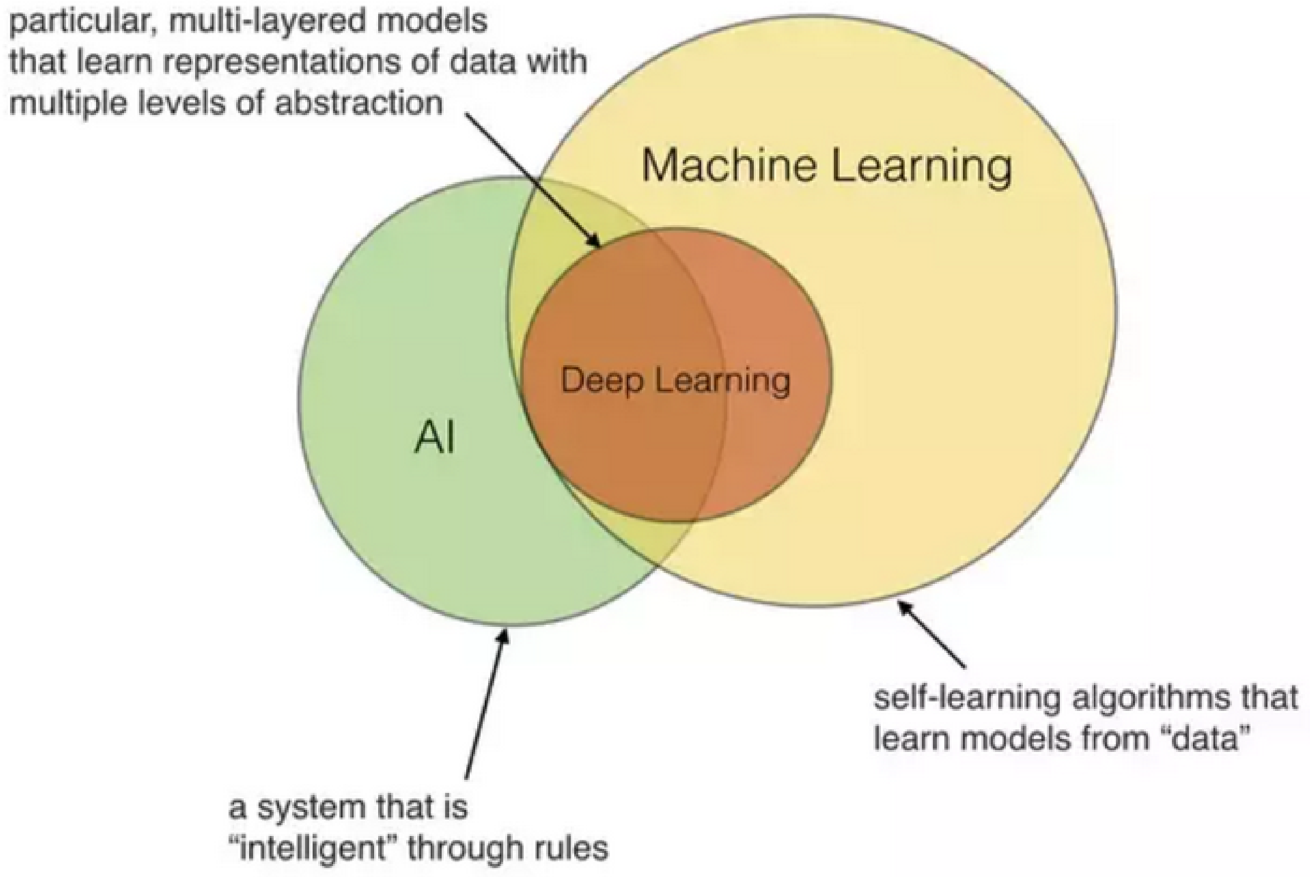
Computer Vision
Deep learning - lecture 7

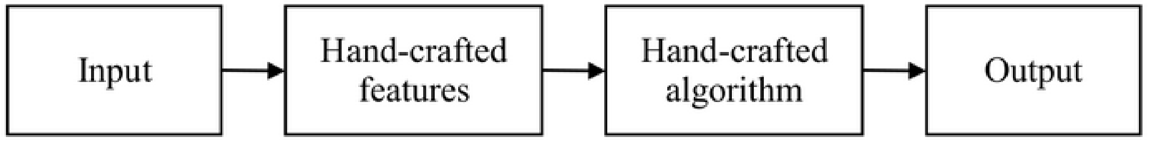
Adam Szmigielski
aszmigie@pjwstk.edu.pl

materials: $ftp(public)://aszmigie/WMAEnglish$

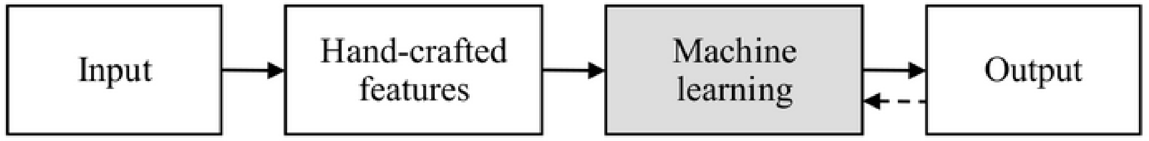
Deep learning


particular, multi-layered models
that learn representations of data with
multiple levels of abstraction
Machine Learning
Deep Learning
AI
self-learning algorithms that
learn models from "data"
a system that is
"intelligent" through rules

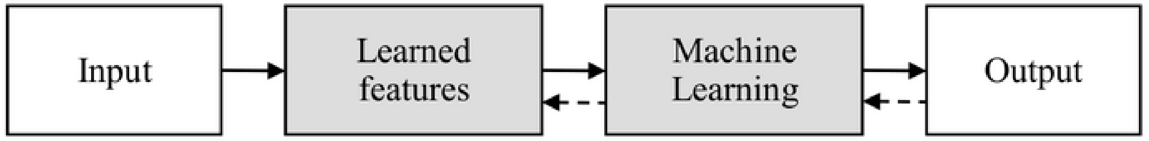
Deep learning in computer vision



(a) Traditional vision pipeline



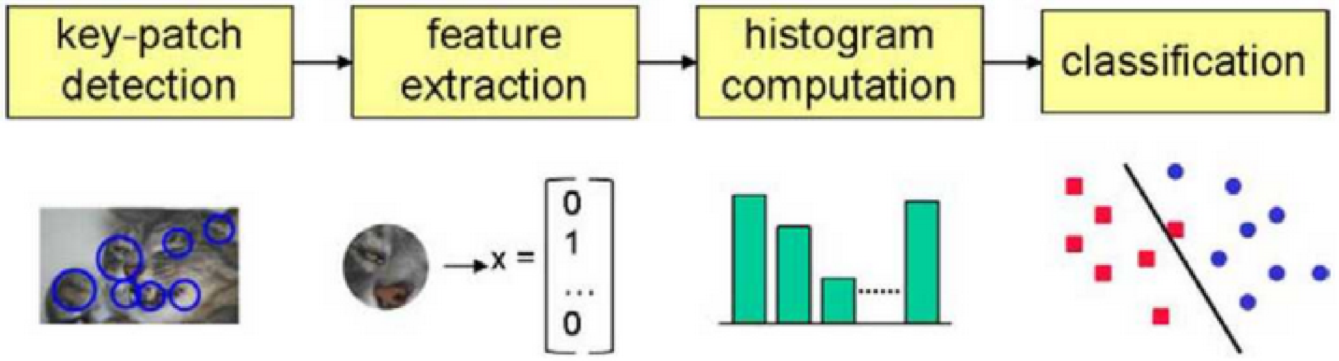
(b) Classic machine learning pipeline



(c) Deep learning pipeline

Currently, deep neural networks are the most popular and widely used machine learning models in computer vision, not just for semantic classification and segmentation, but even for lower-level tasks such as image enhancement, motion estimation, and depth recovery.

Typical processing pipeline



Learning of deep neural network using the standard TensorFlow interface

Learning consists of two phases:

- **Construction phase** - the number of inputs and outputs should be determined, as well as the number of neurons in each layer,
- **Executive phase** - opens the session and starts the init node initializing all variables. Then we enter the main loop - learning the neural network.

Construction phase

- Replacement nodes represent learning data (input) X and target (output) y - node X as $(None, n_i nputs)$, y one-dimensional tensor,

```
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int64, shape=(None), name="y")
```

- The network layer will be described by the input parameters, the number of neurons, the activation function and the name of the layer,

```
def neuron_layer(X, n_neurons, name, activation=None):
with tf.name_scope(name):
n_inputs = int(X.get_shape()[1])
stddev = 2 / np.sqrt(n_inputs)
init = tf.truncated_normal((n_inputs, n_neurons), stddev=stddev)
W = tf.Variable(init, name="jadro")
b = tf.Variable(tf.zeros([n_neurons]), name="obciazenie")
Z = tf.matmul(X, W) + b
if activation is not None:
return activation(Z)
else:
return Z
```

- We define the network constructor

```
with tf.name_scope("gsn"):
    hidden1 = tf.layers.dense(X, n_hidden1, name="ukryta1",
    activation=tf.nn.relu)
    hidden2 = tf.layers.dense(hidden1, n_hidden2, name="ukryta2",
    activation=tf.nn.relu)
    logits = tf.layers.dense(hidden2, n_outputs, name="wyjscia")
```

- Calculating the loss of information (by calculating *cross entropy*)

```
with tf.name_scope("strata"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,
    logits=logits)
    loss = tf.reduce_mean(xentropy, name="strata")
```

- Gradient calculation with the optimizer:

```
learning_rate = 0.01
with tf.name_scope("uczenie"):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    training_op = optimizer.minimize(loss)
```

- Assessment of the network's operation:

```
with tf.name_scope("ocena"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
```

- Network initialization node:

```
init = tf.global_variables_initializer()
saver = tf.train.Saver()
```

Executive phase

We import data, open the *TensorFlow* session and start the init node initializing all variables.

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/dane/")
n_epochs = 40
batch_size = 50

init = tf.global_variables_initializer()
with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        for iteration in range(mnist.train.num_examples // batch_size):
            X_batch, y_batch = mnist.train.next_batch(batch_size)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_test = accuracy.eval(feed_dict={X: mnist.validation.images,y:
        mnist.validation.labels})
        print(epoch, "Dokladnosc uczenia:", acc_train,
        "Dokladnosc testowania:", acc_test)
    save_path = saver.save(sess, "./moj_model_ostateczny.ckpt"
```

The network can be read and used again.

Tuning hyperparameters of the neural network

- **The number of hidden layers** - they can model complicated functions with an exponentially fewer number of neurons than in the case of “ shallow ” networks (significantly speeds up their learning),
- **Number of neurons forming the hidden layer** - depends on the input and output data required by a specific task (usually the subsequent layers have fewer neurons),
- **Activation functions** - In most cases, in the hidden layers the function ReLU is used, in the output layer the activation function depends on the purpose of the network (softmax for classification, linear for regression).

Deep network learning problems

- The problem of vanishing gradients (or closely related problem of exploding gradients), which concerns deep networks (significantly hinders learning of lower layers of the network),
- Learning an extensive network can be very time-consuming.
- A model containing millions of parameters is significantly exposed to overtraining.

Problems of disappearing / exploding gradients

- The back propagation algorithm runs from the output layer to the input layer and distributes the gradient along the way.
- Gradient values often decrease along with the algorithm's progress to the lower layers of the network - the vanishing gradients problem.
- Sometimes we can find the opposite phenomenon: gradients are constantly increasing, which in many layers of the scale are updated rapidly - the problem of exploding gradients,
- Neural networks show unstable gradients syndrome; individual layers can learn at radically different speeds.

Xavier and He weights initiation

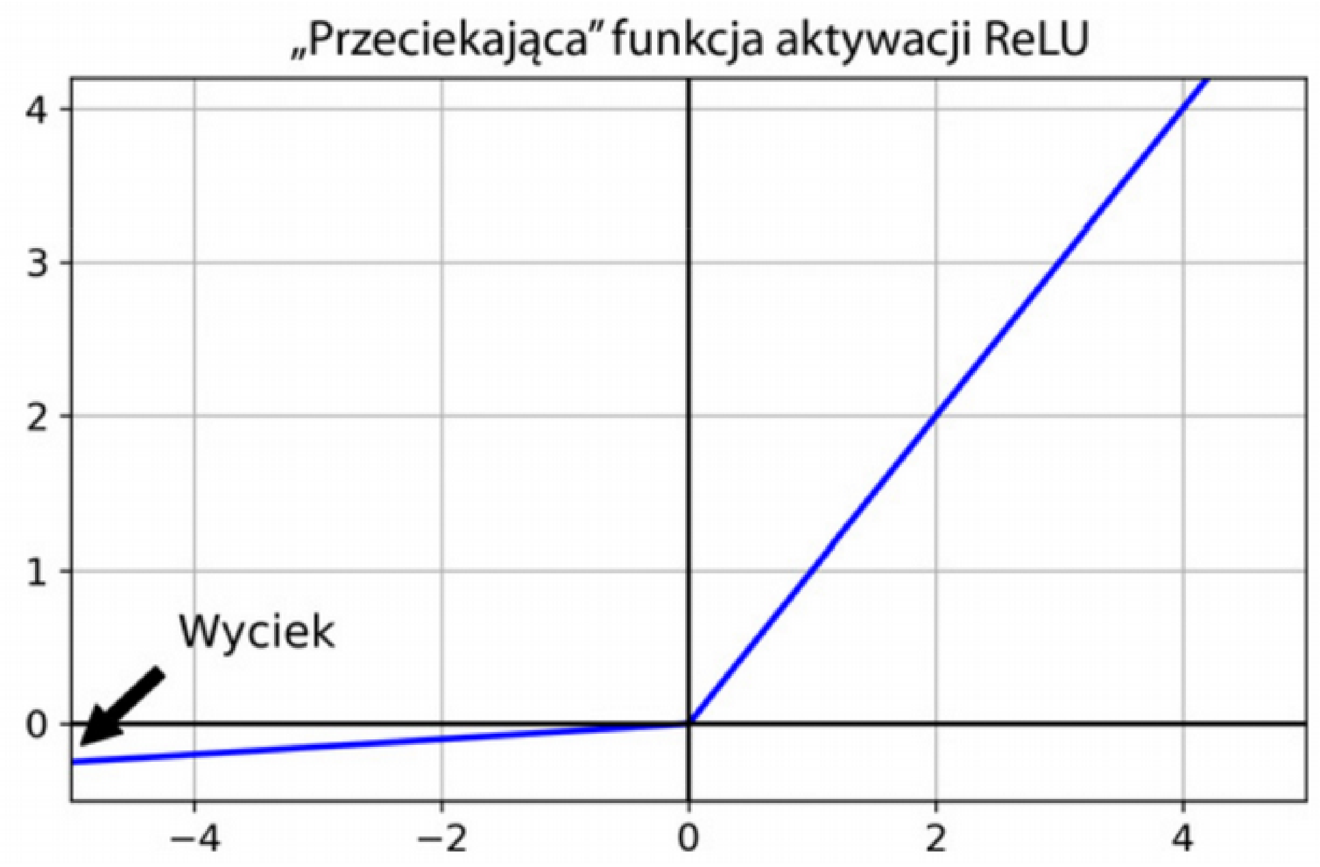
Activation function	Uniform distribution <-r, r>	Normal distribution
logistic	$r = \sqrt{\frac{6}{n_{inputs}+n_{outputs}}}$	$\sigma = \sqrt{\frac{6}{n_{inputs}+n_{outputs}}}$
tang. hiperbolic	$r = 4\sqrt{\frac{6}{n_{inputs}+n_{outputs}}}$	$\sigma = 4\sqrt{\frac{6}{n_{inputs}+n_{outputs}}}$
RELU type	$r = \sqrt{2}\sqrt{\frac{6}{n_{inputs}+n_{outputs}}}$	$\sigma = \sqrt{2}\sqrt{\frac{6}{n_{inputs}+n_{outputs}}}$

- In order to properly signal the direction of information flow, the variance of outputs in a given layer must be equal to the variance of its inputs,
- Gradients must have the same variance before and after passing through the layer in the opposite direction.

Nonsaturation activation functions

- In the case of deep neural networks, the ReLU function performs better than sigmoidal, because it does not saturate for positive values,
- The ReLU function is not perfect - dying ReLUs. During learning, some neurons permanently “get lost”, i.e. the only signal they send is 0,
- It is difficult to re-enter the neuron to the network - the gradient of the ReLU function is 0 with negative values on the input.

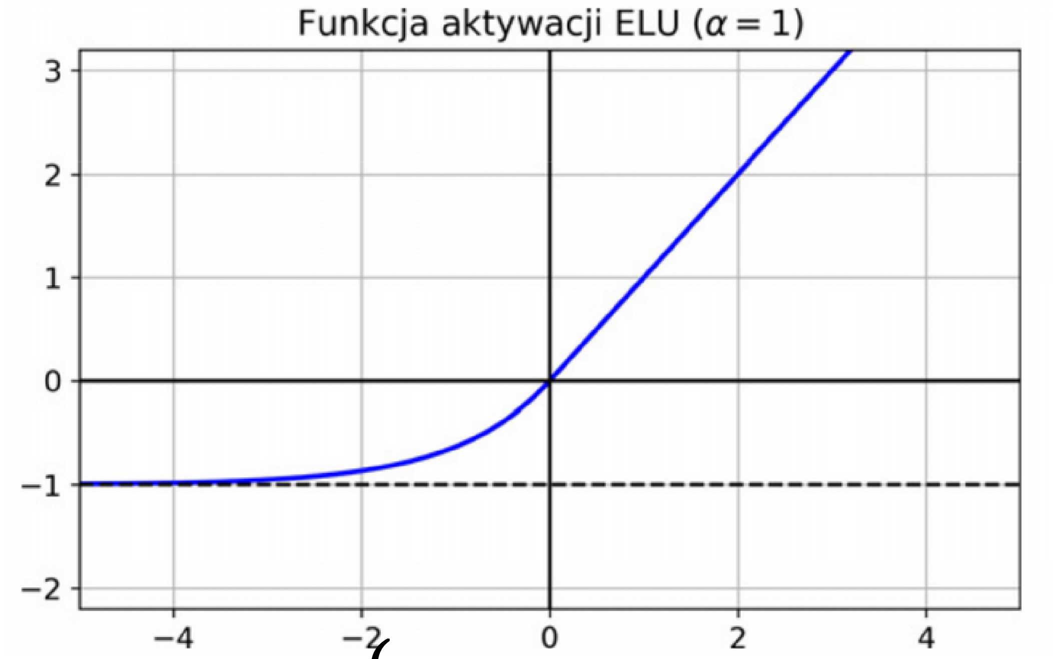
Leaky ReLU function



Variants of the ReLU function

- **Leaky ReLU**:
$$ReLU_\alpha(z) = \max\{\alpha_z, z\}$$
Hyperparameter α determines the degree of " leakage " function: it determines the slope of the function for $z < 0$ and usually takes the value of $0,01$,
- **Random, leaking function ReLU** (called randomized leaky ReLU - RReLU) - the value of the *alpha* hyperparameter is randomly selected in the specified range during learning,
- **Parametric leaky ReLU function** - parameter α "learns" along with the whole model (similar to other parameters, it can be modified in the context of back propagation).

Exponential Linear Unit - ELU



$$ELU_{\alpha}(z) = \begin{cases} \alpha(e^z - 1) & for \quad z < 0 \\ z & for \quad z \geq 0 \end{cases}$$

- In the experiments performed, it turned out to be better than all the variations of the ReLU function: the learning time was shorter, and the neural network itself performed better against the test set.

Properties of the ELU activation function

- For $z < 0$ it takes negative values - it solves this the problem of disappearing gradients,
- For $z < 0$ it has a non-zero gradient, which is a solution to the problem of "dying" neurons.
- The function is smooth at every point, also in $z = 0$, which allows you to speed up the simple gradient method,
- It is slowerly calculated from the standard ReLU function and its variations, and during learning this is compensated by a better convergence coefficient. The network from ELU will be slower than the network from ReLU.

Batch normalization

- In deep learning there is a problem of distribution of input changes in each layer during learning, resulting from changes in parameters in the previous layer,
- Before the activation function in each layer, the input data is normalized, and then rescales and shifts the result with two new parameters:
  - one is responsible for scaling
  - second for shifting
- This operation allows the model to determine the optimal scale and average of the input data for each layer.

Batch normalization using the TensorFlow module

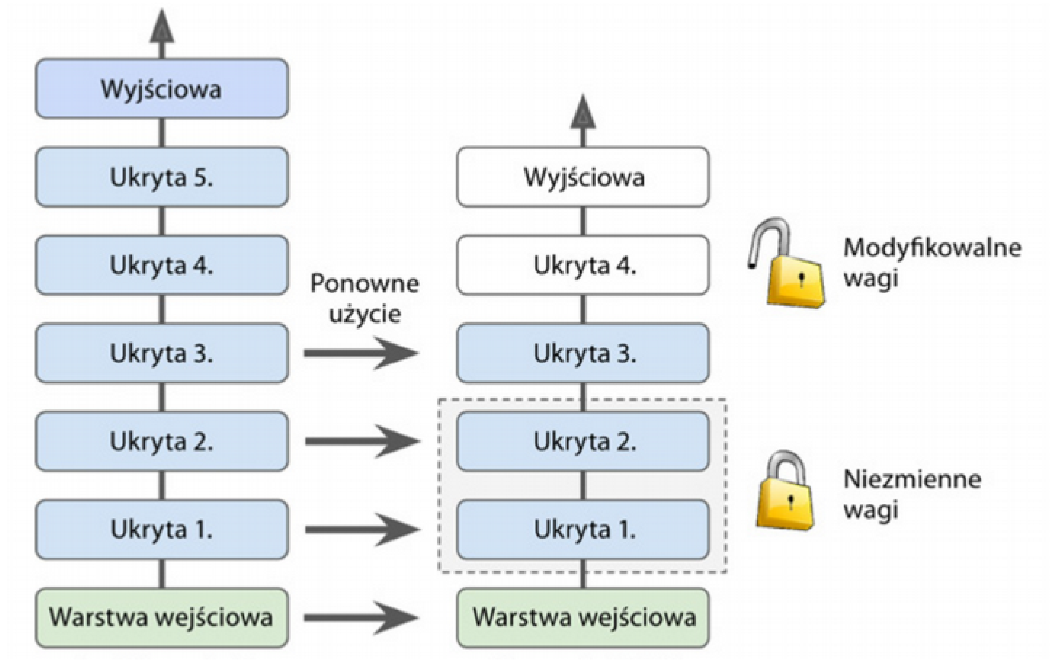
```
hidden1 = tf.layers.dense(X, n_hidden1, name="ukryta1")
bn1 = tf.layers.batch_normalization(hidden1, training=training, momentum=0.9)
bn1_act = tf.nn.elu(bn1)
hidden2 = tf.layers.dense(bn1_act, n_hidden2, name="ukryta2")
bn2 = tf.layers.batch_normalization(hidden2, training=training, momentum=0.9)
bn2_act = tf.nn.elu(bn2)
logits_before_bn = tf.layers.dense(bn2_act, n_outputs, name="wyjscia")
logits = tf.layers.batch_normalization(logits_before_bn,
training=training, momentum=0.9)
```

- The TensorFlow module contains function $tf.nn.batch\ _normalization()$, which centers and normalizes input data,
- The function $tf.layers.batch\ _normalization()$ itself calculates the mean and standard deviation (using the training data of minigroups) while passing them to the function as parameters.
- In batch normalization, no activation function is assigned - this is used after every normalization.

Gradient clipping

- A popular technique for limiting the problem of exploding gradients is to cut gradients in the back propagation stage in such a way that they never exceed a certain threshold,

Multiple use of ready-made layers



- Learning very large networks is expensive.
- You can use an existing network adapted to a similar task and use its lower layers - **transfer learning**,
- It significantly speeds up the learning process, but also requires significantly less learning data.

Freezing of lower layers

```
with tf.name_scope("gsn"):
        hidden1 = tf.layers.dense(X, n_hidden1, activation=tf.nn.relu,
        name="ukryta1") # ponownie uzyta, zamrozona
        hidden2 = tf.layers.dense(hidden1, n_hidden2, activation=tf.nn.relu,
        name="ukryta2") # ponownie uzyta, zamrozona
        hidden2_stop = tf.stop_gradient(hidden2)
        hidden3 = tf.layers.dense(hidden2_stop, n_hidden3, activation=tf.nn.relu,
        name="ukryta3") # ponownie uzyta, niezamrozona
        hidden4 = tf.layers.dense(hidden3, n_hidden4, activation=tf.nn.relu,
        name="ukryta4") # nowa!
        logits = tf.layers.dense(hidden4, n_outputs, name="wyjscia") # nowa!
```

- By inserting the *stop* $_g radient()$ layer into the graph. Any layers beneath it will be frozen.

Model repositories

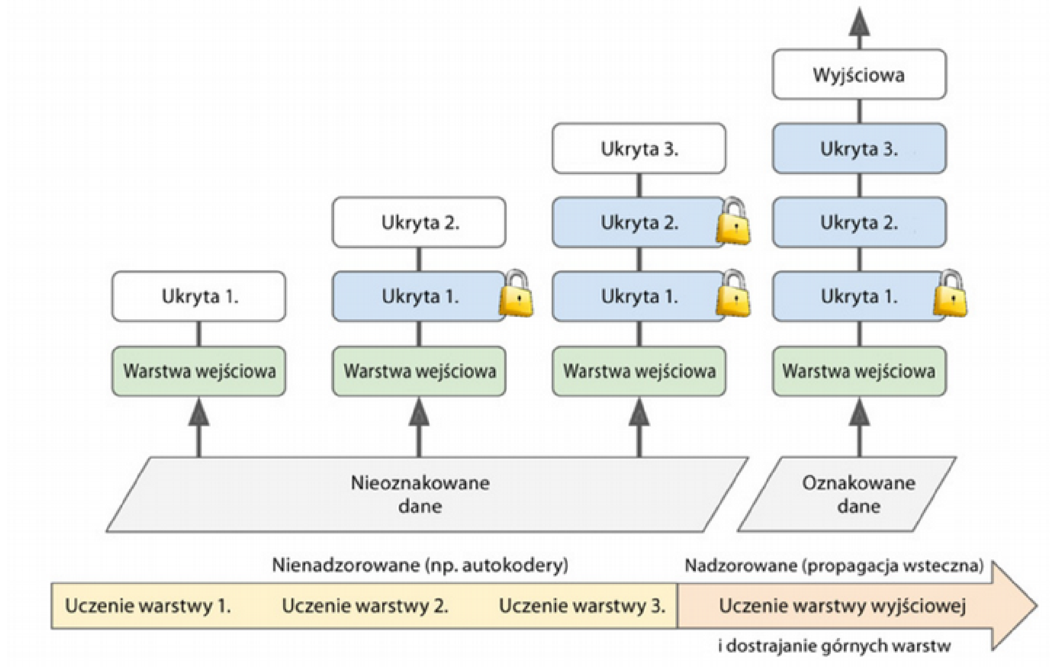
- Module *TensorFlow* has its own repository on the site $https://github.com/tensorflow/models$

  There are networks designed for image classification, such as VGG, Inception or ResNet, ready-made models, as well as tools that allow you to download the most popular learning data files

- Repository Model Zoo Caffe company: $https://github.com/BVLC/caffe/wiki/Model-Zoo$

  There are many models of digital image recognition available (eg LeNet, AlexNet, ZFNet, GoogLeNet, VGGNet, Inception) trained on a variety of data sets (such as ImageNet, Places Database, CIFAR10, etc.).

Unsupervised initial learning



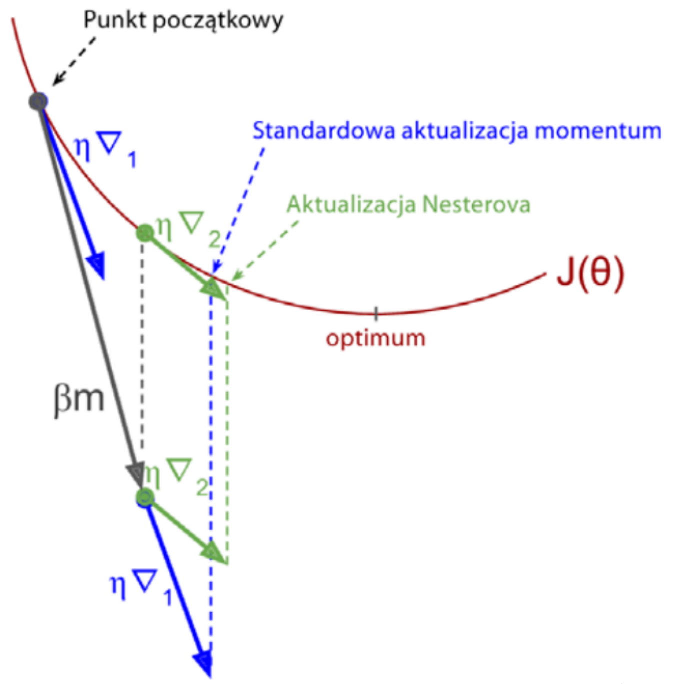
- For unmarked teaching data, you can try to train the model layer by layer, using the algorithm of unattended detection of features (eg Boltzmann machines or autocoders).
- Each subsequent layer is learned using the results obtained from the previous layers,
- After learning all layers, the network tunes backward propagation.

The momentum - optimization algorithm

$$m \leftarrow \beta m - \eta \nabla_\Theta J(\Theta)$$
$$\Theta \leftarrow \Theta + m$$

- In the *simple gradient method*, the Θ weights are updated by subtracting the $J(\Theta)$ cost function gradient from weights - $\nabla_\Theta J(\Theta)$ multiplied by η learning factor. Earlier gradients are not taken into account here,
- The *momentum* optimization takes into account the previous gradients: in each run the local gradient is subtracted from the **moment vector** m (multiplied by the learning factor η).
- The β parameter simply called moment (or momentum) prevents excessive momentum.

Accelerated gradient drop (Nesterov algorithm)

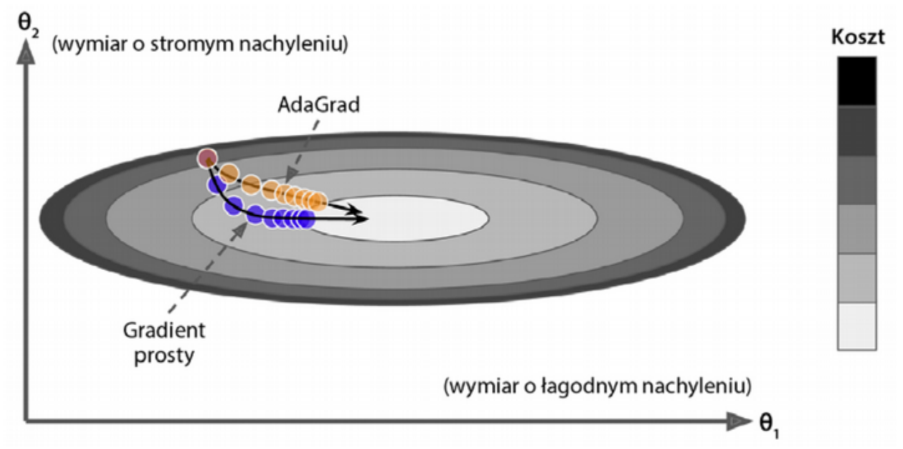


$$m \leftarrow \beta m - \eta \nabla_\Theta J(\Theta + \beta m)$$

$$\Theta \leftarrow \Theta + m$$

The only difference compared to momentum optimization is the measurement of the gradient using the expression $\Theta + \beta m$, and not Θ.

Algorithm AdaGrad



$$s \leftarrow s + \nabla_\Theta J(\Theta) \otimes \nabla_\Theta J(\Theta)$$

$$\Theta \leftarrow \Theta - \eta \nabla_\Theta J(\Theta) \oslash \sqrt{s + \in}$$

- Where the $\otimes$ symbol denotes the multiplication of individual elements, the $\oslash$ symbol denotes division by individual elements, and the $\in$ parameter is the smoothing element used to avoid dividing operations by 0,
- This is done by gradually reducing the gradient vector along the steepest directions

Algorithm RMSProp

$$s \leftarrow \beta s + (1-\beta)\nabla_\Theta J(\Theta) \otimes \nabla_\Theta J(\Theta)$$
$$\Theta \leftarrow \Theta - \eta \nabla_\Theta J(\Theta) \oslash \sqrt{s + \in}$$

- The AdaGrad algorithm never reaches convergence with the global minimum,
- This problem is solved by the **RMSProp** algorithm, by collecting only gradients from the most current waveforms,
- The distribution factor β usually has a value of 0.9.

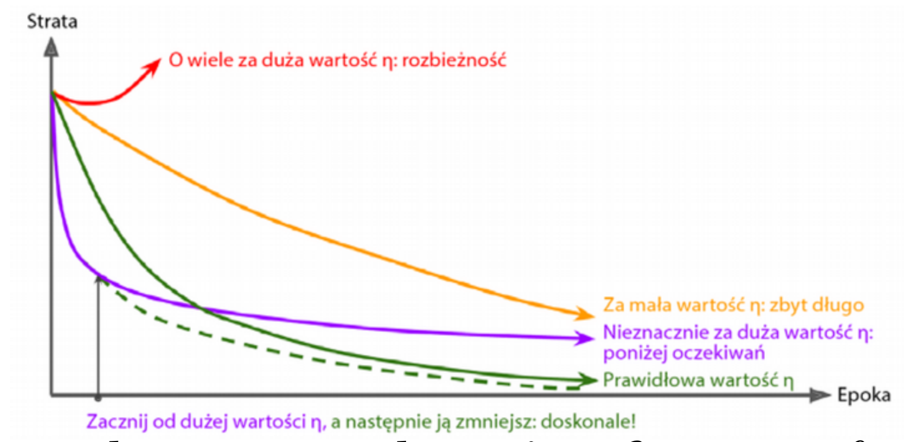
Optimization Adam - Adaptive Moment Estimation

$$m \leftarrow \beta_1 m - (1-\beta_1)\nabla_\Theta J(\Theta)$$
$$s \leftarrow \beta_2 s + (1-\beta_2)\nabla_\Theta J(\Theta) \otimes \nabla_\Theta J(\Theta)$$
$$m \leftarrow \tfrac{m}{1-\beta_1^t}$$
$$s \leftarrow \tfrac{s}{1-\beta_2^t}$$
$$\Theta \leftarrow \Theta + \eta m \oslash \sqrt{s+\in}$$

where t - step number (starting from 1).

- The Adam algorithm combines the concept of momentum optimization and RMSProp optimizer,
- from the momentum optimization takes the exponential distribution of the average of previous gradients,
- from the RMSProp optimizer to track the exponential distribution of the average of the previous squares of gradients

Scheduling the learning coefficient

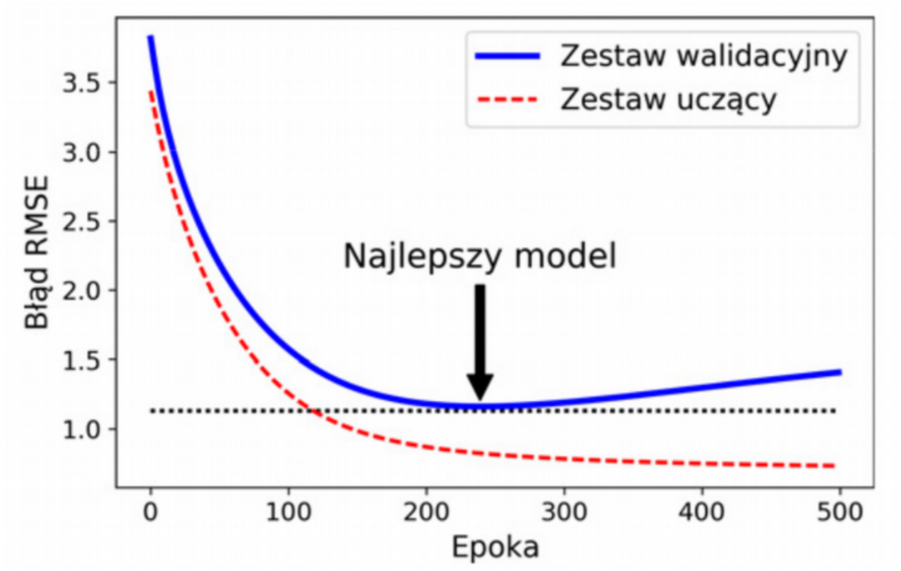


- **Top, interval, constant learning factor -** often requires additional value determination and time of change.
- **Performance scheduling -** we measure the validation error every N of the waveforms and when the error value ceases to decrease, we reduce the learning factor
- **Exponential scheduling -** we set the learning factor based on the t function
- **Power Scheduling -** similar to exponential, but the value of the learning factor decreases much slower.

Methods of protection against overfitting

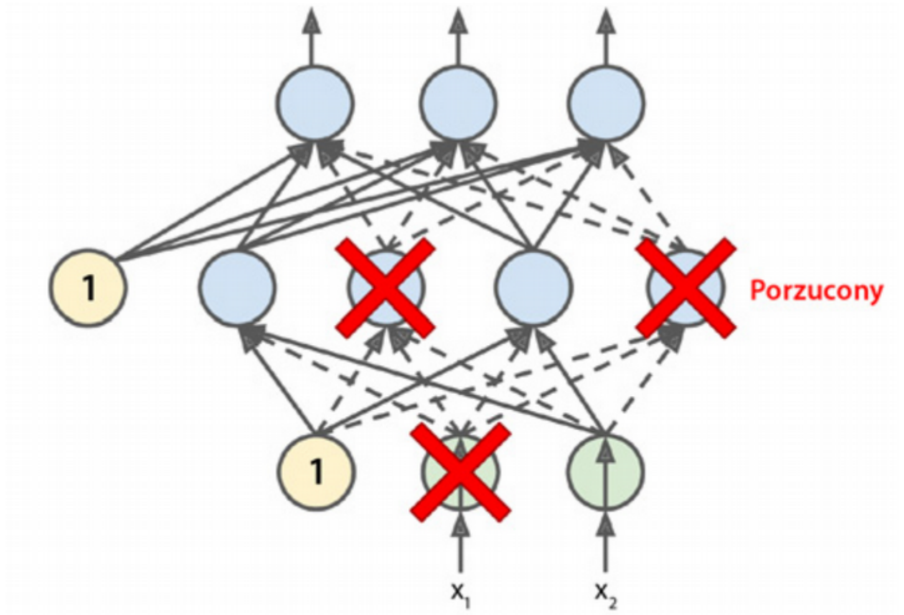
- **Early stopping** - when the performance of the model decreases against the validation set,
- **Regularization L_1 and L_2** - can be used in neural networks to limit connection weights,
- **Dropping** - in individual training courses, the neuron can be temporarily "droped", i.e. completely omitted,
- **Augmentation of data** - it is about creating new teaching examples from already existing samples.

Early stopping



- The end of learning at the moment of reaching the minimum validation error - early stopping,
- As the subsequent epochs pass, the algorithm learns, and its prediction error against the training set decreases in a natural way, just like the error of prediction against validation data.
- After some time, the forecasting error ceases to decrease and begins to increase again (the moment of overfitting).

Dropping

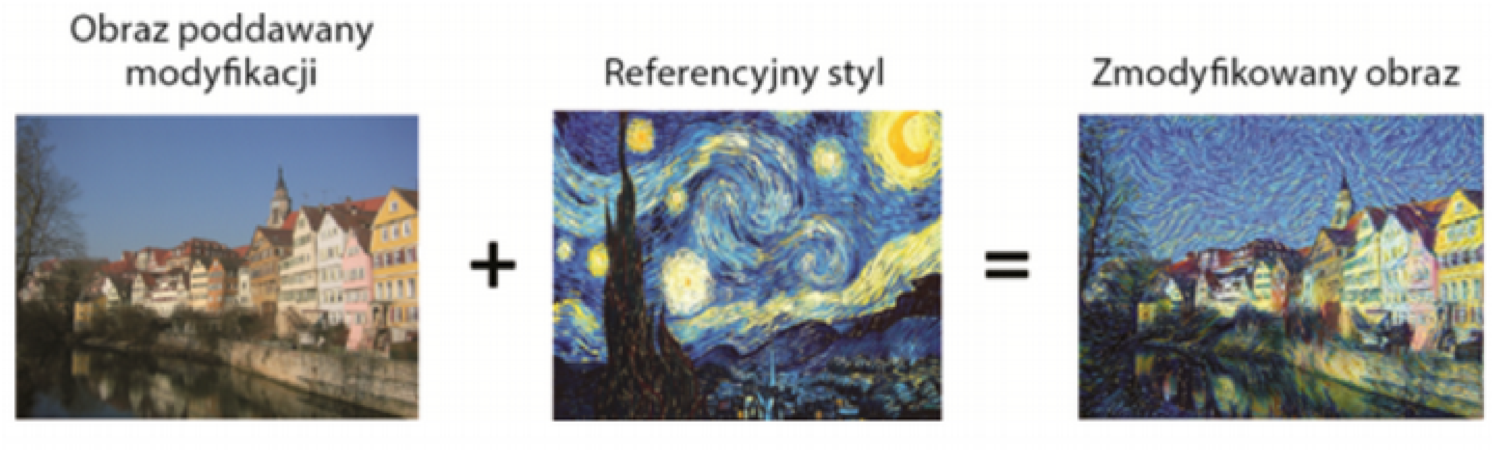


- In individual training cycles, each neuron (in addition to the initial layer) can be temporarily omitted in the learning process,
- In each cycle pass it may be active again,
- The p hyperparameter is called the dropout rate and usually has a value of 50%.
- After finishing learning, neurons stoped to be dropped.

DeepDream

- The DeepDream algorithm attempts to maximize the activation of the whole layer, not the selected filter, which results in simultaneous mixing of the visualization of many features.
- Generating the image does not start with an empty, slightly noisy input image - the finished image is processed at the input, which results in applying effects to the previously created image,
- Input images are processed at different scales referred to as octaves, which aims to improve the quality of visualization.

Neural style transfer



- Neural style transfer involves applying a reference image style to process another image while preserving its contents.
- The concept **style** refers to the textures, colors and presentation of the things shown in the image.
- **Content** is the high-level macrostructure of the image.

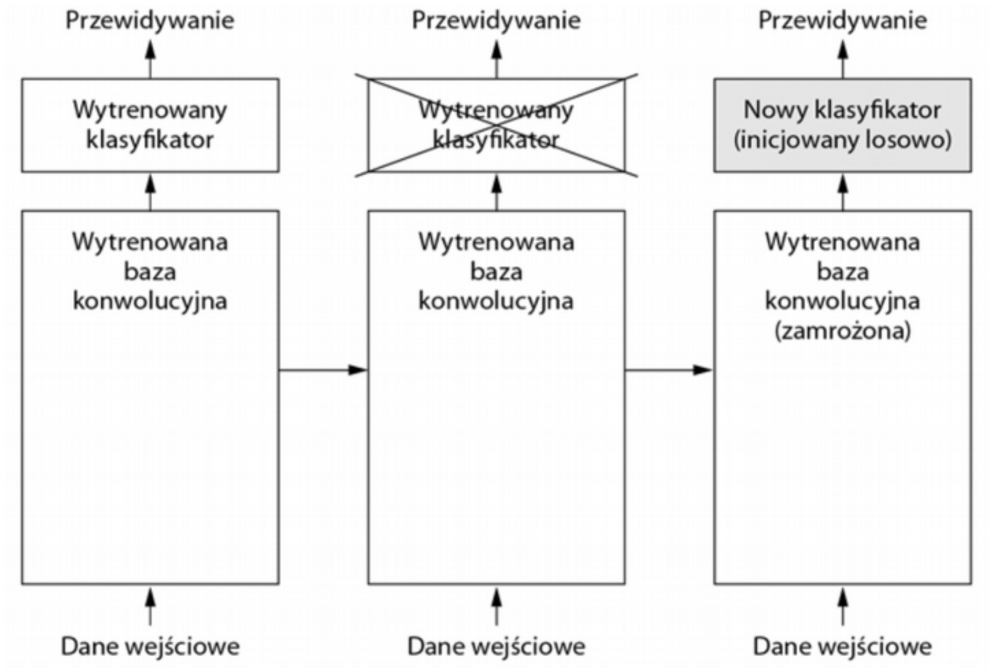
Training a convolutional neural network on a small data set

- A common situation during private work on the problems of image analysis,
- *Small number of samples* can mean a different number - from a few hundred to several dozen,
- Deep learning works mainly when it is possible to access a large amount of data,
- The algorithms of this learning can independently select useful traits from the training data set, but require a large training data set.
- A collection of several hundred examples may be sufficient if the model is small and subject to regularization, and the task will be simple.

Data augmentation technique

- Excessive matching results from too few samples on which the model can learn,
- Data Augmentation is a technique for generating more training elements of a data set by augmenting samples by random transformations that return images,
- In Keras augmentation is assisted by the instance of *ImageDataGenerator*.

Extraction of features


Przewidywanie
Wytrenowany klasyfikator
Wytrenowana baza konwolucyjna
Dane wejściowe
Przewidywanie
~~Wytrenowany klasyfikator~~
Wytrenowana baza konwolucyjna
Dane wejściowe
Przewidywanie
Nowy klasyfikator (inicjowany losowo)
Wytrenowana baza konwolucyjna (zamrożona)
Dane wejściowe

- The extraction of features consists in using the representation learned by the network earlier in order to extract the features of interest from new samples.
- These attributes are then passed through a new classifier trained from scratch.

Freezing a layer or set of layers

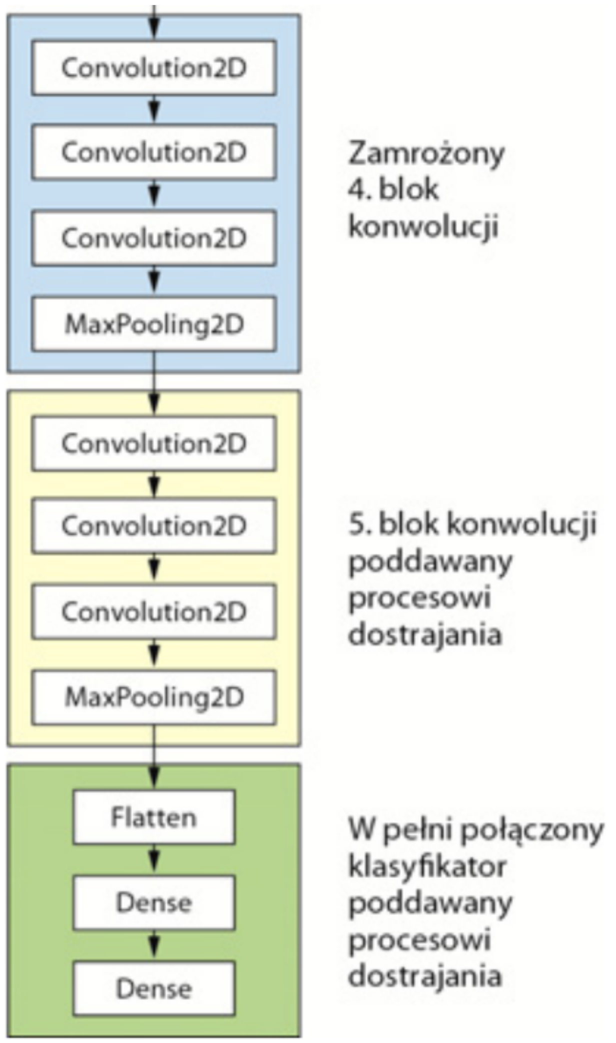
- Freezing the layer or set of layers consists in preventing the update of their weights in the training process.
- Be freezing representations previously learned by the convolutional base will be modified during training.
- Dense layers at the top are initiated randomly, which would make them go to major changes to all network parameters, and this would effectively destroy the previously learned representations.
- In the Keras package, the network freezes up by assigning *False* to the attribute *trainable*

Tuning

Tuning is a technique of reusing models that complement the extraction of features.

- It involves defrosting several top layers of frozen model base used to extract traits and training it together with a new part of the model,
- Most often this part of the model is a fully connected classifier,
- This process is referred to as tuning because it modifies partially previously trained more abstract model representations in order to adapt them to the current problem.

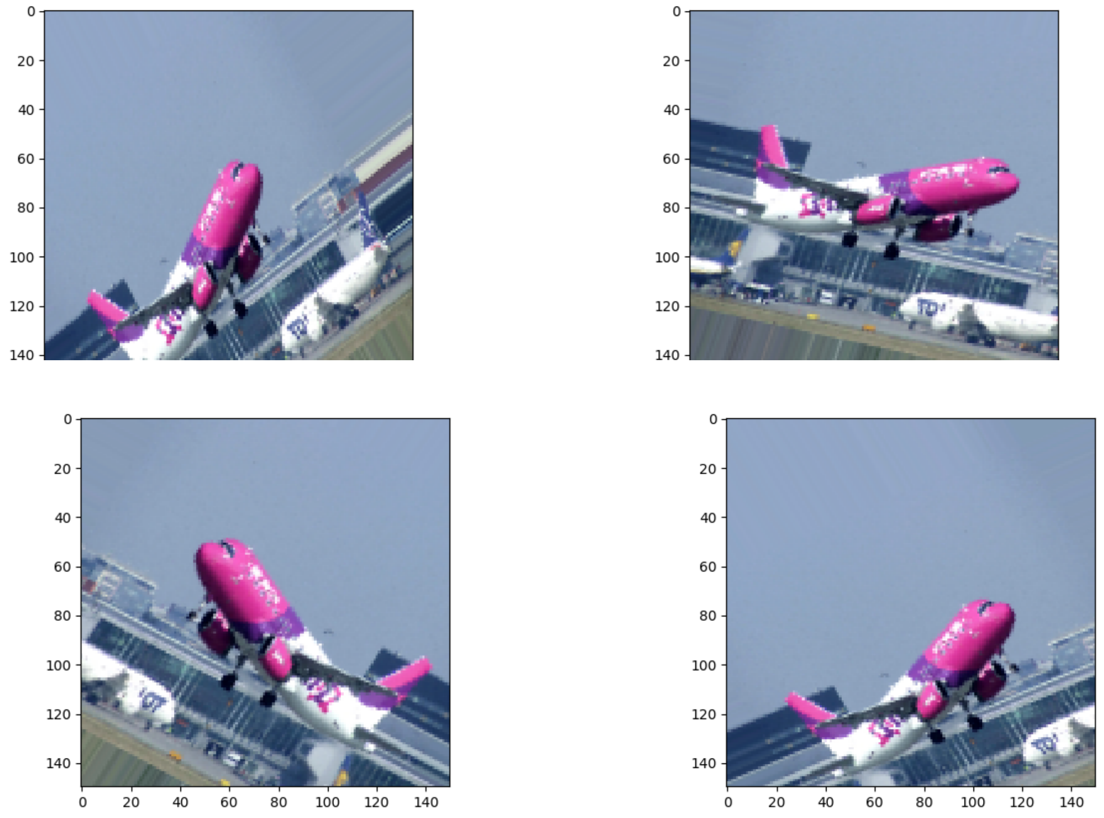
Tuning


Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
Zamrożony 4. blok konwolucji
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
5. blok konwolucji poddawany procesowi dostrajania
Flatten
Dense
Dense
W pełni połączony klasyfikator poddawany procesowi dostrajania

Visualization of the learning effects of convolutional neural networks

- **Visualization of intermediate output data** - a technique useful for understanding how consecutive layers of a convolutional network transform input data and what individual network filters do.
- **Visualization of convolutional network filters** - a technique that enables a precise understanding of a graphic pattern or graphic concept that each of the convolutional network filters reacts to.
- **Visualization of heat maps for activation of image classes** - a technique useful for identifying parts of an image identified as belonging to a given class

Augmentation of data



- It involves creating new examples from existing ones,
- To varying degrees, slightly move, rotate and resize.

Deep learning - data generation

- generating text using the LSTM network,
- DeepDream - technique of artistic image modification that uses representations learned by convolutional networks neuronowe.

Task for laboratory

- Every student is ask to take 10 photos from 3 different categories: cup, keyboard, book,
- Please create a common database (for all students) - standardize the format of all photos,
- Propose a convolutional neural network and teach it to recognize objects,
- Increase the database by using augmentation techniques. Teach the network on an extended dataset. Compare the results.