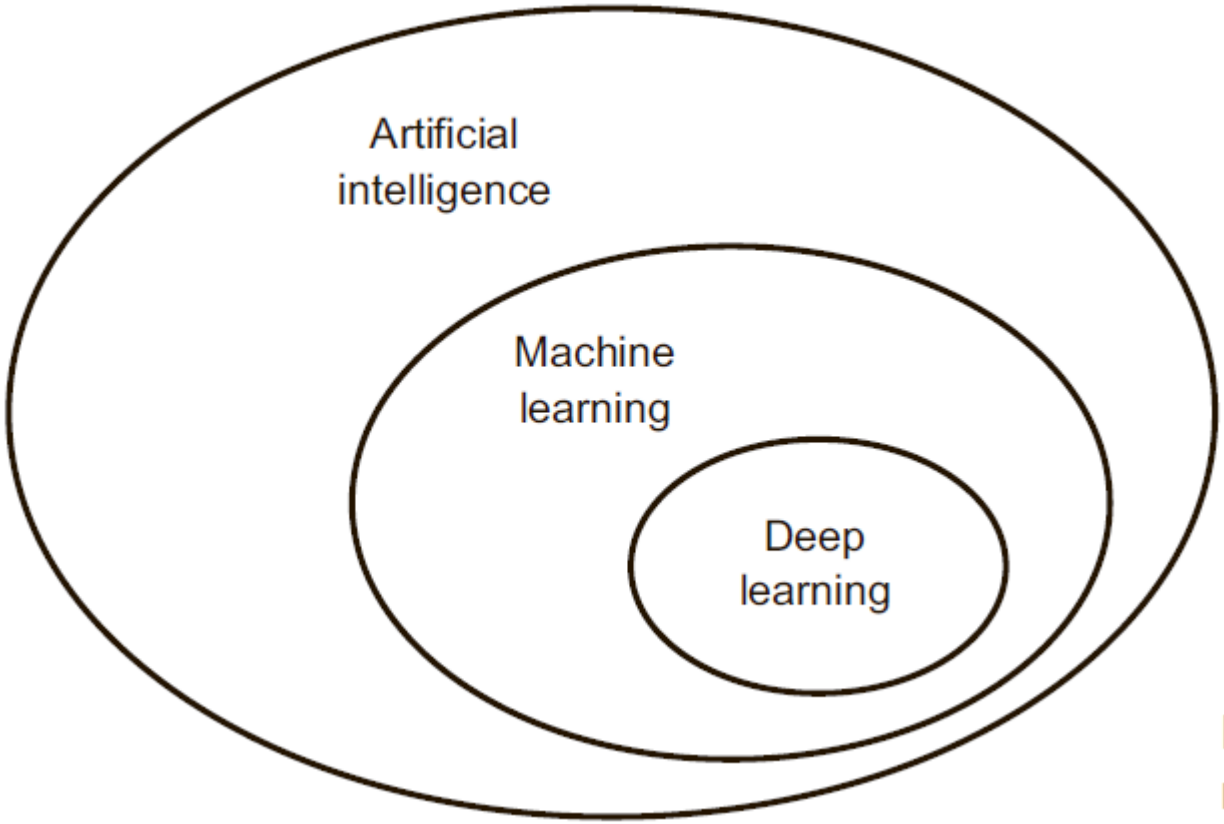
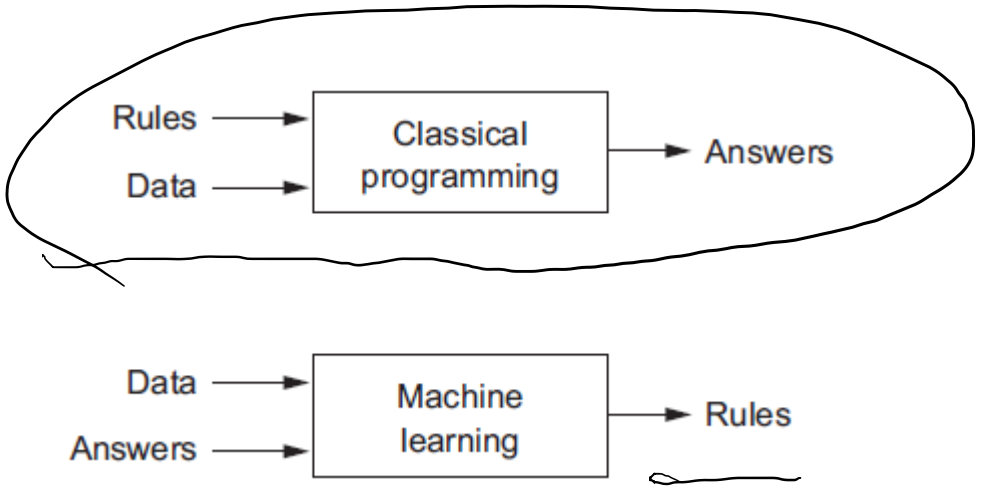
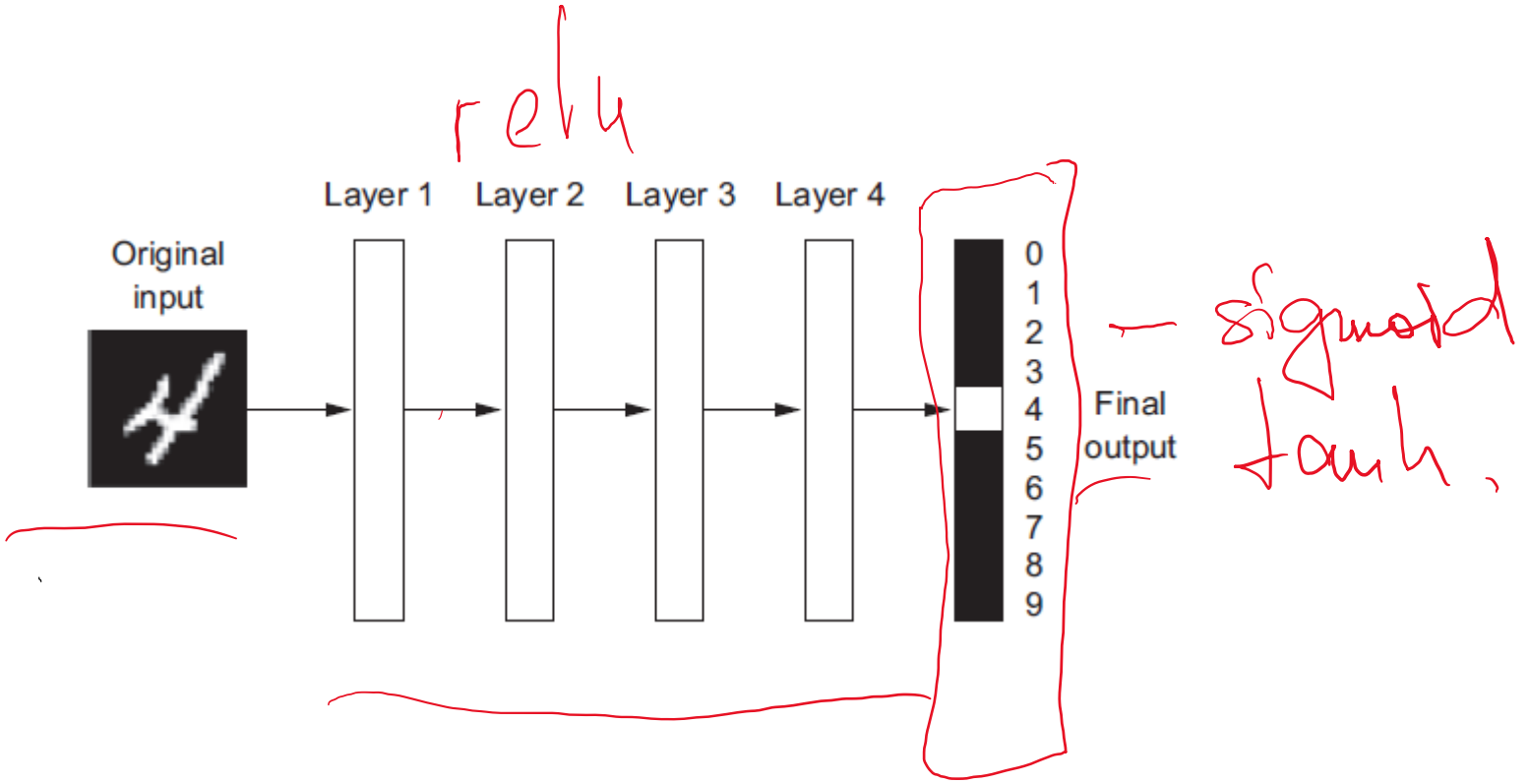
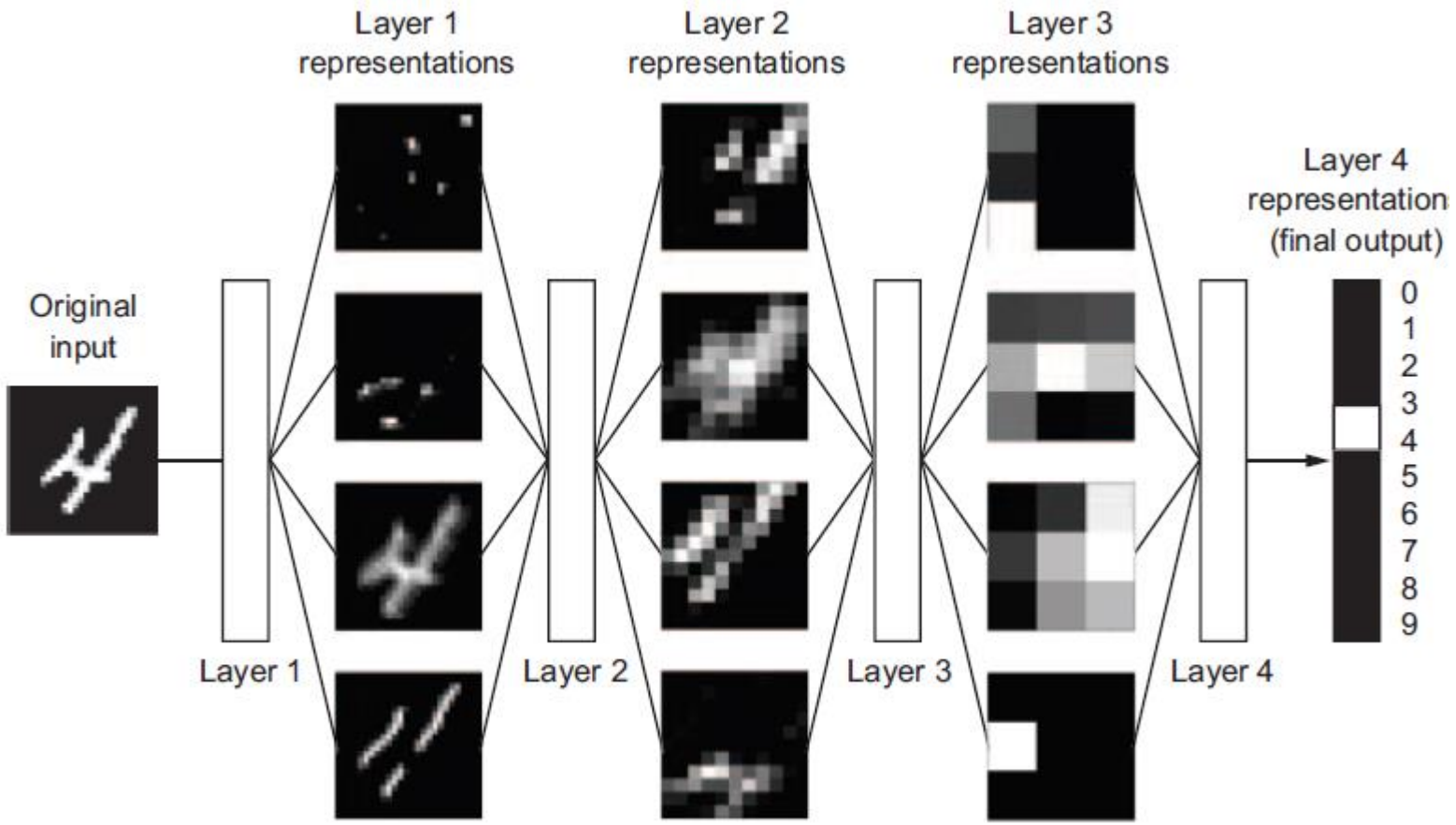
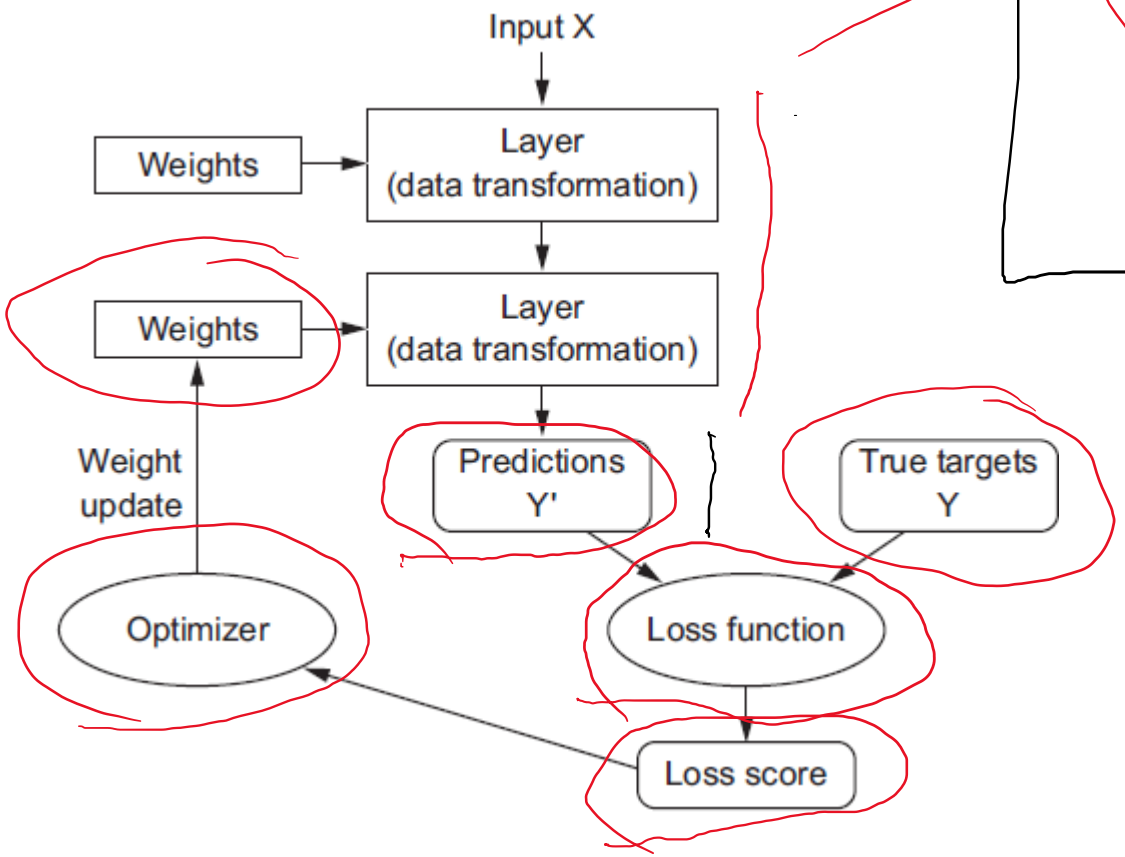
Keras
and basics of
Deep Learning


Artificial
intelligence
Machine
learning
Deep
learning


Rules
Data
Classical
programming
Answers
Data
Answers
Machine
learning
Rules


Layer 1
Layer 2
Layer 3
Layer 4
Original input
0
1
2
3
4
5
6
7
8
9
Final output


Layer 1 representations
Layer 2 representations
Layer 3 representations
Layer 4 representation (final output)
Original input
Layer 1
Layer 2
Layer 3
Layer 4
0
1
2
3
4
5
6
7
8
9

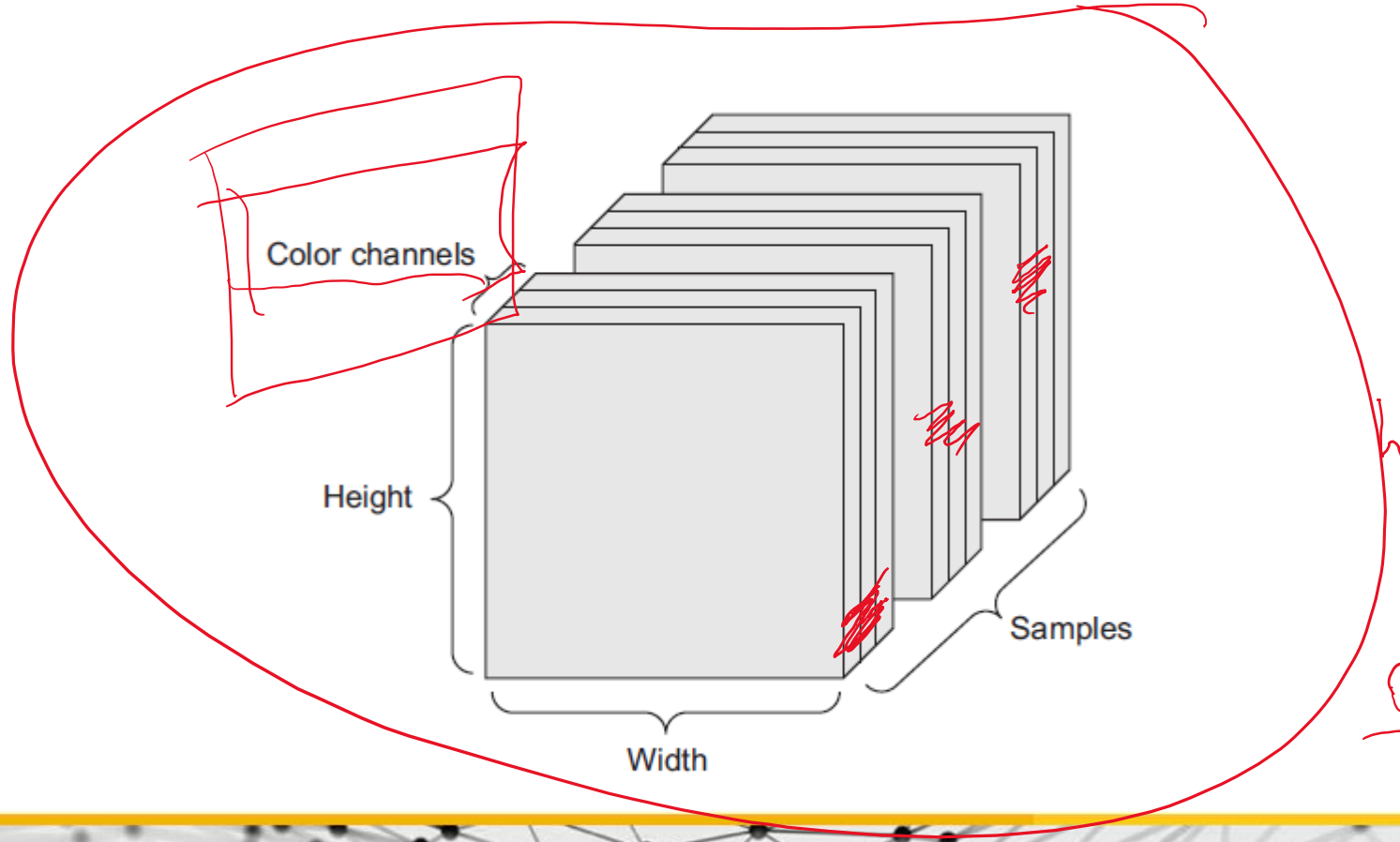

Input X
Weights
Layer
(data transformation)
Weights
Layer
(data transformation)
Weight
update
Predictions
Y'
True targets
Y
Optimizer
Loss function
Loss score


goal for
training

Data representation as tensors

Tensor Flow

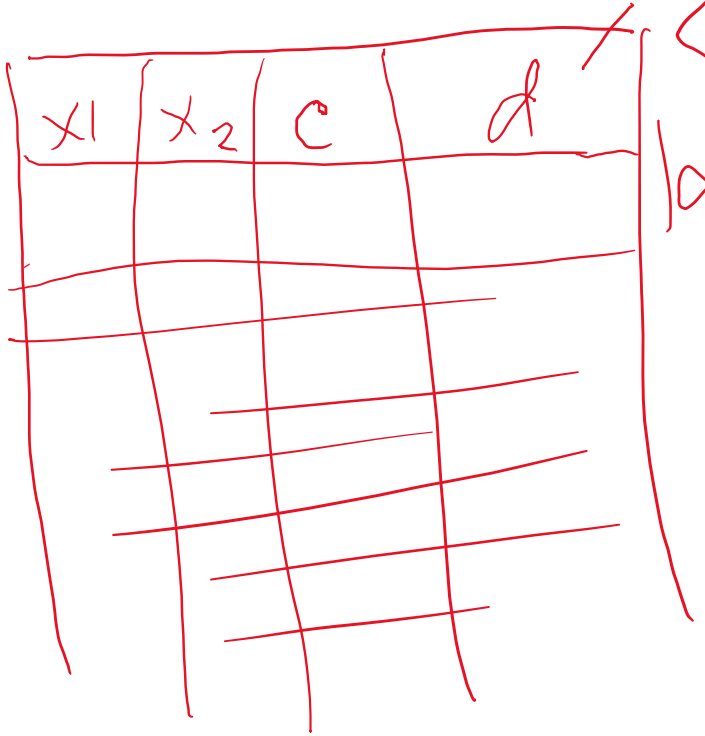
Image Data



1D - time

2D - V

MNIST

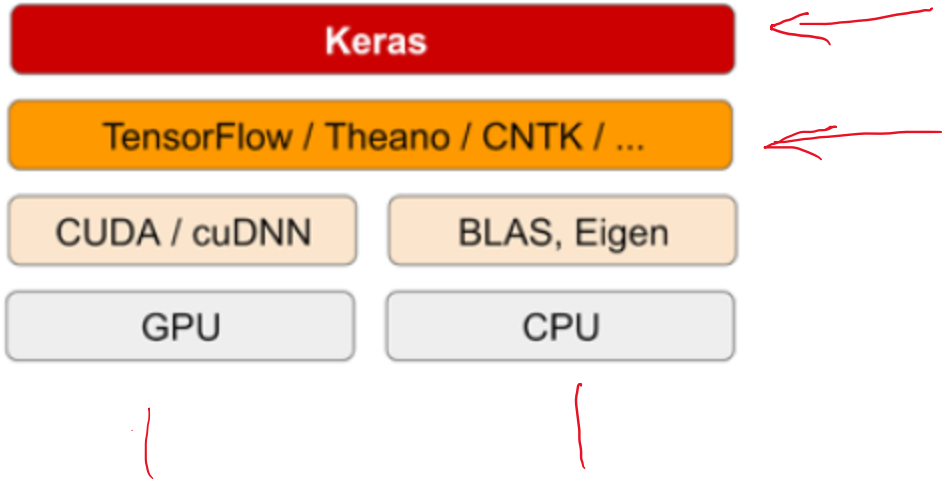


labels

x1 - width
x2 - hight
c - value for color
d - answer - what digit?

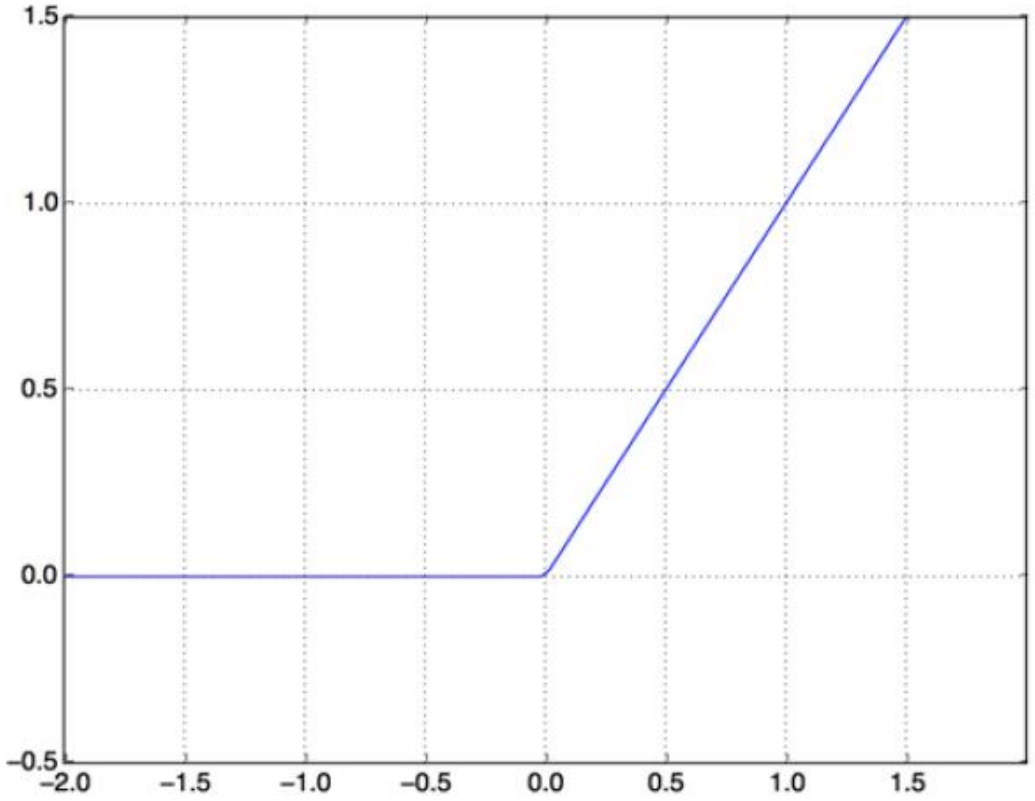
"0"
"7"
"5"
?

Keras Library



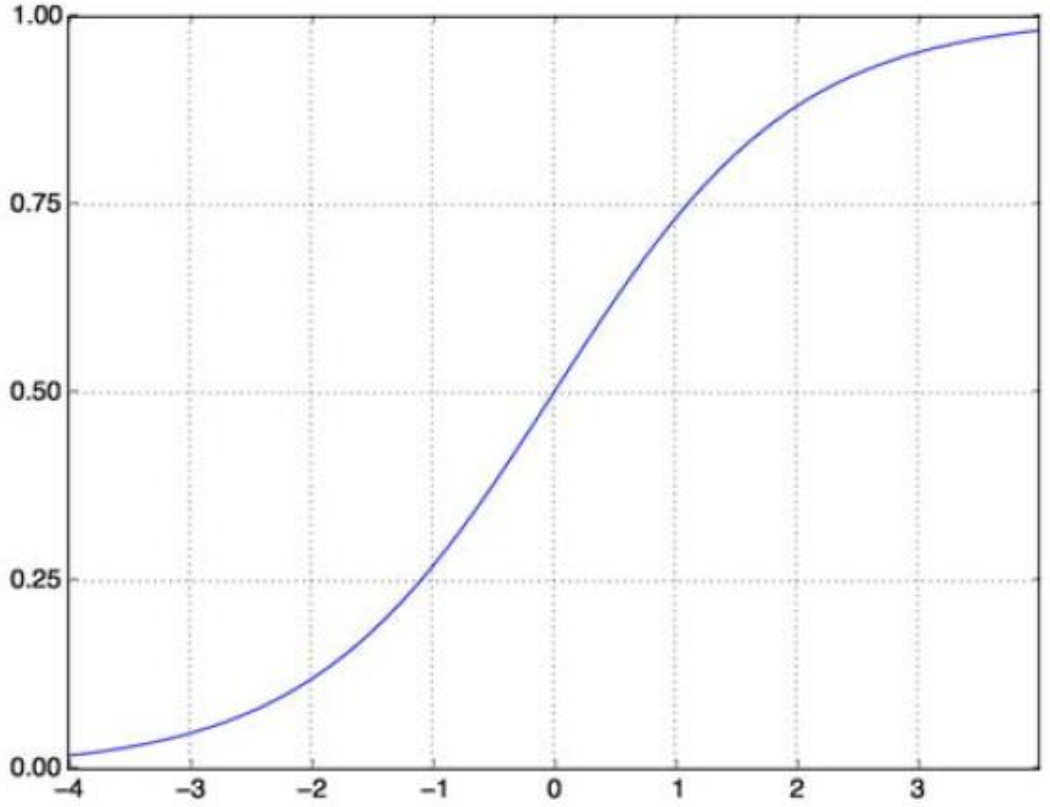
Activation functions

RELU

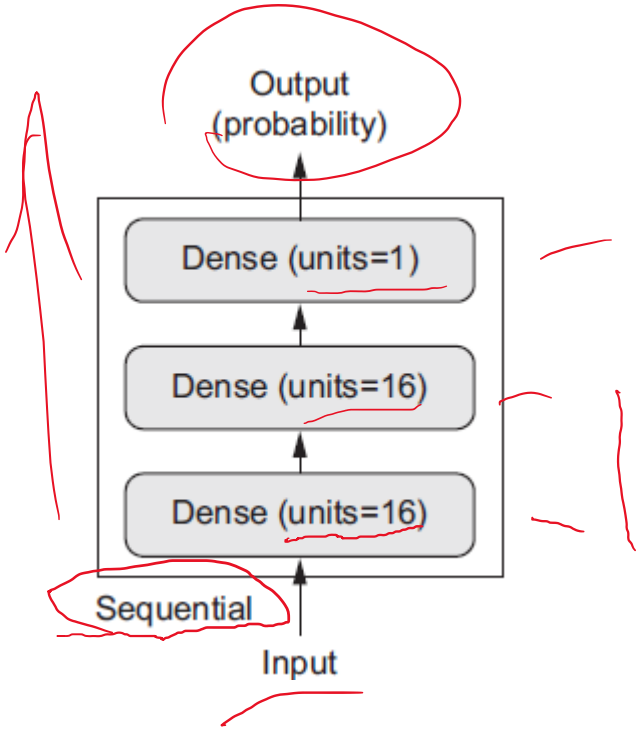


tanh

Sigmoid function



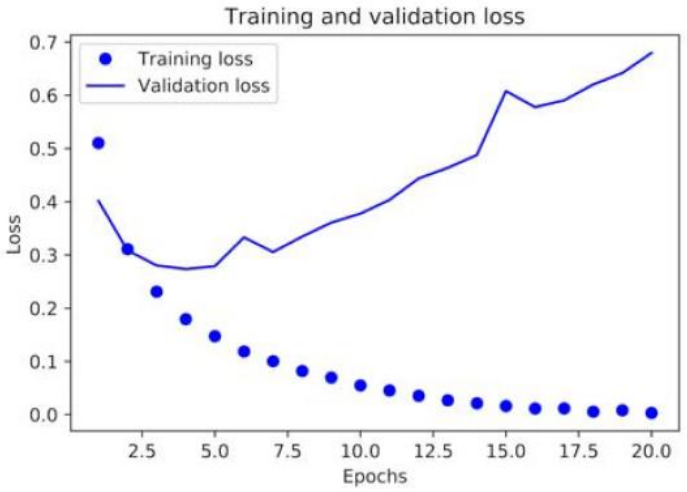
Example. Three-layer network




```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Example: movie reviews

Plotting training and validation loss

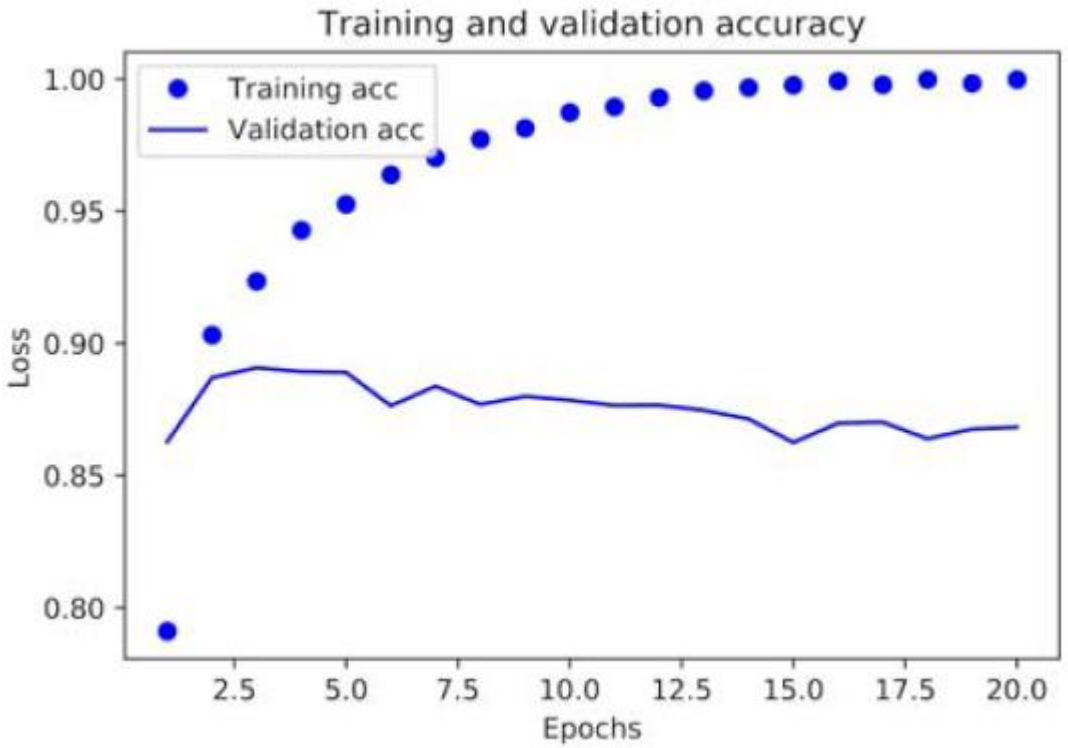



```
import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

"bo" is for "blue dot."

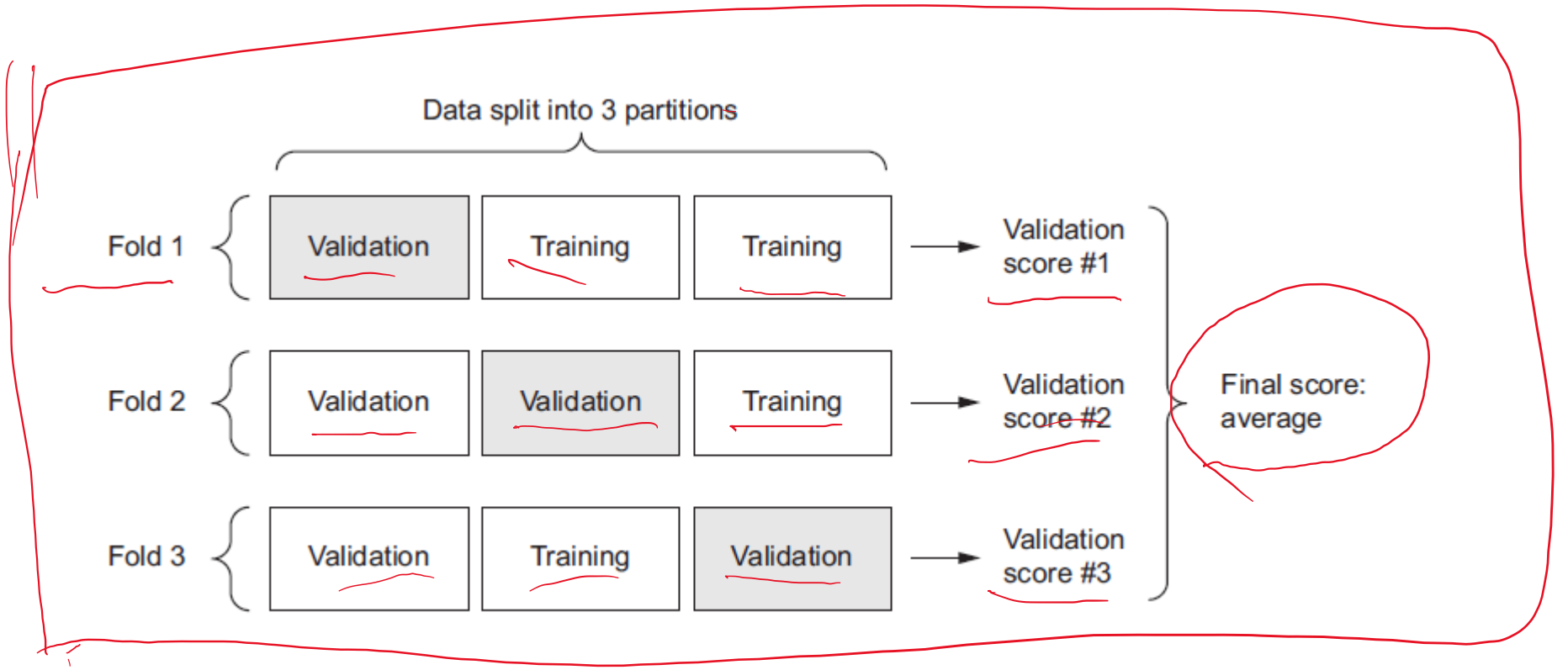
"b" is for "solid blue line."

Training and validation accuracy

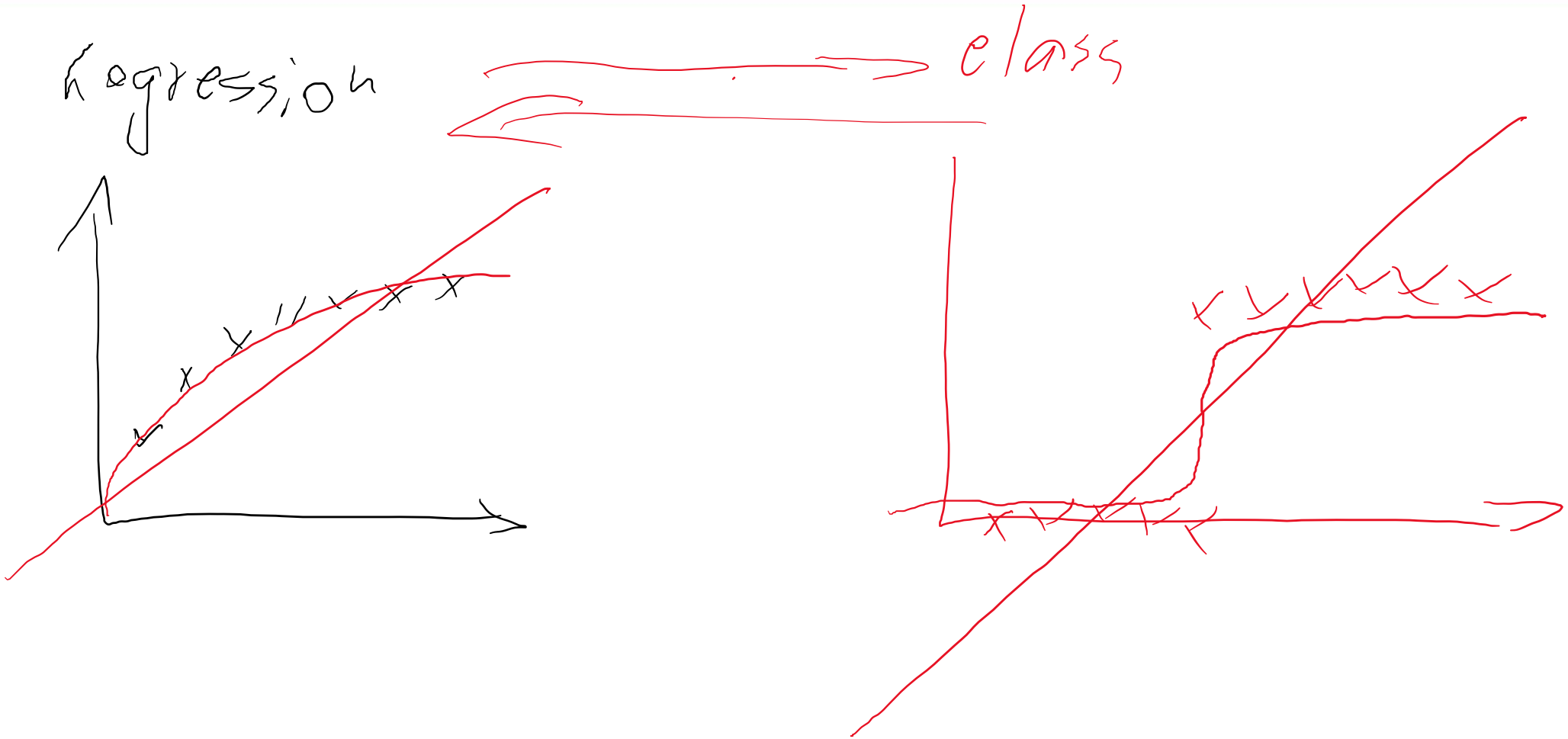



```
plt.clf()                    <-- Clears the figure
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

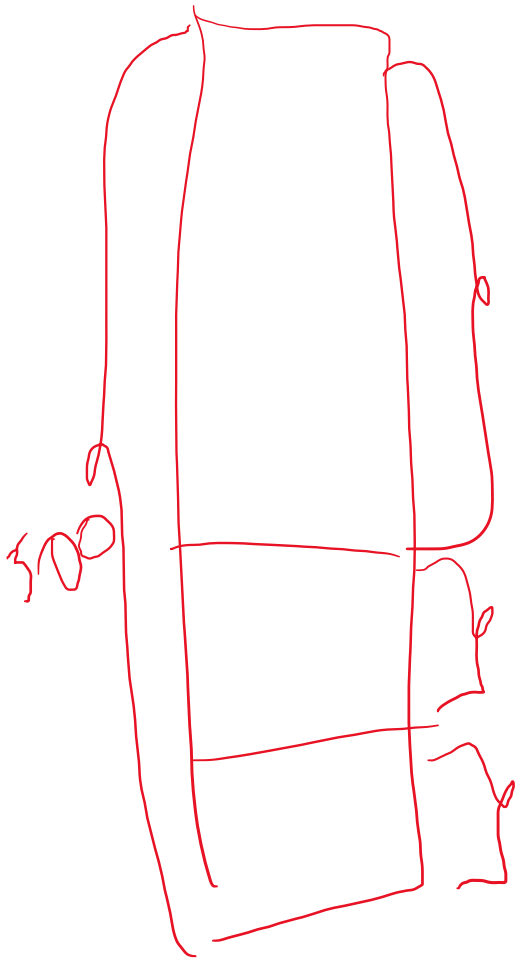
K-fold cross-validation



Example: regression problem


Regression
class

K-folding



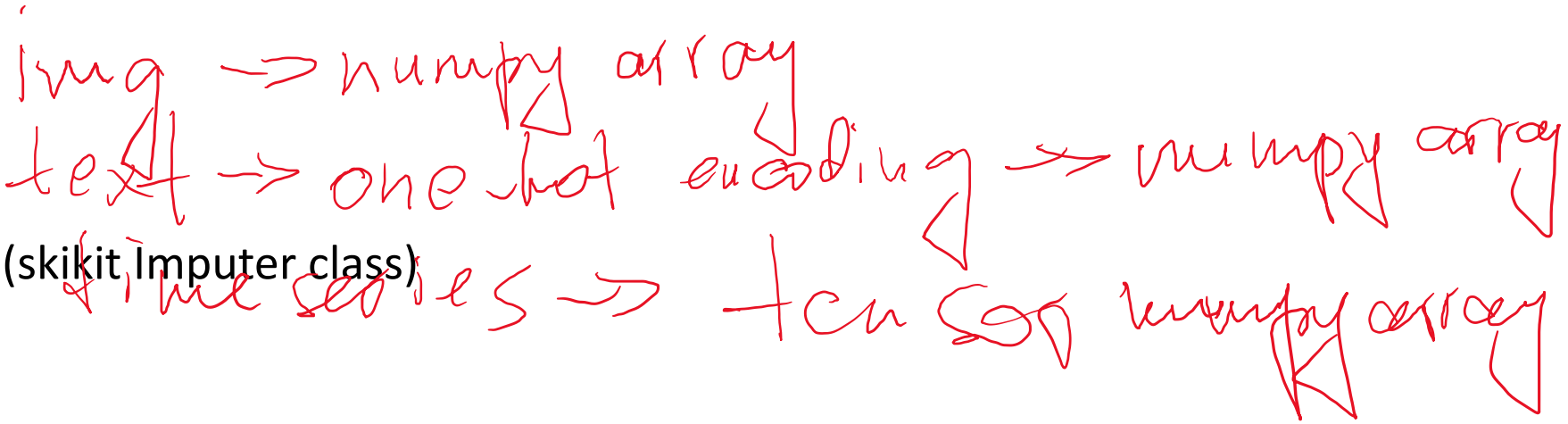
60% train 300

20% valid 100

20% test 100

Data preprocessing

- Vectorization
- Normalization
- Handling missing values (skikit Imputer class)
- Data augumentation



Data preprocessing. Vectorization

- All inputs and targets in a neural network must be tensors of floating-point data (or, in specific cases, tensors of integers). Whatever data you need to process—sound, images, text—you must first turn into tensors, a step called *data vectorization*.
- Example: one-hot-encoding, word2vec

Data preprocessing. Value normalization

- You have to normalize each feature independently so that it had a standard deviation of 1 and a mean of 0.
- In general, it isn't safe to feed into a neural network data that takes relatively large values (for example, multidigit integers, which are much larger than the initial values taken by the weights of a network) or data that is heterogeneous (for example, data where one feature is in the range 0–1 and another is in the range 100–200). Doing so can trigger large gradient updates that will prevent the network from converging. To make learning easier for your network, your data should have the following characteristics:
  - *Take small values*—Typically, most values should be in the 0–1 range.
  - *Be homogenous*—That is, all features should take values in roughly the same range.

```
x -= x.mean(axis=0)
x /= x.std(axis=0)
```

Assuming x is a 2D data matrix of shape (samples, features)

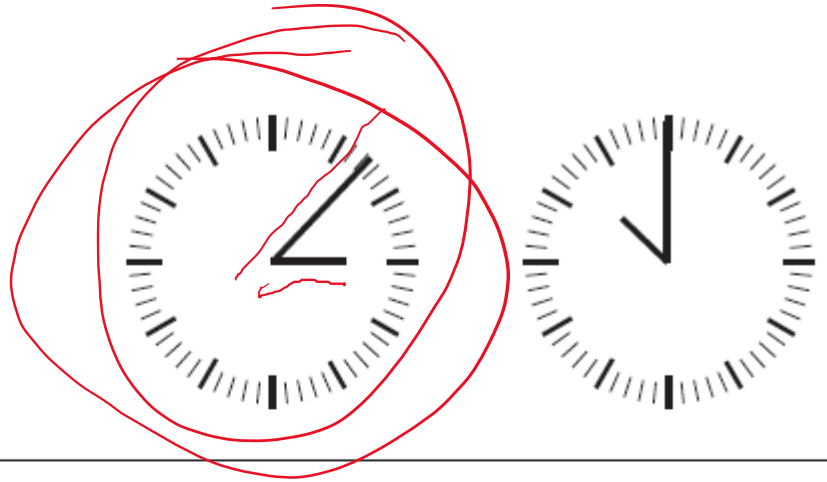
Data preprocessing. Missing values

- In general, with neural networks, it's safe to input missing values as 0, with the condition that 0 isn't already a meaningful value. The network will learn from exposure to the data that the value 0 means *missing data* and will start ignoring the value.

Note that if you're expecting missing values in the test data, but the network was trained on data without any missing values, the network won't have learned to ignore missing values! In this situation, you should artificially generate training samples with missing entries: copy some training samples several times, and drop some of the features that you expect are likely to be missing in the test data.

Feature engineering

- *Feature engineering* is the process of using your own knowledge about the data and about the machine-learning algorithm at hand to make the algorithm work better by applying hardcoded (nonlearned) transformations to the data before it goes into the model:
  - More elegant solution
  - Far less data

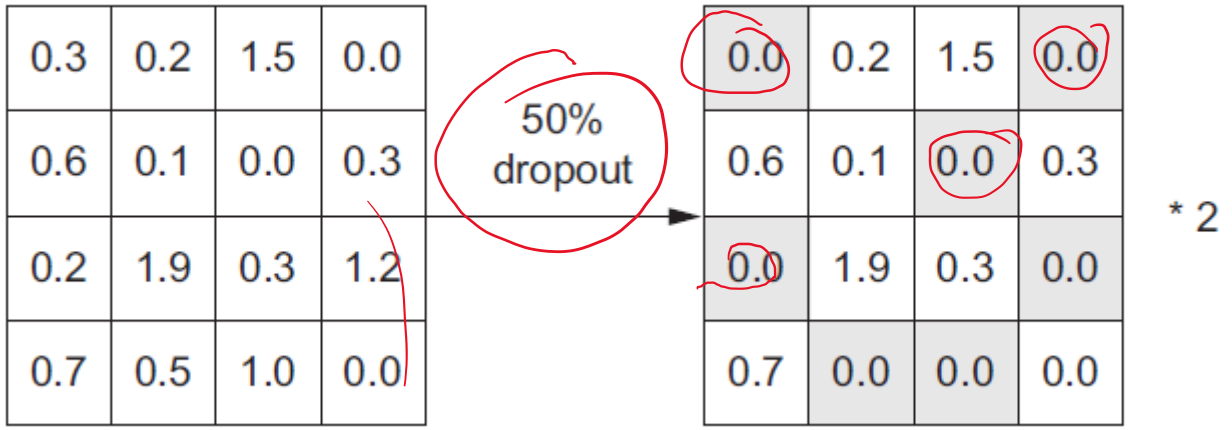
Raw data: pixel grid		
Better features: clock hands' coordinates	{x1: 0.7, y1: 0.7} {x2: 0.5, y2: 0.0}	{x1: 0.0, y2: 1.0} {x2: -0.38, 2: 0.32}
Even better features: angles of clock hands	theta1: 45 theta2: 0	theta1: 90 theta2: 140

Overfitting avoidance

- Take more data
- Reduce the network size
- Regularization (L1, L2)

```
from keras import regularizers

model = models.Sequential()
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                       activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                       activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

- Dropout



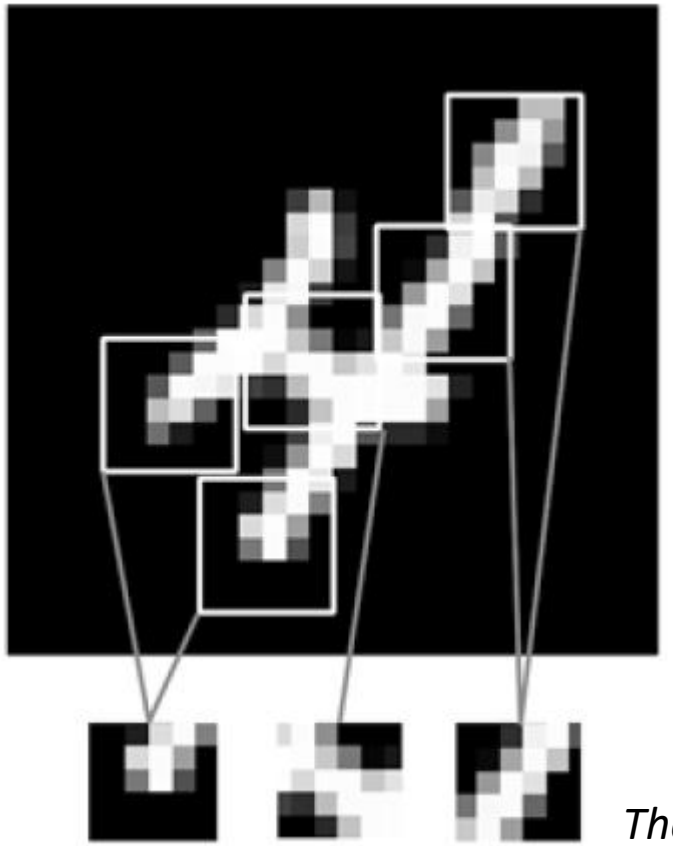
The universal workflow of machine learning

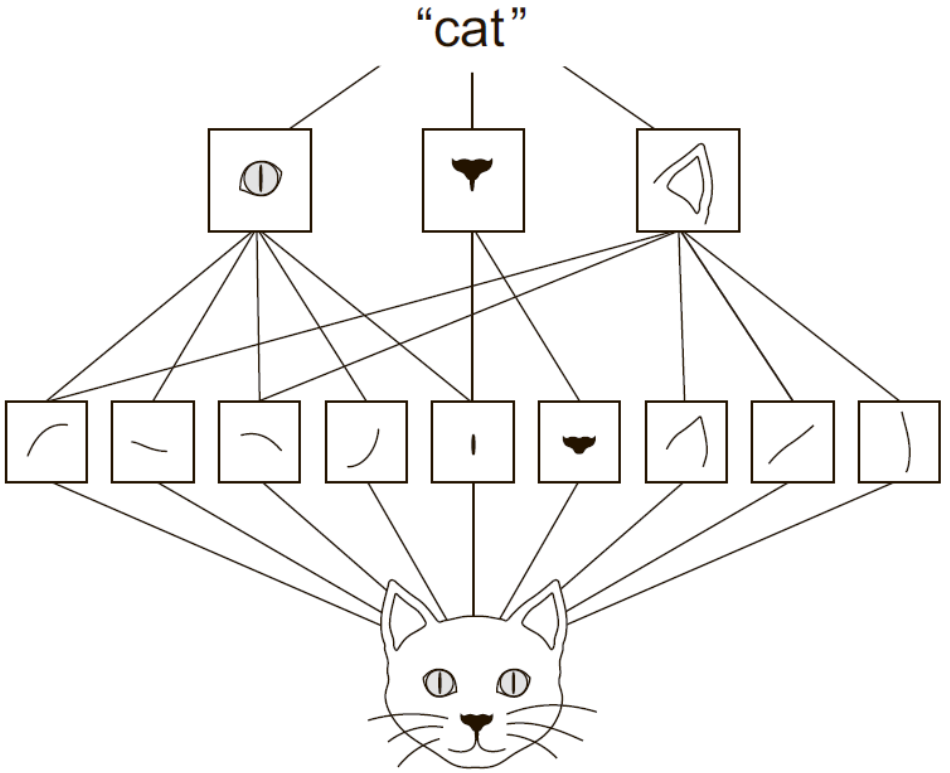
1. Defining the problem and assembling a dataset
2. Choosing a measure of success
3. Deciding on an evaluation protocol
   - *Maintaining a hold-out validation set*
   - *Doing K-fold cross-validation*
   - *Doing iterated K-fold validation*
4. Preparing your data
5. Developing a model that does better than a baseline
   - *Last-layer activation*
   - *Loss function*
   - *Optimization configuration*
6. Scaling up: developing a model that overfits (add layers, make layers bigger, train more epochs)
7. Regularizing your model and tuning your hyperparameters

Last-layer activation

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

Deep Learning. Convolution

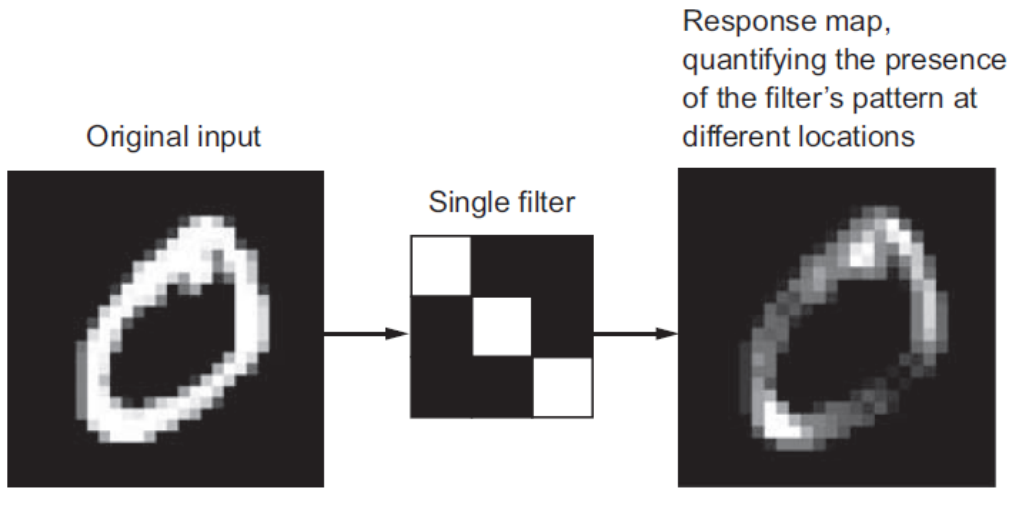




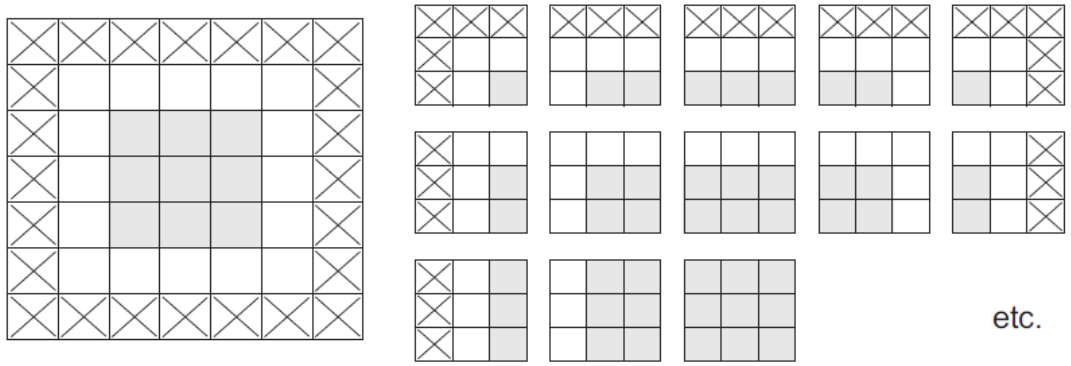
The patterns they learn are translation invariant.
They can learn spatial hierarchies of patterns

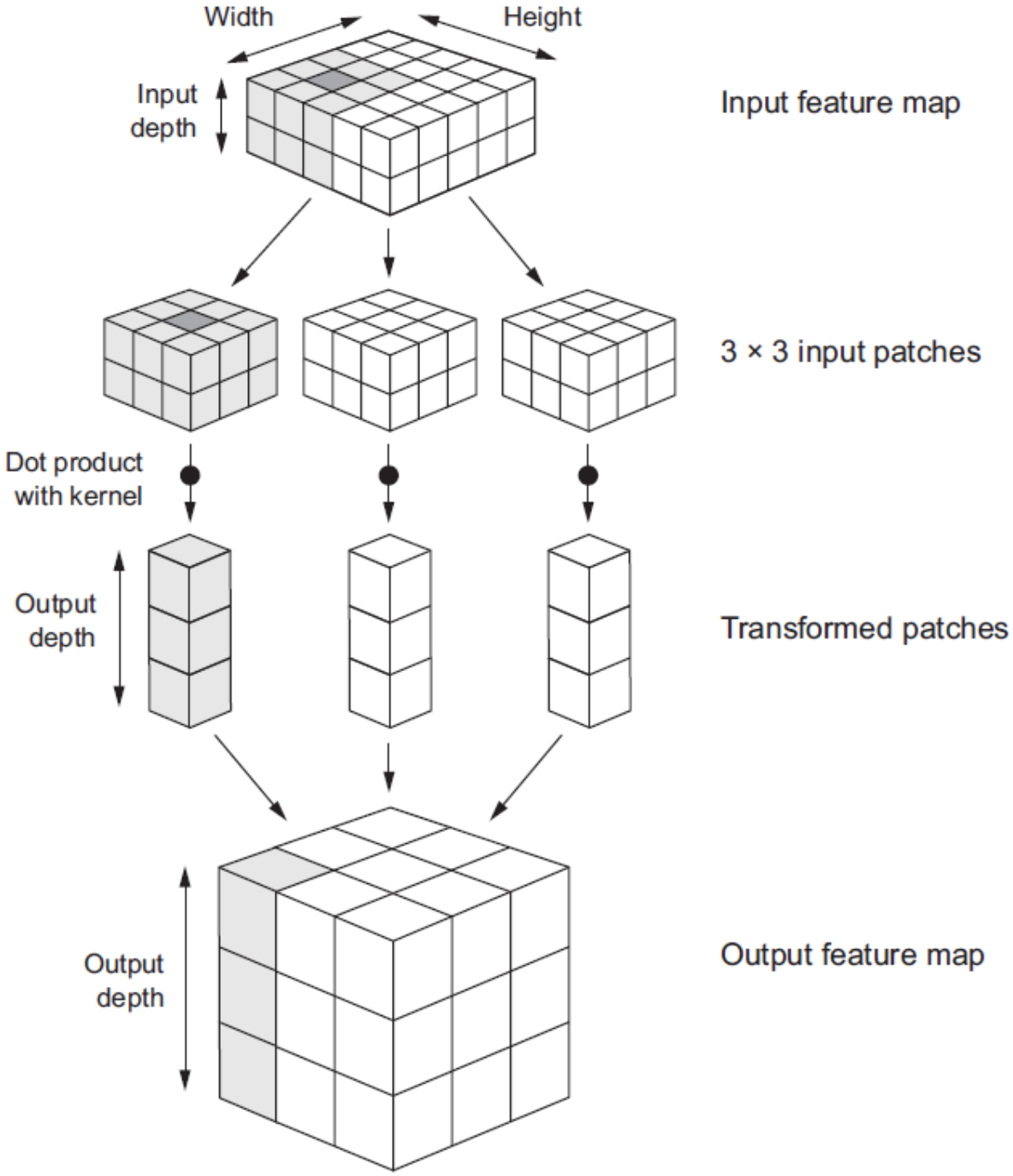
Convolution properties

- *Size of the patches extracted from the inputs*—These are typically 3 × 3 or 5 × 5. In the example, they were 3 × 3, which is a common choice.
- *Depth of the output feature map*—The number of filters computed by the convolution. The example started with a depth of 32 and ended with a depth of 64.

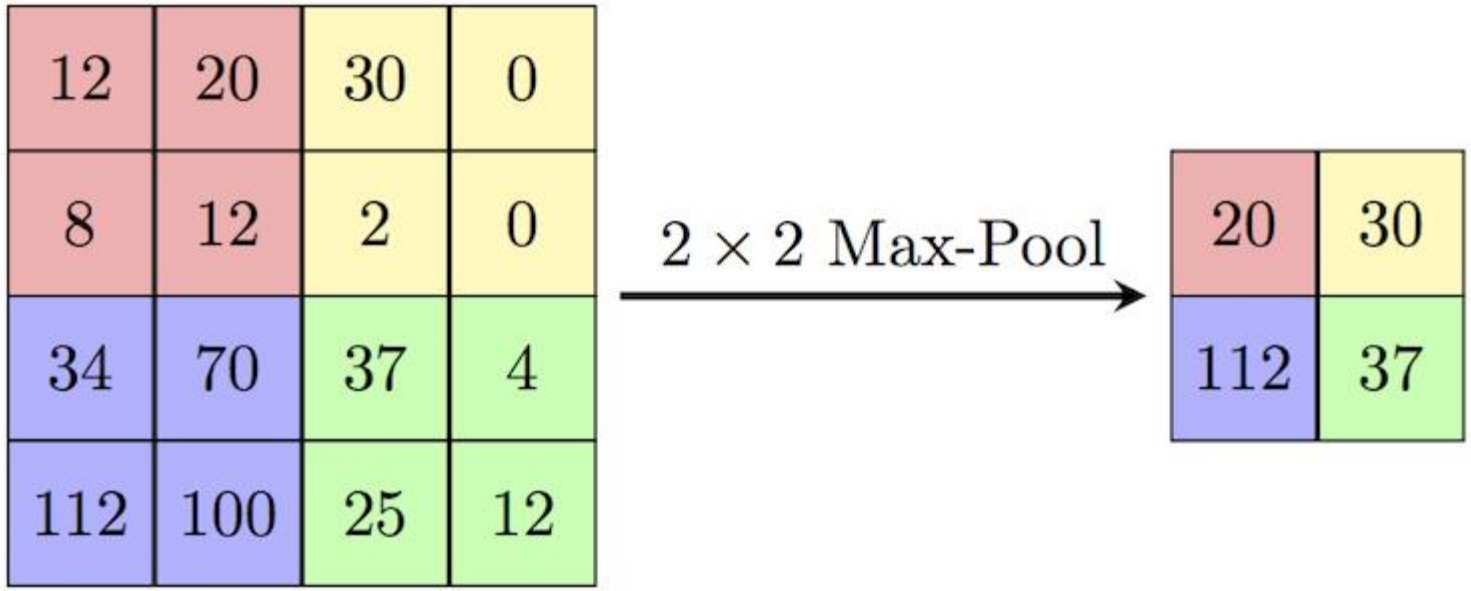


How it works?

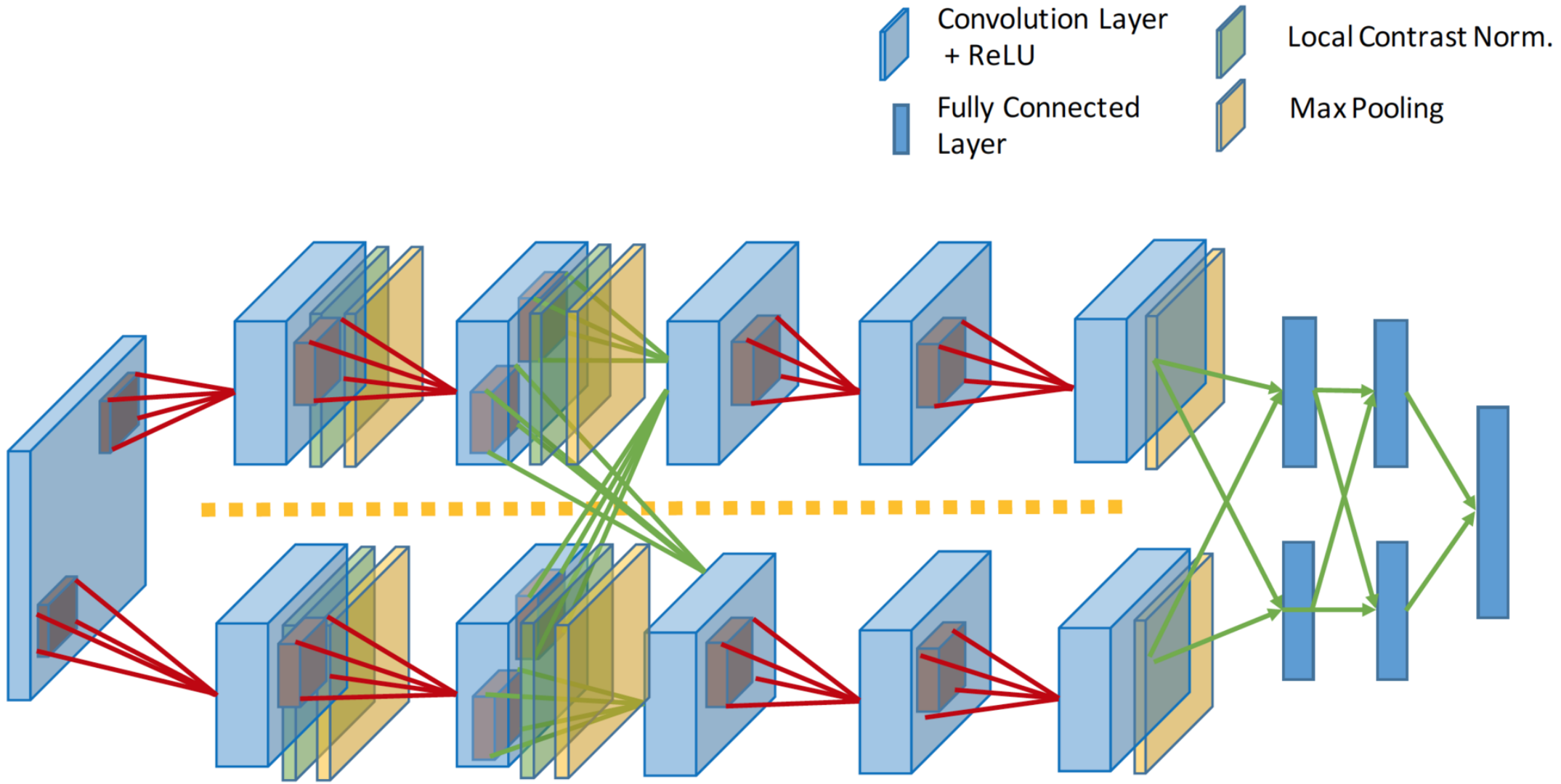

etc.


Width
Height
Input depth
Input feature map
3 × 3 input patches
Dot product with kernel
Output depth
Transformed patches
Output depth
Output feature map

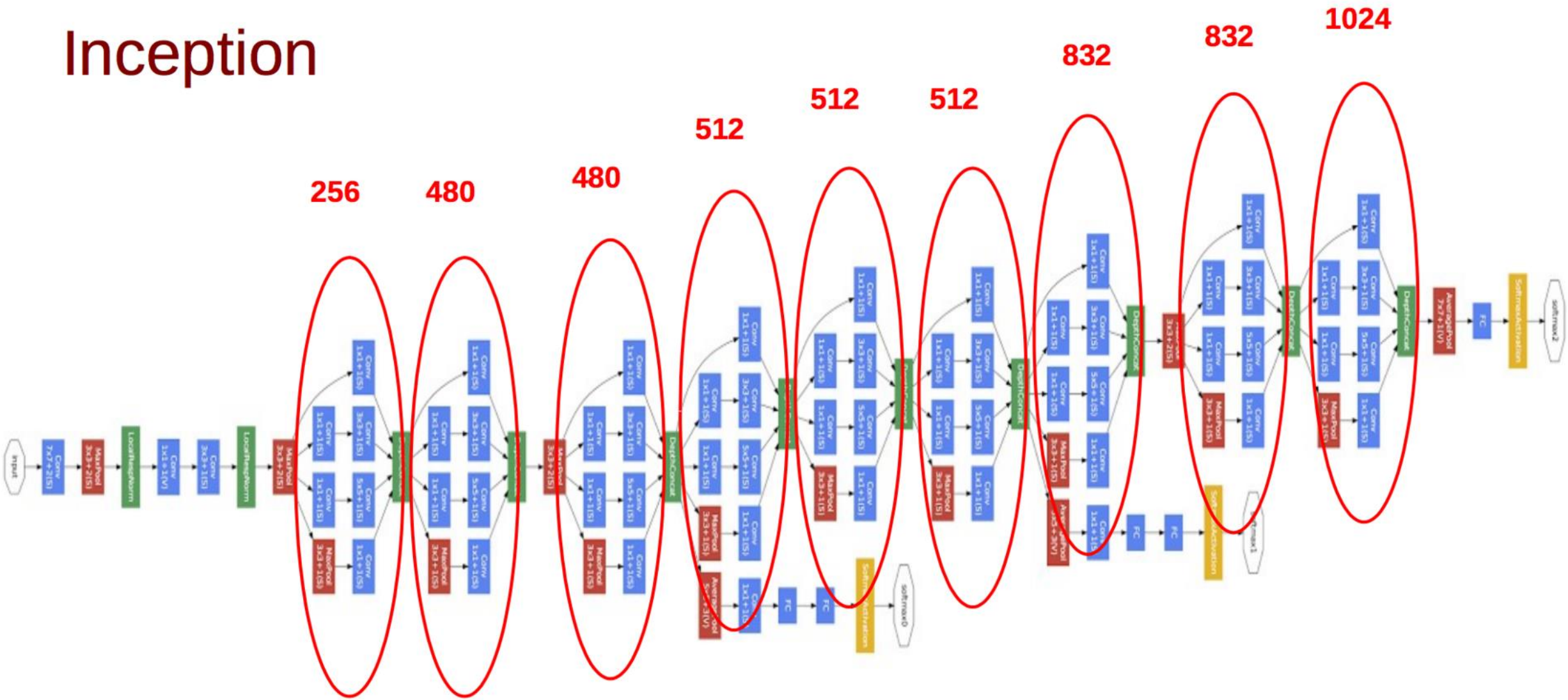
MaxPooling



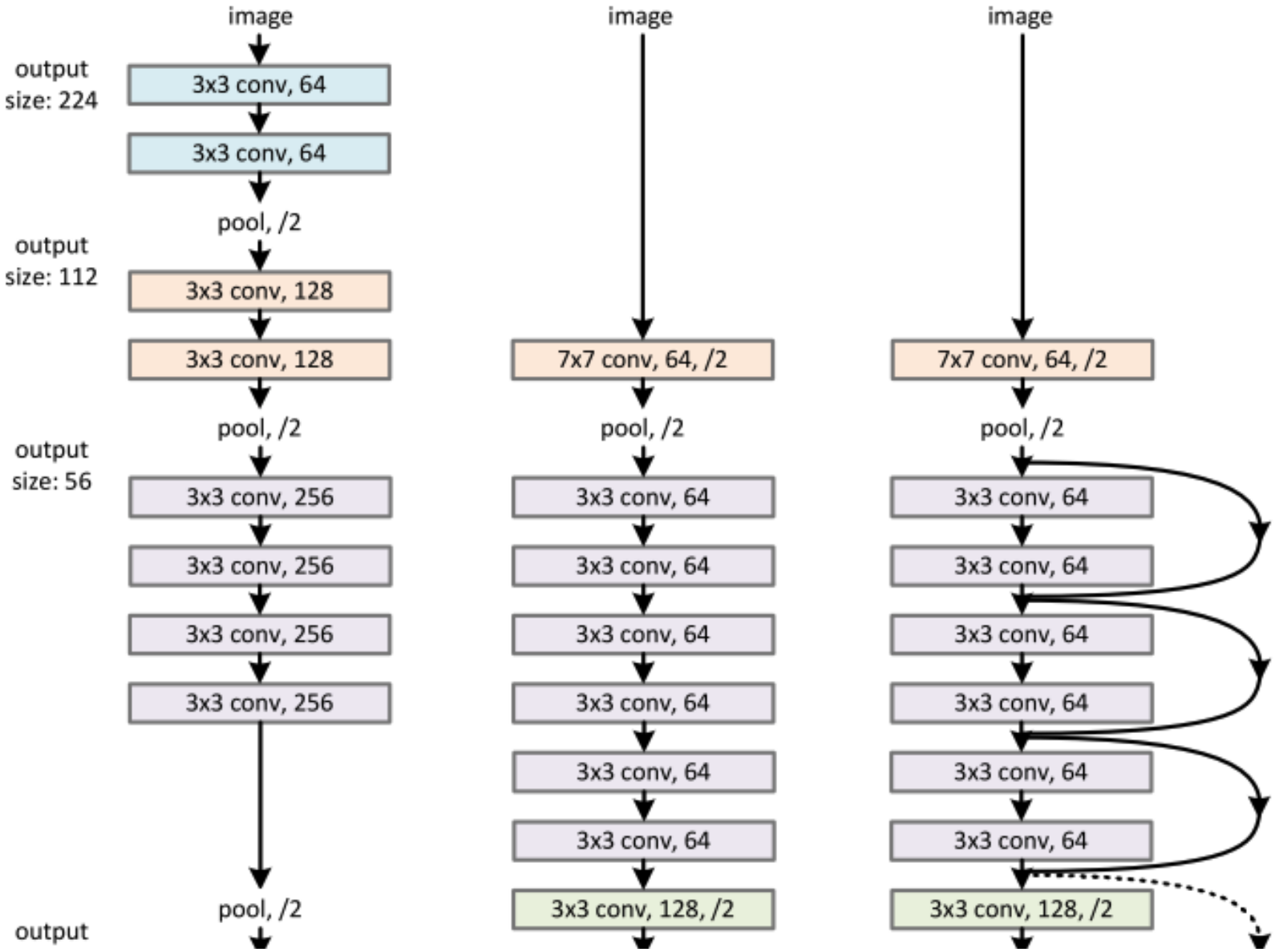
AlexNet (2012)



Inception от Google (17 Sep 2014)



ResNET (10 Dec 2015)



ResNET (10 Dec 2015)

Модель, с которой авторы победили на ImageNet, содержит меньше параметров, чем 19-слойный VGG, при глубине 152 слоя: