Medical Images and Neural Networks

Deep Learning architectures

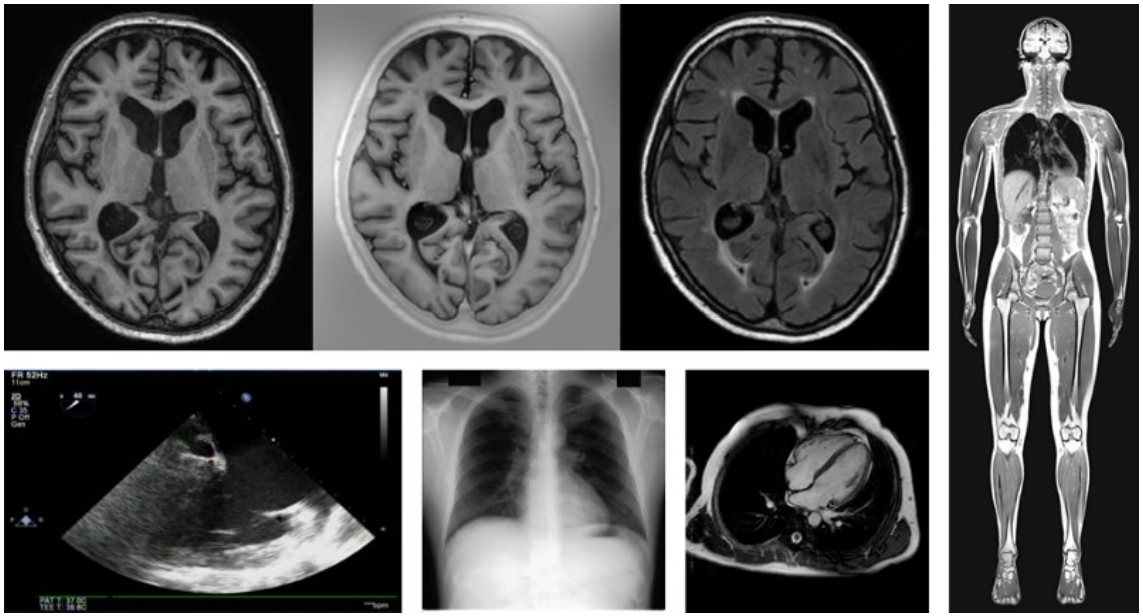
A Tutorial for Beginners

Maria Antonietta Pascali
maria.antonietta.pascali@isti.cnr.it

CNR ISTI, Signal and Images Lab

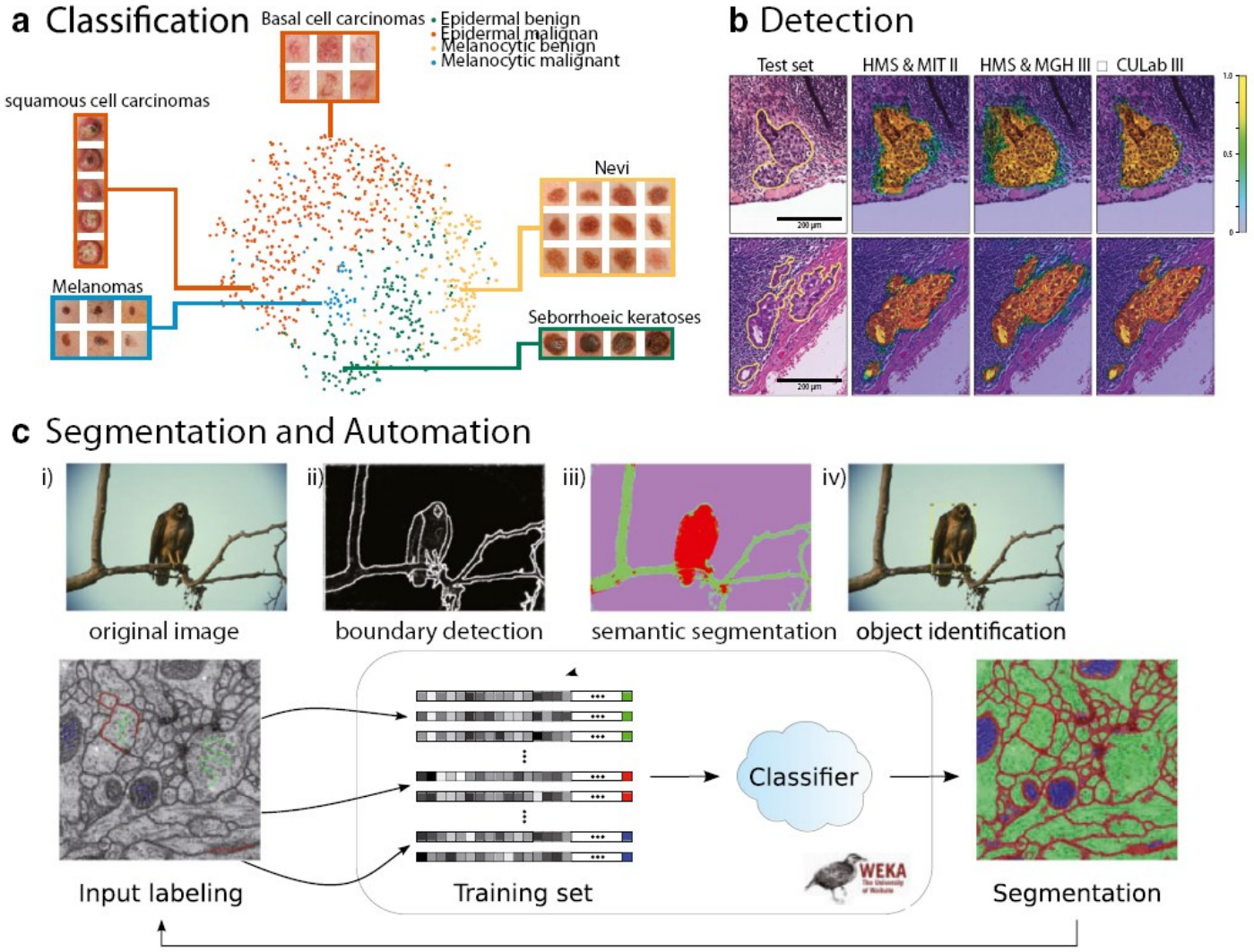
Medical images

Computer vision methods have long been employed to automatically analyze biomedical images.



Deep learning provides new methods to analyse (huge amount of) medical images: performing classification, segmentation, detection, …

Machine Learning for Medical Images



James A. Nichols, Hsien W. Herbert Chan, Matthew A. B. Baker, Machine learning: applications of artificial intelligence to imaging and diagnosis, Biophys Rev (2019) 11:111–118

Many architectures, many models..

- Data quality and availability
- Transfer learning

In the case of medical datasets, it is common to use a pre-trained ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), either as an initialization, or a fixed feature extractor for the task of interest.

Transfer Learning

- ConvNet as fixed feature extractor

Take a ConvNet pretrained on ImageNet, remove the last fully-connected layer (this layer's outputs are the 1000 class scores for a different task like ImageNet), then treat the rest of the ConvNet as a fixed feature extractor for the new dataset.

- Fine-tuning the ConvNet

The second strategy is to not only replace and retrain the classifier on top of the ConvNet on the new dataset, but to also fine-tune the weights of the pretrained network by continuing the backpropagation.

- Pretrained models

Since modern ConvNets take 2-3 weeks to train across multiple GPUs on ImageNet, it is common to see people release their final ConvNet checkpoints. E.g. the Caffe library has a Model Zoo where people share their network weights.

More details: http://cs231n.github.io/transfer-learning/

Data, data, and data

1. ***New dataset is small and similar to original dataset.*** It is not a good idea to fine-tune the ConvNet due to overfitting concerns. The best idea might be to train a Linear SVM or Softmax classifier on the CNN codes (features extracted).
2. ***New dataset is large and similar to the original dataset*.** Allowed fine-tuning through the full network.
3. ***New dataset is small but very different from the original dataset*.** Since the data is small, it is likely best to only train a linear classifier. Instead, it might work better to train the SVM classifier from activations somewhere earlier in the network.
4. ***New dataset is large and very different from the original dataset*.** Since the dataset is very large, we may expect that we can afford to train a ConvNet from scratch. However, in practice it is very often still beneficial to initialize with weights from a pretrained model. we would have enough data and confidence to fine-tune through the entire network.

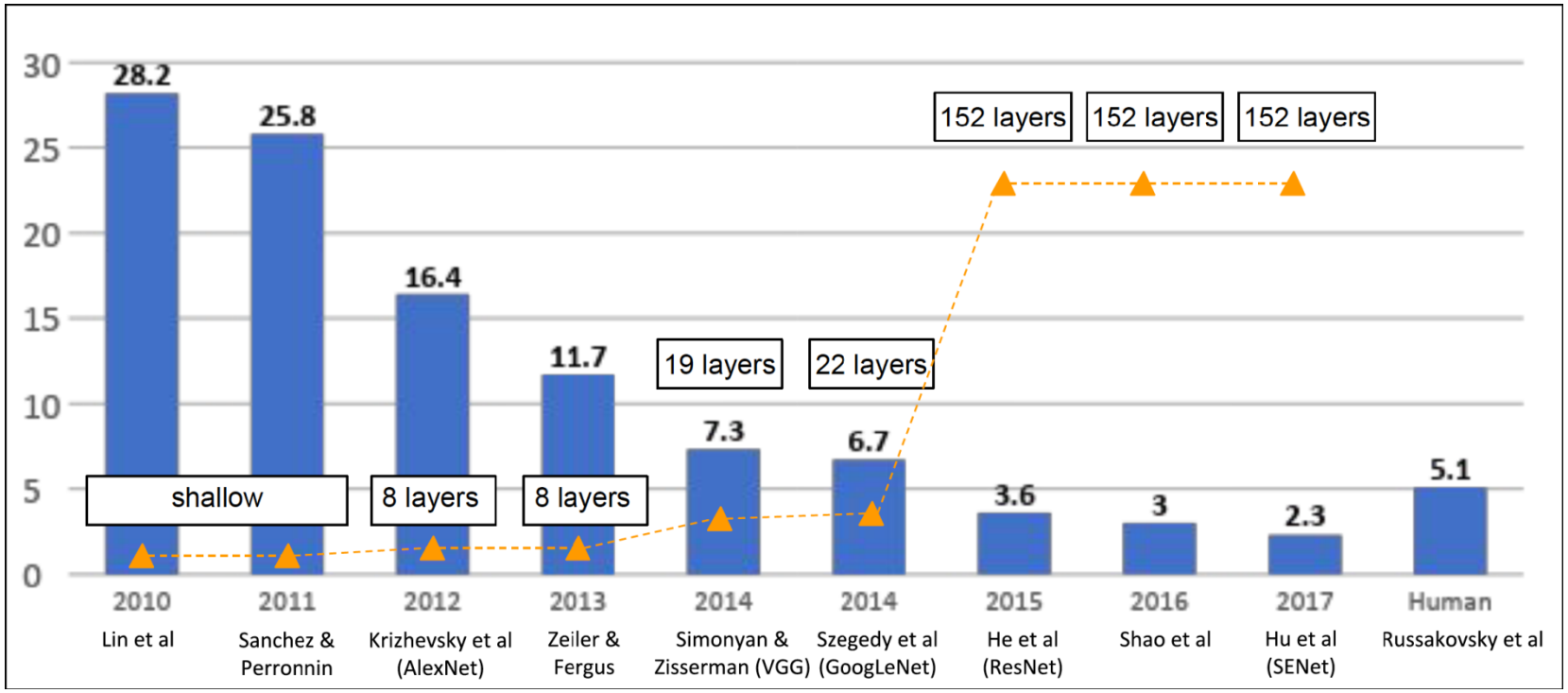
Deep Learning-based tools for medical image analysis

- **DLTK** is a neural networks toolkit written in python, on top of TensorFlow. It is developed to enable fast prototyping with a low entry threshold and ensure reproducibility in image analysis applications, with a particular focus on medical imaging.

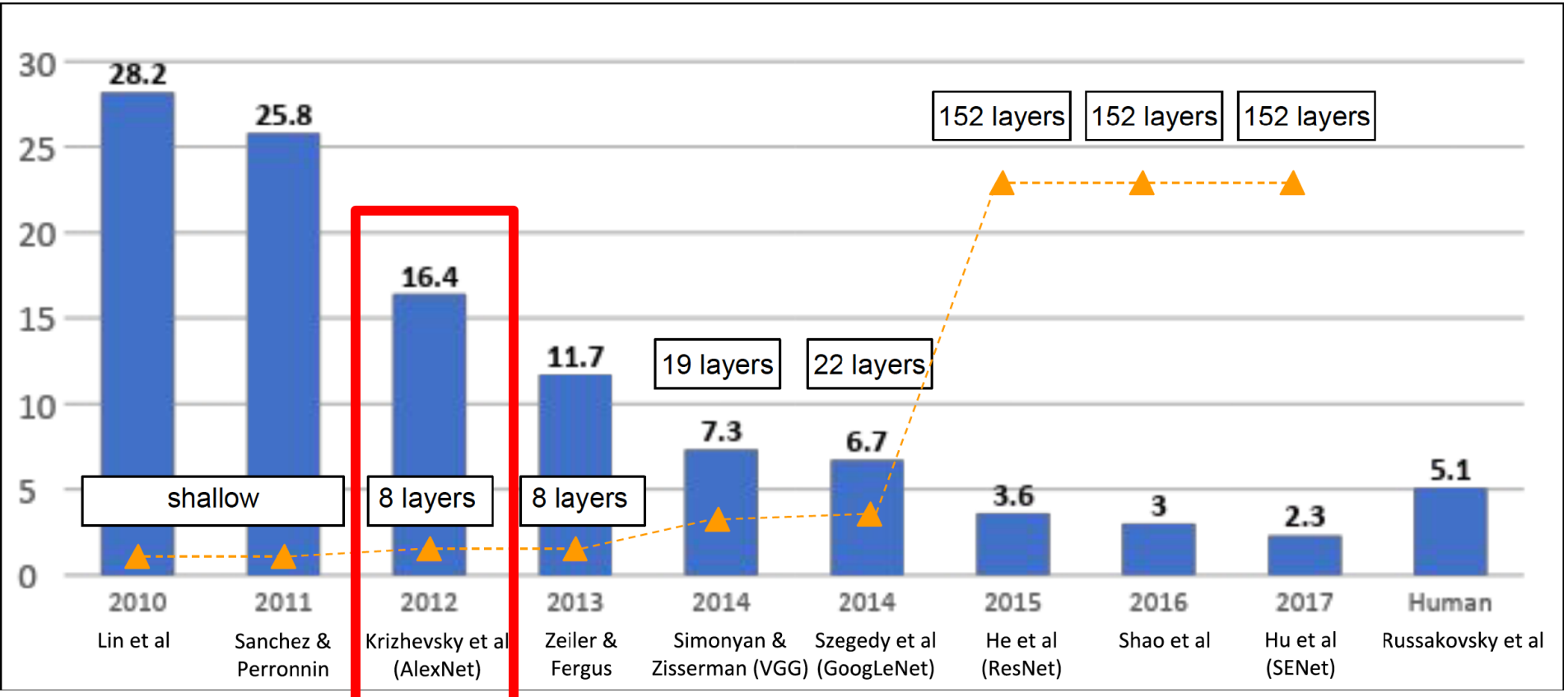
- **Jupyter** is a nonprofit organization created to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages".

Image Classification Challenge

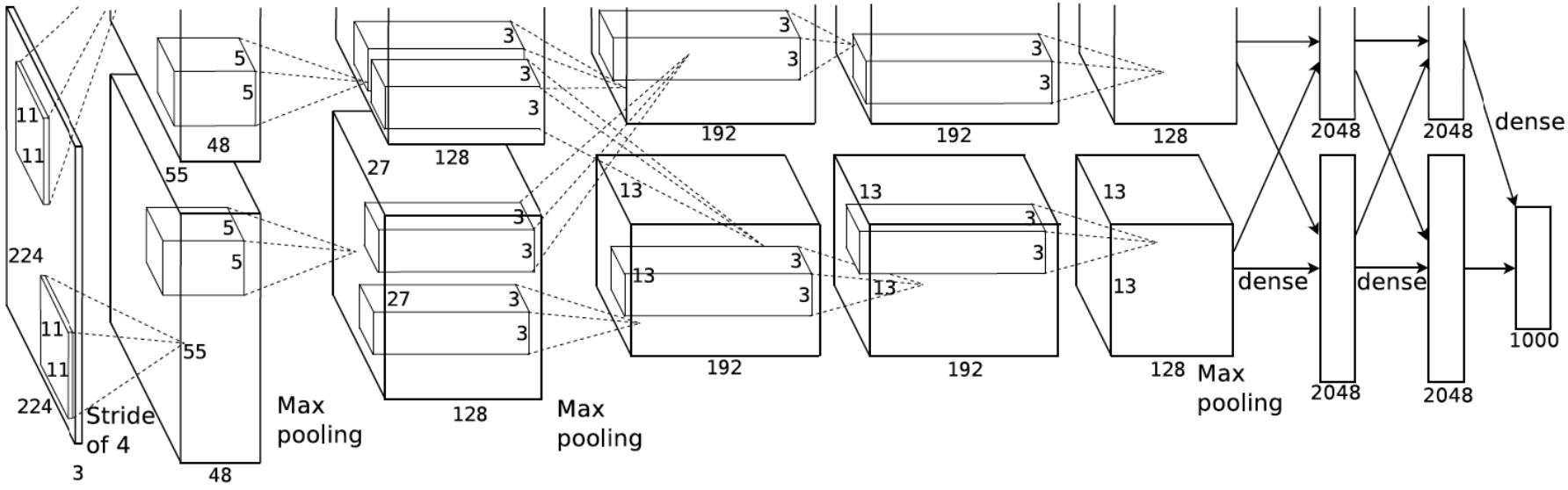
- IMAGENET : Large Scale Visual Recognition
  - 1000 Classes of objects
  - 14 311 670 images



Alexnet [Krizhevsky et al, 2012]



Alexnet [Krizhevsky et al, 2012]



- First model to perform well on the ImageNet dataset (~11% lower error than runner up)
- Combined techniques used in today's architectures, like ReLU, data augmentation and dropout
- Used GPUs for training
- Largely responsible for the deep learning revolution in computer vision

ZFnet [Zeiler and Fergus, 2014]

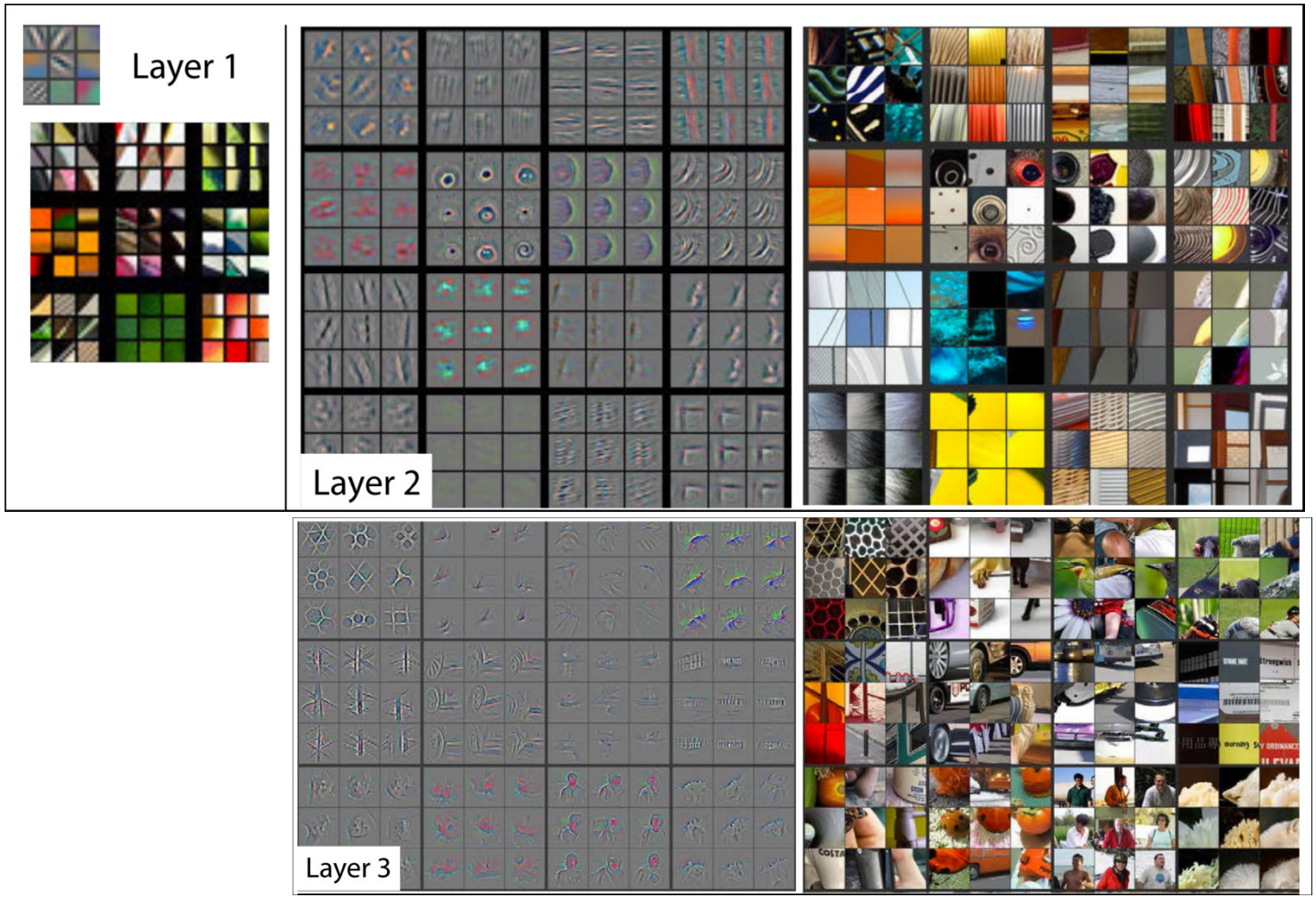
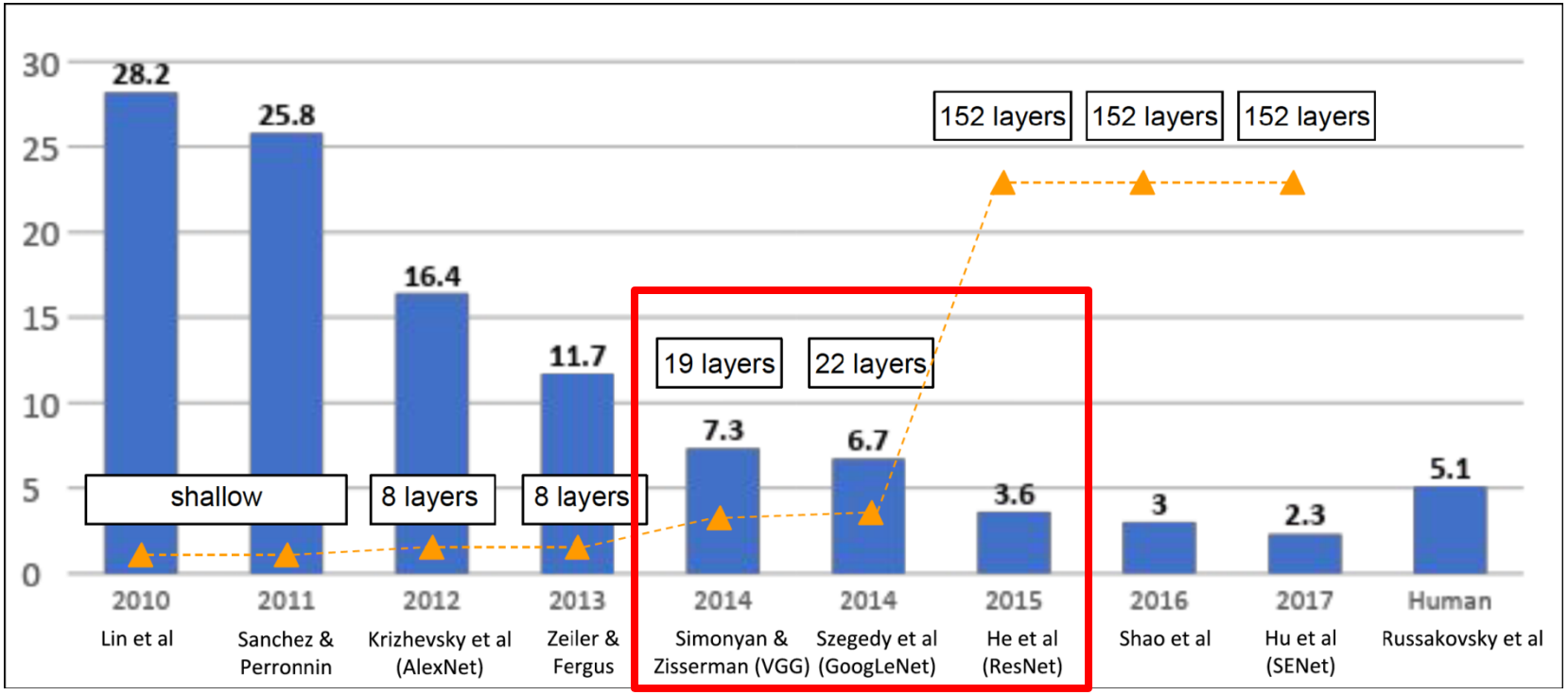


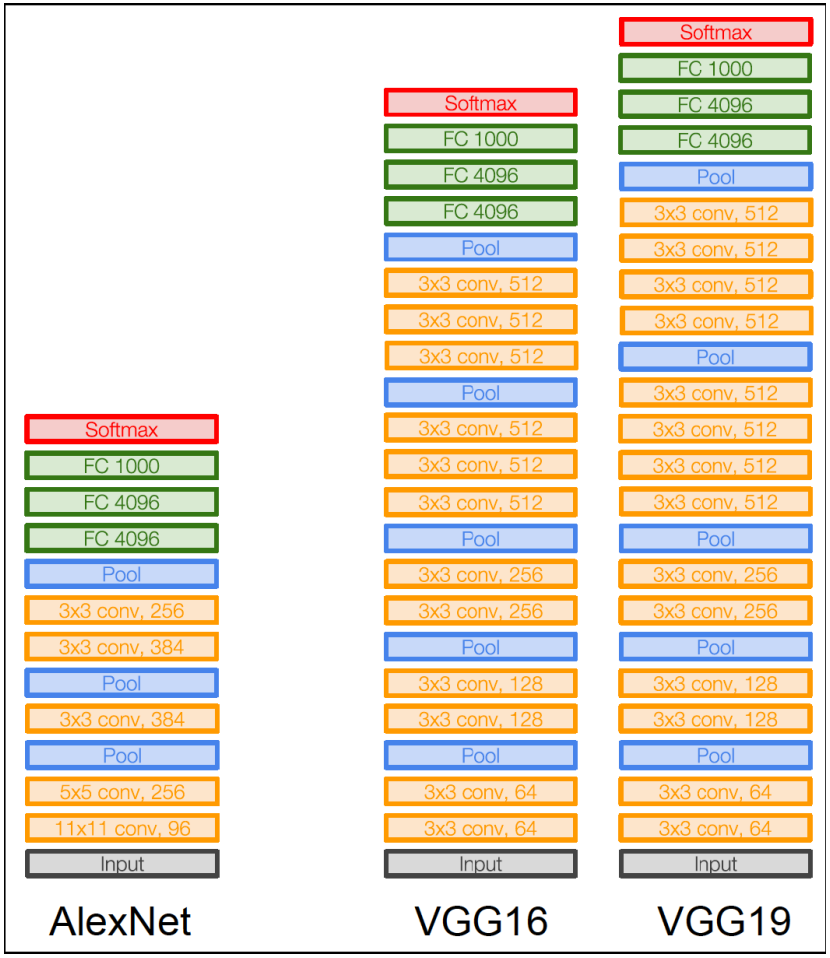
Image from: Zeiler and Fergus. "Visualizing and understanding convolutional networks." ECCV, 2014.

Image Classification Challenge

- IMAGENET : Large Scale Visual Recognition
  - 1000 Classes of objects
  - 14 311 670 images



VGG [Simonyan and Zisserman, 2014]

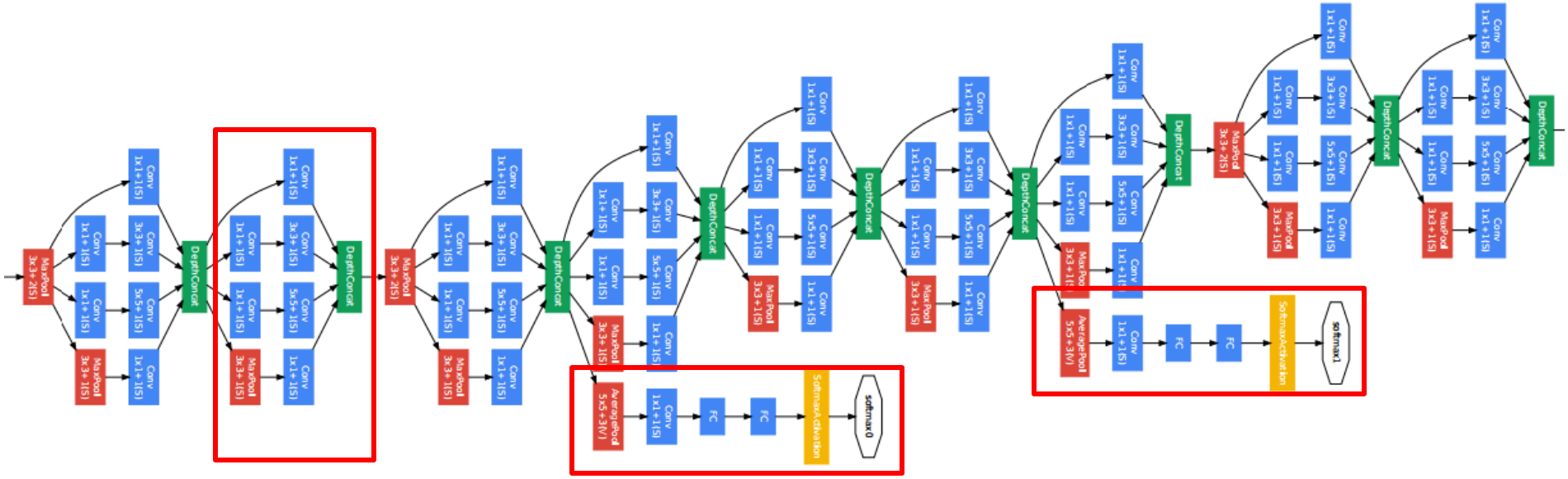


Simpler structure
only 3x3 convolutions, ReLU and 2x2 max pooling

Deeper network
16 and 19 layers (compared to 8 for AlexNet)

Key idea
cascading two 3x3 convolutions gives the same receptive field as a 5x5 convolution, with much less parameters

Inception [Szegedy et al, 2014]

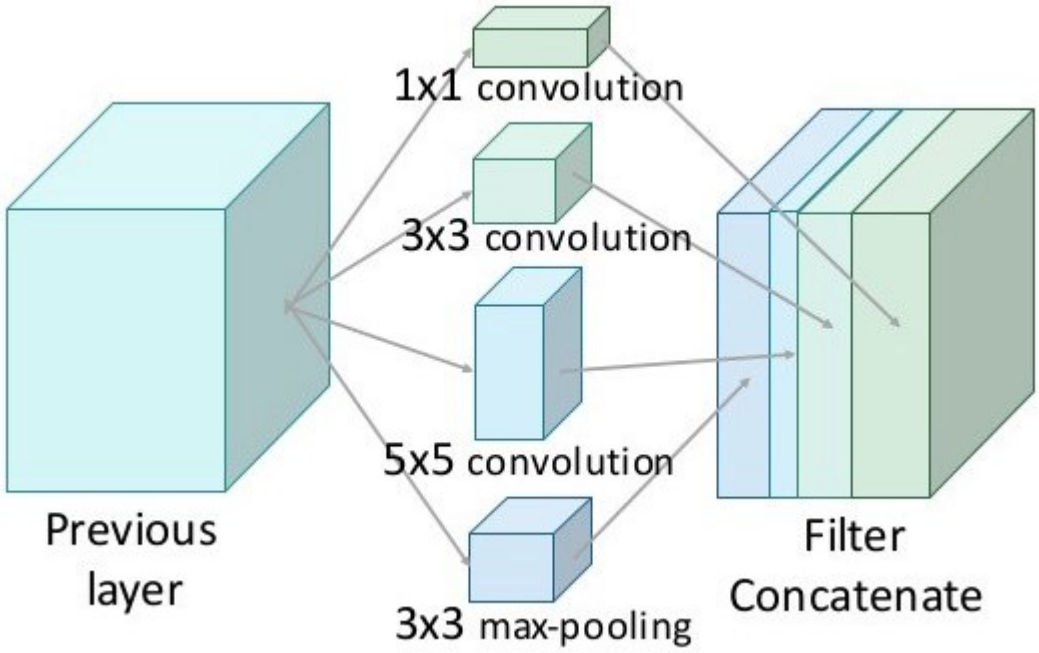


1) Repeating blocks called *Inception module*

2) Intermediate classification losses to inject gradient in middle layers

3) FC layers replaced by average pooling (fewer parameters)

GoogLeNet (Inception V1)



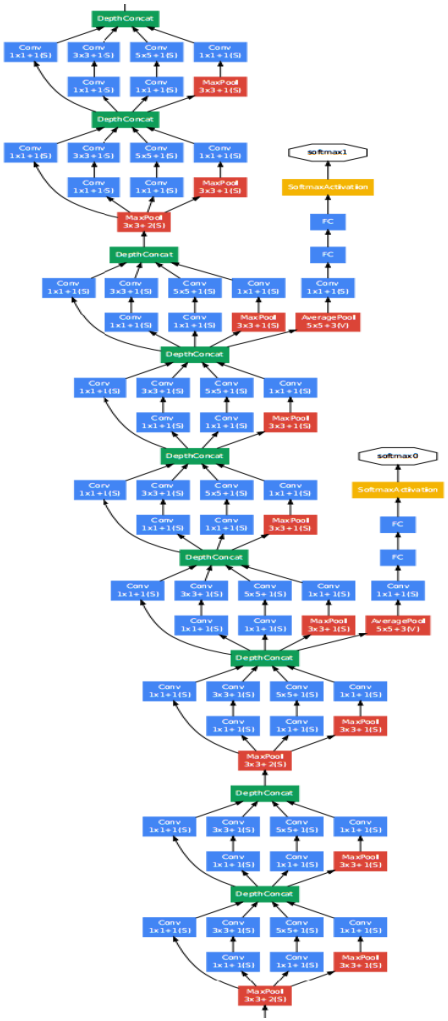
Choice for each layer
Convolution or pooling ?
If convolution, what kernel size ?

Key idea
Compute all in parallel
Concatenate results
Let the learning decide

AlexNet and GoogLeNet

AlexNet

8 layers
~62M parameters

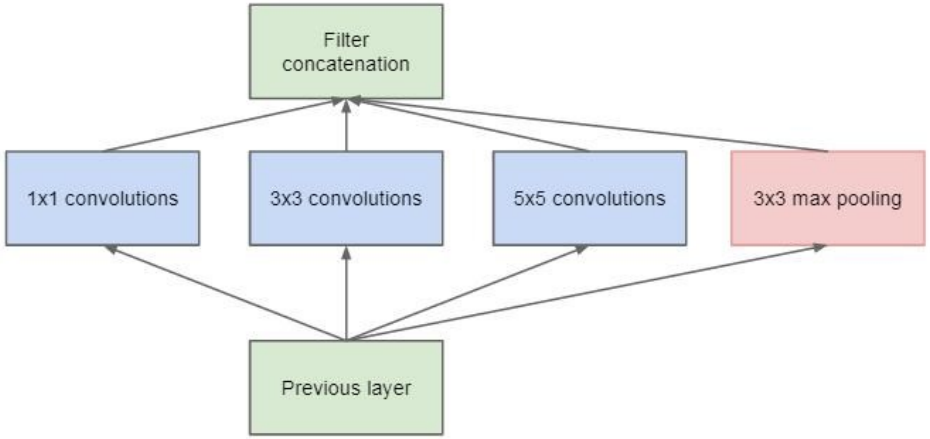


22 layers
~5M parameters

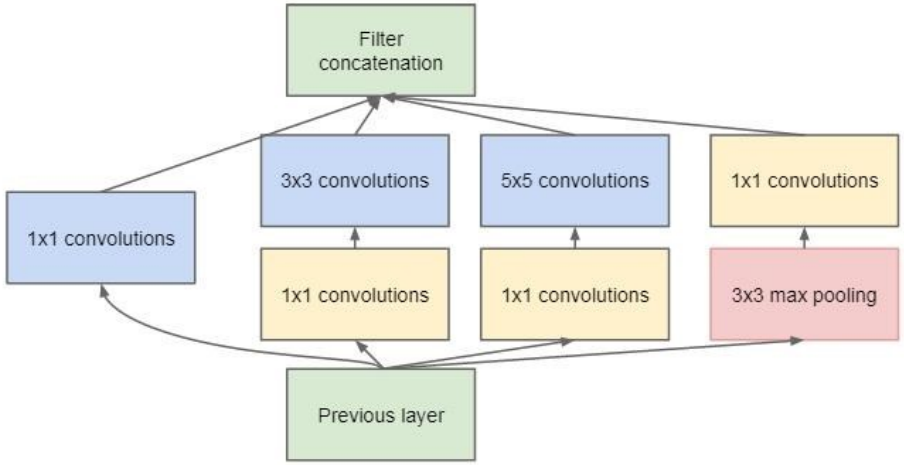
GoogLeNet

Inception V3

Problem: this gives too many outputs and parameters



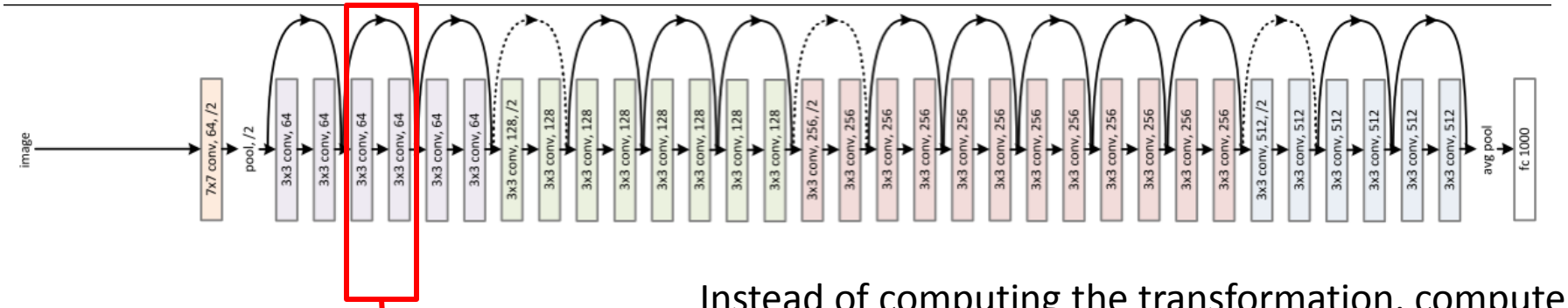
(a) Inception module, naïve version

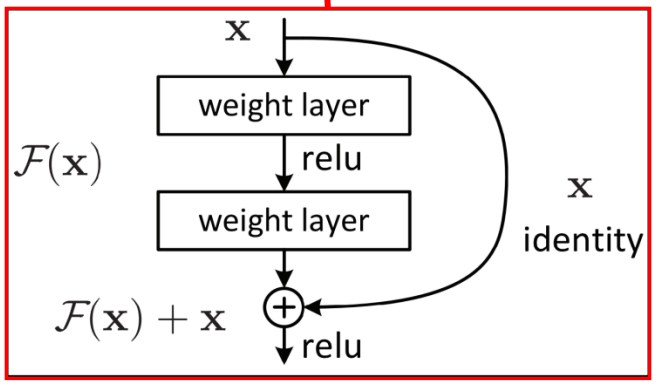


(b) Inception module with dimension reductions

Solution: Reduce dimensionality using *bottleneck layers* composed of 1x1 convolutions

ResNet [He et al, 2016]





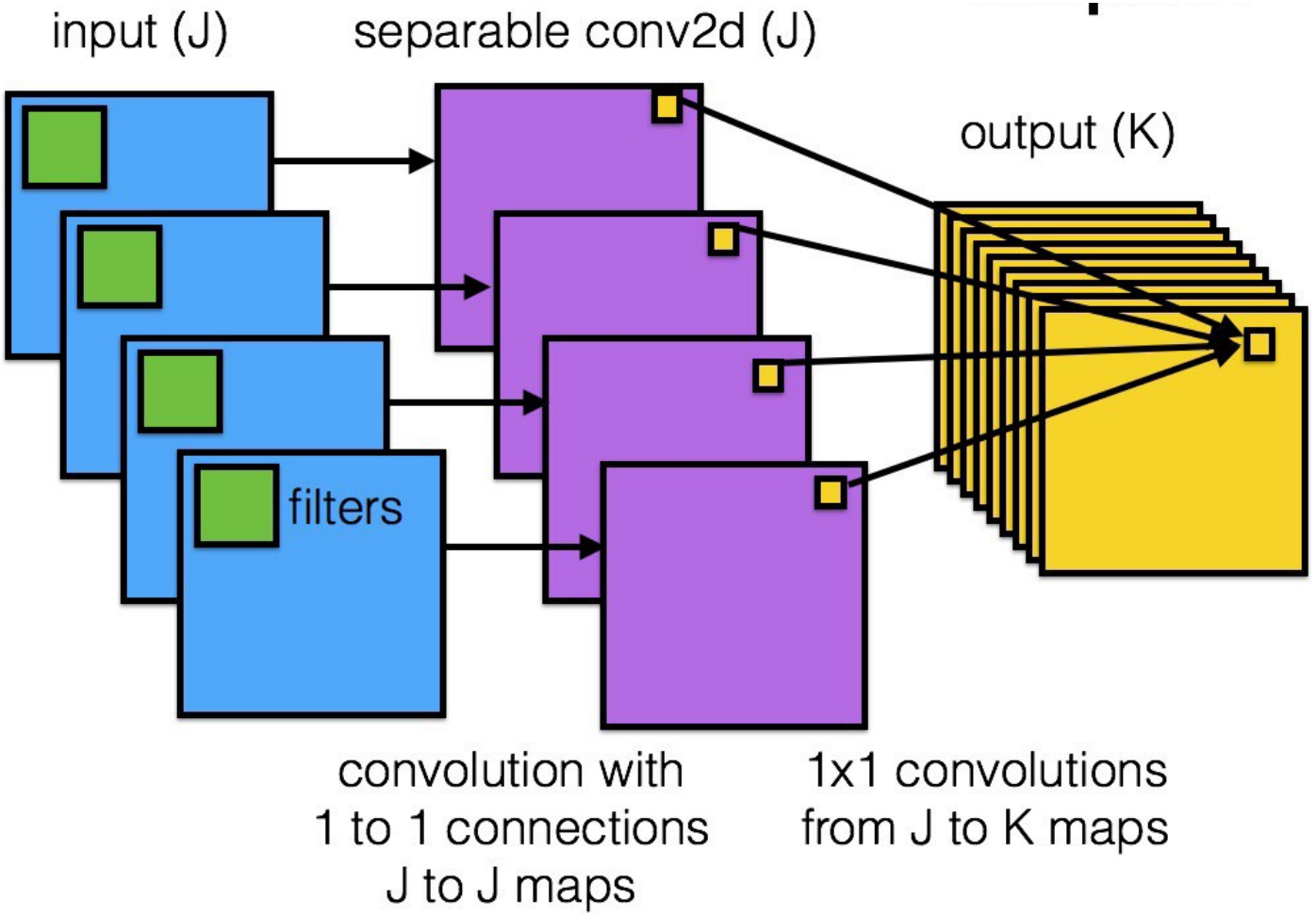
Instead of computing the transformation, compute the residual required to have the transformation

The residual requires less information to model, so *possibly* easier to learn.

Residual connections help gradient flow during back-propagation.

Enables very deep networks (over 100 layers)

Xception


input (J)
separable conv2d (J)
output (K)
filters
convolution with
1 to 1 connections
J to J maps
1x1 convolutions
from J to K maps

Let’s play with Deep Learning tools!

- Objective: Classification of medical images
- Dataset : 75 images (breast and abdomen X-ray)
- DL architecture: Inception V3
- Environment: Jupyter Notebook

Il tutorial segue nel dettaglio :
1. Articolo : https://link.springer.com/article/10.1007/s10278-018-0079-6
2. Github repository & code:
https://github.com/paras42/Hello_World_Deep_Learning/blob/master/HelloWorldDeepLearning.ipynb

Installation on Windows 10

- CPU (or GPU)
- Anaconda 3 (1.9.7)
- Python 3 (3.6.8)
- Jupyter Notebook (5.7.8)
  - TensorFlow 1.13.1
  - Keras 2.2.4

Initial steps

The notebooks are organized into cells, whereby each cell may be run independently.

```
# load requirements from the Keras library
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D
from keras.models import Model
from keras.optimizers import Adam
```

```
# dimensions of our images
img_width, img_height = 299, 299

# directory and image information
train_data_dir = '../data/train'
validation_data_dir = '../data/val'
```

1. Load requirements from the Keras library.
2. Specify information regarding the images.

Setting parameters

```
# epochs = number of passes of through training data
# batch_size = number images processed at same time
train_samples = 65
validation_samples = 10
epochs = 20
batch_size = 5
```

1. Set the number of epochs (number of passes through the training data)
2. Set the batch size (number of images processed at the same time)

Model: import InceptionV3

```
# build the Inception V3 network, use pretrained weights from ImageNet
# remove top fully connected layers by include_top=False

base_model = applications.InceptionV3(weights='imagenet', include_top=False,
input_shape=(img_width, img_height, 3))
```

1. Start with the original Inception V3 model.
2. Remove top or fully connected layers from the original network.
3. Use pretrained weights from ImageNet.

Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception Architecture for computer vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition: 2818–2826, 2016

Model: redefine top layers

```
model_top = Sequential()
model_top.add(GlobalAveragePooling2D(input_shape=base_model.output_shape[1:],
data_format=None)),
model_top.add(Dense(256, activation='relu'))
model_top.add(Dropout(0.5))
model_top.add(Dense(1, activation='sigmoid'))
model = Model(inputs=base_model.input, outputs=model_top(base_model.output))

# Compile model using Adam optimizer with common values and binary cross entropy loss
# Use low learning rate (lr) for transfer learning
model.compile(optimizer=Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-
08,decay=0.0), loss='binary_crossentropy', metrics=['accuracy'])
```

1. Add new layers on top of the original model. There are many possibilities, but here, we add a global average pooling layer, a fully connected layer with 256 nodes, dropout, and sigmoid activation.
2. Define an optimizer; in this case, it is the Adam optimizer with default settings.

Kingma D, Ba JA: A method for stochastic optimization. arXiv preprint arXiv 1412:6980, 2014

Data augmentation

```
# Some on-the-fly augmentation options
train_datagen = ImageDataGenerator(
        rescale= 1./255,    # Rescale pixel values to 0-1 to aid CNN processing
        shear_range=0.2,    # 0-1 range for shearing
        zoom_range=0.2,     # 0-1 range for zoom
        rotation_range=20,    # 0-180 range, degrees of rotation
        width_shift_range=0.2,   # 0-1 range horizontal translation
        height_shift_range=0.2,   # 0-1 range vertical translation
        horizontal_flip=True)    # set True or False


val_datagen = ImageDataGenerator(
        rescale=1./255)    # Rescale pixel values to 0-1 to aid CNN processing
```

Rescale images and specify augmentation methods

Training and validation generator

```
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary')

validation_generator = train_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary')
```

Specification about the directory containing the files, size of images, and batch size.

Class mode is set to ‘binary’ for a 2-class problem.

Generator randomly shuffles and presents images in batches to the network.

Model fitting

```
# Fine-tune the pretrained Inception V3 model using the data generator
# Specify steps per epoch (number of samples/batch_size)

history = model.fit_generator(
          train_generator,
          steps_per_epoch=train_samples // batch_size,
          epochs=epochs,
          validation_data=validation_generator,
          validation_steps=validation_samples // batch_size)
```

After executing the code above, the model begins to train. In only five epochs, the training accuracy equals 89% and validation accuracy 100%. The validation accuracy is usually lower than the training accuracy, but in this case, it is higher likely because there are only 10 validation cases. The training and validation loss both decrease, which indicates that the model is “learning.”

Training

```
Epoch 1/20
13/13 [==============================] - 2s - loss: 0.5701 - acc: 0.7231 - val_loss: 0.7761 - val_acc: 0.6000
Epoch 2/20
13/13 [==============================] - 2s - loss: 0.1420 - acc: 0.9692 - val_loss: 0.4471 - val_acc: 0.8000
Epoch 3/20
13/13 [==============================] - 2s - loss: 0.1645 - acc: 0.9385 - val_loss: 0.2711 - val_acc: 0.9000
Epoch 4/20
13/13 [==============================] - 2s - loss: 0.0807 - acc: 0.9692 - val_loss: 0.2032 - val_acc: 0.9000
Epoch 5/20
13/13 [==============================] - 2s - loss: 0.2372 - acc: 0.9538 - val_loss: 0.4368 - val_acc: 0.8000
Epoch 6/20
13/13 [==============================] - 2s - loss: 0.0766 - acc: 0.9692 - val_loss: 0.0848 - val_acc: 1.0000
```

Training metrics:

loss, training loss; acc, training accuracy; val_loss, validation loss; val_acc, validation accuracy.

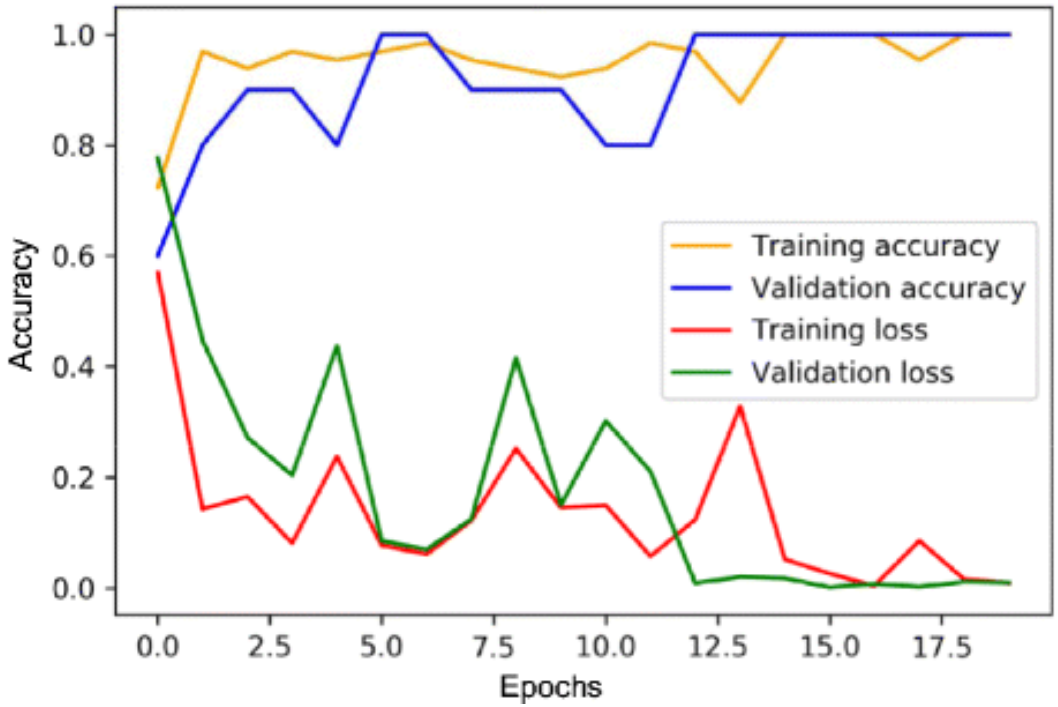
13 refers to the number of batches (13 batches × 5 images per batch = 65 training images).

20 refers to number of epochs.

Accuracy

```
# import matplotlib library, and plot training cuve
import matplotlib.pyplot as plt
print(history.history.keys())

plt.figure()
plt.plot(history.history['acc'], 'orange', label='Training accuracy')
plt.plot(history.history['val_acc'], 'blue', label='Validation accuracy')
plt.plot(history.history['loss'], 'red', label='Training loss')
plt.plot(history.history['val_loss'], 'green', label='Validation loss')
plt.legend()
plt.show()
```



Test

```
# import numpy and keras preprocessing libraries
import numpy as np
from keras.preprocessing import image

# load, resize, and display test images
img_path='../data/test/chest_test_001.png'
img_path2='../data/test/abd_test_001.png'
img = image.load_img(img_path, target_size=(img_width, img_height))
img2 = image.load_img(img_path2, target_size=(img_width, img_height))
plt.imshow(img)
plt.show()

# convert image to numpy array, so Keras can render a prediction
img = image.img_to_array(img)

# expand array from 3 dimensions (height, width, channels) to 4 dimensions (batch size,
height, width, channels)
# rescale pixel values to 0-1
x = np.expand_dims(img, axis=0) * 1./255

# get prediction on test image
score = model.predict(x)
print('Predicted:', score, 'Chest X-ray' if score < 0.5 else 'Abd X-ray')

# display and render a prediction for the 2nd image
plt.imshow(img2)
plt.show()
img2 = image.img_to_array(img2)
x = np.expand_dims(img2, axis=0) * 1./255
score2 = model.predict(x)
print('Predicted:', score2, 'Chest X-ray' if score2 < 0.5 else 'Abd X-ray')
```

Predicted: [[0.00007]] Chest X-ray

Predicted: [[0.99823]] Abd X-ray

Steps for performing inference on test cases.
Displaying of image and generating a prediction score.

Conclusion

With only 65 training cases, the power of transfer learning and deep neural networks, let us to build an accurate classifier that can differentiate chest vs. abdominal radiographs with a small amount of code.

The availability of frameworks and high-level libraries makes machine learning more accessible in medical imaging. Try them!

Thank you
for the attention!