Frederick National Laboratory
for Cancer Research

sponsored by the National Cancer Institute

Generative Adversarial Networks

Dr. Yanling Liu
Manager, Imaging and Visualization Group, ABCS, FNLCR

DEPARTMENT OF HEALTH AND HUMAN SERVICES • National Institutes of Health • National Cancer Institute

Frederick National Laboratory is a Federally Funded Research and Development Center operated by Leidos Biomedical Research, Inc., for the National Cancer Institute

FNLCR

Imaging and Visualization Group (IVG)

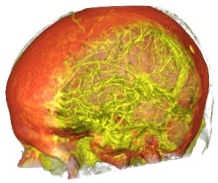
We convert images to knowledge for cancer and biomedical research

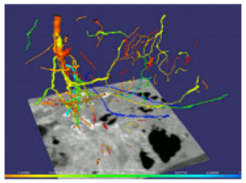
Building systems that automate image processing

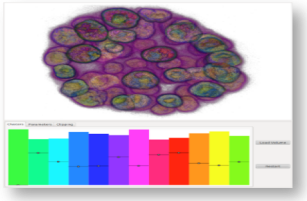
Improving efficiency and enhancing consistency

Creating better tools and workflows for researchers

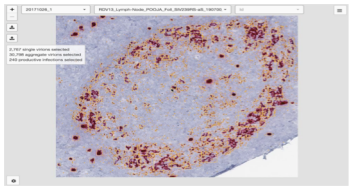
Frederick National Laboratory for Cancer Research
Imaging and Visualization Group (IVG)



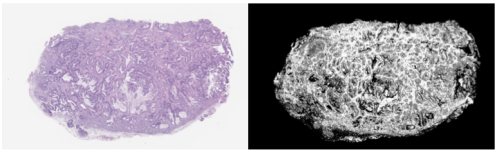




Biomedical Image Visualization

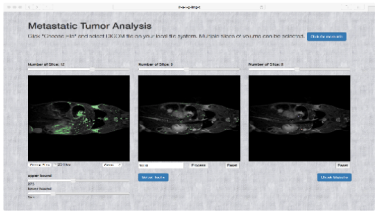


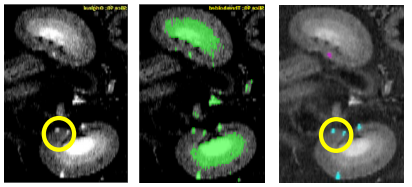
Machine Learning Deep Learning



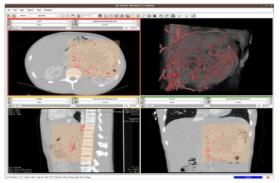
Imaging and Visualization Group

Infrastructure

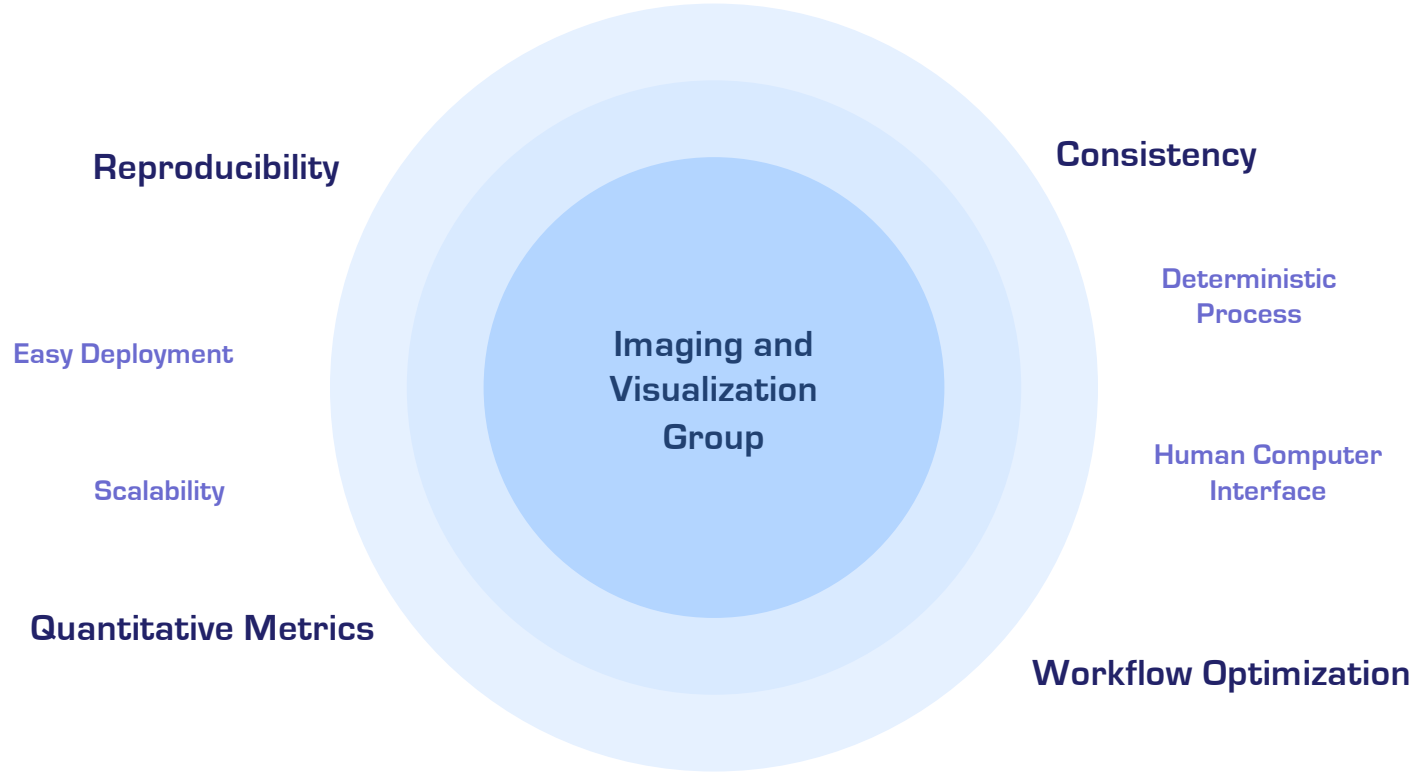




Scientific Software and Algorithm Development



Frederick National Laboratory for Cancer Research
Imaging and Visualization Group (IVG)



“...however, image interpretation by humans is limited due to its subjectivity, large variations across interpreters, and fatigue.”

H. Greenspan, B. van Ginneken and R. M. Summers, "Guest Editorial Deep Learning in Medical Imaging: Overview and Future Promise of an Exciting New Technique," in IEEE Transactions on Medical Imaging, vol. 35, no. 5, pp. 1153-1159, May 2016.

To enhance image **interpretation** by **humans**

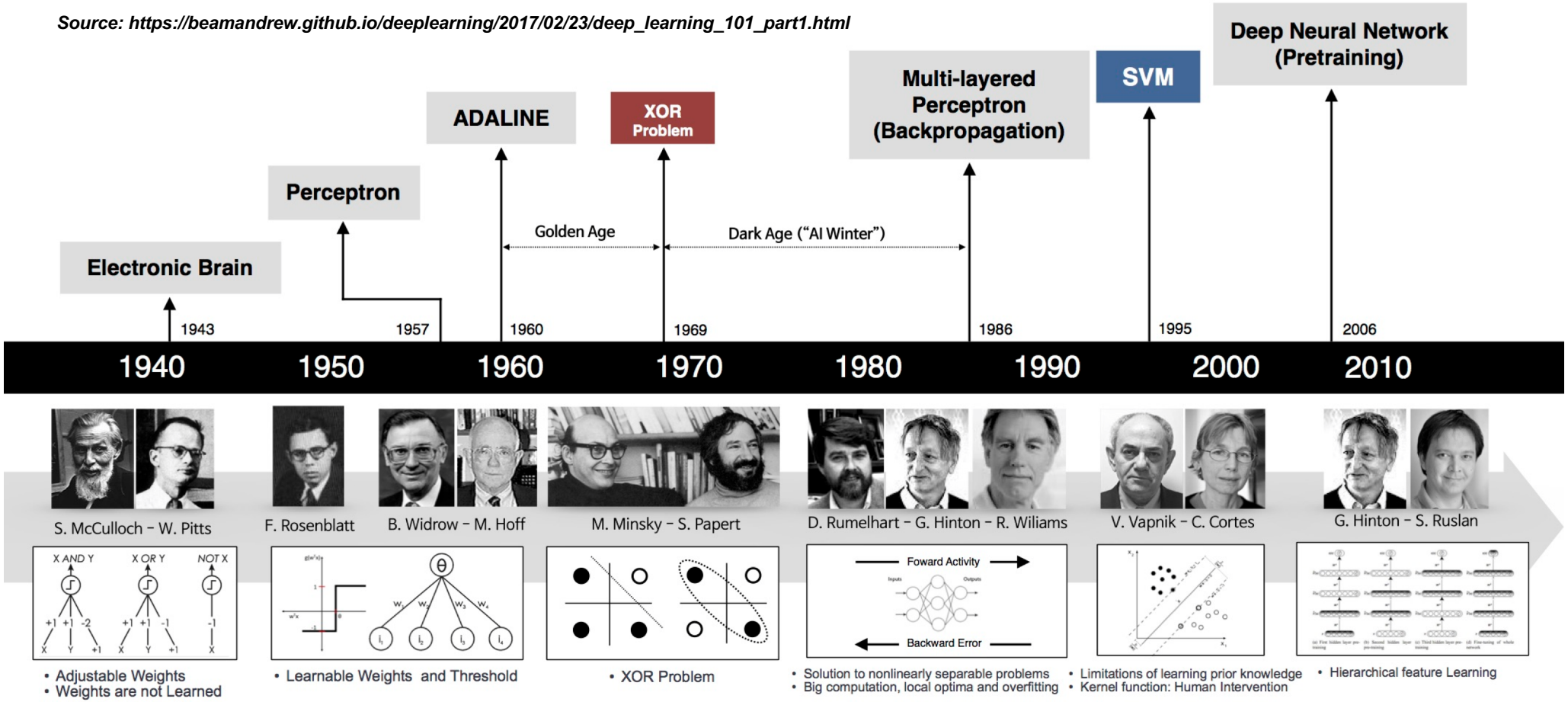
via ***quantitative*** *information derived from* ***Deep Learning*** *based image* ***segmentation***

Machine Learning and Deep Learning

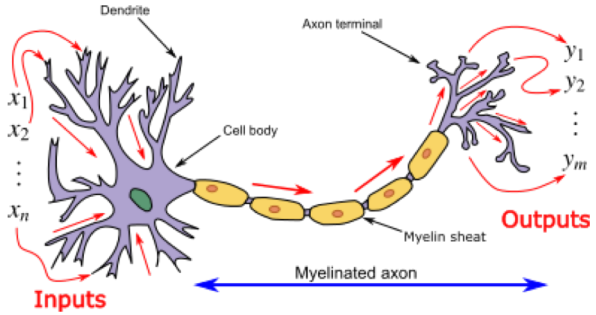
A little bit of history…

Machine Learning: A Brief History

Source: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html



A Simplified View of Deep Learning



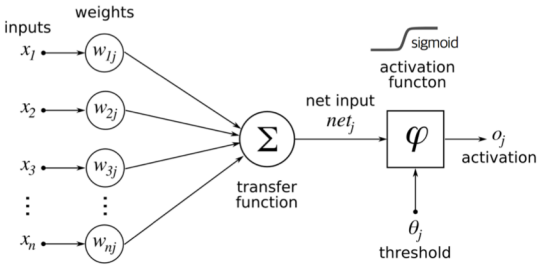


Image credit: Wikipedia

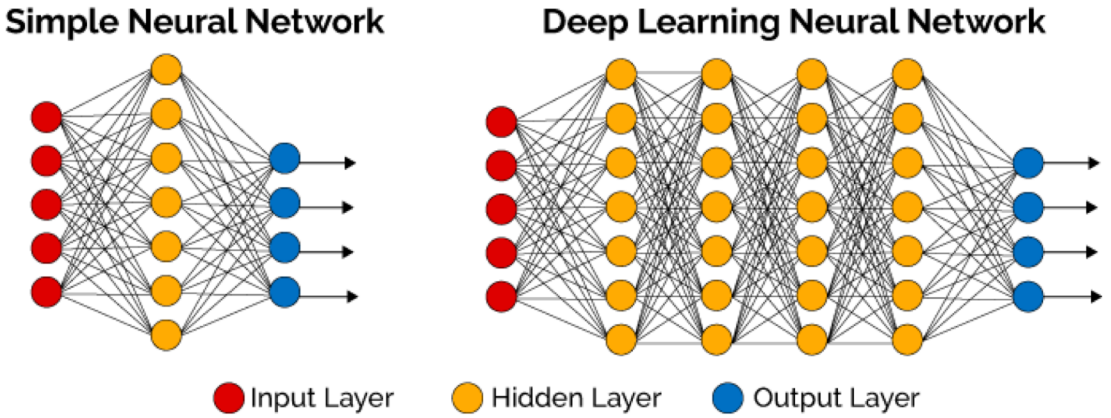
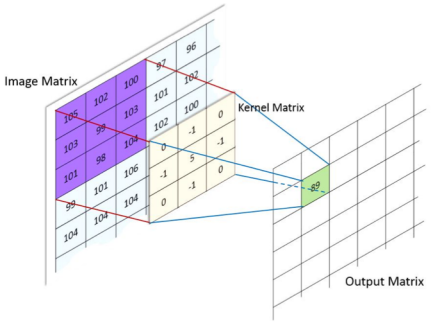


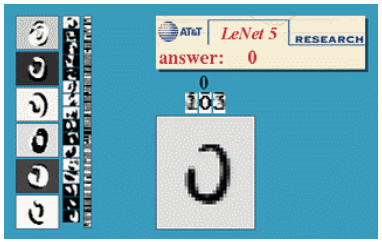
Image credit: Deep Learning made easy with Deep Cognition from Medium

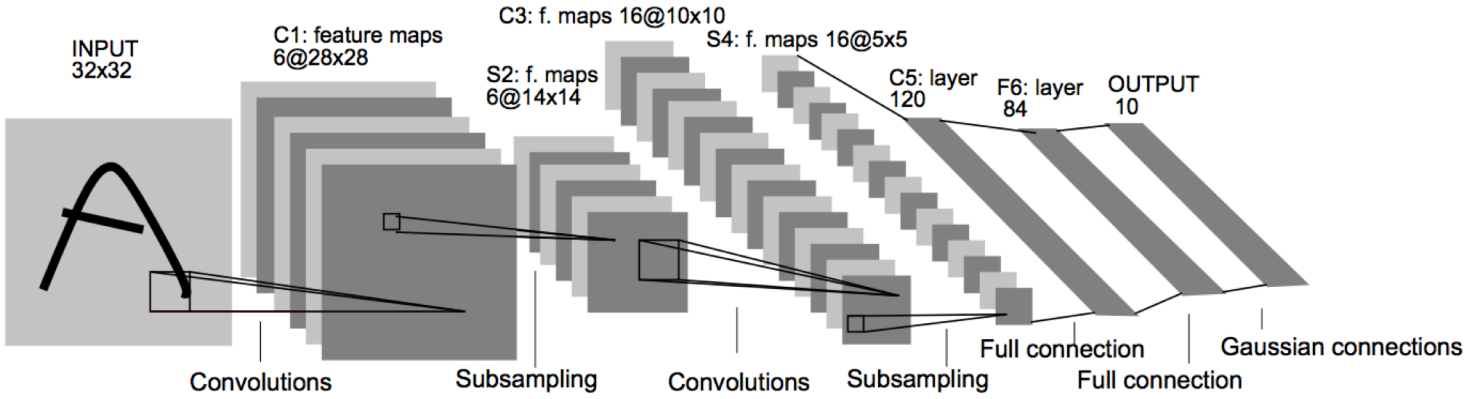
Example of Convolutional Neural Network: LeNet

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner.
Gradient-based learning applied to document recognition.
Proceedings of the IEEE, November 1998.

Source: http://machinelearninguru.com/computer_vision/basics/convolution/image_convolution_1.html





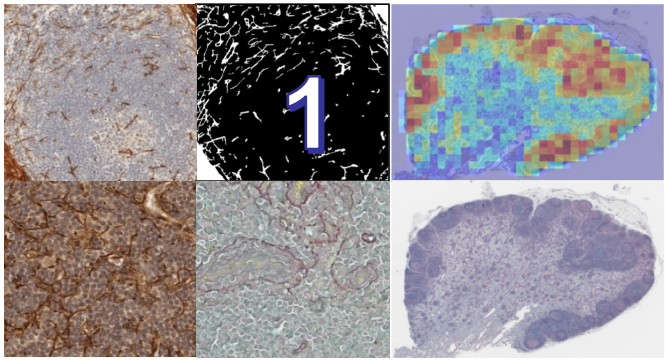


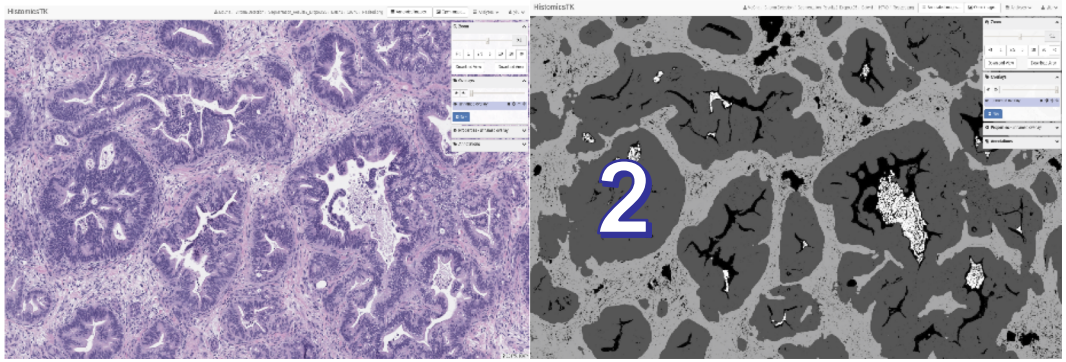
LeNet 5 Architecture

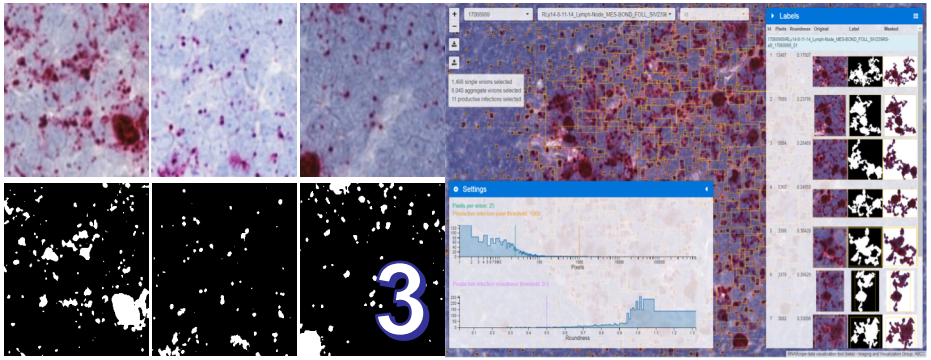
Fun facts: 2-3 days of training time on a Silicon Graphics Origin 2000 using a single 200MHz R10000 processor

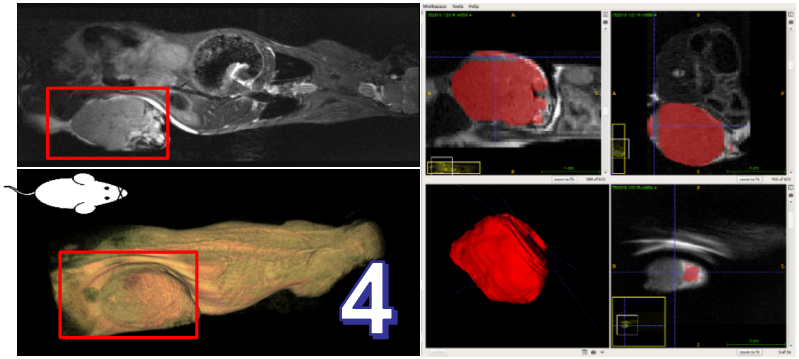
The Training Data Challenge

Examples of Deep Learning enabled projects within IVG





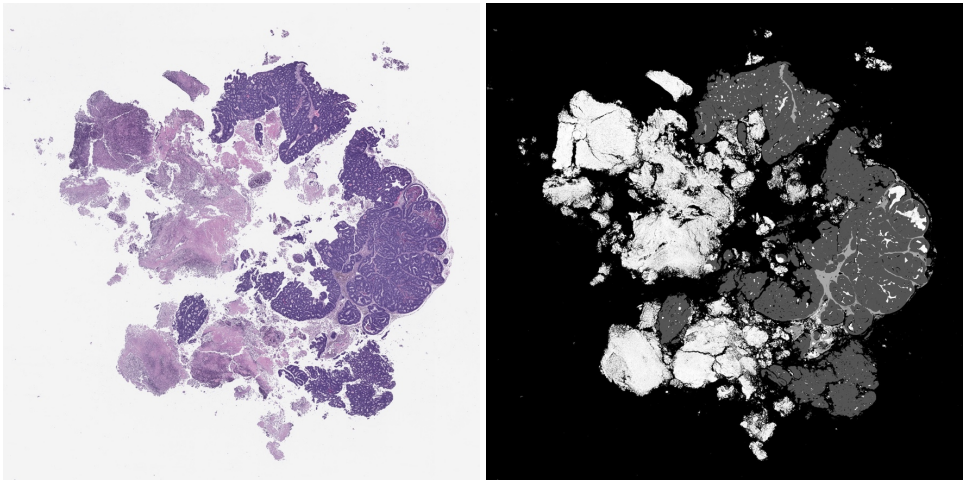




The Training Data Challenge

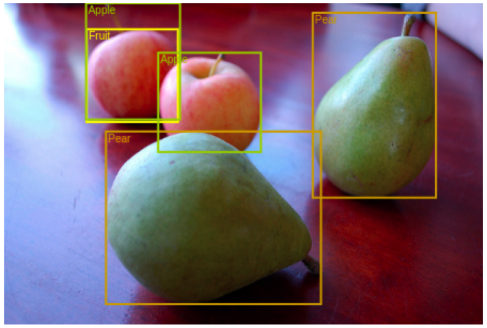
Classification

Source: https://www.pyimagesearch.com/2018/05/07/multi-label-classification-with-keras/



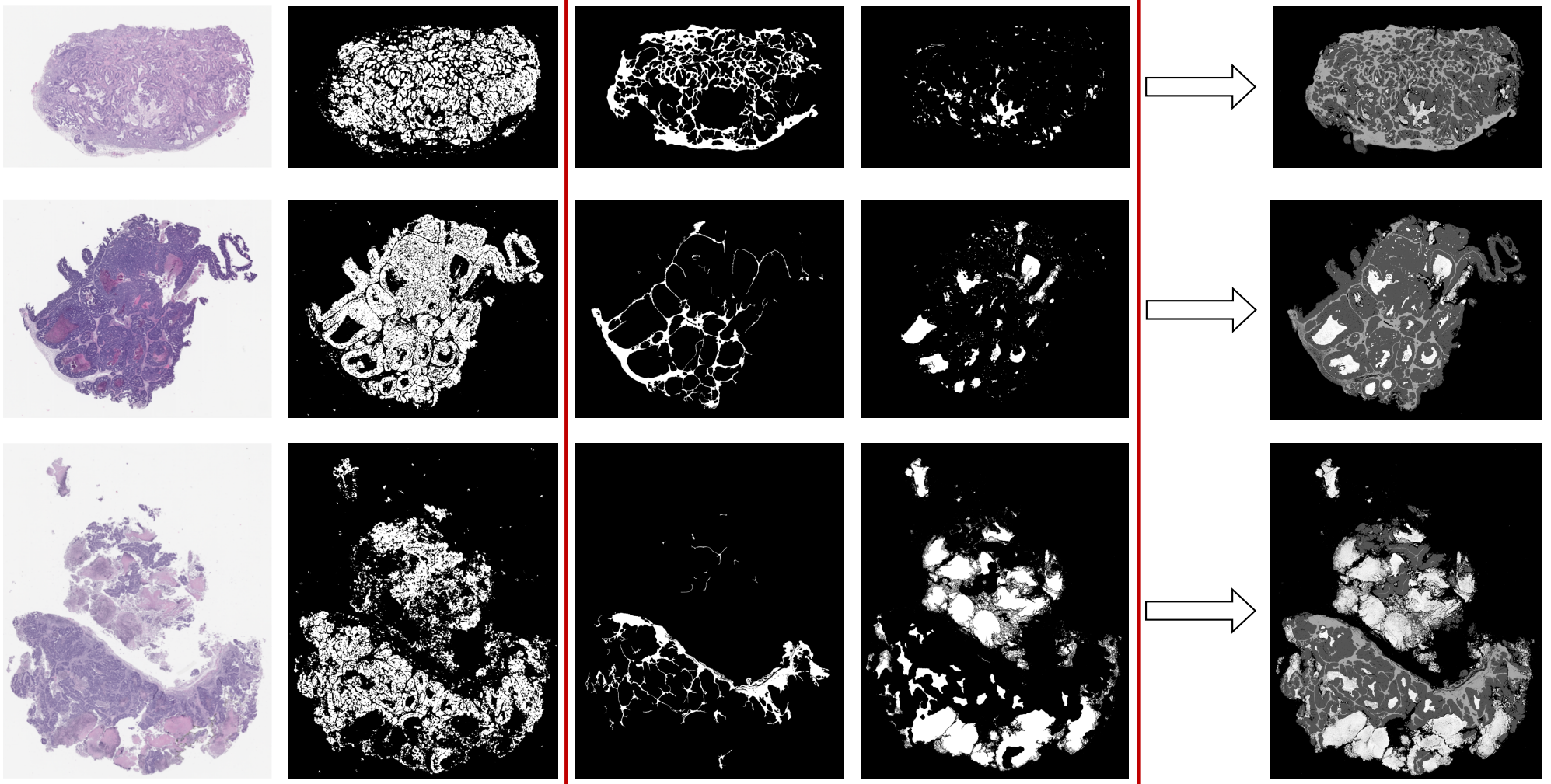
Precise Segmentation

Collaboration with the MoCha group



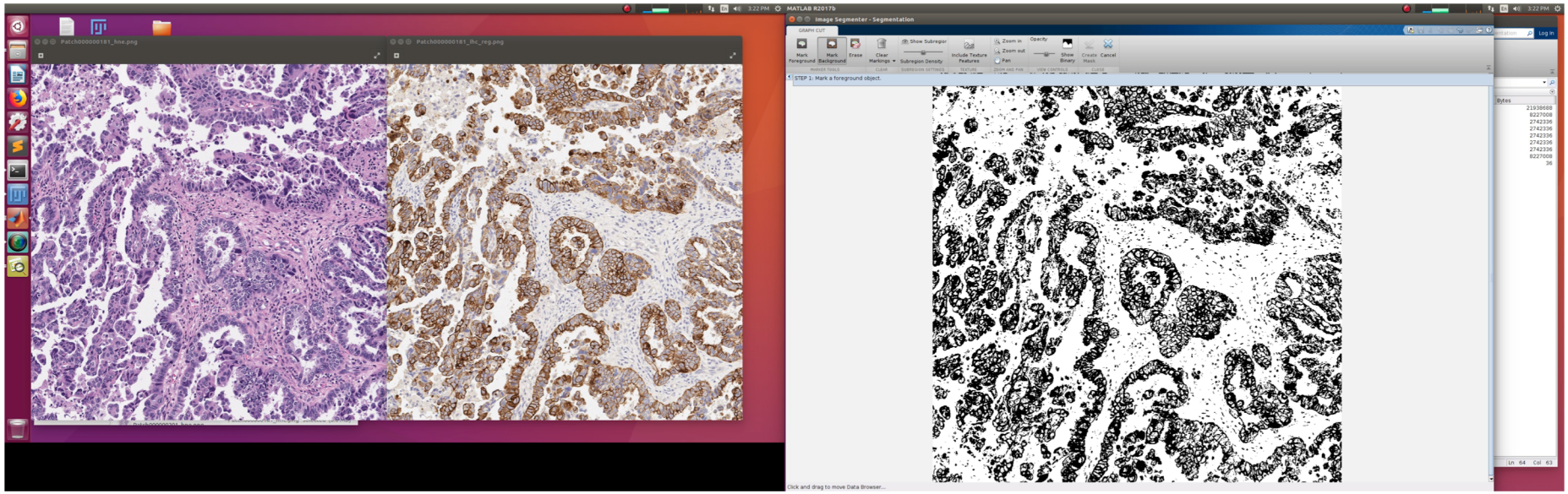
Object Detection

Source: https://blog.algorithmia.com/deep-dive-into-object-detection-with-open-images-using-tensorflow/


Three sample WSIs from the MoCha group
Epithelial cell annotations from the trained U-Net
Manual stroma annotation
Manual necrosis annotation
Guided by a pathologist
Combined annotations

The Training Data Challenge

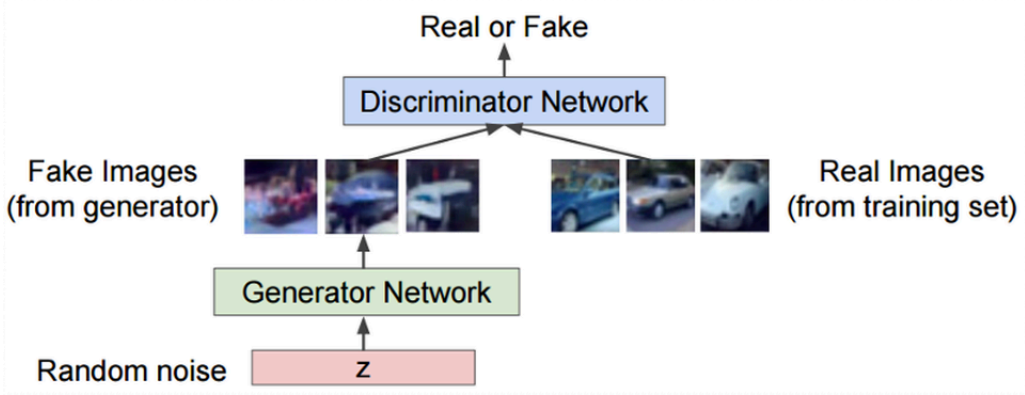
Video showing tedious semi-auto pixel annotation creation process



Generative Adversarial Networks (GANs)

GANs

- A system of two networks
  - Generator
  - Discriminator
  - Double feedback loops
    - The discriminator is in a feedback loop with the ground truth of the images, which we know.
    - The generator is in a feedback loop with the discriminator.



GANs in action. (Source - https://goo.gl/NAN1Jw)

Applications of GANs

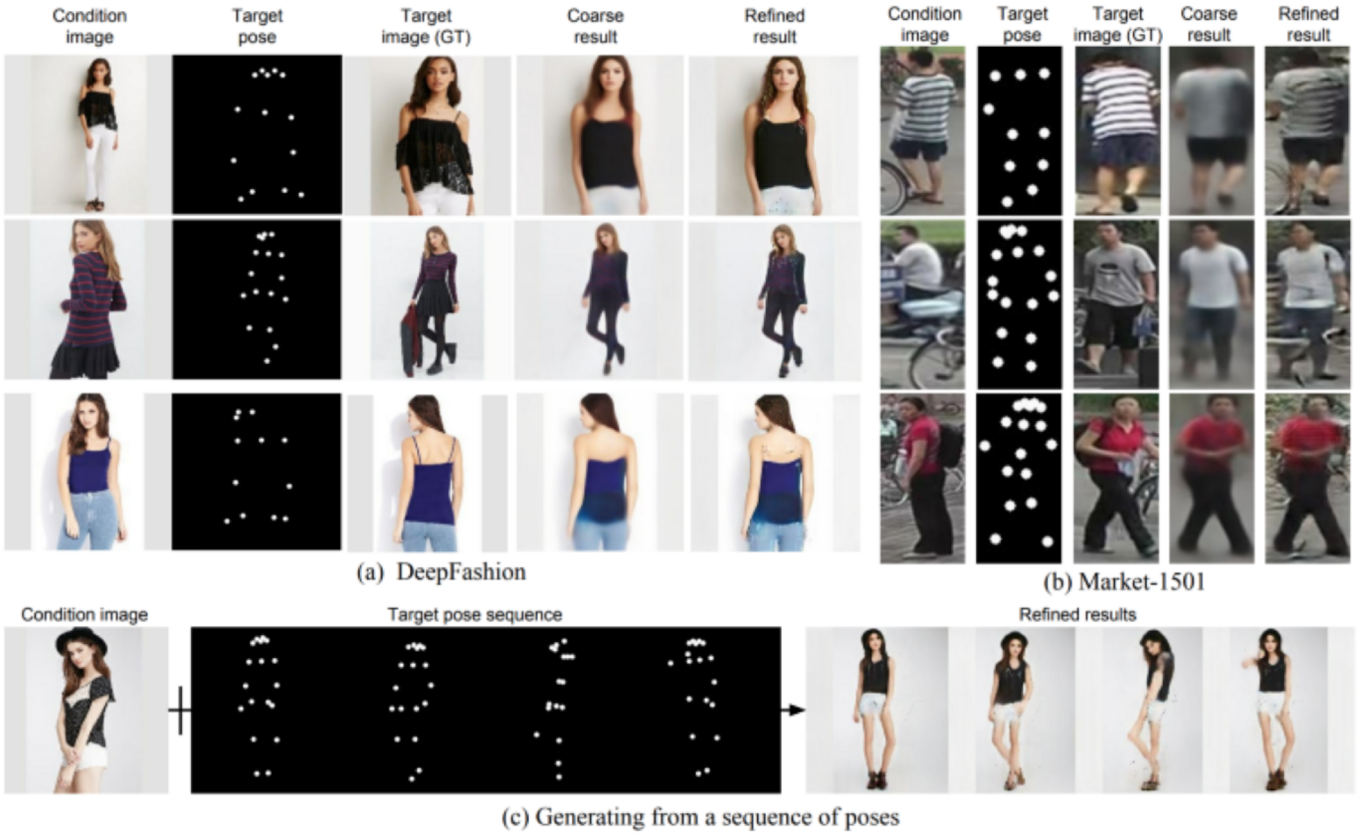
Source

Figure 7: Generated samples

Towards the automatic Anime characters creation with Generative Adversarial Networks

Applications of GANs

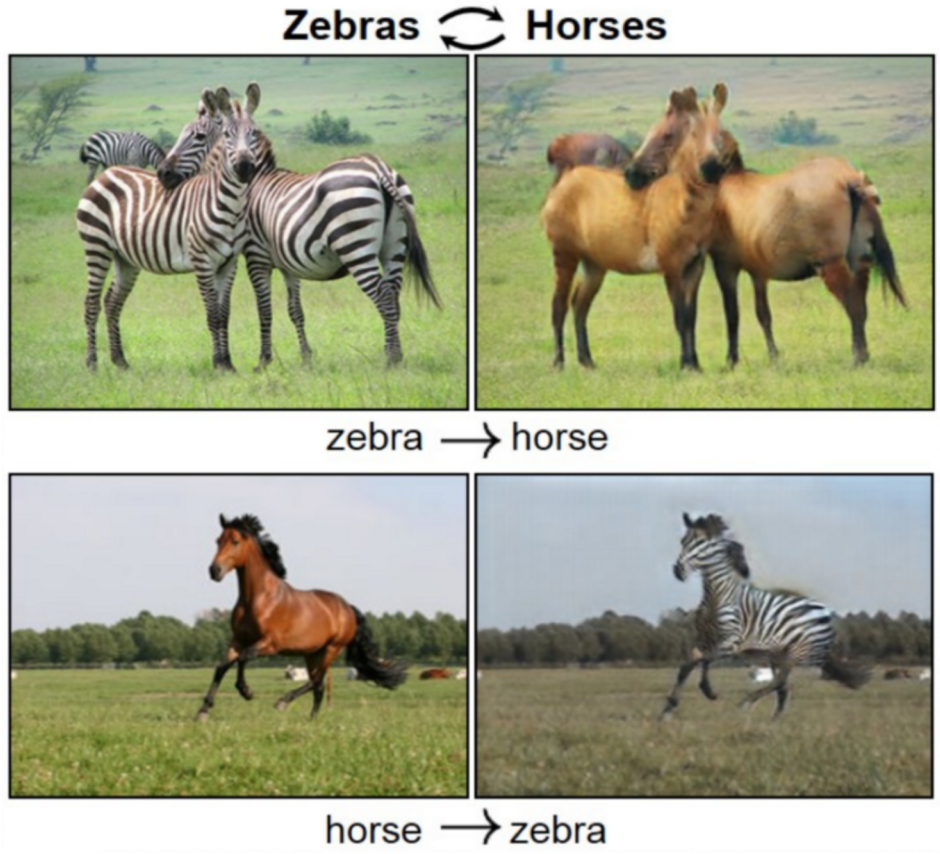
Source



Pose Guided Person Image Generation

Applications of GANs

Source



CycleGAN

Applications of GANs

Source

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

Progressive growing of GANs

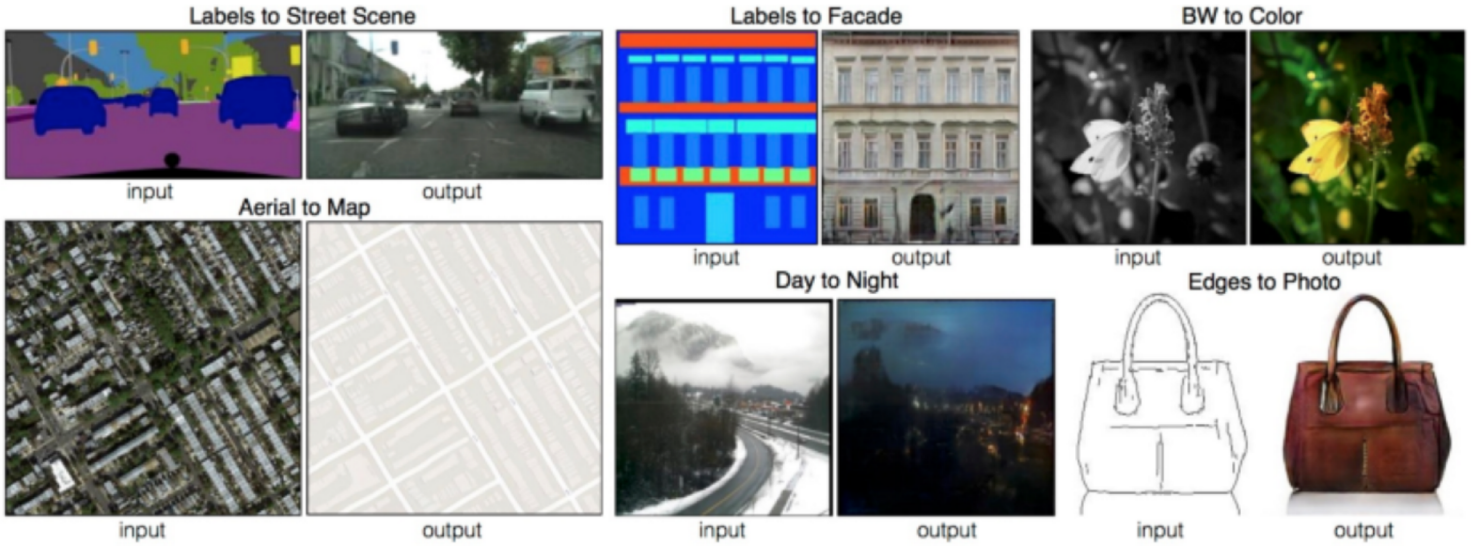
Applications of GANs

Source

IcGAN

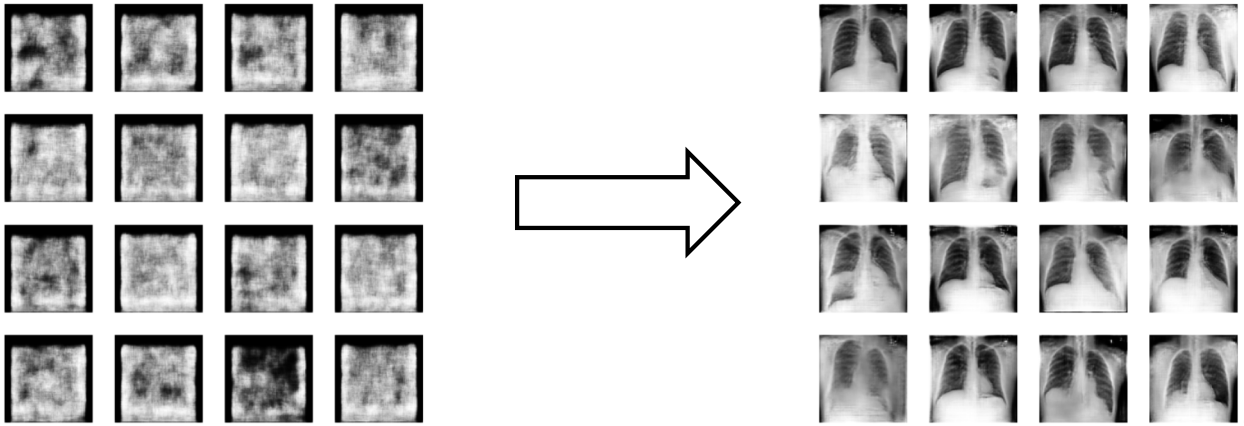
Applications of GANs

Source



pix2pix

Tutorial: GAN for synthetic X-ray images



atory for Cancer Research

RaGAN: modularizing GAN networks for easy data argumentation

- Radiology GAN (RaGAN)
  - GAN: Generative Adversarial Network
  - NIH Hackathon 2018, Sept. 10-12
    - https://github.com/NCBI-Hackathons/RaGAN
  - Attempt to modularize GAN networks for easy and quick (radiology) data argumentation
    - TensorFlow implementation
    - Wrapped as a Docker container
      - https://hub.docker.com/r/johnbamboobilly/ragan/
    - Default 6-layer miniscule network model (proof-of-concept)
    - Future plan
      - Allow drop-in network model integration for quick adoption

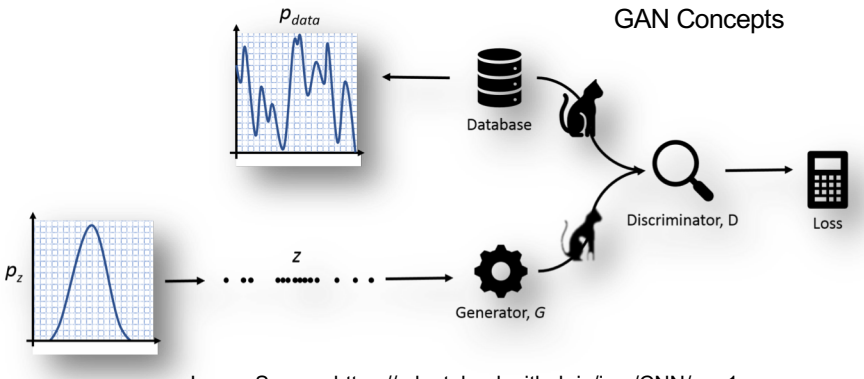
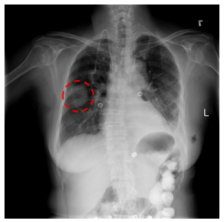
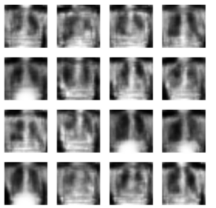


Image Source: https://mlnotebook.github.io/img/CNN/gan1.png



A chest X-ray image
NIH CXR8



Synthesized images of the chest X-Ray data set, 2500 iterations, 28 x 28 pixels each image, default miniscule network

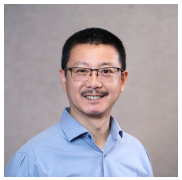
NIH CXR8 Dataset

- Publicly available since 09/2017
  - https://nihcc.app.box.com/v/ChestXray-NIHCC
  - More than 30,000 patients
  - Over 100,000 anonymized chest x-ray images

Nodule	2,705
Effusion	3,955
Atelectasis	4,215
Infiltration	9,547
No Finding	60,361
…	…

Examples of “outliers” in the dataset

Data Preprocessing


Dr. Xinlian Liu

- Source code
  - https://github.com/blazers/gan/blob/master/scripts/p10book.ipynb
  - https://github.com/blazers/gan/blob/master/lists/cxr14_bad_labels.csv
  - https://github.com/abcsFrederick/gan/blob/master/lists/Data_Entry_2017_parsed.csv
- Group selection

```
t_at=table[(table.label=='Atelectasis') & (table.view=='PA') & (table.gender=='M') & (table.age <
71) & (table.age > 19)]
```

```
t_in=table[(table.label=='Infiltration') & (table.view=='PA') & (table.gender=='M') & (table.age <
 71) & (table.age > 19)]
```

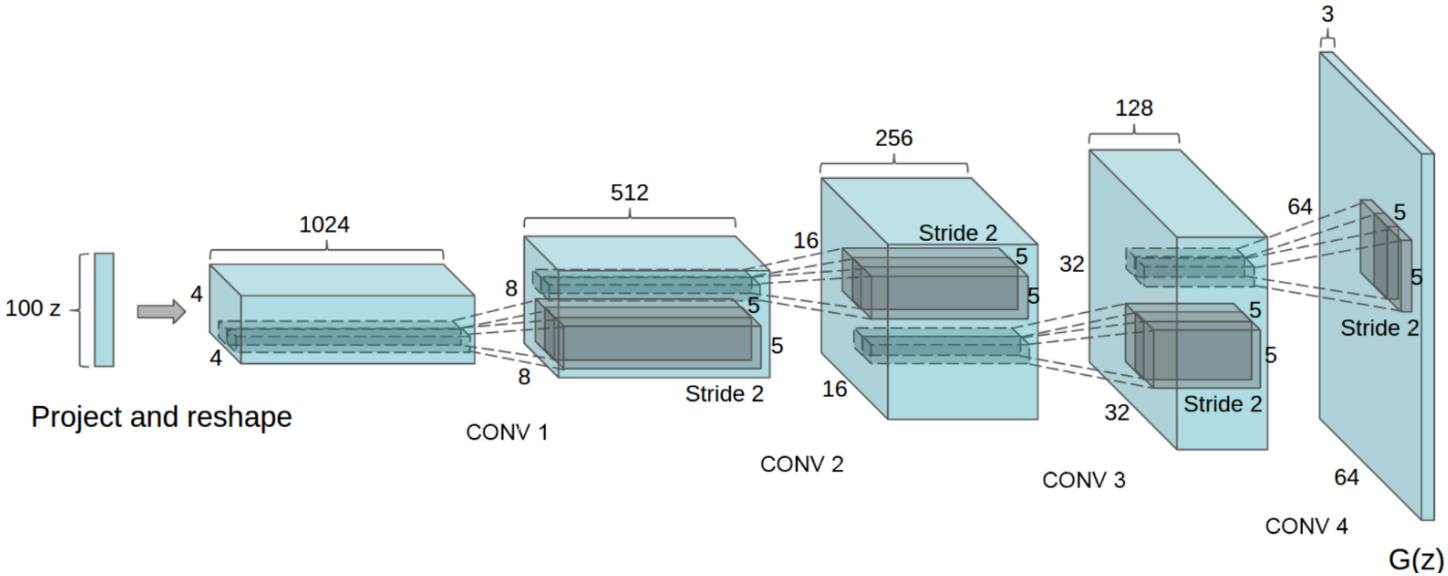
```
t_no=table[(table.label=='No Finding') & (table.view=='PA') & (table.gender=='M') & (table.age < 7
1) & (table.age > 19)]
```

```
: t_ef=table[(table.label=='Effusion') & (table.view=='PA') & (table.gender=='M') & (table.age < 71
) & (table.age > 19)]
```

```
: t_nod=table[(table.label=='Nodule') & (table.view=='PA') & (table.gender=='M') & (table.age < 71)
 & (table.age > 19)]
```

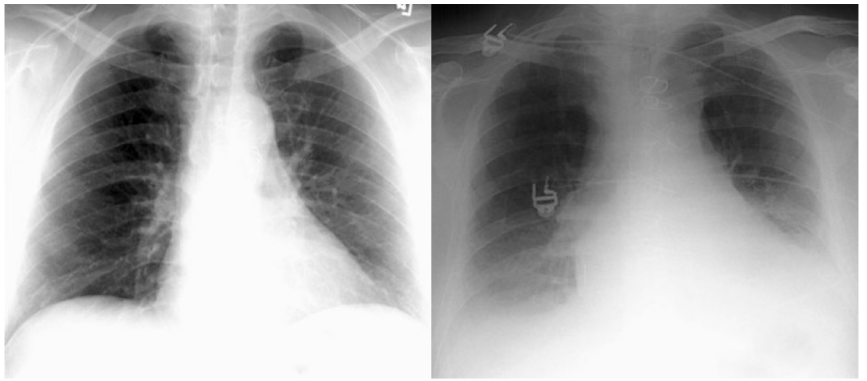
DCGAN

- Original paper
  - https://arxiv.org/abs/1511.06434
  - Deep Convolutional Generative Adversarial Network (DCGAN)



Test 1: GAN -> 64x64 synthetic images

- Training data
  - Infiltration subset
    - Anteriorposterior (AP) view
    - Male
    - Ages between 19 ~ 71
  - 2,126 samples
    - Down-sampled to 64x64 using imagemagick
- Training statistics
  - Four Nvidia Quadro M6000 GPUs
  - 2,000 steps, 256 samples / step
  - Finished in 18 minutes



Posterioranterior (PA) Anteriorposterior (AP)

https://www.med-ed.virginia.edu/courses/rad/cxr/technique3chest.html

Source Code

- https://github.com/abcsFrederick/gan/blob/master/scripts/NIH.AI_DCGAN_64x64_CXR8.py

```
import os
import glob
import numpy as np
import time

from skimage.io import imread

import tensorflow as tf
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Activation, Flatten, Reshape
from tensorflow.python.keras.layers import Conv2D, Conv2DTranspose, UpSampling2D
from tensorflow.python.keras.layers import LeakyReLU, Dropout
from tensorflow.python.keras.layers import BatchNormalization
from tensorflow.python.keras.optimizers import Adam, RMSprop

import matplotlib.pyplot as plt
```

Source Code

- https://github.com/abcsFrederick/gan/blob/master/scripts/NIH.AI_DCGAN_64x64_CXR8.py

```
def load_data(directory_path):
    train_data = []
    for file_ in sorted(os.listdir(directory_path)):
        if file_.endswith('.png'):
            if directory_path.endswith('/'):
                image_path = directory_path + file_
            else: image_path = directory_path + '/' + file_

            #image = imread(image_path)/255.0 # Normalize values
            image = imread(image_path, as_gray=True)
            if np.max(image) > 1.:
                image = image / 255.0 #normalize values
#            print(image.shape)
            image = np.expand_dims(image, axis=-1) # Add channel dim
            train_data.append(image)

    train_data = np.array(train_data)
    return train_data
```

Source

- https://...R8.py

```
# (W-F+2P)/S+1
def discriminator(self):
    if self.D:
        return self.D
    self.D = Sequential()
    depth = 64
    dropout = 0.2

    input_shape = (self.img_rows, self.img_cols, self.channel)
    self.D.add(Conv2D(depth*1, 5, strides=2, input_shape=input_shape, padding='same'))
    self.D.add(LeakyReLU(alpha=0.2))
    self.D.add(Dropout(dropout))

    self.D.add(Conv2D(depth*2, 5, strides=2, padding='same'))
    self.D.add(LeakyReLU(alpha=0.2))
    self.D.add(Dropout(dropout))

    self.D.add(Conv2D(depth*4, 5, strides=2, padding='same'))
    self.D.add(LeakyReLU(alpha=0.2))
    self.D.add(Dropout(dropout))

    self.D.add(Conv2D(depth*8, 5, strides=1, padding='same'))
    self.D.add(LeakyReLU(alpha=0.2))
    self.D.add(Dropout(dropout))

    # Out: 1-dim probability
    self.D.add(Flatten())
    self.D.add(Dense(1))
    self.D.add(Activation('sigmoid'))
    self.D.summary()
    return self.D
```

Source C

- https://githu CXR8.py

```
def generator(self):
    if self.G:
        return self.G
    self.G = Sequential()
    dropout = 0.2
    depth = 64+64+64+64
    dim = 8

    self.G.add(Dense(dim*dim*depth, input_dim=100))
    self.G.add(BatchNormalization(momentum=0.9))
    self.G.add(Activation('relu'))
    self.G.add(Reshape((dim, dim, depth)))
    self.G.add(Dropout(dropout))

    self.G.add(UpSampling2D())
    self.G.add(Conv2DTranspose(int(depth/2), 5, padding='same'))
    self.G.add(BatchNormalization(momentum=0.9))
    self.G.add(Activation('relu'))

    self.G.add(UpSampling2D())
    self.G.add(Conv2DTranspose(int(depth/4), 5, padding='same'))
    self.G.add(BatchNormalization(momentum=0.9))
    self.G.add(Activation('relu'))

    self.G.add(UpSampling2D())
    self.G.add(Conv2DTranspose(int(depth/8), 5, padding='same'))
    self.G.add(BatchNormalization(momentum=0.9))
    self.G.add(Activation('relu'))

    # Out: 64 x 64 x 1 grayscale image [0.0,1.0] per pix
    self.G.add(Conv2DTranspose(1, 5, padding='same'))
    self.G.add(Activation('sigmoid'))
    self.G.summary()
    return self.G
```

Source

- https://...28.py

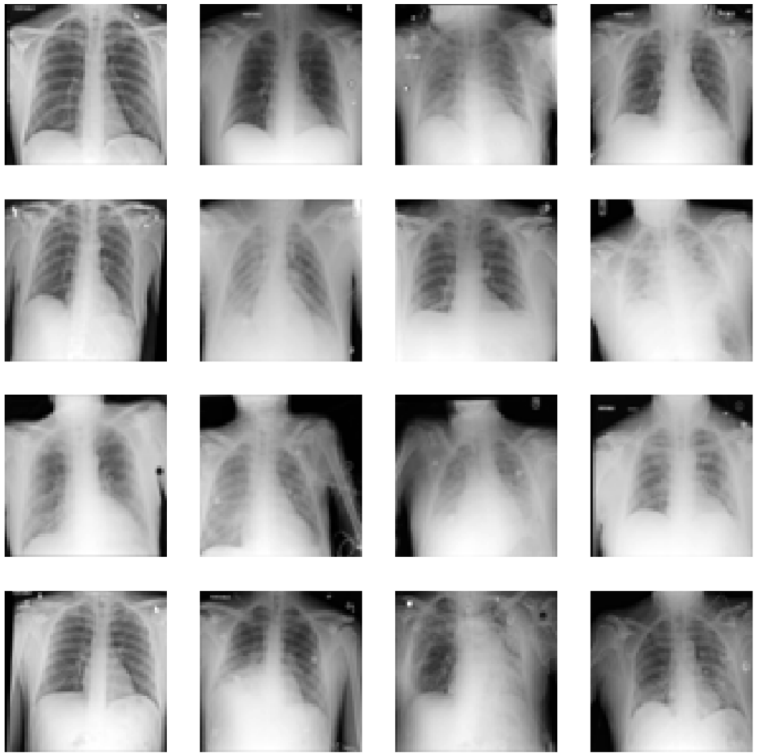
```
def discriminator_model(self):
    if self.DM:
        return self.DM
    optimizer = RMSprop(lr=2e-4, decay=1e-8)
    self.DM = Sequential()
    self.DM.add(self.discriminator())

    #multi_gpu
    with tf.device('/cpu:0'):
        self.DM = multi_gpu_model(self.DM, gpus=4)

    self.DM.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return self.DM

def adversarial_model(self):
    if self.AM:
        return self.AM
    optimizer = RMSprop(lr=1e-4, decay=1e-8)
    self.AM = Sequential()
    self.AM.add(self.generator())
    self.AM.add(self.discriminator())

    #multi_gpu
    with tf.device('/cpu:0'):
        self.AM = multi_gpu_model(self.AM, gpus=4)

    self.AM.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return self.AM
```

Source

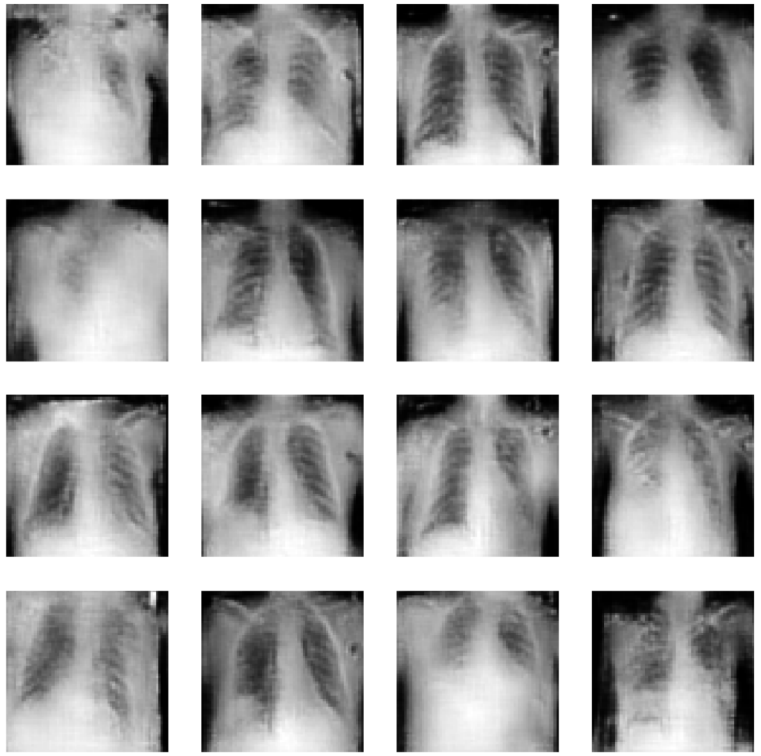
- https://...R8.py

```
def train(self, train_steps=2000, batch_size=256, save_interval=0):
    print(self.x_train.shape[0])
    noise_input = None
    if save_interval>0:
        noise_input = np.random.uniform(-1.0, 1.0, size=[16, 100])
    for i in range(train_steps):
        images_train = self.x_train[np.random.randint(0,
            self.x_train.shape[0], size=batch_size), :, :, :]

        #train the discriminator first
        noise = np.random.uniform(-1.0, 1.0, size=[batch_size, 100])
        images_fake = self.generator.predict(noise)
        x = np.concatenate((images_train, images_fake))
        y = np.ones([2*batch_size, 1])
        y[batch_size:, :] = 0
        d_loss = self.discriminator.train_on_batch(x, y)

        #train the adversarial model second
        y = np.ones([batch_size, 1])
        noise = np.random.uniform(-1.0, 1.0, size=[batch_size, 100])
        a_loss = self.adversarial.train_on_batch(noise, y)

        log_mesg = "%d: [D loss: %f, acc: %f]" % (i, d_loss[0], d_loss[1])
        log_mesg = "%s  [A loss: %f, acc: %f]" % (log_mesg, a_loss[0], a_loss[1])
        print(log_mesg)
        if save_interval>0:
            if (i+1)%save_interval==0:
                self.plot_images(save2file=True, samples=noise_input.shape[0],\
                    noise=noise_input, step=(i+1))
```

Results (2,126 samples, 2,000 steps, 18 minutes)

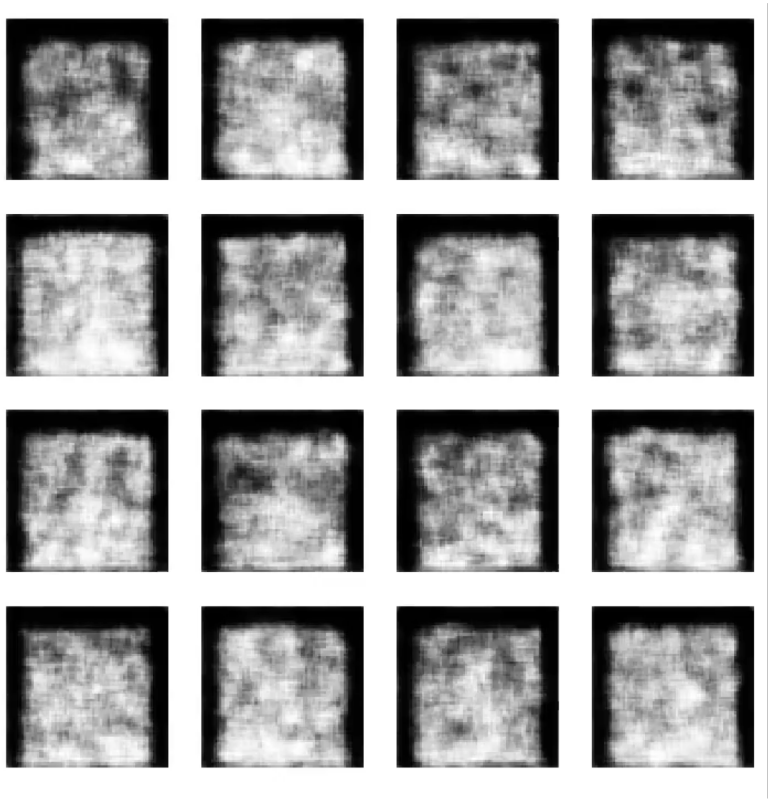


Real



Fake

Training Snapshots Every 10 Steps



Test 2: GAN -> 128x128 synthetic images

- Training data
  - 'No Finding' subset
    - Male
    - Ages between 19 ~ 71
  - 19,011 samples
    - Down-sampled to 128x128 using imagemagick
- Training statistics
  - Four Nvidia Quadro M6000 GPUs
  - 2,000 steps, 256 samples / step
  - Finished in 36 minutes

Source Code

- https://github.com/abcsFrederick/gan/blob/master/scripts/NIH.AI_DCGAN_128x128_CXR8.py

```
# (W-F+2P)/S+1
def discriminator(self):
    if self.D:
        return self.D
    self.D = Sequential()
    depth = 64
    dropout = 0.2

    input_shape = (self.img_rows, self.img_cols, self.channel)
    self.D.add(Conv2D(depth*1, 5, strides=2, input_shape=input_shape, padding='same'))
    self.D.add(LeakyReLU(alpha=0.2))
    self.D.add(Dropout(dropout))

    self.D.add(Conv2D(depth*2, 5, strides=2, padding='same'))
    self.D.add(LeakyReLU(alpha=0.2))
    self.D.add(Dropout(dropout))

    self.D.add(Conv2D(depth*4, 5, strides=2, padding='same'))
    self.D.add(LeakyReLU(alpha=0.2))
    self.D.add(Dropout(dropout))

    self.D.add(Conv2D(depth*2, 5, strides=1, padding='same'))
    self.D.add(LeakyReLU(alpha=0.2))
    self.D.add(Dropout(dropout))

    # Out: 1-dim probability
    self.D.add(Flatten())
    self.D.add(Dense(1))
    self.D.add(Activation('sigmoid'))
    self.D.summary()
    return self.D
```

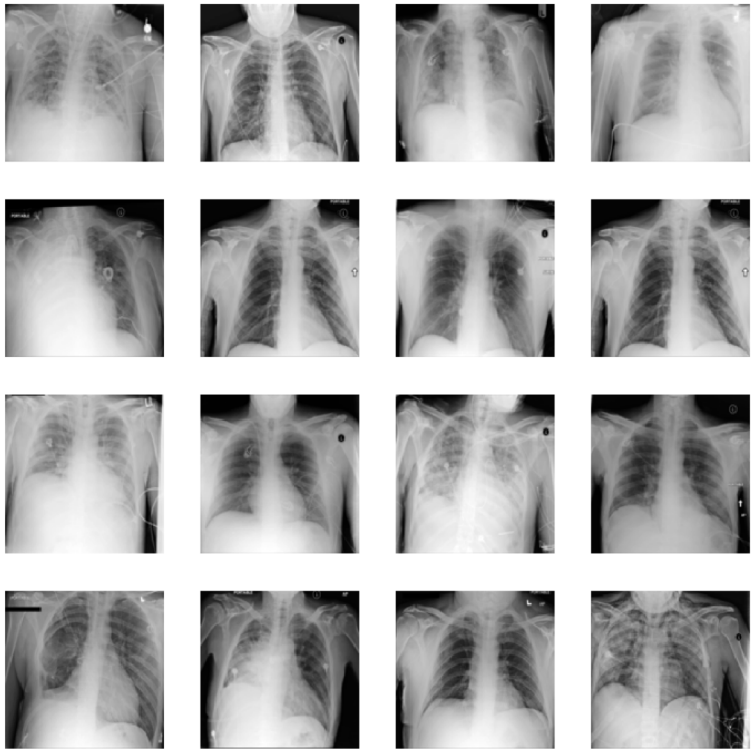
Source Code

- https://github.com/[illegible]XR8.py

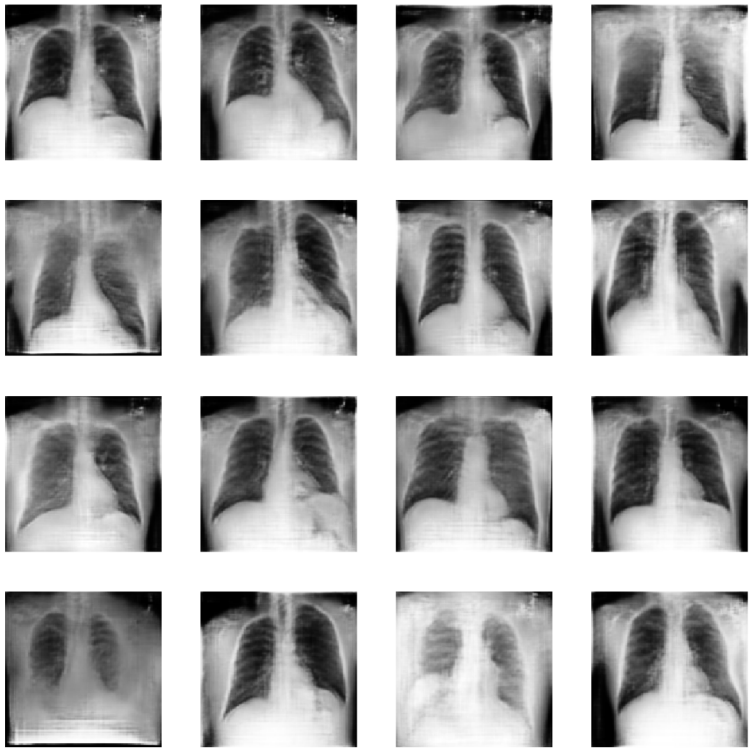
```
def generator(self):
    if self.G:
        return self.G
    self.G = Sequential()
    dropout = 0.2
    depth = 64+64+64+64
    dim = 8
    # In: 100
    # Out: dim x dim x depth
    self.G.add(Dense(dim*dim*depth, input_dim=100))
    self.G.add(BatchNormalization(momentum=0.9))
    self.G.add(Activation('relu'))
    self.G.add(Reshape((dim, dim, depth)))
    self.G.add(Dropout(dropout))

    self.G.add(UpSampling2D())
    self.G.add(Conv2DTranspose(int(depth), 5, padding='same'))
    self.G.add(BatchNormalization(momentum=0.9))
    self.G.add(Activation('relu'))

    self.G.add(UpSampling2D())
    self.G.add(Conv2DTranspose(int(depth/2), 5, padding='same'))
    self.G.add(BatchNormalization(momentum=0.9))
    self.G.add(Activation('relu'))

    self.G.add(UpSampling2D())
    self.G.add(Conv2DTranspose(int(depth/4), 5, padding='same'))
    self.G.add(BatchNormalization(momentum=0.9))
    self.G.add(Activation('relu'))

    self.G.add(UpSampling2D())
    self.G.add(Conv2DTranspose(int(depth/8), 5, padding='same'))
    self.G.add(BatchNormalization(momentum=0.9))
    self.G.add(Activation('relu'))

    # Out: 128 x 128 x 1 grayscale image [0.0,1.0] per pix
    self.G.add(Conv2DTranspose(1, 5, padding='same'))
    self.G.add(Activation('sigmoid'))
    self.G.summary()
    return self.G
```

Results (19,011 samples, 2,000 steps, 36 minutes)

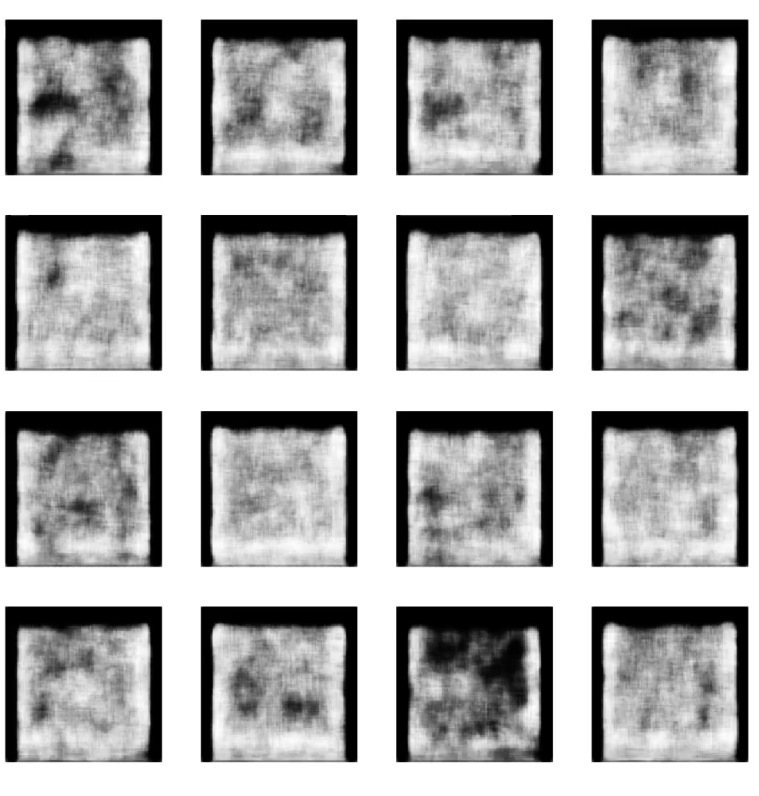
Real



Fake

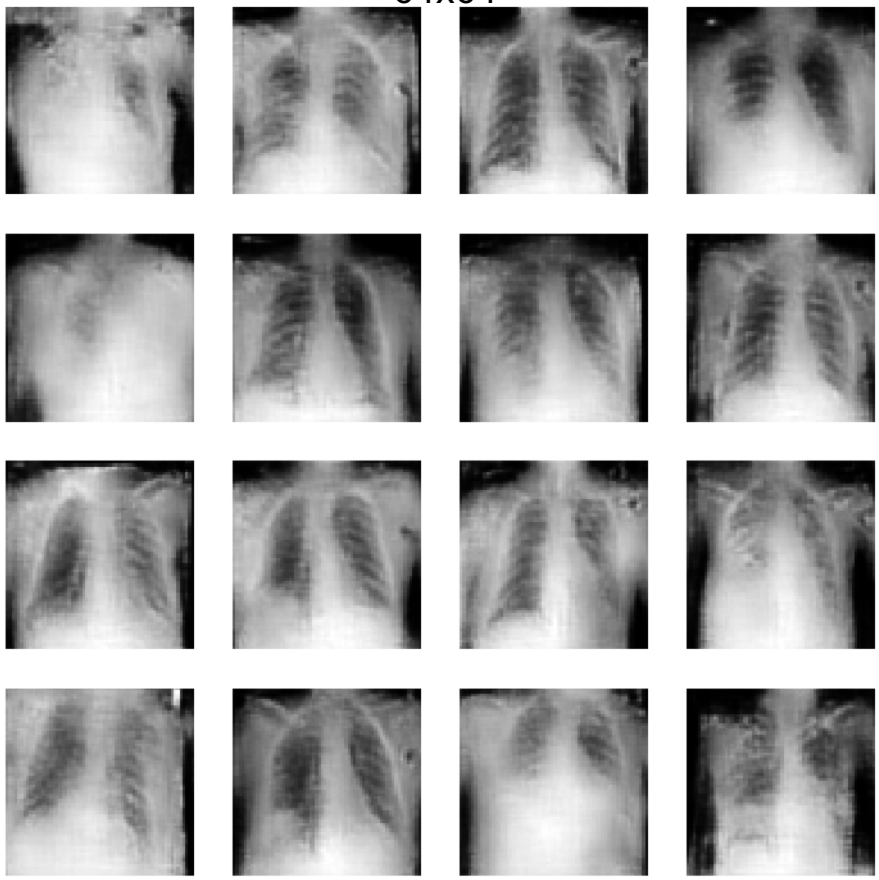


Training Snapshots Every 10 Steps



Conclusion

64x64



128x128