabilistic distribution of a given a training dataset. There are many kinds of generative models:

- Boltzmann Machines (BMs)
- Deep Belief Networks (DBNs)
- Variational Autoencoders (VAEs)
- Generative Adversarial Networks (GANs) etc.

Readers interested in can learn more about different generative models in Deep Learning book (Goodfellow et al., 2016).

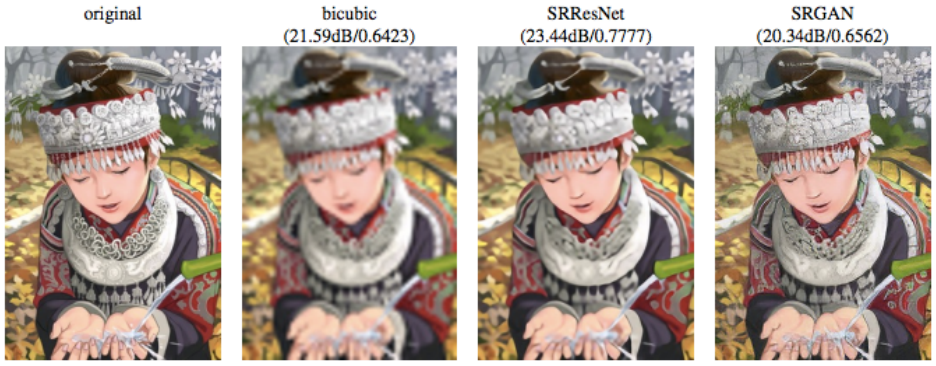


Figure 3.1: Illustration of how different types of generative networks deal with photo resolution. From left to right:original HR image, bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception (Ledig et al., 2016).

There are differences in how these models works but the basic idea is common. Lets the dataset $x_1, ..., x_n$ are samples from a true data distribution $p(x)$ (Figure 3.2). Our model takes data points from some distribution (in this case Gaussian distribution) and maps them through the neural network to the generated distribution $\hat{p}_\theta(x)$. The network is a function with parameters θ and by the changing, those parameters network tries to

produce generated distribution that closely matches the true data distribution – network learns to represent an estimate true data distribution.Applications for generative models are image denoising, inpainting, super-resolution, structured prediction, etc.

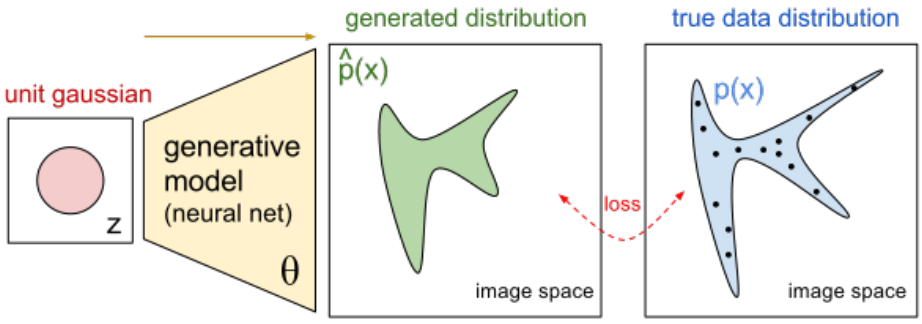


Figure 3.2: Illustration of generative model (GAN).

3.2 Generative Adversarial networks

"Generative Adversarial Networks is the most interesting idea in the last ten years in Machine Learning."

— Yann LeCun, the director of Facebook AI

Generative adversarial network (GAN), were introduced in 2014 by J. Goodfellow (Goodfellow et al., 2014). GAN consist of two different models – generator (G) and discriminator (D). The generator creates samples that capture data distribution. The discriminator estimates the probability that sample is real – came from training data or fake – created by generator. GAN's training can be compared to the game where G tries the best to cheat the D by generating more realistic images and D tries the best to distinguish whether the image is real or fake. At convergence, the G's samples are indistinguishable from real data, and the discriminator outputs $1/2$ everywhere.

Each of player is represented by a differentiable function with respect to it's input. D and G play the following two-player minimax game with value (loss) function $V(G, D)$:

$$\min_{G} \max_{D} V(D,G) = E_{\mathbf{x}\sim p_{data(\mathbf{x})}} [\log D(\mathbf{x})] + E_{\mathbf{z}\sim p_{z(\mathbf{z})}} [\log(1 - D(G(\mathbf{z})))], \quad (3.1)$$

where $D(x)$ represent probability that $\mathbf{x}$ comes from real data and $G(\mathbf{z})$ are data from generator's distribution p_g. At the beginning of learning, D reject samples with high confidence and $\log[1 - D(G(z))]$ saturates. Because of that, it is better to train G to maximize $\log D(G(z))$.

We can divide training of GAN into two phases (Figure 3.3):

- Phase I. – training of discriminator:
  Input noise (or image with noise) flows through G and output gets label *False* or 0. After that we drawn the image from real data and it gets label *True* or 1. Then D gets this images as input and it tries to distinguish whether the images are real or fake. Afterwards loss function (L_D) is computed and parameters of discriminator θ_D are updated. Parameters of G are not updated in this phase.

- Phase II. – training of generator:
  In this case, the output from G gets label *True*, pass through D and loss function L_G is computed and parameters of generator θ_G are updated. Note, that L_G is same as L_D, but it can by extended by another loss function. Parameters of D are not updated in this phase.

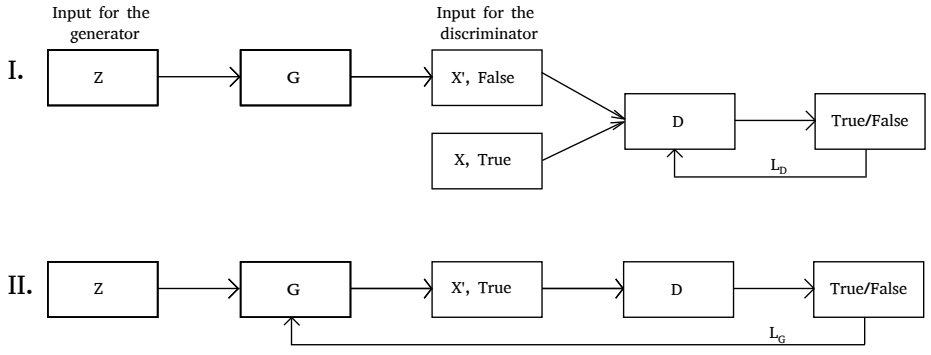


Figure 3.3: Scheme of GAN's training.

3.3 Discriminator network

Discriminator is a convolutional neural network (CNN), which contains several convolutional layers and the output is a single scalar, the probability of the image being real or fake. After every convolutional layer follows activation function, leaky ReLU. Batch normalization (BN) is a layer that has a mean output activation of zero and standard deviation of one. The first layer of the discriminator is not batch normalized so that the model can learn the correct mean and scale of the data distribution. We will call sequence of convolutional layer, activation function and batch normalization block.
Dense or fully connected layer have full connections to all neurons in the previous layer, as in regular neural network (Figure 2.3). An input of dense layer is a vector and output of convolutional layer is not – it means that we have to insert a special layer between them which flattened output of convolutional layer into the vector.When training is done, we don't need discriminator anymore and we can discard it.

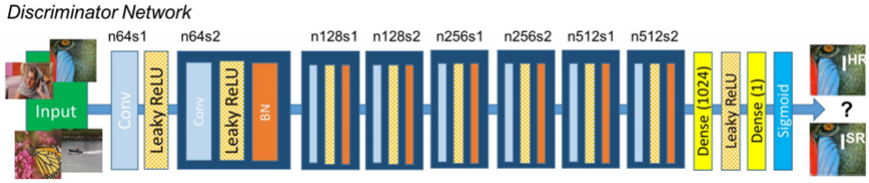


Figure 3.4: Exampe of architecture of discriminator network with corresponding number of feature maps (n), stride (s) and seven blocks.

3.4 Generator network

Architecture of generator's network is more complicated than discriminator's. We can split up generator into two parts, first is made of convolutional layers and second deconvolutional or transpose layers. In both parts batch normalization is used. Input is an image with noise which have same size as output.
First part of generator look like CNN, with batch normalization and leaky ReLU activation. At second part after activation function is applied, deconvolution is made. Before we use another activation function, we concatenate output of deconvolution layer with output of penultimate block from first

part. See Figure 3.5 for better understanding. Batch normalization is not used in last layer of generator.

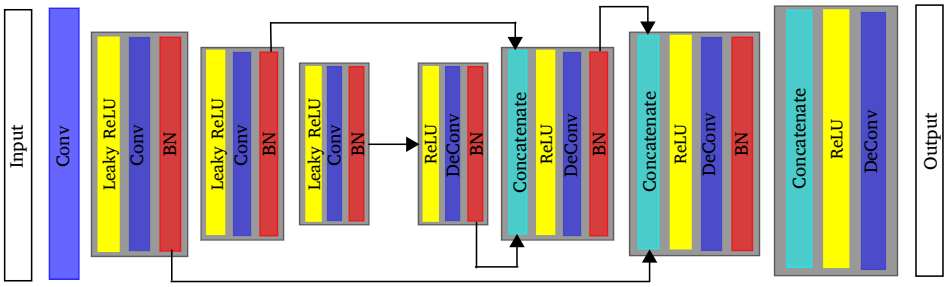


Figure 3.5: Exampe of architecture of generator network with three blocks.

3.5 Conditional GAN

Mehdi Mirza and Simon Osindero in 2014 (Mirza and Osindero, 2014) introduce a new type of GAN – conditional GAN (CGAN). The difference is that CGANs are extended to a conditional model by adding some extra information, condition $\mathbf{y}$, to both the generator and discriminator. The loss function (3.1) of a two-player minimax game would change as:

$$\min_{G} \max_{D} V(D,G) = E_{\mathbf{x}\sim p_{data(\mathbf{x})}} \log D(\mathbf{x},\mathbf{y}) + E_{\mathbf{z}\sim p_{z(\mathbf{z})}} \log[1 - D(G(\mathbf{z},\mathbf{y}))]. \quad (3.2)$$

In our case, we use additional condition when loss function of a generator is calculated. Input for the generator is a blurred image with noise. We use the original image without noise as a condition and we can calculate the MAE and add it to the loss function L_G. Loss of generator L_G is no longer affected just by discriminatory ability to differ fake and real image.

Application of network

4.1 Tests of generator

At first, we were interested in how neural network works in use. There is no better way to answer this question than play with an architecture of the neural network and change the dataset. We tried different architecture of our generator and summarized results in the section below answer our questions. Our first set of the test was made with the STL-10 dataset (Coates et al., 2011). This dataset consists of different type of pictures with ten different classes with 12500 images. Size of images is $96 \times 96 \times 3$.

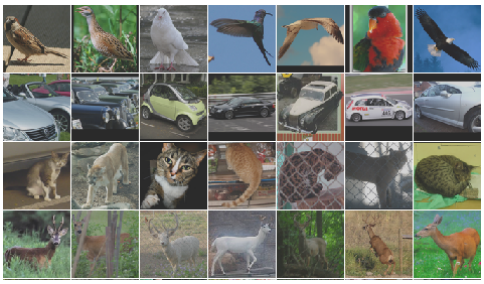


Figure 4.1: The STL-10 dataset examples

Our dataset contains both, the real image also blurred image (Figure 4.2). Because of that, we don't need to use discriminator – our lost function is MAE (Section 2.3). We will compare the results with graphs of change of loss function depending on the number of the epoch. Whoever was reading it here, I invited him to beer.

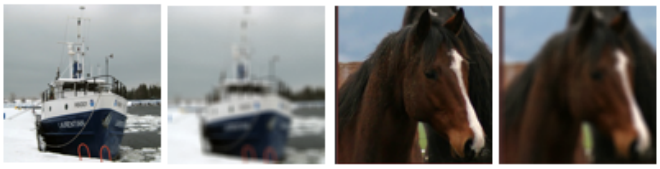


Figure 4.2: Examples of random images from the STL-10 dataset with and without noise.

- Test I

  In our first test, we want to determine how the change of size of the dataset affect training. Size of dataset was 5000 images (N1) and 8500 images (N2). The architecture of generator is as follows: 3 convolutional layers with batch normalization and leaky ReLU activation, and three deconvolutional layers without batch normalization and ReLU activation. The number of the epoch is different, we trained N1 for 50 epoch and 35 epoch for N2. At Figure 4.3 we can see that with the bigger dataset is loss smaller, which is no surprise. We conclude that with bigger dataset we will obtain better results.

- Test II.

  In this test, we examine the influence of batch normalization in the network. We will use network N2 from the test above and network with batch normalization after first and deconvolution layer (N3) and network with batch normalization after every deconvolution layer (N4). Size of the dataset is 8500 images. As we can see networks which contain batch normalization have a smaller loss than the network without it. From a comparison of N3 and N4 we see that match of validation and train loss of N4 is not as good as of N3, so we conclude that it is preferable if the network doesn't contain batch normalization after last deconvolution layer.

- Test III.

  Dropout helps to prevent neural network from overfitting. We use same architecture as N3 plus add dropout after first and second deconvolutional layer (N5) and compare it with N3. Note that a slight decrease in loss is observed, which leads to the conclusion that the use of the dropout causes better results.

- Test IV.

  In the last test, we added extra convolution and deconvolution layer into N5 and we kept the number of batch normalizations and dropouts (N6). We trained N6 at the dataset with 12500 images and compare it with N5 trained at 8500 images. We can see that decrease of lost of N6 is significantly bigger in comparison with N5.

Conclusion: Our experiments proof that bigger dataset and bigger network leads to better results. Adding batch normalization and dropout into the network undeniably improves work of the generator.

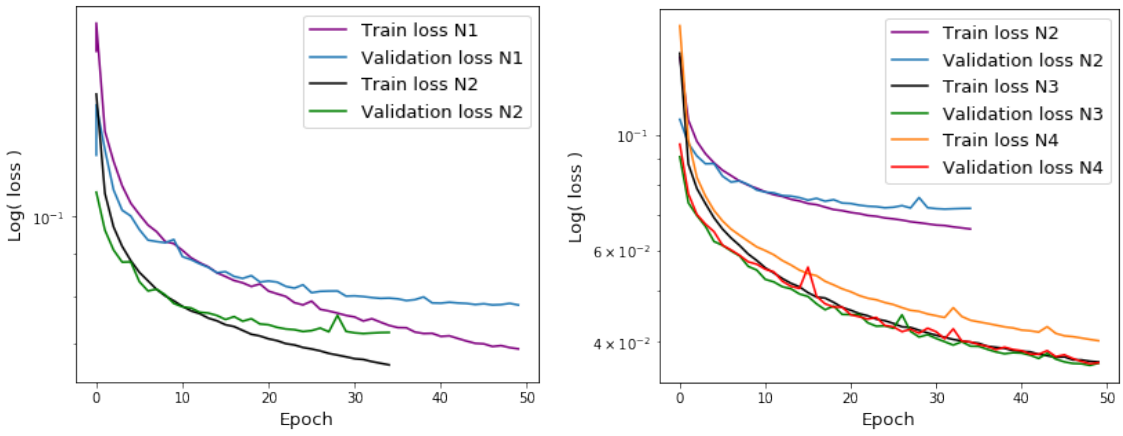


Figure 4.3: Left: Test I. Loss of networks N1 and N2. Right: Test II. Loss of networks N2, N3 and N4. Big difference between validation and train loss of N4 network suggest that network suffers from overfitting.

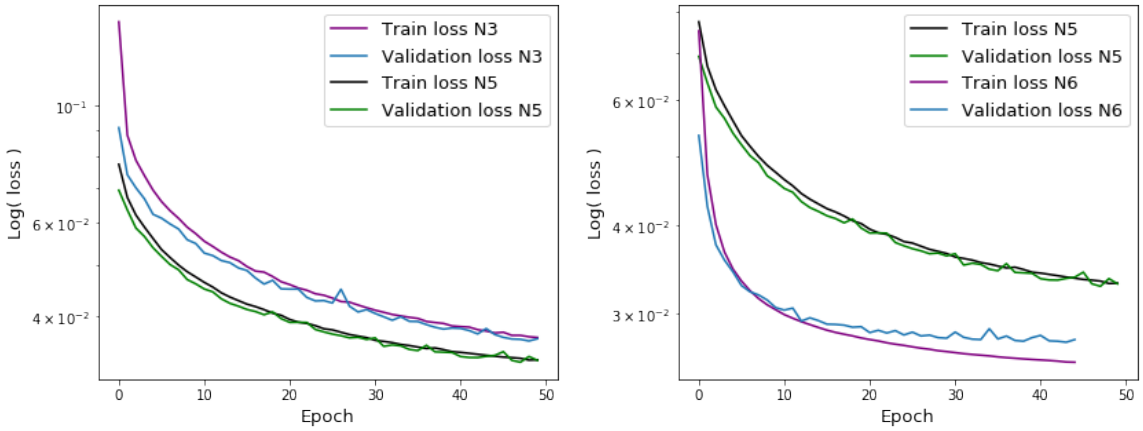


Figure 4.4: Left: Test III. Loss of networks N3 and N5. Right: Test IV. Loss of networks N5 and N6. Network N6 has additional layer to help improve the performance.

4.2 Image processing

Our dataset holds 4550 galaxies from the Sloan Digital Sky Survey (SDSS York et al. (2000)). Before we use it to train the network we have to process the fits and set them in range (0,1). According to Lupton et al. (1999) we use the inverse hyperbolic sine (arcsinh) to create RGB images. One of the advantage of arcsinh is that for fluxes near to zero is transformation linear and we will not loose the information as it was with logarithmic function which were commonly use for image processing. This dataset serves as a label to compute loss function for the generator and as training samples for the discriminator. The code is available online[1]. Subsequently, we add noise to images and create a new dataset, training samples for the generator.

4.2.1 Degraded image

Blurring
When neglecting other disturbing phenomena, an astronomical point source viewed by primary telescope mirror would be an Airy diffraction disk. Size of the disk would be established by mirror diameter and wavelength. We can describe this effect with point spread function (PSF), with spread parameter the full width at half maximum intensity ($FWHM$). The image is, therefore, a convolution of true light distribution with PSF.

Observation of night sky from the ground is affected by the turbulent atmosphere which causes seeing. Seeing is one of the reasons, why point source (like a star) is spread over multiple pixels in an image. Combination of diffraction and seeing resulting into fuzzy disk. It is possible to approximate PSF with Gaussian curve (Figure 4.6). A disadvantage of the approximation is that Gaussian curve does not contains "wings" of PSF profile. To generate blurred image we convolve the images with Gaussian filter, which represent seeing.

Noise
Noise produces undesirable effects and its origin is different, in astronomy we encounter sky noise, noise caused by dark current, read-out noise,photon noise, etc.. Arriving of photons, which are detected by CCD, is random process and distribution of their arrival is theoretically described as Poisson distribution (PD). Probability of detection x photons per unit interval is:

$$P_x = \frac{m^x \mathrm{e}^{-m}}{x!}, \tag{4.1}$$

[1] `https://is.muni.cz/auth/th/tcbgi/Model_Bakalarka.py?studium=727887`

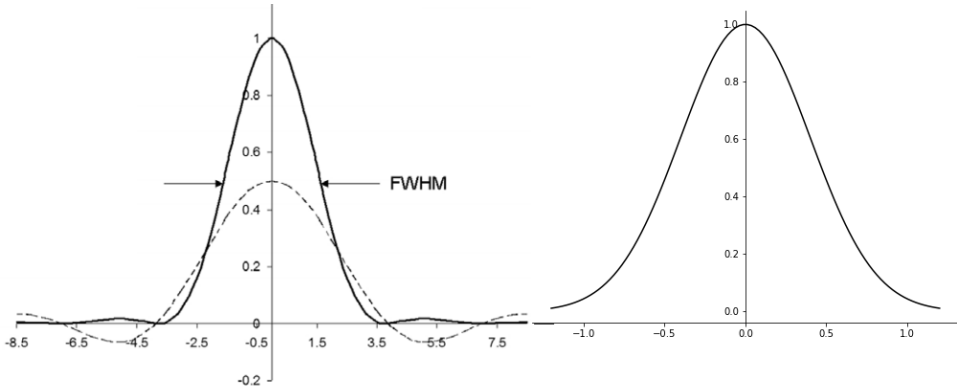


Figure 4.5: Left: point spread function (solid line) of Airy diffraction pattern. Right: Gaussian curve (McLean, 2009)

where m is the mean value of photon detection per time interval. If value of $m \gg 0$, we can approximate Poisson distribution with normal (Gaussian) distribution with variance $\sigma^2 = m$ (Brandt, 1976). The noise associated with photons detection will also have Poisson distribution. We trained our network on two different type of noise, Gaussian and Poisson. Another difference was how to create noise. Gaussian noise was added after arcsinh was applied. The parameter of distribution, m, is a mean intensity of pixels at the original image. Poisson noise was added before images were processes. This creates noise, which corresponds to real astronomical noise distribution. We tried both types of noise because we were interested in how the network will respond. Results of images with Gaussian noise are in Section 5.2 and images with Poisson noise in Section 5.3 (McLean (2009), Boyat and Joshi (2015)).

4.3 Image evaluation

To evaluate results of the network we used mathematical approach and subjective evaluation method (human opinion). For objective evaluation, we have chosen peak signal-to-noise ratio and an observational error.

Peak signal-to-noise ratio
Peak signal-to-noise ratio (Ψ) is a method to quantify differences between original image and reconstructed (denoise) image. Mathematically, the Ψ is

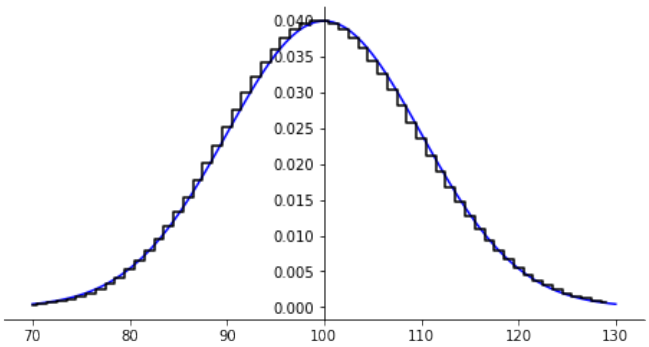


Figure 4.6: Approximation of Poisson distribution (black) with normal distribution (blue) with $m = 100$ and $\sigma^2 = m$.

defined via variance:

$$\text{var}(I - I') = \frac{1}{N} \sum_{h,w,d}^{H,W,D} (I_{h,w,d} - I'_{h,w,d})^2, \tag{4.2}$$

where I and I' are original and reconstructed images and H, W, D are hight, width and depth and N is dimensionality of the image. Then, Ψ expressed in logarithmic decibel scale is:

$$\Psi = 10 \log \left(\frac{I_0^2}{\text{var}(I - I')} \right). \tag{4.3}$$

Here, I_0^2 means the highest possible value of the signal. If original and reconstructed image are almost identical, the variance is approaching zero and Ψ goes to infinity. Therefore, the good result for the network is high Ψ.

Relative error
Second metrics for evaluation of result is a relative error. Relative error is a ratio of variation and a real value (real image). Mathematically,

$$\delta = \frac{\text{var}(I - I')}{I^2}. \tag{4.4}$$

Error $\delta = 0$ indicates, that images are identical. (Mastriani (2016), Sadykova and Pappachen James (2018), Fardo et al. (2016))

Structural Similarity index

Structural Similarity index (κ) is distance metric for two images that takes into account spatial correlations between different pixels (**?**). The κ of two images that are the same is 1 and it decreases as one of the two images is degraded. Structural similarity index formula is based on three comparison measurements between two images x and y: luminance (l), contrast (c) and structure (s). It is defined as:

$$\kappa(x,y) = (l(x,y)^{\alpha} c(x,y)^{\beta} s(x,y)^{\gamma}) \tag{4.5}$$

$$l(x,y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \tag{4.6}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \tag{4.7}$$

$$s(x,y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3}, \tag{4.8}$$

where μ_x, μ_y are the average of x and y, σ_x^2 and σ_y^2 is the variance of x, y, σ_{xy} is the covariance of x and y, c_1, c_2, c_3 are constants, $c_1 = 0.01R$, $c_2 = 0.03R$, $c_3 = {}^{c_2}/_2$ and R is the data range. We set α, β and γ to one.

Level of noise	0.	I.	II.	III.	IV.	V.
Ψ [dB]	∞	18.6	16.4	14.6	13.1	12.4
δ	0	0.15	0.28	0.37	0.45	0.53
κ	1	0.30	0.17	0.10	0.07	0.06

Table 4.1: Use of peak signal-to-noise ratio (Ψ), relative error (δ) and Structural Similarity index (κ) to compare quality between different noisy and original images. If we use Ψ at identical images, the result is infinity as expected. Relative error increase with higher noise level. For similar images is $\kappa = 1$ and it decrease as we add more noise into image. Figures 4.8 and 4.9 contains more data for images with different noise level. Level of noise is changing by linearly changing parameters of Gaussian distribution – mean and sigma.

Figure 4.7: Image with Poisson noise. $\Psi = 23.3$dB for center image region.

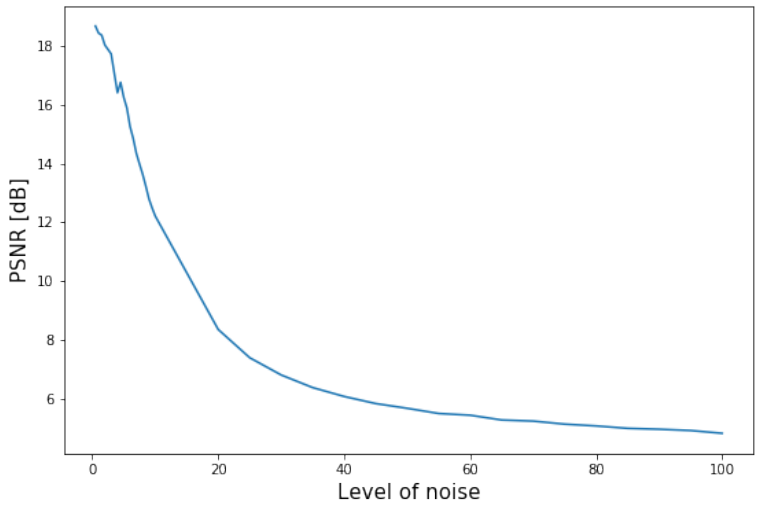


Figure 4.8: Dependence of the peak signal-to-noise ratio (PSNR) on the level of noise. It is obvious that Ψ decreases with the higher noise level. Level of noise is changing by variating parameters of Gaussian distribution – mean and sigma.

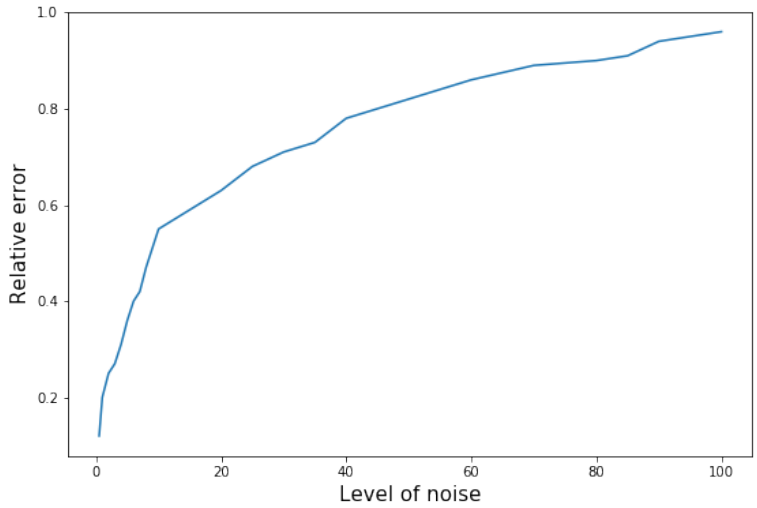


Figure 4.9: Dependence of the relative error on the level of noise. Level of noise is changing by variating parameters of Gaussian distribution – mean and sigma.

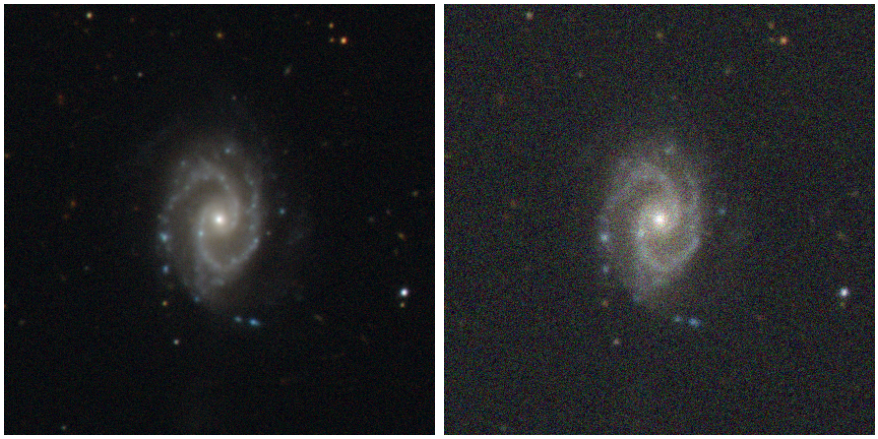


Figure 4.10: Left: the original image. Right: noisy image I. See Table 4.1 for information about value of Ψ and relative error.

Figure 4.11: Images with different contribution level of Gaussian noise. Top: left – noisy image II.; right – noisy image III.. Bottom: left – noisy image IV.; right – noisy image V. See Table 4.1 for information about value of Ψ and relative error. We recommend to exterminate images in pdf form of our work.

4.4 Network architecture

For more information about the network see Section 3.
Generator
The architecture of the generator network is separable into two parts. First, convolution part, consist of convolution blocks (C-block). Second, deconvolution part is made of deconvolution blocks (D-block). The inner structure of blocks is in the Figure 4.12. After every D-block is its output information

concatenates with output information of C-block with same the spatial size (Figure 3.5). Our generator is made of five C and D blocks. Detailed architecture is displayed in Figure 4.12.

Discriminator

Discriminator is also made of C-blocks without batch normalization and few other layers are used (details in section 3.3). We can see whole architecture in Figure 4.13.

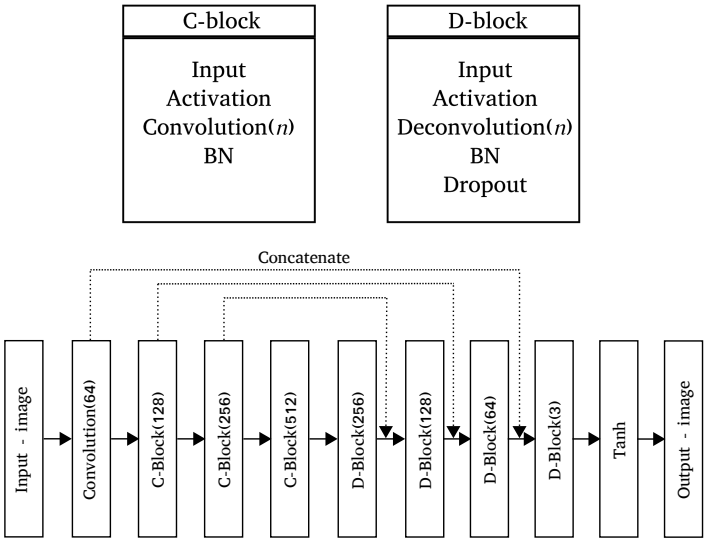


Figure 4.12: The architecture of the generator. A number next to C and D block is a count of filters, and D in D-blocks signify the use of dropout.

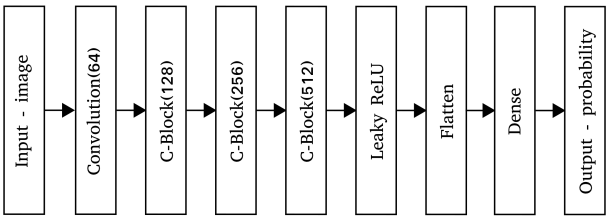


Figure 4.13: The architecture of the discriminator.

Review of results and discussion

This section contains review of our results, their examination and comparison with technique which is nowadays available.

5.1 Total-variation chambolle

One of the available technique for direct image restoration is a total-variation chambolle denoise (TV-chambolle). Rudin et al. (1992) first introduced the total variation in Computer Vision as a regularizing criterion for solving inverse problem and implementation of used code was proposed by Chambolle (Chambolle, 2004). Total variation denoising algorithm tend to minimize the total variation of the image. Benefits of TV-chambolle algorithm is that you do not need any dataset, single image is required, and it is not as computationally costly as to trains neural networks. We will compare results of this method with our network to see, which is more efficient.

5.2 Stage I – the first network application

Because of computing efficiency we trained our network with smaller images from our SDSS dataset. Size of training images was $96 \times 96 \times 3$ pixels. We trained on NVIDIA GeForce GTX 970 graphics. In Stage I. we add Gaussian noise into images.

During the first 180 epochs, we gradually changed the weight of the losses of the generator, we were increasing the weight of the MAE. For about 200 epochs, we trained the network of 0.9 for MAE and 0.1 for binary-cross entropy. We finished the training, we saw that the discriminator is no longer able to distinguish between the original and the generated image. Training of the network took about forty hours. We save the weights and perform phase I. tests. Afterwards, we trained another 200 epochs without discriminator network. This test lasted twenty hours and we perform phase II. tests after training.

Reconstruction of 400 images with TV-chambolle denoise took approximately four seconds, with generator after phase I. and after phase II. around twelve seconds. As we see the computational efficiency of TV-chambolle is better than generator's efficiency. The Figure 5.1 below shows reconstructed images. Peak signal-to-noise ratio information are available in the Table 5.1. We can see, that TV-chambolle denoise is not able to fully reconstruct degraded images. In the contrast, generated images shows ability to reconstruct point objects as stars and galaxy center, even finer structures are visible. But the generator is not able to reconstruct all the images so well. As we can see in the Figure 5.2, the network fails to reconstruct the image after

both phase I. and phase II. After phase I. we can see structure around center of galaxy and it became more visible pass phase II. This can be attributed to the fact that the training dataset is small and unable to capture all the details. In the Figure 5.3 we see that the network missed structure at the edges of the galaxy but it managed to reconstruct central part of the galaxy better.

Images at Figures 5.4 and 5.6 are from SDSS and have unusual shape. Hoag's object is a non-typical galaxy, its shape can be consider as the result of the collision of multiple galaxies. As we can see, the network was not able to reconstruct fine ring. Mayall's object is also result of collision of galaxies. Even when the network miss structure, as we see at residuals, it denoise image better than TV-chambolle denoise did.

To study noise distribution of residual images from Figure 5.2, 5.3 and 5.6, we uses Kolmogorov–Smirnov test (K-S test), which determine if residuals have Gaussian distribution or not. Probability of chosen distribution base on p-value (λ), if $\lambda > 5\%$ than we can assume that residuals has Gaussian distribution. K-S test was computed for every row and channel of image separately and we plotted histograms (Figure 5.8) to see how value of λ differs (Table 5.3 and 5.5). Results of test for Gaussian noise with different parameters shows that $\bar{\lambda} \sim 50\%$ and percentage of values $\lambda > 5\%$ is around five percent (Figure 5.7). The conclusion is that residuals D do not have a Gaussian distribution, but the E residuals are closer to it.

Conclusion
Results of this test shows us, that the network is able to learn to reconstruct images, but it has certain problems with structures details as we can see on the residuals images. This problem could be caused by inappropriate dataset or small architecture of the generator. With small dataset, outliers become more-likely to not to be reconstructed good. To measure how our network denoise the images we use different technique – peak signal-to-noise ratio, structural similarity index and Kolmogorov–Smirnov test. Every of them is focused on different aspects, it is good to combine them. It is also important to examine the images and the residuals by humans. We decided to run another test with small changes as we are planning to pre-train the generator on generated data and we will add rotated images into dataset.

Ψ[dB]	Noisy image	TV-chambolle	Phase I.	Phase II.
A	13.6	14.6	24.3	25.1
B	13.1	14.1	24.2	25.7
C	19.2	20.26	33.2	33.1
D	18.2	19.3	32.8	26.9
E	12.9	13.7	24.2	24.1
F	16.8	17.4	32.4	31.8
G	13.9	14.9	26.6	27.3

Table 5.1: Peak signal-to-noise ratio Ψ for different images, as we can see Ψ is increasing if we compare noisy image and image after phase I. and II. and is always higher than Ψ of image after TV-chambolle which suggests that our network denoise image better than TV-chambolle. Images A – C are in Figure 5.1, image D in Figure 5.2, image E in Figure 5.3. Images F (Hoag's object) and G (Mayall's object) are in Figure 5.4.

κ	Noisy image	TV-chambolle	Phase I.	Phase II.
A	0.44	0.71	0.81	0.81
B	0.54	0.78	0.87	0.87
C	0.50	0.81	0.91	0.91
D	0.53	0.87	0.91	0.85
E	0.43	0.69	0.80	0.80
F	0.34	0.74	0.68	0.72
G	0.42	0.72	0.77	0.65

Table 5.2: Structural similarity index κ for different images, as we can see κ is almost one for images after phase I. and II.– this means that they are close to original images (for similar images $\kappa = 1$). Images A – C are in Figure 5.1, image D in Figure 5.2, image E in Figure 5.3. Images F (Hoag's object) and G (Mayall's object) are in Figure 5.4.

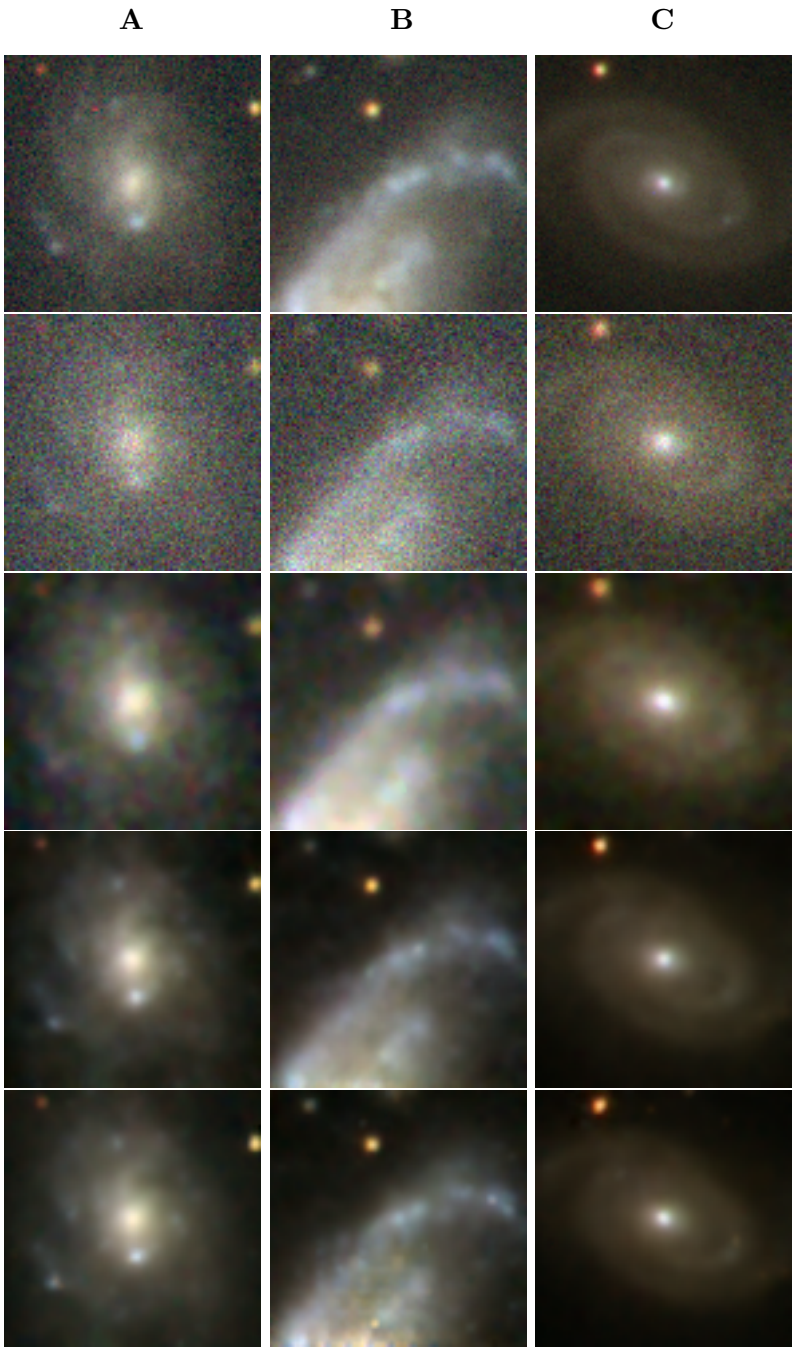


Figure 5.1: From above: Original images, noisy images, TV-chambolle denoise, generated after phase I., generated after phase II.

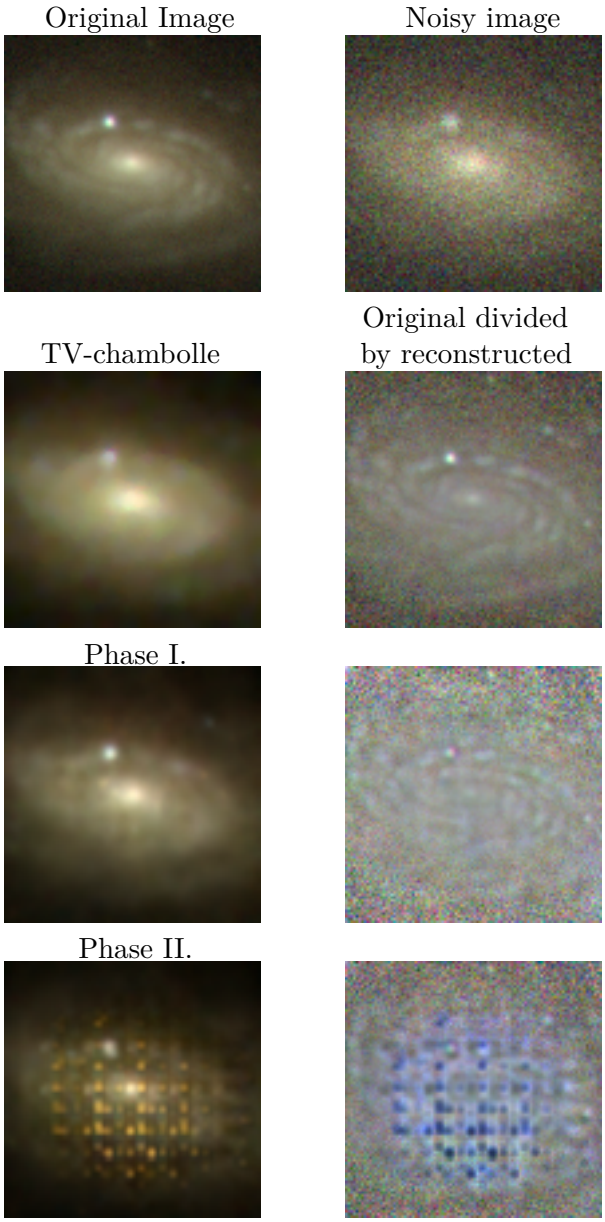


Figure 5.2: (D) From above: Original images, noisy images, TV-chambolle denoise, generated after phase I., generated after phase II.

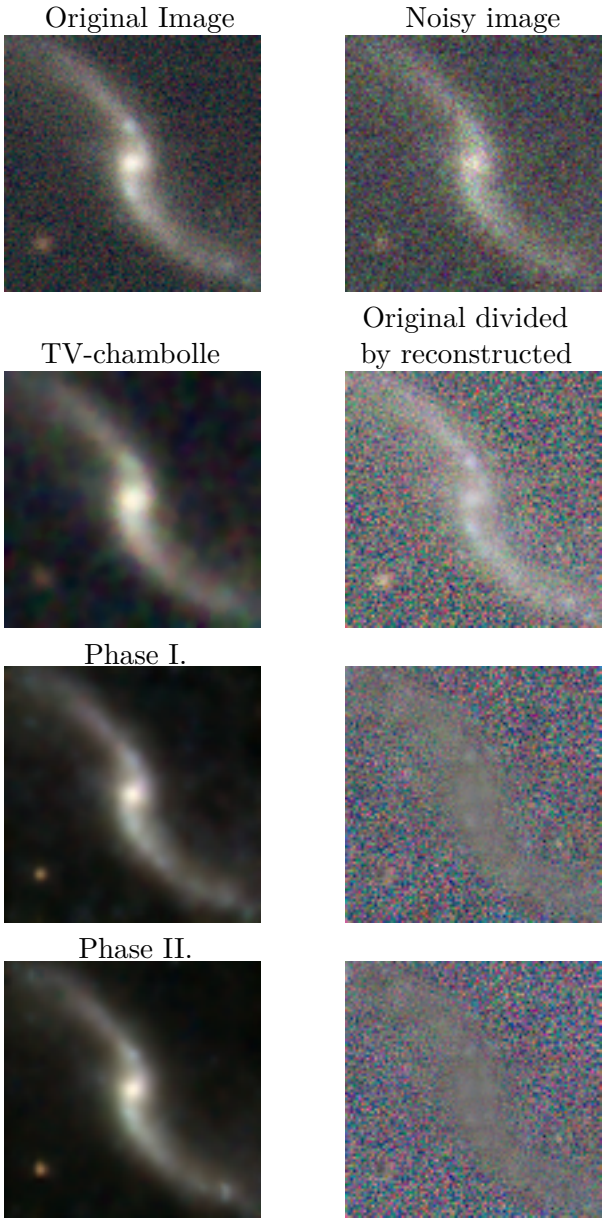


Figure 5.3: (E) From above: Original images, noisy images, TV-chambolle denoise, generated after phase I., generated after phase II.

Figure 5.4: From above: Original images, noisy images, TV-chambolle denoise, generated after phase I., generated after phase II.

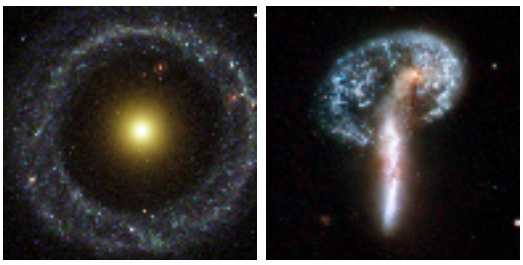


Figure 5.5: Downscaled images of Hoag's and Mayall's object from Hubble space telescope. Nevertheless we see much more detail in the images. This gives us goal to try to improve our network to make better job.

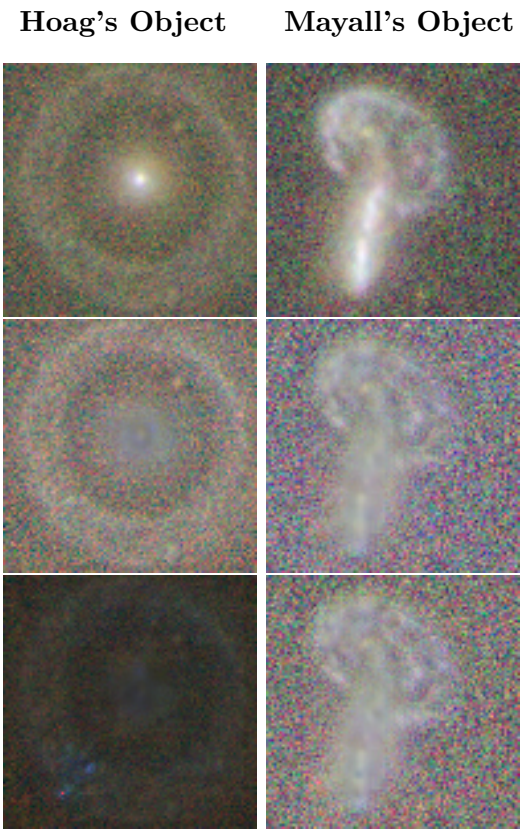


Figure 5.6: From above: residuals of images after TV-chambolle denoise, phase I., phase II.

G	TV-chambolle	Phase I.	Phase II.
$\bar{\lambda}$ [%]	32.0	34.2	34.3
$\lambda < 0.05$ [%]	20.8	29.0	22.0
F	TV-chambolle	Phase I.	Phase II.
$\bar{\lambda}$ [%]	25.4	30.3	31.1
$\lambda < 0.05$ [%]	27.5	25.8	24.2

Table 5.3: The analysis of residuals centre of G and F image. K-S test shows that there are more values $\lambda > 0.05$, so we can expect that noise has the Gaussian distribution, but it is necessary to improve our results. It is good sign is, that central parts of Mayas's object residuals from the network has less values $\lambda < 0.05$ than TV-chambolle denoise. That means that our network makes better work.

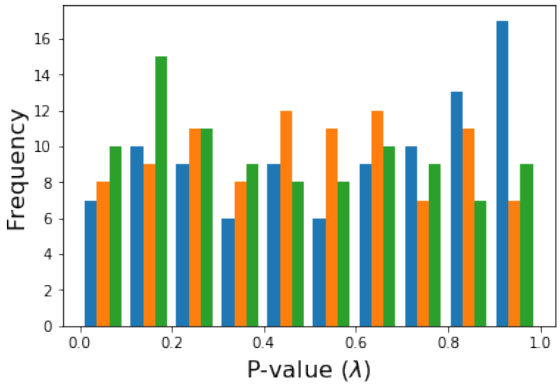


Figure 5.7: Histogram of λ values for Gaussian noise, $\bar{\lambda} = 50.1\%$ and 5% of values are $\lambda < 0.05$

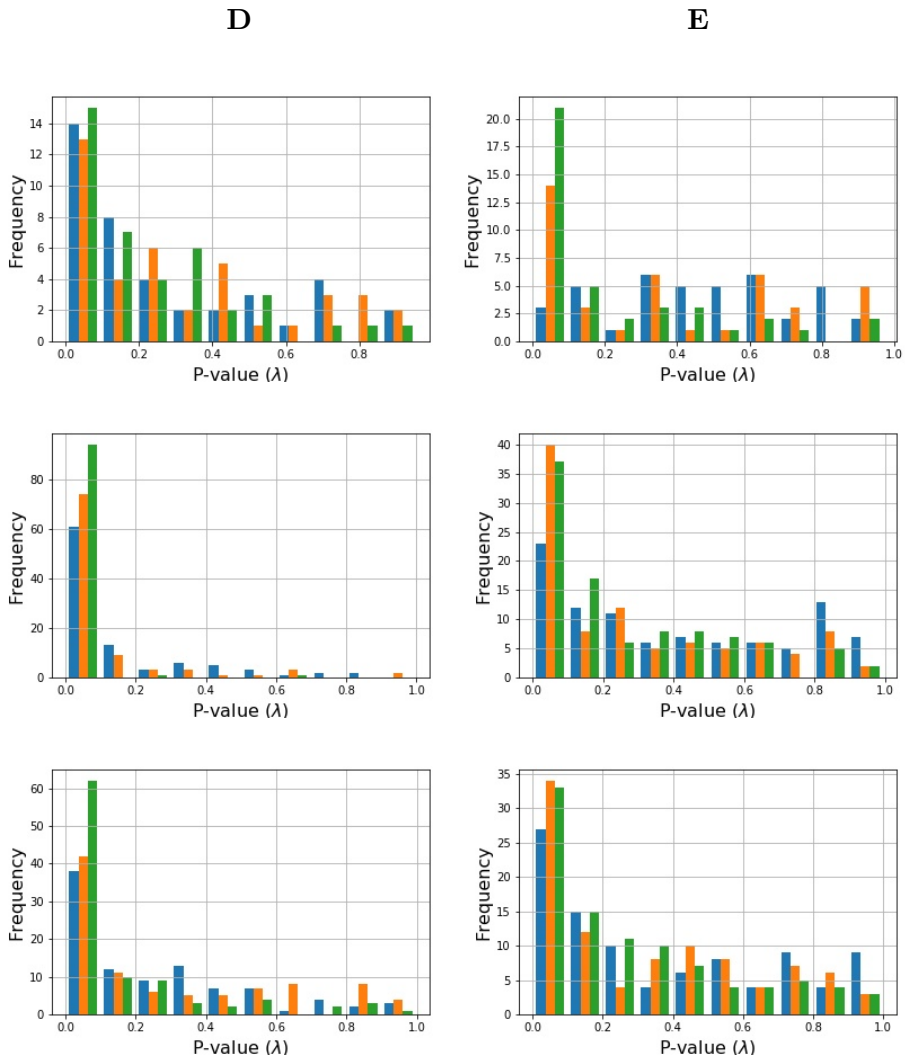


Figure 5.8: Histograms of p-values for residual images D and E. First row corresponds to TV-chambolle denoise, second to phase I. and third to phase II. P-value was calculated for three image channels separately, colors in histogram belongs to corresponding image channel.

D	TV-chambolle	Phase I.	Phase II.
$\bar{\lambda}$ [%]	27.3	8.1	22.8
$\lambda < 0.05$ [%]	25.8	75.0	39.6

Table 5.4: The analysis of residuals of D image. Mean λ value is higher than 0.05, which is good, but percentage rate of $\lambda < 0.05$ is undesirable.

E	TV-chambolle	Phase I.	Phase II.
$\bar{\lambda}$ [%]	35.3	32.0	32.5
$\lambda < 0.05$ [%]	26.7	25.3	23.3

Table 5.5: The analysis of residuals of E image. The results for residuals after phase II. suggest that distribution is near to be Gaussian.

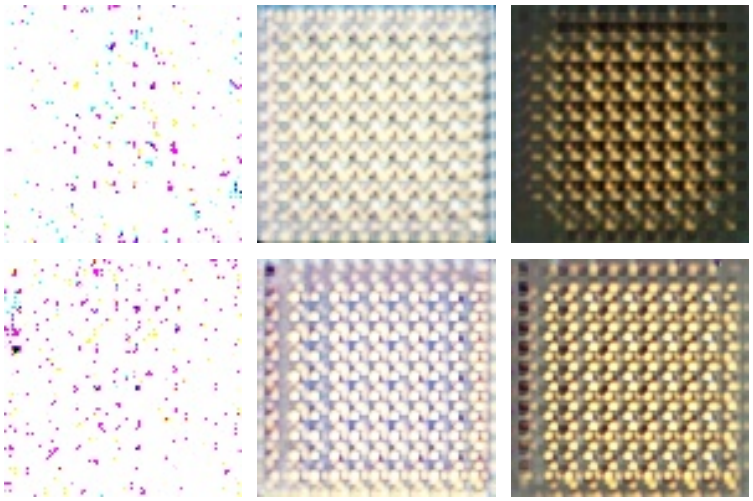


Figure 5.9: We tried how will the generator behave if we show image of noise, white and black image after phase I (first row) and phase II (second row).

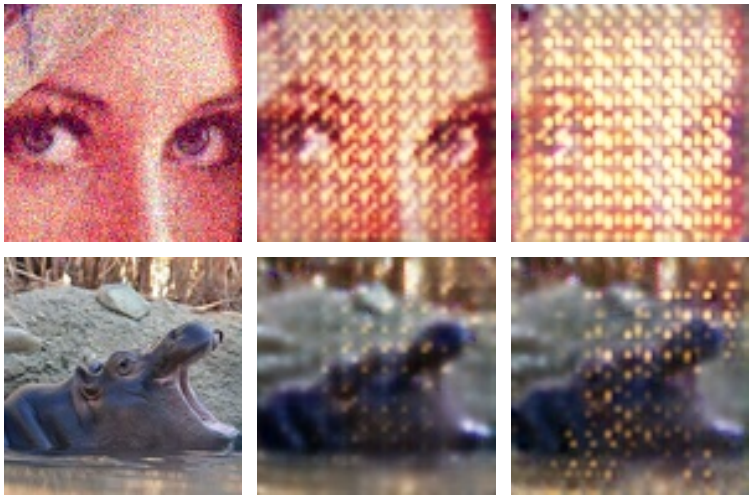


Figure 5.10: Test of the generator on non-astro images. From left: original image, after phase I, after phase II. Analysis of residuals of women's image shows, that $\bar{\lambda} = 1\%$ and 92% of all values are $\lambda < 0.05$. This analysis prove that residuals have not Gaussian distribution as expected (Chung and Yim, 2014).

5.2.1 The network vizualization

To see what is happening in the network we visualized how the image from Figure 5.3 pass through it. The visualization is influenced by the selection effect – each layer contains many activation maps and filters. Numbers I – VI in Figure 5.12 and 5.13 mean layer number. Activation layers appertain to previous convolution/deconvolution layer. When comparing a colour-bars in the pairs convolution/deconvolution layer and the activation layer we see that the activation function works as expected. At Figure 5.10 we see same sets of filters after phase I and II. Filters are searching for patterns in the image, and they are changing markedly at the beginning of training. In the final stages of training, changes are no longer significant as we can see.

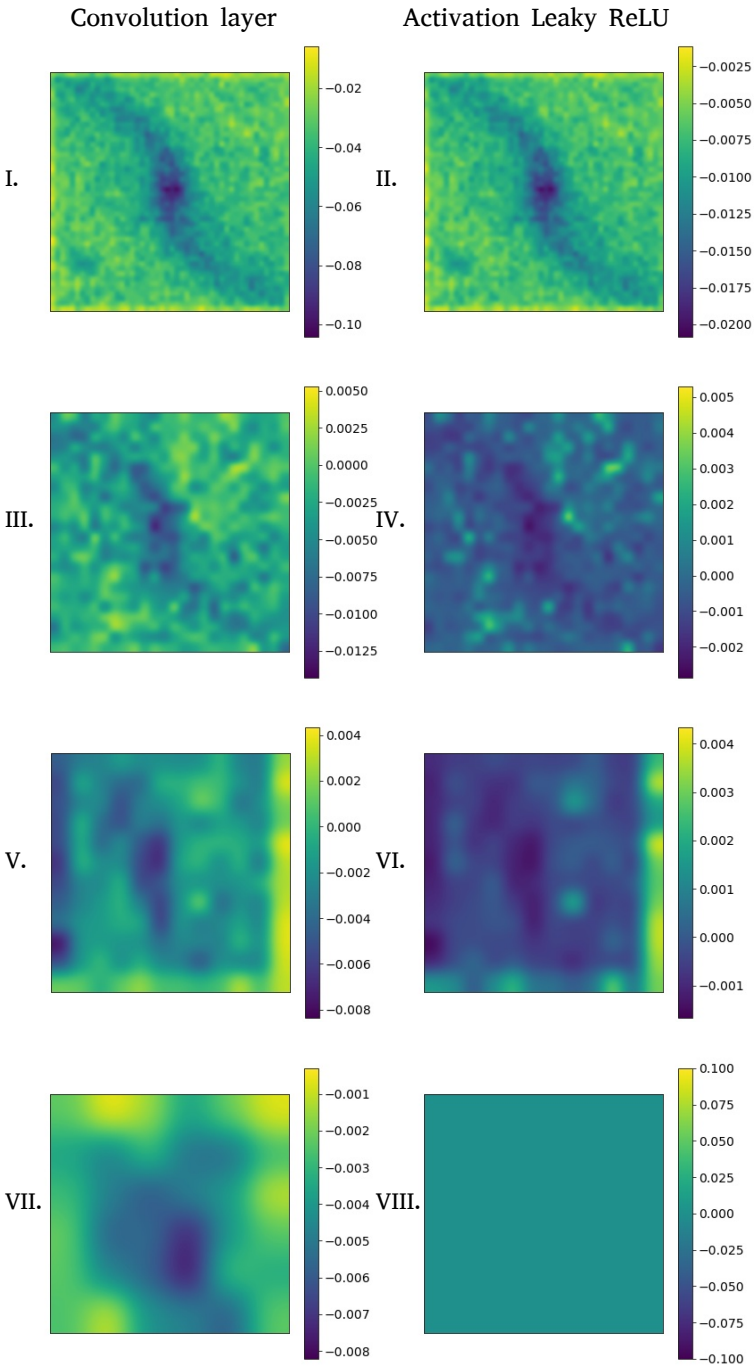


Figure 5.11: Visualization of convolution part of the network.

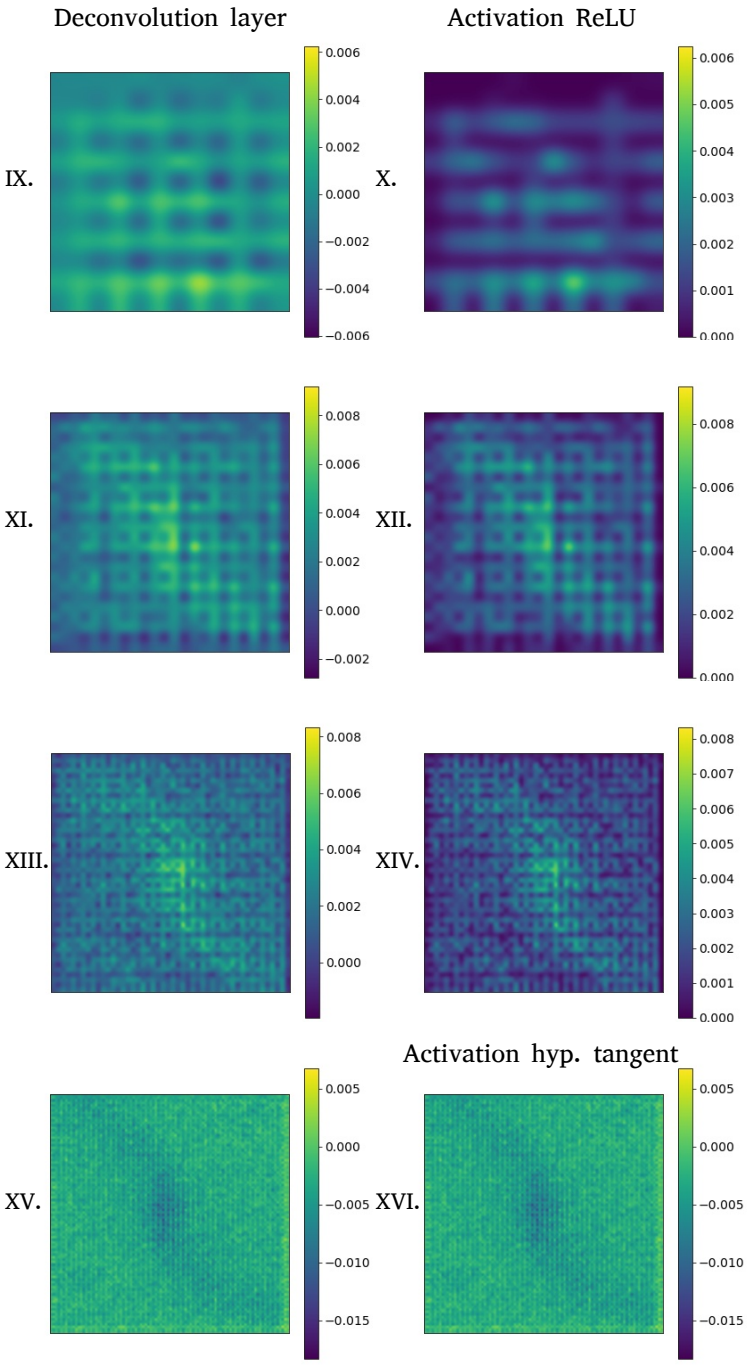


Figure 5.12: Visualization of deconvolution part of the network.

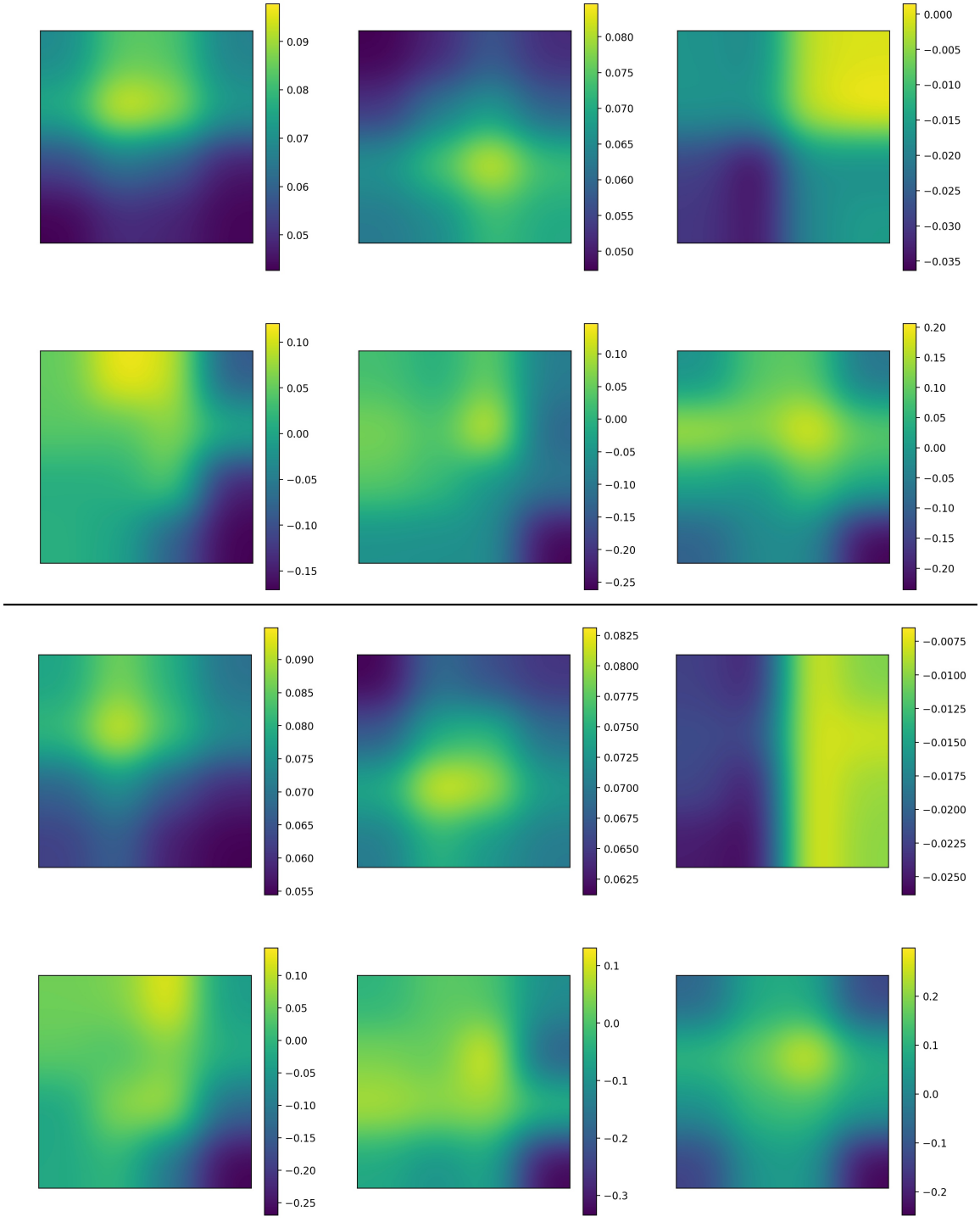


Figure 5.13: Vizualization of same set of filters from first convolutional layer after phase I and phase II.

5.3 Stage II. – the second network application

First step of Stage II. was pretrain our network to get better initialization for our training. For this purpose we generate data, 20000 images and we trained for two epochs. Another change was that we add one convolutional and deconvolutional layer to our generator. We have been train for two hundred epochs with weighted loss 0.9 for MAE and 0.1 for binary cross entropy. Before every training we randomly rotated and shifted training samples – this was made because our dataset is small and this method should help to avoid overfitting. One of the most important changes was the type of noise added into image – noise with Poisson distribution.

For a better understanding of the network, we saved different randomly selected filters, layers and denoise images during training to see how was system evolving. At Figure 5.14 is evolution of the denoising galaxy. The real image of the galaxy is at Figure 5.3 (notice that image is rotated and shifted differently). At Figure 5.15 we can see how were weights changing during training, the biggest changes are at the beginning of training and smallest at the end of training between 160 and 2000 epoch. At Figure 5.17 we see same convolution and deconvolution layer during training. It is interesting, that at the beginning of training convolutional layer was able to capture the shape of galaxy better than at the end. Take a look at the colorbars of deconvolution layer – notice that some values are over two hundred, we have to remind that it is the same layer and that images were at the beginning normalized. This can suggest that there could something goes wrong during the training. Phase IV. shows improvement not just in Ψ and κ but even in visualized convolution layer at Figure 5.16 – the layer capture structure of the galaxy. A more detailed analysis is beyond the scope of this work.
At Figures 5.18 – 5.20 and Tables 5.6 – 5.6 are results of our test of generator for images with Poisson noise. For comparison with our previous results, we use phase I. Clearly, from figures and from Tables we can see that Stage II. – phase III. has worse results as Stage I., this can be caused by:

- We used bigger architecture and it could be required to train it for more epoch
- The larger the network, the more images are needed for prevent to memorize images by the network
- The discriminator was overfited and leads training in wrong way

Examination of this problems should help to create a better network which

can give us better results than obtained one. Because of that, we continue to train the generator (phase IV) without discriminator from the point where we think the discriminator became overfit, after one hundred epochs. Because of time-consuming, we train just for another 25 epochs (125 epochs total), but results are much better than for the phase III. As we can see values of peak signal-to-noise ratio and structural similarity index (Tables 5.6, 5.7) are in almost every case better for Stage I., but phase III. and IV. do better than total-variation Chambolle denoising, which gives us promising beginning for future examination of using the neural networks for this type of problems. Images reconstructed after phase IV. do, according to our opinion, looks better than after any other phase.

Conclusion
This test show us that we have to carefully control loss of the discriminator – if it is overefit it will lead the training in wrong way as we see if we compare phase III. and phase IV. Study of neural networks is also about understanding inner processes, which could help to develop better network.

Ψ[dB]	Noisy image	TV-chambolle	Phase I.	Phase III.	Phase IV.
A	21.0	20.2	24.3	20.7	22.8
B	18.6	18.1	24.2	23.7	25.5
C	25.7	25.4	33.2	29.1	28.5
D	19.6	19.5	32.8	24.5	21.4
E	19.5	19.0	24.2	20.0	24.0

Table 5.6: Peak signal-to-noise ratio (Ψ) for different images, as we can see Ψ is decreasing if we compare images after phase I. and images after phase III. and IV. phase III. and IV. do better than total-variation Chambolle denoising, which gives us promising beginning for future examination of using the neural networks for this type of problems. Images A – C are in Figure 5.18, image D in Figure 5.19, image E in Figure 5.20.

κ	Noisy image	TV-chambolle	Phase I.	Phase III.	Phase IV.
A	0.80	0.73	0.81	0.71	0.80
B	0.83	0.79	0.87	0.81	0.86
C	0.85	0.84	0.91	0.84	0.87
D	0.86	0.84	0.91	0.87	0.82
E	0.76	0.69	0.80	0.71	0.81

Table 5.7: Structural similarity index κ for different images, as we can see κ is almost one for images after phase I. but for images after phase III. are worser – this means that images after phase I. are close to original images (for similar images $\kappa = 1$), better reconstructed. Images A – C are in Figure 5.18, image D in Figure 5.19, image E in Figure 5.20.

Figure 5.14: Vizualization of same galaxy during training for phase III. From upper left: at the beginning, 20 epochs, 40 epochs, 80 epochs, 160 epochs, 200 epochs.

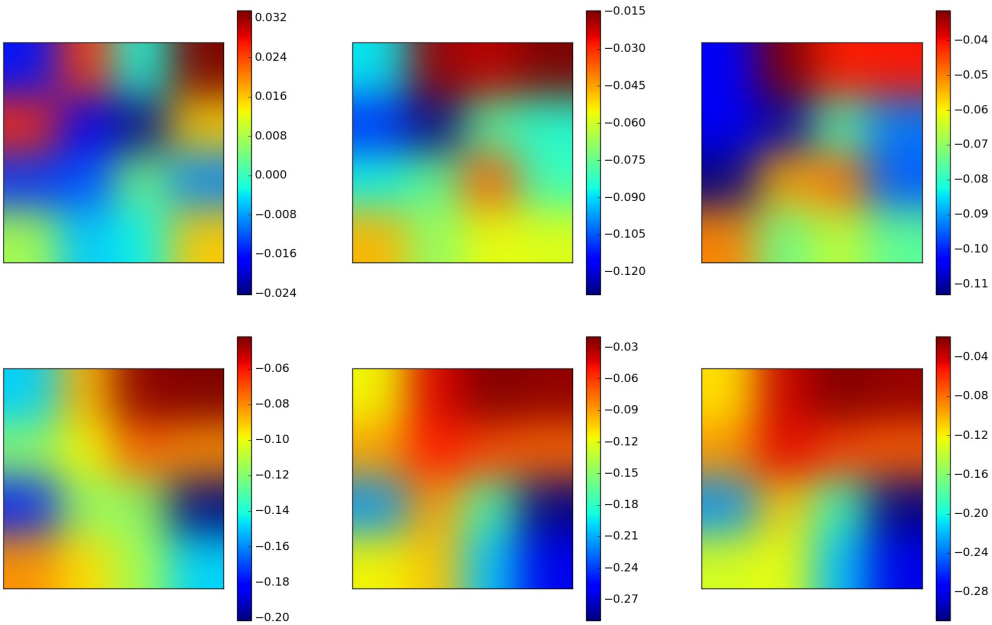


Figure 5.15: Vizualization of same set of filters for phase III. from randomly selected convolution layer. We can see how was weights changing during the training. From upper left: at the beginning, 20 epochs, 40 epochs, 80 epochs, 160 epochs, 200 epochs.

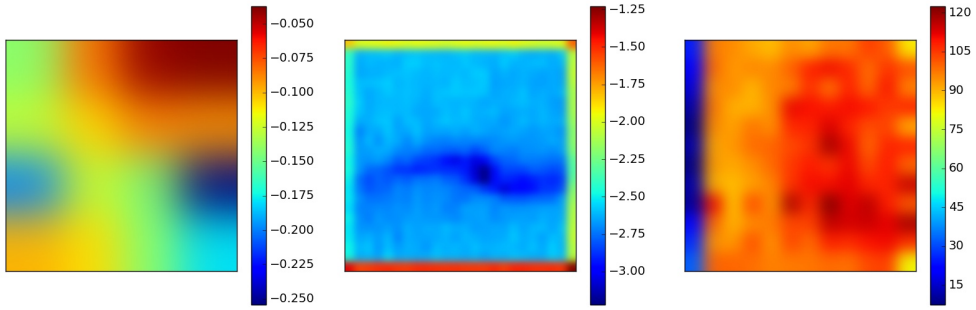


Figure 5.16: From left: Vizualization of filters, convolution layer, deconvolution layer for final state of phase IV, trained 125 epochs. We see that the generator can capture shape of galaxy better than after phase III., Figure 5.17.

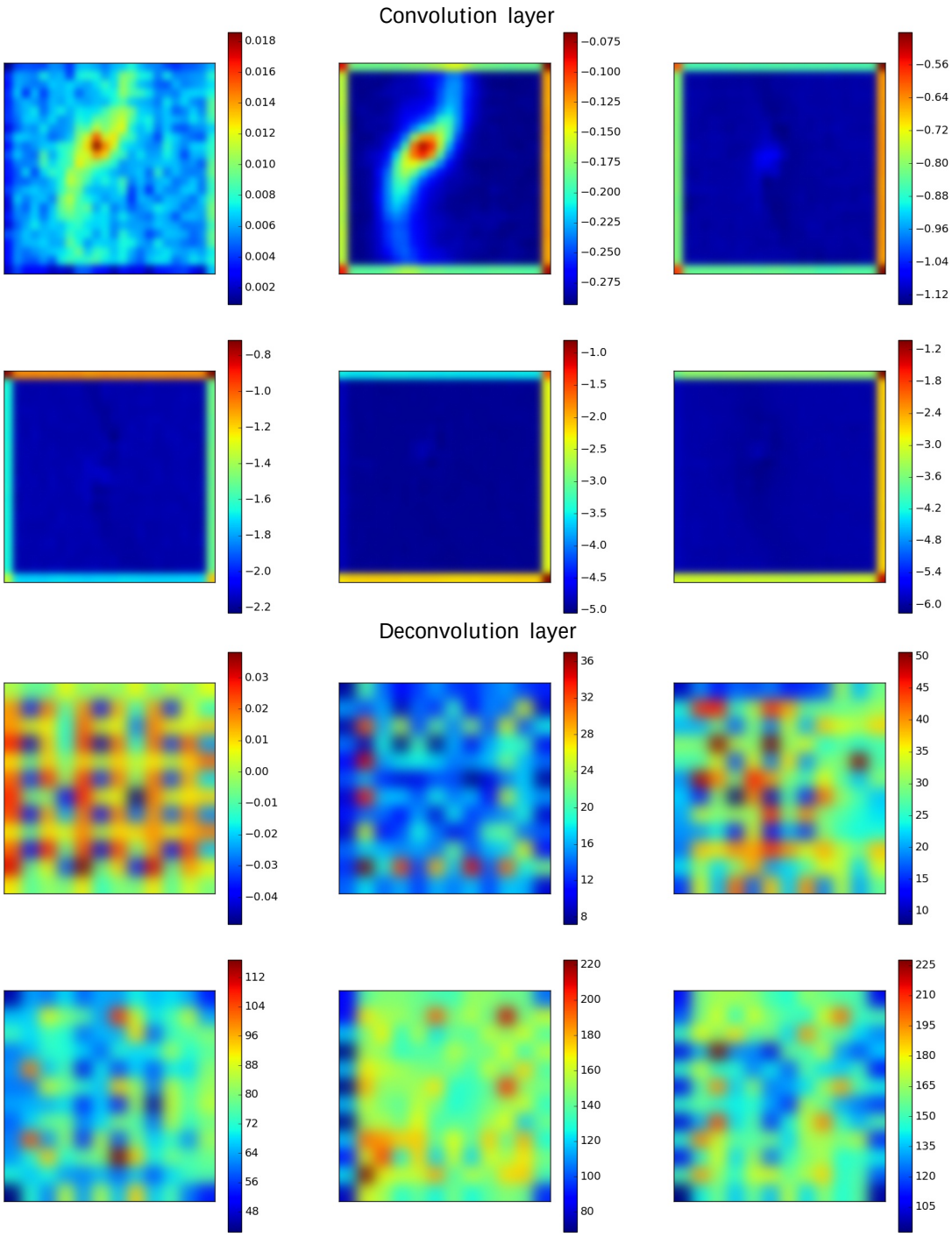


Figure 5.17: Vizualization of same sets of convolution and deconvolution layers for ohase III. From upper left: at the beginning, 20 epochs, 40 epochs, 80 epochs, 160 epochs, 200 epochs.

Figure 5.18: From above: Original images, noisy images, TV-chambolle denoise, generated after phase III., generated after phase IV.

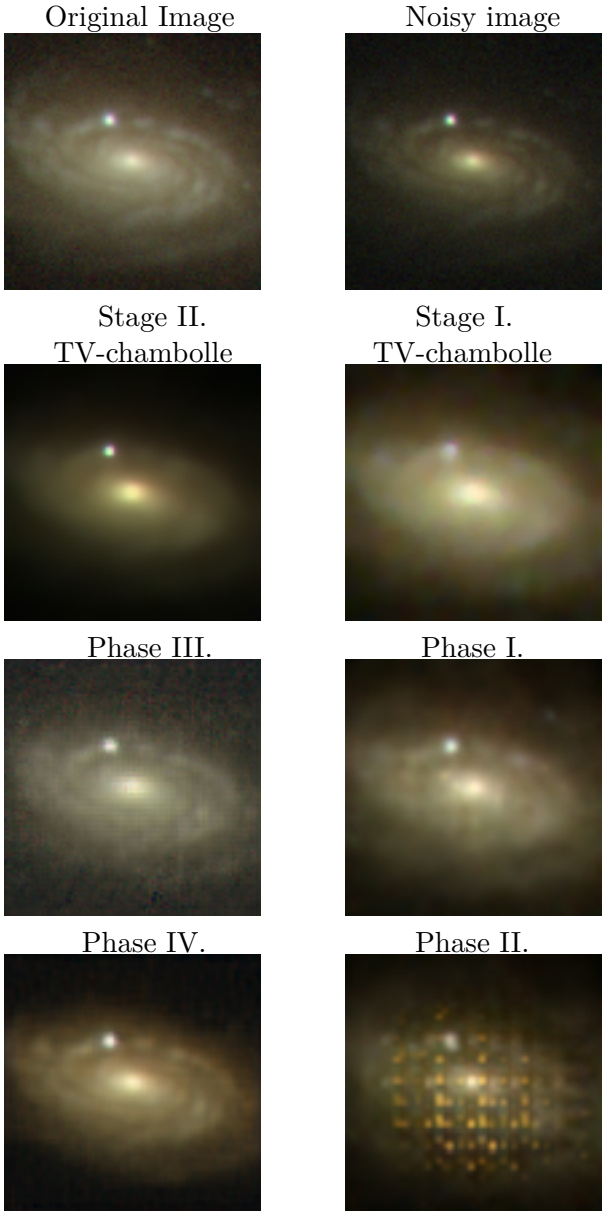


Figure 5.19: (D) Structure which appears after phase II. is not visible after phase IV. which means that there is no sight of overfitting in this image.

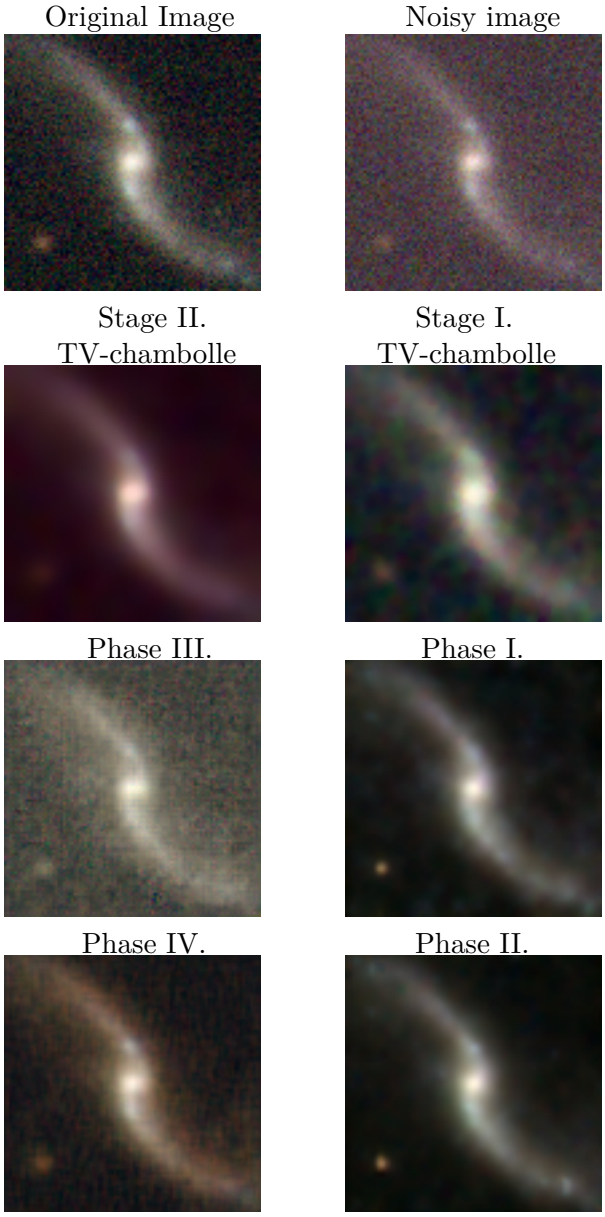


Figure 5.20: (E) As we can see point sources as stars were reconstructed better after phase I. and II, but structure in center of galaxy is the best after phase IV.

Summary and conclusion

Application of neural networks (not just GANs) is widely spread in data-science and has many advantages in processing big datasets and as we are expecting to get a large amount of data in new sky surveys, we need to develop an efficient method for their analysis and processing.

We present generative adversarial network as alternative to image denoising to process big image dataset. Results of our work are as good as nowadays available methods, in some cases even better. This work is of great importance for the future – finding the best method of data processing reflects on the results we get and will help us better understand the universe and the objects in it.

Our experience can by expressed in the rules:

- It is required to have a large dataset for training and use of different types of procedure to prevent the network from overfitting (random rotation and shifting of image, dropout)
- The network architecture have to be adjusted to our problem – unnecessarily large network cause an overfitting
- The number of epoch is also important – again it can cause overfitting or underfitting

By right application of the points above, it is possible to create an efficient network for the desired purpose – in our case for the image denoising. We have proof strength of generative adversarial network for this kind of task and we are expecting to continue in finding another use of neural network in astronomy. We tend to create a bigger dataset with better images and this could help to improve our network. Also, we would like to try Karras et al. (2017) method to see if it would improve the network. Despite our results are not as good as theirs, we support study in Schawinski et al. (2017), which claims that generative adversarial network is able to fully reconstruct noisy images.

We can imagine use of our network as part of processing big amount of data, for example: data would be first denoised and then sent to process with another neural network (e.g galaxy classification). Image denoising would help to better classification of galaxies. User can then choose what type of data he want (e.g spiral galaxies) and he receive data process by our network and raw data of wanted spiral galaxies. Future plans could be to implement different types of neural networks into virtual observatories and study improvement.

References

cs231n: Convolutional neural networks for visual recognition. http://cs231n.github.io/.

Illustration of generative model. https://blog.openai.com/content/images/2017/02/gen_models_diag_2.svg.

Boyat, A. K. and Joshi, B. K. (2015). A Review Paper: Noise Models in Digital Image Processing. *ArXiv 1505.03489*.

Brandt, S. (1976). *Statistical and computational methods in data analysis*. North-Holland Pub. Co.

Chambolle, A. (2004). An Algorithm for Total Variation Minimization and Applications. *J. Math. Imaging Vis.*, 20(1-2):89–97.

Chung, B. and Yim, C. (2014). Hybrid error concealment method combining exemplar-based image inpainting and spatial interpolation. *Signal Processing: Image Communication*, 29(10):1121–1137.

Coates, A., Ng, A., and Lee, H. (2011). An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 215–223, Fort Lauderdale, FL, USA. PMLR.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.*, 12:2121–2159.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *ArXiv 1603.07285*.

Fardo, F. A., Conforto, V. H., de Oliveira, F. C., and Rodrigues, P. S. (2016). A Formal Evaluation of PSNR as Quality Measurement Parameter for Image Segmentation Algorithms. *ArXiv 1605.07116*.

Fort, S. (2017). Towards understanding feedback from supermassive black holes using convolutional neural networks. *ArXiv 1712.00523*.

Gauthier, J. (2015). Conditional generative adversarial nets for convolutional face generation.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Networks. *ArXiv 1406.2661*, (January):53–65.

Janák, Z. (2012). Klasifikace světelných křivek s pomocí neuronových sítí. (Masters Thesis).

Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2017). Progressive Growing of GANs for Improved Quality, Stability, and Variation. *ArXiv 1710.10196*.

Kosiba, M. (2017). Klasifikace galaxií pomocí strojového učení. (Bachelor Thesis).

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551.

Ledig, C., Theis, L., Huszar, F., Caballero, J., Aitken, A. P., Tejani, A., Totz, J., Wang, Z., and Shi, W. (2016). Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. *ArXiv 1609.04802*.

Lupton, R. H., Gunn, J. E., and Szalay, A. S. (1999). A Modified Magnitude System that Produces Well-Behaved Magnitudes, Colors, and Errors Even for Low Signal-to-Noise Ratio Measurements. *The Astronomical Journal*, 118(3):1406–1410.

Mastriani, M. (2016). Union is strength in lossy image compression. *ArXiv 1608.00268*.

McLean, I. S. (2009). *Electronic Imaging in Astronomy: Detectors and Instrumentation*. Springer Praxis Books. Springer Berlin Heidelberg.

Mirza, M. and Osindero, S. (2014). Conditional Generative Adversarial Nets. *ArXiv 1411.1784*.

Noh, H., Hong, S., and Han, B. (2015). Learning Deconvolution Network for Semantic Segmentation. *ArXiv 1505.04366*.

Raschka, S. (2015). *Python Machine Learning*. Packt Publishing.

Rodriguez, A. C., Kacprzak, T., Lucchi, A., Amara, A., Sgier, R., Fluri, J., Hofmann, T., and Réfrégier, A. (2018). Fast Cosmic Web Simulations with Generative Adversarial Networks. *ArXiv 1801.09070*.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *ArXiv 1609.04747*.

Rudin, L. I., Osher, S., and Fatemi, E. (1992). Nonlinear total variation based noise removal algorithms. *Physica D Nonlinear Phenomena*, 60:259–268.

Sadykova, D. and Pappachen James, A. (2018). Quality assessment metrics for edge detection and edge-aware filtering: A tutorial review. *ArXiv 1801.00454*.

Schawinski, K., Zhang, C., Zhang, H., Fowler, L., and Santhanam, G. K. (2017). Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit. *ArXiv 1702.00403*, 467:L110–L114.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

York, D. G., Adelman, J., Anderson, Jr., J. E., and Anderson, S. F. e. a. (2000). The Sloan Digital Sky Survey: Technical Summary. *The Astronomical Journal*, 120(3):1579–1587.