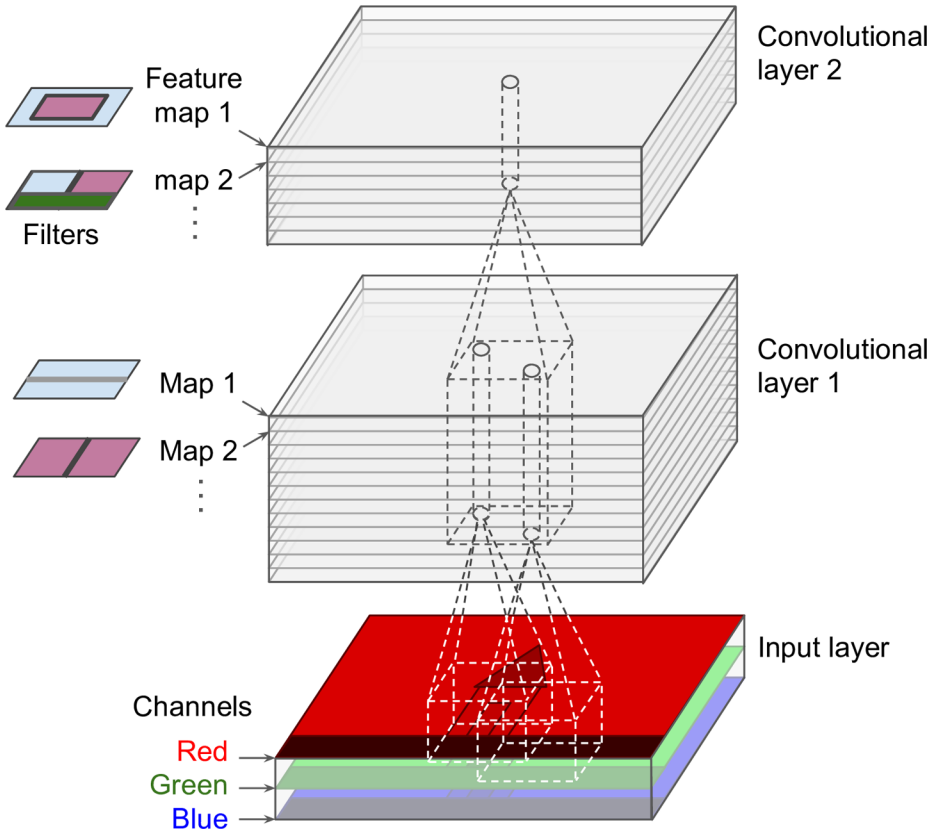
CMT307 Applied Machine Learning

Session 13

Convolutional Neural Networks: Architectures, Applications and Fine-tuning; Autoencoders

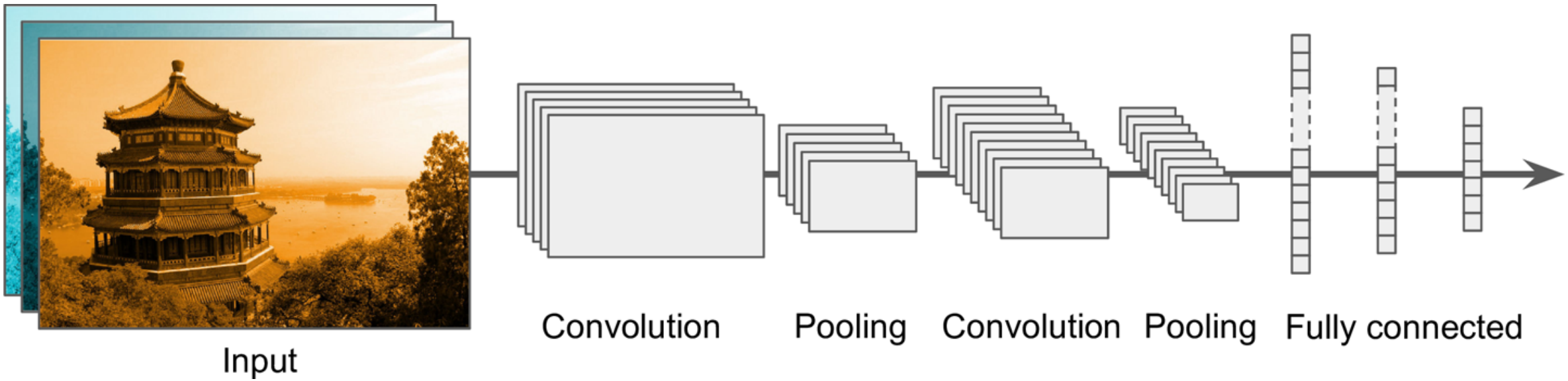
Recap: Convolutional Layers

- Key ideas:
  - Multiple filters lead to multiple feature maps at the same layer
  - Local receptive fields: convolutional kernels are local (unlike fully connected layers)
  - The same convolutional kernel is applied to different spatial locations (weight sharing)
  - Both lead to substantial reduction of weight parameters



Recap: Convolutional Neural Networks (CNNs)

- Typical architecture



Recap: CNNs – A typical example

```
model = keras.models.Sequential([
    keras.layers.Conv2D(64, 7, activation="relu", padding="same", input_shape=[28, 28, 1]),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation="softmax")
])
```

Typical CNN Architectures

- We talk about a few well-known CNN architectures (for image classification, but can be adapted for other applications)
  - AlexNet
  - GoogLeNet (a.k.a. InceptionNet)
  - VGGNet
  - ResNet
  - Xception
- They often involve new ideas for training, and new modules that can be reused in new architectures

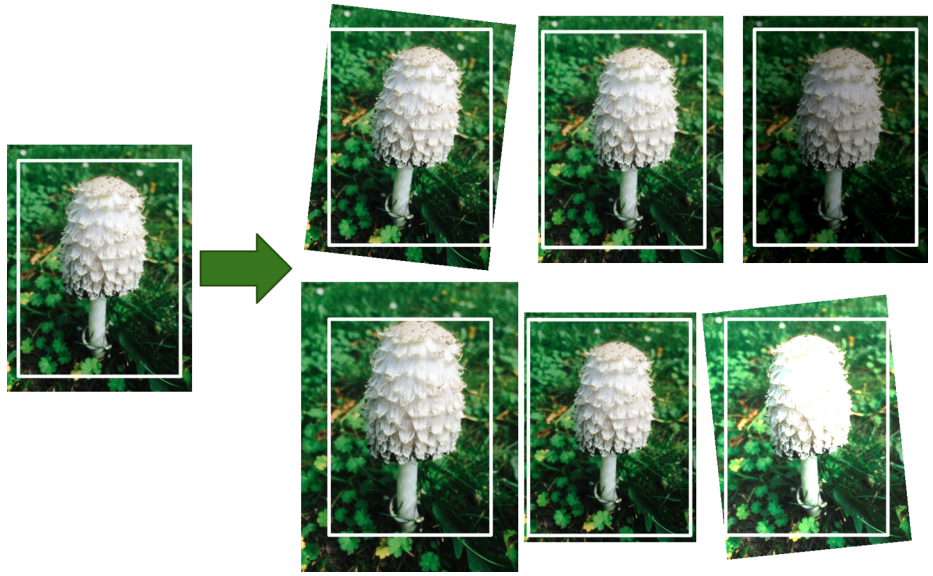
AlexNet

- Developed by Alex Krizhevsky and colleagues
- Won 2012 ImageNet ILSVRC (Large Scale Visual Recognition Challenge): (top-5 error rate 17% vs. second best 26%)
- Input: 227×227 RGB image
- Output: probability of 1,000 classes
- Use stacked convolutional layers

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully connected	–	1,000	–	–	–	Softmax
F10	Fully connected	–	4,096	–	–	–	ReLU
F9	Fully connected	–	4,096	–	–	–	ReLU
S8	Max pooling	256	6 × 6	3 × 3	2	`valid`	–
C7	Convolution	256	13 × 13	3 × 3	1	`same`	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	`same`	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	`same`	ReLU
S4	Max pooling	256	13 × 13	3 × 3	2	`valid`	–
C3	Convolution	256	27 × 27	5 × 5	1	`same`	ReLU
S2	Max pooling	96	27 × 27	3 × 3	2	`valid`	–
C1	Convolution	96	55 × 55	11 × 11	4	`valid`	ReLU
In	Input	3 (RGB)	227 × 227	–	–	–	–

AlexNet: Data Augmentation

- Training data is often limited
- Data augmentation gets additional training data “for free”:
  - Reducing overfitting
  - Making the model more generalisable
  - Changes should be as realistic as possible
  - Changes should be learnable (not random noise)
- Examples:
  - Translation, Rotation, Resizing, Flip, Contrast Adjustment, etc.
- Widely used in training of modern deep networks



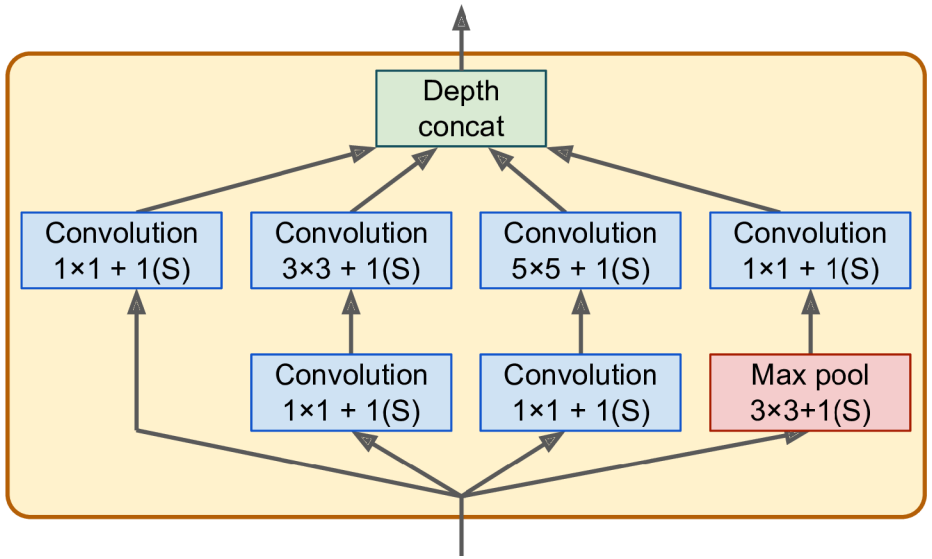
AlexNet: Local Response Normalisation (LRN)

- Local Response Normalisation
  - The most strongly activated neurons inhibit other neurons located at the same position in neighbouring feature maps
  - Similar competitive activation has been observed in biological neurons
  - r: controls the range of feature maps
  - AlexNet uses $r = 2$: a strong neuron will inhibit neurons at the same spatial location and adjacent feature maps (immediately above or below the current)

$$b_i = a_i \left(k + \alpha \sum_{j=j_{\text{low}}}^{j_{\text{high}}} {a_j}^2 \right)^{-\beta} \quad \text{with} \quad \begin{cases} j_{\text{high}} = \min \left(i + \frac{r}{2}, f_n - 1 \right) \\ j_{\text{low}} = \max \left(0, i - \frac{r}{2} \right) \end{cases}$$

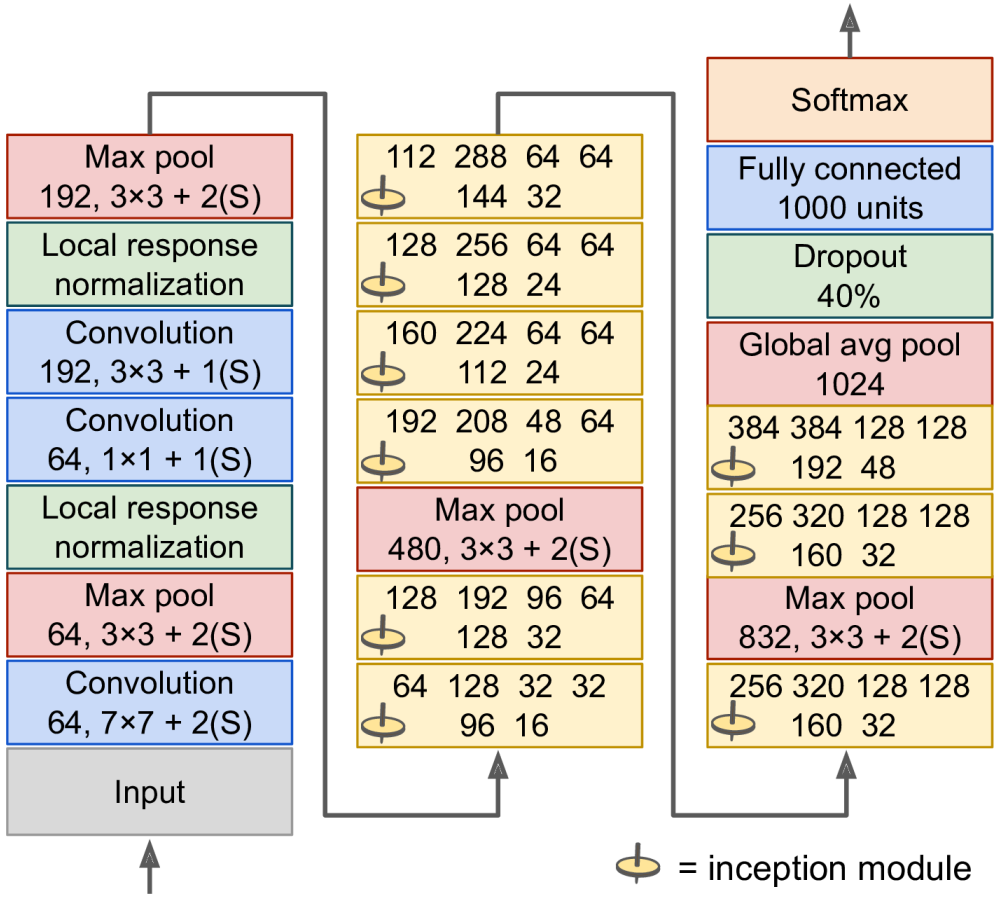
GoogLeNet (InceptionNet)

- Developed by Christian Szegedy et al. from Google Research
- Won ILSVRC 2014 challenge (top-5 error rate < 7%)
- Introduce *Inception* modules
  - Involves 1x1 convolutions
    - Cannot capture spatial patterns
    - But can capture patterns of different feature maps
  - Combining 1x1 with other layers to increase learning capabilities
  - 6 convolutional layers (with learnable weights)
  - Hyperparameters: the number of feature maps for each conv. layer



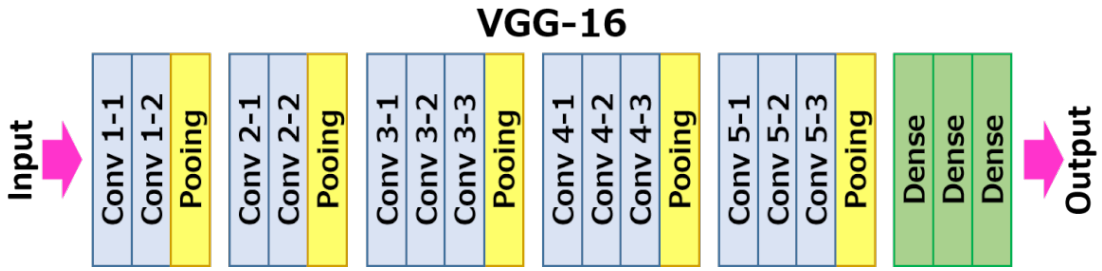
GoogLeNet: Architecture

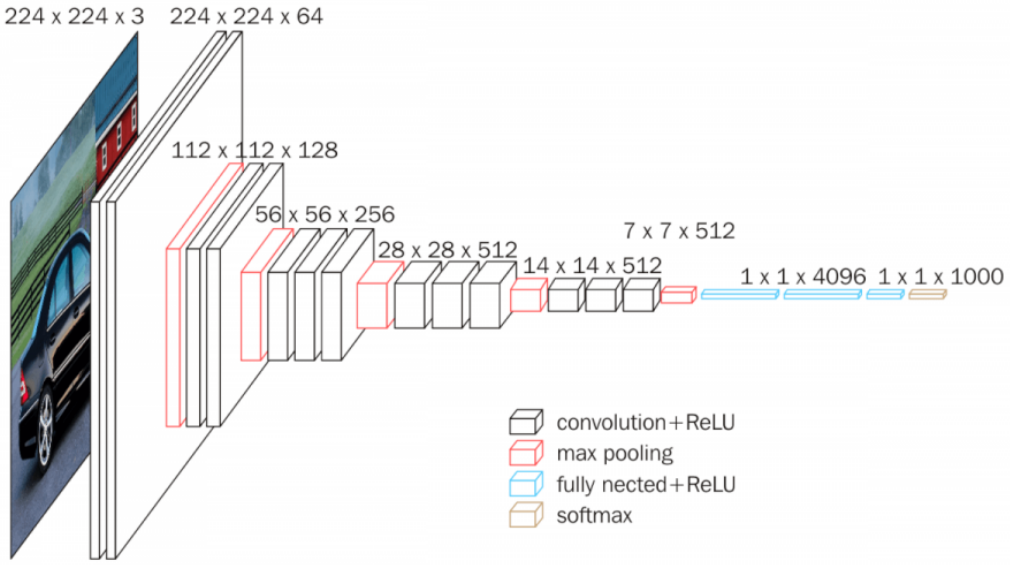
- Much deeper than AlexNet
  - But fewer learnable parameters than AlexNet (6 million vs. 60 million)
  - Use of Inception modules extensively
  - Use of stride 2 layers to reduce spatial resolution



VGGNet

- Runner-up in ILSVRC 2014
- Developed by Karen Simonyan and Andrew Zisserman from the Visual Geometry Group (VGG), University of Oxford
- Classic design, VGG-16 and VGG-19 (16 & 19 layers with learnable weights)



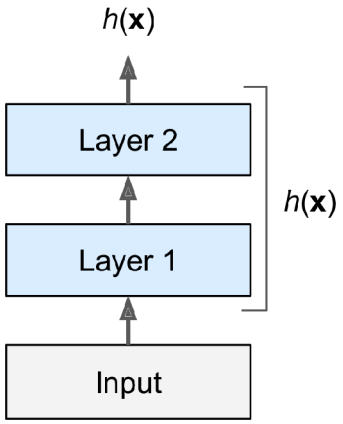


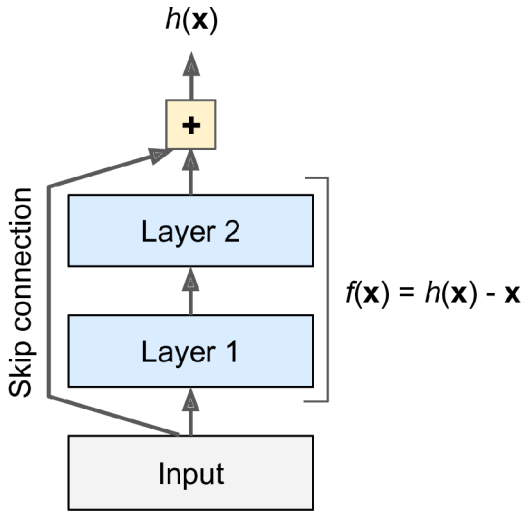
ResNet

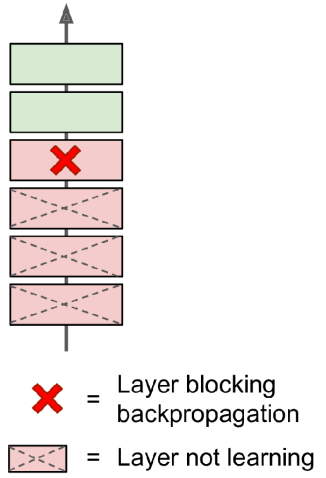
- Meaning *Residual Network*
- Developed by Kaimin He et al.
- Won ILSVRC 2015 with top-five error rate < 3.6%
- Very deep network architecture
  - Different variants with 34, 50, 101 and 152 layers
  - But this will make training difficult
  - The key is using *Residual Units*

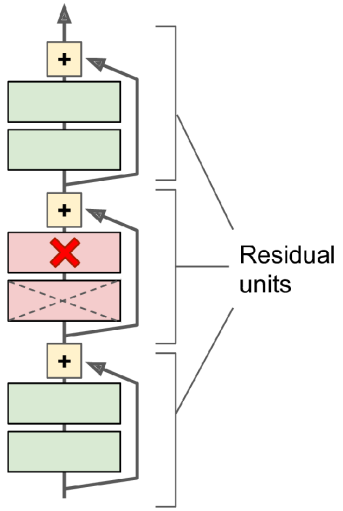
Residual Learning

- Structure
  - Skip connection with the input combined as part of output
  - More efficient to train:
    - Initialised network layers tend to output 0, which now replicates the input (often expected)
    - Layers not learning are skipped over, and won't block the overall network to improve



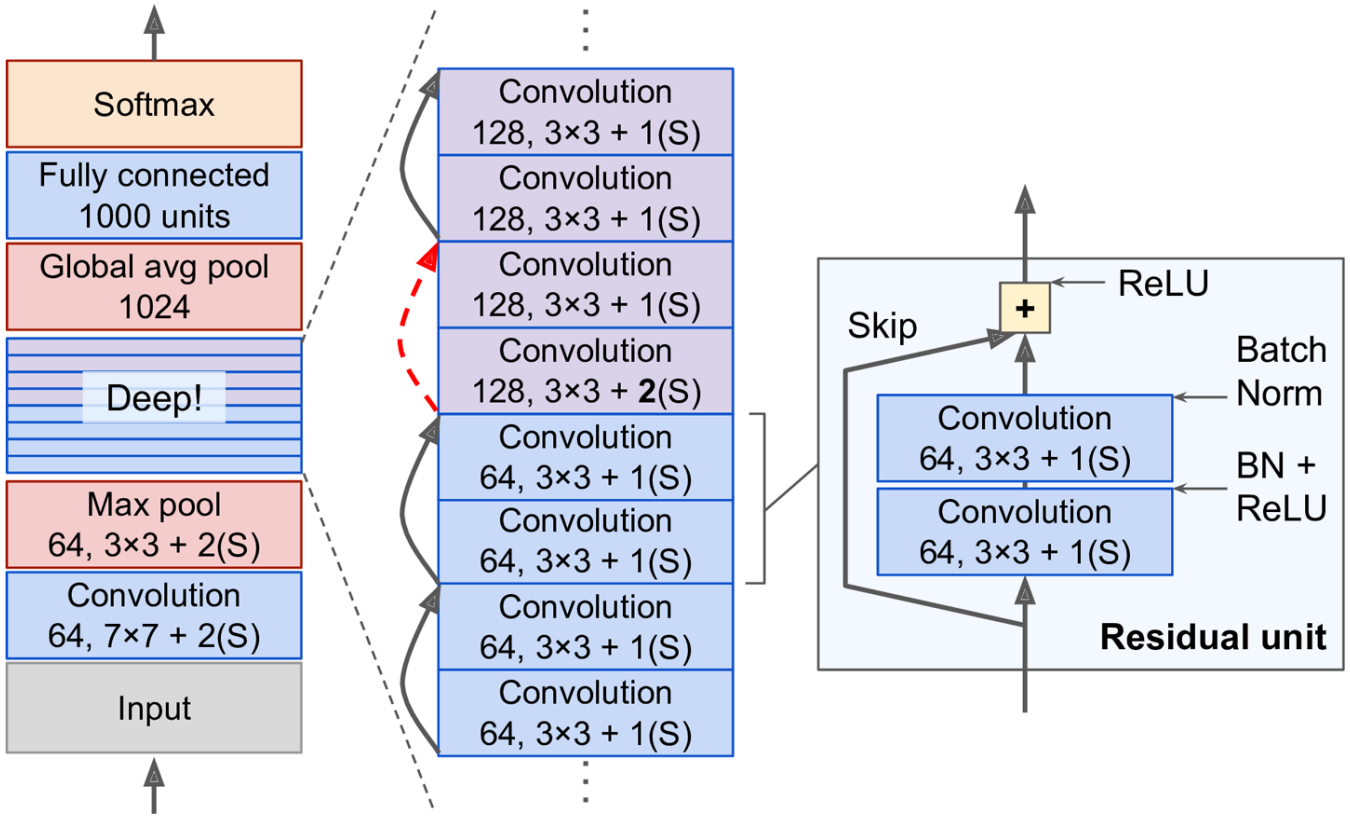




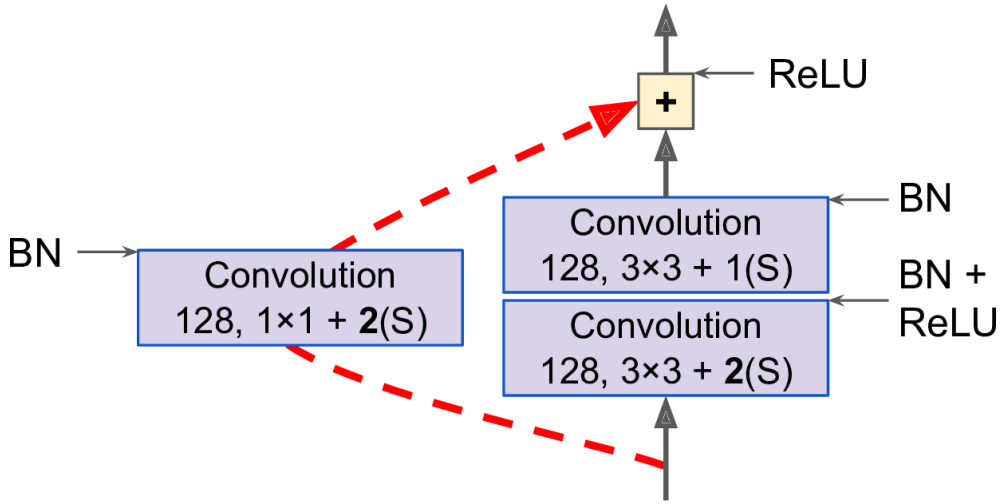


ResNet: Architecture

- Starts/ends as GoogLeNet
- Deep stack of Residual Units



- Skip connection when changing feature map size/depth

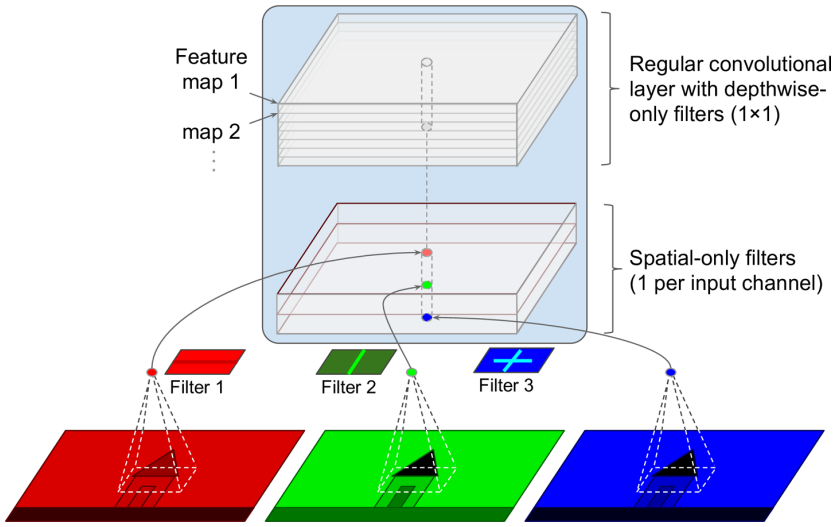


ResNet

- Residual Units are widely used as building blocks in other network architectures
- For example, Google Inception-v4 combines GoogLeNet and ResNet ideas, achieving ~3% top-five error rate

Xception

- Meaning *Extreme Inception*
- Similar to Inception-v4, combining the ideas of GoogLeNet and ResNet
- But replace Inception modules with Depthwise Separable Convolutional Layers
  - Spatial only filters
  - Cross-channel patterns (1x1 convolutional layer)
- Assumes spatial and cross-channel patterns can be modelled separately
- Fewer parameters, less memory, fewer computations, and even perform better
- Not suitable for very few channels (e.g. should apply after some normal conv. layers)



Using Pre-trained Models

- Training these models can take a large amount of resources
- Keras supports typical models with pre-trained weights on ImageNet
  - E.g. Xception, VGG16, VGG19, ResNet, ResNetV2, InceptionV3, etc.
  - See https://keras.io/applications/ for more details
  - These are for classifying images (into 1,000 categories)
- How do you use this to classify other images?

[Demo: using pre-trained models]

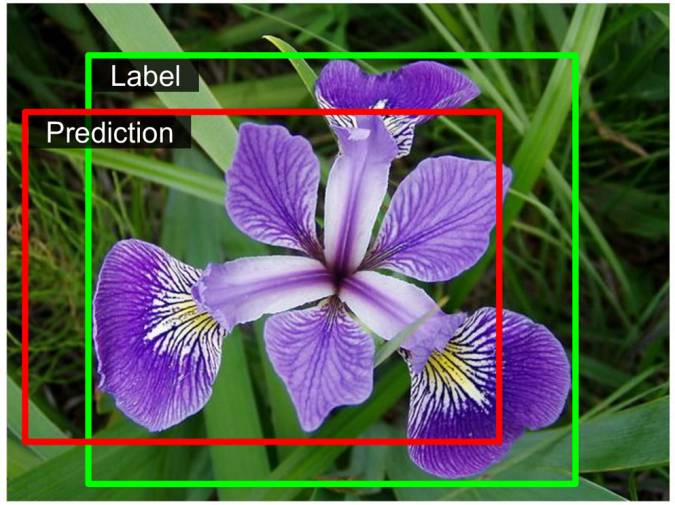
Fine-tuning Existing Models

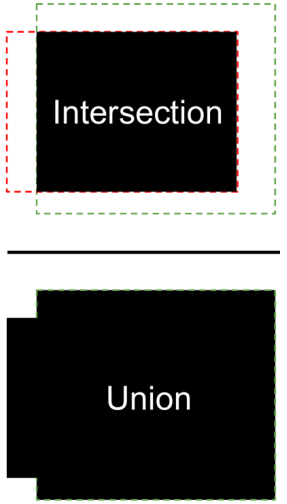
- Although pre-trained models are for a different purpose, it is likely that the learned weights can be useful for other applications
- Transfer Learning
  - Take a pre-trained model
  - Remove the top layer(s) (as it is related to the specific task)
  - Add new layers based on the need of the application
  - Train the model with existing layer weights frozen
  - Unfreeze all layers and train the whole model
- Benefits from existing models which are trained on a large amount of data for a different purpose (where data is more limited)

[Demo: fine-tuning existing models]

CNN Application: Classification and Localisation

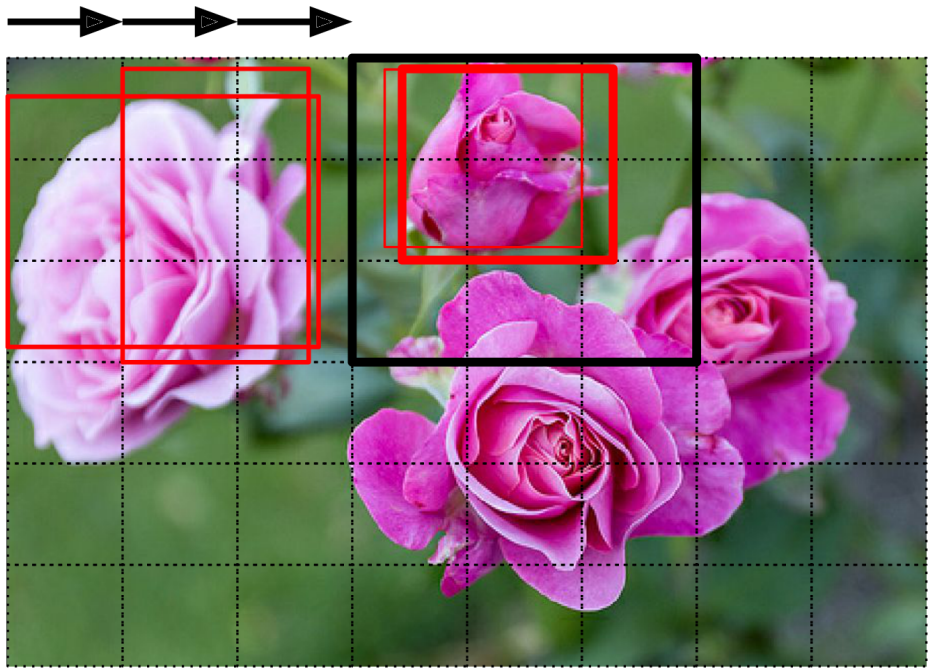
- In addition to object classification, also needs to identify the location
- Location: bounding box (e.g. centre position x, y; width and height)
- Losses:
  - Cross entropy for classification
  - MSE (Mean Squared Error) for bounding box
- Evaluation:
  - Intersection over Union (IoU)





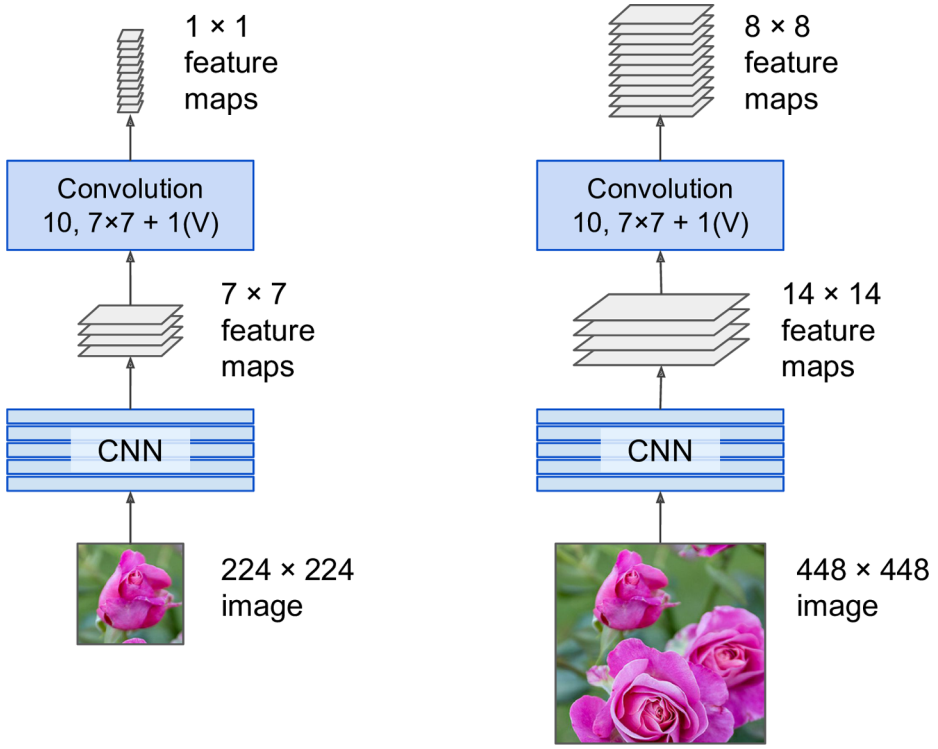
CNN Application: Object Detection

- Identify the location of each object
- Object Detection – Sliding Windows
  - Using sliding windows of different sizes
  - Similar to object classification and localisation, but look at local regions
  - Predict bounding box location (within the current window)
  - Also need to predict *objectness* (as an extra output)
  - Using *non-maximum suppression*: avoid reporting multiple times for the same object



Fully Convolutional Networks (FCNs)

- FCNs:
  - Apply to input images of arbitrary sizes
  - Achieving the effect of sliding windows through convolution
  - Basis of many applications, e.g. for object detection, semantic segmentation, image enhancement, etc.

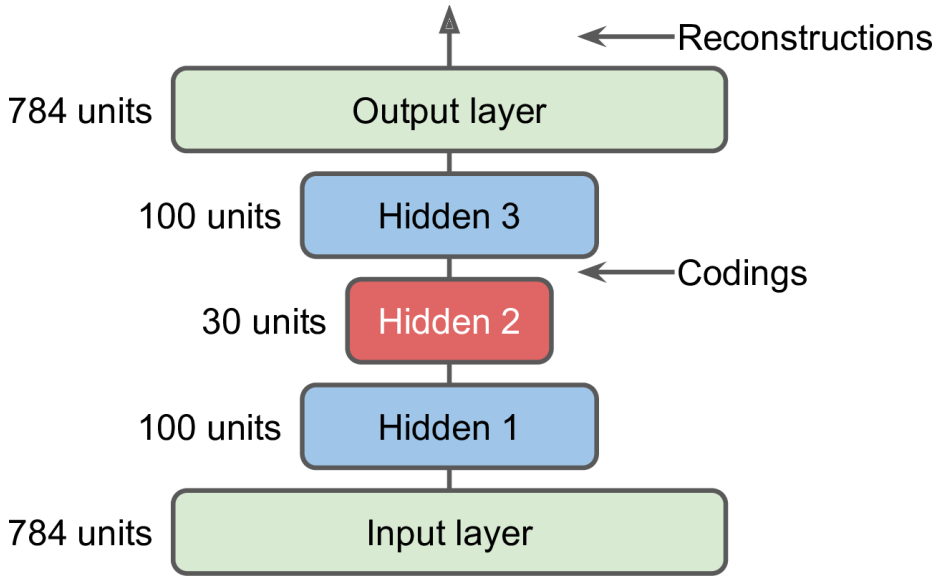


CNN Application: Object Detection

- Techniques for object detection methods:
  - YOLO (You Only Look Once)
    - Single-stage approach for object detection
    - Based on Fully Convolutional Network
    - Very efficient (real-time on a video)
  - Faster R-CNN
    - Two-stage approach, so slower than YOLO
    - First identify potential object regions, using a *Region Proposal Network*
    - Then classify objects based on the content in each bounding box

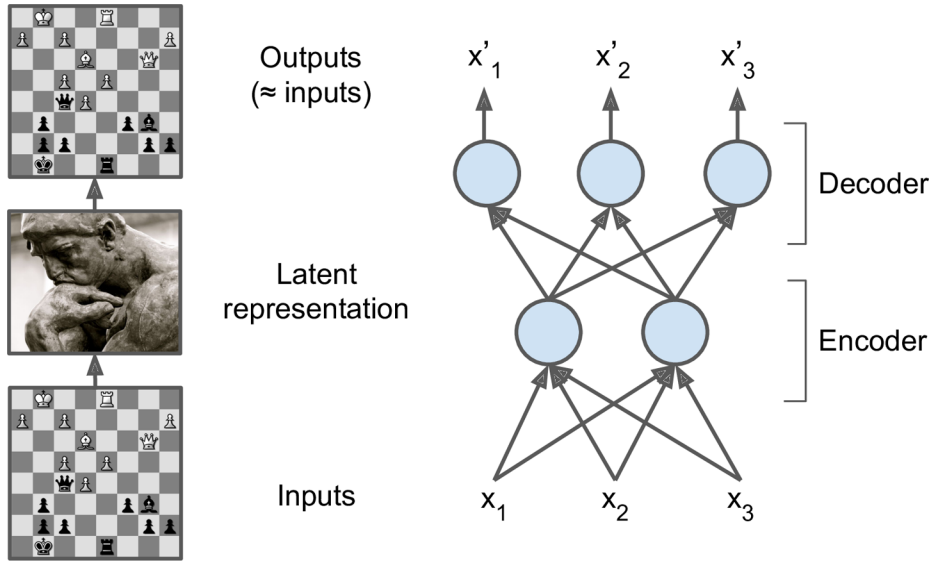
Autoencoders

- Labelled training data is usually expensive to obtain
- Autoencoders
  - A neural network architecture without any supervision
  - Key idea: the target is to make the output identical to (*reconstruct*) the input
  - Involves an encoder and a decoder
  - Produces a coding or latent representation for the input
- Avoiding trivial solutions (that just copy the input)
  - Bottleneck layer (limit the size of the latent representation)
  - Adding noise
- Useful for
  - Dimensionality reduction
  - Unsupervised pre-training
  - Generative models



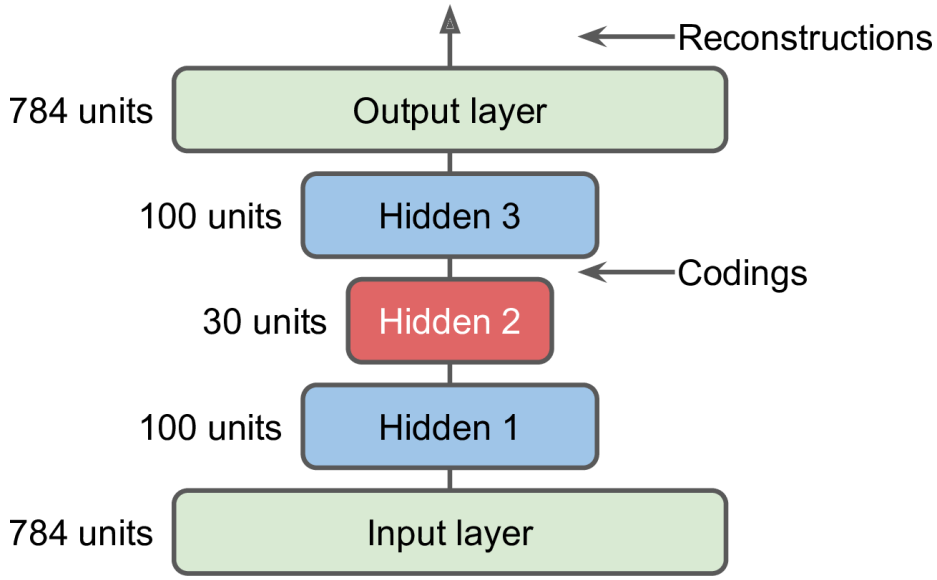
Representation Learning

- Which of the following number sequences do you find easier to memorise?
  - 40, 27, 25, 36, 81, 57, 10, 73, 19, 68
  - 50, 48, 46, 44, 42, 40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14
- Encoder: recognition network that maps input to latent representation
- Decoder: generative network that converts the latent representation the output



Typical Autoencoder Architecture

- Deep autoencoders
  - Typically symmetric for encoder and decoder
  - Encoder has layers with gradually reduced neurons
  - Decoder has layers with gradually increased neurons
  - The output layer matches the input layer

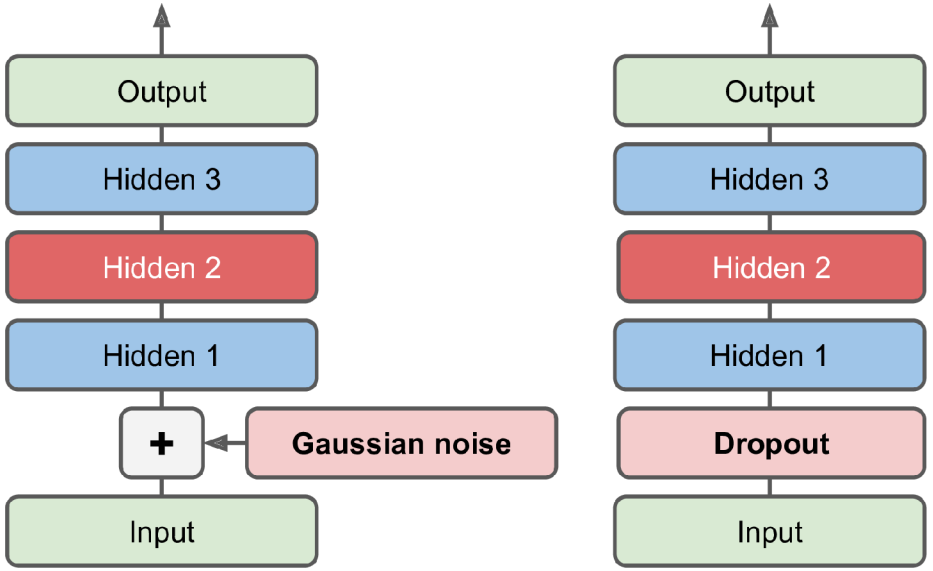


Experiment and Visualisation

[Demo: autoencoder]

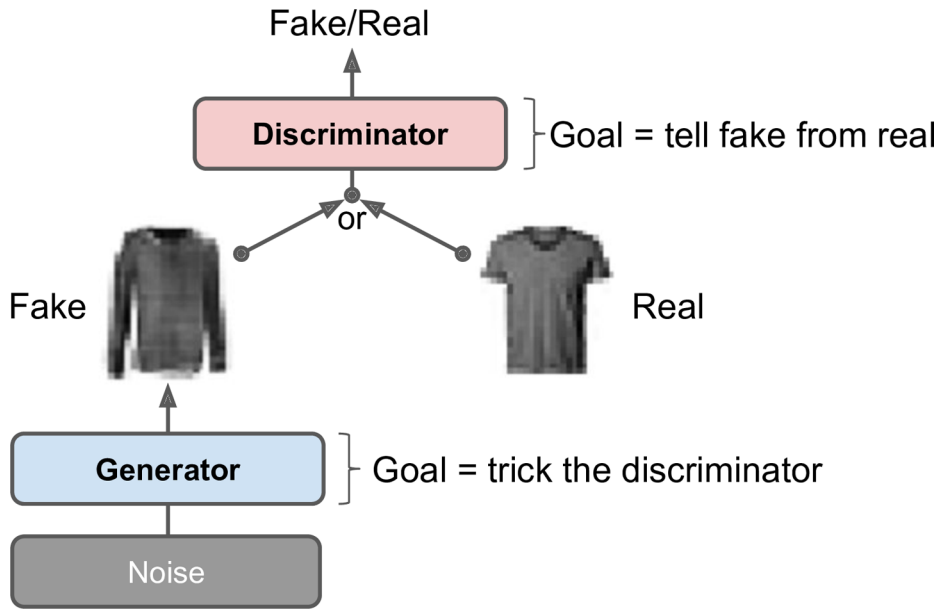
Denoising Autoencoders

- Denoising autoencoders
  - Making reconstructions non-trivial by
    - Adding Gaussian noise (using `GaussianNoise` layer)
    - Or using Dropout



Deep Generative Models (key ideas)

- Variational antoencoders
  - Latent representation is a Gaussian distribution
  - Has generation capability (by sampling latent distribution)
- Generative Adversarial Networks (GANs)



- Real or generated?

- Time for Practice!