Neural networks & deep learning
GANs, recurrent networks, practical stuff
Lecture block 6

Anders Eklund

anders.eklund@liu.se

Department of Biomedical Engineering (IMT)
Department of Computer and Information Science (IDA)
Center for Medical Image Science and Visualization (CMIV)
Linköping University, Sweden

December 3, 2019

Outline - part 4 of 6

- Data augmentation, increase amount of training data
- Image synthesis, create completely new images
  - Noise-to-image GANs
  - Image-to-image GANs

Image synthesis / data augmentation

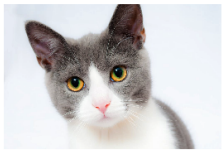
- Add more realistic training data to improve training
- Rotations / Mirroring
- Scaling
- Elastic (non-linear) deformations
- For "classical" image processing,
  features were designed to be invariant
  to scale and rotation (e.g. log polar transform)
- In deep learning, this is solved with data augmentation

Rotation

What happens if you train a network on 1 million cat images, and then show an upside down cat?

Add randomly rotated versions of all images as training data

Scale



What happens if you train a network on 1 million cat images, and then show a much smaller cat?

Add randomly scaled versions of all images as training data

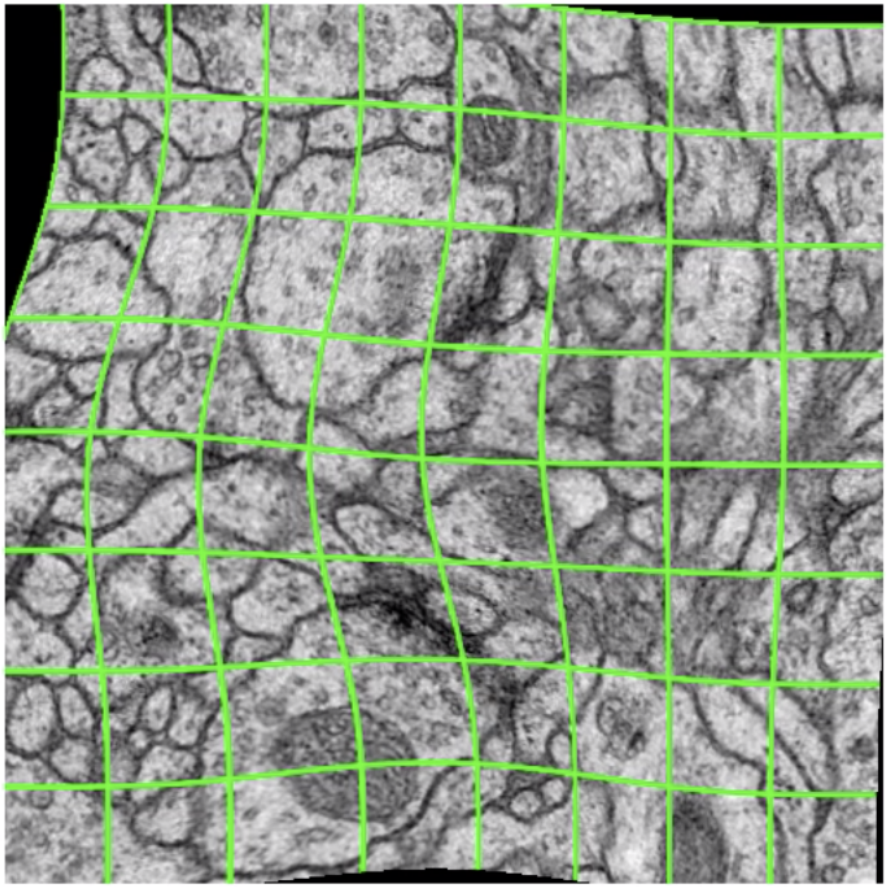
Quality

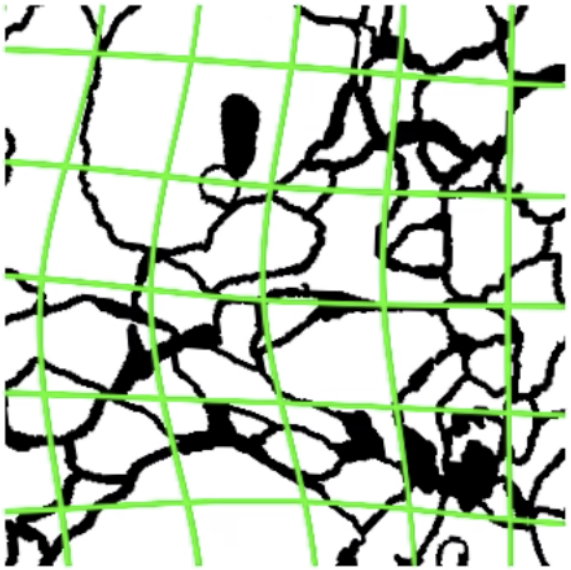
What happens if you train a network on 1 million sharp cat images, and then show a blurry cat?

Add blurred versions of all images as training data

Add noise to all training images

Elastic deformations





correspondingly deformed
manual labels

https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

On the fly augmentation in Keras

```
datagen = ImageDataGenerator(
    featurewise_std_normalization=True,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)
# compute quantities required for normalization
datagen.fit(x_train)
# fits the model on batches with real-time data augm.
model.fit_generator(datagen.flow(x_train, y_train,
                                 batch_size=32),
                                 steps_per_epoch=len(x_train) / 32,
                                 epochs=epochs)
```

See https://keras.io/preprocessing/image/ for details

No elastic deformations? No Keras function for 3D ?

Augmentation outside Keras

- https://github.com/MIC-DKFZ/batchgenerators

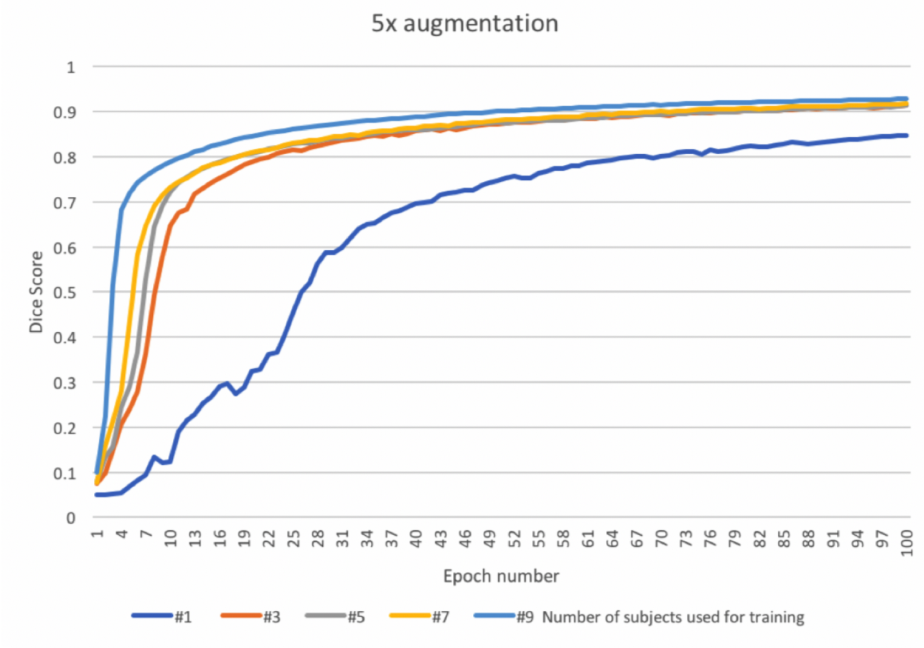
Suported Augmentations

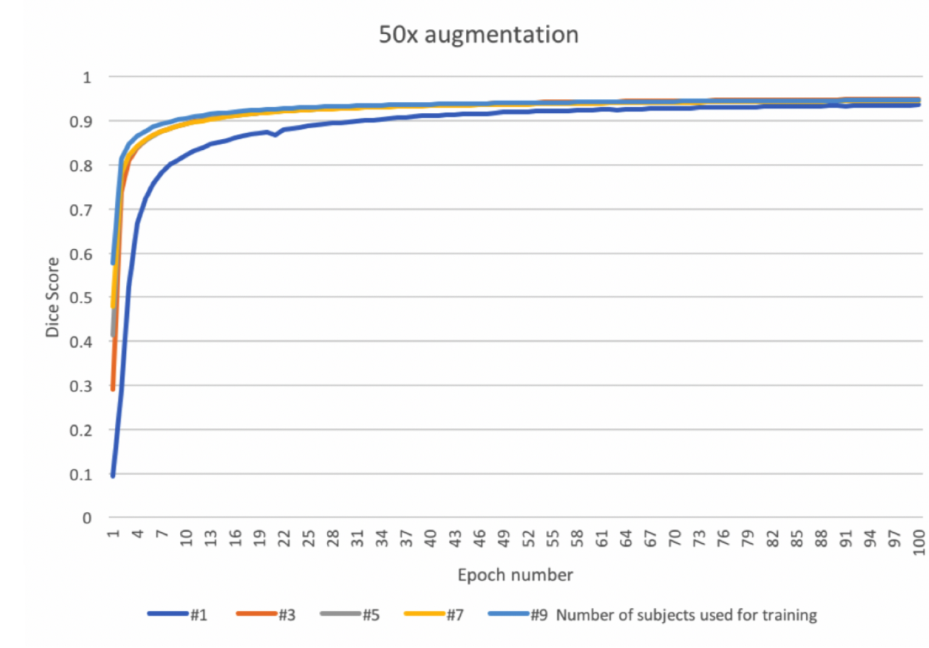
We supports a variety of augmentations, all of which are compatible with **2D and 3D input data!** (This is something that was missing in most other frameworks).

- **Spatial Augmentations**
  - mirroring
  - channel translation (to simulate registration errors)
  - elastic deformations
  - rotations
  - scaling
  - resampling
- **Color Augmentations**
  - brightness (additive, multiplivative)
  - contrast
  - gamma (like gamma correction in photo editing)
- **Noise Augmentations**
  - Gaussian Noise
  - Rician Noise
  - ...will be expanded in future commits
- **Cropping**
  - random crop
  - center crop
  - padding

-

Degree of augmentation





Gaonkar, Bilwaj, et al. Extreme Augmentation: Can deep learning based medical image segmentation be trained using a single manually delineated scan?, arXiv:1810.01621, 2018

QUIZ

Notice anything specific?

QUIZ

thispersondoesnotexist.com

Generative adversarial networks

- Standard data augmentation cannot create new images, can only modify existing images through rotations, scaling, ...
- Generative adversarial networks (GANs) can be trained to synthesize new images, given a large set of training images
- A GAN consists of a generator G (generates new images), and a discriminator D (classifies an image as real or fake)
- $$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$
- For images, both the generator and discriminator are CNNs
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).

GAN ARCHITECTURE

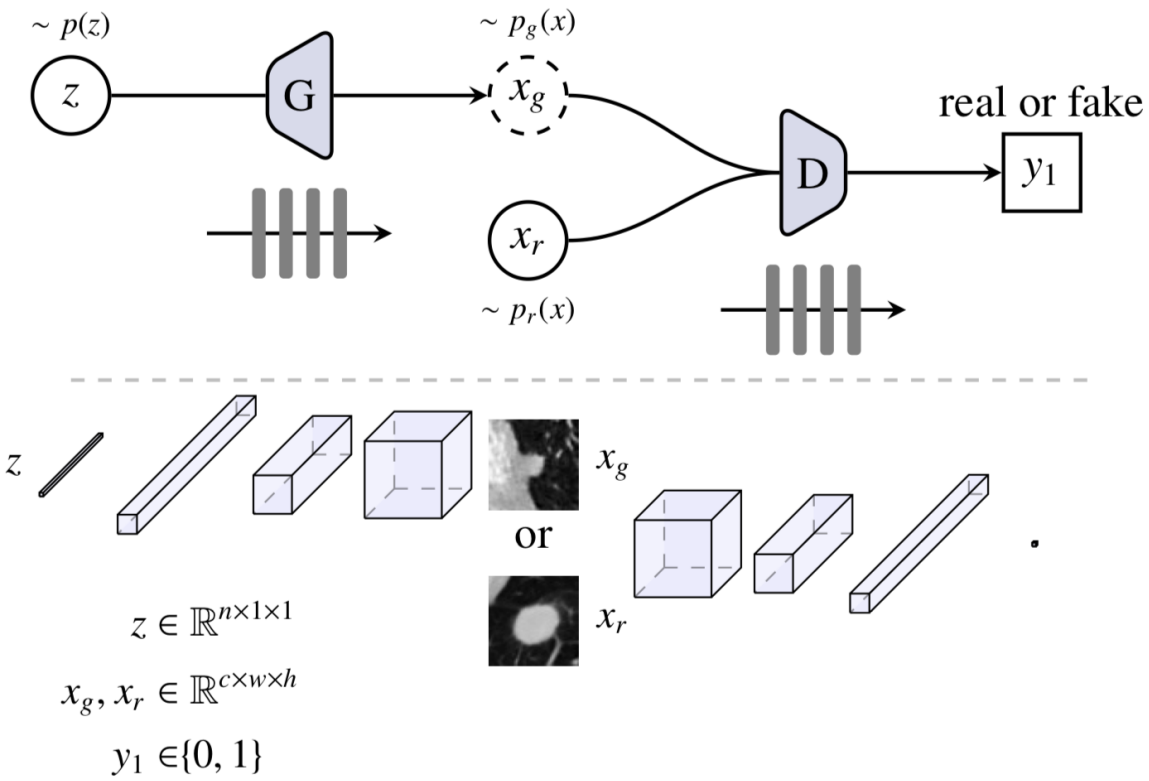
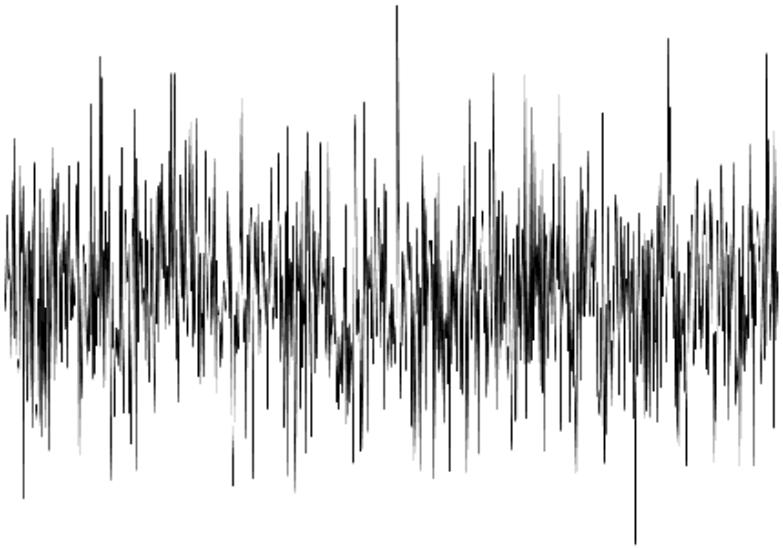


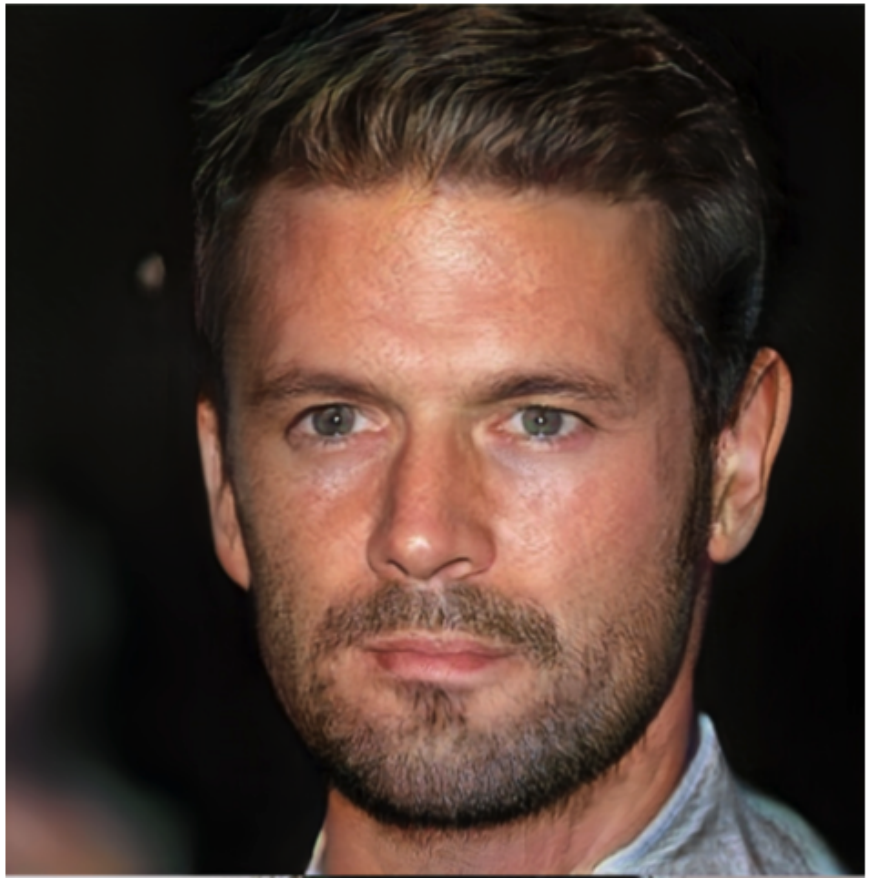
Figure 2: Schematic view of the vanilla GAN for synthesis of lung nodule on CT images. Top of the figure shows the network configuration. The part below shows the input, output and the internal feature representations of the generator G and discriminator D. G transforms a sample z from $p(z)$ into a generated nodule x_g. D is a binary classifier that differentiates the generated and real images of lung nodule formed by x_g and x_r respectively.

Yi, X., Walia, E., & Babyn, P. (2019). Generative adversarial network in medical imaging: A review, Medical Image Analysis

NOISE-TO-IMAGE GAN

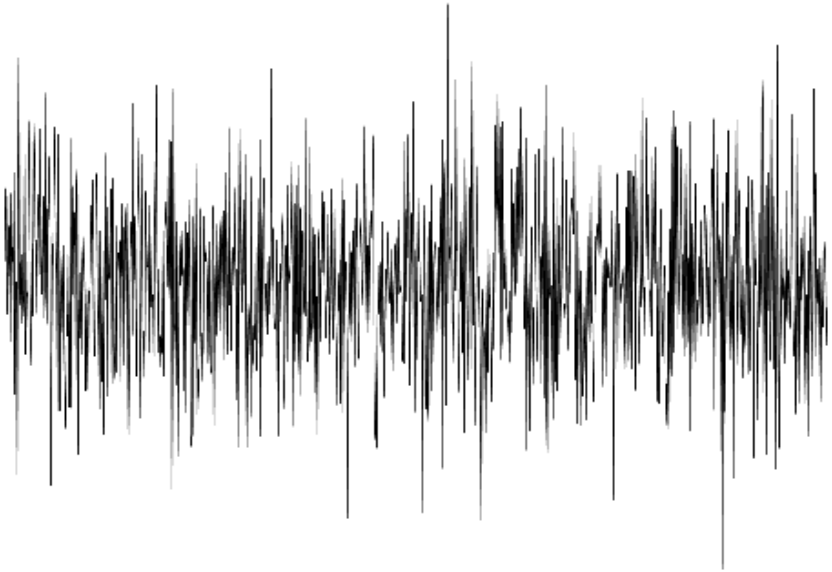


$\rightarrow$



Karras, T., Aila, T., Laine, S., & Lehtinen, J. Progressive growing of GANs for improved quality, stability, and variation, ICLR, 2018

NOISE-TO-IMAGE GAN



→

Karras, T., Aila, T., Laine, S., & Lehtinen, J. Progressive growing of GANs for improved quality, stability, and variation, ICLR, 2018

NOISE-TO-IMAGE GAN - PROGRESSIVE

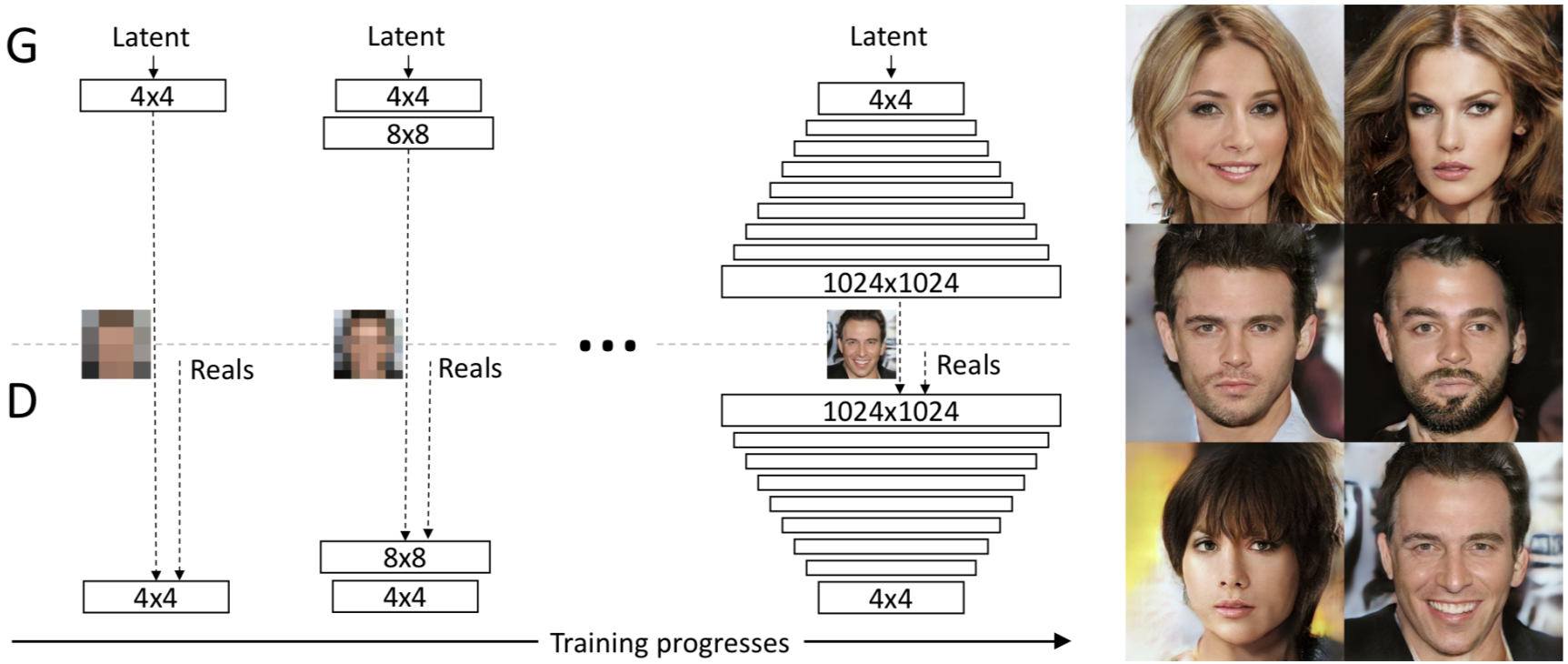


Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $\boxed{N \times N}$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at 1024×1024.

NOISE-TO-IMAGE GAN - VIDEOS

1 INTRODUCTION

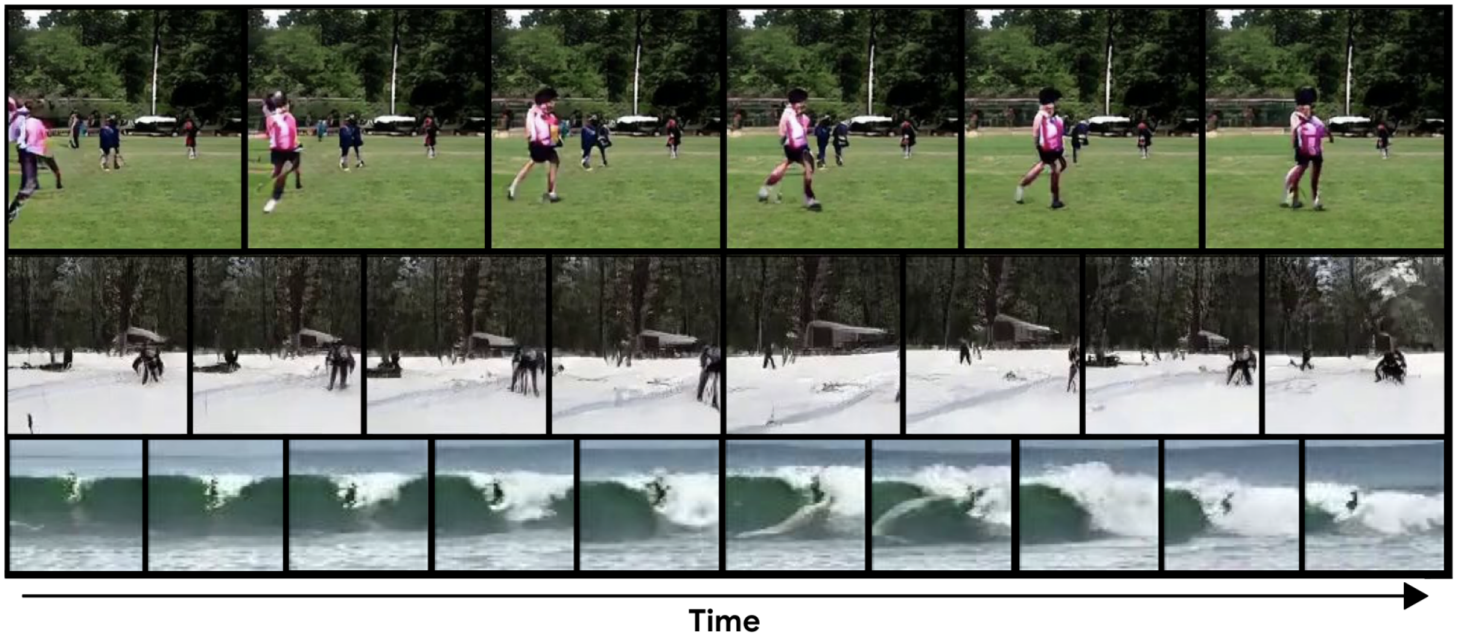


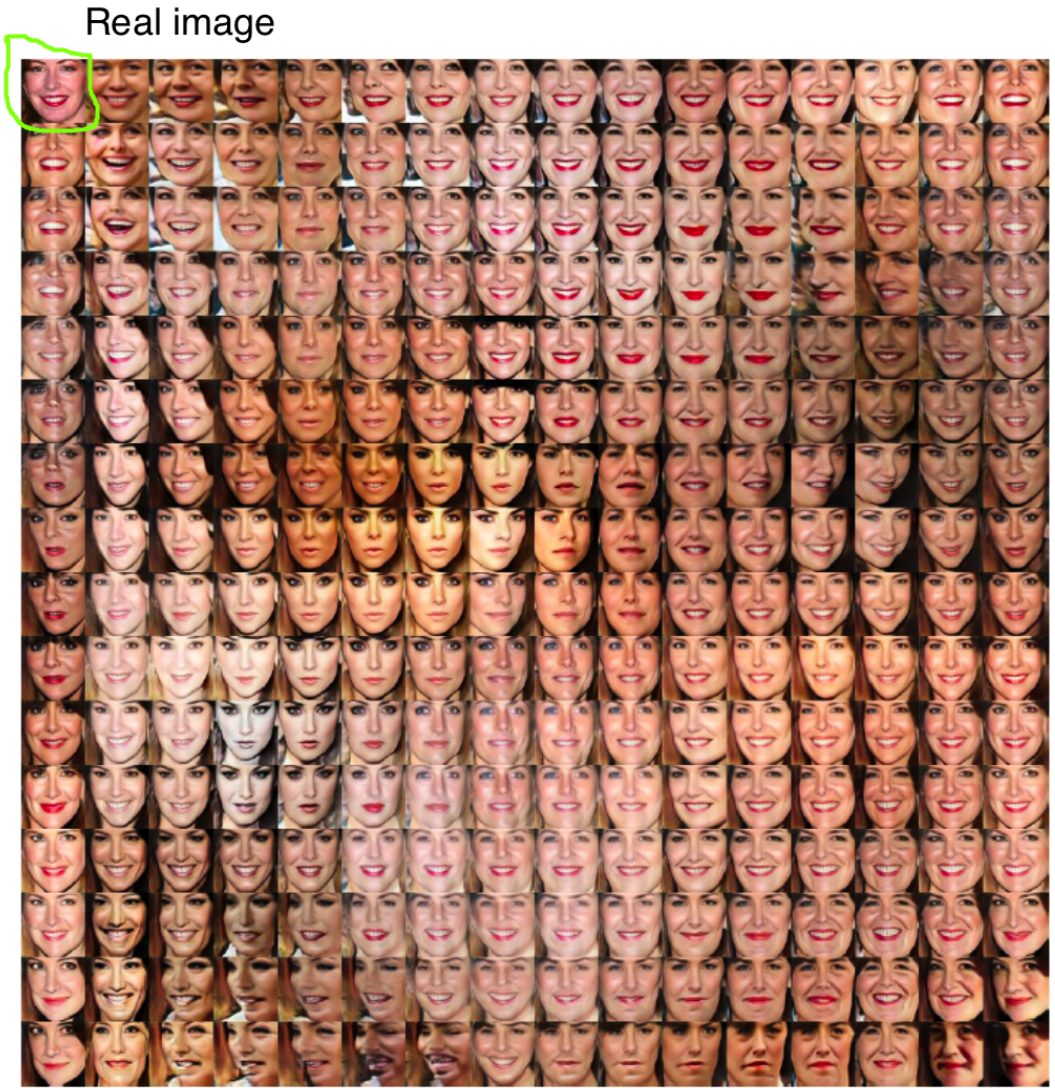
Figure 1: Selected frames from videos generated by a DVD-GAN trained on Kinetics-600 at 256×256, 128×128, and 64×64 resolutions (top to bottom).

- https://drive.google.com/file/d/1RGRVKCpVaG8z3p9GBCamRk4apiIR7jUc/view
- Clark, A., Donahue, J., & Simonyan, K. (2019). Efficient video generation on complex datasets. arXiv preprint arXiv:1907.06571.

NOISE-TO-IMAGE GAN - 3D

My own attempt at training a 3D GAN to synthesize fake brains
GDPR: Cannot share medical data from person A
Synthetic data does not belong to a specific person

GANS FOR DATA AUGMENTATION



Antoniou, A., Storkey, A., Edwards, H., Data Augmentation Generative Adversarial Networks, arXiv:1711.04340

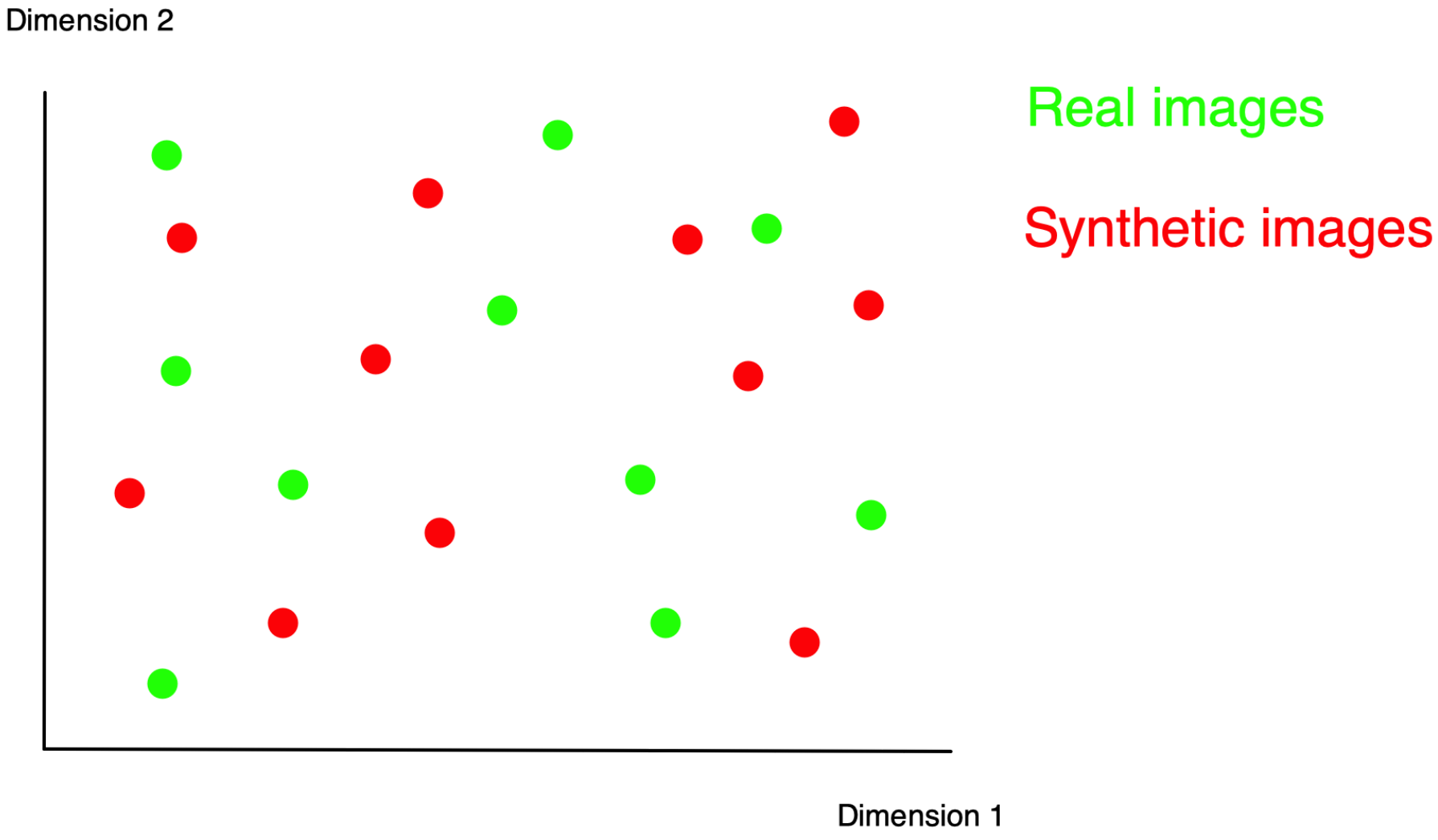
GANS FOR DATA AUGMENTATION

Omniglot DAGAN Augmented Classification		
Experiment ID	**Samples Per Class**	**Test Accuracy**
Omni_5_Standard	5	0.689904
Omni_5_DAGAN_Augmented	5	**0.821314**
Omni_10_Standard	10	0.794071
Omni_10_DAGAN_Augmented	10	**0.862179**
Omni_15_Standard	15	0.819712
Omni_15_DAGAN_Augmented	15	**0.874199**
EMNIST DAGAN Augmented Classification		
Experiment ID	**Samples Per Class**	**Test Accuracy**
EMNIST_Standard	15	0.739353
EMNIST_DAGAN_Augmented	15	**0.760701**
EMNIST_Standard	25	0.783539
EMNIST_DAGAN_Augmented	25	**0.802598**
EMNIST_Standard	50	0.815055
EMNIST_DAGAN_Augmented	50	**0.827832**
EMNIST_Standard	100	0.837787
EMNIST_DAGAN_Augmented	100	**0.848009**
Face DAGAN Augmented Classification		
Experiment ID	**Samples Per Class**	**Test Accuracy**
VGG-Face_Standard	5	0.0446948
VGG-Face_DAGAN_Augmented	5	**0.125969**
VGG-Face_Standard	15	0.39329
VGG-Face_DAGAN_Augmented	15	**0.429385**
VGG-Face_Standard	25	0.579942
VGG-Face_DAGAN_Augmented	25	**0.584666**

Table 1: Vanilla Classification Results: All results are averages over 5 independent runs. The DAGAN augmentation improves the classifier performance in all cases. Test accuracy is the result on the test cases in the test domain

Antoniou, A., Storkey, A., Edwards, H., Data Augmentation Generative Adversarial Networks, arXiv:1711.04340

DATA AUGMENTATION - LOW DIM PLOT



GENERAL

- GANs are not only for images
- GANs can be used to synthesize virtually any data
- https://magenta.tensorflow.org/gansynth

GANSynth: Making music with GANs

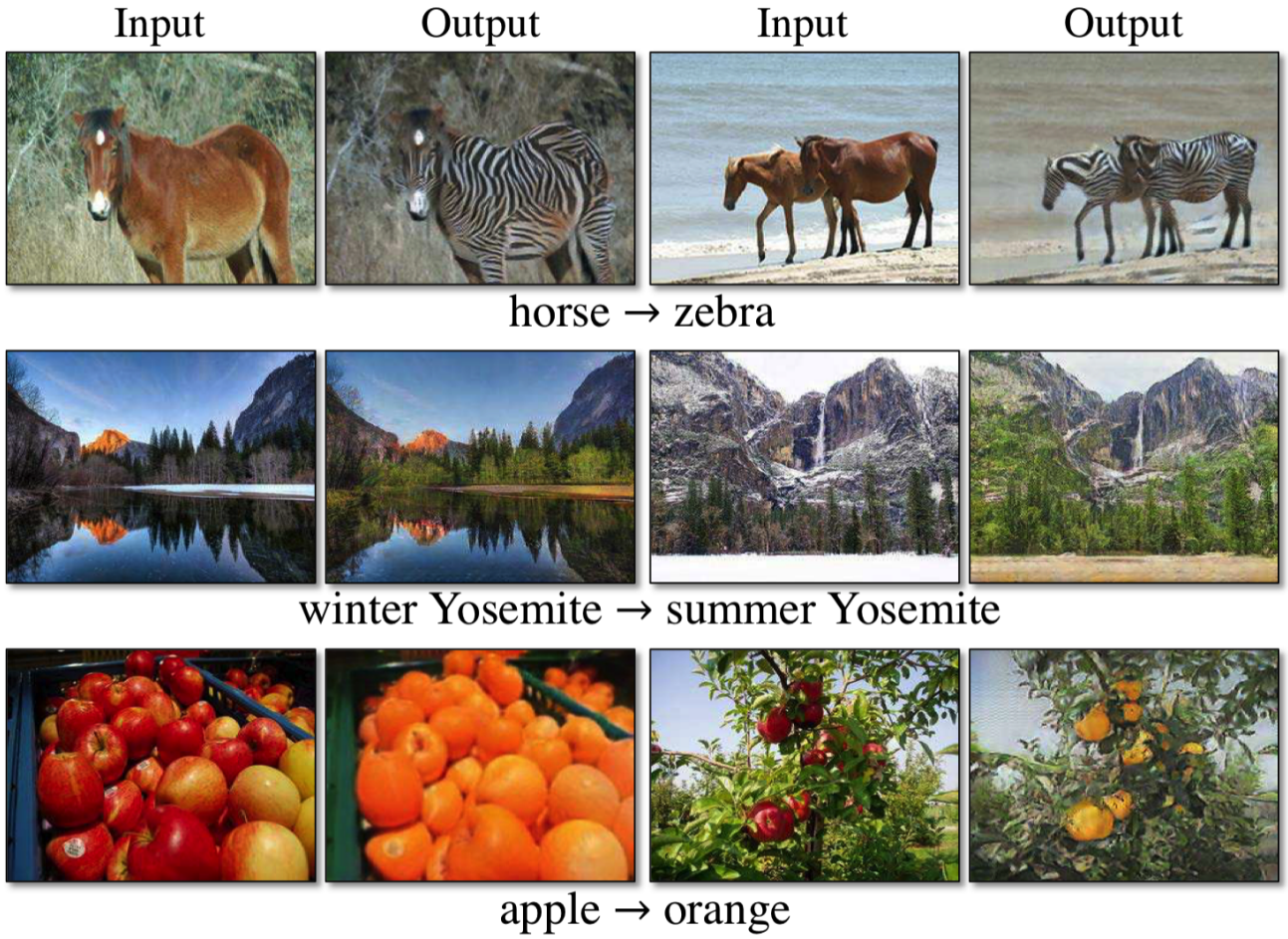
Feb 25, 2019
• Jesse Engel jesseengel jesseengel

In this post, we introduce GANSynth, a method for generating high-fidelity audio with Generative Adversarial Networks (GANs).

Colab Notebook	Audio Examples	ICLR 2019 Paper	GitHub Code

-

IMAGE-TO-IMAGE GAN (CONDITIONAL GANS)



Zhu et al., Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV, 2017

IMAGE-TO-IMAGE GAN - IN SNAPCHAT

VRSCOUT

NEWS

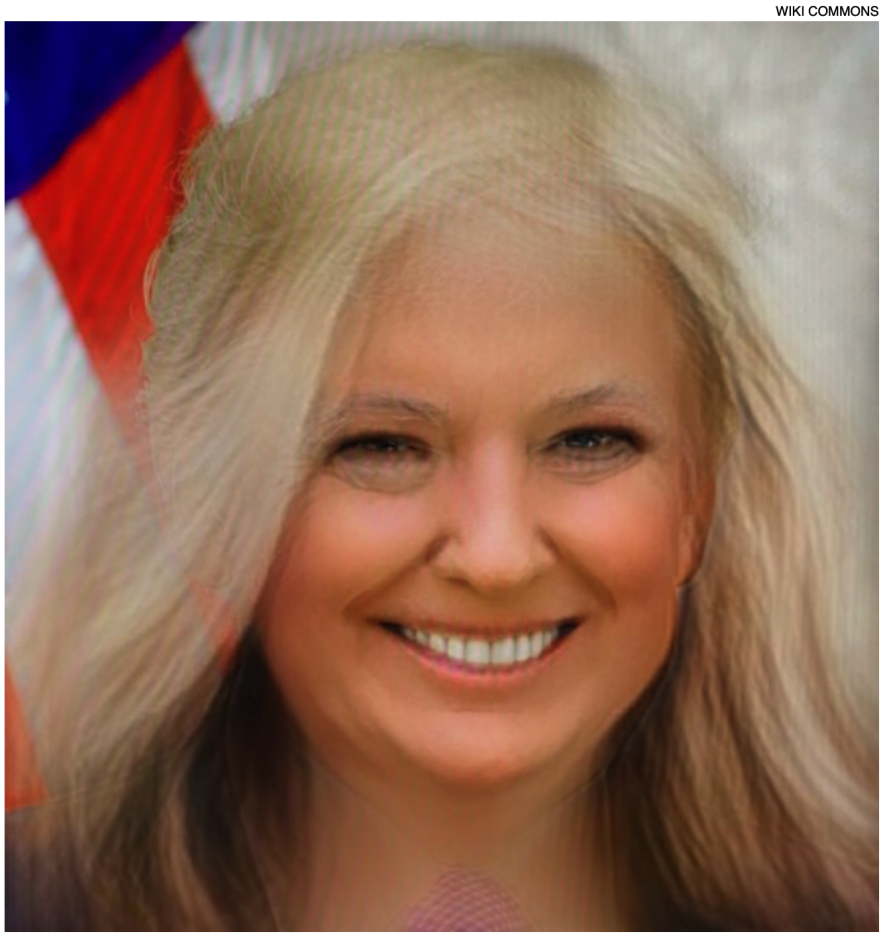
News

Snapchat's Gender Swap AR Face Filter Is A Big Hit

May 13, 2019 • by Bobby Carlton

https://vrscout.com/news/snapchat-gender-swap-ar-face-filter/

IMAGE-TO-IMAGE GAN - IN SNAPCHAT



https://www.pocket-lint.com/apps/news/snapchat/148095-snapchat-gender-swap-how-to-do-it-and-check-out-these-crazy-celebrity-results

APPLICATIONS - DAY TO NIGHT

Figure 5.15: *Synthetic results from the CycleGAN baseline model on 256x256 pixel images. The model is trained using the large day and night dataset. Top and bottom results are from city and open road street view environments.*

S. Karlsson, P. Welander, Generative Adversarial Networks for Image-to-Image Translation on Street View and MR Images, LIU-IMT-TFK-A—18/554—SE, 2018

APPLICATIONS - MR TO CT

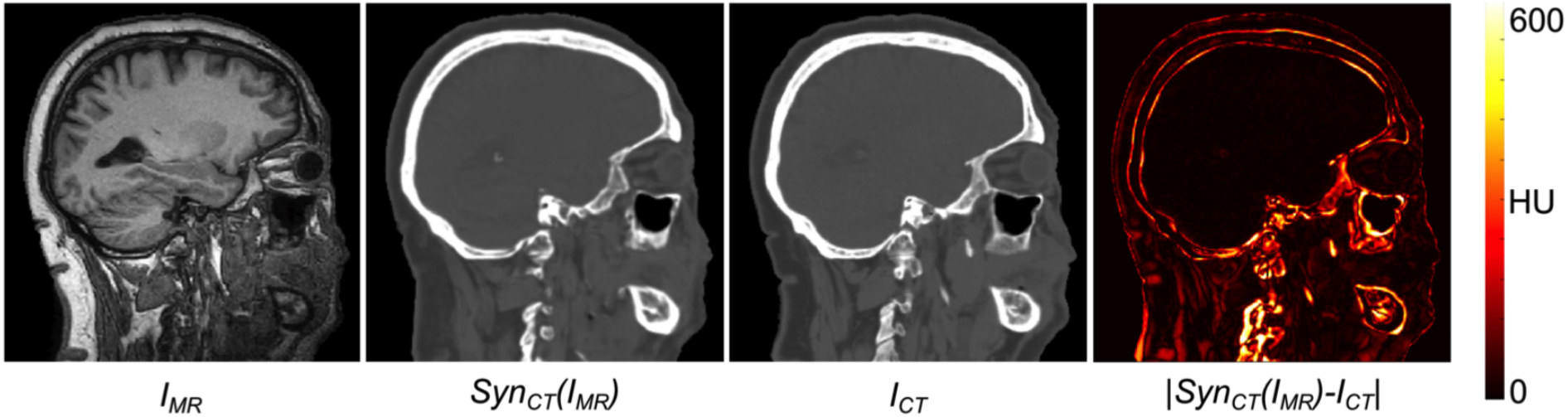


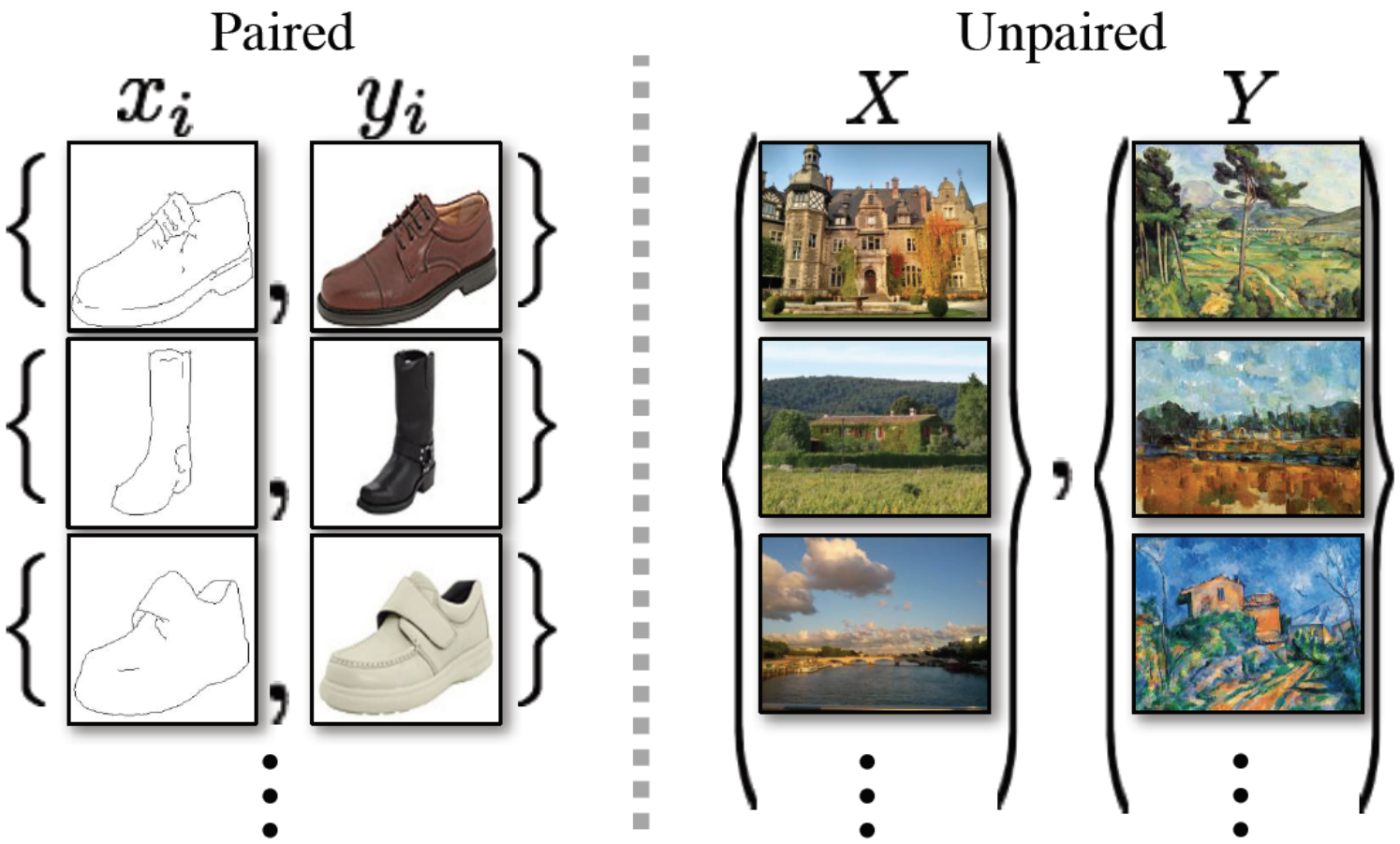
Fig. 4: *From left to right* Input MR image, synthesized CT image, reference real CT image, and absolute error between real and synthesized CT image.

Wolterink et al., Deep MR to CT synthesis using unpaired data. International Workshop on Simulation and Synthesis in Medical Imaging, 2017

SUPERVISED VS UNSUPERVISED GANS

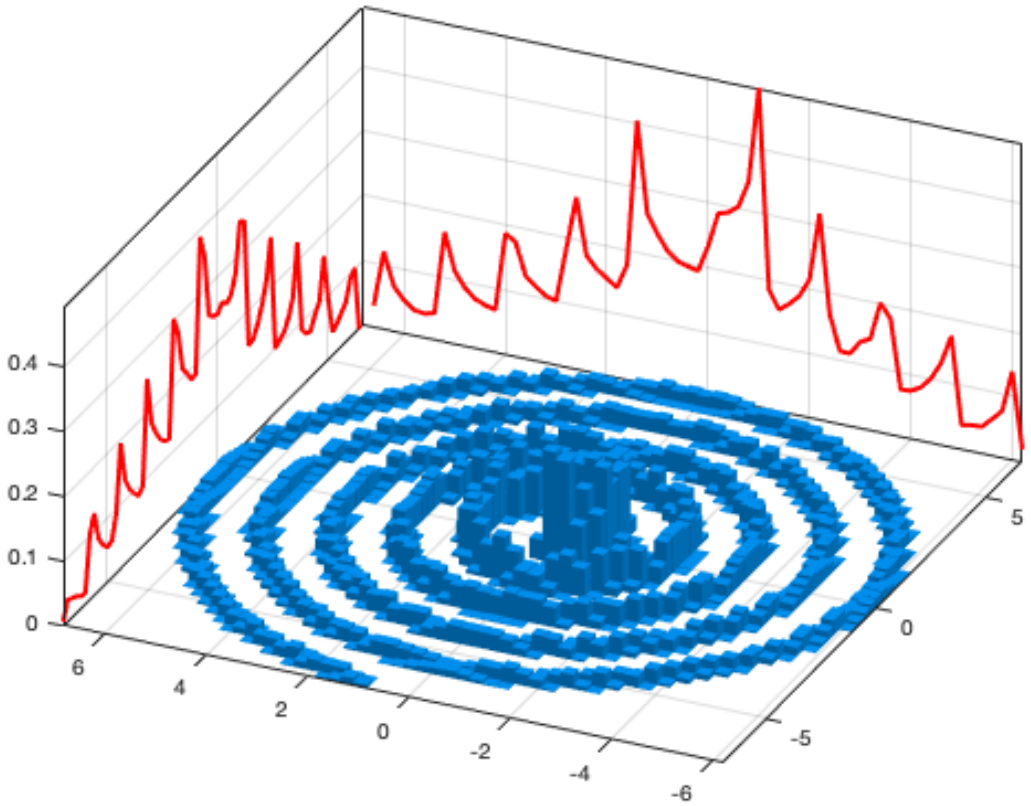
- We have images from two domains, X & Y
- We can train a GAN to convert from domain X to domain Y
  - If we have every image represented in both domains (x1; y1); (x2; y2); we can train a *supervised* GAN, which learns from the joint distribution of the data in both domains
  - If we have unmatched images in both domains (x1; ?); (?; y2); we can train an *unsupervised* GAN, which learns from the marginal distributions of the data in both domains.

PAIRED AND UNPAIRED DATA



Zhu et al., Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV, 2017

SUPERVISED VS UNSUPERVISED GANS

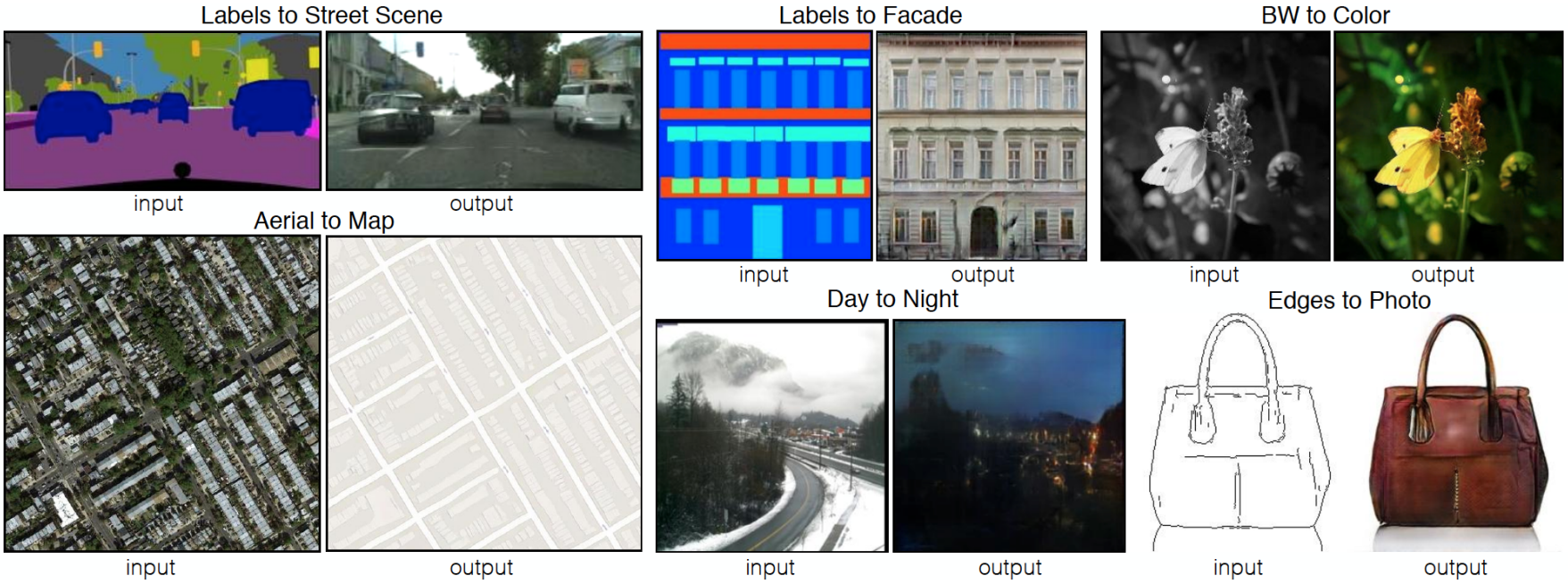


Supervised GAN: Learn the blue distribution
from paired examples from the blue distribution

Unsupervised GAN: Learn the blue distribution
from examples from the red distributions

PAIRED TRAINING - PIX2PIX

- Given images in two domains, X and Y, the GAN learns to convert an image in domain X to domain Y
- Drawback: We need paired (and registered) training images



Isola et al., Image-to-Image Translation with Conditional Adversarial Networks, CVPR, 2017

UNPAIRED TRAINING - CYCLEGAN

- Unsupervised image-to-image translation is more difficult
- Learning a joint distribution from marginal distributions is a vastly underdetermined problem, need regularization.
- One solution: CycleGAN. Learn transformations in both directions $X \leftrightarrow Y$ and impose a consistency constraint in forward-backward conversion
- $$\begin{cases} x_i \to \widehat{y_i} \to \widetilde{x_i} \approx x_i \\ y_i \to \widehat{x_i} \to \widetilde{y_i} \approx y_i \end{cases}$$

Loss functions for training

- Mappings $G : X->Y$ and $F : Y->X$, discriminators D_X and D_Y
- Adversarial loss function (GAN), L_{GAN}

  $$L_{GAN}(G, D_Y, X, Y) = E\left[logD_Y(y)\right] + E\left[log(1 - D_Y(G(x))\right]$$

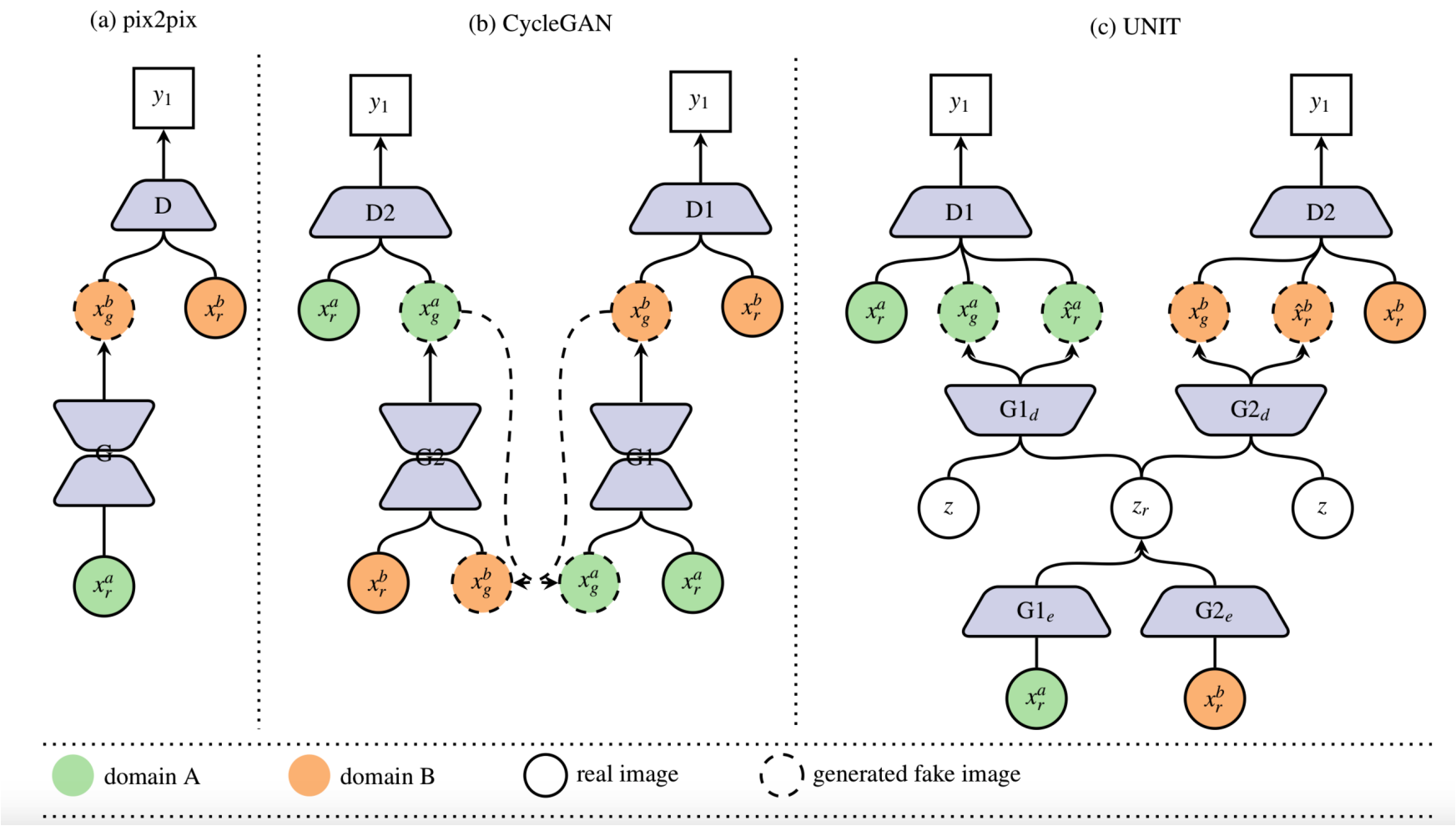
- Cycle consistency loss function (CycleGAN), L_{cyc}

  $$L_{cyc}(G, F) = E\left[||F(G(x)) - x||_1\right] + E\left[||G(F(y)) - y||_1\right]$$

- Full loss function for CycleGAN

  $$L(G, F, D_X, D_Y) = \\ L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + \lambda L_{cyc}(G, F)$$

PIX2PIX VS CYCLEGAN VS UNIT



Yi, X., Walia, E., & Babyn, P. (2018). Generative adversarial network in medical imaging: A review. Medical Image Analysis

CycleGAN - generator

- What does the generator look like in Keras?

Generator architectures We adopt our architectures from Johnson et al. [23]. We use 6 residual blocks for 128×128 training images, and 9 residual blocks for 256×256 or higher-resolution training images. Below, we follow the naming convention used in the Johnson et al.'s Github repository.

Let `c7s1-k` denote a 7×7 Convolution-InstanceNorm-ReLU layer with k filters and stride 1. `dk` denotes a 3×3 Convolution-InstanceNorm-ReLU layer with k filters and stride 2. Reflection padding was used to reduce artifacts. `Rk` denotes a residual block that contains two 3×3 convolutional layers with the same number of filters on both layer. `uk` denotes a 3×3 fractional-strided-Convolution-InstanceNorm-ReLU layer with k filters and stride $\frac{1}{2}$.

The network with 6 residual blocks consists of:
`c7s1-64,d128,d256,R256,R256,R256,R256,R256,R256,u128,u64,c7s1-3`

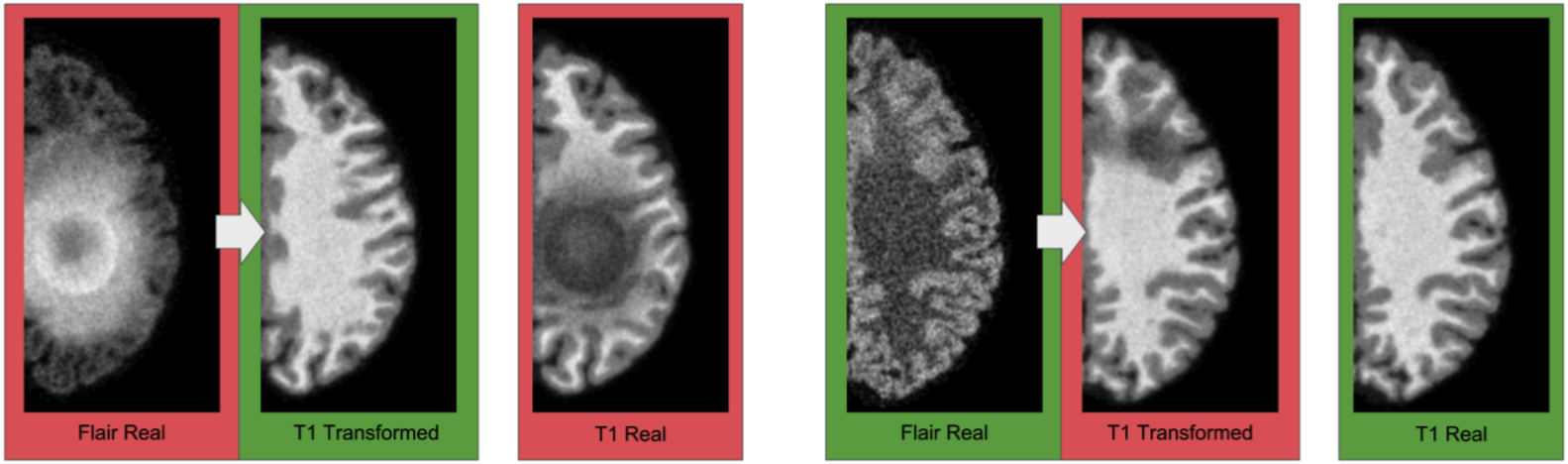
The network with 9 residual blocks consists of:
`c7s1-64,d128,d256,R256,R256,R256,R256,R256,R256,R256,R256,R256,u128 u64,c7s1-3`

```python
def modelGenerator(self, name=None):
    # Specify input
    input_img = Input(shape=self.img_shape)
    # Layer 1
    x = ReflectionPadding2D((3, 3))(input_img)
    x = self.c7Ak(x, 32)
    # Layer 2
    x = self.dk(x, 64)
    # Layer 3
    x = self.dk(x, 128)

    # Layer 4-12: Residual layer
    for _ in range(4, 13):
        x = self.Rk(x)

    # Layer 13
    x = self.uk(x, 64)
    # Layer 14
    x = self.uk(x, 32)
    x = ReflectionPadding2D((3, 3))(x)
    x = Conv2D(self.channels, kernel_size=7, strides=1)(x)
    x = Activation('tanh')(x)  # They say they use Relu but really they do not
    return Model(inputs=input_img, outputs=x, name=name)
```

- https://github.com/simontomaskarlsson/CycleGAN-Keras

CHALLENGES - TRAINING ON CONTROLS

- If an image-to-image GAN is trained on healthy controls, can it be used for diseased subjects?



(a) A translation removing tumors (b) A translation adding tumors

-

- Cohen, J. P., Luck, M., & Honari, S., Distribution matching losses can hallucinate features in medical image translation. In International Conference on Medical Image Computing and Computer-Assisted Intervention, 2018

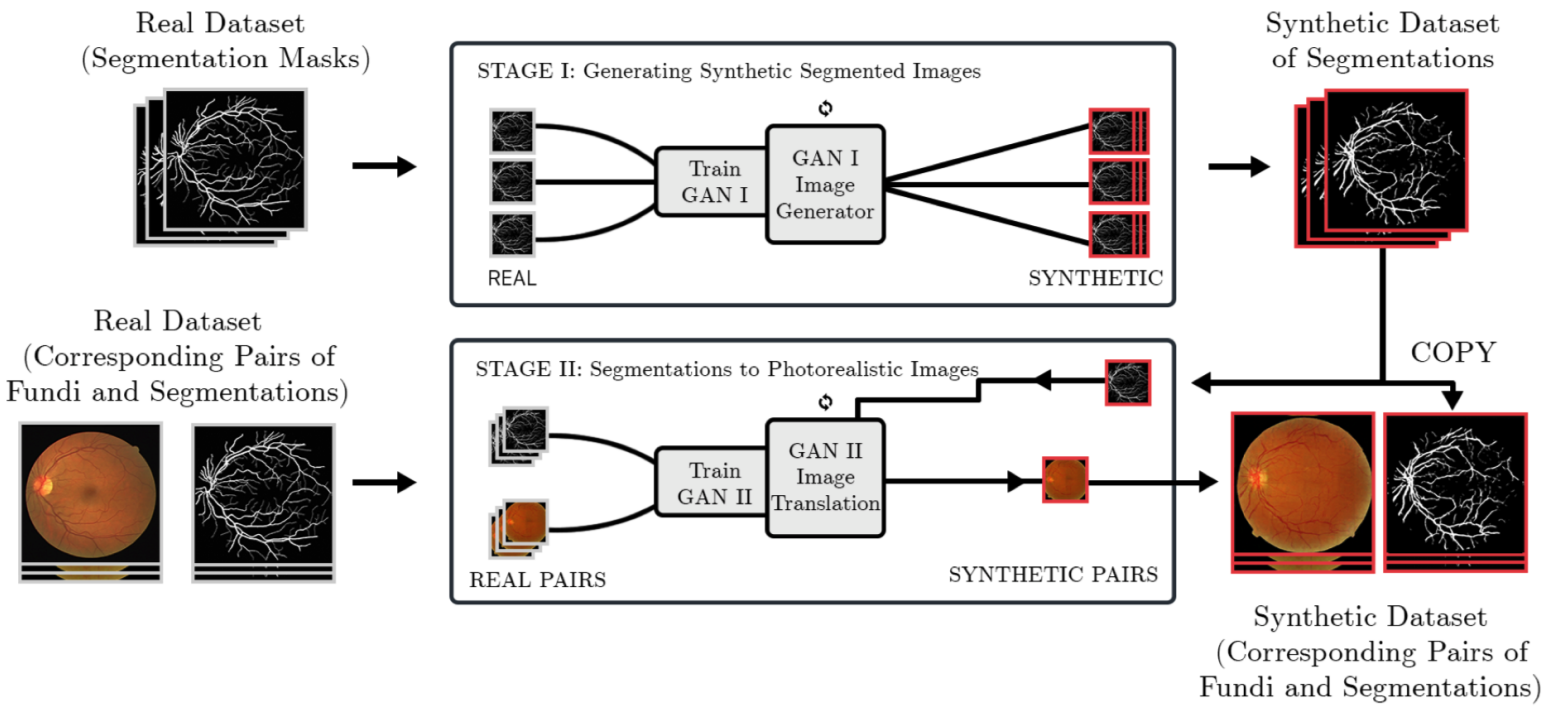
More advanced - combining GANs

- Difficult to synthesize realistic data from scratch (image-to-image GANs are normally easier / more stable to train compared to noise-to-image GANs)
- Use noise-to-image GAN to synthesize object of interest (as a (small) binary image / volume)
- Use image-to-image GAN to transform binary image / volume into a realistic dataset

More advanced - combining GANs

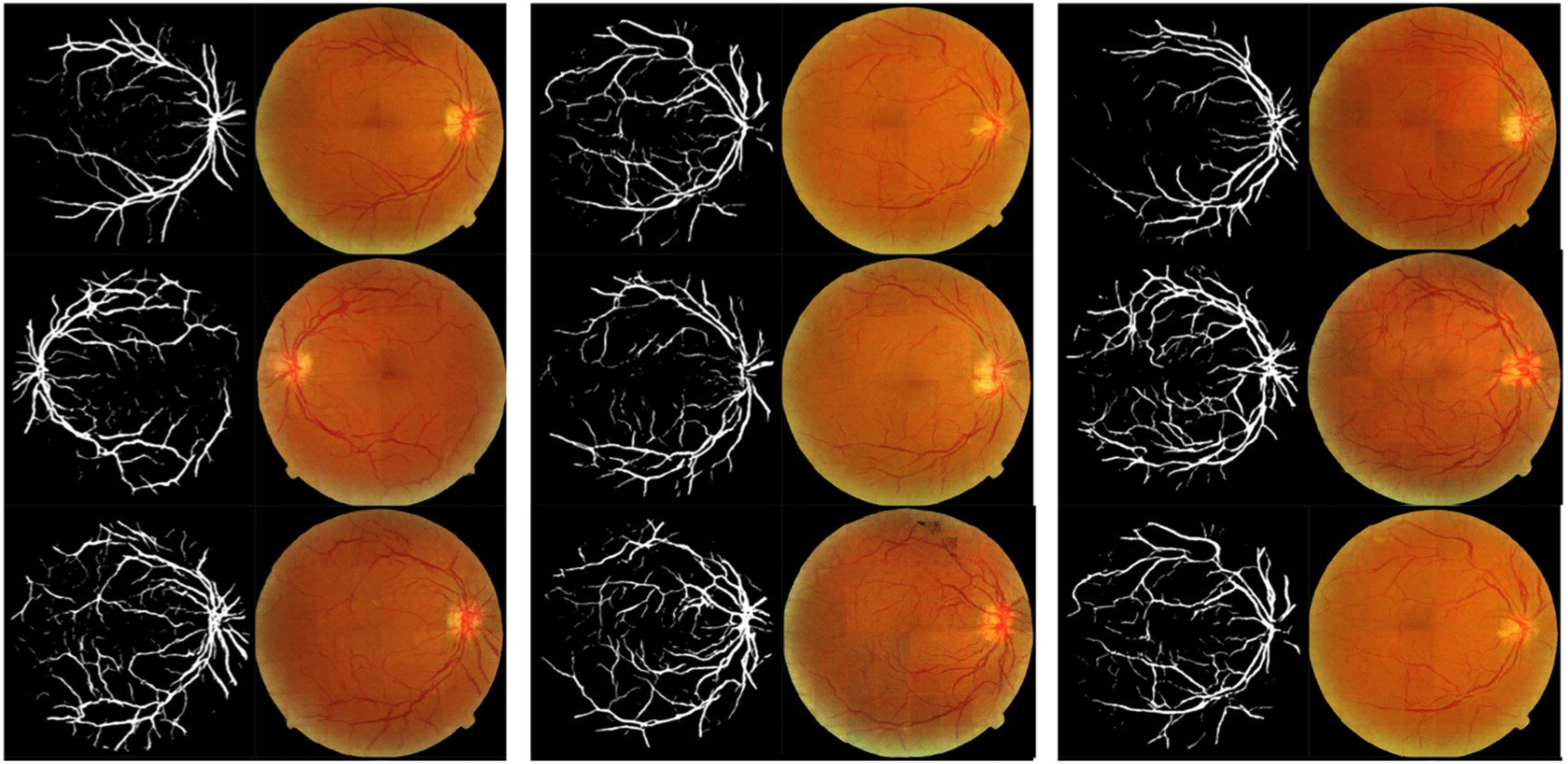
1. Stage-I GAN: Produce segmentation masks that represent the variable geometries of the dataset.
2. Stage-II GAN: Translate the masks produced in Stage-I to photorealistic images.

We illustrate the process with retinal fundi images.



Guibas, J. T., Virdi, T. S., & Li, P. S. Synthetic medical images from dual generative adversarial networks. arXiv:1709.01872.

More advanced - combining gans



Guibas, J. T., Virdi, T. S., & Li, P. S. Synthetic medical images from dual generative adversarial networks. arXiv:1709.01872.

More advanced - collagan

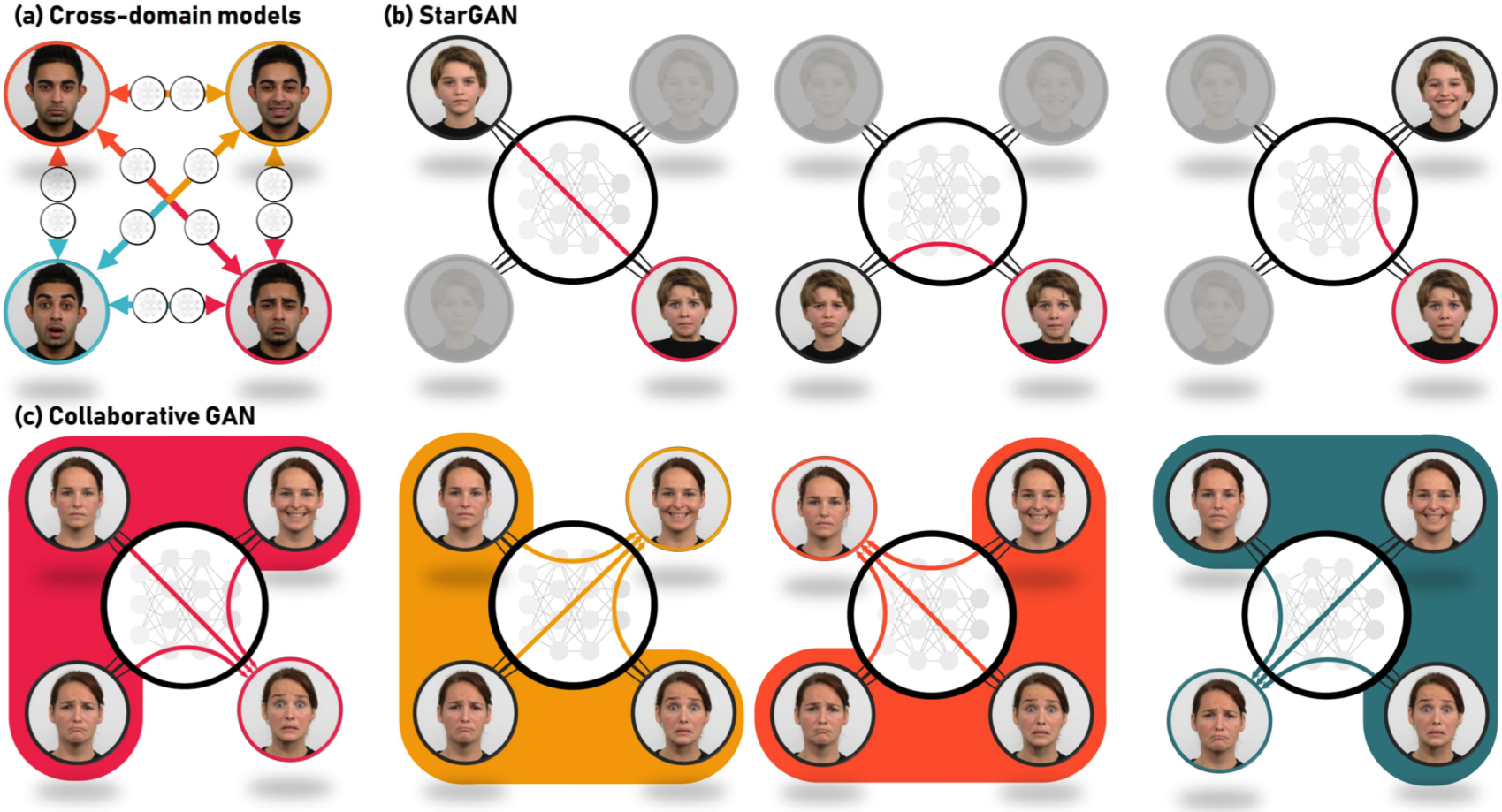


Figure 1: Image translation tasks using (a) cross-domain models, (b) StarGAN, and (c) the proposed collaborative GAN (CollaGAN). Cross-domain model needs large number of generators to handle multi-class data. StarGAN and CollaGAN use a single generator with one input and multiple inputs, respectively, to synthesize the target domain image.

Lee, D., Kim, J., Moon, W. J., & Ye, J. C. (2019). CollaGAN: Collaborative GAN for missing image data imputation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2487-2496).

Outline - part 5 of 6

- Recurrent neural networks (RNNs)
  - Applications
  - Training
- Bidirectional RNNs
- LSTM (long short term memory)

Recurrent neural networks

- RNNs are designed for processing sequential data
- Time sequences
  - Speech
  - Videos
  - Time series (economy, weather, ...)
- Timeless sequences
  - Text
  - Genetic data

Recurrent neural networks

- Applications of RNNs
  - Machine translation (e.g. Swedish to English)
  - Handwriting recognition
  - Speech recognition
  - Time series prediction
  - Speech synthesis

Recurrent neural networks

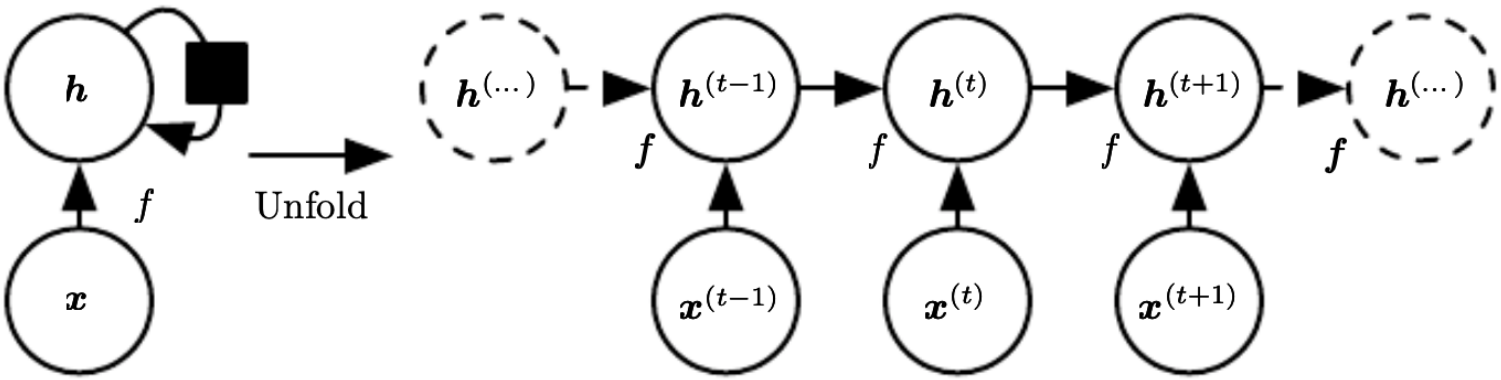


Figure 10.2: A recurrent network with no outputs. This recurrent network just processes information from the input $\boldsymbol{x}$ by incorporating it into the state $\boldsymbol{h}$ that is passed forward through time. *(Left)* Circuit diagram. The black square indicates a delay of a single time step. *(Right)* The same network seen as an unfolded computational graph, where each node is now associated with one particular time instance.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

Recurrent neural networks

- $h^{(t)} = f\big(h^{(t-1)}, x^{(t)}, \theta\big)$
- $h^{(t)}$ is the (hidden) state at time point t,
  can be seen as a (lossy) summary of past inputs up to t
- $x^{(t)}$ is the input at time point t
- θ are the parameters of the network,
  to be learned, used for all time steps

Recurrent neural networks - types

- Produce output at each time step (e.g. translating Swedish to English)
  - 1. Recurrent connections between hidden units (more powerful, harder to train)
  - 2. Recurrent connections only from output at one time step to hidden units in next time step (less powerful, easier to train)
- 3. Produce single output for entire sequence (e.g. classification), (using recurrent connections between hidden units)

RNN TYPE 1 - GRAPH

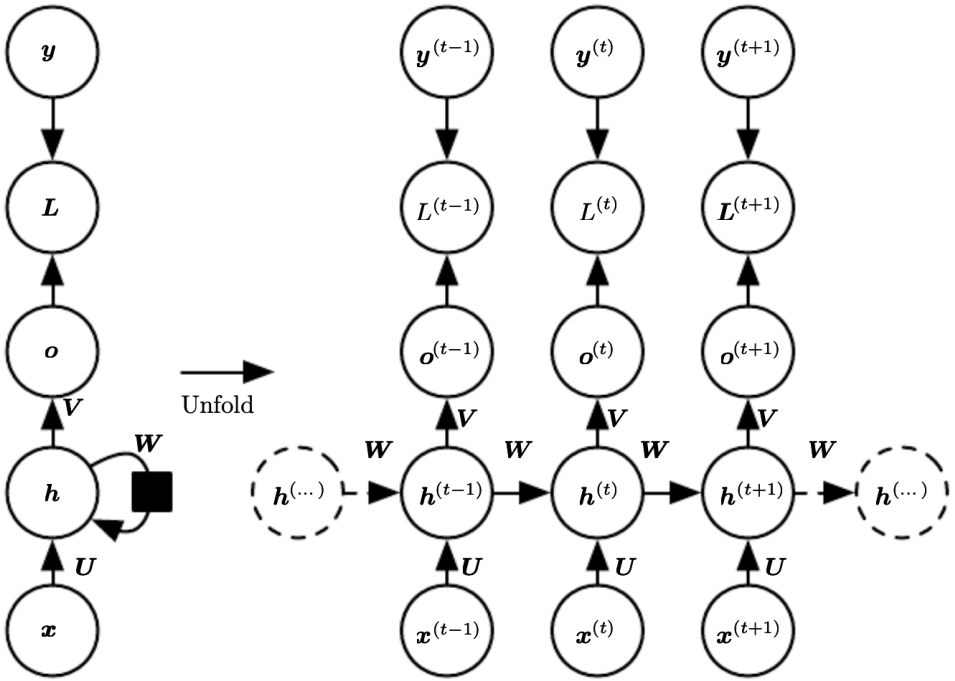


Figure 10.3: The computational graph to compute the training loss of a recurrent network that maps an input sequence of $\boldsymbol{x}$ values to a corresponding sequence of output $\boldsymbol{o}$ values. A loss L measures how far each $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. When using softmax outputs, we assume $\boldsymbol{o}$ is the unnormalized log probabilities. The loss L internally computes $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{o})$ and compares this to the target $\boldsymbol{y}$. The RNN has input to hidden connections parametrized by a weight matrix $\boldsymbol{U}$, hidden-to-hidden recurrent connections parametrized by a weight matrix $\boldsymbol{W}$, and hidden-to-output connections parametrized by a weight matrix $\boldsymbol{V}$. Equation 10.8 defines forward propagation in this model. *(Left)* The RNN and its loss drawn with recurrent connections. *(Right)* The same seen as a time-unfolded computational graph, where each node is now associated with one particular time instance.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

RNN type 1 - equations

- $a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$
- $h^{(t)} = tanh(a^{(t)})$
- $o^{(t)} = c + Vh^{(t)}$
- $\hat{y} = softmax(o^{(t)})$
- Bias vectors b and c
- Input-to-hidden weight matrix U
- Hidden-to-hidden weight matrix W
- Hidden-to-output weight matrix V

RNN TYPE 3

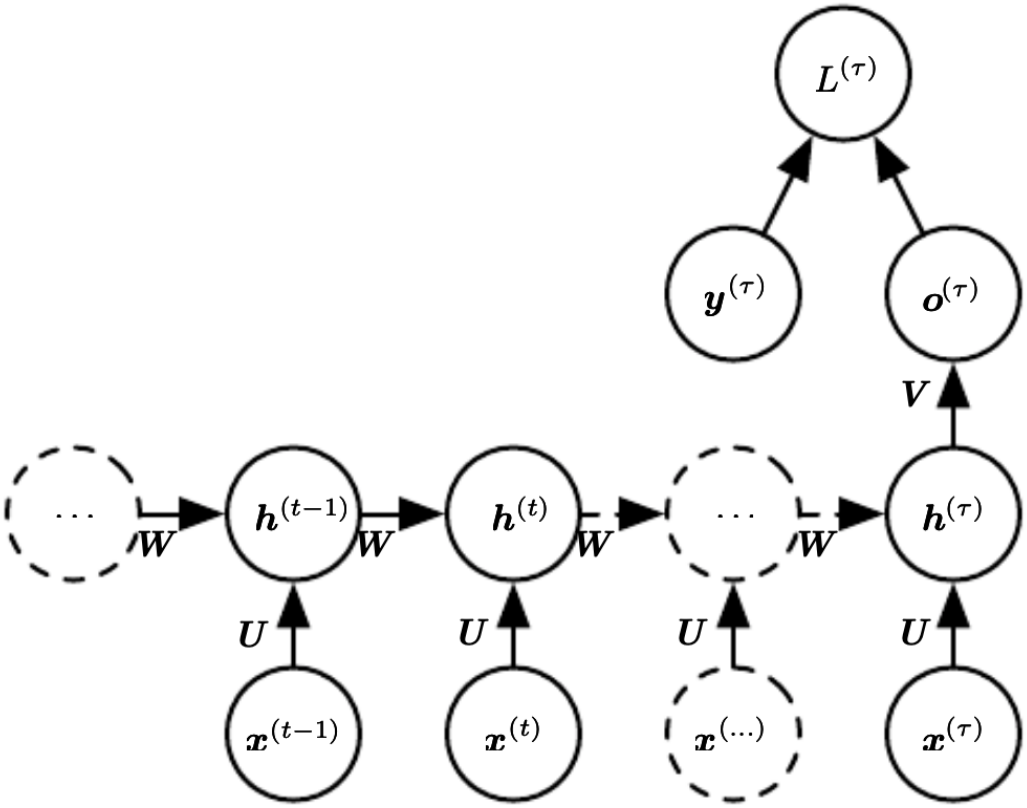


Figure 10.5: Time-unfolded recurrent neural network with a single output at the end of the sequence. Such a network can be used to summarize a sequence and produce a fixed-size representation used as input for further processing. There might be a target right at the end (as depicted here), or the gradient on the output $\boldsymbol{o}^{(t)}$ can be obtained by back-propagating from further downstream modules.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

Training RNNs

- RNNs can be trained using backpropagation.
  for RNNs called backpropagation through time
  (unrolled RNN is a feedforward network)
- Standard gradient based optimization algorithms do
  not perform very well (vanishing or exploding gradients)
- Forward pass in backpropagation,
  contains activation functions to limit activations
- Backward pass is linear, no limits, need other ways
  to prevent very large or very small gradients
  (weight scaling, clipping, more advanced optimization)

RNNs in Keras

```
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN

model = Sequential()
model.add(SimpleRNN(units=32, input_shape=(1,step),
                    activation="relu"))
model.add(Dense(8, activation="relu"))
model.add(Dense(1))
model.compile(loss='mean_squared_error',
              optimizer='rmsprop')
```

See https://keras.io/layers/recurrent/ for details

Bidirectional RNNs

- In some applications, a correct classification depends on previous and future data
- Bidirectional RNNs, one RNN moves forward in time, one RNN moves backward in time
- $h^{(t)}$ is a lossy summary of events up to t, $g^{(t)}$ is a lossy summary of events after t
- Output can combine summaries of past and future data

Bidirectional RNNs

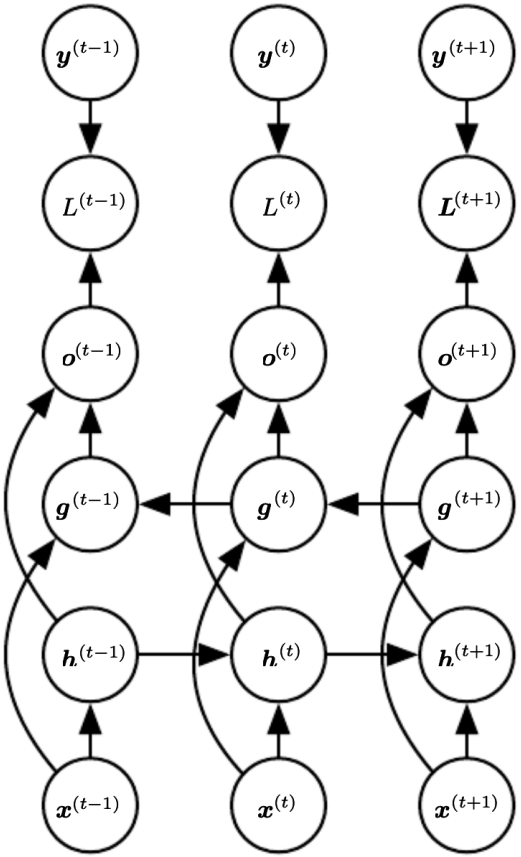


Figure 10.11: Computation of a typical bidirectional recurrent neural network, meant to learn to map input sequences $\boldsymbol{x}$ to target sequences $\boldsymbol{y}$, with loss $L^{(t)}$ at each step t. The $\boldsymbol{h}$ recurrence propagates information forward in time (toward the right), while the $\boldsymbol{g}$ recurrence propagates information backward in time (toward the left). Thus at each point t, the output units $\boldsymbol{o}^{(t)}$ can benefit from a relevant summary of the past in its $\boldsymbol{h}^{(t)}$ input and from a relevant summary of the future in its $\boldsymbol{g}^{(t)}$ input.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

Deep RNNs

- How can RNNs become deep? (Depth != number of time points)
- Replace the three transformations with deeper ones (input to hidden, hidden to hidden, hidden to output)

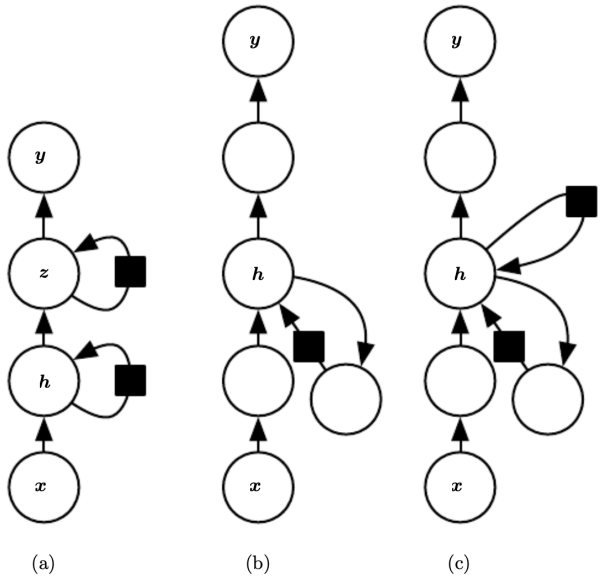


Figure 10.13: A recurrent neural network can be made deep in many ways (Pascanu *et al.*, 2014a). *(a)* The hidden recurrent state can be broken down into groups organized hierarchically. *(b)* Deeper computation (e.g., an MLP) can be introduced in the input-to-hidden, hidden-to-hidden, and hidden-to-output parts. This may lengthen the shortest path linking different time steps. *(c)* The path-lengthening effect can be mitigated by introducing skip connections.

- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

LSTM (long short term memory)

- Designed to handle the vanishing gradient problem
  (to make short-term memory last as long as possible)
- Can combine current information with information
  from e.g. 1,000 - 1,000,000 time steps earlier
- Self-loops, gradient can flow for long durations
- The weight of the self-loop is gated
  (controlled by another hidden unit)
  by a 'forget gate', which has its own parameters
- Three gates; input (write), forget (keep), output (read),
  with their own learned weight matrices

LSTM (LONG SHORT TERM MEMORY)

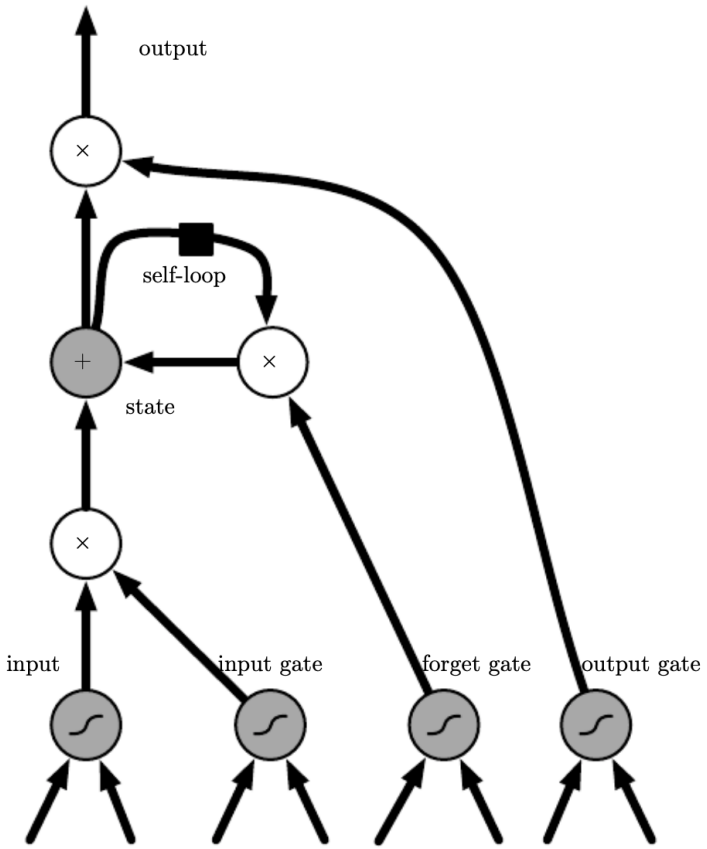


Figure 10.16: Block diagram of the LSTM recurrent network "cell." Cells are connected recurrently to each other, replacing the usual hidden units of ordinary recurrent networks. An input feature is computed with a regular artificial neuron unit. Its value can be accumulated into the state if the sigmoidal input gate allows it. The state unit has a linear self-loop whose weight is controlled by the forget gate. The output of the cell can be shut off by the output gate. All the gating units have a sigmoid nonlinearity, while the input unit can have any squashing nonlinearity. The state unit can also be used as an extra input to the gating units. The black square indicates a delay of a single time step.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

LSTM in Keras (sequence classification)

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import Embedding, LSTM

max_features = 1024

model = Sequential()
model.add(Embedding(max_features, output_dim=256))
model.add(LSTM(128))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

See https://keras.io/layers/recurrent/ for details

1D CNN OR RECURRENT NETWORK?

- I have time series data, should I use 1D CNN or recurrent network?
- Depends on how local the correlations are
  - Only "local" correlations, 1D CNN
  - Also "long" range correlations, RNN
- Depends on application
  - Classification, both RNNs and CNNs can be used
  - Translate sequence to another; RNN

Outline - part 6 of 6

- Practical methodology
  - Deep learning frameworks
  - Docker containers & Singularity
  - GPU hardware & acceleration
  - Measuring performance & uncertainty
  - Hyper parameter optimization
  - Quiz

Deep learning frameworks

- The most common deep learning frameworks are
  - Tensorflow (low level programming, more common for production)
  - Keras (high level programming)
  - Pytorch (low level programming, more common for research)
  - ...

Deep learning backends

- The backend performs the actual work / calculations
- Possible to use Keras with different backends
  - Tensorflow
  - Microsoft CNTK
  - Amazon MXNet
  - Theano
  - ...
- Training time and accuracy differ between backends

Anaconda

- Easy to setup deep learning environment, https://www.anaconda.com/distribution/
- conda create -n mykeras python=3.7
- conda install keras-gpu
- conda install tensorflow-gpu
- Each install command guarantees the correct versions (CUDA driver, CUDA toolkit, CUDNN, Python, ...)
- conda activate mykeras
- Can easily switch between different environments

Docker containers

- Versions of Python, Tensorflow, graphics drivers, CUDNN, must match
- May get different results for different operating systems
- After upgrading the OS, things may stop working...
- Maybe need to test something 2 years later, with the same setup
- This is simplified with Docker containers, small virtual machines
- A Docker container can be setup with a text file (Dockerfile)
- You can also send a built container to someone
- www.docker.com

Docker containers

- Run CentOS 6.8 container on your machine

  docker run -it --rm centos:6.8

- Run CentOS 7 container on your machine

  docker run -it --rm centos:7

- Run Ubuntu 12

  docker run -it --rm ubuntu:12.04

- Run Ubuntu 14

  docker run -it --rm ubuntu:14.04

- The first time the container will be downloaded from docker hub (hub.docker.com)

Docker files

- Dockerfiles define how to setup a container
- A text file that can easily be distributed and built on any computer
- Example
- FROM centos:6.8

  RUN yum -y install wget && \
  yum -y install git && \
  yum -y install gcc-c++
- Need to install all dependencies, nothing included as default
- Can of course run Anaconda inside Docker

Running a Docker container

- Built a container called 'nnimage-gpu'
- docker run --rm --gpus <all,0,1,...> -v /path/to/input/images:/in -v /path/to/output/segmentations:/out nnimage-gpu
- Runs 3D segmentation on volumes in input path (nifti files), saves results in output path
- Full example at https://github.com/wanderine/nnunetdocker

Docker containers - performance

- Some performance optimizations are only available through containers
- Mixed precision - a mix of 16 and 32 bit floats for higher performance
- Improved performance for 3D convolutions in CUDNN
- Nvidia makes improvements, quickly released as container, integration with Tensorflow may take a much longer time
- ngc.nvidia.com

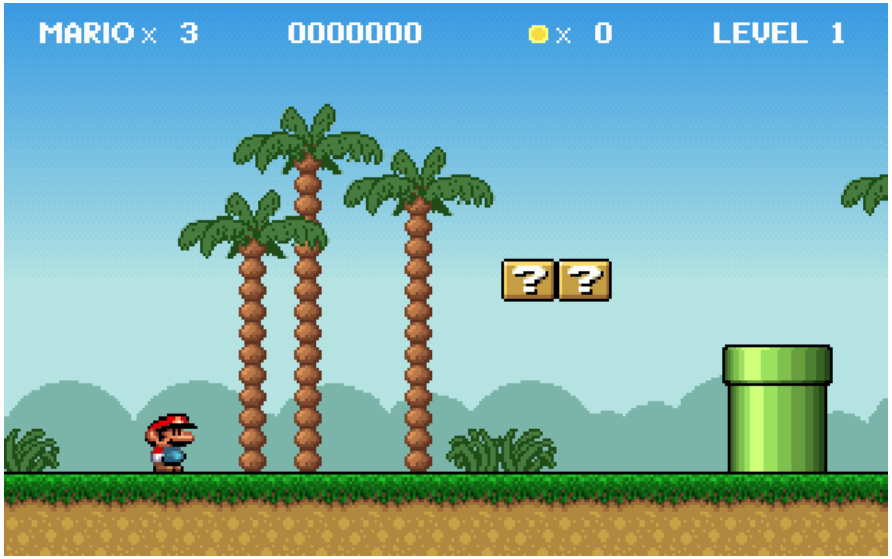
Singularity

- Docker containers typically require root access
- Several compute clusters do not allow Docker
- A Docker container can be converted into Singularity, which does not require root access
- Kurtzer, G. M., Sochat, V., & Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. PloS one, 12(5), e0177459.

GPU hardware & acceleration

- Virtually all deep learning requires GPU acceleration
- DL frameworks can run on CPU, but will be slow
- Built-in support for Nvidia GPU acceleration in most DL frameworks
- AMD GPU acceleration? ROCm, HipCaffe

Why GPU computing?



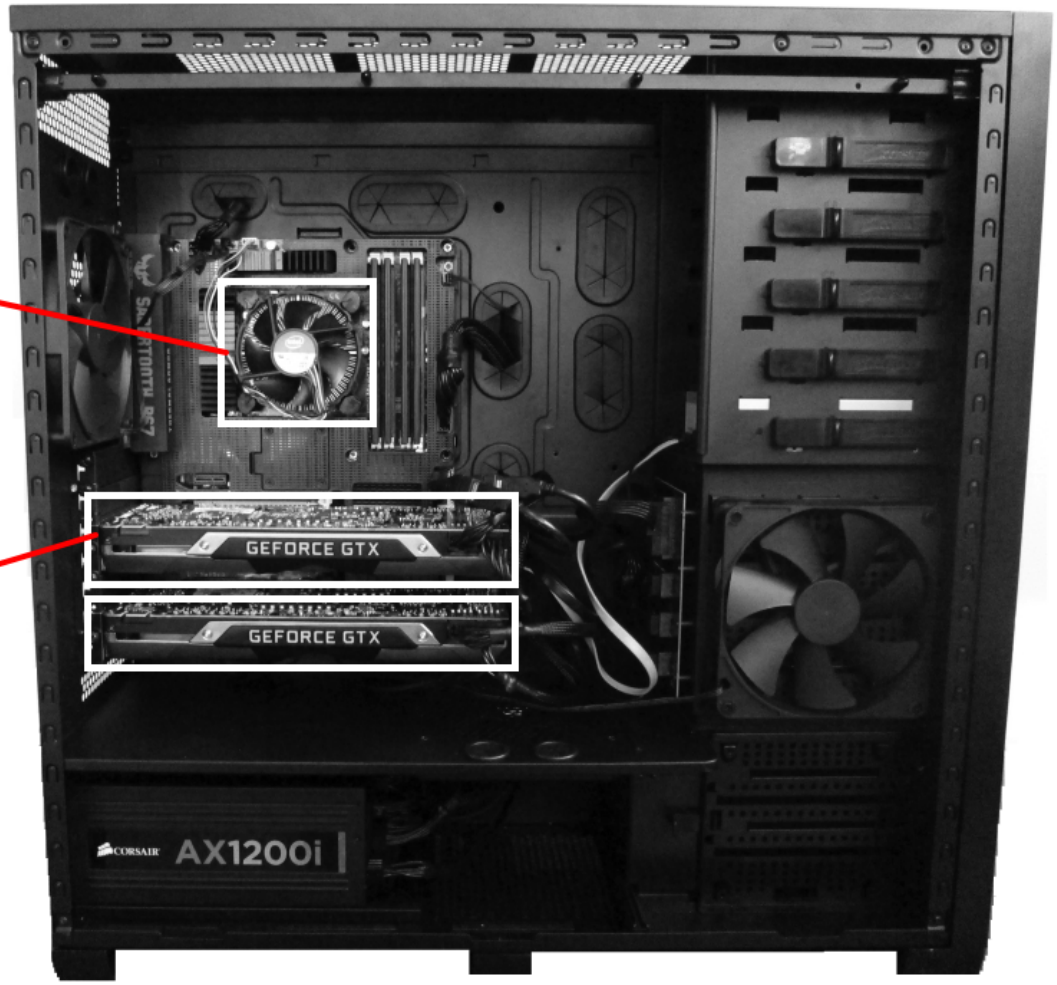
1986

2016

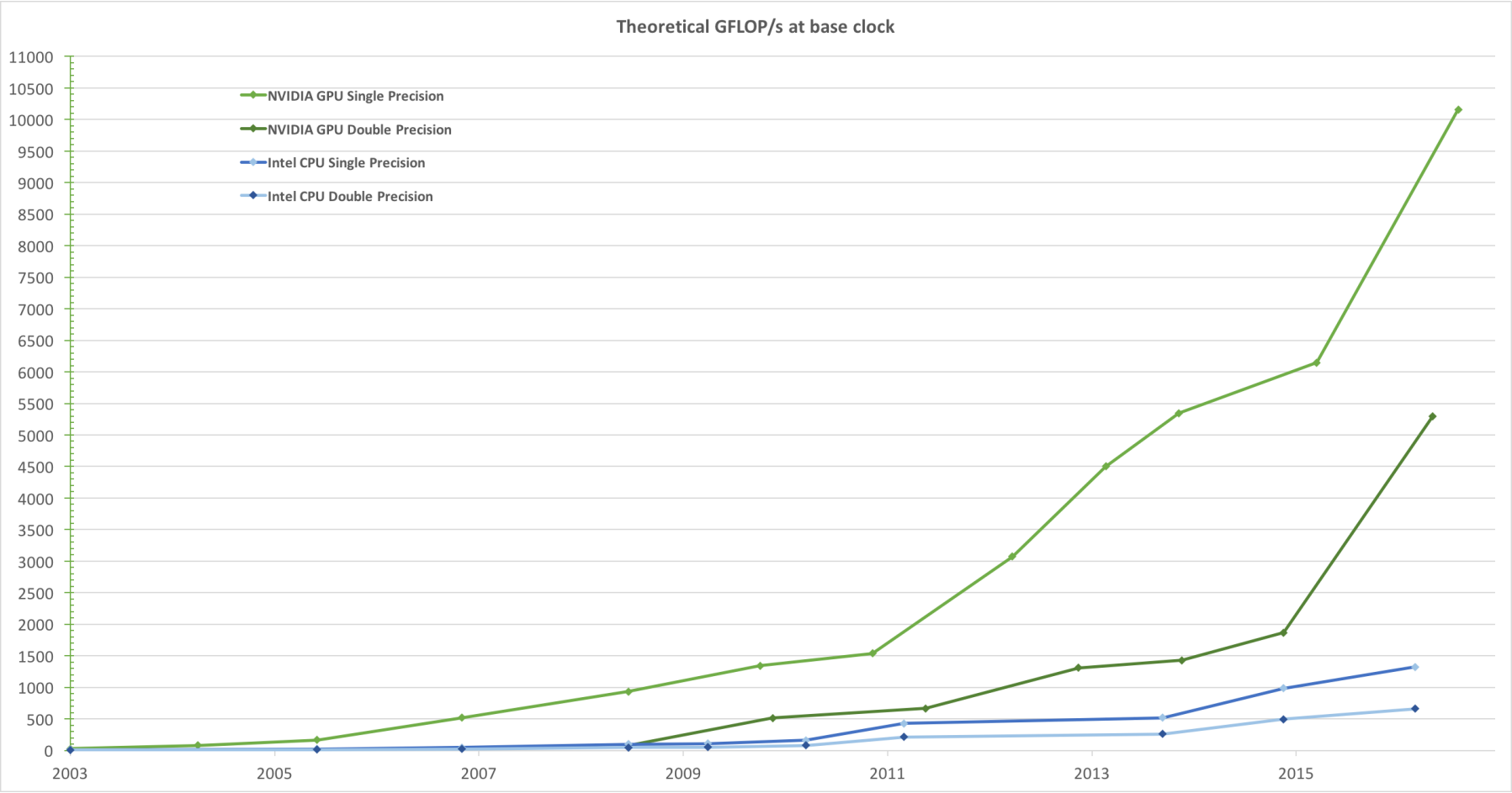
Why GPU computing?

CPU
4 cores

Graphics card
Nvidia GTX 980
2048 cores



WHY GPU COMPUTING?



https://docs.nvidia.com/cuda/archive/10.0/cuda-c-programming-guide/index.html

CUDA LIBRARIES

- CUDA programming language for Nvidia GPUs
- CUBLAS, dense matrix vector operations
- CUSPARSE, sparse matrix vector operations
- CUFFT, fast Fourier transform
- CURAND, random numbers
- CUDNN, deep neural network,
  called by Tensorflow / Pytorch / Theano ...

GPU memory

- The amount of GPU memory often sets the limit for how complex models that can be trained
- 'nvidia-smi' in a terminal tells you how much memory is being used (if using config.gpu_options.allow_growth = True)
- Maximum batch size depends on depth of network, and how many nodes / filters per layer (filter responses)
- Amount of CPU memory is also important, ideally all data is put into CPU memory, to feed GPU(s) (otherwise the GPU may wait for your hard drive...)

Measuring accuracy and uncertainty

- "The network accuracy is 70% on test data"
- OK, but what is the uncertainty of the accuracy?
- 70% $\pm$1%, or 70% $\pm$15% ?
- Cross-validation can be used to estimate uncertainty,
  split the data into different folds and perform training + testing,
  fold 1: 74%, fold 2: 82%, fold 3: 68%, fold 4: 72%
- In deep learning running 10 folds takes 20 days
  if training the network takes 2 days...

Uncertainty through dropout

- Use dropout during testing, run the same data through the network N times, every result will be slightly different
- Much more efficient compared to re-training the network
- Monte Carlo dropout, uncertainty = standard deviation over N results
- Gal, Y., & Ghahramani, Z. (2016, June). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In international conference on machine learning (pp. 1050-1059).

Uncertainty through dropout

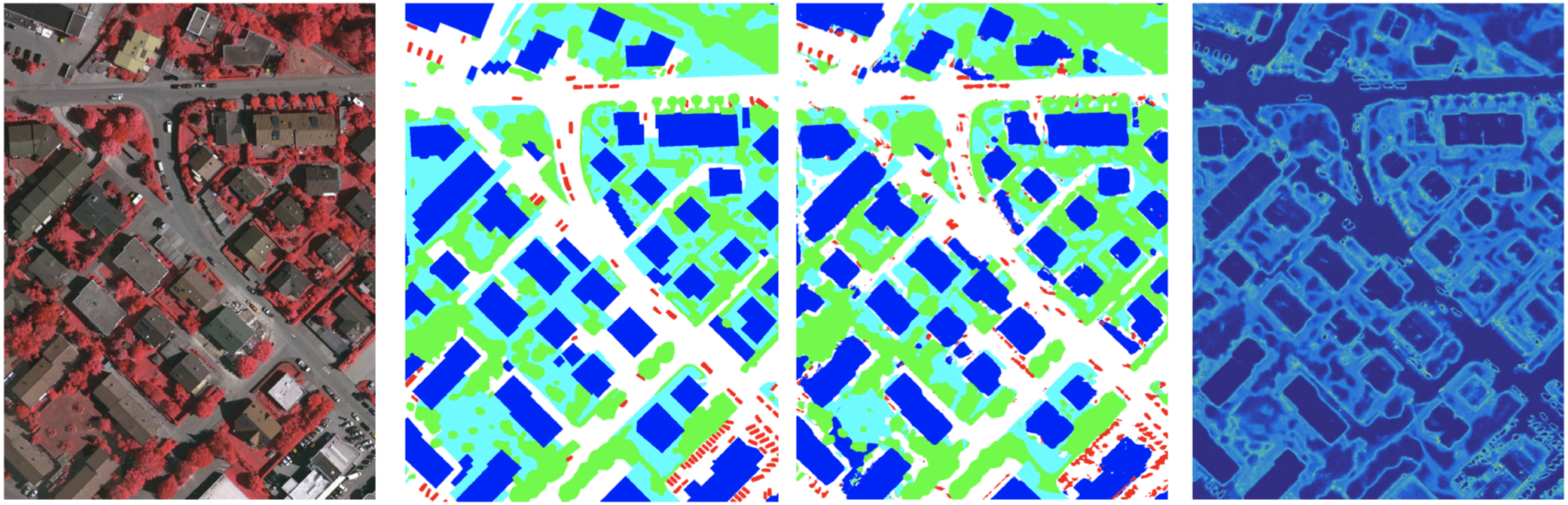


Figure 3: Results of the patch-based CNN. From left to right: One of the validation images, its ground truth, the results for the patch-based CNN and the uncertainty map.

Kampffmeyer et al. (2016). Semantic segmentation of small objects and modeling of uncertainty in urban remote sensing images using deep convolutional neural networks. IEEE conference on computer vision and pattern recognition workshops

Hyper parameter tuning

- Many hyper parameters to tune
  (number or layers, number of filters, learning rate, regularization)
- How can we find the best hyper parameters?
  - Manual tuning
    (requires good knowledge)
  - Automatic tuning
    (time consuming, grid search, random search),
    see book for details

Manual tuning - learning rate

- The learning rate is perhaps the most important hyper parameter
- Too high learning rate: training error can increase
- Too low learning rate: optimization can get stuck in local minima

Manual tuning - depends on error type

- Too low accuracy on training data: make network more complex
- Too low accuracy on test data: increase regularization, add more data

Understanding deep networks

4. Explaining DNN Decisions

In this section, we ask for a given data point $\boldsymbol{x}$, what makes it representative of a certain concept ω_c encoded in some output neuron of the deep neural network (DNN). The output neuron can be described as a function $f(\boldsymbol{x})$ of the input. A common approach is to view the data point $\boldsymbol{x}$ as a collection of features $(x_i)_{i=1}^d$, and to assign to each of these, a score R_i determining how *relevant* the feature x_i is for explaining $f(\boldsymbol{x})$. An example is given in Figure 3.

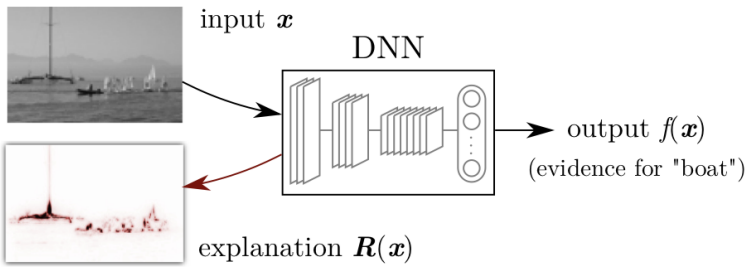
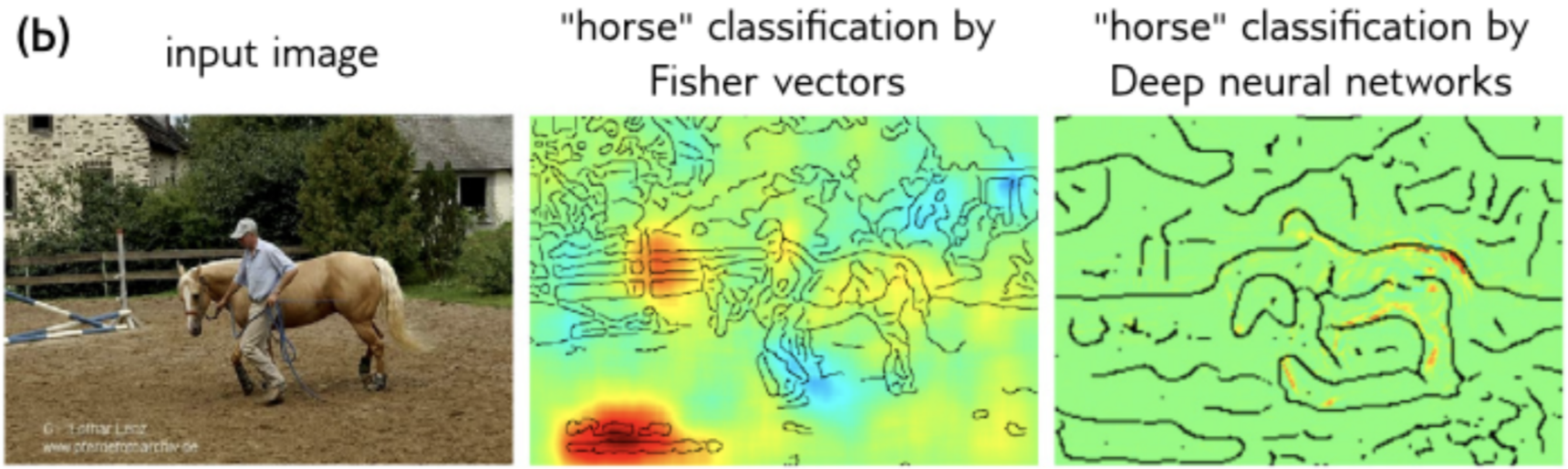


Figure 3: Explanation of the DNN prediction "boat" for an image $\boldsymbol{x}$ given as input.

In this example, an image is presented to a DNN and is classified as "boat". The prediction (encoded in the output layer) is then mapped back to the input domain. The explanation takes the form of a heatmap, where pixels with a high associated relevance score are shown in red.

Montavon, G., Samek, W., & Müller, K. R. (2018). Methods for interpreting and understanding deep neural networks. Digital Signal Processing, 73, 1-15.

Be careful with training data



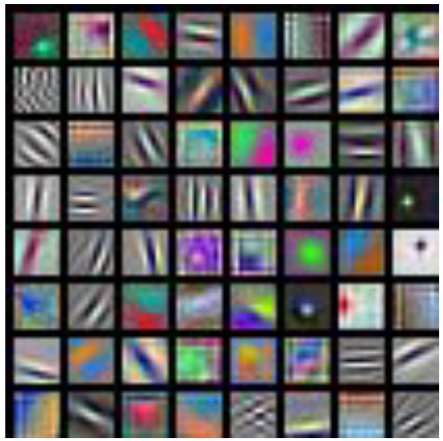
Based on Lapuschkin et al. (2016) "Analyzing classifiers: Fisher vectors and deep neural nets"

Fig. 21. Examples taken from the literature of model validation via explanation. (a) Explanation of the concept "`sci.space`" by two text classifiers. (b) Unexpected use of copyright tags by the Fisher vector model for predicting the class "horse".

Montavon, G., Samek, W., & Müller, K. R. (2018). Methods for interpreting and understanding deep neural networks. Digital Signal Processing, 73, 1-15.

UNDERSTANDING DEEP NETWORKS

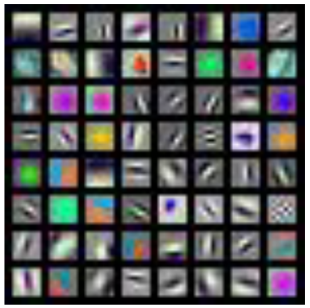
First Layer: Visualize Filters



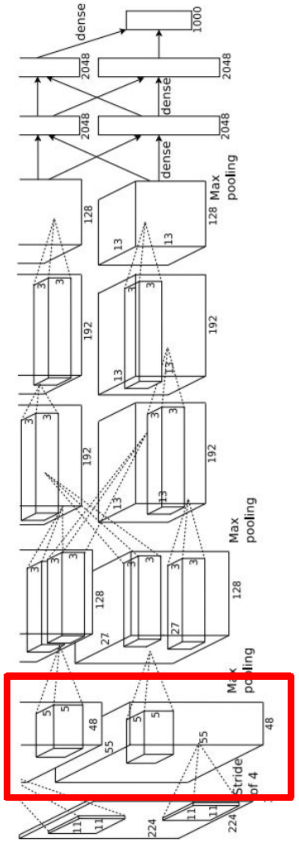
AlexNet:
64 x 3 x 11 x 11

ResNet-18:
64 x 3 x 7 x 7

ResNet-101:
64 x 3 x 7 x 7

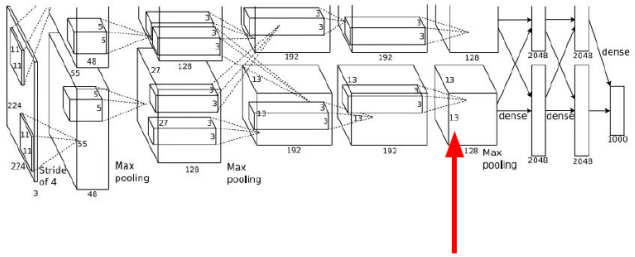


DenseNet-121:
64 x 3 x 7 x 7



Krizhevsky, "One weird trick for parallelizing convolutional neural networks", arXiv 2014
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Huang et al, "Densely Connected Convolutional Networks", CVPR 2017

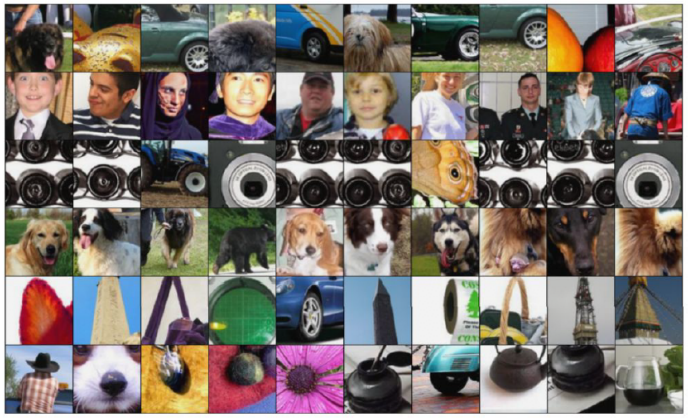
Maximally Activating Patches



Pick a layer and a channel; e.g. conv5 is 128 x 13 x 13, pick channel 17/128

Run many images through the network, record values of chosen channel

Visualize image patches that correspond to maximal activations



Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015
Figure copyright Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.

TENSORBOARD

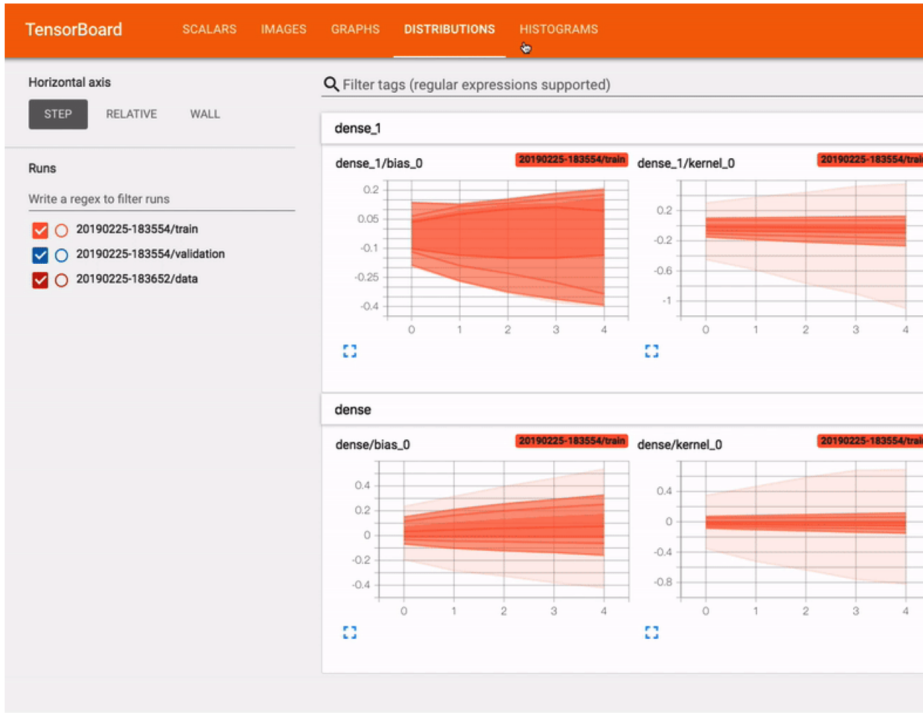
TensorFlow Install Learn API Resources Community Why TensorFlow Search Language GitHub Si

TensorBoard: TensorFlow's visualization toolkit

TensorBoard provides the visualization and tooling needed for machine learning experimentation:

- Tracking and visualizing metrics such as loss and accuracy
- Visualizing the model graph (ops and layers)
- Viewing histograms of weights, biases, or other tensors as they change over time
- Projecting embeddings to a lower dimensional space
- Displaying images, text, and audio data
- Profiling TensorFlow programs
- And much more

Get started



Quiz time

- I present a problem, you propose how to solve it

GPU UTILIZATION

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 367.48                 Driver Version: 367.48                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 1070    Off  | 0000:01:00.0      On |                  N/A |
|  0%   58C    P8    17W / 230W |    249MiB /  8105MiB |     40%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|    0       957    G   /usr/lib/xorg/Xorg                             247MiB |
+-----------------------------------------------------------------------------+
```

GPU UTILIZATION

- Check GPU utilization with 'nvidia-smi' in terminal
- If GPU utilization is only 40%,
  training will take 2.5 times longer compared to 100%
- Increase batch size to increase GPU utilization,
  will reduce training time (if you have enough GPU memory)

Low accuracy - training

- Accuracy on training data is 65%
- What do you do?

Low accuracy - training

- Things to try
  - Make network more complex, add layers
  - Increase number of epochs
  - Change learning rate
    (or tune other parameters of optimization algorithm)

Tensorflow crash

- W tensorflow/core/kernels/cwise_ops_common.cc:56] Resource exhausted: OOM when allocating tensor with shape[10000,23000] W tensorflow/core/common_runtime/executor.cc:1102] 0x50f40e0 Compute status: Resource exhausted: OOM when allocating tensor with shape[10000,23000] [[Node: add = Add[T=DT_FLOAT, _device="/job:localhost/replica:0/task:0/gpu:0"](MatMul, Variable_1/read)]] W tensorflow/core/common_runtime/executor.cc:1102] 0x3234d30 Compute status: Resource exhausted: OOM when allocating tensor with shape[10000,23000] [[Node: add = Add[T=DT_FLOAT, _device="/job:localhost/replica:0/task:0/gpu:0"](MatMul, Variable_1/read)]] [[Node: range_1/_13 = _Recv[client_terminated=false, recv_device="/job:localhost/replica:0/task:0/cpu:0", send_device="/job:localhost/replica:0/task:0/gpu:0",

Tensorflow crash

- Using too much GPU memory
  - Reduce batch size
  - Use smaller network (fewer layers and/or fewer filters per layer)
  - Use patches instead of full images
  - Buy a better graphics card

Low accuracy - testing

- Accuracy on training data is 97%, but 62% on test data
- What do you do?

Low accuracy - testing

- Probably due to overfitting
- Things to try
  - Increase regularization
  - Early stopping (stop when accuracy decreases for validation data)
  - Get more training data
  - Increase amount of data augmentation

Comparing networks

- Network A: performance on test data is 92%
- Network B: performance on test data is 94%
- Which network is best??

Comparing networks

- Without knowing the uncertainty, hard to pick a network!
- Estimate uncertainty using cross-validation or dropout
- Network A: 92% $\pm$ 2%, Network B: 94% $\pm$ 4%
- Cannot really say that one network is better, confidence intervals overlap

Thanks

- Thanks to David Abramian and Marco Cirillo for preparations

Keep learning