Training Deep Neural Networks (part 2) and Network Architectures

Using Machine Learning Tools

Reading: Géron Chapter 11 & Chapter 14

Last Time …

- Training deep NNs uses gradient descent
- Prevent vanishing and exploding gradients with
  - Non-saturating activation functions
  - Batch normalisation layers
  - Gradient clipping
  - Initialisation
- Several options for optimisers:
  - No guarantees on what is best but try starting with recent ones (e.g. Nadam)
- Learning rate is critical - 1cycle schedule is a recent and good method to try

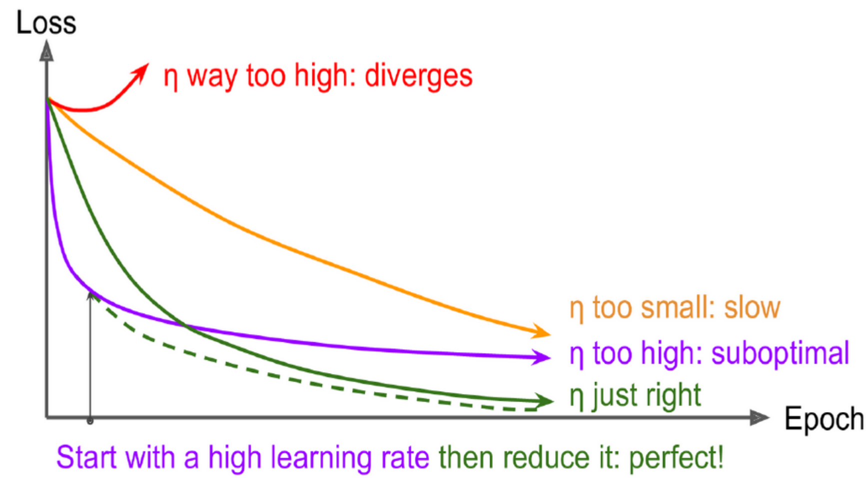


Image: Géron, Hands On ML

L1 and L2 Regularisation

- Prevent overfitting by adding regularisation term (as in lecture 5)
- Optimal solution becomes a compromise between data fit and smaller and/or sparser parameter values

Cost = data_term + α * regularisation_term

$$+ \alpha \frac{1}{2} \Sigma_{i=1}^{n} \theta_i^2$$

L2 norm: keep weights small

$$+ \alpha \Sigma_{i=1}^{n} |\theta_i|$$

L1 norm: eliminate least important features

- Need to adjust α term to make regularisation term similar in magnitude to data term —> *hyperparameter*

```
keras.layers.Dense(.. kernel_regularizer=keras.regularizers.l2(0.01))
```

Dropout Regularisation

- Avoid overfitting by forcing distributed and robust learning
- In every *training iteration* randomly select a set of nodes to drop/zero, based on probability *p* —> hyperparameter
- Training:
  - Dropped nodes output 0
  - Divide remaining node weights by *(1-p)* to maintain variance of signal
- Prediction:
  - No dropout (All nodes active)
- In practice: *p* = 10-50%, only in last 1-3 layers
- Less often used in CNNs; *if* used then whole filters/layers are usually dropped (but it's a bigger disruption). More often just use dropout in dense layers at end of network.

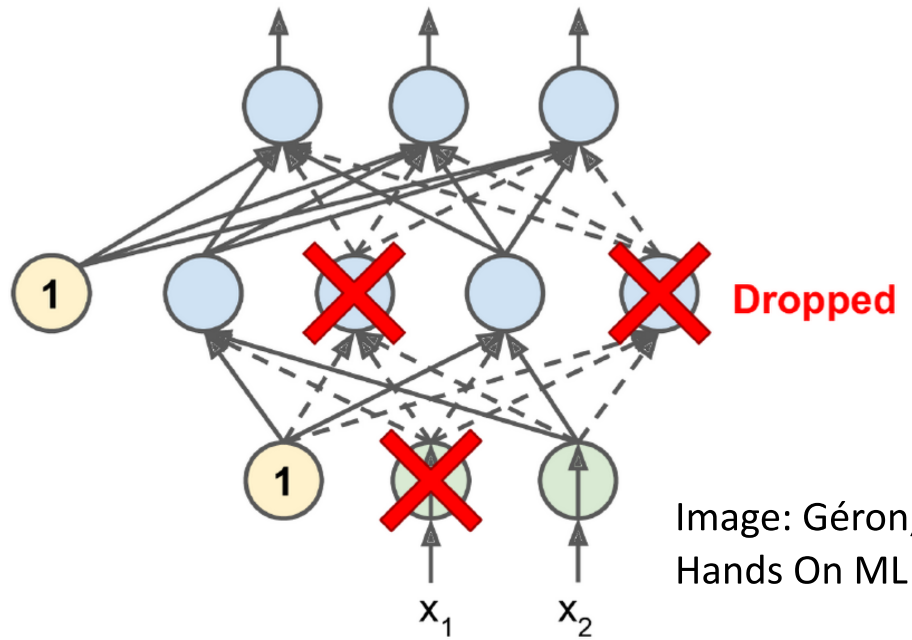


Image: Géron, Hands On ML

```
model=keras.models.Sequential(
..
keras.layers.Dropout(rate=0.2)
keras.layers.Dense(100)
keras.layers.Dropout(rate=0.2)
keras.layers.Dense(3,activation="softmax")
)
```

Monte Carlo Dropout

- Monte Carlo approach = repeated random sampling to provide a probabilistic output/answer

- For Deep Neural Networks:
  - Train NN with dropout layers as before
  - *Predict repeatedly with dropout layer active*
    - Every prediction uses different randomly dropped nodes
  - Take average/distribution of output values = probabilities

- Benefits:
  - More robust confidence estimate
  - Class confusions highlighted
- Drawbacks:
  - Slower and does not necessarily provide the type of probability desired (i.e. it is a probability over different models, not inputs)

Max-Norm Constraint

- Soft constraint: condition that is preferred
  - E.g. implemented as a cost function term (like regularisation)
- Hard constraint: condition that must be satisfied

- Max-Norm Constraint: $\|\boldsymbol{w}\|_2 \leq r$
  - *Hyperparameter* = **maximum norm r** of **weight vector w** (per layer)
  - Implemented as:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} * r / \|\boldsymbol{w}\|_2$$

  - Acts similarly to regularisation

```
keras.layers.Dense(100, kernel_constraint=keras.constraints.max_norm(1.))
```

Transfer Learning

- Transfer knowledge learned from one problem/ dataset to a similar problem
- A commonly used method to (often massively) speed up training/convergence
- In practice:
  - *Copy layers* of an existing NN model to make initial model for new problem
  - Start from bottom, *lower level features most transferable*
  - Can also *make reused layers trainable* from top to bottom (one at a time)
- Benefits:
  - Convergence faster, because initialisation closer to optimum
  - Needs less data (good for situations with small datasets)
- Drawbacks:
  - Inputs must be same type (e.g. 1D/2D/3D, RGB/ grayscale)
  - Existing DNN may not generalise to new problem well

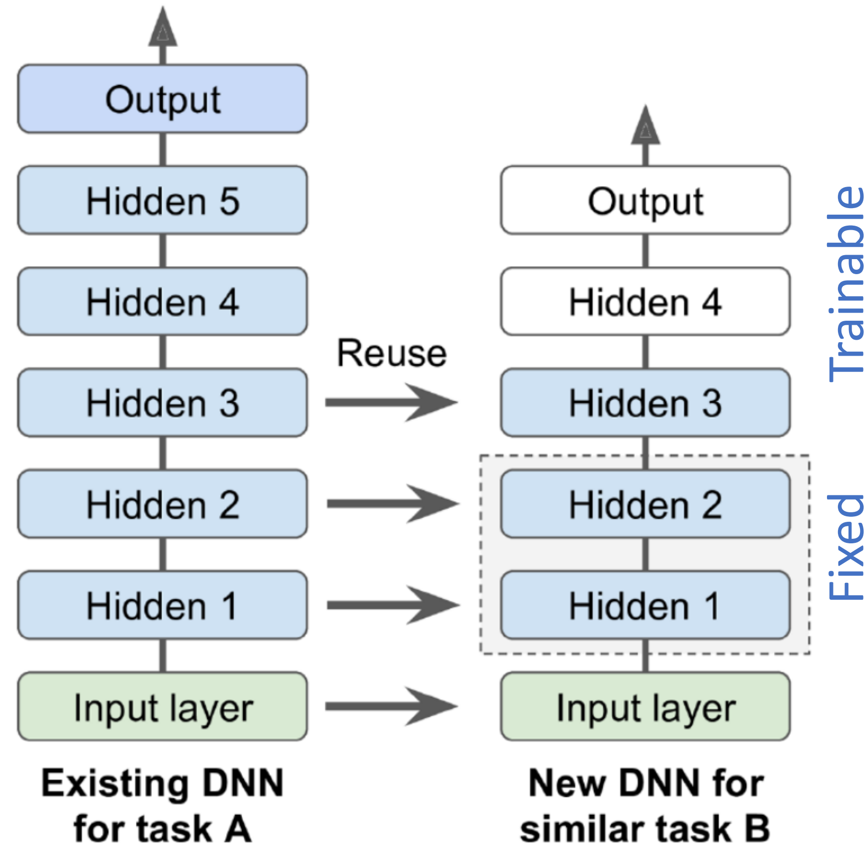


Image: Géron, Hands On ML

Part 2

Network architectures

LeNet-5 (LeCun et al. 1998)

- First modern CNN published, applied to MNIST handwritten digits data set
- Low-level customised network design - most of the details are no longer used (e.g. average vs max pooling) but the overall style is still followed

Euclidean radial basis function: square of difference of inputs and weights

Multiply mean with learned weight per feature map and add learned bias term

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully connected	-	10	-	-	RBF
F6	Fully connected	-	84	-	-	tanh
C5	Convolution	120	1x1	5x5	1	tanh
S4	Avg. pooling	16	5x5	2x2	2	tanh
C3	Convolution	16	10x10	5x5	1	tanh
S2	Avg. pooling	6	14x14	2x2	2	tanh
C1	Convolution	6	28x28	5x5	1	tanh
In	Input	1	32x32	-	-	-

Selected connection patterns only

No padding

Zero pad from 28x28 to 32x32 & normalise data

AlexNet (2012)

Highlighted elements still commonly used today

- Published by Alex Krizhevsky et al.
- *Dropout* 50% applied to outputs of F9&F10
- Introduced *stacks of convolution layers*
- Local response normalisation of outputs of C1 & C3
- *Data augmentation* with random shifts, flipping, lighting

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully connected	-	1000	-	-	-	Softmax
F10	Fully connected	-	4096	-	-	-	ReLU
F9	Fully connected	-	4096	-	-	-	ReLU
S8	Max pooling	256	6x6	3x3	2	Valid	-
C7	Convolution	256	13x13	3x3	1	Same	ReLU
C6	Convolution	384	13x13	3x3	1	Same	ReLU
C5	Convolution	384	13x13	3x3	1	Same	ReLU
S4	Max pooling	256	13x13	3x3	2	Valid	-
C3	Convolution	256	27x27	5x5	1	Same	ReLU
S2	Max pooling	96	27x27	3x3	2	Valid	-
C1	Convolution	96	55x55	11x11	4	Valid	ReLU
In	Input	3 (RGB)	227x227	-	-		-

- Use "same" padding

Data Augmentation

- Increase (augment) dataset with *realistic variations* of the available data
- Reduce overfitting, *increase generalisability*
- For images: rotation, position in image, size/cropping/flipping, blurring, intensity distribution, cutout/occlusion, distortion, artefacts, etc…

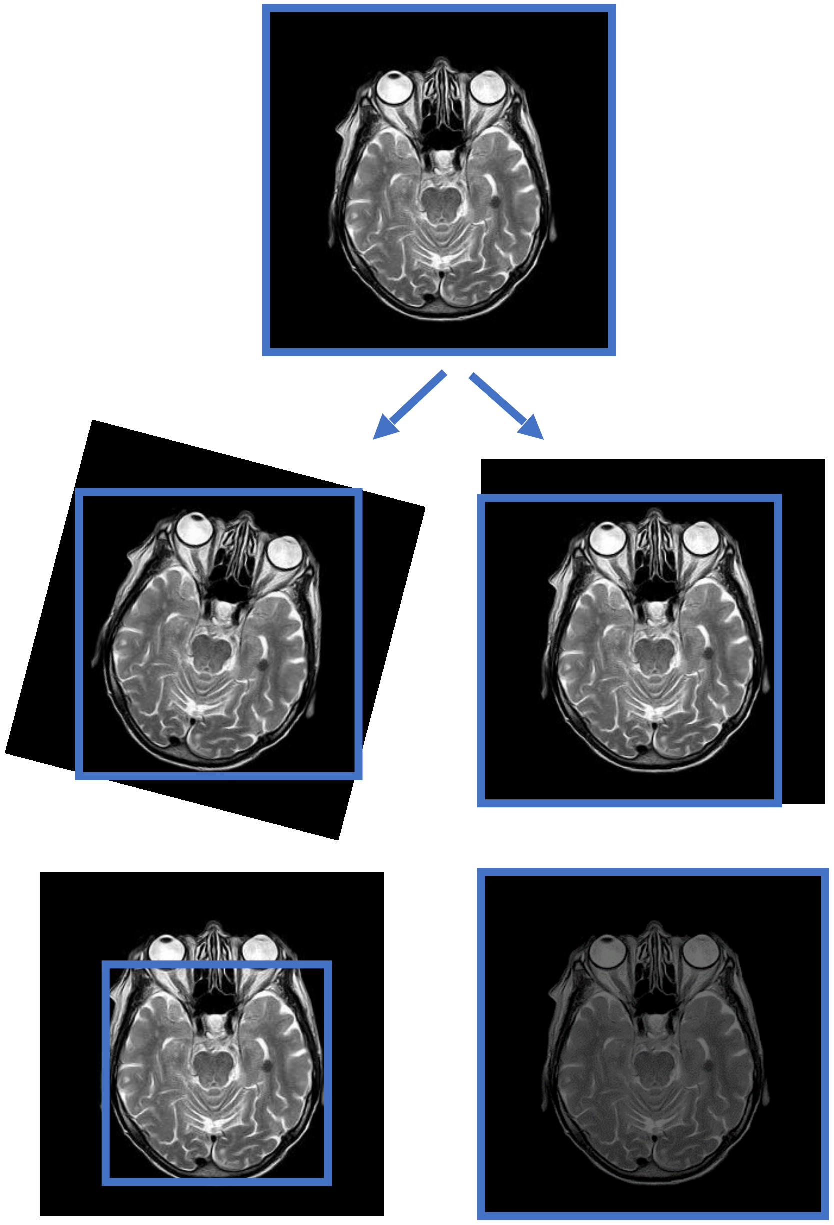


Image at top: Pixbay open license by toubibe.

VGGNet (2014)

Highlighted elements still commonly used today

- VGG = Visual Geometry Group, University of Oxford
- Investigated influence of *depth* (11 to 19 layers)
- *Small 3x3 filters*, stride 1, zero-padding, ReLU activation
- Better accuracy by *increasing depth* (16-19 layers)
- Sequence of many small filters replicates effect of larger filters

LRN = Local Response Normalisation

Each column describes a different network

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

FC = Fully connected

1000 classes → FC-1000

Validation error decreases →

Image: Simonyan & Zisserman 2015, Very deep convolutional networks for large-scale image recognition, https://arxiv.org/abs/1409.1556

Common Convolutional Networks

- The most typical/conventional CNNs in current usage are like this…
  - *If* needed, first layer to can reduce resolution with large kernel & stride (e.g. 5x5, stride 2)
  - One or more convolution layers, each with activation (e.g. ReLU), then pooling layer, then repeat
  - *Height & width decreases* through convolutional section
  - *Feature maps (depth) increases*
  - *Fully connected (dense) layers* at end, with activation (e.g. ReLU)
  - At top, prediction/output layer (typically softmax)

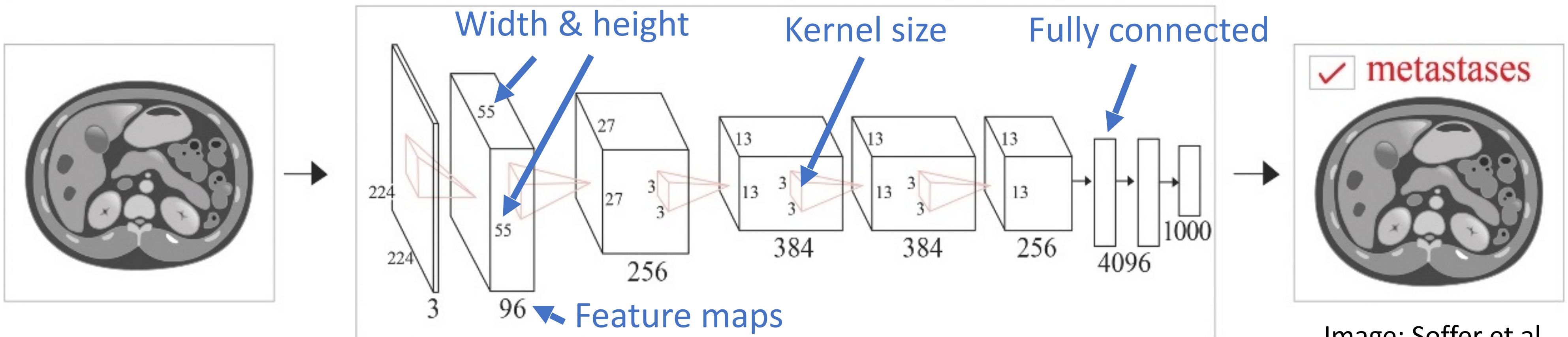


Image: Soffer et al. Radiology 2019

Residual Network – ResNet (2015)

- Much deeper CNN (34, 50, 101, 152 layers)
- *Skip connection* adds input of a block of layers to the output
- Residual learning: network learns how to predict residual = target - input
- Speeds up training if target is similar to inputs (*skip* layers)
    - Helps avoid vanishing gradients

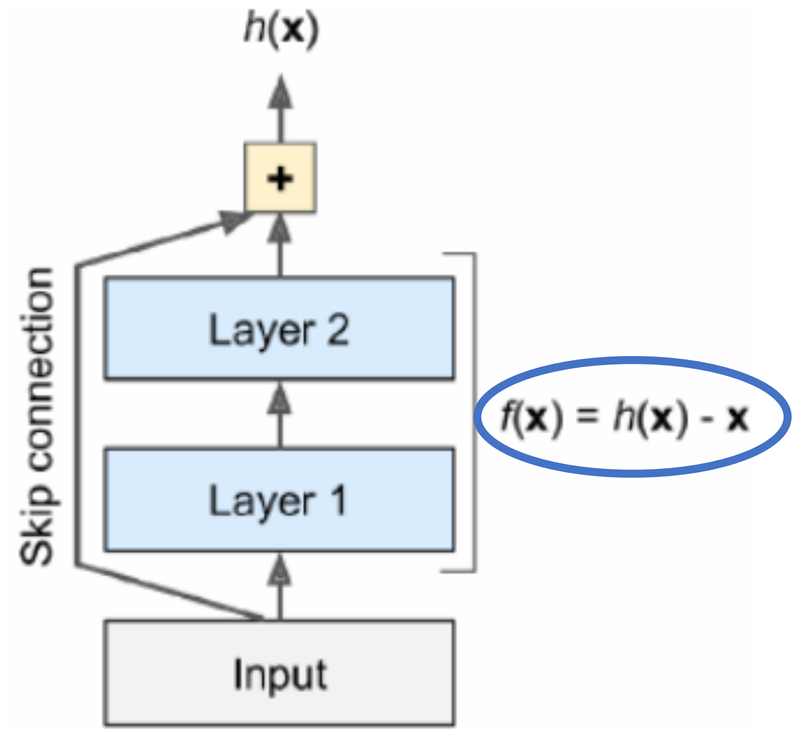
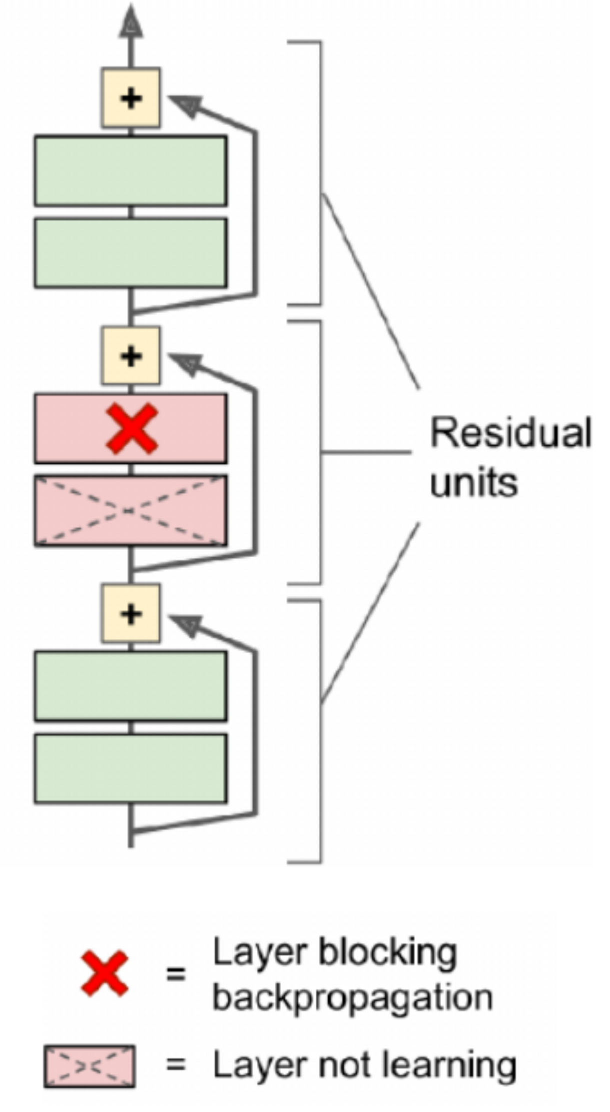


Image: Géron, Hands On ML

Residual Unit

- Residual Unit:
  - A stack of convolutional layers with a skip connection around them
  - Usually no pooling layer (keeping same size in residual block)
  - Can have many residual units in one network
  - Can exclude residual units during training (trainable=False)
  - Can train selected units at different times



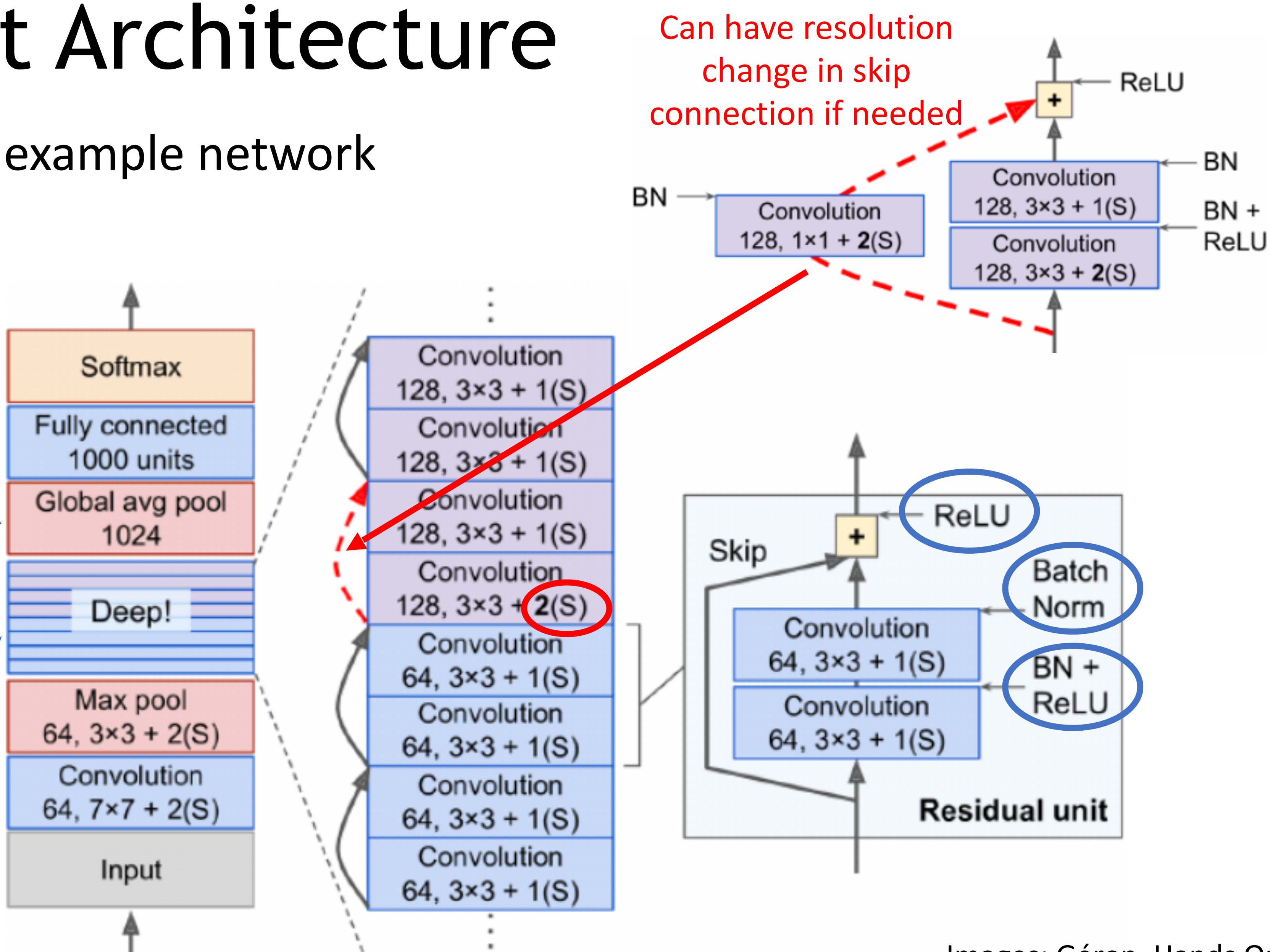
Images: Géron, Hands On ML

ResNet Architecture

An example network

Global average pooling: Vector of means of each feature map

Over 100 layers!

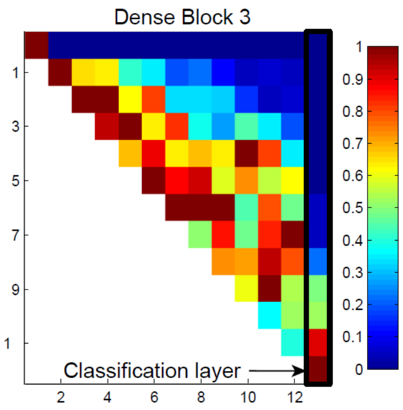


Images: Géron, Hands On ML

DenseNet – Densely Connected CNN (2018)

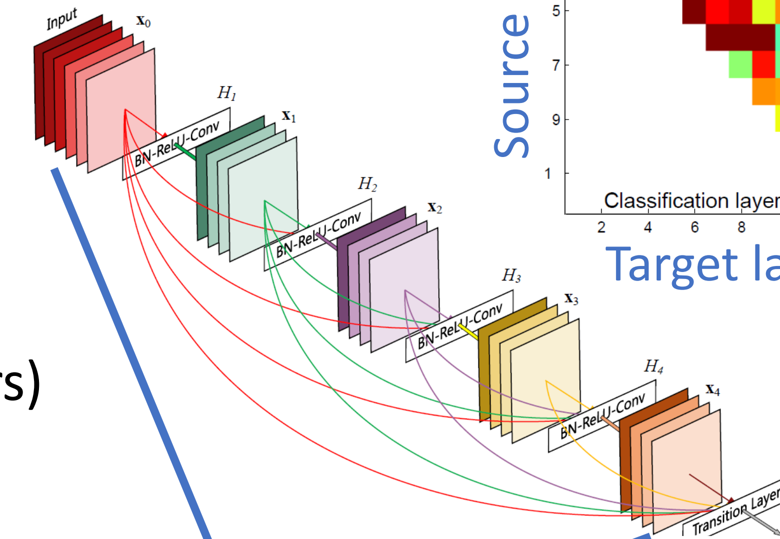
- Lots of skip connections
- Connect each layer to every subsequent layer (usually concatenating)
- Combine dense blocks with transition layers
- Transition layers can reduce feature map sizes

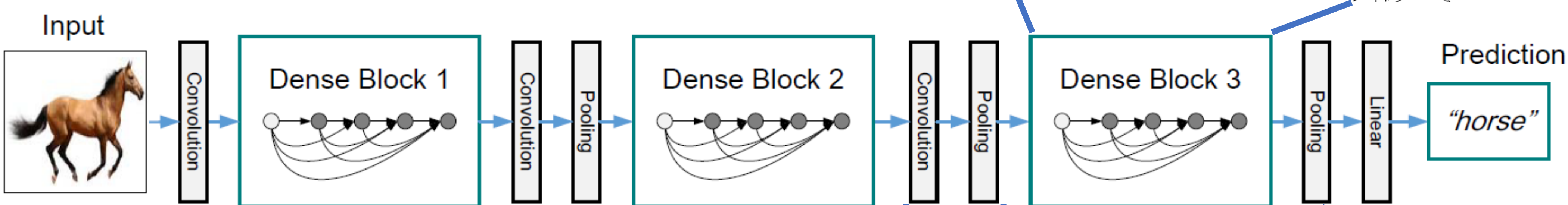
- Deep supervision through short connections - helps bigger/deeper networks (e.g. 200+ layers) to be trainable



Source layer

Target layer





Images: Huang 2018 Densely connected convolutional networks. Available online: https://arxiv.org/abs/1608.06993

Transition layer: 1x1 conv, 2x2 avg pool, stride 2

Global avg pool

Complex Networks, Functional API

Functional API: connect layer objects by *manually defining inputs and outputs* via function arguments

- *Multiple inputs*, e.g. different types
  - Wide paths for simple patterns
  - Deep paths for complex patterns
- *Multiple outputs*, e.g. different tasks on same data, or auxiliary output used for regularisation
  - Multiple loss functions

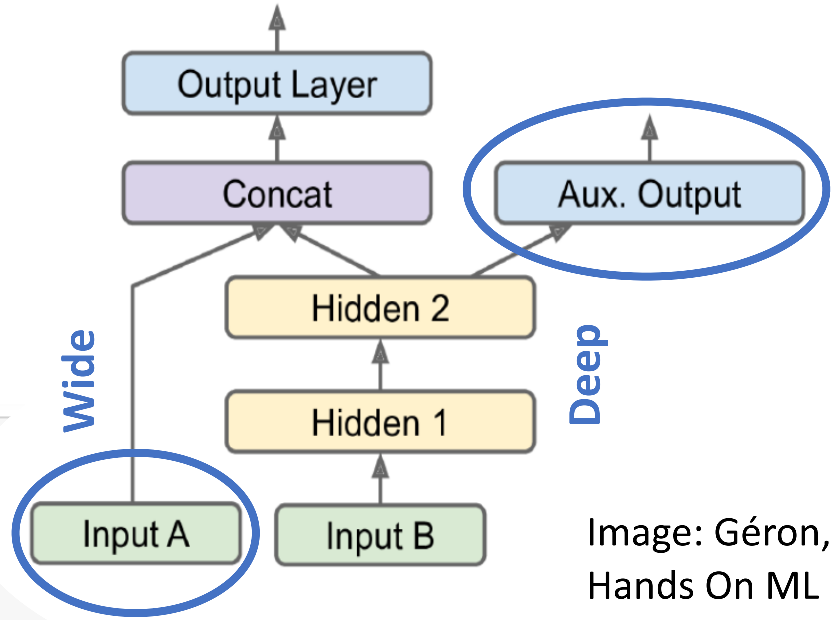


Image: Géron, Hands On ML

```
input_A = keras.layers.Input(shape=[5], name="wide_input")
input_B = keras.layers.Input(shape=[6], name="deep_input")
hidden1 = keras.layers.Dense(30, activation="relu")(input_B)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.concatenate([input_A, hidden2])
output = keras.layers.Dense(1, name="main_output")(concat)
output_aux = keras.layers.Dense(1, name="aux_output")(hidden2)
model = keras.Model(inputs=[input_A, input_B], outputs=[output, output_aux])
model.compile(loss=["mse", "mse"], loss_weights=[0.9, 0.1], optimizer="sgd")
history = model.fit([X_train_A, X_train_B], [y_train, y_train_aux], epochs=20,
                    validation_data=([X_valid_A, X_valid_B], [y_valid, y_valid_aux]))
total, test, test_aux = model.evaluate([X_test_A, X_test_B], [y_test, y_test_aux])
y_pred, y_pred_aux = model.predict([X_new_A, X_new_B])
```

Adjust relative influence of multiple outputs

Summary

- Options to *reduce overfitting* with Regularisation, Dropout, Max-Norm Constraint
- *Transfer learning*: Reuse lower layers in new network as a good initialisation
  - Earlier network layers: more general, lower-level features → more transferable to similar task
- Architectures:
  - Overall sequential from inputs to outputs
  - Can include branching, joining, skipping over layers, recurrency
  - Many more sophisticated architectures exist, adapted to specific tasks
  - Key design elements:
    - Step sequence: feature extraction, downsampling, upsampling, merging
    - Depth: how many steps, how many layers per step, how many filters
    - Connectivity: which nodes are connected to which other nodes
    - Complexity: number of parameters, convergence, deployment size
  - Best network choice depends on many factors, especially dataset size and number of parameters - more recent developments are not always better, but worth trying!