,,

Deeplearning

Autoencoders

Vladimír Kunc
FEL ČVUT

October 20, 2017

OUTLINE

3 parts

- Brief summary of artificial neural networks
- General deeplearning
- Auto encoders

BASIC ARCHITECTURE

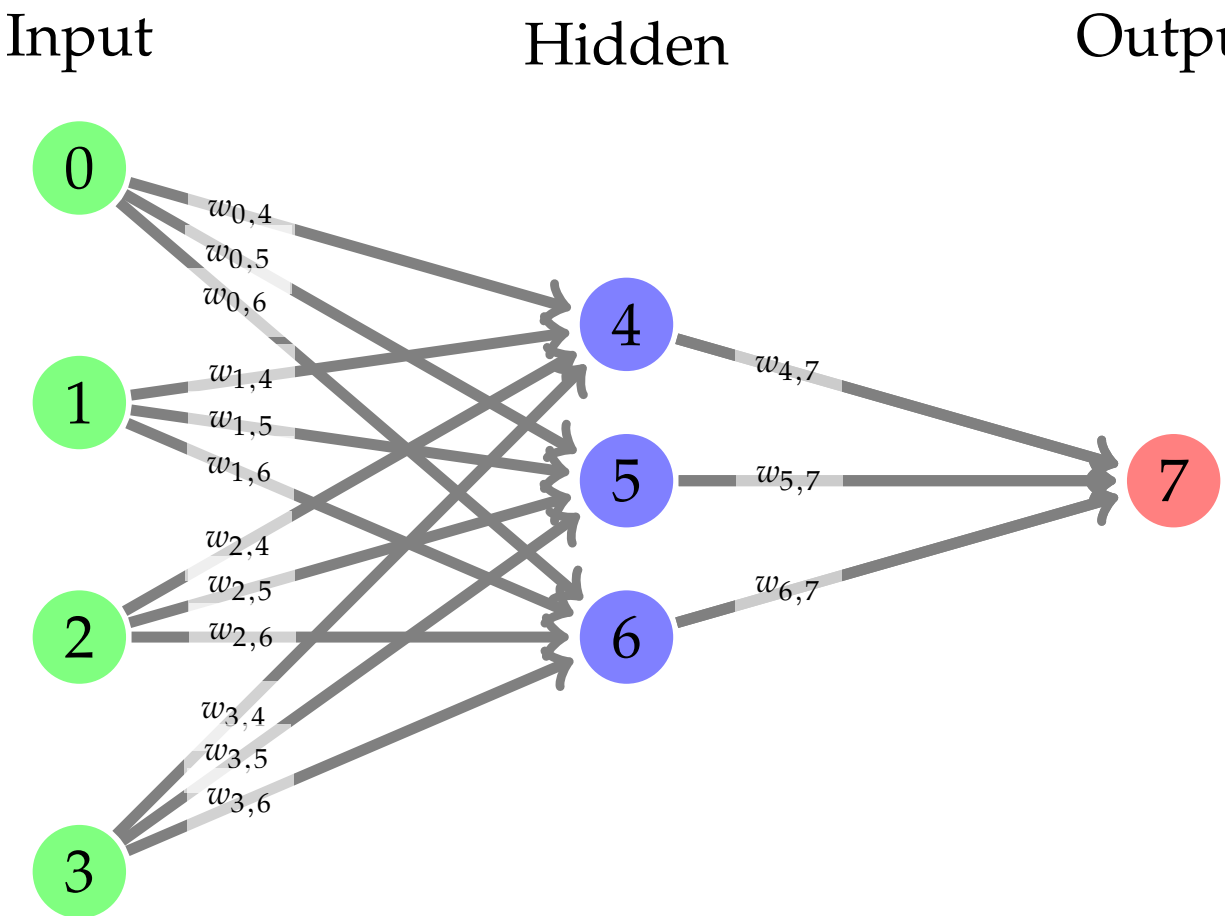


Figure: An example of a simple feedforward neural network with 3 input neurons, 2 neurons in the hidden layer and 1 output neuron.

SINGLE NEURON

$$x_i[t] = f\left(b_i + \sum_{j=0}^{n} w_{i,j} x_j[t-1]\right) \tag{1}$$

The Eq. 1 is often written in a matrix form:

$$x_i = f\left(\mathbf{w}_i^{\mathrm{T}} \mathbf{x}\right), \tag{2}$$

where

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_0 \\ \vdots \\ x_n \end{bmatrix} \tag{3}$$

and

$$\mathbf{w}_i = \begin{bmatrix} b_i \\ w_{i,0} \\ \vdots \\ w_{i,n} \end{bmatrix} \tag{4}$$

OUTPUT OF NNS

The output y of the example network from fig. 10 for given inputs $x_0, \ldots, x_3$ is:

$$y = \mathrm{S}\left(b_7 + w_{4,7}\mathrm{S}\left(b_4 + \sum_{j=0}^{3} w_{j,4}x_j\right) + w_{5,7}\mathrm{S}\left(b_5 + \sum_{j=0}^{3} w_{j,5}x_j\right) + \right.$$
$$\left. + w_{6,7}\mathrm{S}\left(b_6 + \sum_{j=0}^{3} w_{j,6}x_j\right)\right) \quad (5)$$

ACTIVATION FUNCTIONS

- usually non-linear
- many possibilities (e.g. sigmoid, tanh)
- favorite in recent applications:
  - ReLU

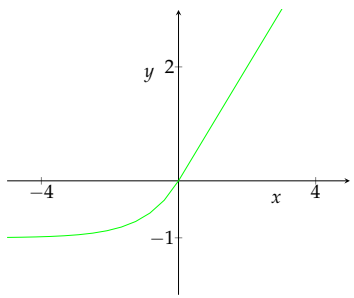
$$f(x) = \begin{cases} 0 \text{ for } x < 0 \\ x \text{ for } x \geqslant 0 \end{cases}$$

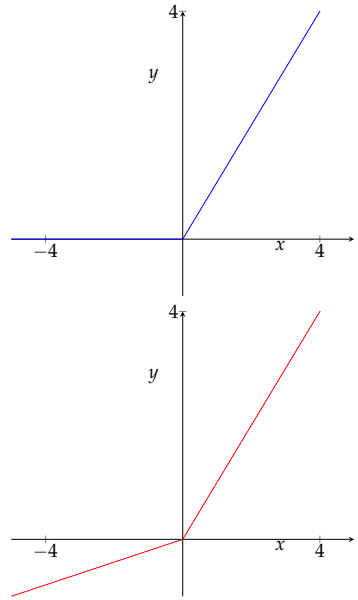
  - ELU — $\alpha \geqslant 0$, hyperparameter

$$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) \text{ for } x < 0 \\ x \text{ for } x \geqslant 0 \end{cases}$$

  - PReLU — α trainable

$$f(\alpha, x) = \begin{cases} \alpha x \text{ for } x < 0 \\ x \text{ for } x \geqslant 0 \end{cases}$$





CONVOLUTION AND POOLING

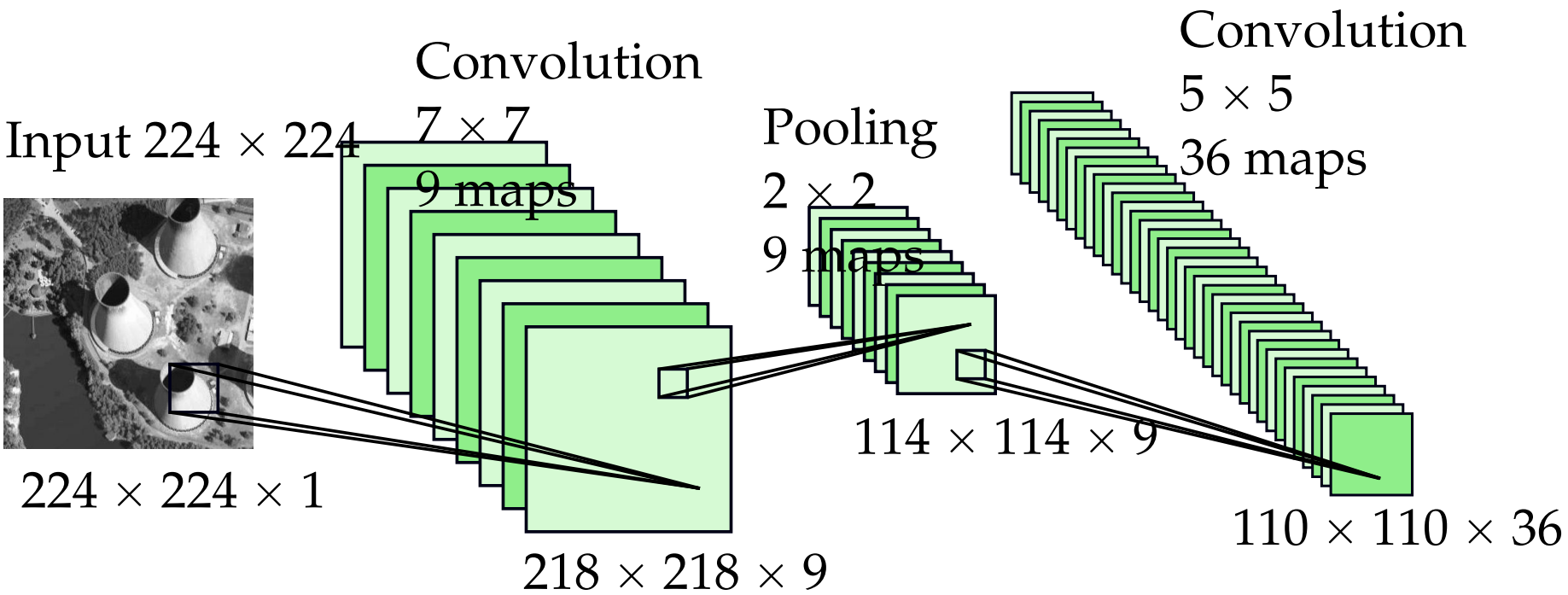


Figure: Example of usage of convolutional and pooling layers. Note that this example has no padding and thus the convolutions also reduce dimension.

CONVOLUTION

weight sharing

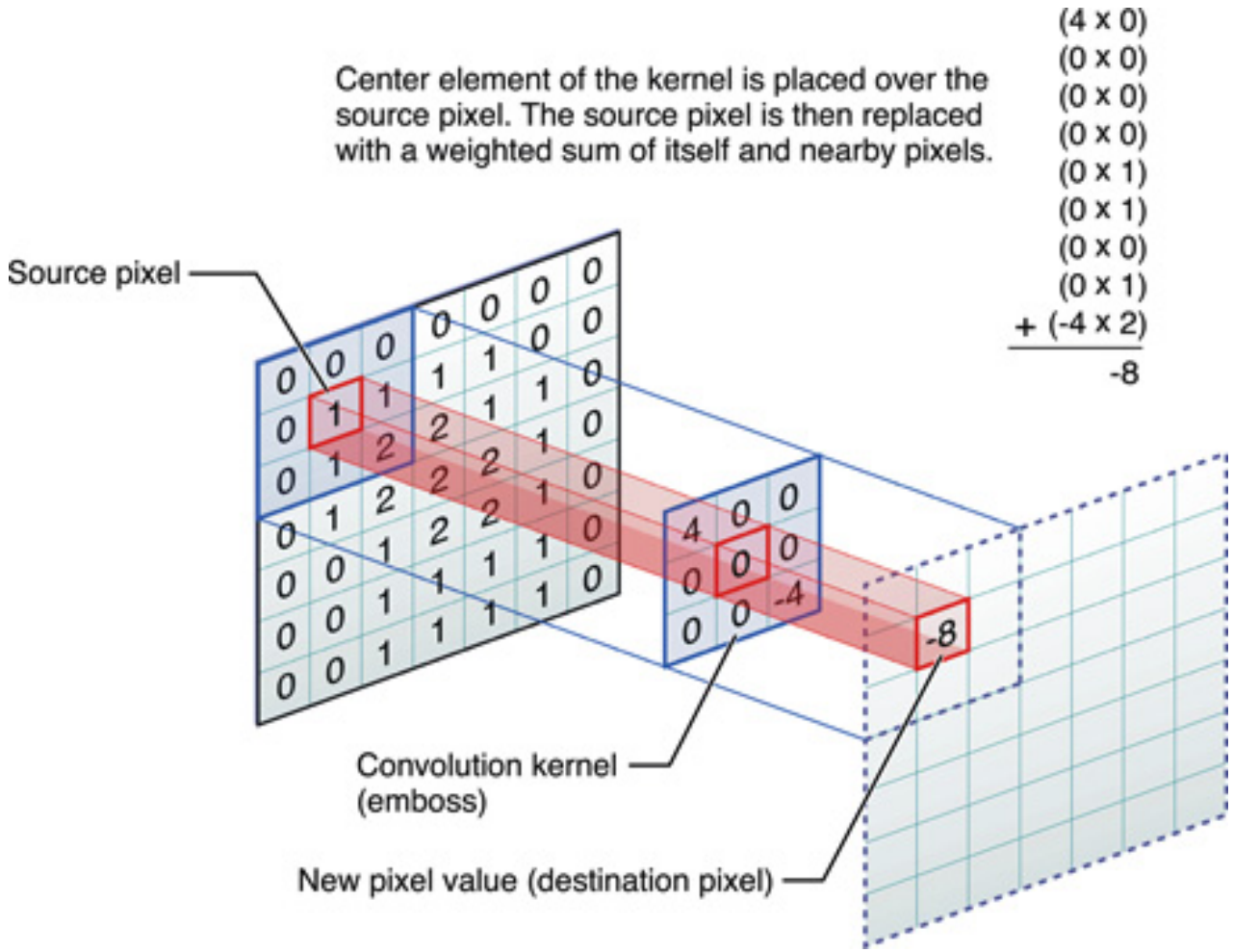


Figure: Convolution detail. Taken from `https://developer.apple.com`.

POOLING

dimensionality reduction

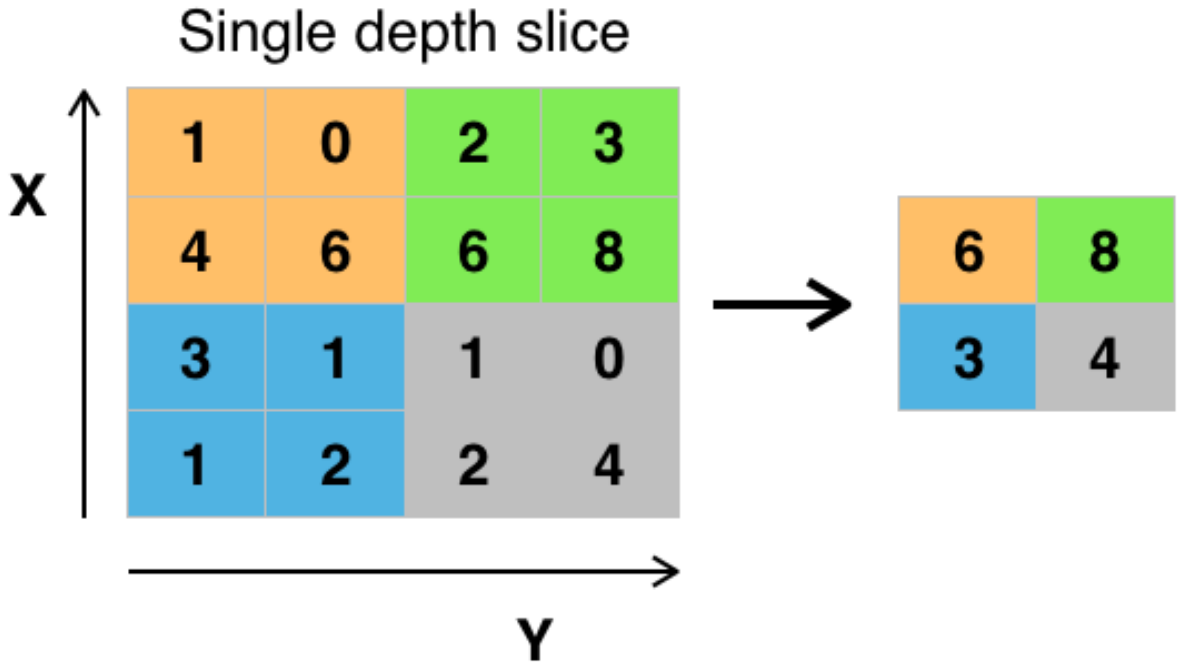


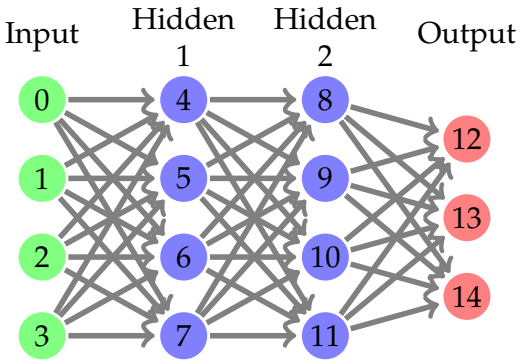
Figure: Max pooling with a 2×2 filter and stride 2. Taken from `https://en.wikipedia.org`.

TRAINING

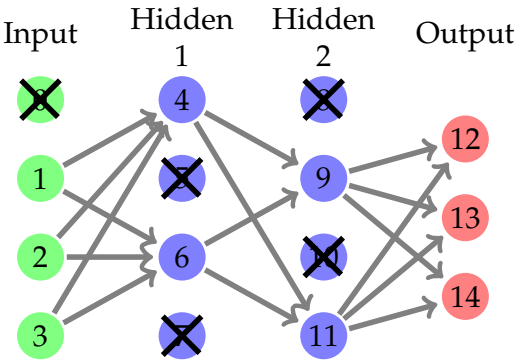
- difficult
  - overfitting
    - caused by too many free parameters
    - approaches: weight sharing (e.g. convolution), dropout, weight regularization, stochastic pooling, early stopping, weight decay
  - usually by gradient descent
  - numerical problems
    - exploding or vanishing gradients
    - caused by high number of layers

DROPOUT

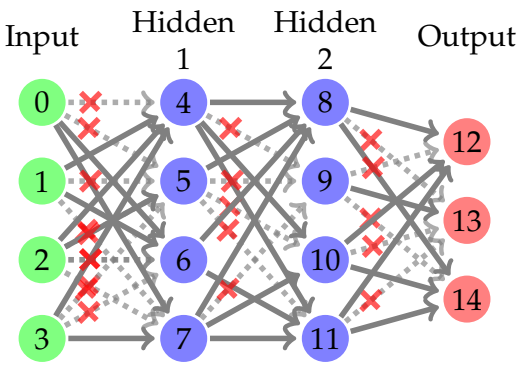
forcing the network to learn robustly



(a) without dropout



(b) with dropout



(c) with DropConnect

Figure: Example of dropout regularizations

COMPUTING GRADIENTS — BACK PROPAGATION

Based on iterative use of the *chain rule of differentiation* [**?**]. For $x \in \mathcal{R}, f(x) : \mathcal{R} \to \mathcal{R}, g(x) : \mathcal{R} \to \mathcal{R}, y = g(x)$, and $z = f(y) = f(g(x))$, then the chain rule is [**?**]:

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x} \tag{6}$$

The vector case is more relevant for neural networks. Let $\mathbf{x} \in \mathcal{R}^m, \mathbf{y} \in \mathcal{R}^n, z \in \mathcal{R}, g(\mathbf{x}) : \mathcal{R}^m \to \mathcal{R}^n, f(\mathbf{y}) : \mathcal{R}^n \to \mathcal{R}, \mathbf{y} = g(\mathbf{x})$, and $z = f(\mathbf{y})$, then the chain rule is [**?**] is:

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{7}$$

GRADIENT DESCENT

- simplest — fixed length step in the direction of steepest descent
- more complex methods include
  - momentum (or Nesterov momentum)
  - learning rate update schedule (e.g. rate decay)
  - adaptive learning rate
  - second order methods

`http://cs231n.github.io/assets/nn3/opt2.gif`

VGG16

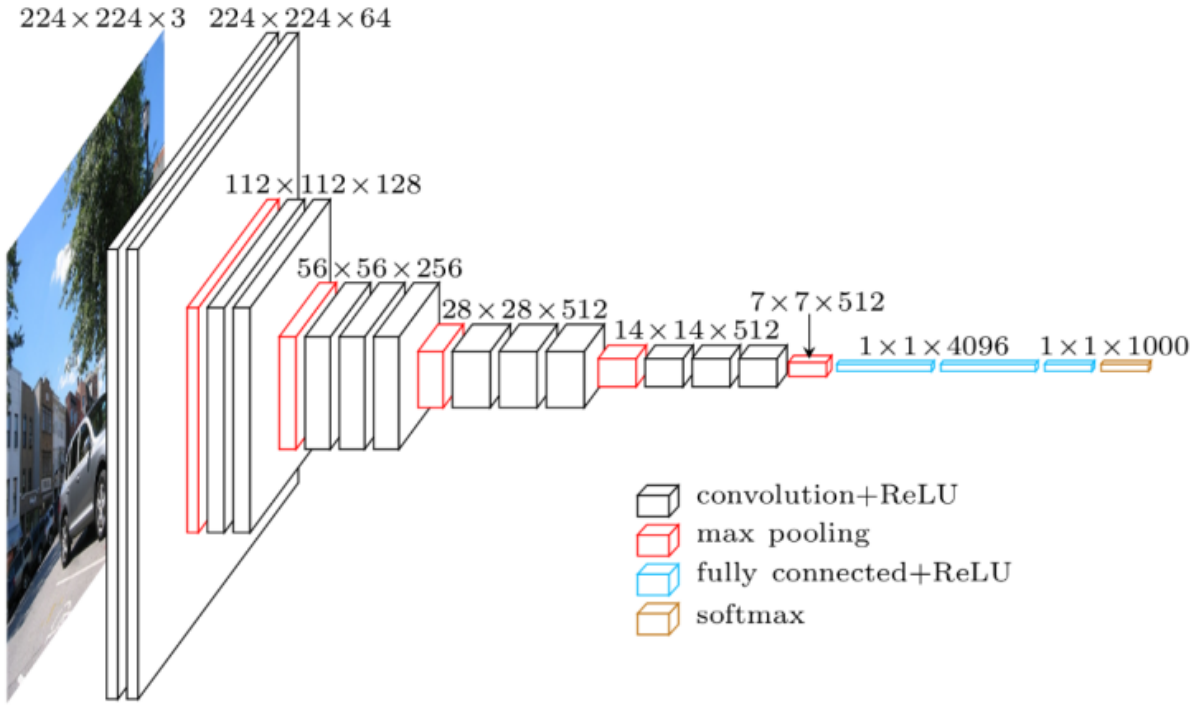


Figure: VGG16 architecture. Reused from
`https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/`.

RESNET

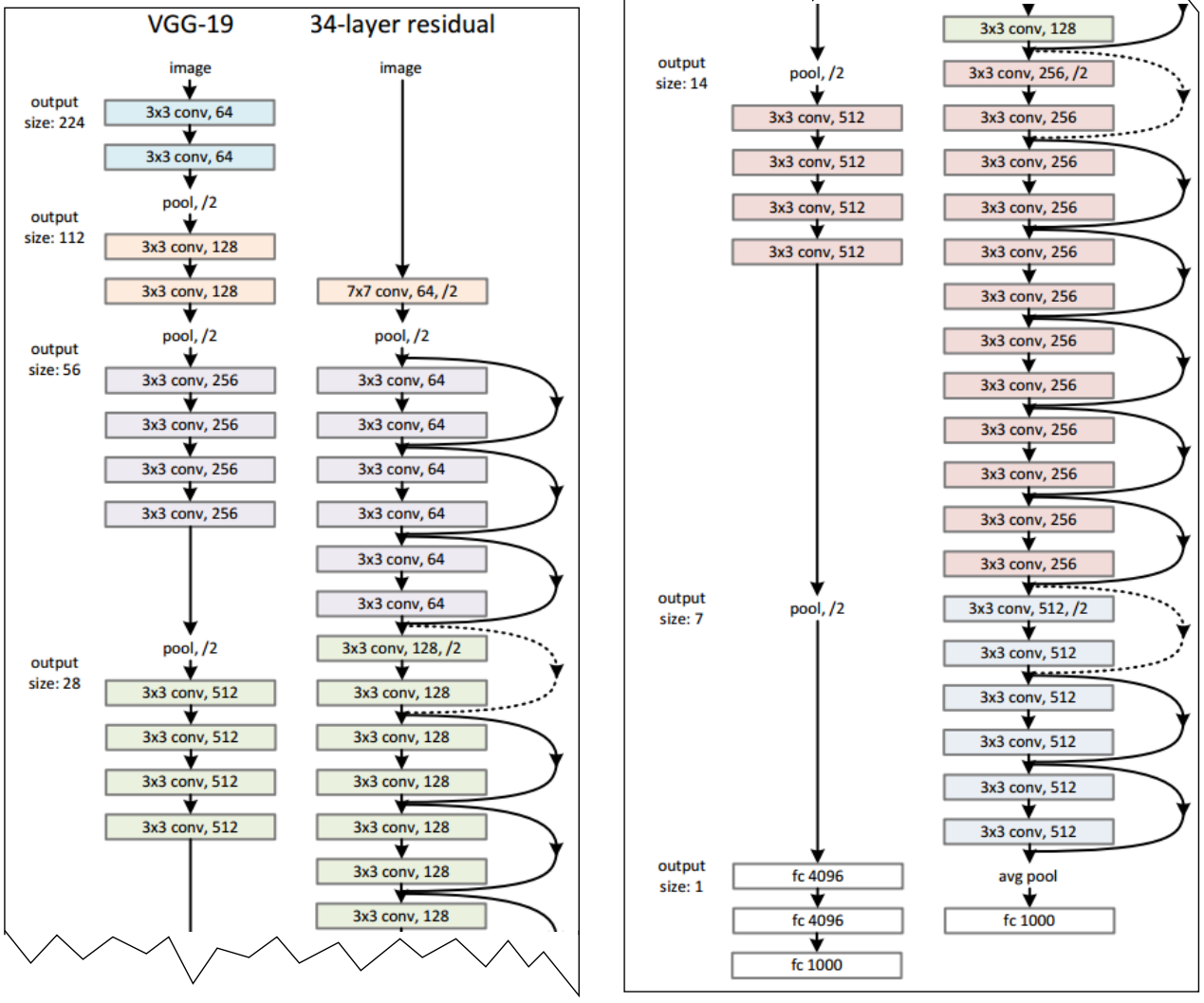


Figure: Comparison of architectures VGG19 and ResNet34. Adapted from He et al. 2016.

INCEPTION V3

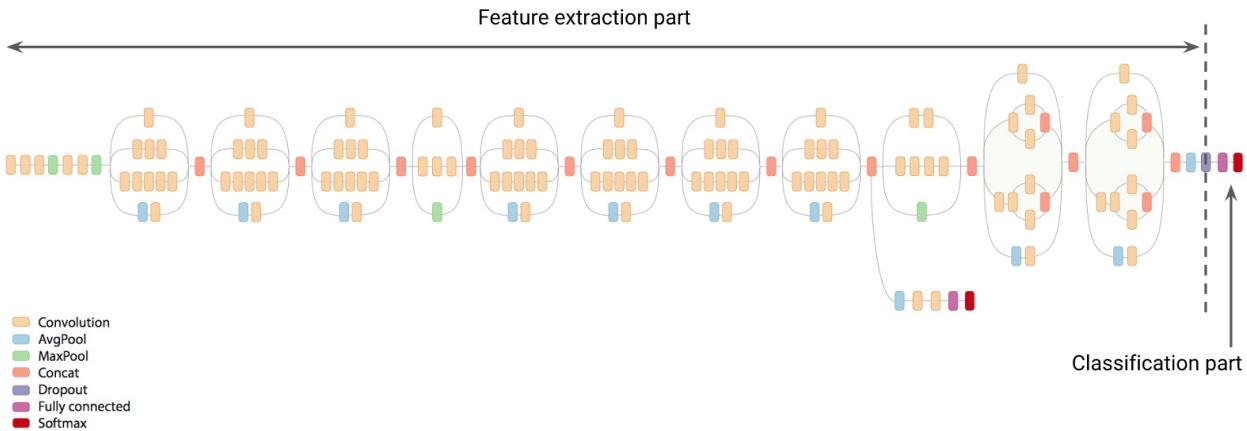


Figure: Google's Inception V3 architecture. Reused from `https://codelabs.developers.google.com/codelabs/cpb102-txf-learning/`.

VIZUALIZATIONS GRADCAM — EXAMPLES

Figure: Visualization using gradCAM for image classification task

AUTOENCODERS

- trying to find mapping from input to output = input
  - bottleneck
  - output = decode(encode(input))
- *unsupervised* learning (actually *selfsupervised* learning)

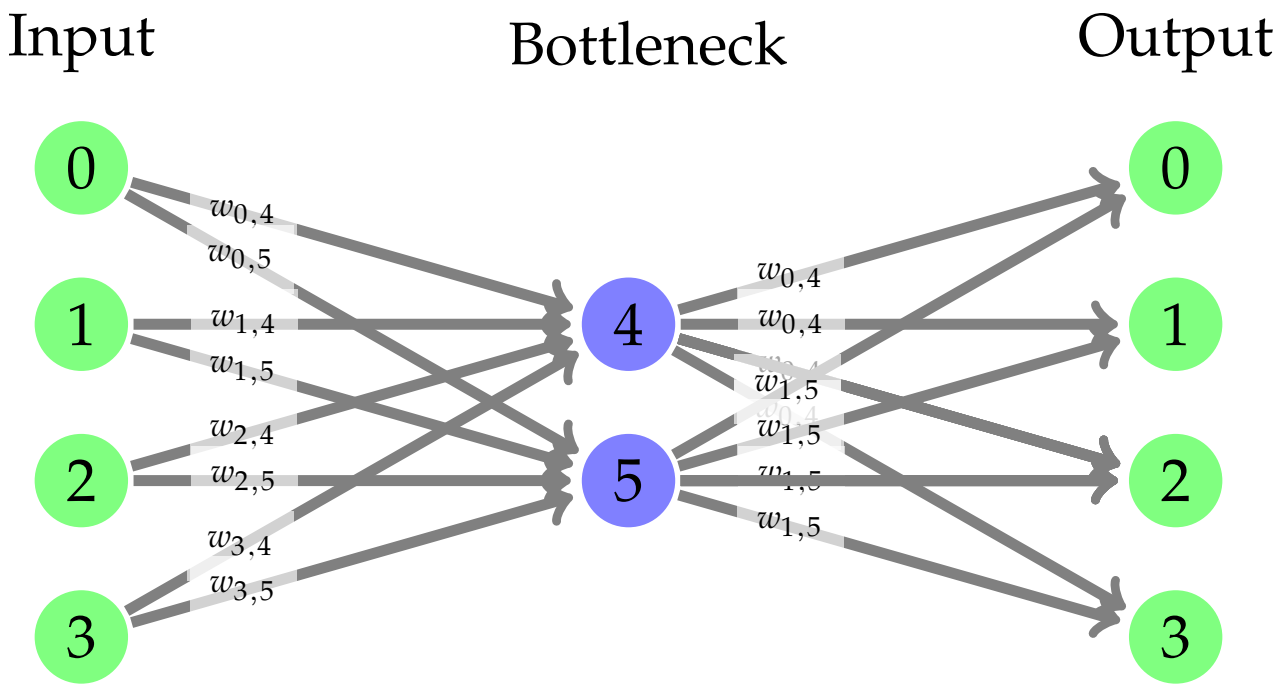


Figure: An example of a simple autoencoder.

AE — EXAMPLE

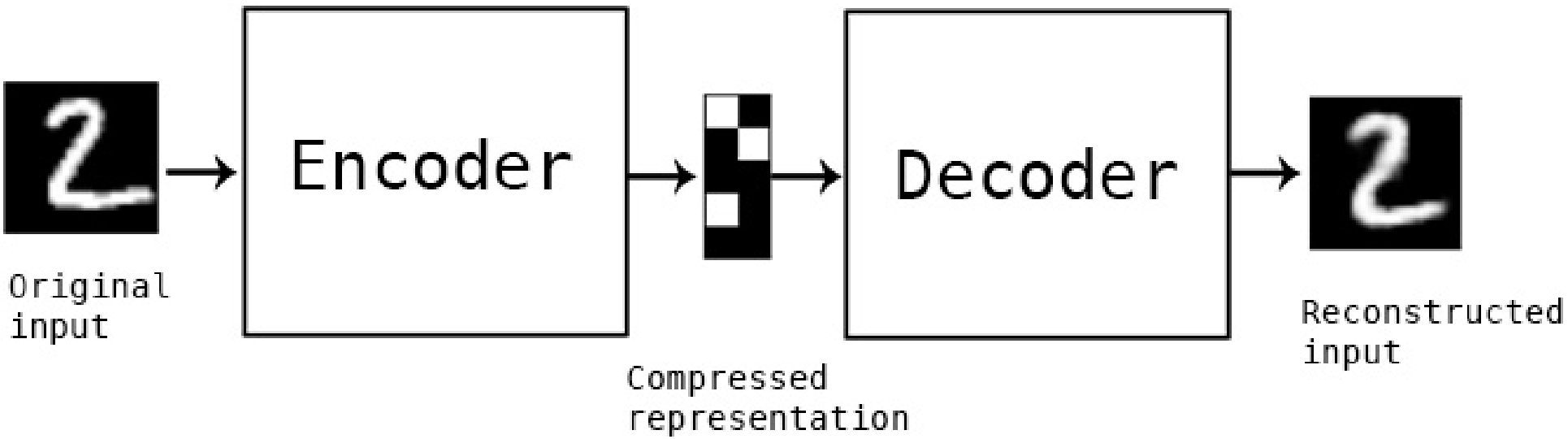


Figure: Schema of AE for MNIST. Taken from `https://blog.keras.io/building-autoencoders-in-keras.html`

AE — TRAINING

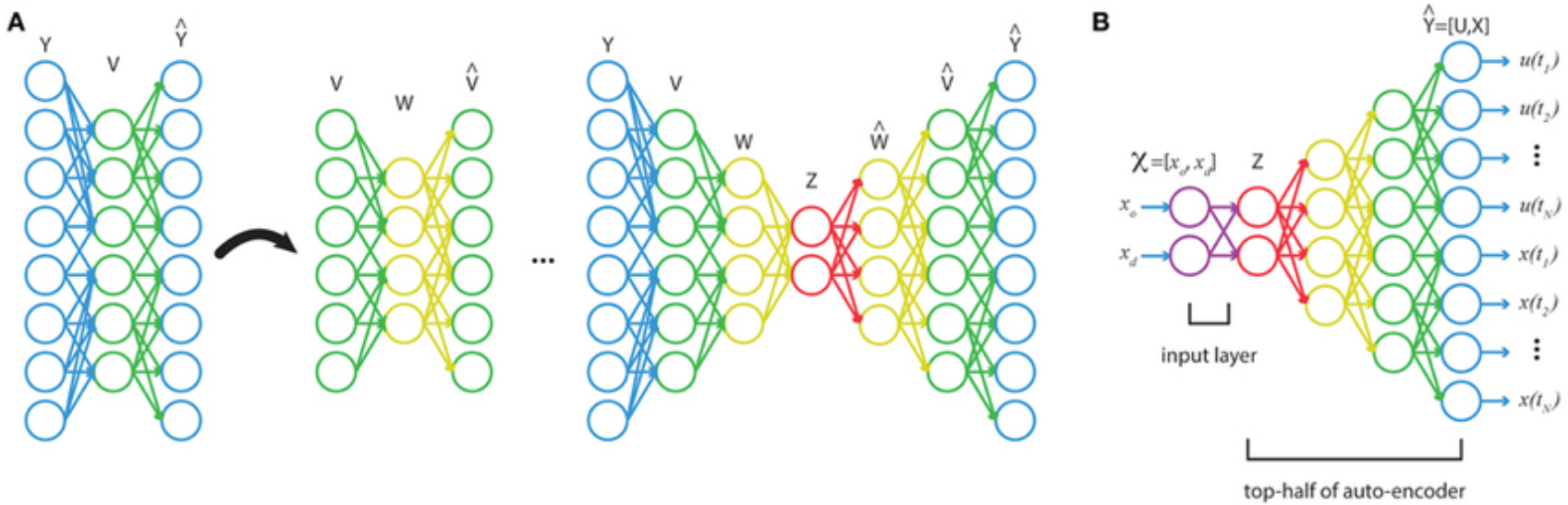
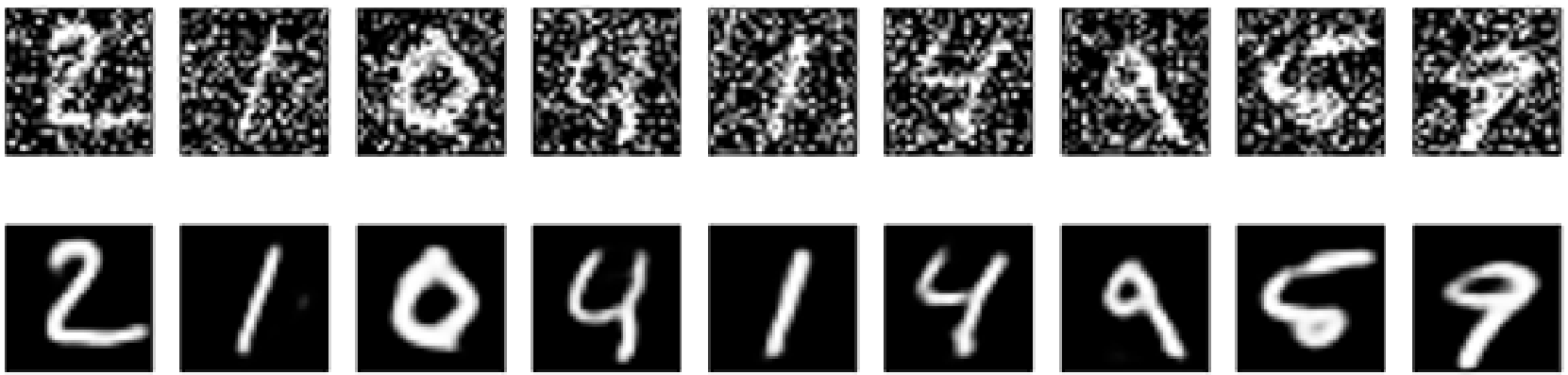


Figure: Possible training procedure consisting of adding bottleneck layers iteratively. Taken from Berniker 2015

APPLICATIONS

- only a few practical applications
  - however, the number is rising
- actually not suitable for compression
  - traditional algorithms are better for compression
- data denoising
- dimensionality reduction for data visualization/analysis
  - for gaining knowledge



VARIATIONAL AUTOENCODER (VAE)

- constraints on the learned representation
- latent variable model

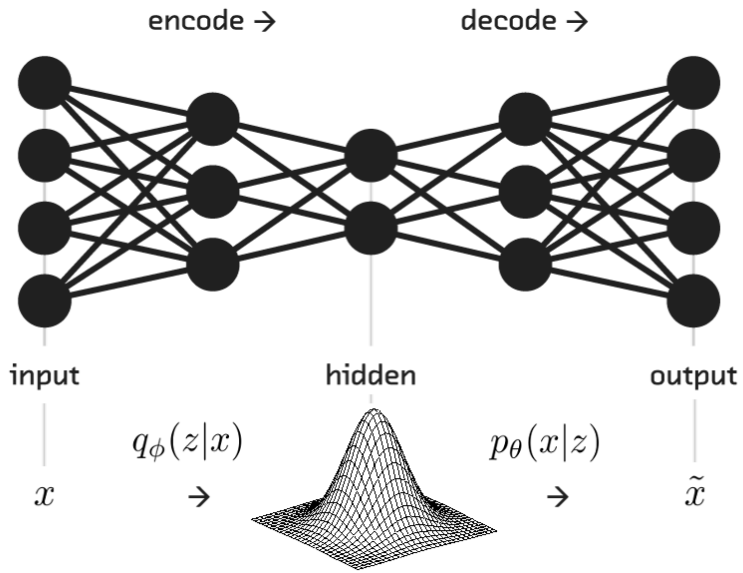


Figure: Visualization of VAE. Taken from
`http://blog.fastforwardlabs.com/2016/08/22/under-the-hood-of-the-variational-autoencoder-in.html`

VAE — MAPPING TO LATENT SPACE

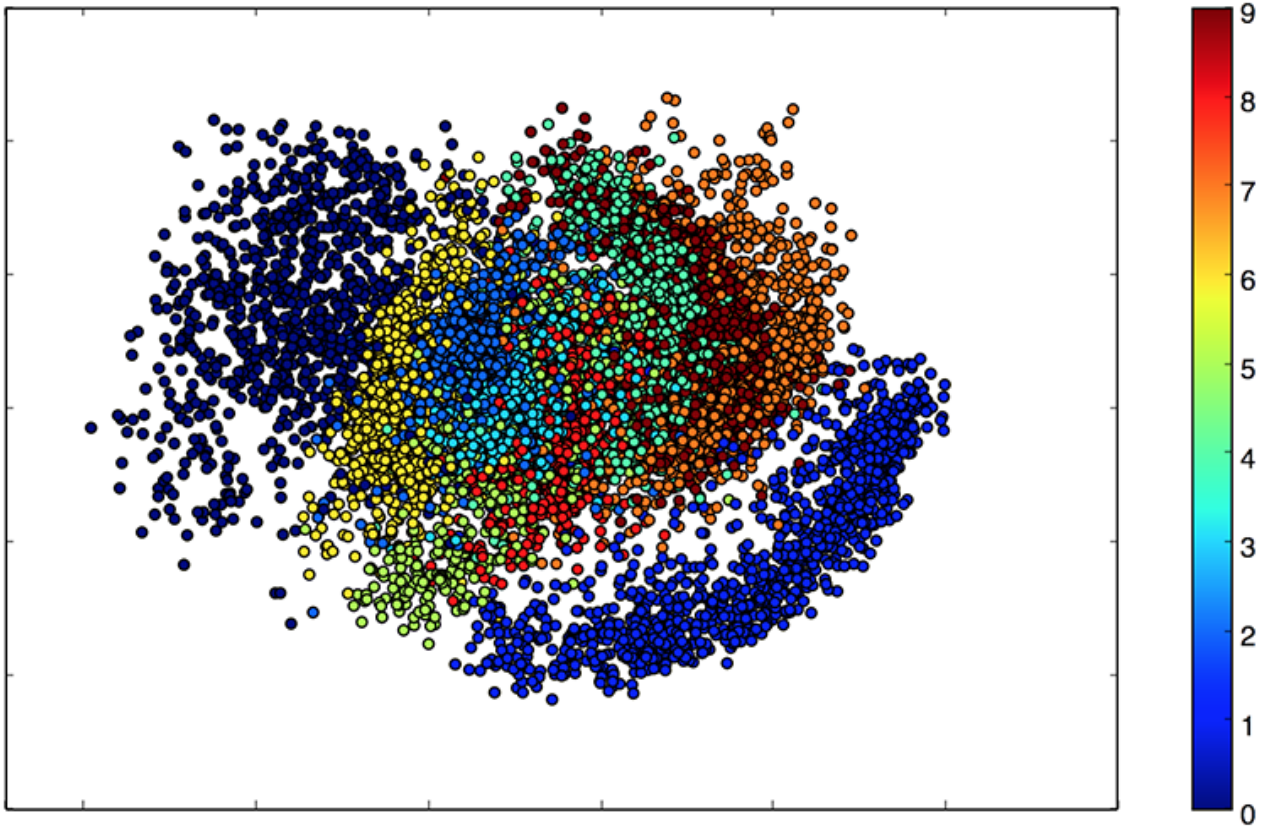


Figure: Visualization of VAE mapping for the MNIST dataset. Taken from `https://blog.keras.io/building-autoencoders-in-keras.html`

VAE — GENERATIVE MODEL

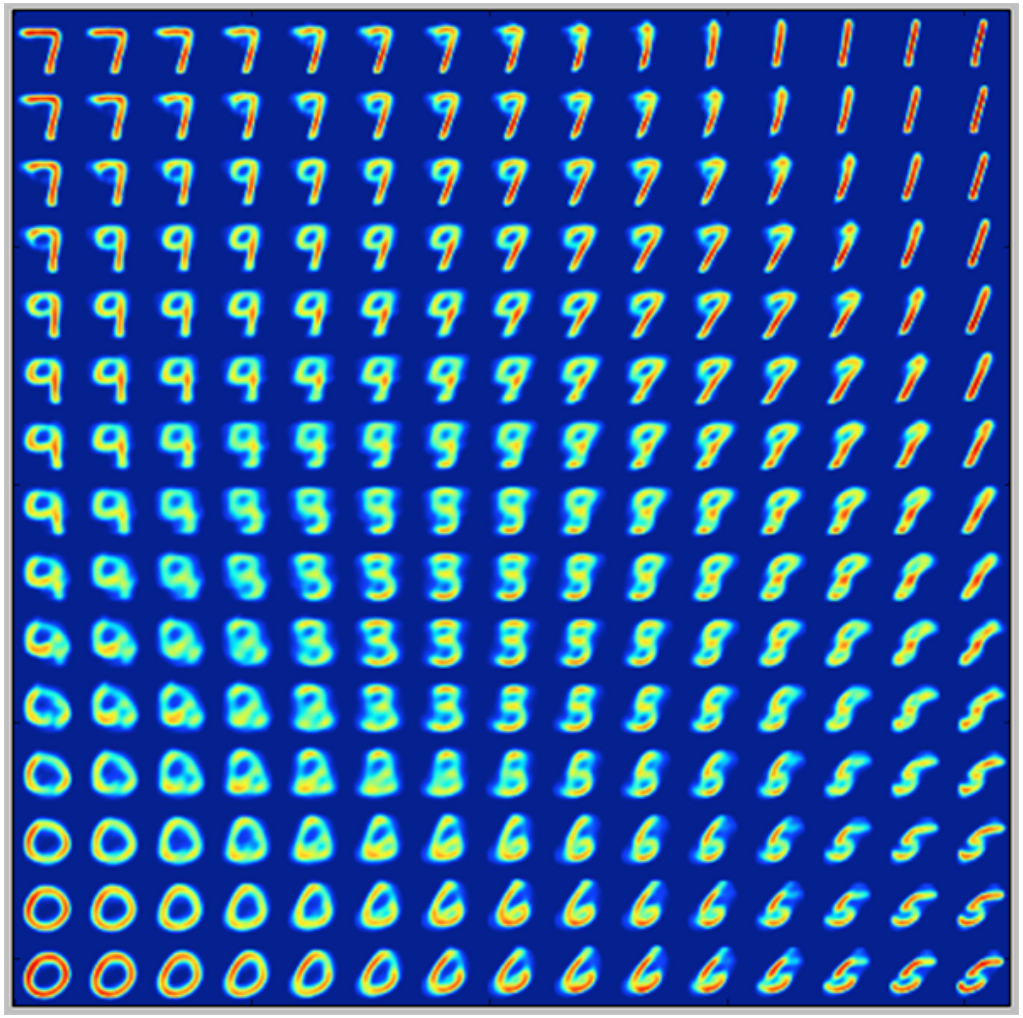


Figure: Visualization of VAE mapping for the MNIST dataset. Taken from `https://blog.keras.io/building-autoencoders-in-keras.html`