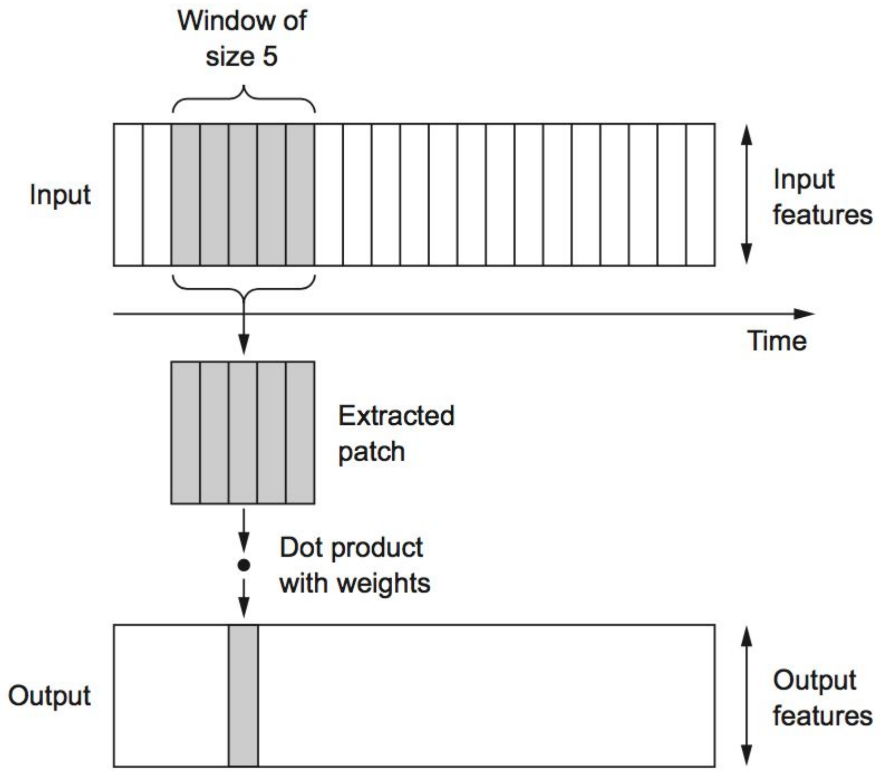
TX00DH43-3001
Introduction to Deep Learning

peter.hjort@metropolia.fi

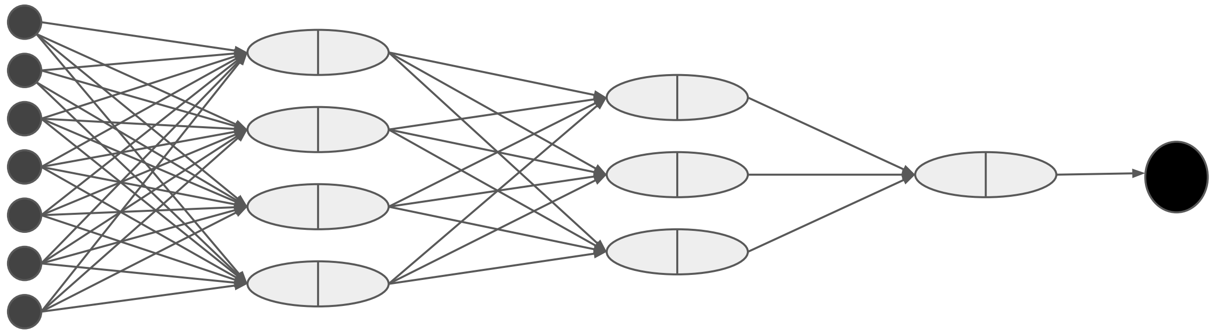
1D convolution for sequence data



No concept of sequence order outside convolution window.

Conv1D layer works well at least as a preprocessing step.

Batch normalization



We usually normalize the input data (center it at 0, and scale to standard deviation 1) before feeding it to the network. But what happens after that? Data coming out from network layers is not guaranteed to normalized any more. **Batch normalization** layer maintains moving average and variance per batch, and uses that to adaptively normalize the data. In practice batch normalization is needed with deeper networks architectures.

Batch normalization in Keras

```
model.add(Dense(120, activation='relu'))
model.add(BatchNormalization())
```

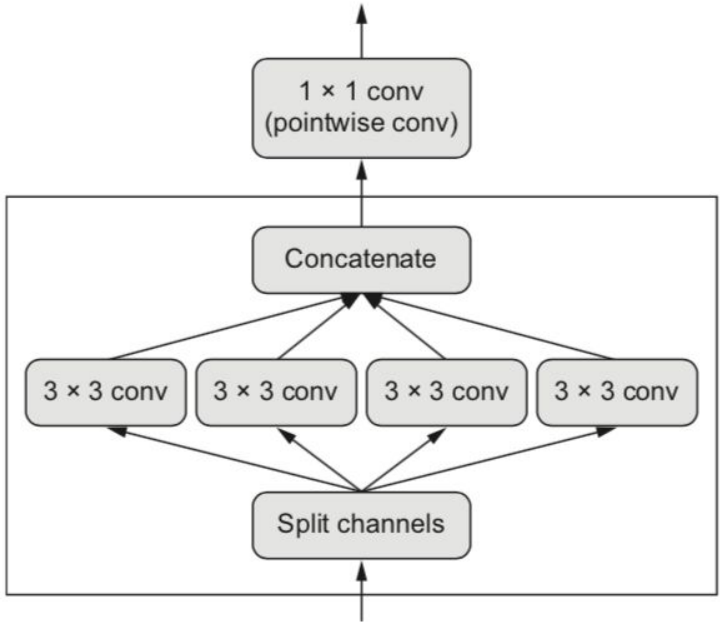
So, normalization takes place after applying nonlinearity. To apply normalization before nonlinearity:

```
model.add(Dense(120))
model.add(BatchNormalization())
model.add(Activation('relu'))
```

Original paper on batch normalization (https://arxiv.org/pdf/1502.03167.pdf) uses the latter approach.

Depthwise separable convolution

Separate learning of spatial features and channel-specific features in a convnet.
Computationally `SeparableConv2D` is cheaper than ordinary `Conv2D`.



From:

```
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 12, 12, 6)         0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 8, 8, 6)           900
_________________________________________________________________
```

To:

```
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 12, 12, 6)         0
_________________________________________________________________
separable_conv2d_2 (Separabl (None, 8, 8, 6)           186
_________________________________________________________________
```

Model ensembling

Create multiple models (with different architectures) to solve the problem and train them separately.

Use weighted average of model predictions as the final prediction. Weighting can be based on (or learned from) validation data.

Why would this work? If the models are different enough, their biases could be hoped to cancel each other out. Note that this is not restricted to deep models only; shallow and deep model may well be mixed in ensembling.

Text generation (Chollet 8.1)

Create the language model to be used in generation either in character level or in word level.

Text as sequence of words or sequence of characters suggests to use RNNs for this task. What would be the target of the model?

For character-level learning, pick (overlapping) sequences and try to predict the next character with 26-way (ascii text) softmax. This can be done without explicitly labeling the training data.

Training data

“In Xanadu did Kubla Khan A stately pleasure-dome decree: Where Alph, the sacred river, ran…”

Length 6, step 3

1. “In Xan” - “a”
2. “Xanadu” - “ “
3. “adu di” - “d”
4. “ did Ku” - “b”
5. ...

Note: there are other strategies to learn text statistics (= create language model) from training material. One popular is learning to predict the next word/character in a sentence.

Model

```
model = keras.models.Sequential()
model.add(layers.LSTM(128, input_shape=(maxlen, len(chars))))
model.add(layers.Dense(len(chars), activation='softmax'))

optimizer = keras.optimizers.RMSprop(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 128)               95232

_________________________________________________________________
dense_1 (Dense)              (None, 57)                7353
=================================================================
```

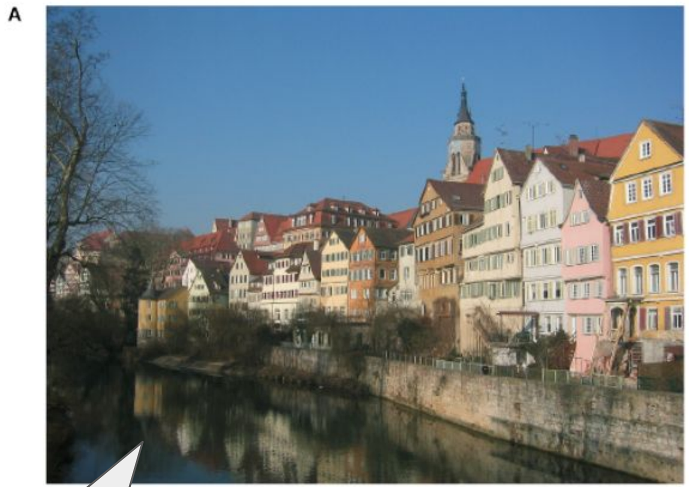
Text generation (character level)

After the model has been trained, give it some input (seed), and start to randomly sample characters.

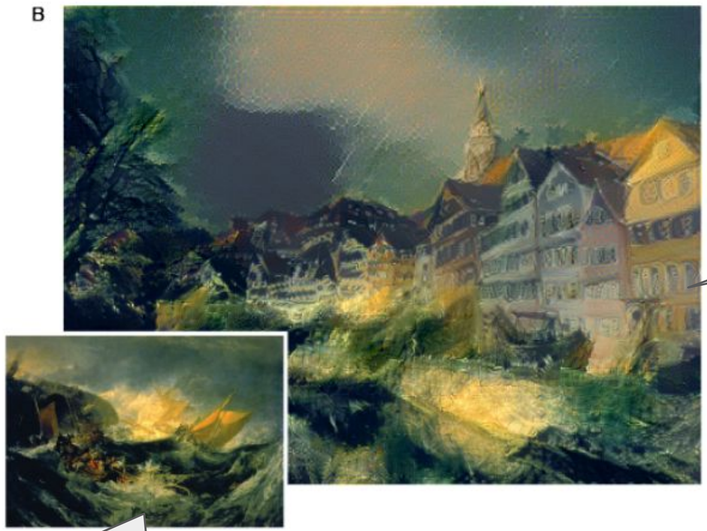
When generating text, sample from the probability distribution the model gives. In the training loop (see notebook `8.1-text-generation-with-lstm`) predict with input = seed + characters sampled this far and sample from softmax output. Note that `sample()` does not just pick the most probable character; it samples from the distribution (with the addition of temperature parameter to control the amount of randomness).

```
preds = model.predict(sampled, verbose=0)[0]
next_index = sample(preds, temperature)
```

Neural style transfer



Original image



Original image content mixed with reference style

Style reference image

From https://arxiv.org/pdf/1508.06576.pdf

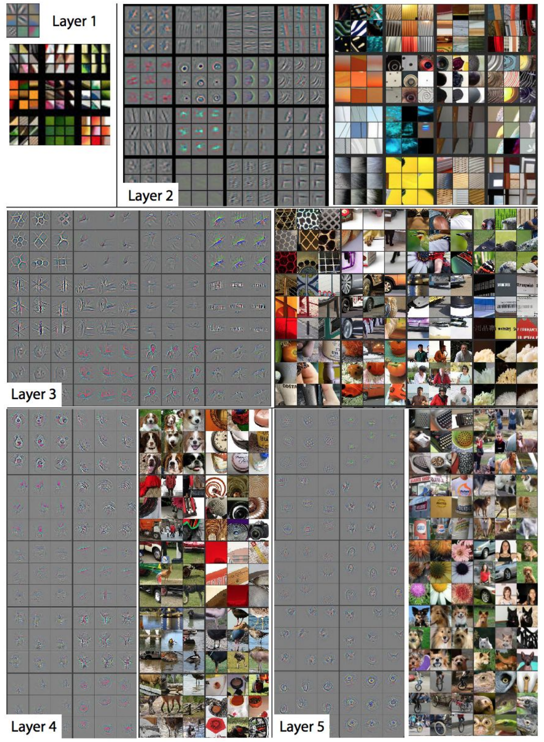
Neural style transfer

How to drive the system towards producing images that apply reference style and at the same time maintain image content?

If we would have functions `cont()` and `style()` to compute representation of content and style of an image, we could define the loss as:

```
 dist(style(ref) - style(gen)) + dist(cont(orig) - cont(gen))
```

Neural style transfer - content and style



Spatial scale increases when moving in convnet from lower layers upwards. Activations in upper layers are more global, and depict the content of the whole image.

Content loss ≈ distance between (single) upper layer activations in original image and generated image.

Style loss ≈ distance between a layer-local correlations matrix (Gram matrix) of filters computed over all layers. Captures the style over all scales.

Latent space of representations (for images, mostly)

Try to learn a representation for an image that would be something like the embedding space for word vectors (with word2vec or Glove).

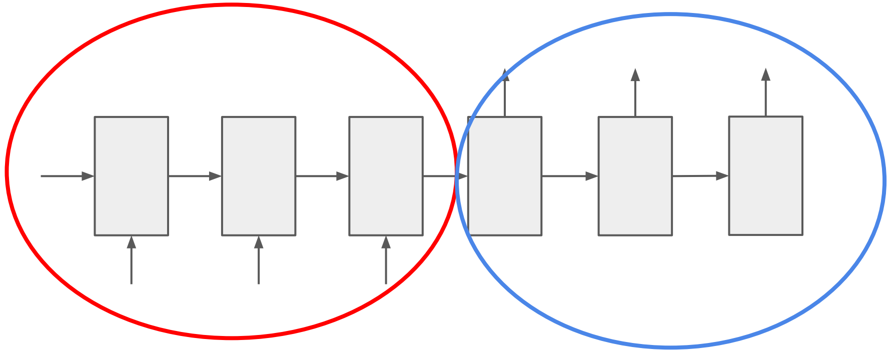
In text embedding one could imagine adding **concept vector** “royalness” to a “woman” and get “queen”. For images this would be like adding concept vector “smile” to an image and get a … image with smile.

So, how could we find such a latent space of representations for images? **Variational autoencoders** and **Generative Adversarial Networks (GANs)** are two techniques for doing this.

Note: if this would work one could add “youngness”, “Barack Obamaness” etc. to images, videos. Relatively easy fake evidence creation?

Autoencoders

Autoencoder is a encoder-decoder combination whose output is equal to its input. By placing restrictions on encoded representation, one can learn to compress the input, for example.

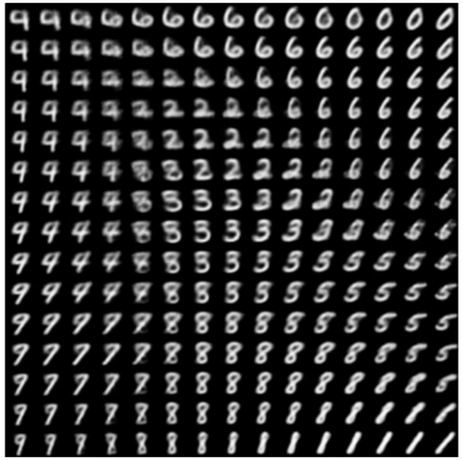


In practice autoencoders are not used much anymore. The representation learned by an autoencoder turn out not to be very useful, for example compression is not very efficient.

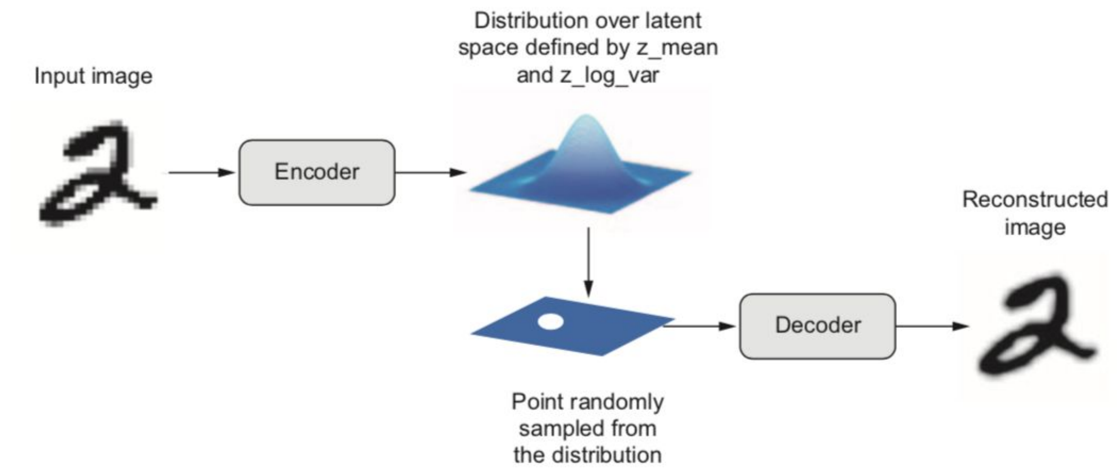
Image generation - variational autoencoder

Variational autoencoder takes the autoencoder idea further by

- Forcing the encoded (latent) space to be more **structured**
- Making the latent space more **continuous** - the space should not contain “gaps” that don’t have a mapping to reasonable image, and when moving a small distance in the latent space does not cause the mapped image to be drastically different



Variational encoder architecture

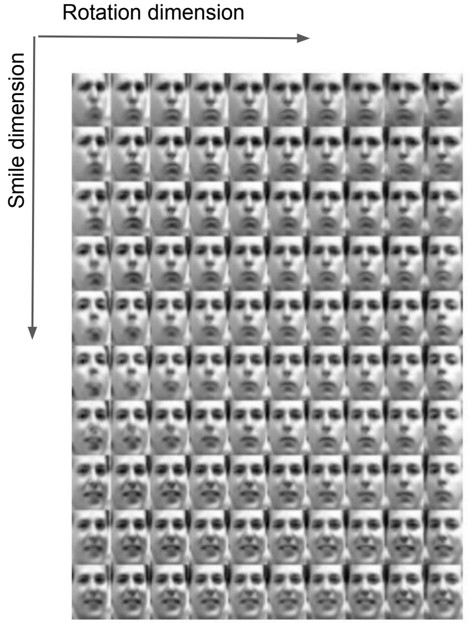

Input image
Encoder
Distribution over latent space defined by z_mean and z_log_var
Point randomly sampled from the distribution
Decoder
Reconstructed image

For Keras implementation, see 8.4-generating-images-with-vaes in book examples github.

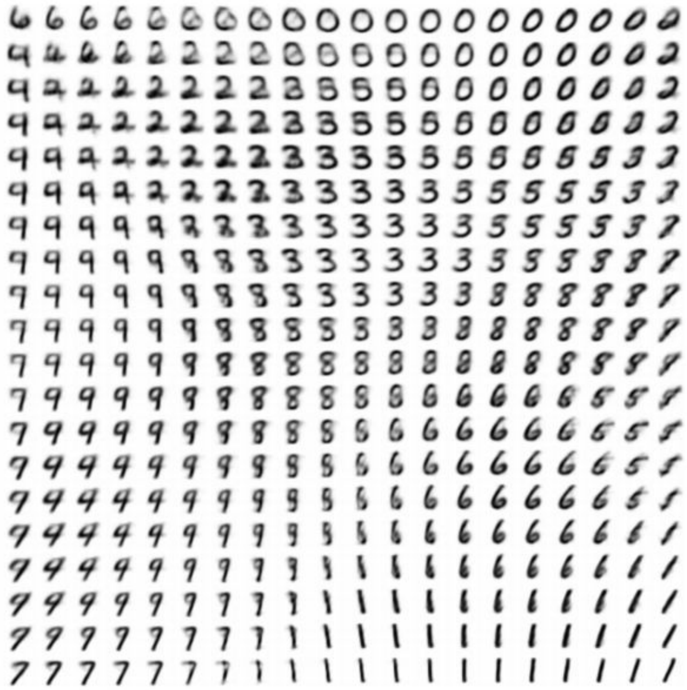
Both the encoder and decoder are convnets that are trained with the same idea as an autoencoder - reconstructed image should match input image.

Variational: the encoded image representation fed into decoder is sampled from random point close to the point to which encoder mapped the image.

Variational encoder latent space (learned)



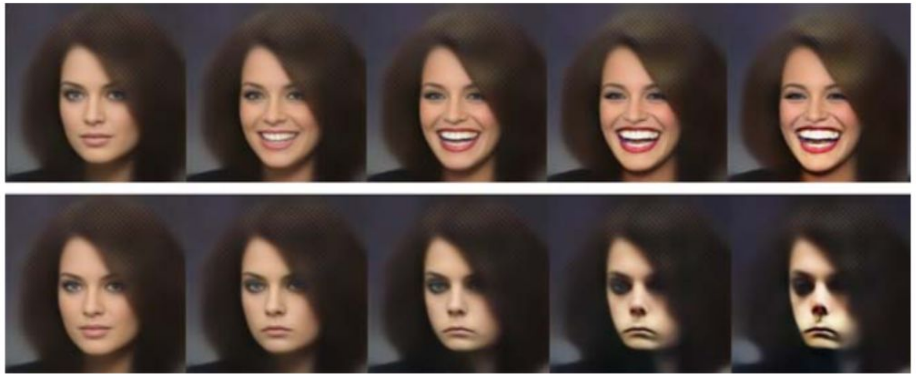
(a) Learned Frey Face manifold



(b) Learned MNIST manifold

From https://arxiv.org/pdf/1312.6114.pdf

Variational autoencoder



Add (or remove) “smile” vector to/from images

Create fictional faces

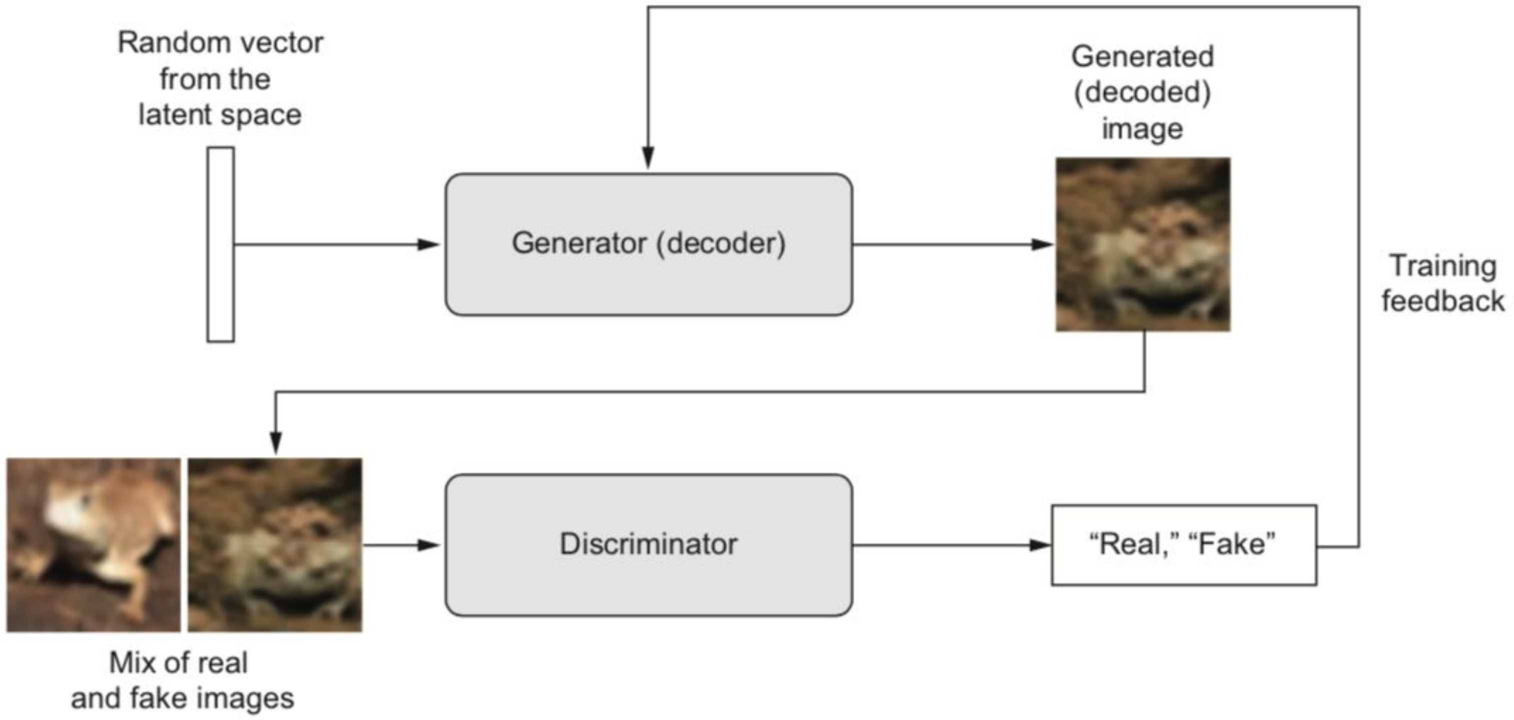
GAN (Generative Adversarial Network)

Same target as in VAEs (Variational Autoencoders) - learn a useful latent space (typically for images).

Two components:

- **Generative network**: generate a synthetic image based on random point in the latent space - learn to generate more and more real-looking images
- **Discriminator network (adversary)**: predict whether a given image is real (is taken from training set) or fake (is generated) - learn to detect fakes better and better

GAN architecture


Random vector from the latent space
Generator (decoder)
Generated (decoded) image
Training feedback
Mix of real and fake images
Discriminator
"Real," "Fake"

Optimization in GAN

Loss function for a GAN is not straightforward; the architecture is quite like a dynamic system.

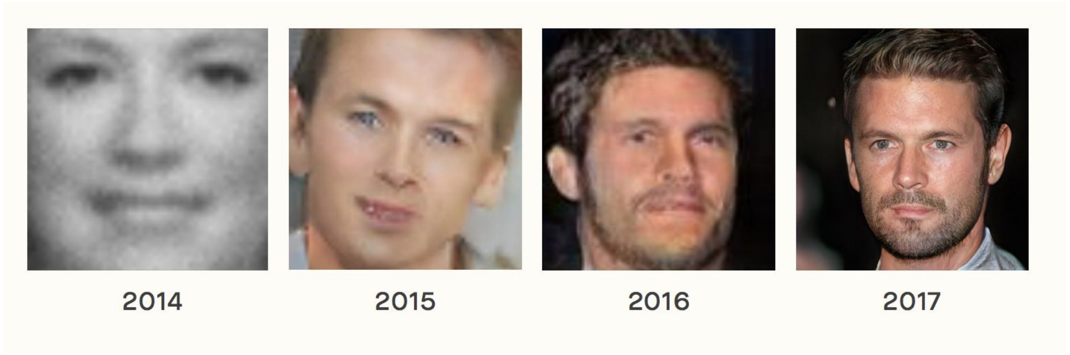
To minimize loss in GAN network (or network pair), a (Nash) equilibrium between generator and discriminator needs to be maintained.

For this reason training GAN system is hard: model architectures and training parameters need to be carefully matched.

GAN latent space

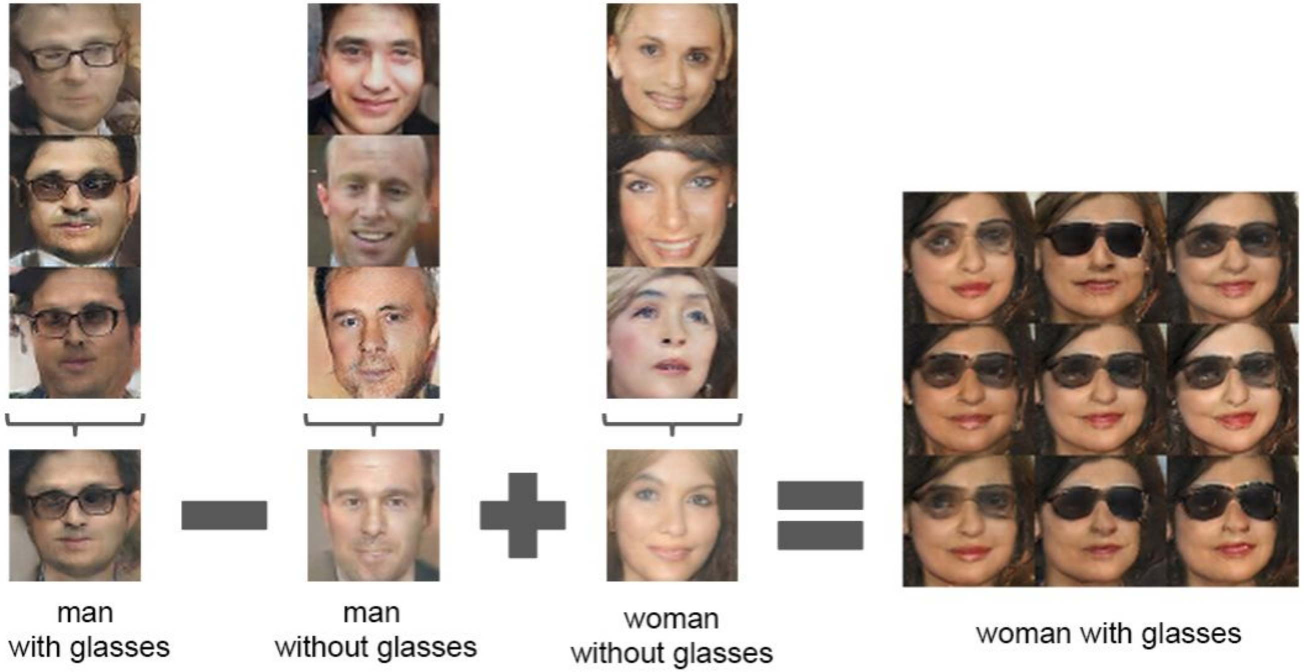
GAN-induced latent space can have the same features as VAE latent spaces (they not necessarily do).

Images generated from latent space created by GAN method can be more realistic than those from VAE. They also tend to exhibit less softness than images generated from VAE latent space:



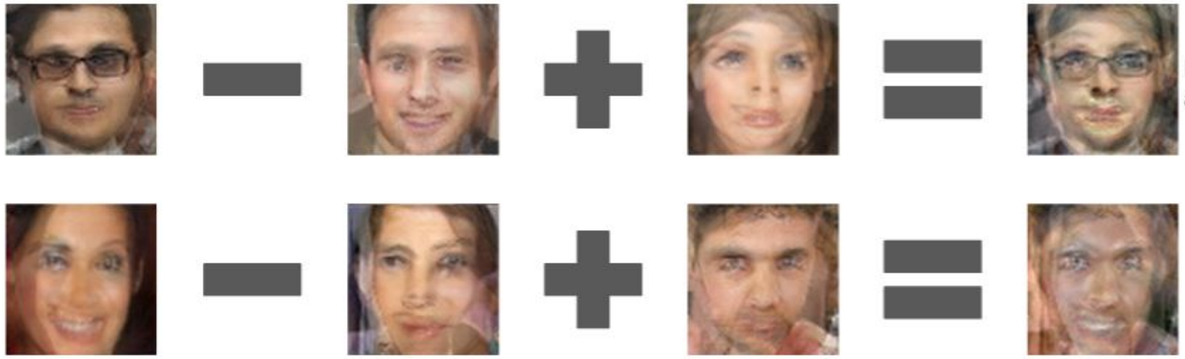
From https://www.eff.org/files/2018/02/20/malicious_ai_report_final.pdf

(GAN) latent space vector arithmetics



From https://arxiv.org/pdf/1511.06434.pdf

(Doing pixel arithmetic directly)



Encoding images to latent space and operating in that seems to make sense

Another GAN latent space vector - turn

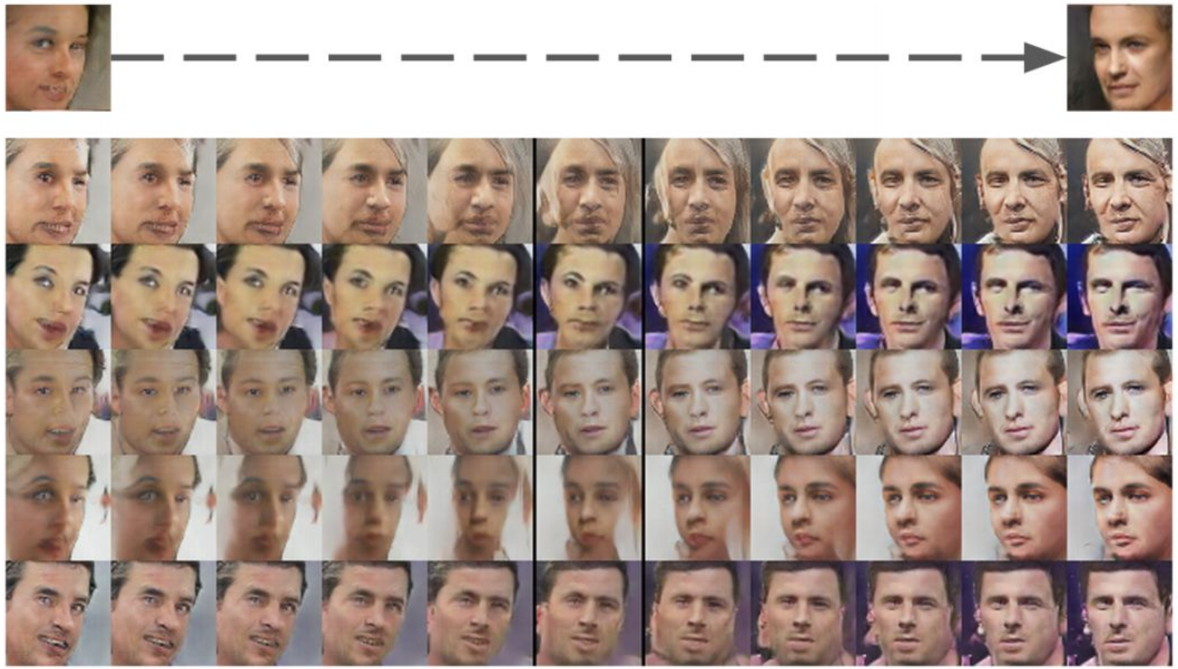
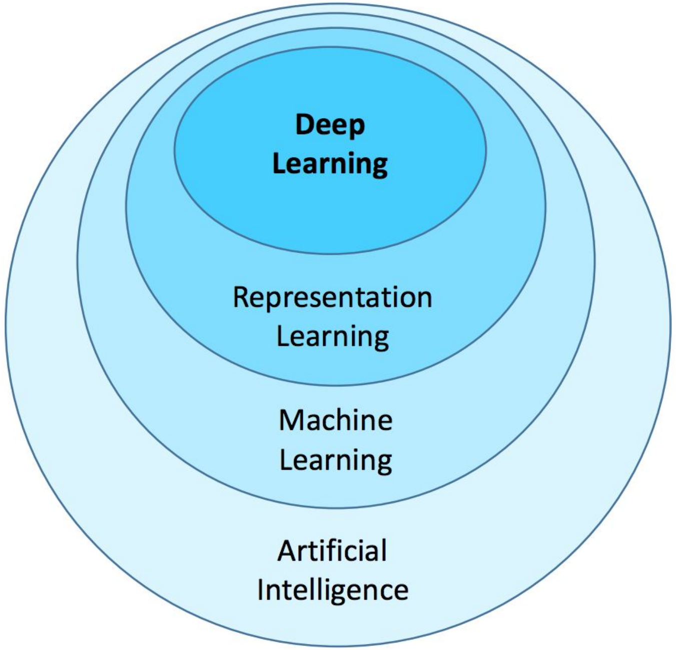
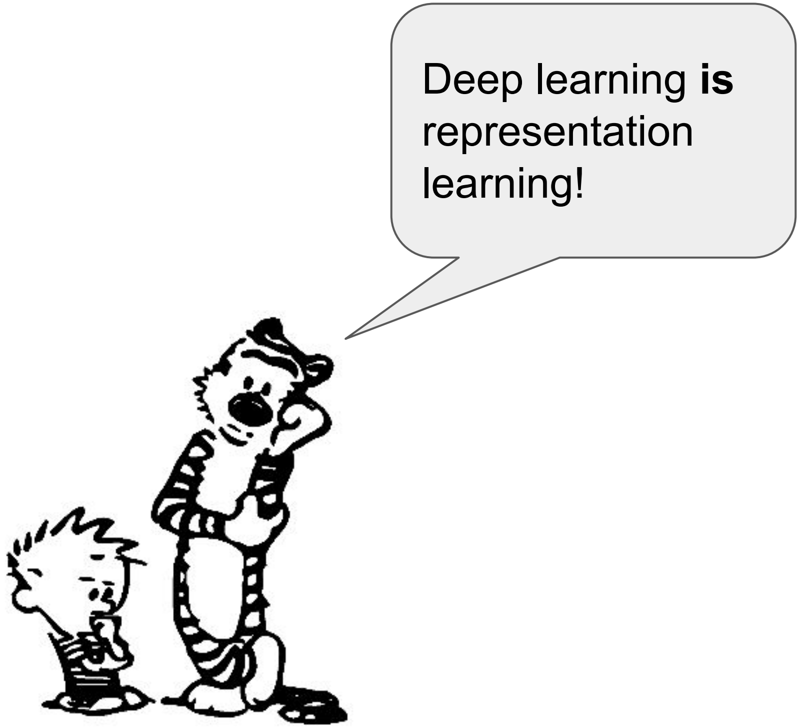


Figure 8: A "turn" vector was created from four averaged samples of faces looking left vs looking right. By adding interpolations along this axis to random samples we were able to reliably transform their pose.

Why all this talk about representation spaces?





Additional reading

https://spectrum.ieee.org/the-human-os/robotics/artificial-intelligence/hacking-the-brain-with-adversarial-images

Adversarial Examples that Fool both Human and Computer Vision: https://arxiv.org/pdf/1802.08195.pdf

http://blog.kaggle.com/2018/01/18/an-intuitive-introduction-to-generative-adversarial-networks/

https://www.theverge.com/2016/12/20/14022958/ai-image-manipulation-creation-fakes-audio-video

(Measuring the tendency of CNNs to Learn Surface Statistical Regularities: https://arxiv.org/pdf/1711.11561.pdf)

Reading list for exam 12.3

session07.pdf

Chollet: 6.4, 7.3, 8.1, 8.3, 8.4, 8.5

Exercise: depth-wise separable convolution

Try out SeparableConv2D in place of ordinary Conv2D in for example convolutional fashion-MNIST network. What are you findings on training speed and accuracy?

Exercise*: hyperparameter optimization

Try out hyperparameter optimization (see Chollet 7.3.2) with hyperopt (https://github.com/hyperopt/hyperopt) and hyperas(https://github.com/maxpumperla/hyperas). Include with your tryout jupyter notebook including explanation of your findings.

*) Extra exercise, not mandatory

Exercise*: word-level text generation

Train an RNN (use LSTM or GRU) with 5-10 Metropolia BSc thesis texts (you can get pdfs from theseus, google for options to convert pdf to plain text, if needed). Train the model in word level (you might want to use embedding layer with either learned or reused weights). Use your model to generate some sample sentences in different phases of training like in the example in Chollet 8.1.

*) Extra exercise, not mandatory. This exercise gives double points.

Exercise*: reuters newswire classification with 1D convolution

Train a model that uses 1D convolution to classify reuters newswire data.
Compare accuracy with dense network (Chollet book example) and/or your own previous solution.

*) Extra exercise, not mandatory

Exercise*: RNN for recognizing sorted int sequences

Create and train an RNN model that recognizes whether a given integer sequence is sorted or not.

*) Extra exercise, not mandatory