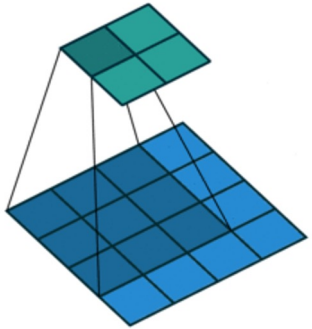
Tutorial 3

Neuro 140/240

Convolutional Neural Networks (CNNs)

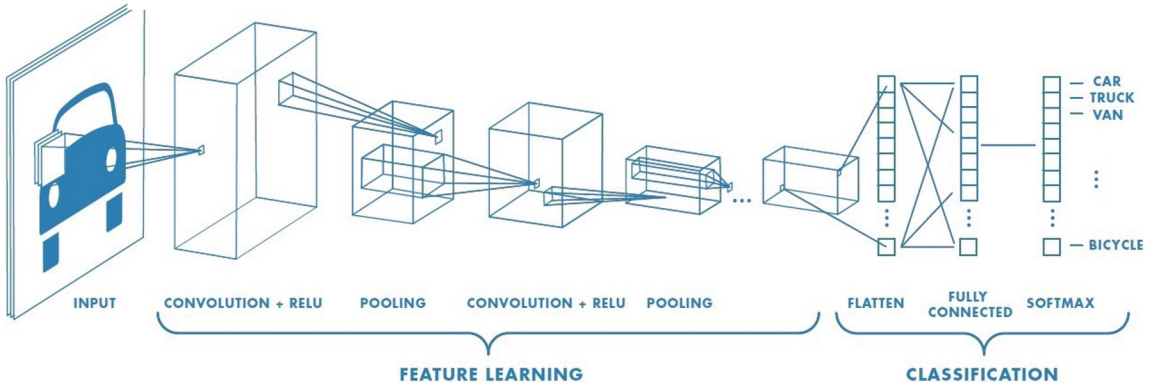
With fully connected NNs, we had to unravel all the image pixel values into one long vector. But images have 2D structure - let's exploit that!



Visualization of a convolution operation

Legend:
Blue grid = input image
Dark blue shadow = "kernel"
Green grid = activation values of second NN layer

Example kernels. Each kernel is a feature detector!



One convolution produces a new "image" that encodes *features* instead of *pixels*. We can apply another convolution to this!

After a number of convolutions, we can "flatten" the resulting tensor into one long vector and feed it into a fully connected NN. (see "classification" part of diagram)

CNN

COLAB TUTORIAL

Transfer Learning

Should we train a CNN from scratch?

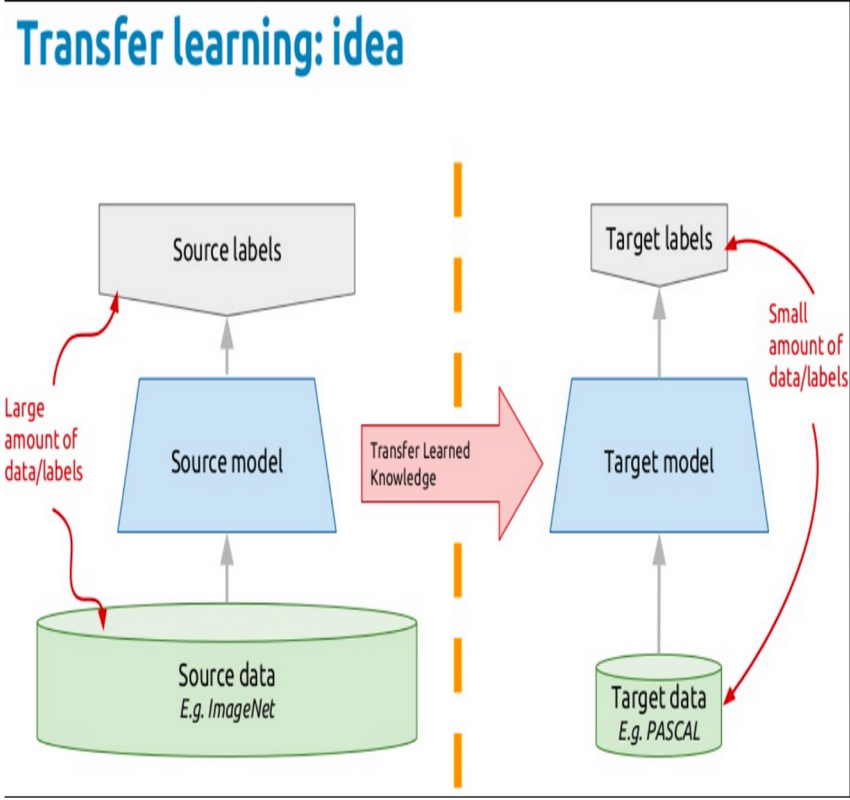
You can use the bulk of an already trained powerful CNN and modify some parts of the architecture
to fit to your specific task

How?

1. Freeze weights of network
2. Remove last fully connected layer
3. Add new fully connected layer with random weights
4. Train on new layer

MAIN POINT:

Instead of random initialization of weights, you can initialize network with known weights and fine tune all or a portion of them through training



Transfer Learning

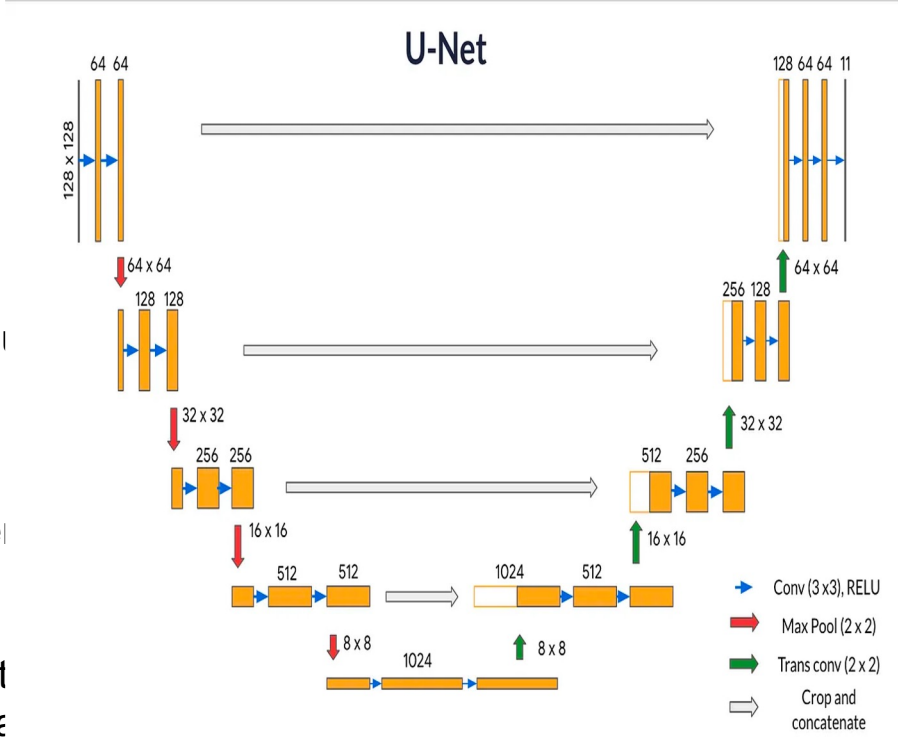
COLAB TUTORIAL

U-Net

Encoding: Contracting Path
Decoding: Expanding Path

What happens in the contracting path?
Captures context and reduces spatial resol
through convolutional and pooling layers
What happens in the expanding path?
Recovers spatial information
through convolutional and upsampling laye

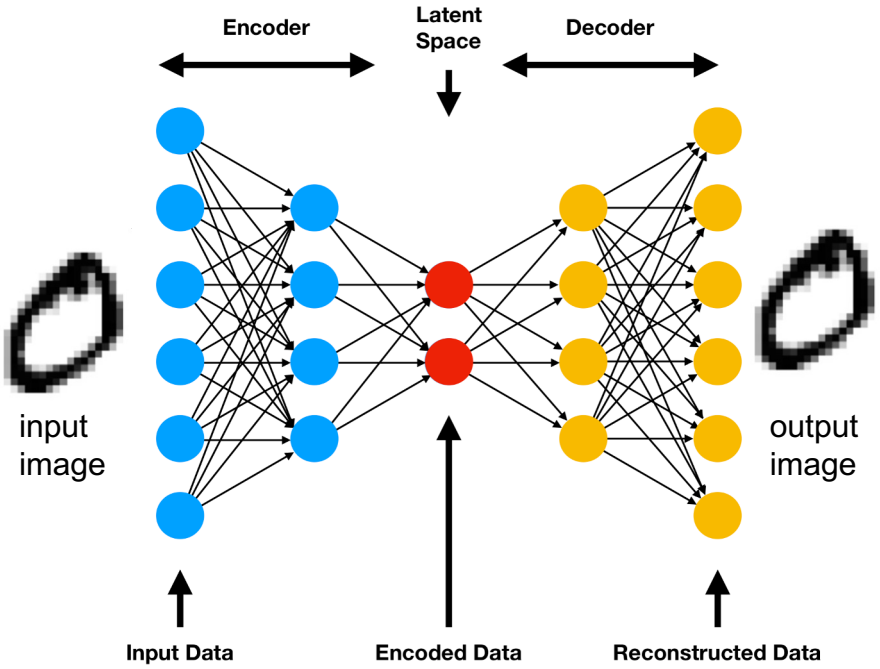
Preserving high-resolution features?
Skip connections! A concatenation operat
which allows network to retain information a
extracted features at each depth



U-Net

KAGGLE TUTORIAL

Generative models: autoencoders



Input: A picture in our dataset

Output: The same picture, reconstructed (we use the original image as our “label”)

Loss:
mean squared error of pixel values
Intuition: the more different the images are from each other, the higher the loss

We have our inputs, outputs, network model, and loss. *Everything we need for gradient descent and backpropagation!*

Our model learns to reconstruct images using a low-dimensional vector (in the latent space)

We can generate a new image by feeding a randomly-generated low-dim vector to the “decoder”

Autoencoder

GITHUB TUTORIAL