Deep learning, CNNs

Anders Eklund

anders.eklund@liu.se

Department of Biomedical Engineering (IMT)
Department of Computer and Information Science (IDA)
Center for Medical Image Science and Visualization (CMIV)
Linköping University, Sweden

April 3, 2022

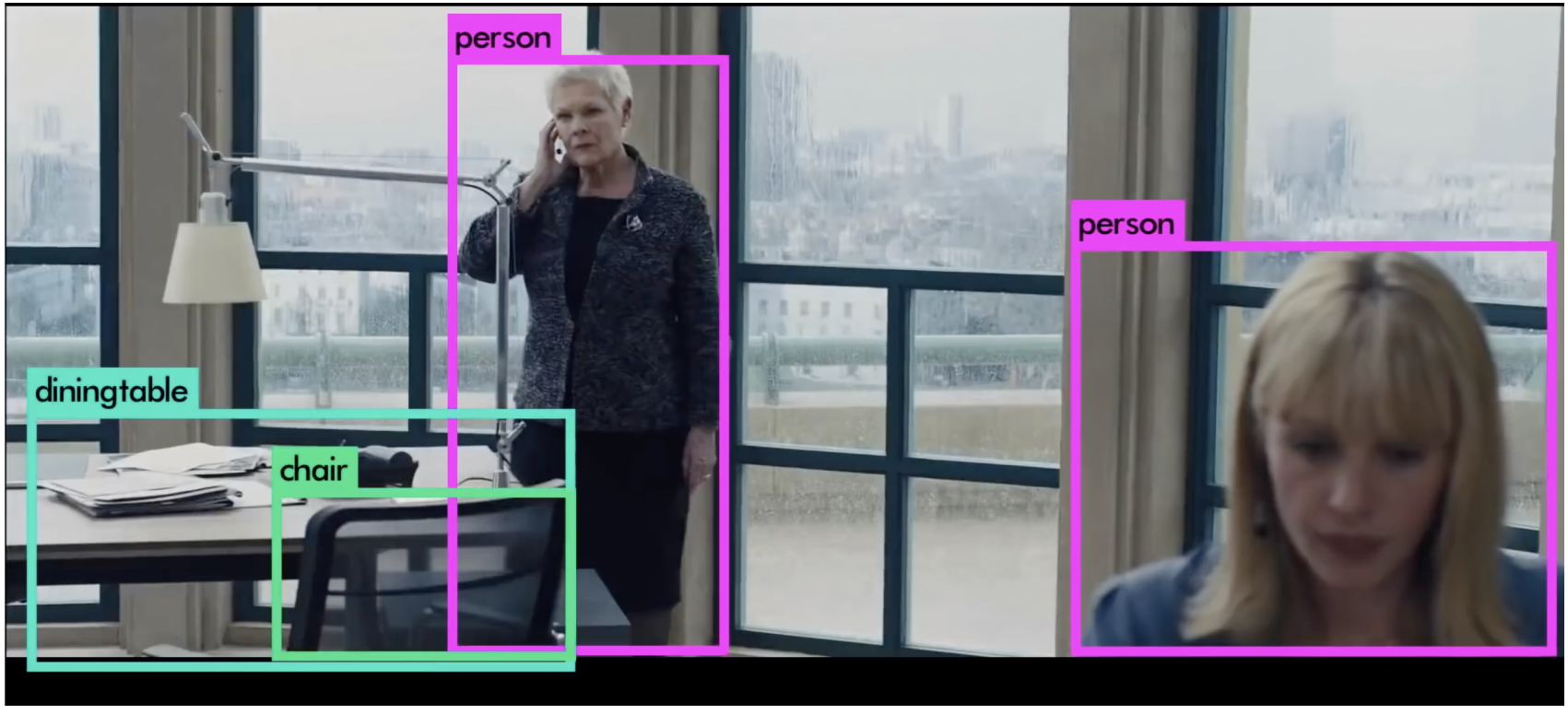
Outline

- Convolutional neural networks (CNNs)
  - Convolutional layers, pooling layers, ...
- Image classification
  - AlexNet, GoogLeNet, ResNet, ...
- Image segmentation
  - U-Net
- Understanding CNNs

Image classification / detection

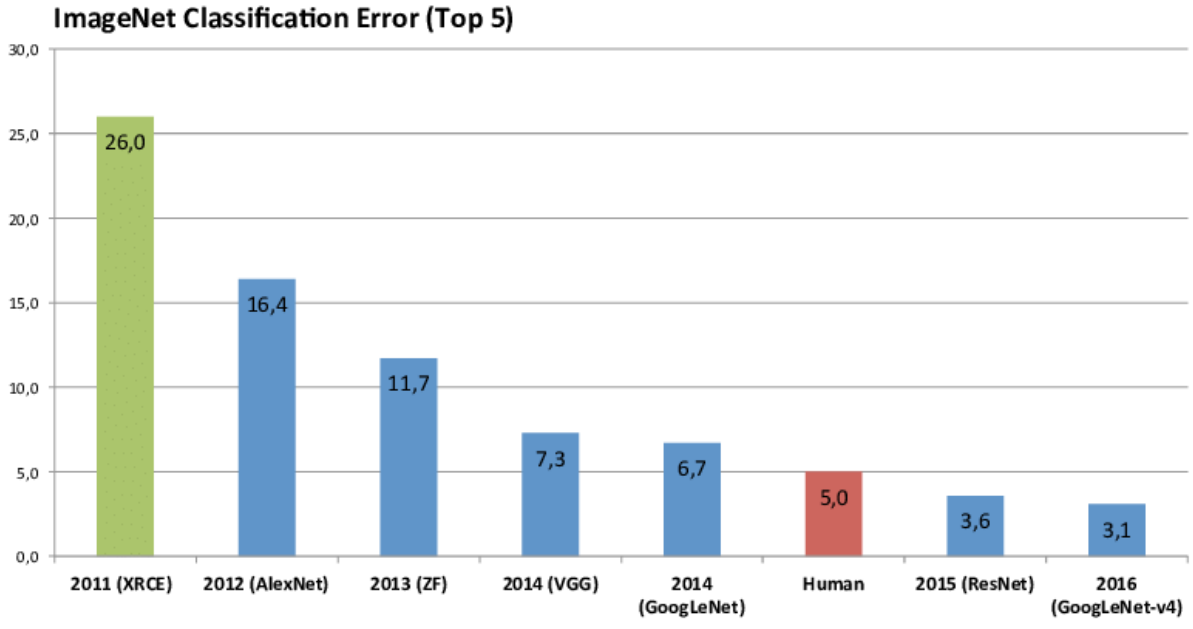
- In computer vision
  - What kind of object is present in an image? Many classes
  - Where is the object in an image? Many objects in one image (object detection)
- In medical imaging
  - Is the person sick or healthy? Binary
  - What is the disease? Many classes
  - What type of cancer is present? Several classes

Object detection - computer vision



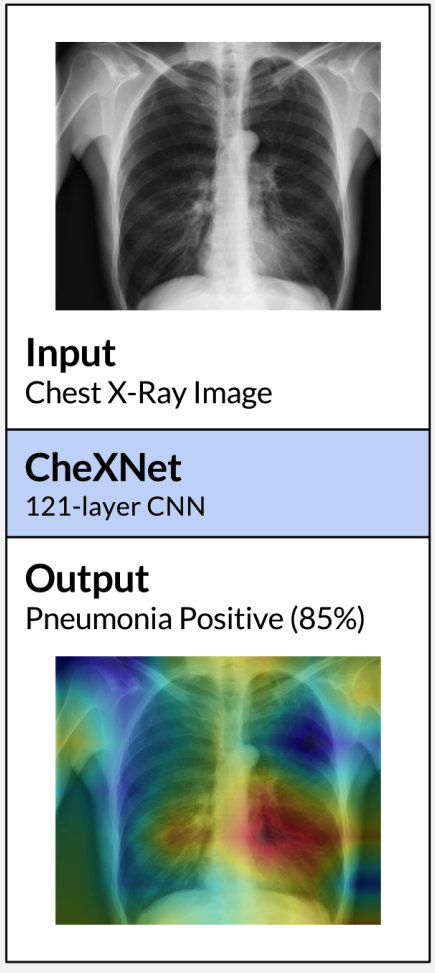
https://www.youtube.com/watch?v=VOC3huqHrss

Image classification - 2D example, CV

- ImageNet dataset / challenge, 1.2 million training images, 1000 classes
- http://www.image-net.org
- 

- https://devopedia.org/imagenet
- Cats, dogs, cars, airplanes, ...

IMAGE CLASSIFICATION - 2D EXAMPLE, MI



Our model, CheXNet, is a 121-layer convolutional neural network that inputs a chest X-ray image and outputs the probability of pneumonia along with a heatmap localizing the areas of the image most indicative of pneumonia.

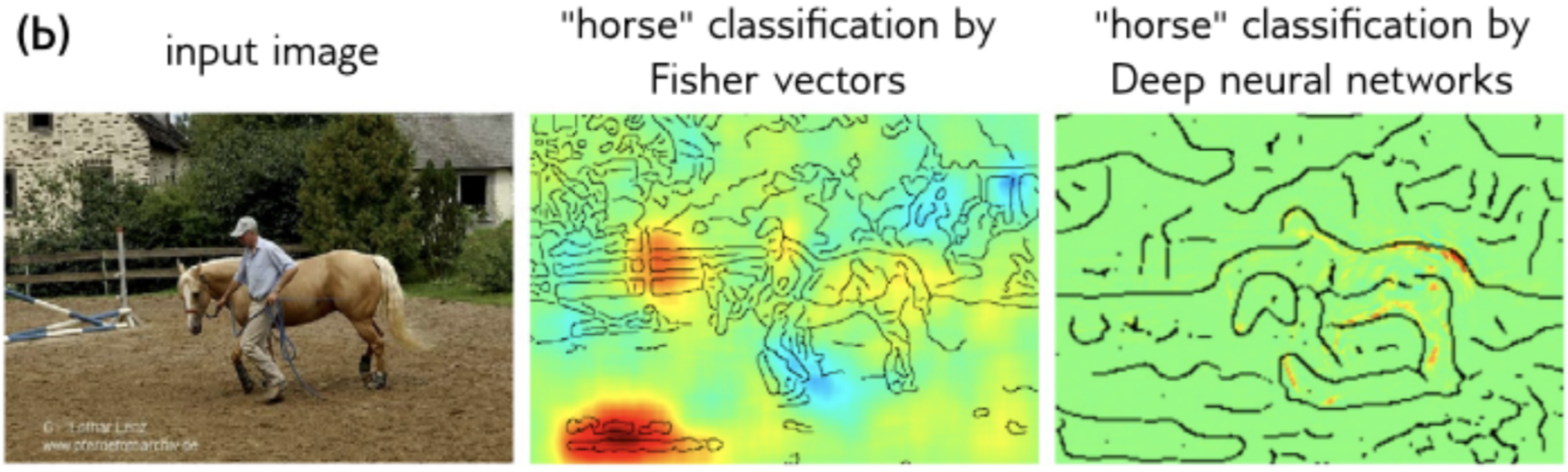
We train CheXNet on the recently released ChestX-ray14 dataset, which contains 112,120 frontal-view chest X-ray images individually labeled with up to 14 different thoracic diseases, including pneumonia. We use dense connections and batch normalization to make the optimization of such a deep network tractable.

Rajpurkar et al. (2017). Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. arXiv preprint arXiv:1711.05225.

Image regression - 2D example, CV

- Object detection (where is an object in an image?)
  can for example be framed as a regression problem
- Train a CNN to predict (x,y) coordinates of 2 corners
  of a bounding box (top left corner, bottom right corner)
- Of course requires lots of training data with this information...

Be careful with your training data



Based on Lapuschkin et al. (2016) "Analyzing classifiers: Fisher vectors and deep neural nets"

Fig. 21. Examples taken from the literature of model validation via explanation. (a) Explanation of the concept "`sci.space`" by two text classifiers. (b) Unexpected use of copyright tags by the Fisher vector model for predicting the class "horse".

Montavon, G., Samek, W., & Müller, K. R. (2018). Methods for interpreting and understanding deep neural networks. Digital Signal Processing, 73, 1-15.

IMAGE CLASSIFICATION - 3D EXAMPLE, MI

NeuroImage
Volume 145, Part B, 15 January 2017, Pages 137-165

Single subject prediction of brain disorders in neuroimaging: Promises and pitfalls

Mohammad R. Arbabshirani [a, b], Sergey Plis [a], Jing Sui [a, c], Vince D. Calhoun [a, d]

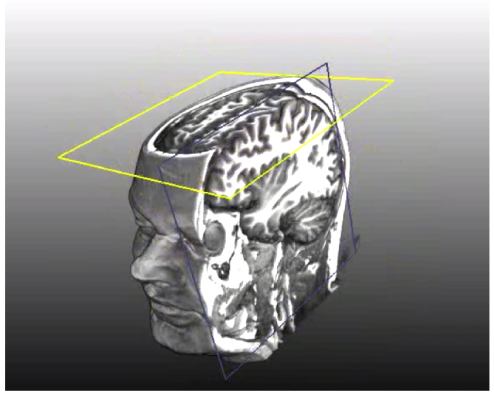
Show more

https://doi.org/10.1016/j.neuroimage.2016.02.079

Get rights and content

Highlights

- Past efforts on classification of brain disorders are comprehensively reviewed.
- The common pitfalls from machine learning point of view are discussed.
- Emerging trends related to single-subject prediction are reviewed and discussed.



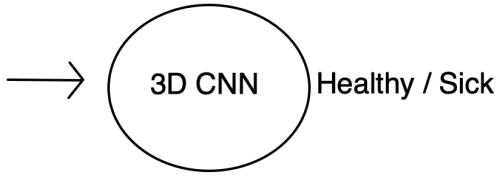


IMAGE REGRESSION - 3D EXAMPLE, MI

ELSEVIER

NeuroImage

Volume 163, December 2017, Pages 115-124

Predicting brain age with deep learning from raw imaging data results in a reliable and heritable biomarker

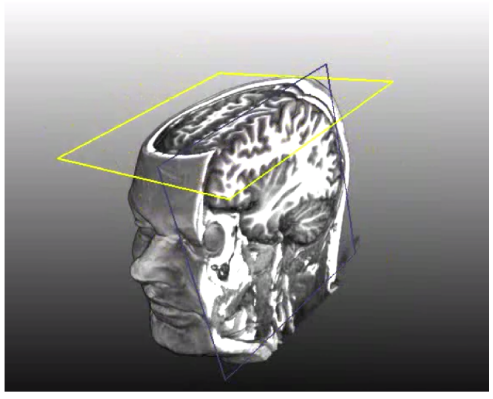
James H. Cole [a], Rudra P.K. Poudel [b], Dimosthenis Tsagkrasoulis [c], Matthan W.A. Caan [d], Claire Steves [e], Tim D. Spector [e], Giovanni Montana [b, c]

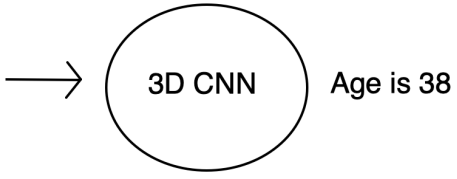
Show more

https://doi.org/10.1016/j.neuroimage.2017.07.059

Highlights

- Chronological age can be accurately predicted using convolutional neural networks.
- Age predicted is accurate even using raw structural neuroimaging data.
- Brain-predicted age can be generated in a clinically applicable timeframe.
- Brain-predicted age is significantly heritable.
- Brain-predicted age is highly reliable, both within and between scanners.





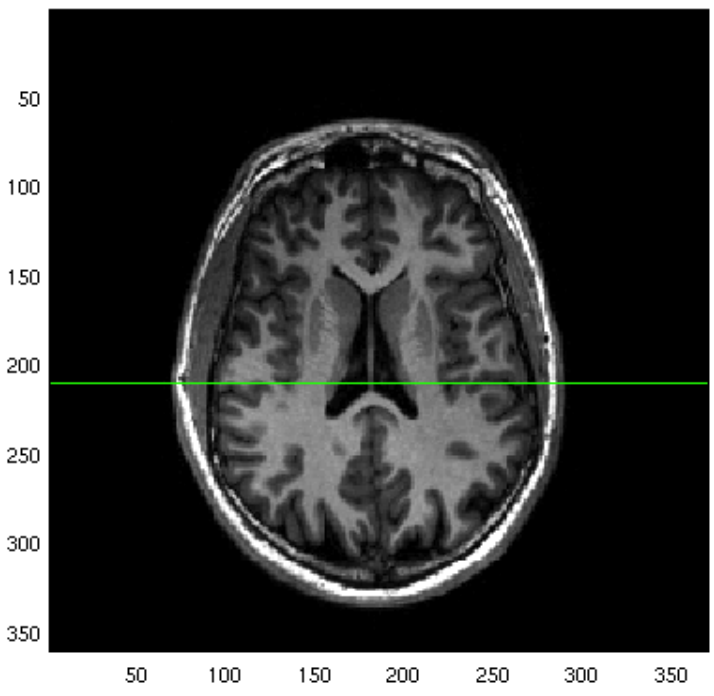
COMPUTER VISION VS MEDICAL IMAGING

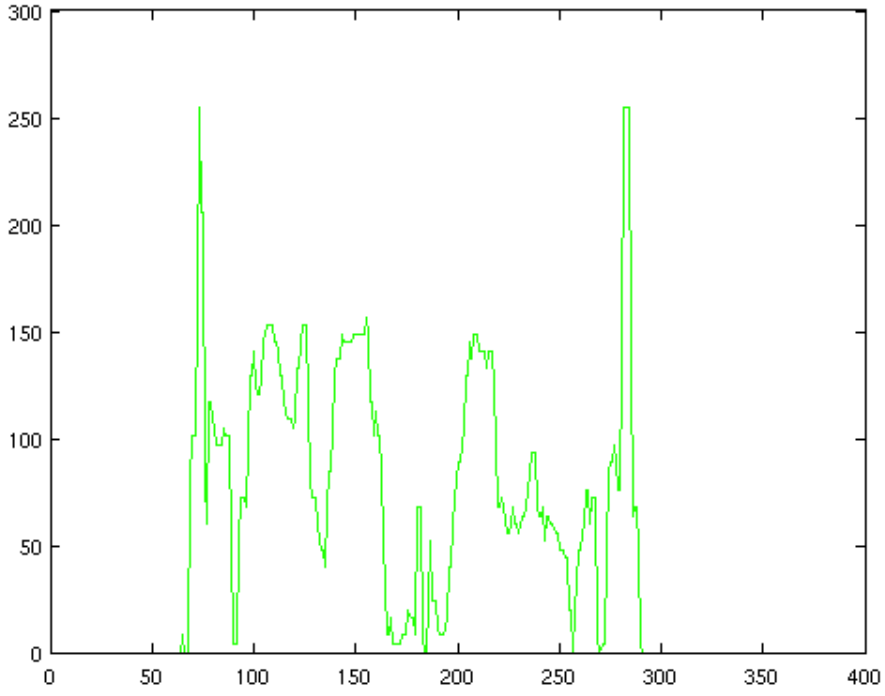
- Computer vision
  - Big datasets
  - Easy to share data
  - 2D / 3D data
  - ImageNet (millions of images and labels)
  - Many pretrained 2D networks
- Medical imaging
  - Small datasets (few subjects, 50 - 1000)
    (hospitals of course contain large datasets)
  - Not so easy to share data (ethics, GDPR)
  - 3D / 4D data
  - No VolumeNet
  - Few pretrained 3D networks

What is an image?

- An image can be seen as a 2D signal, where each pixel represents some property (e.g. light intensity)
- Most common case, images from a digital camera
- Each pixel can be accessed by a 2D coordinate (x,y)
- Each pixel can store more than one value,
  colour images store red, green and blue values (3 channels)
- Gray scale images are most common in medical imaging (single channel)
- Spatial autocorrelation function,
  neighbouring pixels often have similar values

2D EXAMPLE - EACH LINE / COLUMN IS A 1D SIGNAL





3D Data

- 2D images cannot represent parts of the human body, like the heart or the brain
- The image concept can be extended to 3 dimensions
- A volume can be seen as a 3D signal, where each voxel has some property
- Pixel = picture element, voxel = volume element
- MRI head volume: 256 x 256 x 150
- Video is also 3D (2D + time)

4D Data

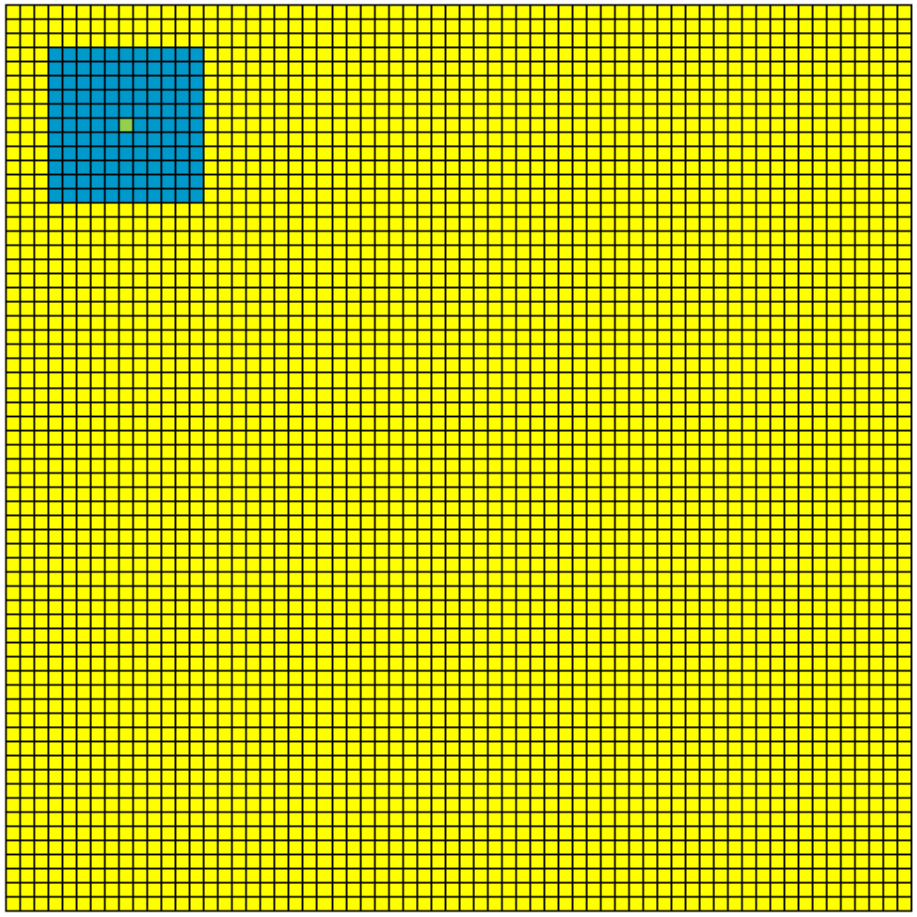
- Several medical imaging modalities produce 4D data (ultrasound, magnetic resonance imaging, computed tomography)
- 3D + time, e.g. a beating heart
- 4D CT dataset, 512 x 512 x 512 x 20
- Functional MRI (brain activity), 64 x 64 x 32 x 300
- There are not (yet) any Conv4D functions in Keras / Tensorflow
- Deep learning for 4D data would also be very memory demanding...

Convolution - 2D

- Convolution = scalar product between filter coefficients and signal values in each neighbourhood
- Slide the filter over all pixels, save each result in the filter center
- The filter first needs to be flipped along x and y, otherwise we will instead calculate the correlation
- The pixel value at coordinate (x,y) is given by $image[x][y]$
- The filter is N x N pixels, where N is often odd

$$(image * f)[x, y] = \sum_{f_{x=-(N-1)/2}}^{(N-1)/2} \sum_{f_{y=-(N-1)/2}}^{(N-1)/2} image[x - f_x][y - f_y] \cdot f[f_x][f_y]$$

Convolution - 2D Example



- Filter in blue (11 x 11)
- Filter response in green
- Filter response is sum of filter coefficients multiplied with pixel values (local scalar product)
- Repeat for all pixels
- Special treatment of border pixels

Convolution - 2D Example

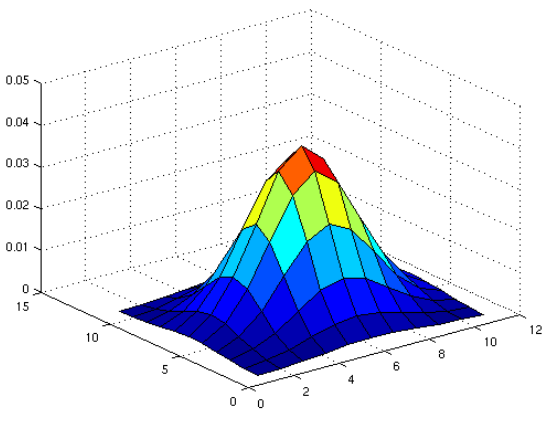
$$Image = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, Filter = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

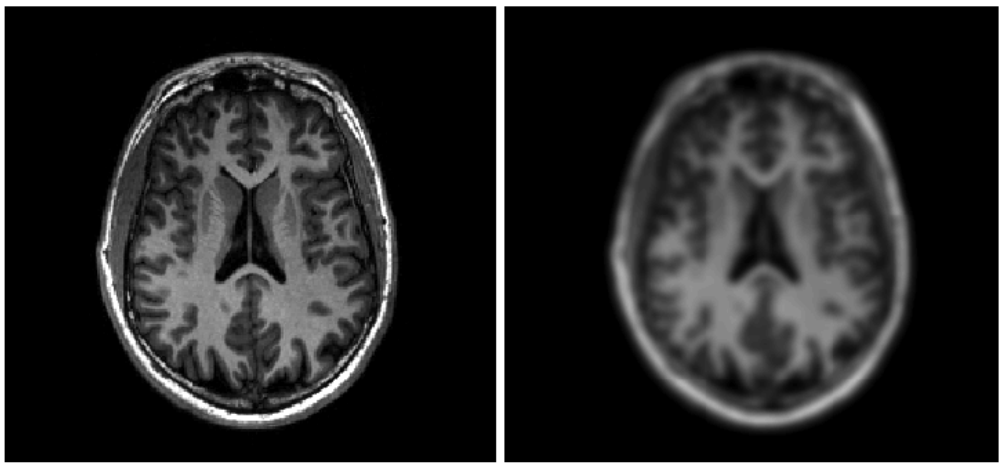
$$Filter' = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

- Filter response =
  $1 \cdot 1 + 2 \cdot 2 + 1 \cdot 3 + 0 \cdot 4 + 0 \cdot 5 + 0 \cdot 6 + -1 \cdot 7 + -2 \cdot 8 + -1 \cdot 9 = -24$

Common filters in image processing

- Gaussian lowpass filter, 2D / 3D Gaussian





Common filters in image processing

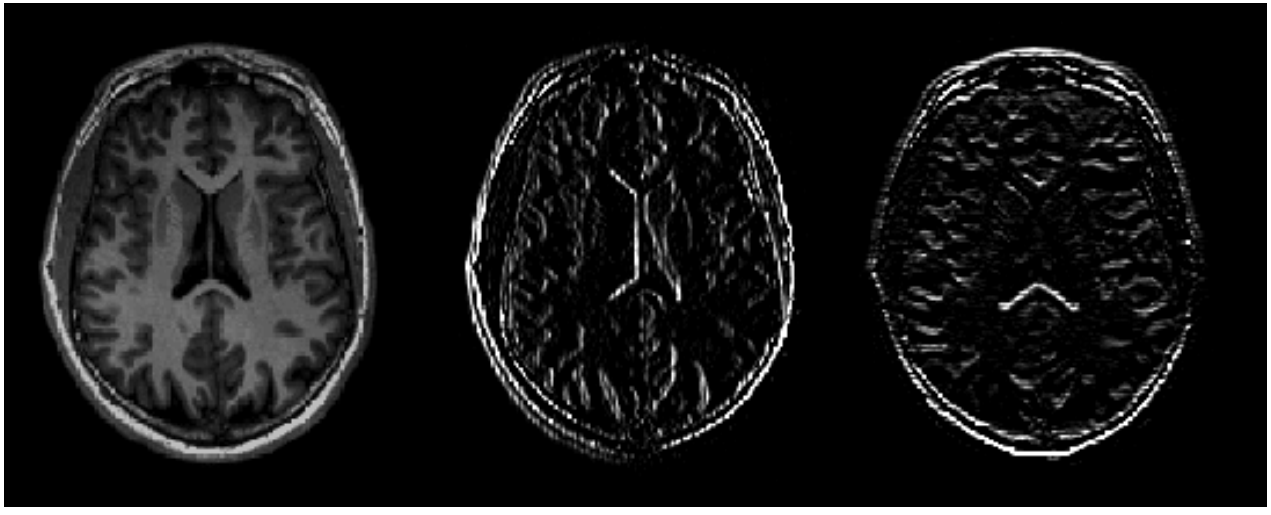
- Sobel filters, edge detection

-1	0	+1
-2	0	+2
-1	0	+1

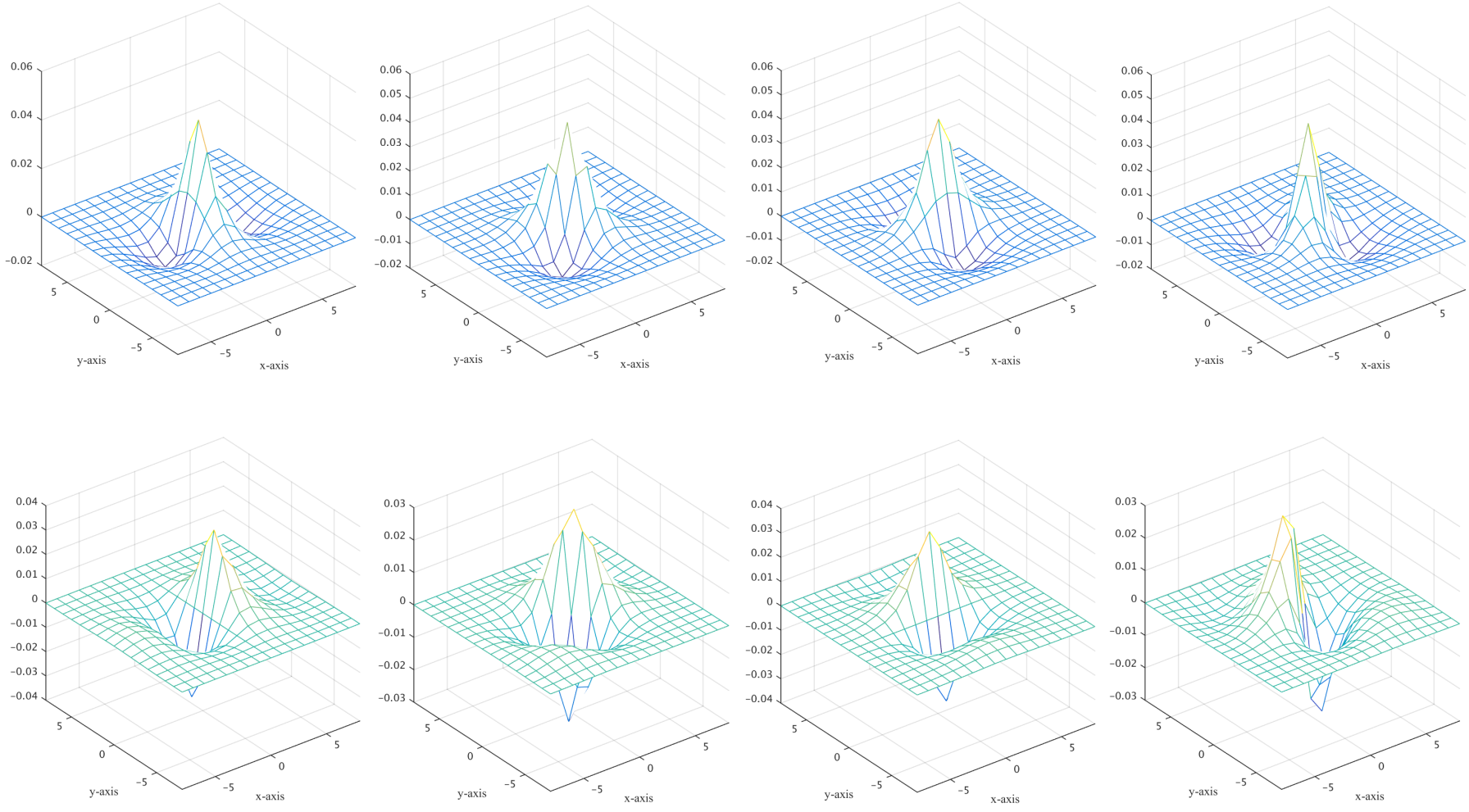
x filter

+1	+2	+1
0	0	0
-1	-2	-1

y filter

- 

Real even (top) and imaginary odd (bottom) optimized quadrature filters in spatial domain - different directions



Filter optimization

- Filters can be optimized using weighted least squares, (combining the error in the spatial domain and in the frequency domain)

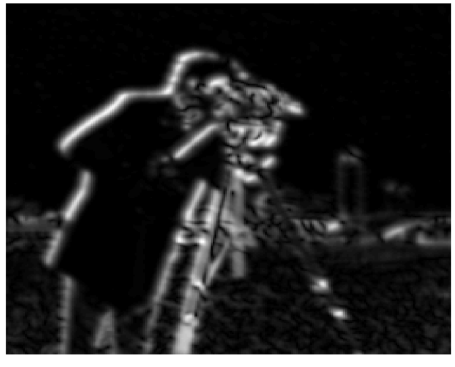
- $\epsilon^2 = \|F_w(F_i - B\,f)\|^2 + \alpha\|f_w(f_i - f)\|^2$

- You need to define the following properties

  - How many quadrature filters should be used
  - Filter directions
  - Size in spatial domain, size in frequency domain
  - Center frequency (size of structures)
  - Bandwidth
  - Importance of error in spatial domain and frequency domain (α)

Quadrature filter responses, magnitude - different directions

Filter 1



Filter 2

Filter 3

Filter 4

Why deep learning?

- Before deep learning,
  most researchers worked on hand made features
- Instead of feeding a raw image to the network,
  extract features such as edges, lines, corners
- CNNs became very popular in 2012,
  when a 5-layer CNN outperformed other methods on ImageNet
- Krizhevsky, A., Sutskever, I., Hinton, G., ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012
- 106 000 citations!

Why convolution?

- Why use convolution instead of a standard deep neural network?
  - To lower the number of parameters to train
  - To use the local spatial correlation in the data
  - To reduce processing time
- A one megapixel image (1000 x 1000) has 1 000 000 "dimensions"
- For a single neural network layer with 100 nodes,
  need to train 100 000 000 parameters / weights for standard network
- 100 filters of size 3 x 3,
  1000 parameters / weights per layer for CNN (900 + 100 bias)

Finding the best filters

- In classic image processing,
  the filters are designed manually, filter optimization
  (e.g. to detect lines and edges of a certain size/frequency)

- In deep learning,
  the filters are learned through back propagation

- Benefit of deep learning,
  can learn very advanced filters, specific for your data

- Drawback of deep learning,
  need to perform convolution with many filters during many iterations
  (time consuming, memory demanding)

CNN ARCHITECTURE - CLASSIFICATION

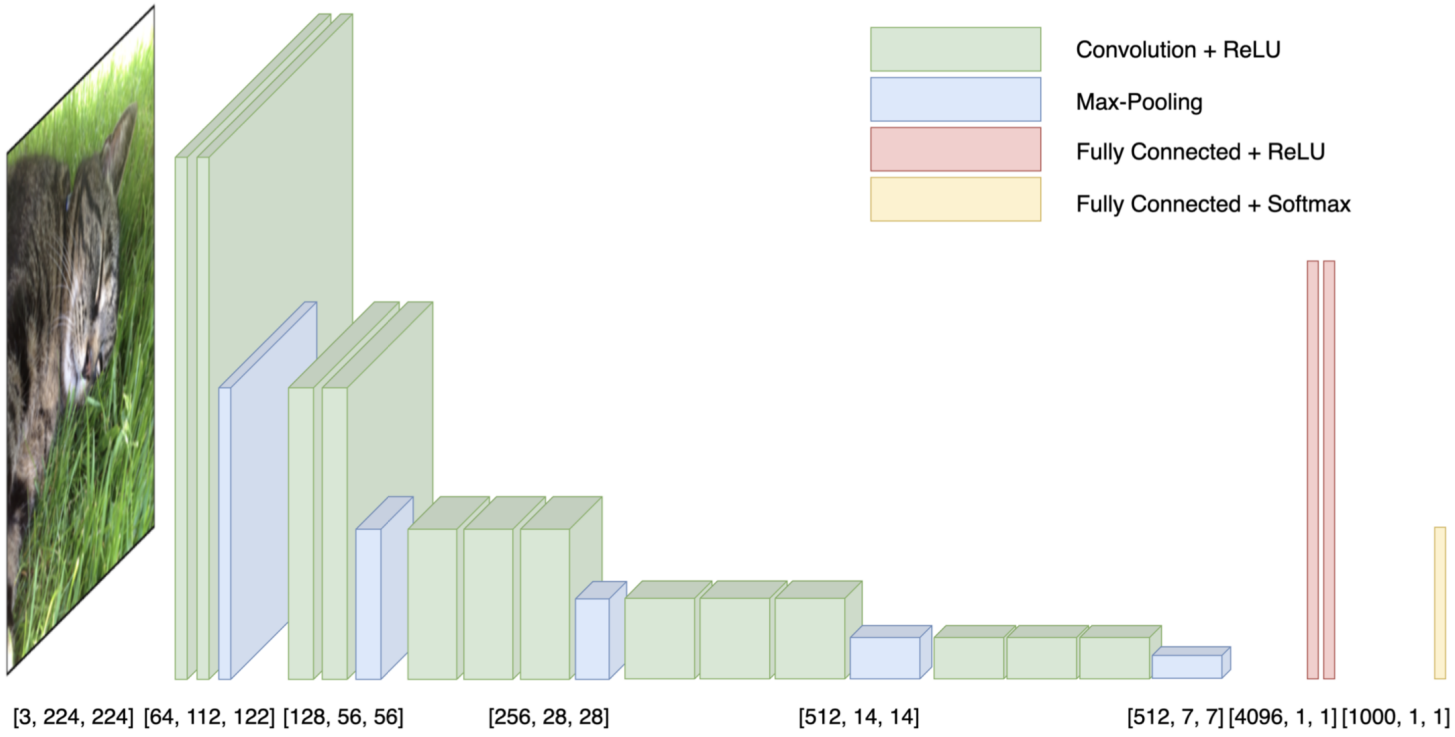


Figure 2.4: *Illustration of the data flow through the network VGG16. Data size, of the format* $[c, h, w]$*, is shown for the input image, output of each max-pooling layer, output after the first two fully connected layers, and the final network output.*

Jesper Westell, Multi-Task Learning using Road Surface Condition Classification and Road Scene Semantic Segmentation, LIU-IMT-TFK-A–19/570–SE

Convolutional layer

- Every 2D convolutional layer takes a 4D tensor
  and returns another 4D tensor
- The 4 dimensions are batch_size, height, width, channels
  (or batch_size, channels, height, width)
- You set the number of filters to be used in every Conv2D layer
  (gives the number of channels in output tensor), filter size, padding etc
- A 1D convolutional layer works in the same way, but with 3D tensors
- A 3D convolutional layer works in the same way, but with 5D tensors

Conv2D is 3D, sort of

- Conv2D performs 2D convolution over the image, but each filter also learns how to linearly combine all channels in the input tensor
- This is not a true 3D convolution, but sort of
- Filter size 5 x 5 and 3 input channels, each learned filter is 5 x 5 x 3
- Filter size 3 x 3 and 128 input channels, each learned filter is 3 x 3 x 128
- Filter 1: 0.25 * channel 1 + 0.8 * channel 2 ...
- Filter 2: -0.4 * channel 1 + 0.2 * channel 2 ...

Convolutional layer - Keras

```
from keras.models import Sequential, Model
from keras.layers import Input, Conv2D

model = Sequential()

# Layer 1
model.add(Conv2D(filters=32,kernel_size=(5,5),
                 padding = 'same',
                 activation='relu',
                 input_shape=input_shape))
# Layer 2
model.add(Conv2D(filters=64,kernel_size=(5,5),
                 padding = 'same',
                 activation='relu')
```

See https://keras.io/layers/convolutional/ for details

Convolution as matrix vector operation

- Convolution is a linear operation,
  all linear operations can be written as (sparse) matrix multiplications!
  (see Nx x Ny image / matrix as (Nx · Ny) x 1 vector)

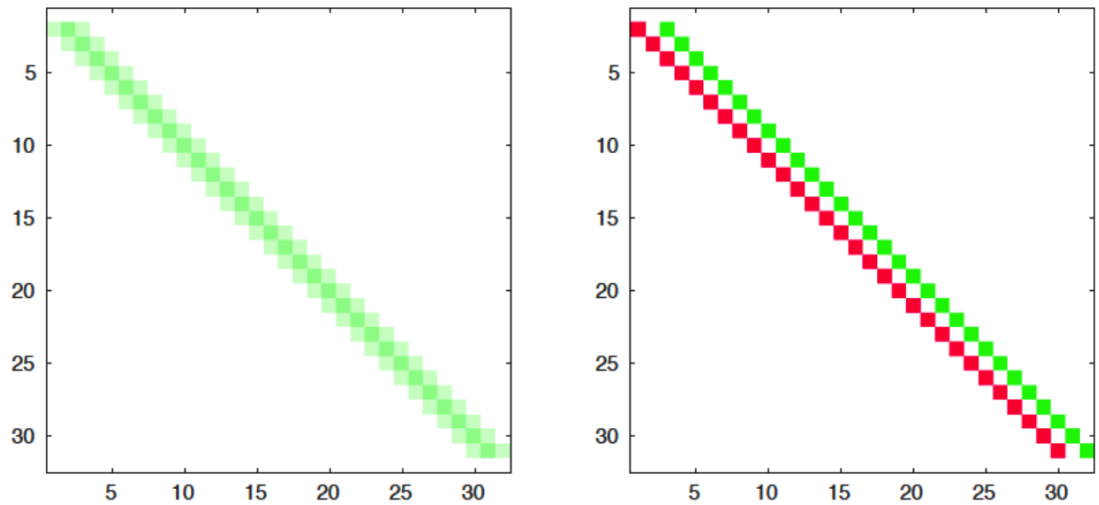
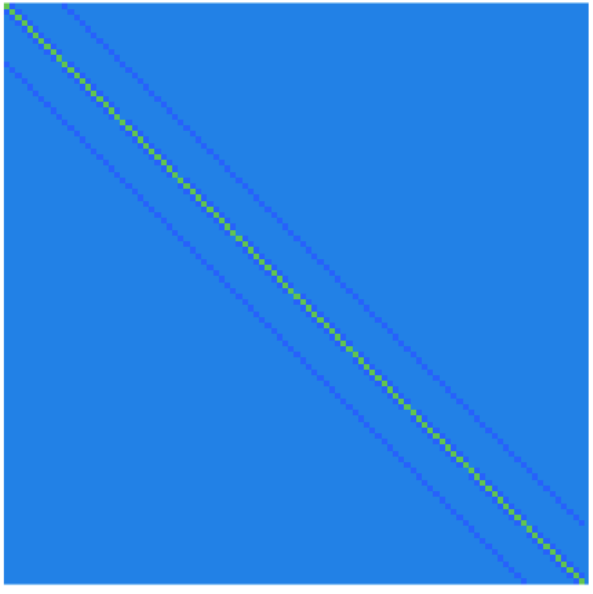


Figure 2.1: Left: $\frac{1}{4}[1, 2, 1]$ filter matrix. Right: $[1, 0, -1]$ filter matrix.

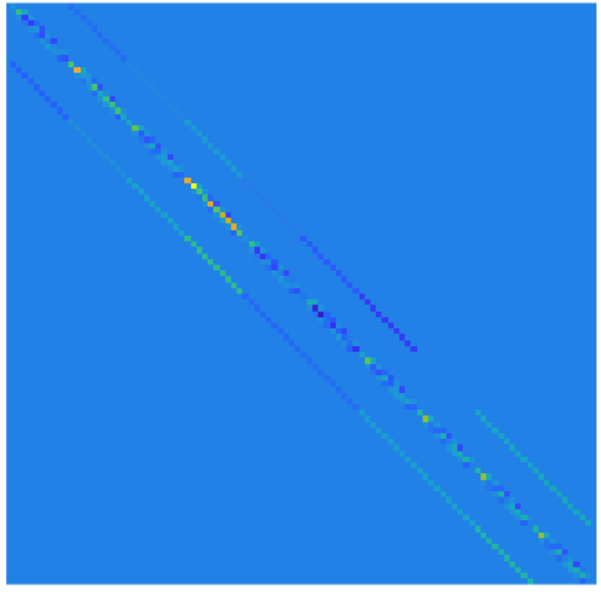
- G. Johansson, A Global Linear Optimization Framework
  for Adaptive Filtering and Image Registration, 2015

- Sparse matrices and parameter sharing (same filter everywhere)

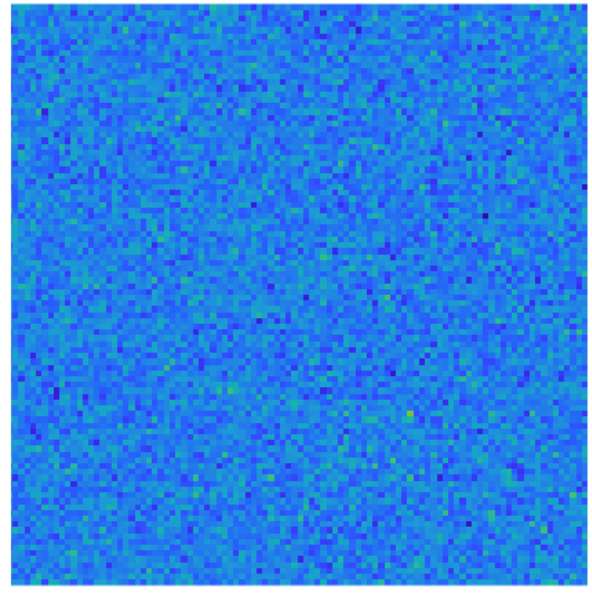
CONVOLUTION AS MATRIX VECTOR OPERATION



Convolutional network



Unshared convolution / Locally connected layers



Fully connected network

Convolution as matrix vector operation

- Every image in batch is represented as a long vector
- Put all images in batch into a matrix X
  (one image per column)
- Put filter into sparse matrix W
- Apply filter W in a layer to all images using $WX + B$
- Operations performed by CUDNN / CUSPARSE, details are secret
- We can re-use all methods / algorithms for neural networks!
  (since convolution is a linear operation)

Handling the border of the image

- Values outside each image are undefined
- Solution 1, only calculate valid filter response,
  filter response will be (Nf - 1) pixels smaller (Nf size of filter)
- Solution 2, assume values are 0 (zero padding),
  filter response will be same size as image
- Why is this important?
- For a filter of size 7 x 7, the image will be 6 x 6 smaller
  in every layer of the network for solution 1
  (and eventually we run out of pixels)

Convolutional layer - padding

- Input 4D tensor of size (batch_size, height, width, channels)
- 64 filters of size 5 x 5
- 'same' padding,
  - output 4D tensor is of size (batch_size, height, width, 64)
- 'valid' padding,
  - output 4D tensor is of size (batch_size, height - 4, width - 4, 64)

Same convolution - zero padding

$$Image = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, Filter = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

$$Filter' = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

- Filter response = ?

Convolutional layer - stride

- The stride determines how the filter moves, usually 1 pixel / voxel at a time
- Stride = 2, jump 2 pixels / voxels at a time
- Stride > 1 will reduce the size of the output, stride = 2 will reduce the number of pixels a factor 2 in each dimension (approximately)

Pooling layers

- Pooling is normally done in every (second) layer of a CNN
- Most common, max or average operation over a small neighbourhood (pixels/voxels)
- Without pooling, the data would have the same size in every layer (and the final fully connected layer would be very large)
- Can do pooling over filter responses in each pixel/voxel
- Pooling can be used to become invariant to
  - Translation / shift
  - Scale ?
  - Rotation (Figure 9.9)

POOLING - ROTATION INVARIANCE

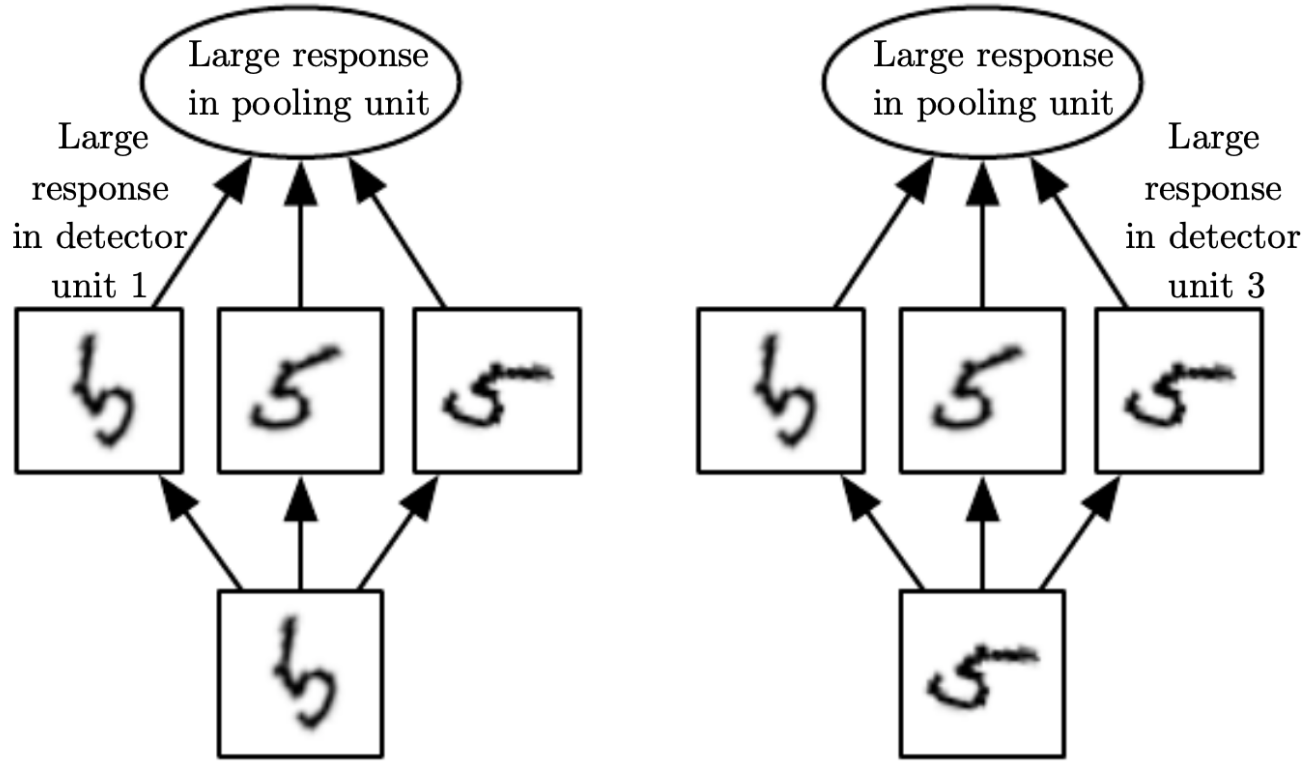


Figure 9.9: Example of learned invariances. A pooling unit that pools over multiple features that are learned with separate parameters can learn to be invariant to transformations of the input. Here we show how a set of three learned filters and a max pooling unit can learn to become invariant to rotation. All three filters are intended to detect a hand written 5.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

Pooling layer - Keras

```
from keras.models import Sequential, Model
from keras.layers import Input, Conv2D, MaxPooling2D

model = Sequential()

model.add(Conv2D(filters=32,kernel_size=(5,5),
                 padding = 'same',
                 activation='relu',
                 input_shape=input_shape))

model.add(MaxPooling2D(pool_size=(2, 2)))
```

See https://keras.io/layers/pooling/ for details

POOLING - REDUCE NUMBER OF PARAMETERS

	Activation shape	Activation size	# parameters
Input	(32,32,3)	3072	0
Conv2D (8 filters, size 5)	(28,28,8)	6272	608
MaxPooling2D (2,2)	(14,14,8)	1568	0
Conv2D (16 filters, size 5)	(10,10,16)	1600	3216
MaxPooling2D (2,2)	(5,5,16)	400	0
Fully Connected (120 nodes)	(120,1)	120	48001
Fully Connected (84 nodes)	(84,1)	84	10081
Softmax	(10,1)	10	841

First layer, 8 filters of size 5 x 5 x 3, + 8 biases = 608 parameters

Second layer, 16 filters of size 5 x 5 x 8, + 16 biases = 3216 parameters

Fully connected layer 1, 400 * 120 + 1 = 48 001 parameters

Fully connected layer 2, 120 * 84 + 1 = 10 081 parameters

Fully connected layers

- After a number of layers with convolution and max-pooling, flatten tensor to a long vector, connect to fully connected layer(s) (standard dense neural network)
- Note: This will put a restriction on the image input size
- Fully convolutional networks can be trained on images of size A, and then be applied on images of size B
  (fully convolutional and fully connected are different things)
- No dense layers, all layers are convolutional
  (mainly used for image segmentation)
- 1 x 1 convolution, just combine all channels (features) into one or more scalars (same combination for all pixels)

Fully connected layer - Keras

```
from keras.models import Sequential, Model
from keras.layers import Input, Conv2D, MaxPooling2D
from keras.layers import Flatten, Dense

model = Sequential()
model.add(Conv2D(filters=32,kernel_size=(5,5),
                 padding = 'same',
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(numClasses, activation='softmax'))
```

See https://keras.io/layers/core/ for details about Flatten and Dense

Popular 2D cnn classification architectures

- LeNet-5 (1998), 7 layer CNN
- AlexNet (2012), 5 convolutional layers, 3 fully connected layers
- GoogLeNet (Inception V1) (2014), 22 layer CNN
- VGGNet (2014), 16 convolutional layers
- ResNet (2015), 152 convolutional layers, skip connections
- DenseNet (2017), dense connections

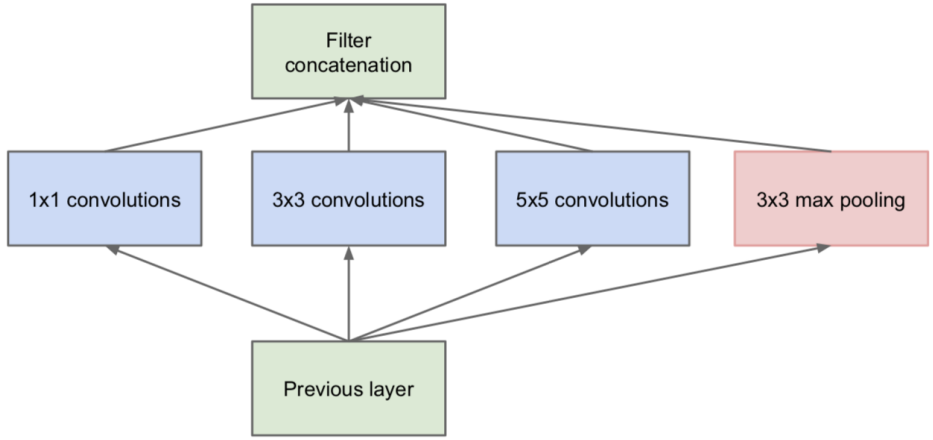
GoogLeNet (Inception V1)

- Promoted the idea of stacking the layers in CNNs more creatively, as networks in networks. Inside a relatively standard architecture (called the stem), GoogLeNet contains multiple inception modules, in which multiple different filter sizes are applied to the input and their results concatenated. This multi-scale processing allows the module to extract features at different levels of detail simultaneously.

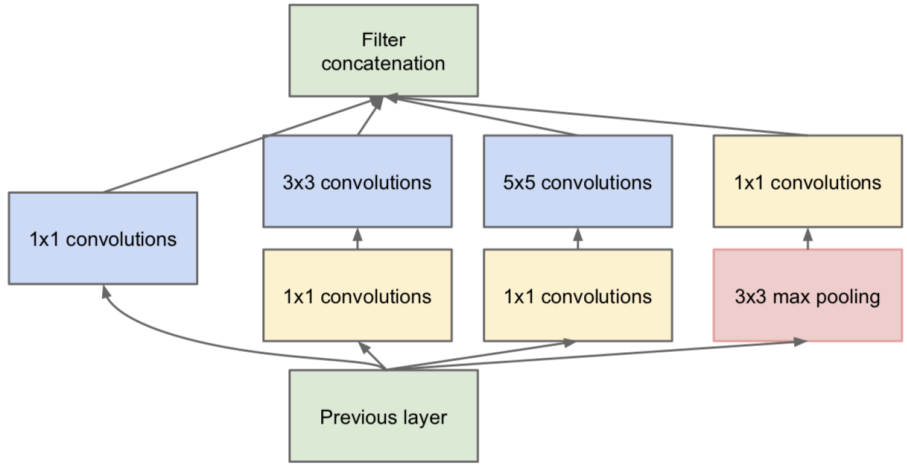
- GoogLeNet also popularized the idea of not using fully-connected layers at the end, but rather global average pooling, significantly reducing the number of model parameters.

- Szegedy et al., Going deeper with convolutions, IEEE conference on computer vision and pattern recognition, 2015

GoogLeNet (Inception V1)



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

Figure 2: Inception module

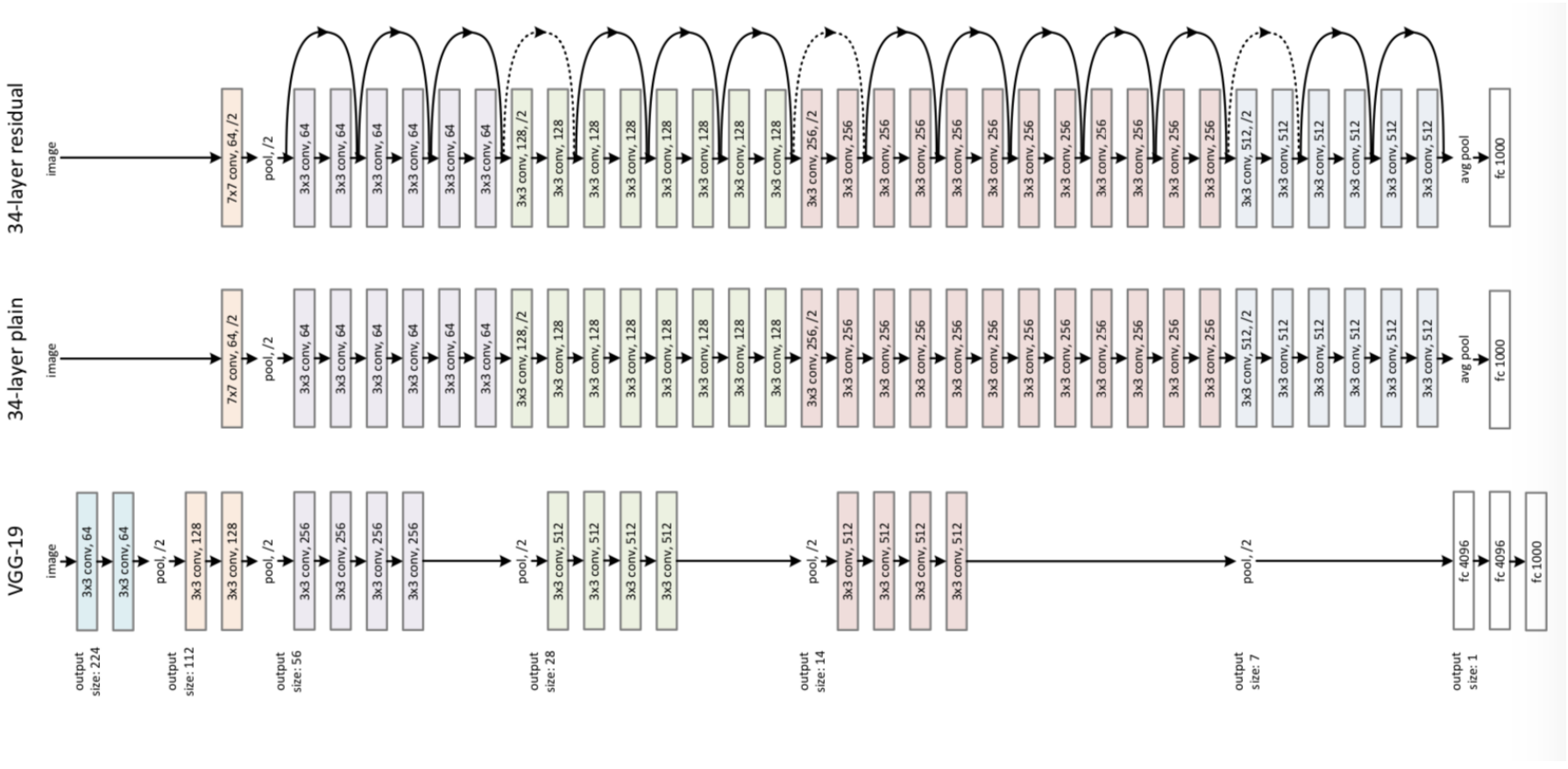
VGGNet

- Popularized the idea of using smaller filter kernels and therefore deeper networks (up to 19 layers for VGG19, compared to 7 for AlexNet)
- Trained the deeper networks using pre-training on shallower versions
- Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556

ResNet

- Noticed that performance on training data dropped when having too many layers
- Introduced skip connections, which makes it possible to train much deeper networks. Some features are best constructed in shallow networks, while others require more depth. The skip connections facilitate both at the same time, increasing the network's flexibility when fed input data.
- Skip connection -> easy to learn identity function (no convolution)
- He et al., Deep residual learning for image recognition, IEEE conference on computer vision and pattern recognition, 2016

RESNET - SKIP CONNECTIONS



Pretrained 2D architectures

- Several popular 2D architectures are available as pretrained networks (for example trained on ImageNet)
- Instead of starting training from scratch, start from pretrained network
- Most of these pretrained networks are trained for color images, not for grayscale images (most common in medical imaging)
- Not so many pretrained networks available for 3D

Pretrained 2D architectures - Keras

Applications

Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.

Weights are downloaded automatically when instantiating a model. They are stored at `~/.keras/models/`.

Available models

Models for image classification with weights trained on ImageNet:

- Xception
- VGG16
- VGG19
- ResNet, ResNetV2
- InceptionV3
- InceptionResNetV2
- MobileNet
- MobileNetV2
- DenseNet
- NASNet

See https://keras.io/applications/

Pretrained 2D architectures - Keras

```
from keras.models import Model
from keras.applications.vgg16 import VGG16

model = VGG16(weights='imagenet')
model.summary()
```

Model summary - vgg16

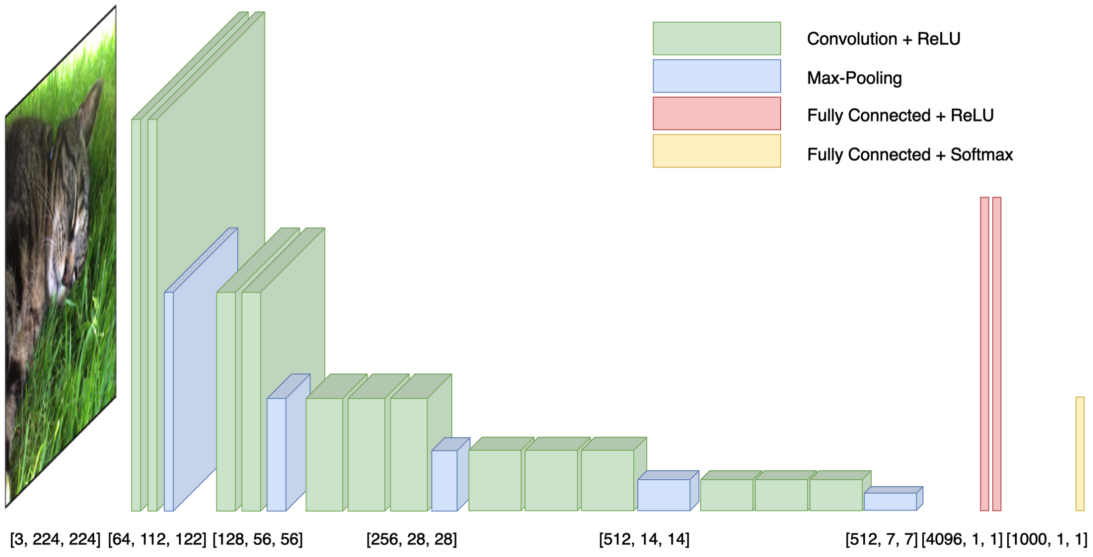


Figure 2.4: *Illustration of the data flow through the network VGG16. Data size, of the format* $[c, h, w]$*, is shown for the input image, output of each max-pooling layer, output after the first two fully connected layers, and the final network output.*

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```

Changing default classifier - Keras

```
from keras.models import Model
from keras.applications.vgg16 import VGG16

model = VGG16(weights='imagenet',
              include_top=False,
              input_shape=(224,224,3))
model.add(Flatten())
model.add(Dense(10))
```

Freezing layers in pretrained models

- Pretrained models are good for initialization
- Some parts of the network still need to be trained, to fit your specific data, to improve performance
- Early convolutional layers learn basic features (edges, lines, similar to how the brain works)
- Freeze early (top) layers, train deep (bottom) layers

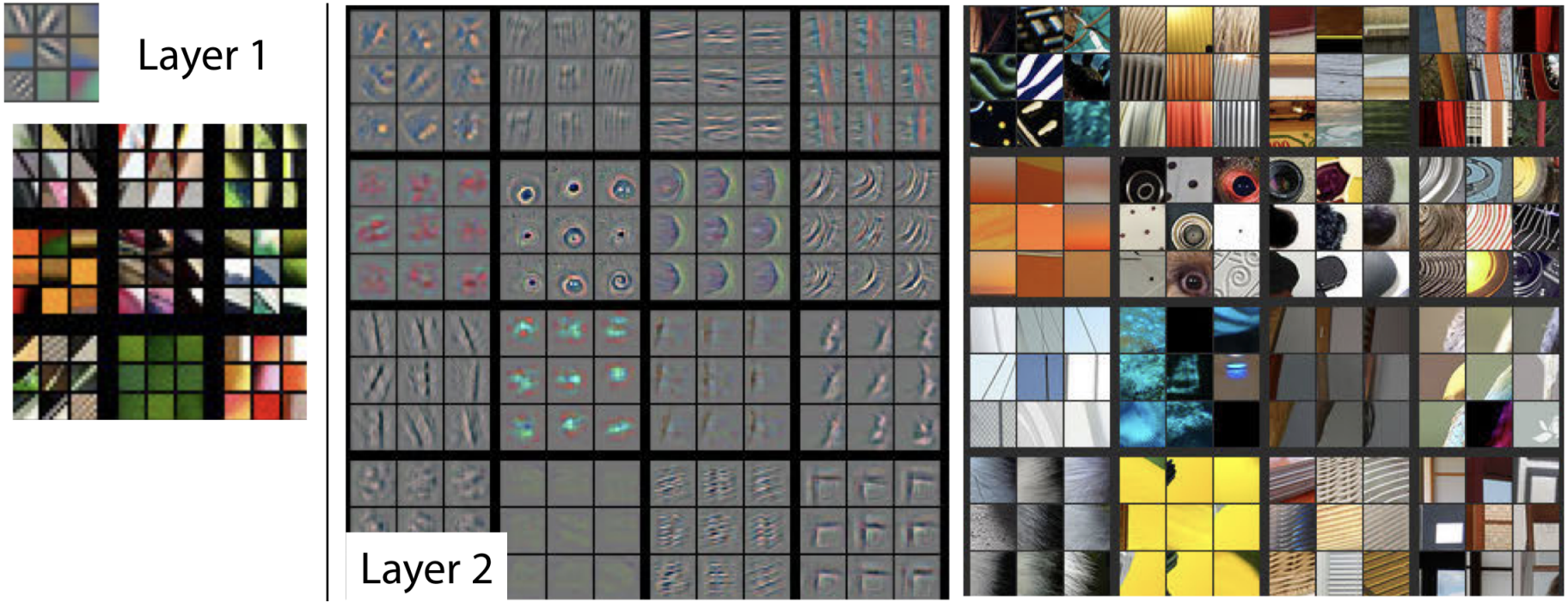
Freezing layers - Keras

```
from keras.models import Model
from keras.applications.vgg16 import VGG16

model = VGG16(weights='imagenet',
              include_top=False,
              input_shape=(224,224,3))

# Freeze the first 10 layers
for layer in VGG16.layers[:10]:
    layer.trainable = False
```

What about 3D CNNs? Volume data

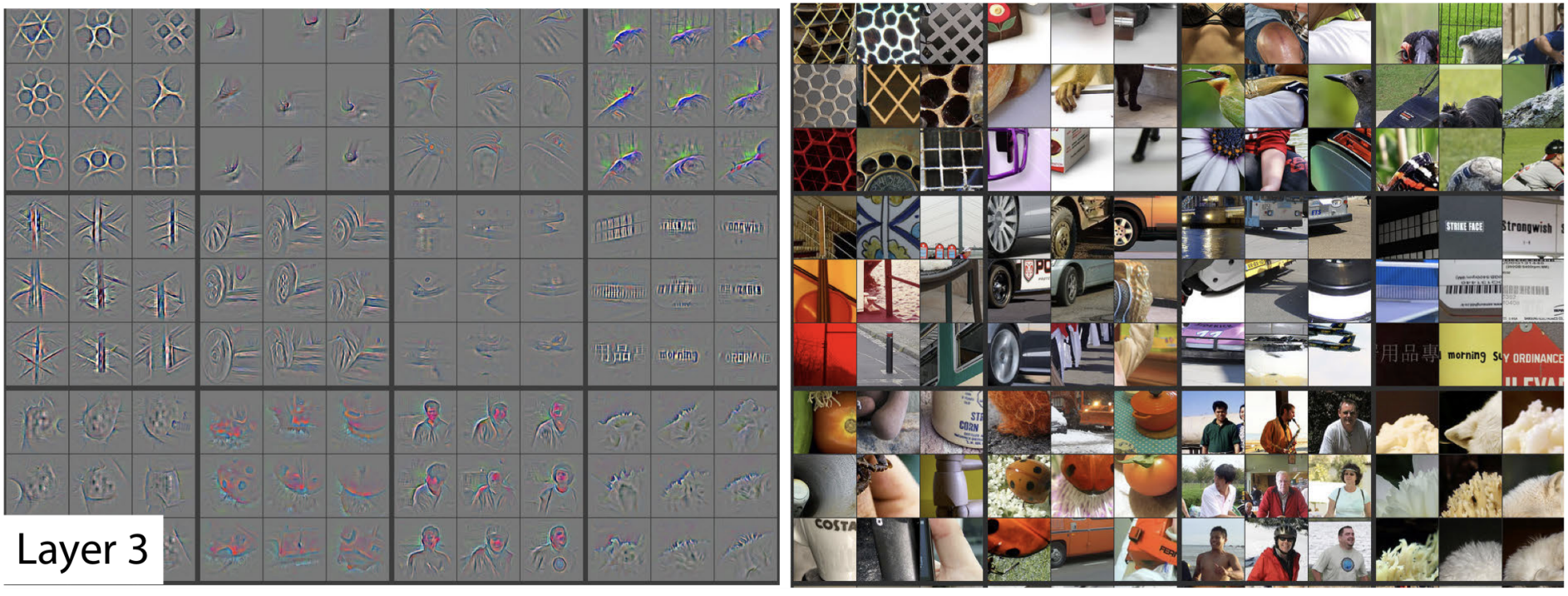
- Easy to change Keras code from Conv2D to Conv3D
- MaxPooling2D -> MaxPooling3D
- Will increase graphics memory usage,
  cannot use as many filters per layer
- Will increase processing time
- Keras has built-in 2D data augmentation, but not for 3D

UNDERSTANDING CNNS



Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.

Understanding CNNs



Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.

Understanding CNNs

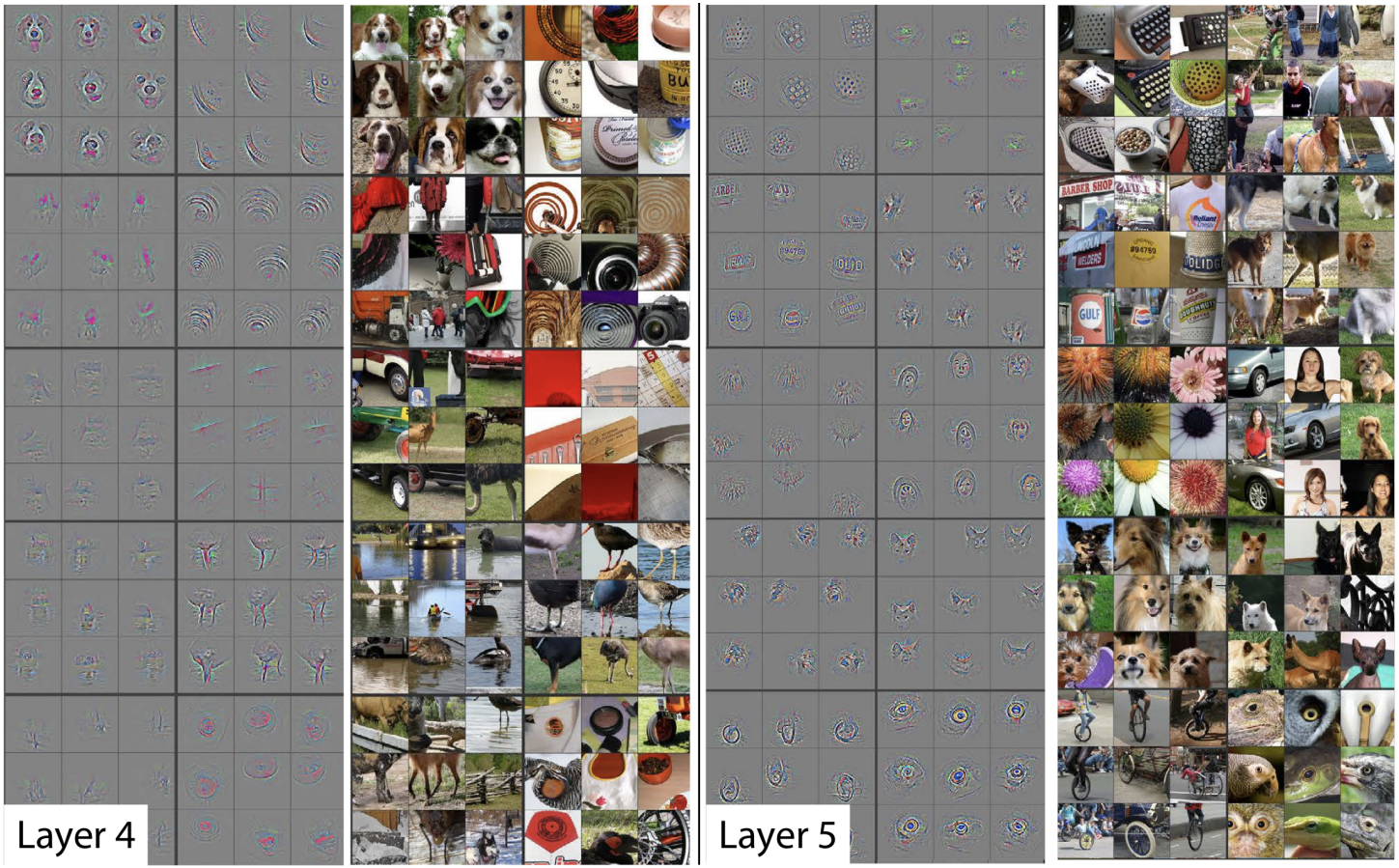


Figure 2. Visualization of features in a fully trained model. For layers 2-5 we show the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our deconvolutional network approach. Our reconstructions are *not* samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1). Best viewed in electronic form.

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.

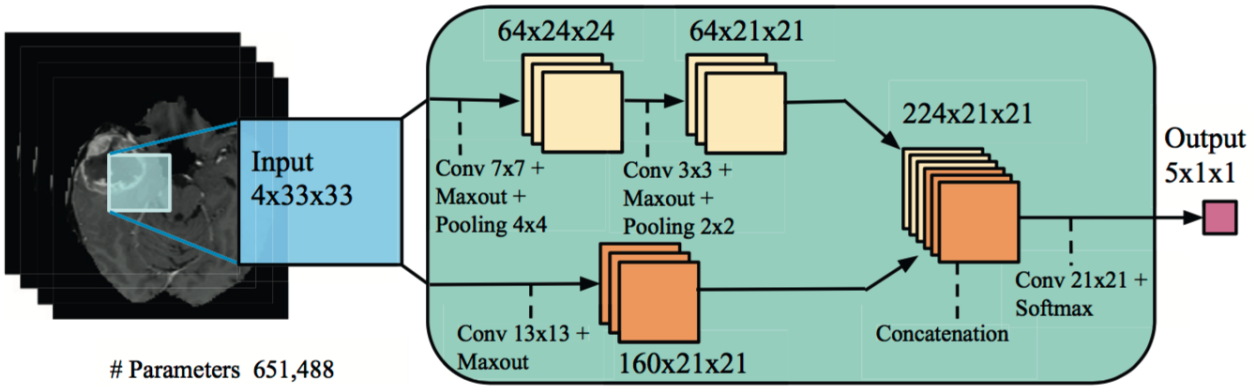
Image segmentation

- Compared to image classification,
  image segmentation can be seen as classification of every pixel / voxel
- The output is not a scalar per image, but a new image
- Each pixel / voxel is one training example,
  less training data required compared to image classification

Applications - scene segmentation

- 
- https://www.youtube.com/watch?v=PNzQ4PNZSzc

APPLICATIONS - BRAIN TUMOUR SEGMENTATION



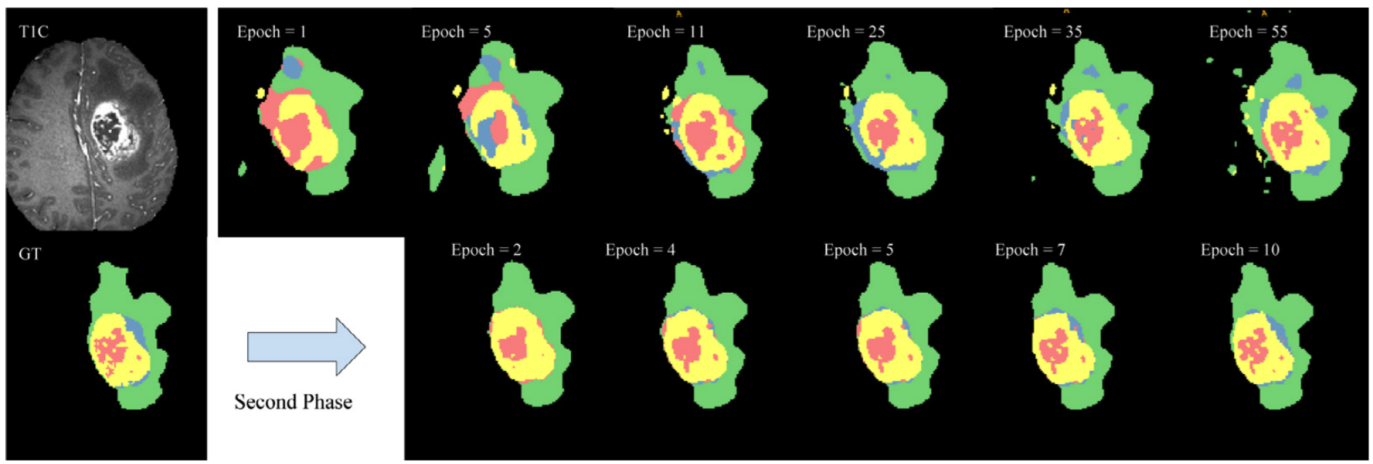
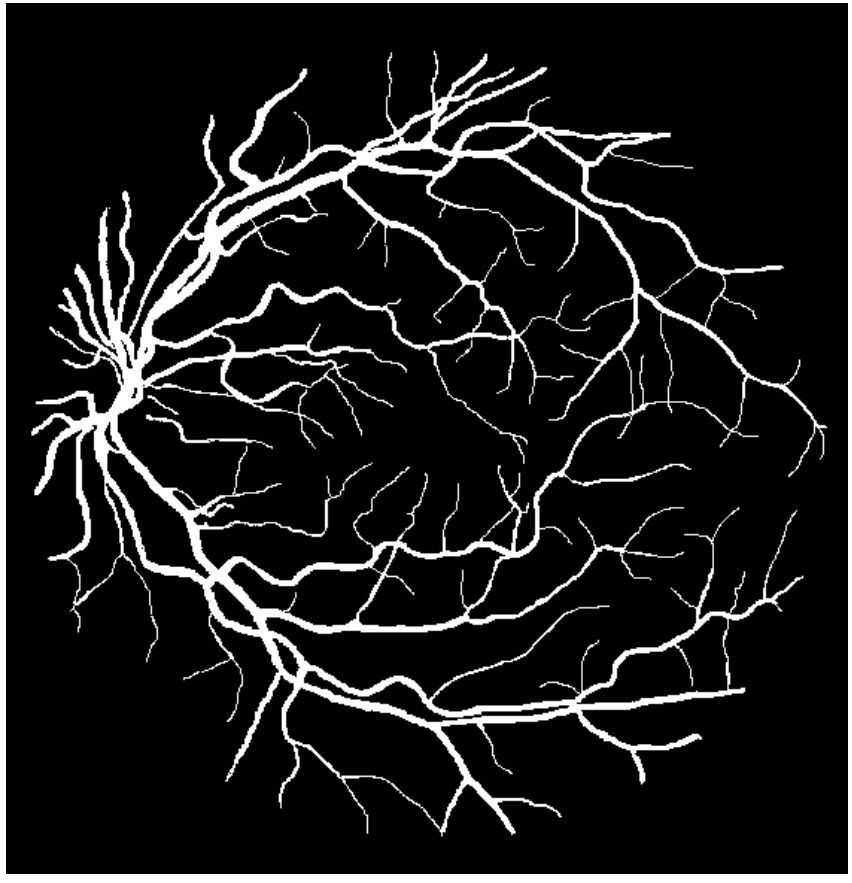


Fig. 6. Progression of learning in INPUTCASCADECNN*. The stream of figures on the first row from left to right show the learning process during the first phase. As the model learns better features, it can better distinguish boundaries between tumor sub-classes. This is made possible due to uniform label distribution of patches during the first phase training which makes the model believe all classes are equiprobable and causes some false positives. This drawback is alleviated by training a second phase (shown in second row from left to right) on a distribution closer to the true distribution of labels. The color codes are as follows: edema, enhanced tumor, necrosis, non-enhanced tumor.

Havaei, M., Davy, A., Warde-Farley, D., Biard, A., Courville, A., Bengio, Y., ... & Larochelle, H. (2017). Brain tumor segmentation with deep neural networks. Medical image analysis, 35, 18-31.

Training data for image segmentation - binary

Manual segmentation (time consuming)

Use some other segmentation algorithm, and edit segmentations

Loss functions for image segmentation

- Binary segmentation, binary cross entropy
- Weighted cross entropy, to handle class imbalance,

  $L = -(\beta\, t\, logy + (1 - t) log(1 - y))$

- To decrease false negatives, use $\beta > 1$,
  to decrease false positives, use $\beta < 1$
  (Tensorflow, weighted_cross_entropy_with_logits)
- Multi-class segmentation, categorical cross entropy

Loss functions for image segmentation

- Segmentation results are often evaluated using Dice score, but not so meaningful to calculate Dice for single pixel

  $$Dice = 1 - \frac{2\,t\,y+1}{y+t+1}$$

- Global Dice loss,

  $$Dice = 1 - \frac{2\sum_{pixels} t\,y}{\sum_{pixels} y+\sum_{pixels} t}$$

- Combinations such as cross-entropy (local) + Dice (global)

Segmentation architectures - U-net

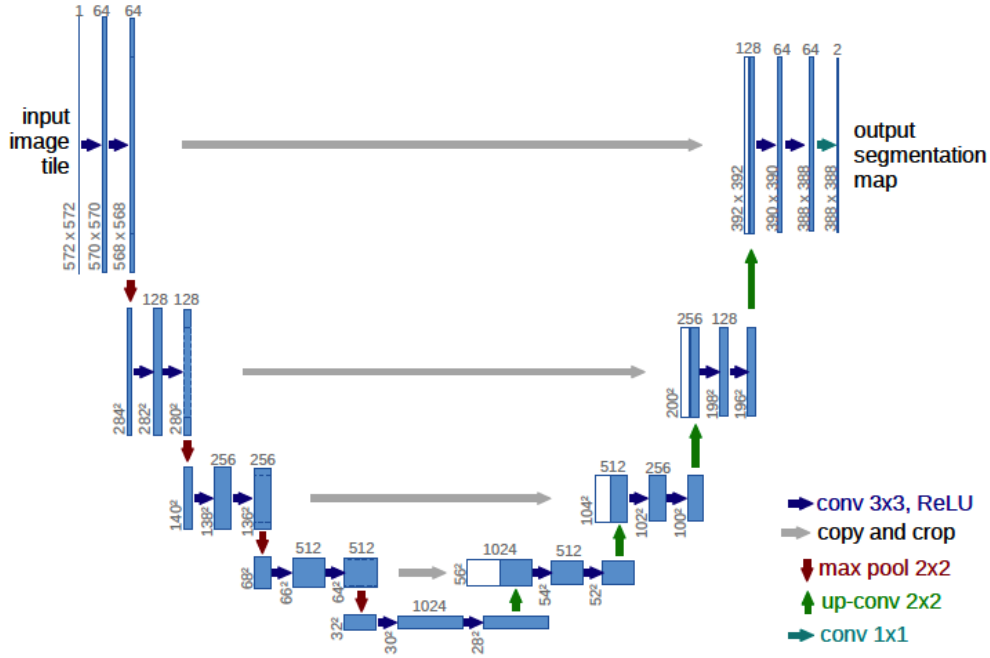


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

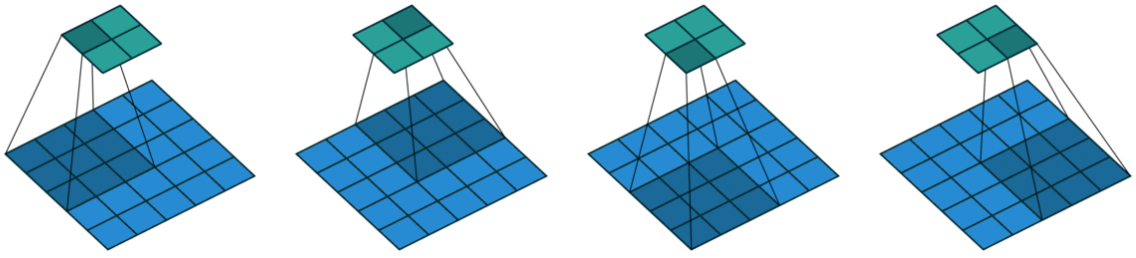
Consists of encoder (standard CNN), bridge, and decoder (upsampling)

Encoder uses Conv2D, decoder uses Conv2DTranspose

Uses skip connections, inspired by ResNet

Ronneberger et al., U-Net: Convolutional Networks for Biomedical Image Segmentation, arXiv:1505.04597

Conv2DTranspose layer

- Transposed convolution layer
  (also called Deconvolution or fractionally strided convolution)
- The need for transposed convolutions generally arises from the desire to use a transformation going in the opposite direction of a normal convolution, i.e., from something that has the shape of the output of some convolution to something that has the shape of its input while maintaining a connectivity pattern that is compatible with said convolution.
- 

Increasing resolution

- There are at least two ways to increase the resolution in a CNN; transposed convolution and resize-convolution
- Resize-convolution; first upsample image (using nearest neighbour interpolation), then perform regular convolution, good quality but slow
- Transposed-convolution; fast but often lower quality (e.g. checkerboard artifacts)

Multi-task learning (MTL)

- Train a network for several tasks; classification and segmentation
- Shared feature extractor (e.g. VGG16, ResNet-101, ...)
- Forcing a feature extractor to be good at several tasks
  can lead to better regularization => better performance

Multi-task learning (MTL)

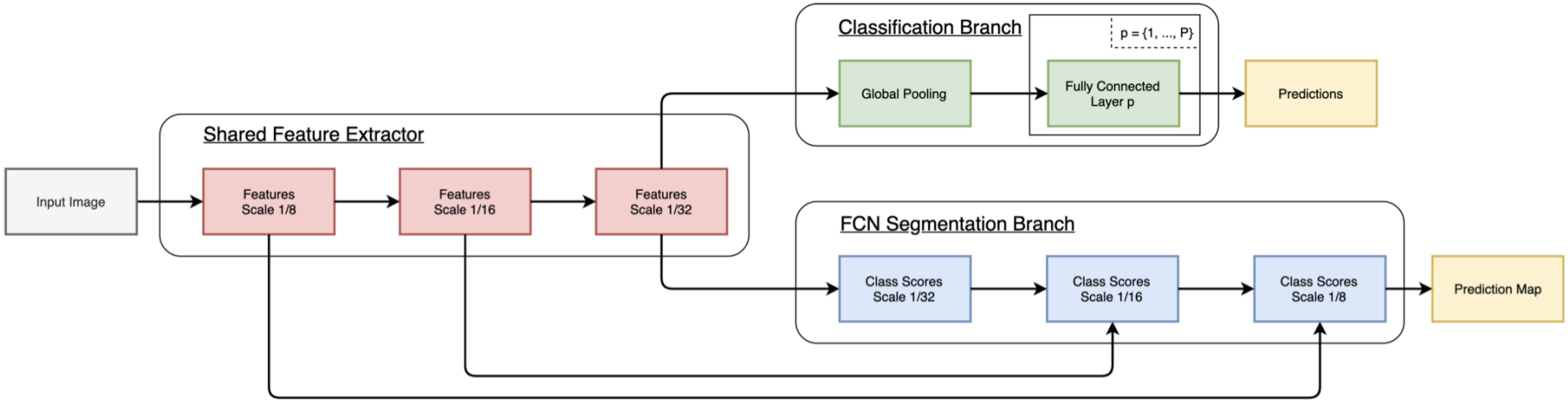


Figure 3.5: *MTL network architecture.*

Jesper Westell, Multi-Task Learning using Road Surface Condition Classification and Road Scene Semantic Segmentation, LIU-IMT-TFK-A–19/570–SE