Fourth Industrial Summer School

Advanced Machine Learning

Neural Networks and Deep learning-Part4

Session Objectives

- ✓ Introduction
- ✓ Fundamentals
- ✓ Neural Network Intuitions
- ✓ 2-Layer Neural Network
- ✓ Deep Neural Networks
- ✓ CNNs
- ✓ Keras with Tensorflow

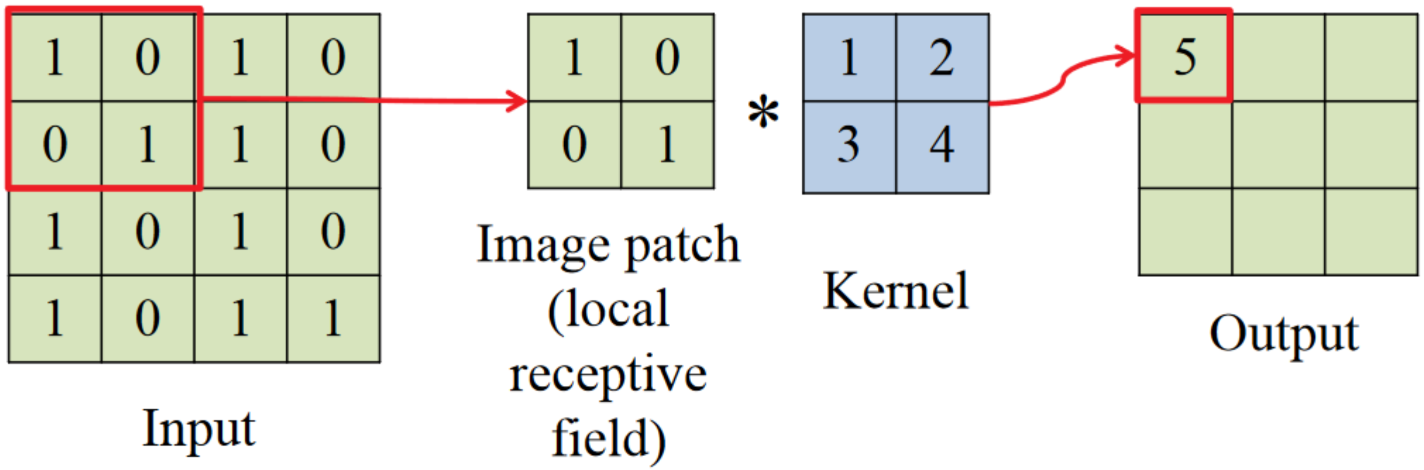
CNNs

Why CNNs

- Used mostly with images
- Translation invariance
- How was it before NNs?

CNN:

- A dot product of a kernel (aka filter) and a patch of an image (local receptive field) of the same size.



CNNs-Edge detection

Original image

Kernel

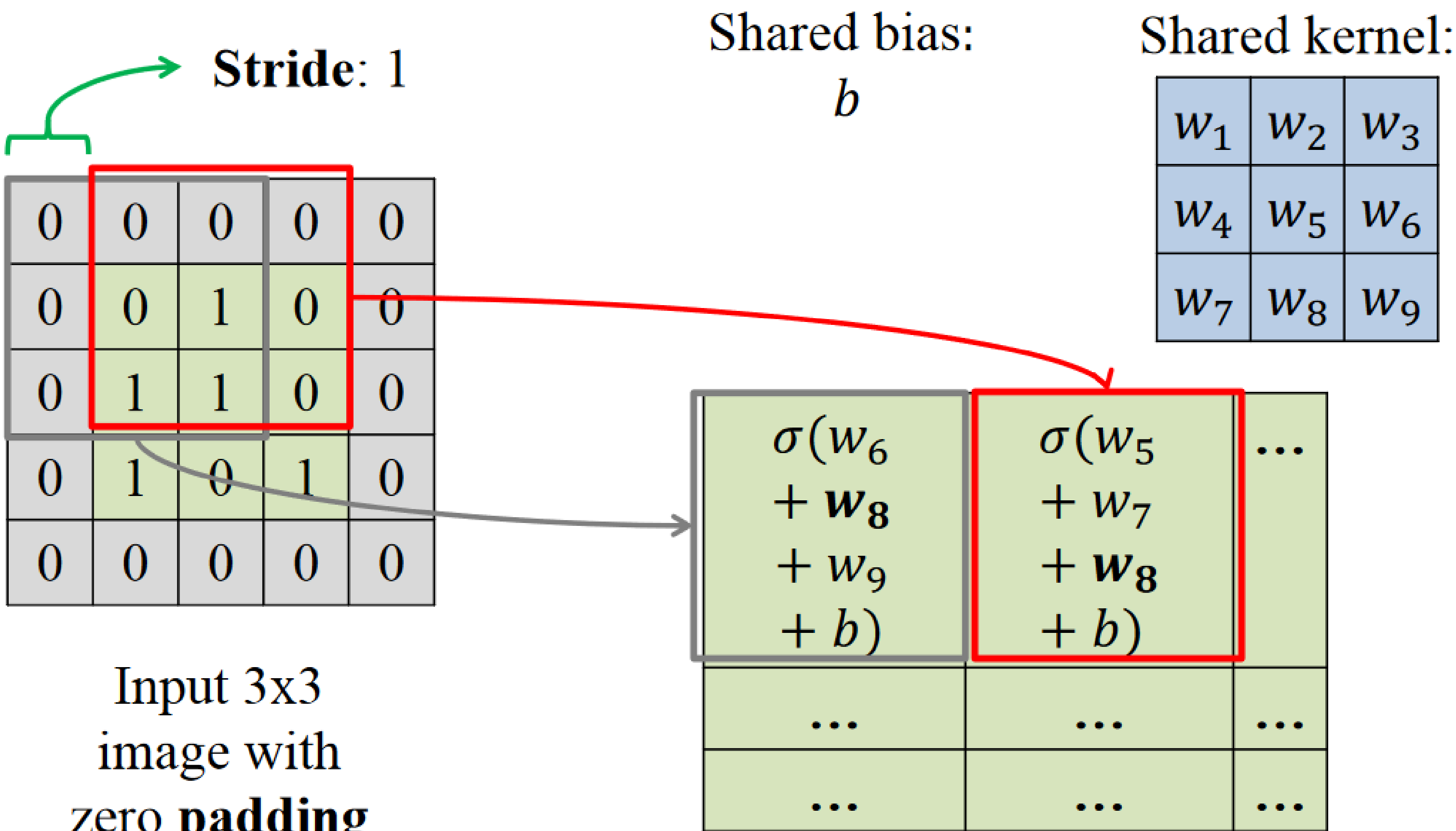
$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

Edge detection

Sums up to 0 (black color) when the patch is a solid fill

- It can also do sharpening and blurring.

CNN-At work



Input 3x3 image with zero **padding** (grey area)

9 output neurons (**feature map**) with only 10 parameters

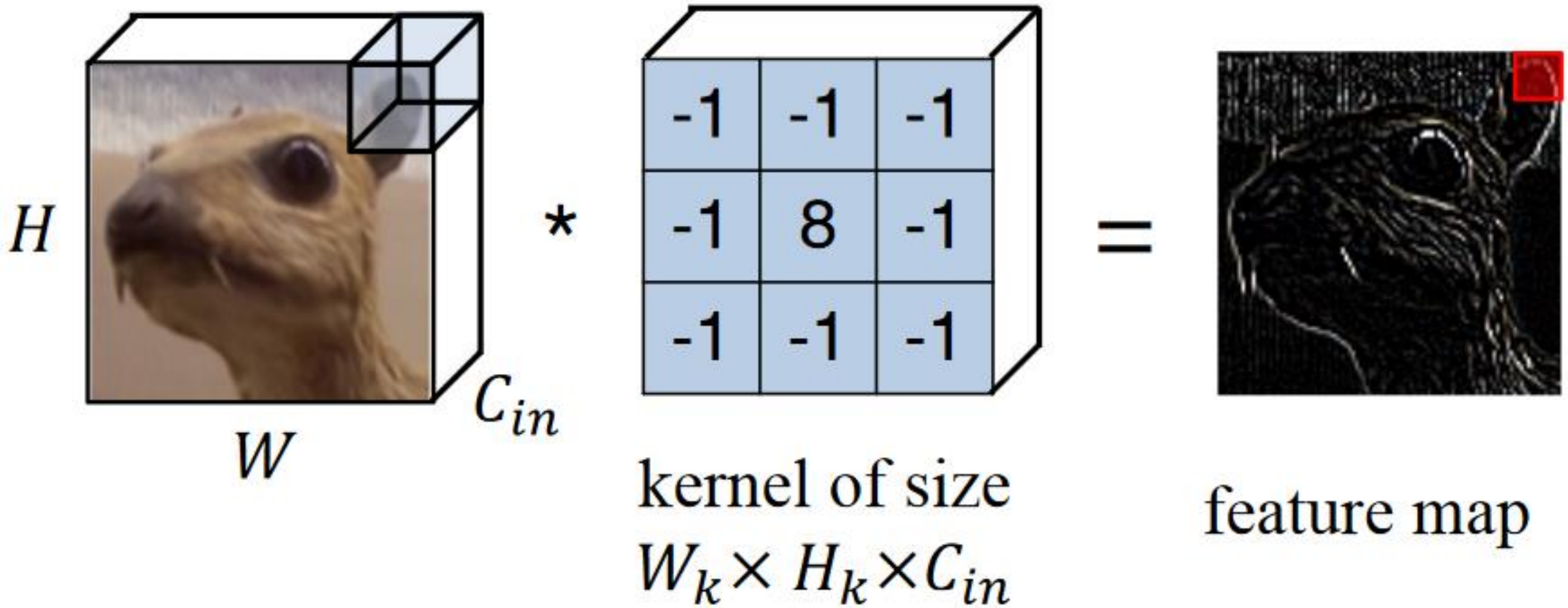
Backpropogation for CNN

- Sum of all the gradients used by a weight in a layer.

Other properties

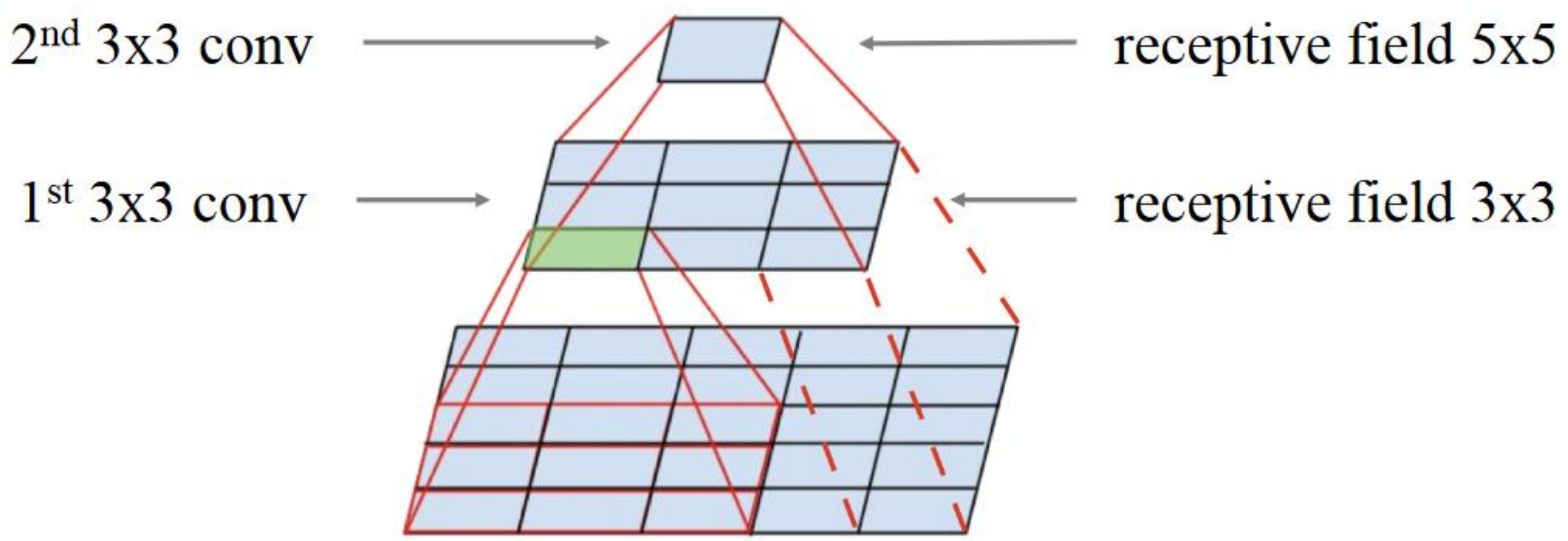
- Same kernel is used for every output neuron of a layer, thus fewer weights to be trained.
- So, 30x30 image with 5x5 kernel will have how many parameters?
- What about fully connected layer?
- Very important in high-dimension images.

CNN for colored images



Stacking CNNs

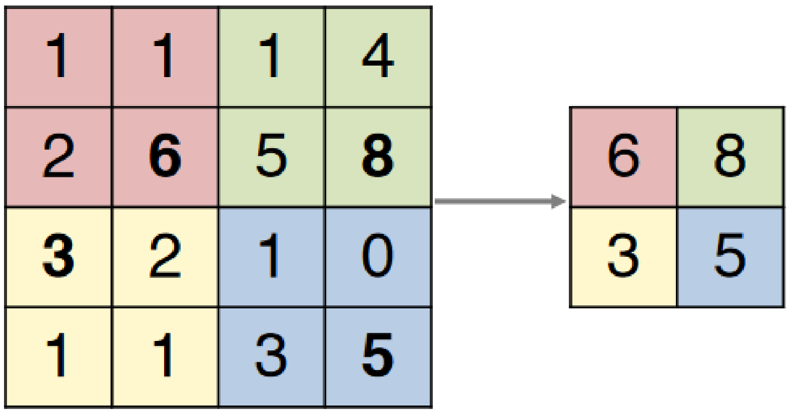
- To capture region of interest of different resolutions
- One kernel is not enough.



What about translation of regions

- Pooling solves it a greater degree.
- Pooling layer does not have a kernel
  - It calculates maximum (or average) over the patch.

Single depth slice

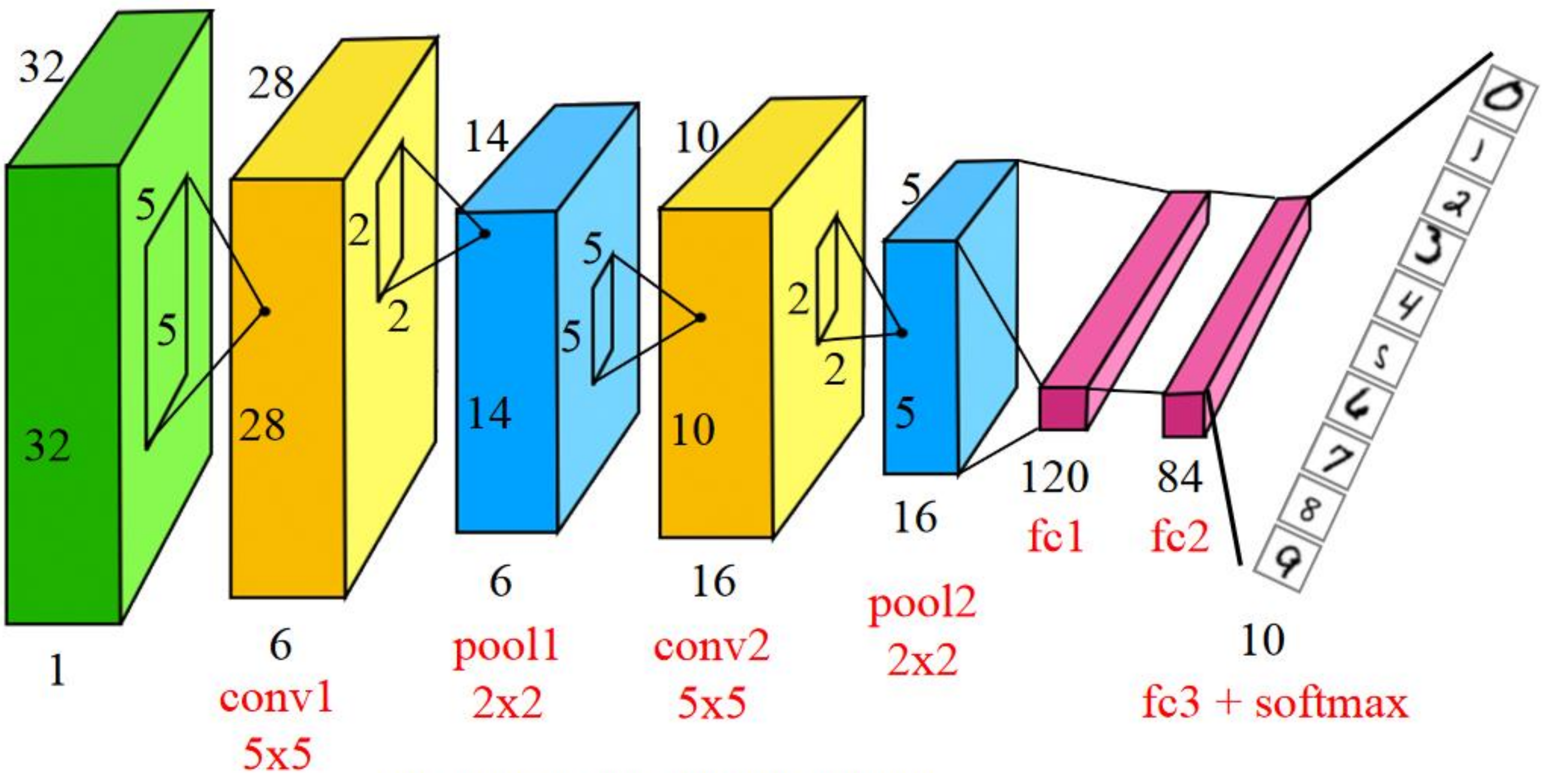


2x2 **max pooling** with stride 2

- Backpropagation?

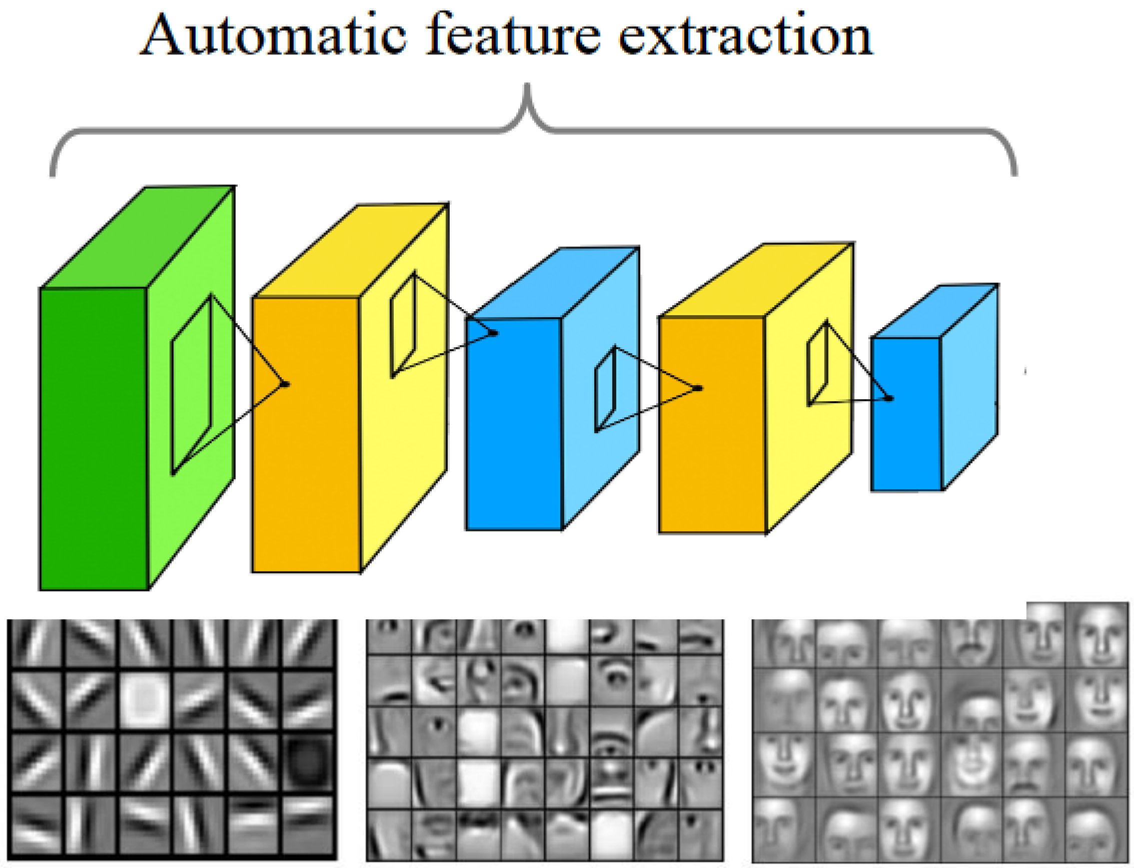
An example CNN

- LeNet-5 architecture



http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

Learning deep representations

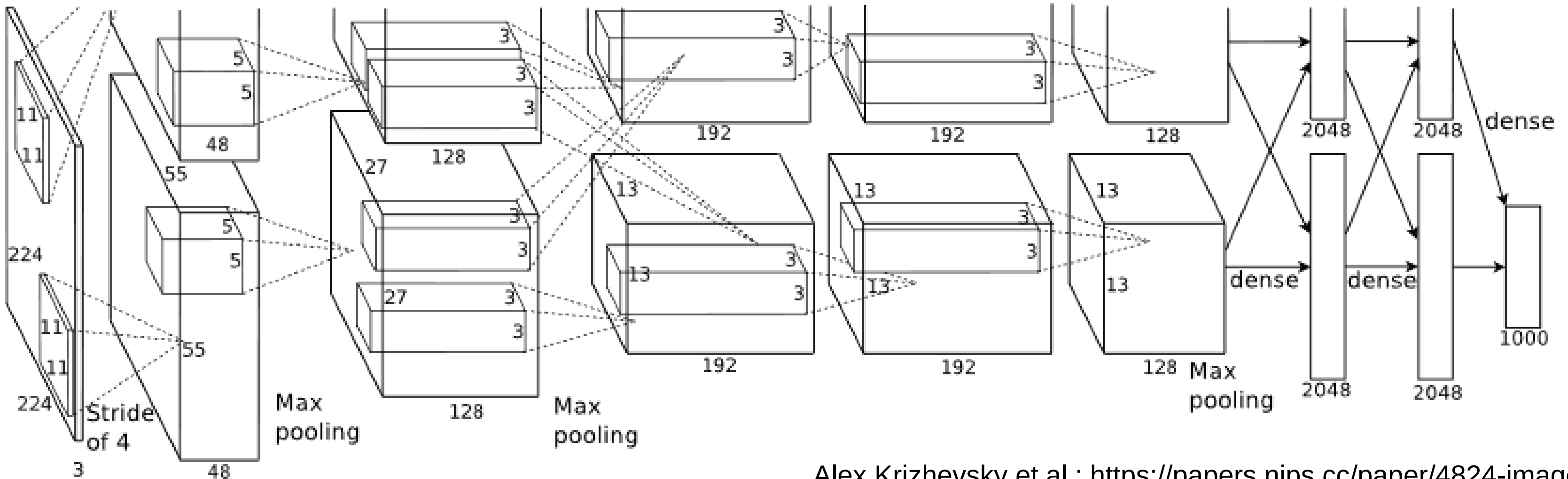


http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf

Optimization, Hyper-parameters, etc.

- Learning rate
- Activation function
- Weight initialization
- Batch normalization
- Dropout
- Data augmentation

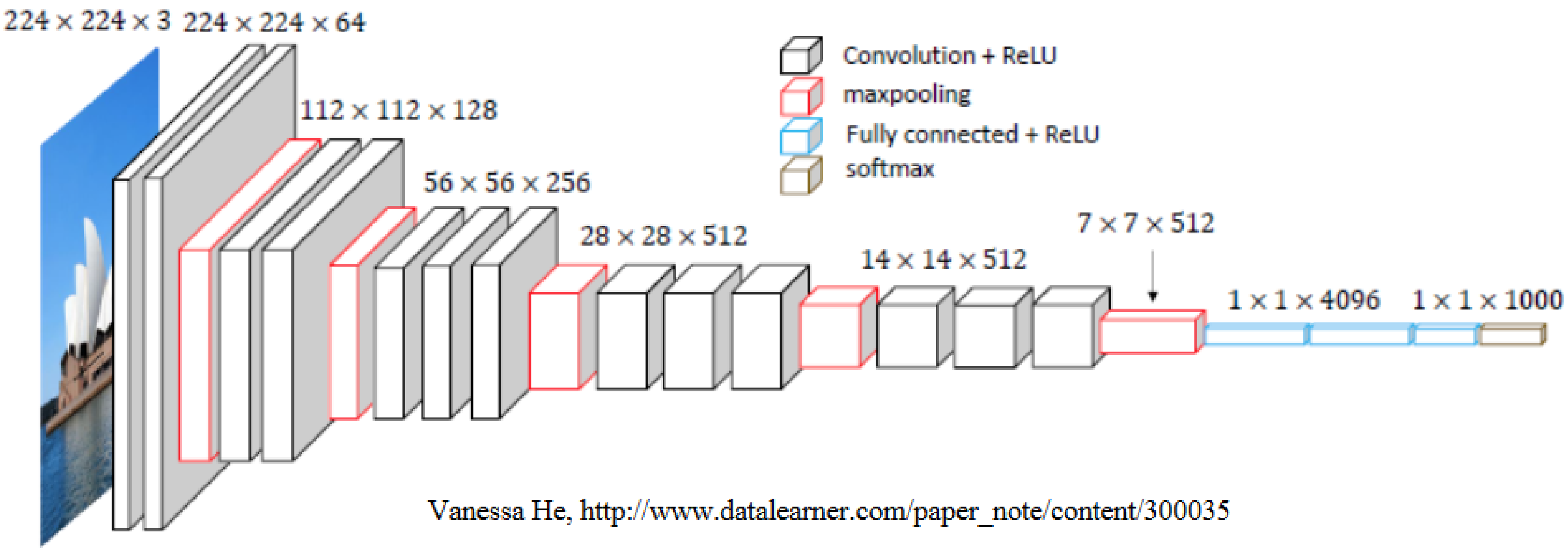
AlexNet



Alex Krizhevsky et al.: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

- First deep convolutional neural net for ImageNet
- No of parameters: 60 million
- Training time: 6 days (2 GPUs)
- 11x11, 5x5, 3x3 convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum

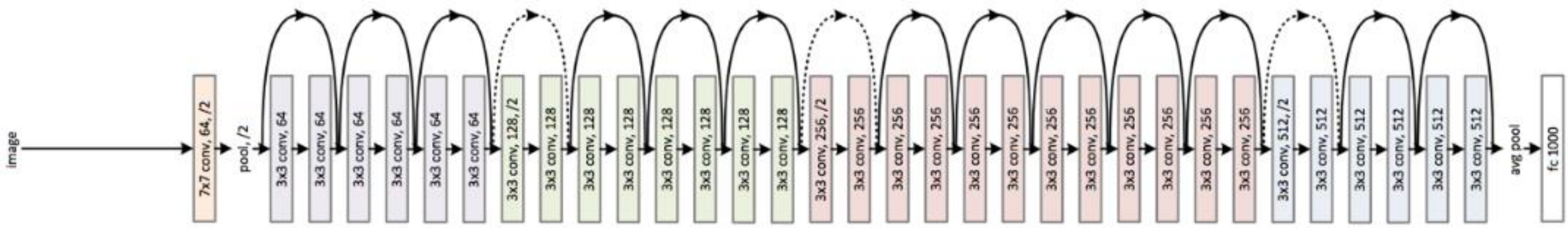
VGG



Vanessa He, http://www.datalearner.com/paper_note/content/300035

- 138 million parameters
- Training: 2-3 week (4 GPUs)

ResNet



Kaiming He, https://arxiv.org/pdf/1512.03385.pdf

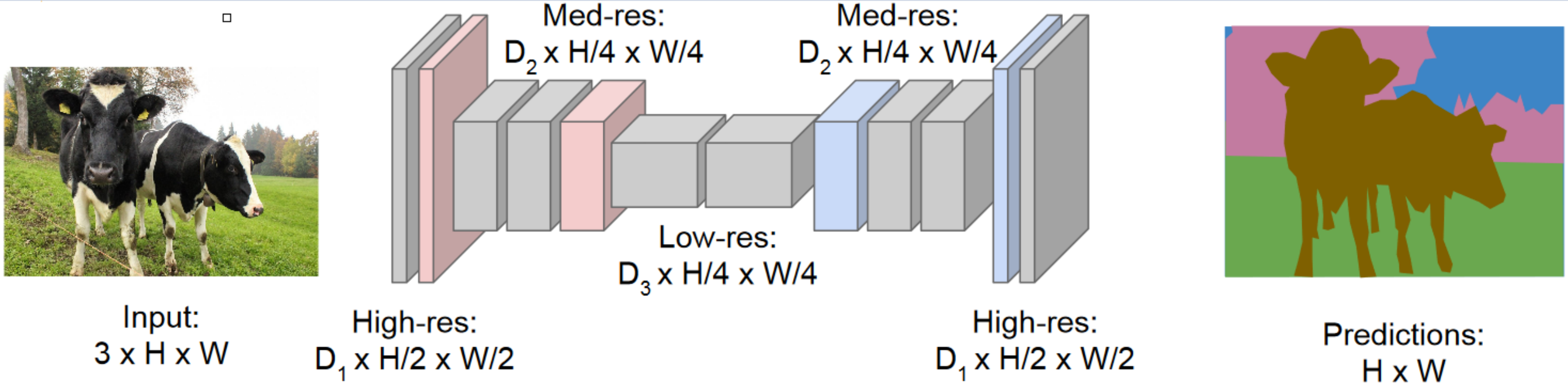
- 152 layers!!!
- 7x7 convolutional layers, 3x3 convolutional layers, batch normalization, max and average pooling.
- Parameters: 60 million
- Training time: 2-3 weeks (8 GPUs)

Transfer Learning

- CNNs as feature extractor
- Use number of layers based on the similarity of the data/task
- Other layers can be used for fine-tuning
- Different scenarios based on data size and similarity

Other tasks involving CNNs

- Image segmentation



Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015

- Object localization

Keras

with TensorFlow

Keras

A high-level library which can work over tensorflow

- Quick model development
- User friendliness
- Modularity
- Easy extensibility
- Work with Python

Keras Models

- Sequential: A linear stack of layers
- Model (Functional API): Any arbitrary setup

Keras Sequential

- A linear stack of layers

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

```
model = Sequential()
model.add(Dense(32, input_shape=(784,)))
```

Note!!! With tensorflow implementation of keras, we need to use tensorflow.keras instead of keras:
e.g., from tensorflow.keras.model import Sequential

model.add()

- Stacking layers in sequence

```
from keras.layers import Dense

model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))
```

Core Layers:

- Dense: Fully connected layers
- Activation
- Dropout
- Flatten

Layers- Conv2D

- 2D convolution layer (e.g. spatial convolution over images).

Important Arguments

- Filters: No of filters (the dimensionality of the output space)
- kernel_size
- Strides
- Activation
- kernel_initializer

Layers-MaxPooling2D

- Max pooling operation for spatial data.

Important Arguments:

- pool_size
- strides

model.compile()

- Once the model structure is ready, configure its learning process with:

```
# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# For a binary classification problem
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# For a mean squared error regression problem
model.compile(optimizer='rmsprop',
              loss='mse')
```

- https://keras.io/optimizers/
- https://keras.io/losses/

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.9, nesterov=True))
```

model.fit()

Perform the actual training, it has following main arguments:

- Xs and Ys
- batch_size
- Epochs
- validation_split

```
# x_train and y_train are Numpy arrays --just like in the Scikit-Learn API.
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

model.evaluate()

- Evaluate the performance

```
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
```

Keras Models-Functional API

- The Keras functional API is the way to go for defining complex models:
  - Multi-output models
  - Models with shared layers

```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

References

- https://keras.io
- Introduction to Deep Learning, National Research University Higher School of Economics
- Fei-Fei Li Convolutional Neural Networks for Visual Recognition, Stanford University (http://cs231n.stanford.edu/)
- Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015
- Andrew Ng, Neural Networks and Deep Learning, Stanford University
- http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf
- Andrew Ng, Machine Learning Yearning, deeplearning.ai