Data Community

SQL Day

GOLD SPONSORS

elitmind think bright

TECHNOLOGY INNOVATION DATA KNOWLEDGE tidk

VOLVO

Volvo Group IT

SILVER SPONSORS

dbWatch DATABASE CONTROL

it KONTRAKT®

UBS

VEEAM

BRONZE SPONSOR

KSIĄŻKI IT Z CAŁEGO ŚWIATA

NOVATECH

www.novatech.com.pl

STRATEGIC PARTNER

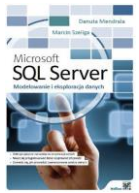
Microsoft

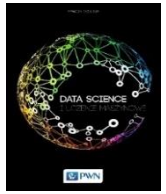
Image classification with Microsoft Cognitive Toolkit

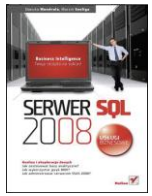
marcin.szeliga@datacommunity.pl

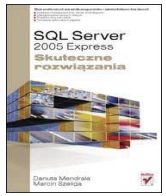
Marcin Szeliga

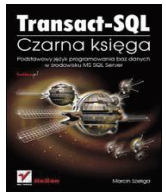
- Data philosopher
- 20 years of experience with SQL Server
- Data Platform MVP & MCT
- Microsoft Certified Solutions Expert
  - Data Management and Analytics
  - Cloud Platform and Infrastructure
  - Business Intelligence
- Microsoft Certified Solutions Developer
  - Azure Solution Architect

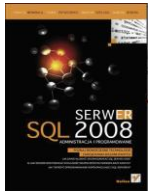






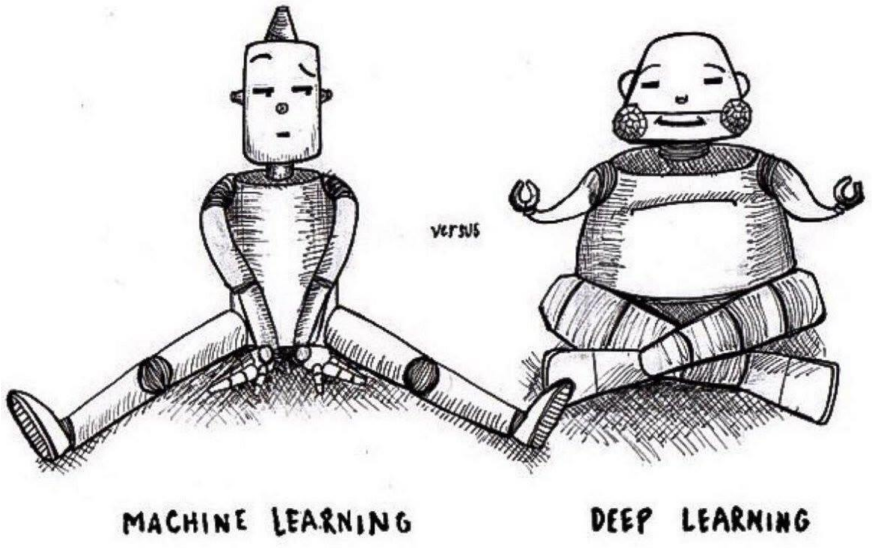




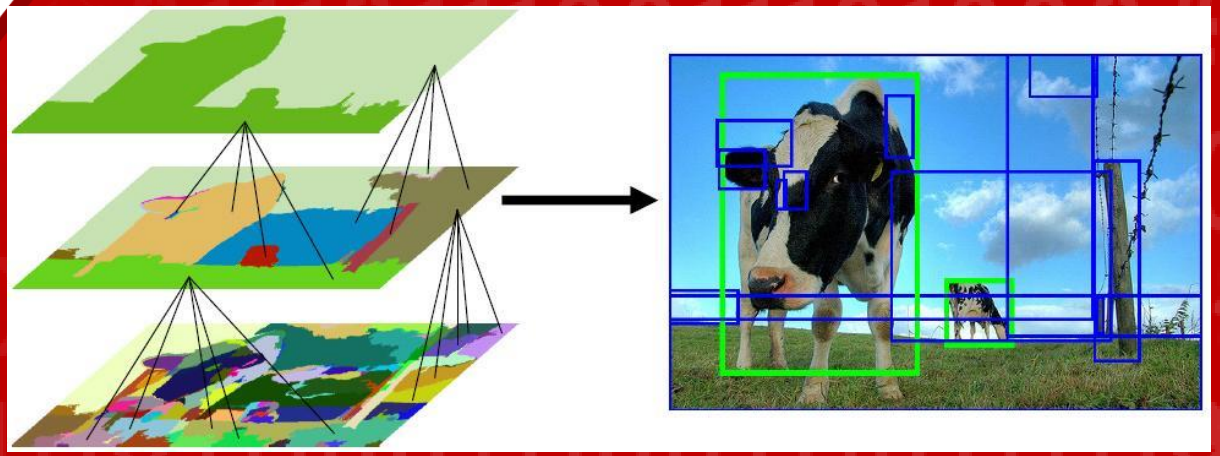


Agenda

- Why image classification is so difficult?
- Nearest Neighbor Classifier
- Linear classification
- Regular Neural Networks
- Convolutional Neural Networks

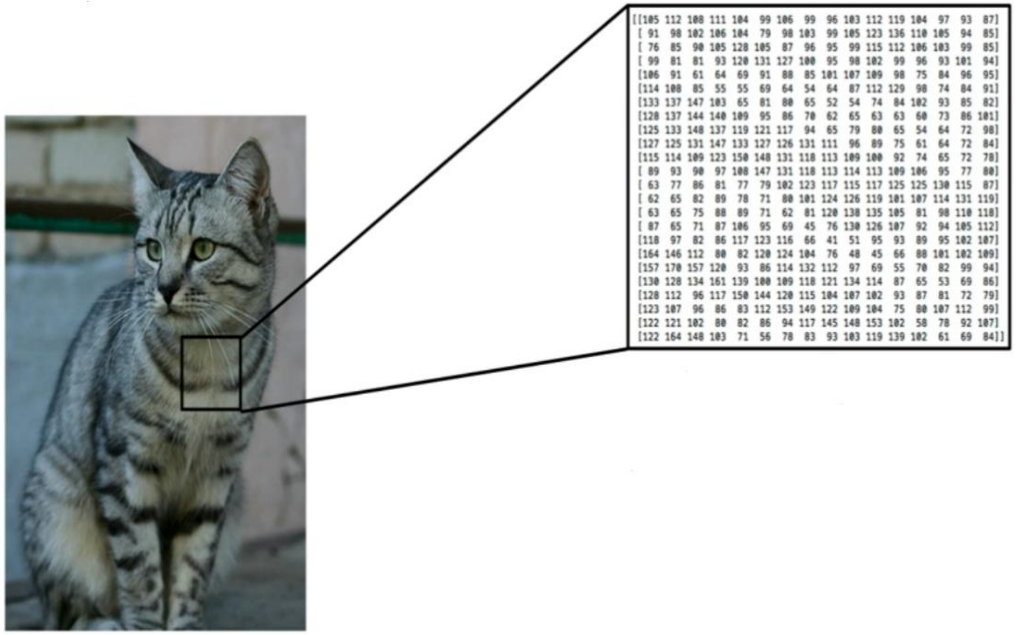


Why image classification is so difficult?

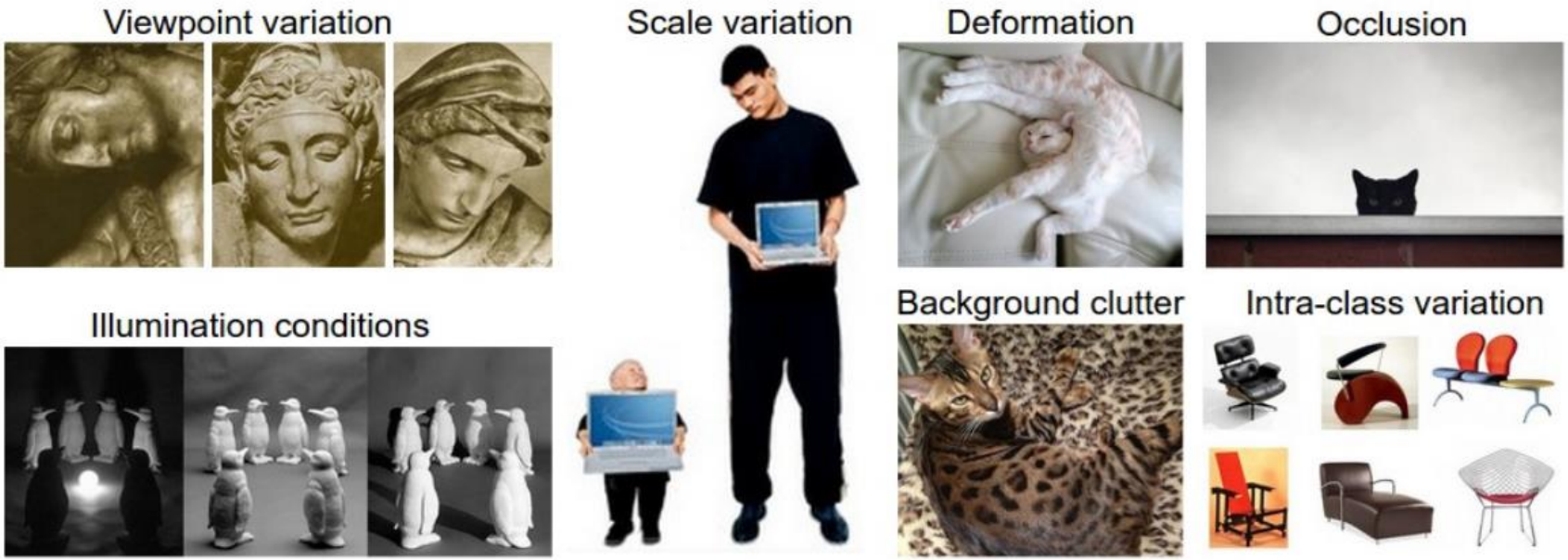


To a computer an image is represented as one large 3-dimensional array of numbers

- Cat image is 248 pixels wide, 400 pixels tall, and has three color channels Red,Green,Blue
- The image consists of 248 x 400 x 3 numbers = 297,600
- Each number is an integer that ranges from 0 (black) to 255 (white)



Challenges



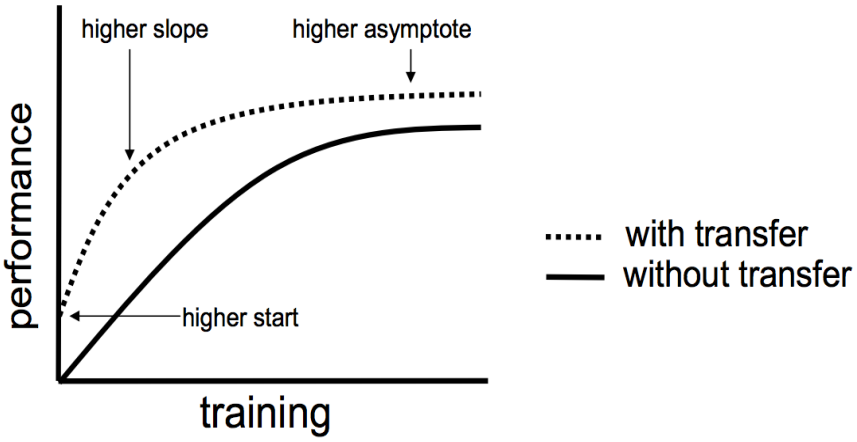
Good image classification model must be invariant to the cross product of all these variations

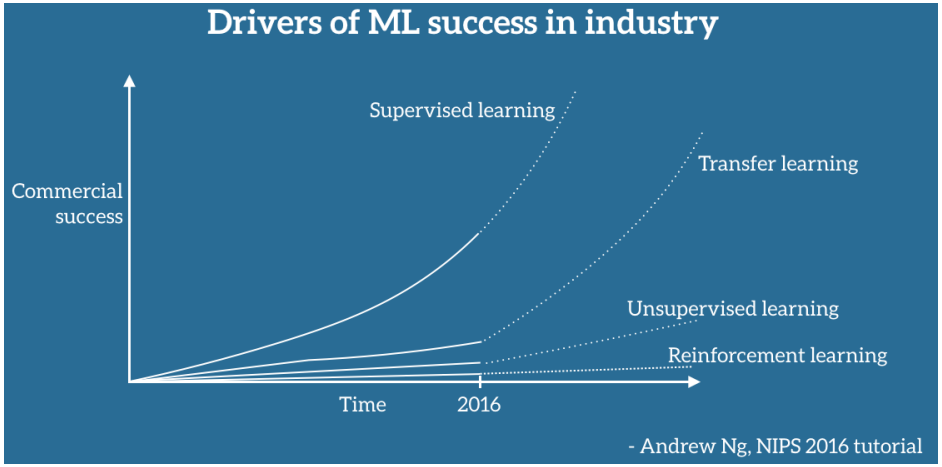
Fast solution

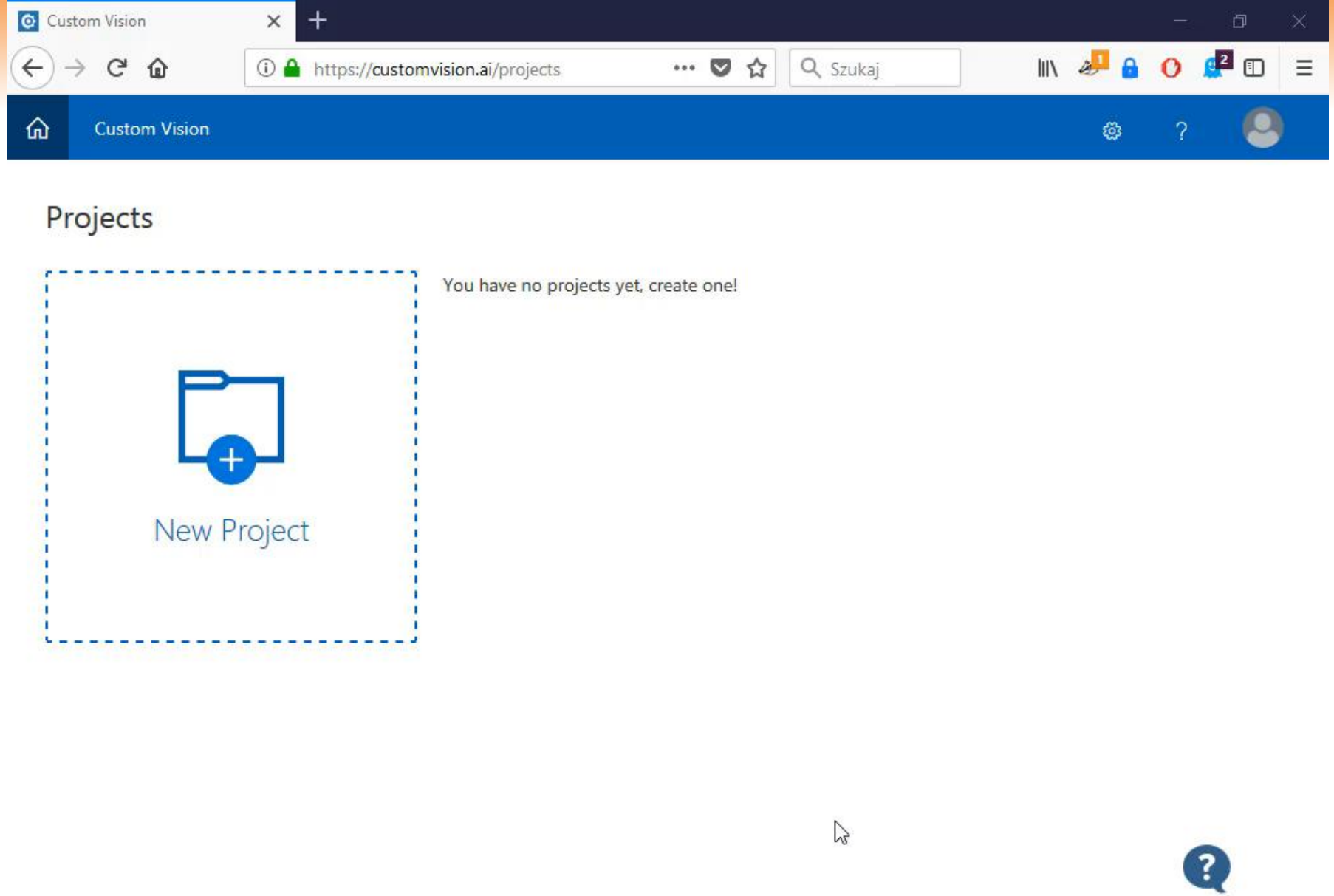
- Custom vision: https://www.customvision.ai/
- Easily customize your vision models that fit perfectly with your unique use case
- Bring your own labeled images, or use Custom Vision to quickly add tags to any unlabeled images
- Use your labeled images to teach Custom Vision the concepts you care about
- Use simple REST API calls to quickly tag images with your new custom computer vision model

Quick introduction to transfer learning

- Machine learning method where a model developed for a task is reused as the starting point for a model on a second task
- Very popular in deep learning given the enormous resources required to train deep learning models
- Use a deep learning model pre-trained for a large and challenging image classification task such as the ImageNet 1000

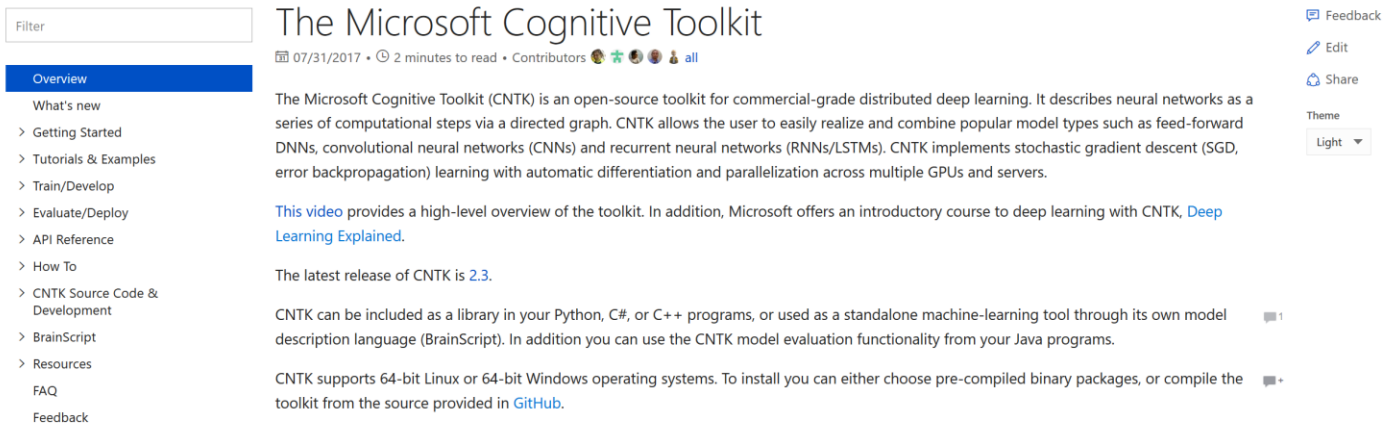





Custom Vision
https://customvision.ai/projects
Szukaj
Custom Vision
Projects
You have no projects yet, create one!
New Project

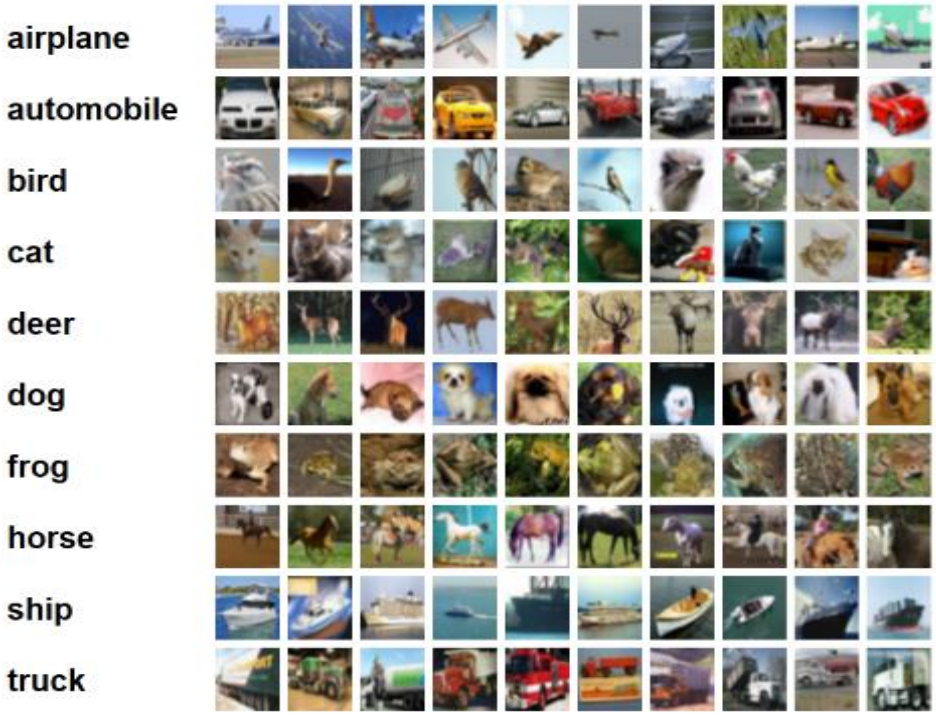
Microsoft Cognitive Toolkit

- A free, easy-to-use, open-source, commercial-grade toolkit that trains deep learning algorithms to learn like the human brain
  - https://www.microsoft.com/en-us/cognitive-toolkit/
- Highly optimized, built-in components
  - Automatic hyperparameter tuning
  - Built-in readers optimized for massive datasets
- Efficient resource usage
  - Parallelism with accuracy on multiple GPUs/machines via 1-bit SGD and Block Momentum



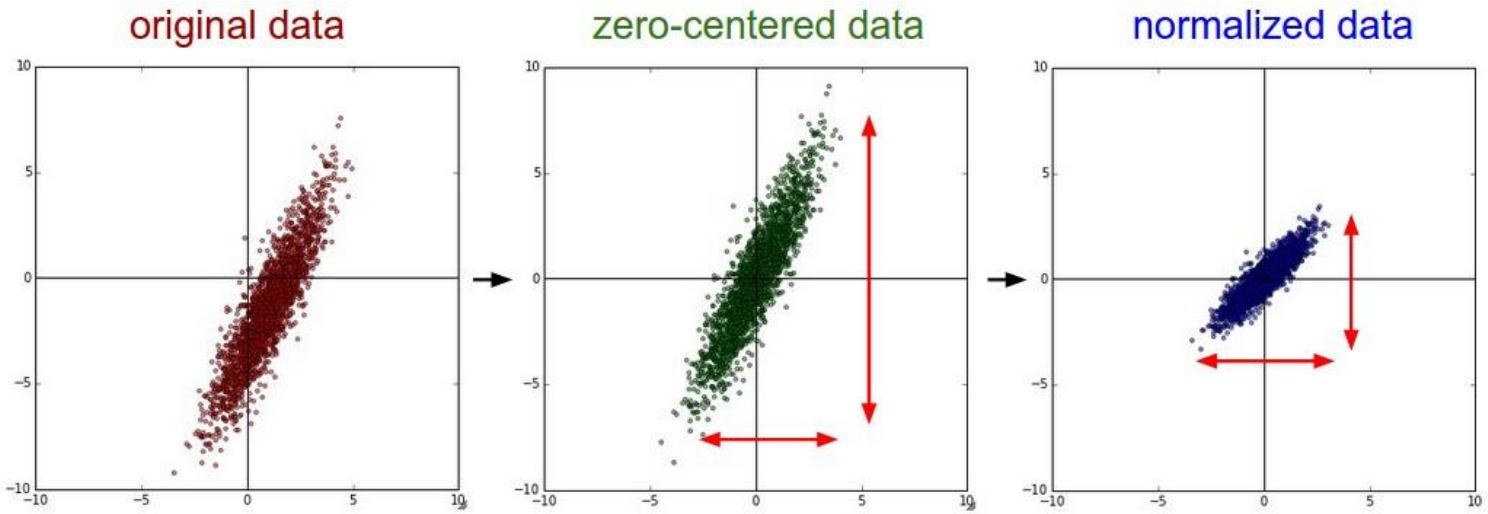
CIFAR-10 dataset

- Consists of 60 000 32x32 colour images in 10 classes, with 6 000 images per class
- Classes are completely mutually exclusive
- 50 000 training images and 10 000 test images
- https://www.cs.toronto.edu/~kriz/cifar.html



Data Preprocessing

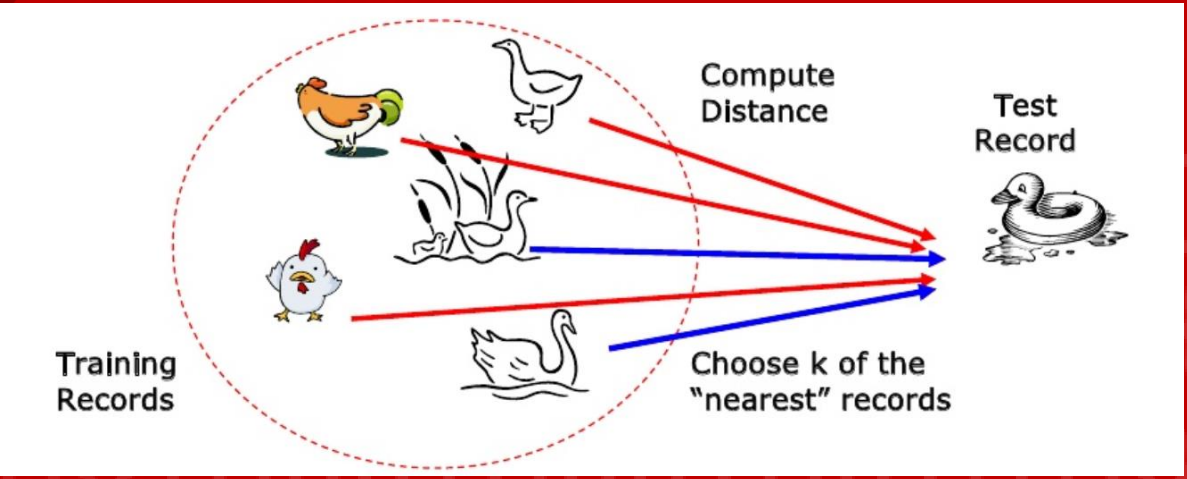
- Mean subtraction is the most common form of preprocessing
  - Subtract a single value from all pixels or do so separately across the three color channel
- Normalization scale values to a specific range



C:\Windows\system32\cmd.exe - jupyter notebook

```
C:\local\CNTK-2-2\cntk\Scripts>cntkpy35.bat

(C:\local\Anaconda3-4.1.1-Windows-x86_64\envs\cntk-py35) C:\local\CNTK-2-
2\cntk\Scripts>jupyter notebook
[I 07:33:16.928 NotebookApp] Serving notebooks from local directory: C:\l
ocal\CNTK-2-2\cntk\Scripts
[I 07:33:16.929 NotebookApp] 0 active kernels
[I 07:33:16.929 NotebookApp] The Jupyter Notebook is running at: http://l
ocalhost:8888/?token=815c7c37f92035a3a9afe254c6d8c01e8c469680ebfa9cf4
[I 07:33:16.929 NotebookApp] Use Control-C to stop this server and shut d
own all kernels (twice to skip confirmation).
[C 07:33:16.934 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first
time,
    to login with a token:
        http://localhost:8888/?token=815c7c37f92035a3a9afe254c6d8c01e8c46
9680ebfa9cf4
[I 07:33:17.710 NotebookApp] Accepting one-time-token-authenticated conne
ction from ::1
[I 07:33:26.815 NotebookApp] Kernel started: 93368b60-504d-4635-89e2-e79c
2e8d75a7
[I 07:33:28.066 NotebookApp] Adapting to protocol v5.1 for kernel 93368b6
0-504d-4635-89e2-e79c2e8d75a7
[I 07:33:42.914 NotebookApp] Saving file at /LogisticRegression.ipynb
[I 07:33:48.087 NotebookApp] Kernel started: 6e0d6ed2-fb9c-49d9-a55c-ad27
```

Nearest Neighbor Classifier



How to compare two images?

- L1 distance - compare the images pixel by pixel and add up all the differences

Train image	
30	95
60	43

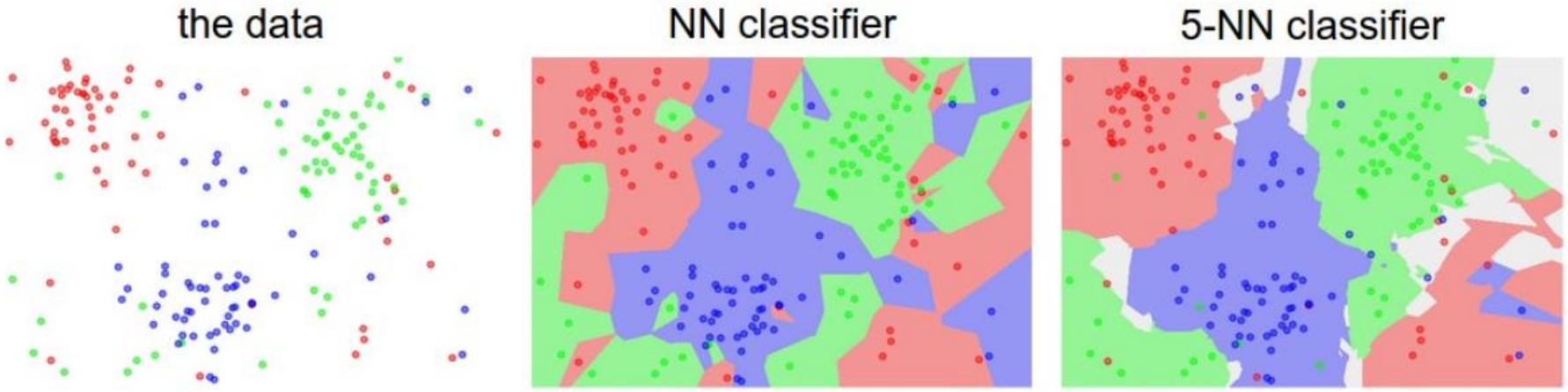
Test image	
50	200
33	140

Pixel-wise absolute difference	
20	105
27	97

  - $L1 = d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p| = 249$
- L2 distance has the geometric interpretation of computing the Euclidean distance between two vectors
  - $L2 = d_1(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2} = 147$
  - Prefers many medium disagreements to one big one

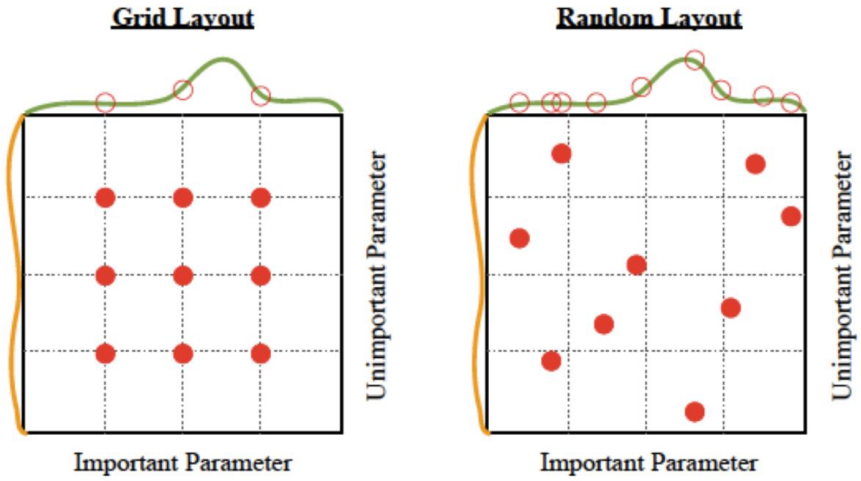
K-Nearest Neighbor Classifier

- Image can be labeled as the more similar train image
- Nonparametric models can capture more subtle aspects of the data, but are huge and slow
- What value of *k* should you use?

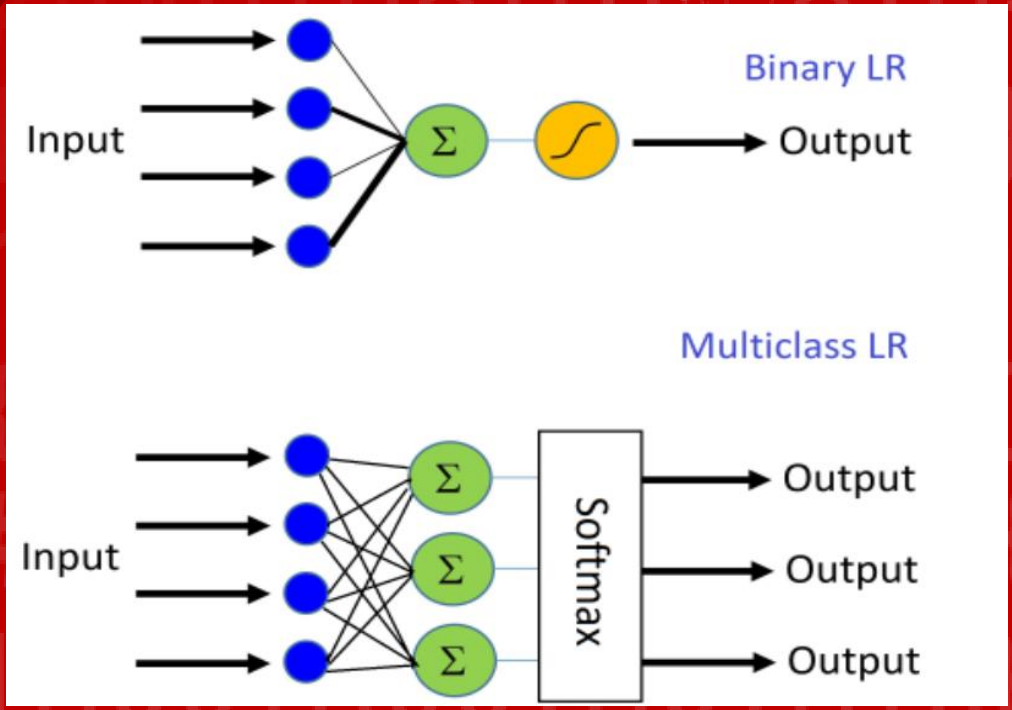


Hyperparameters

- Distance function and number of nearest neighbors are model hyperparameters
- We should try out many different values and see what works best
- But we cannot use the train set neither test set for the purpose of tweaking hyperparameters
- The idea is to split training set in two:
  - Slightly smaller training set
  - And a validation set

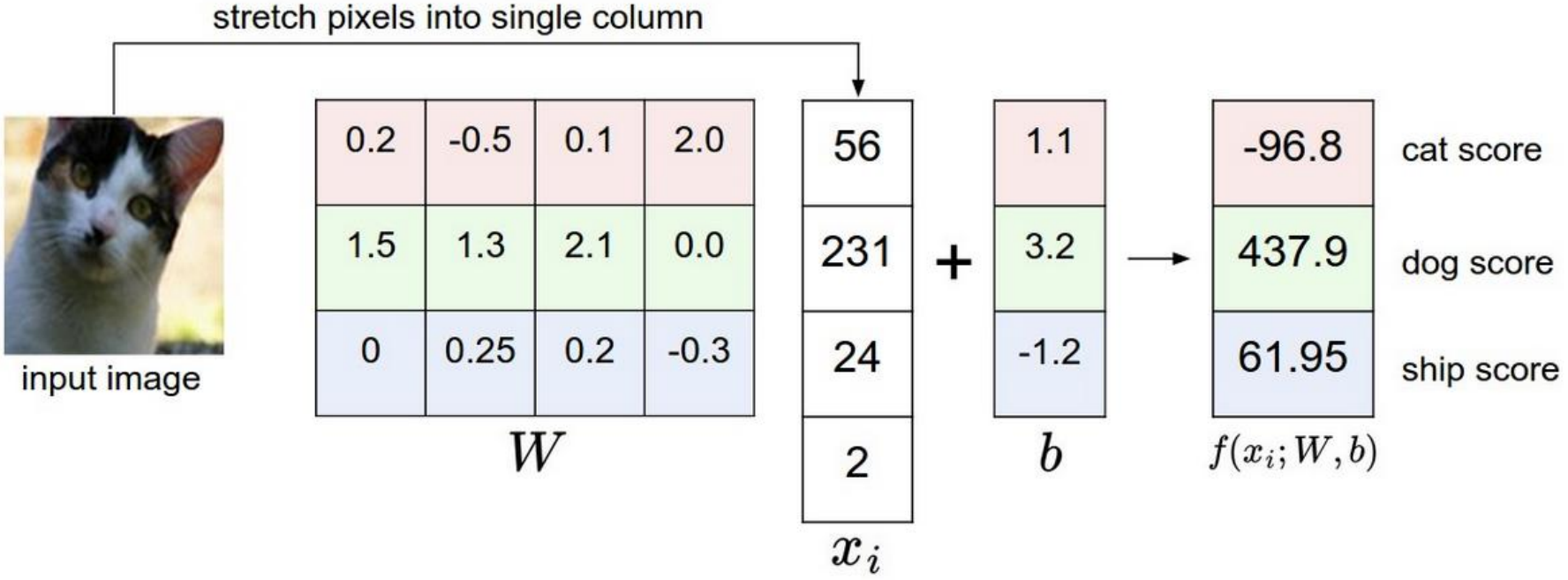


Linear Classification



Parameterized models

- We need a score function that maps the pixel values of an image to confidence scores for each class
- The simplest is linear regression: $y = f(x) = Wx_1 + b$



Template matching

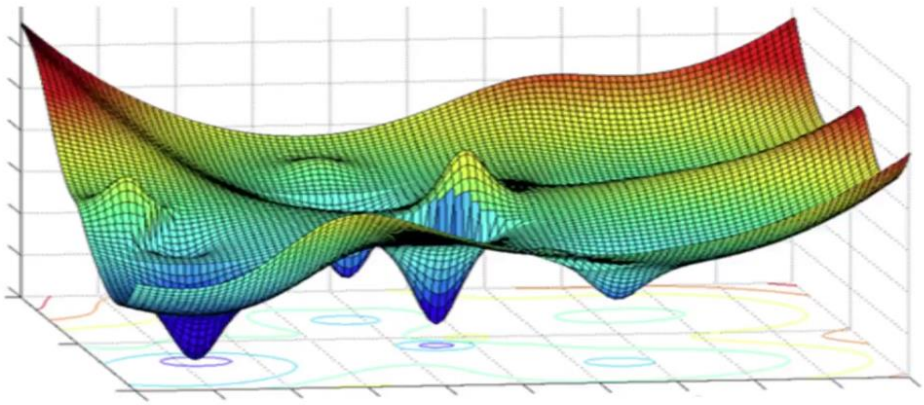
- Each row of W corresponds to a template
- The score of each class for an image is obtained by comparing each template with the image using dot product one by one to find the one that fits best
- Like Nearest Neighbor, but instead of having thousands of training images we are only using a single image per class

Loss function

- The task is to find a function that minimizes the difference between predicted and actual values
- The loss function quantifies our unhappiness with predictions on the training set

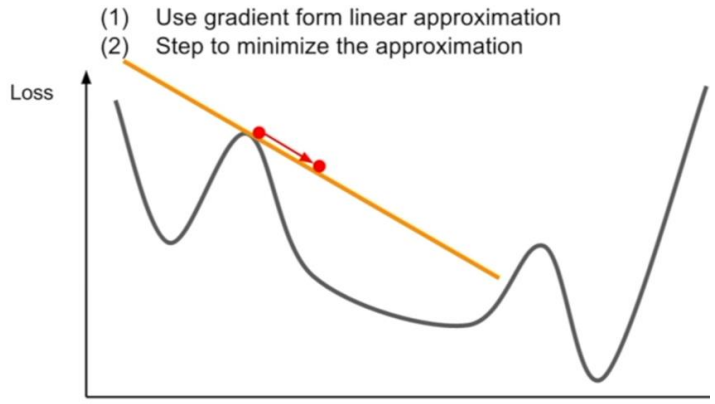
$$se = \sum_{j=0}^{9} (y_j - p_j)^2$$

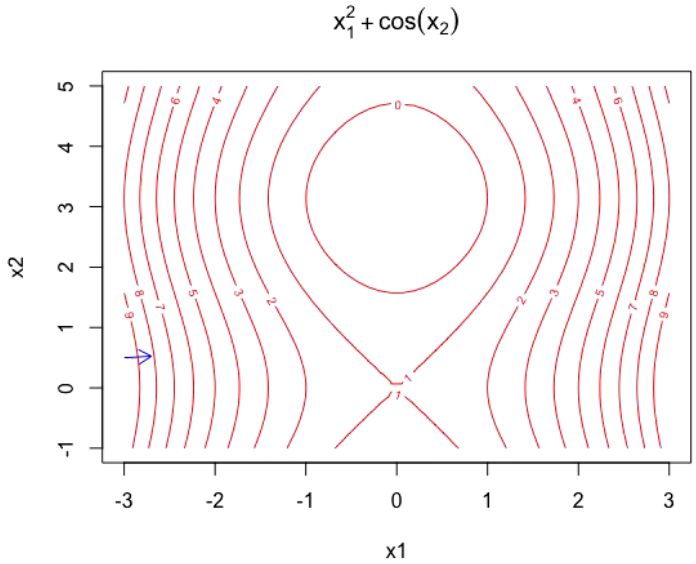
$$ce = -\sum_{j=0}^{9} y_j \, log(p_j)$$



Stochastic gradient descent

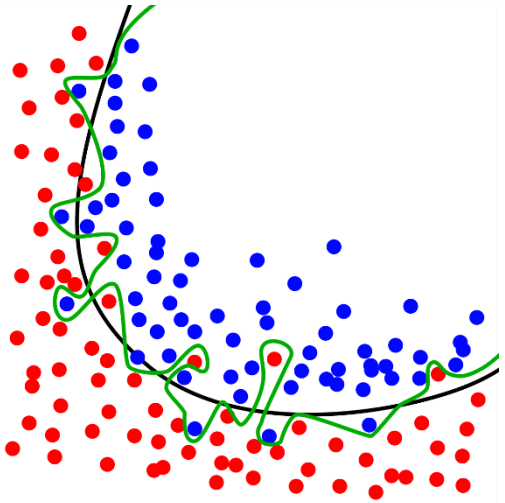
- Loss function lets us quantify the quality of any particular set of weights W
- Goal of optimization is to find W that minimizes the loss function
- We can compute the best direction along which we should change our weight vector that is mathematically guaranteed to be the direction of the steepest descend





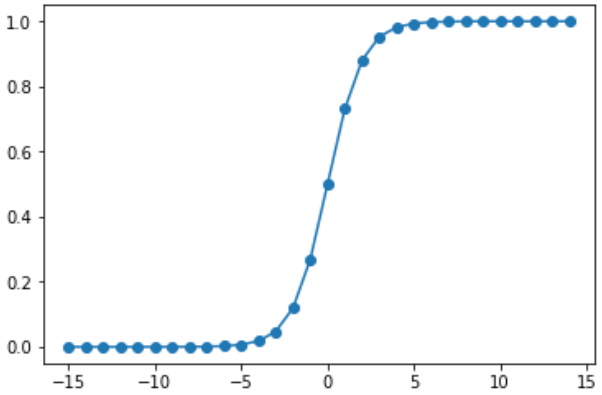
Overfitting

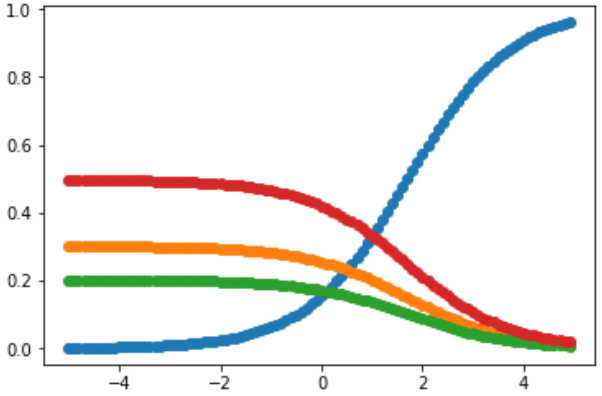
- For a given formula loss function can be very small for train set
- To prevent overfitting the model should be generalized
  - We can add regularization term to loss function
    - L2 regularization (Ridge) adds squared magnitude of weights as penalty term to the loss function
    - L1 regularization (Lasso) adds absolute value of magnitude of weights as penalty term to the loss function
  - We also can
    - Stop training early
    - Dropout random parameters per minibatch
  - But adding more images yields the best results
    - Use data augumentation



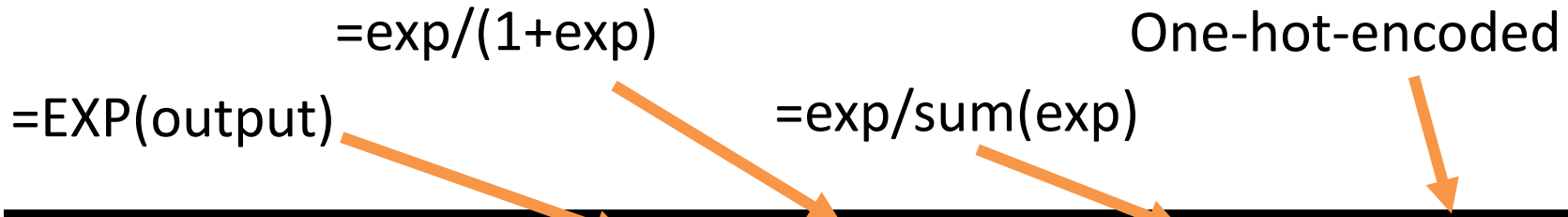
Probability distribution

- Our goal is try to predict the probability that a given example belongs to the specific class
- We need a s-shaped function that squashes values into the range [0,1] so that we may interpret them as a probability
- Model has to search for a value of loss function so that the probability is large when *x* belongs to the right class and small when *x* belongs to any other classes

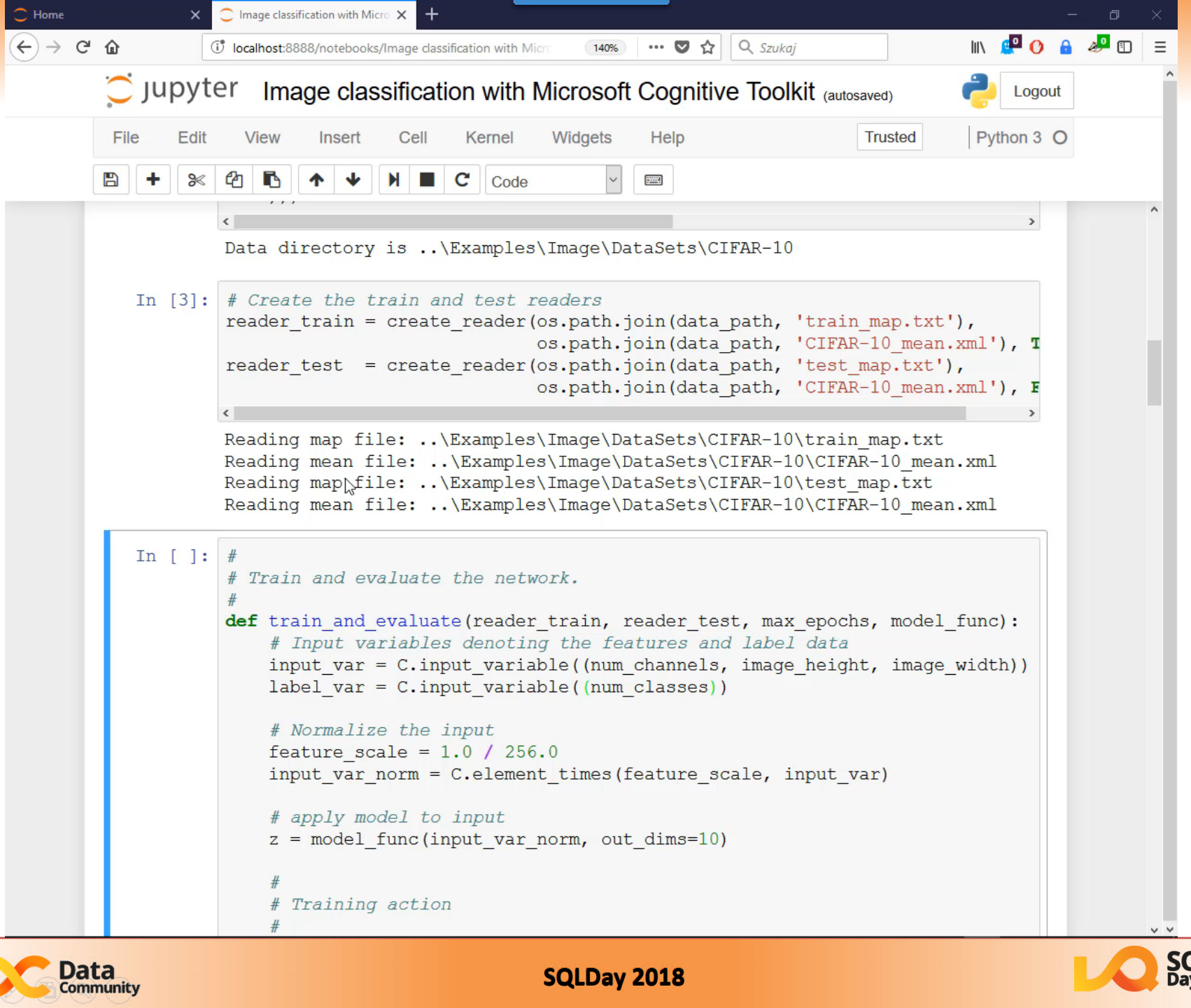




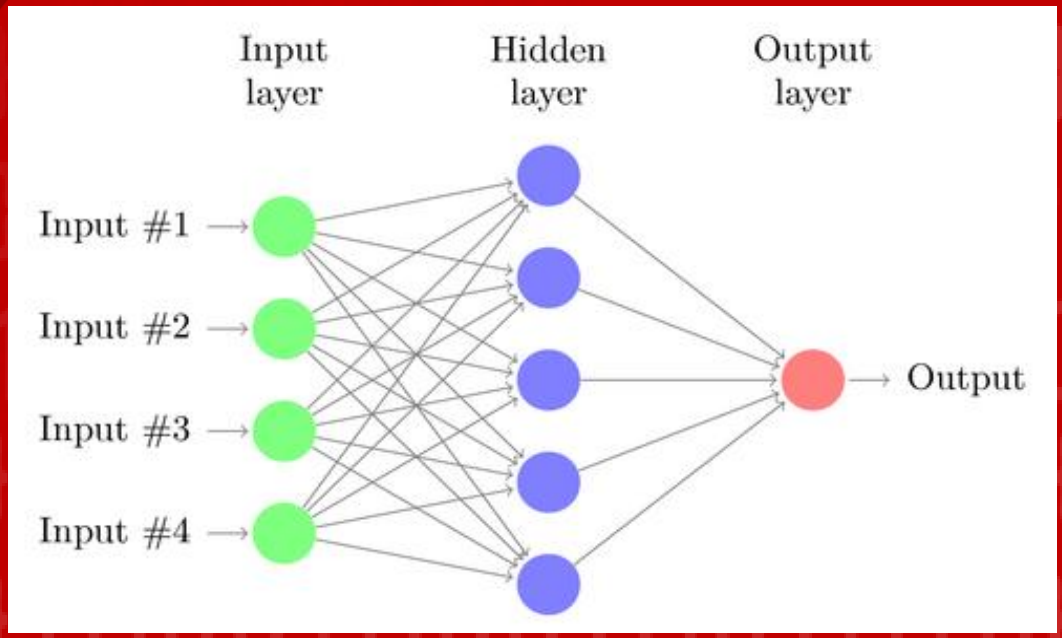
Sigmoid and Softmax explained



class	output	exp	sigmoid	softmax	actual
cat	-4.64	0.01	0.009535211	0.00	0
dog	-2.68	0.07	0.064288227	0.00	0
plane	2.87	17.55	0.946089038	0.98	1
fish	-2.08	0.13	0.11129408	0.01	0
building	-2.26	0.10	0.094315593	0.01	0
sum	**-8.80**	**17.86**	**1.23**	**1.00**	

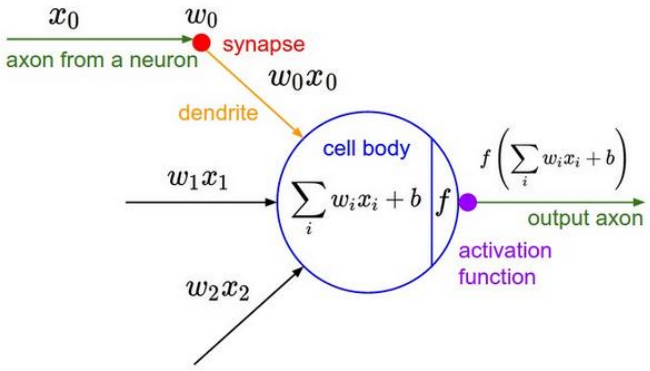

Home
Image classification with Micro
localhost:8888/notebooks/Image classification with Micro
140%
Szukaj
jupyter
Image classification with Microsoft Cognitive Toolkit (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Code
Data directory is ..\Examples\Image\DataSets\CIFAR-10
In [3]:
# Create the train and test readers
reader_train = create_reader(os.path.join(data_path, 'train_map.txt'),
os.path.join(data_path, 'CIFAR-10_mean.xml'), T
reader_test = create_reader(os.path.join(data_path, 'test_map.txt'),
os.path.join(data_path, 'CIFAR-10_mean.xml'), F
Reading map file: ..\Examples\Image\DataSets\CIFAR-10\train_map.txt
Reading mean file: ..\Examples\Image\DataSets\CIFAR-10\CIFAR-10_mean.xml
Reading map file: ..\Examples\Image\DataSets\CIFAR-10\test_map.txt
Reading mean file: ..\Examples\Image\DataSets\CIFAR-10\CIFAR-10_mean.xml
In [ ]:
#
Train and evaluate the network.
#
def train_and_evaluate(reader_train, reader_test, max_epochs, model_func):
# Input variables denoting the features and label data
input_var = C.input_variable((num_channels, image_height, image_width))
label_var = C.input_variable((num_classes))
# Normalize the input
feature_scale = 1.0 / 256.0
input_var_norm = C.element_times(feature_scale, input_var)
# apply model to input
z = model_func(input_var_norm, out_dims=10)
#
Training action
#

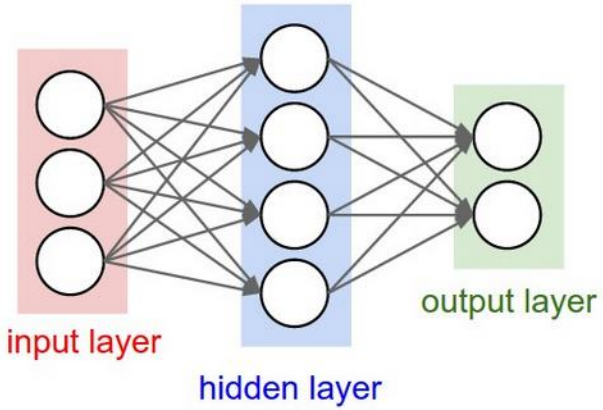
Regular Neural Networks



Artificial neural networks

- Basic computational unit in is a neuron (perceptron)
- Input signals interact multiplicatively with outputs of the other neurons based on weight
- Weight signals get summed and bias is added
- Activation function applies the non-linearity
- Think of :
  - Regression as NN without hidden layer
  - Slope as weight
  - Intercept as bias

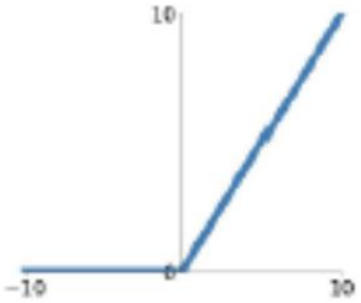




Activation functions

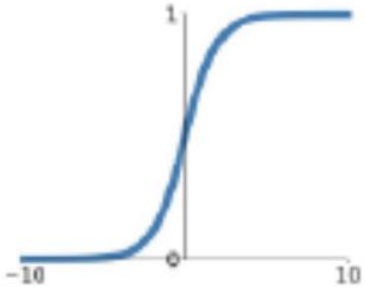
- Rectified linear unit (ReLU) is a neuron, in which the output signal appears when the inputs exceeded specified threshold

$$z = b + \sum_i x_i w_i;\ y = \begin{cases} z, & \text{if } z \geq 0, \\ 0, & \text{if } z < 0 \end{cases}$$



- In shallow NN sigmoid neurons are used

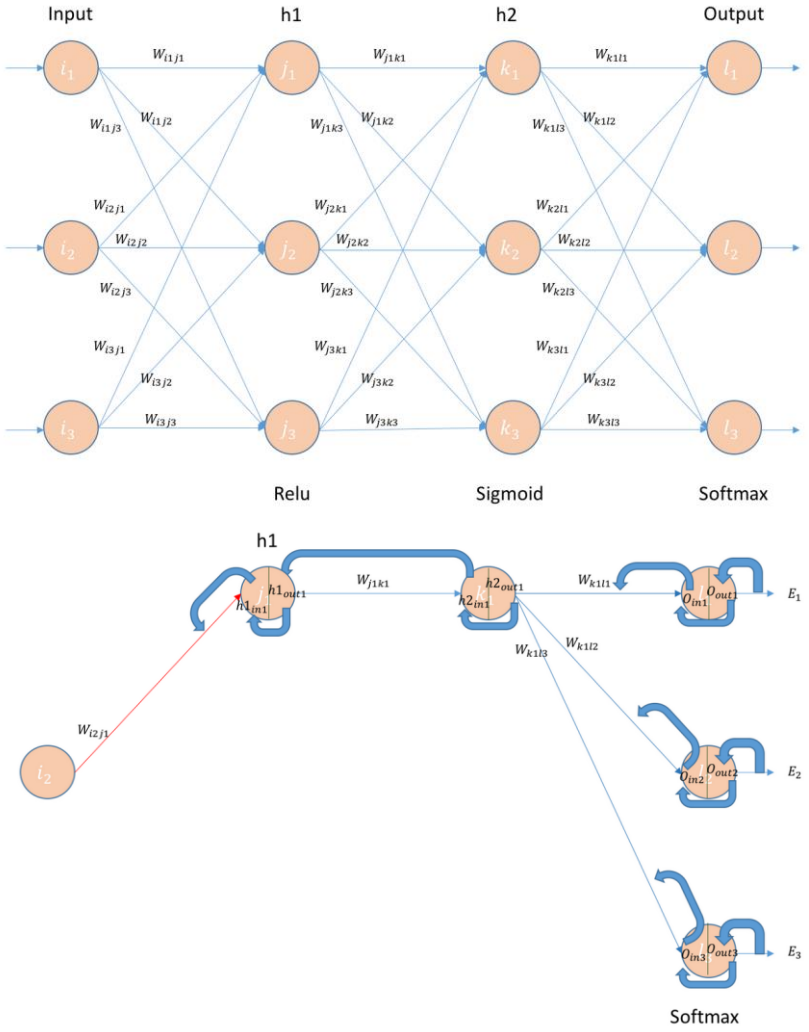
$$z = b + \sum_i x_i w_i;\ y = \frac{1}{1 + e^{-z}}$$



- Both allows artificial neural network to learn using backpropagation method

Backward propagation of errors

- Learning repeats two phase cycle, propagation and weight update
- When an input vector is presented to the network, it is propagated forward through the network, layer by layer, until it reaches the output layer
- The predicted output is then compared to the desired (true) one, using a loss function, and an error value is calculated
- The error values are then propagated backwards using chain rule



Home | Image classification with Micro

localhost:8888/notebooks/Image classification with Micr 140% Szukaj

jupyter Image classification with Microsoft Cognitive Toolkit (autosaved) Logout

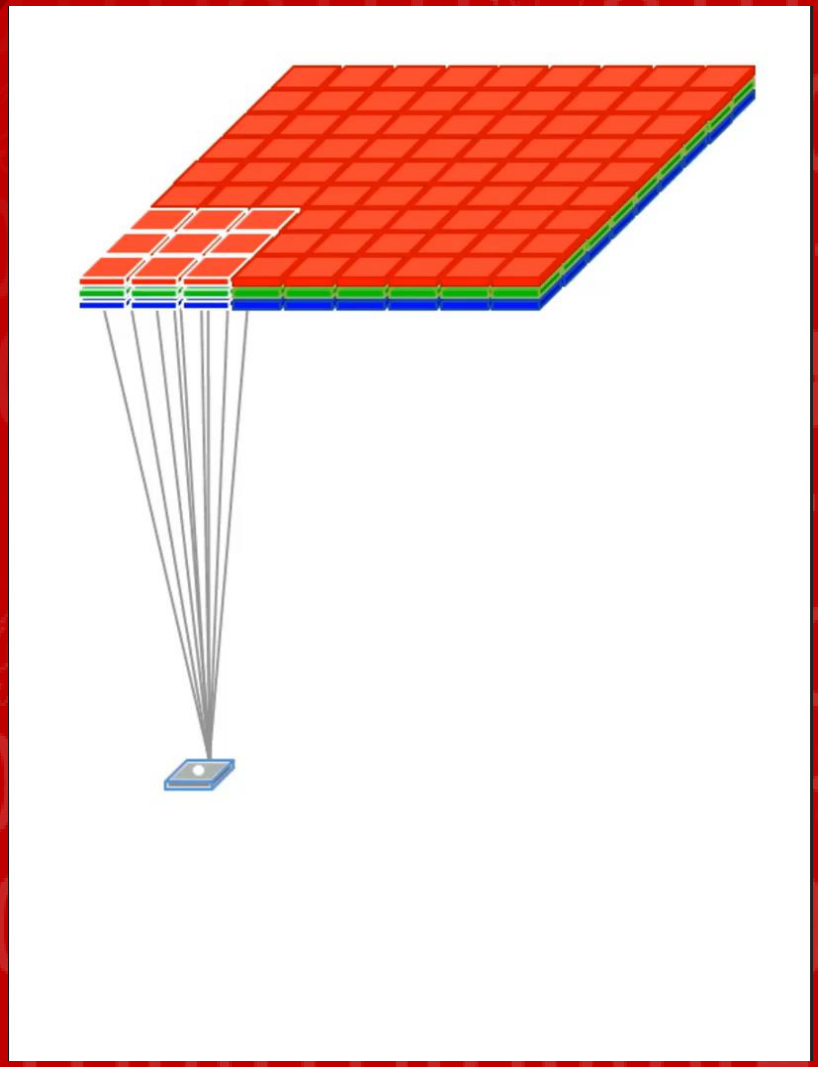
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

In [4]:

```
#
# Train and evaluate the network.
#
def train_and_evaluate(reader_train, reader_test, max_epochs, model_func):
    # Input variables denoting the features and label data
    input_var = C.input_variable((num_channels, image_height, image_width))
    label_var = C.input_variable((num_classes))

    # Normalize the input
    feature_scale = 1.0 / 256.0
    input_var_norm = C.element_times(feature_scale, input_var)

    # apply model to input
    z = model_func(input_var_norm, out_dims=10)

    #
    # Training action
    #

    # loss and metric
    ce = C.cross_entropy_with_softmax(z, label_var)
    pe = C.classification_error(z, label_var)

    # training config
    epoch_size     = 50000
    minibatch_size = 32

    # Set training parameters
    lr_per_minibatch       = C.learning_rate_schedule([0.01]*6 + [0.003]*3 +
                                                      C.UnitType.minibatch,
    momentum_time_constant = C.momentum_as_time_constant_schedule(-minibatch
    l2_reg_weight          = 0.001
```

Convolutional Neural Networks

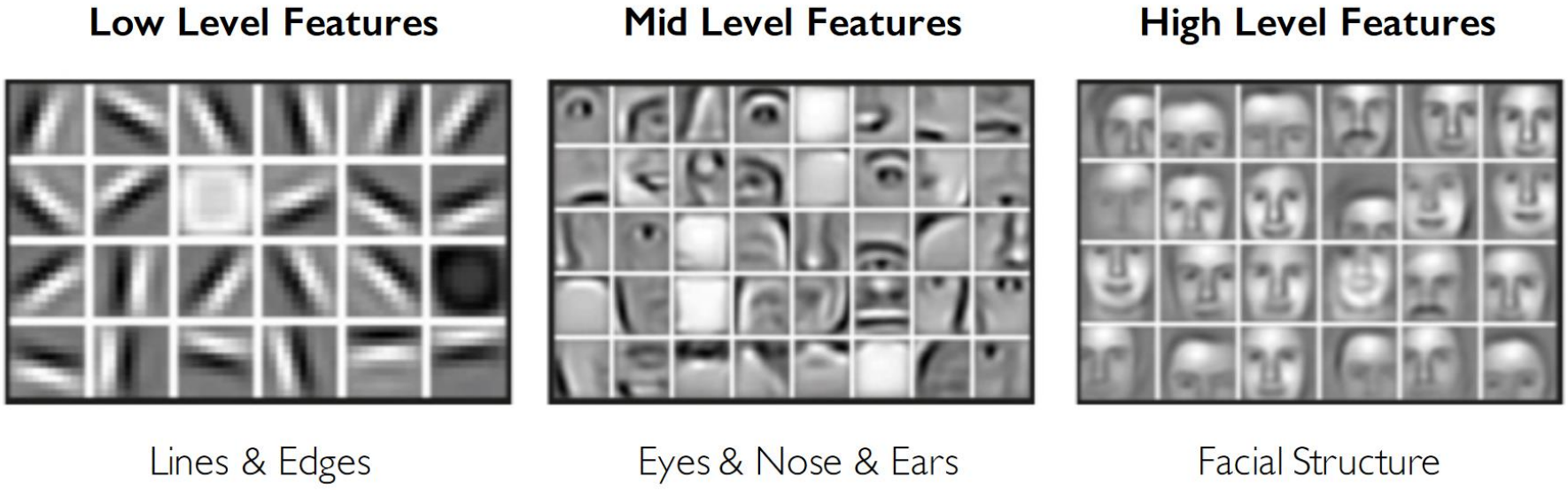


Neural Networks and images

- Regular Neural Nets don’t scale well to full images
  - For 32x32x3 images a single fully-connected neuron in a first hidden layer of a regular NN would have 32*32*3 = 3072 weights
  - 200x200x3, images would lead to neural net that have 200*200*3 = 120 000 weights in a single layer
- Regular Neural Nets don't preserve image spatial structure
  - Each image is unfolded into single vector
  - For 32x32x3 images we got a vector of 3 072 digits
  - For 200x200x images we got a vector of 120 000 digits

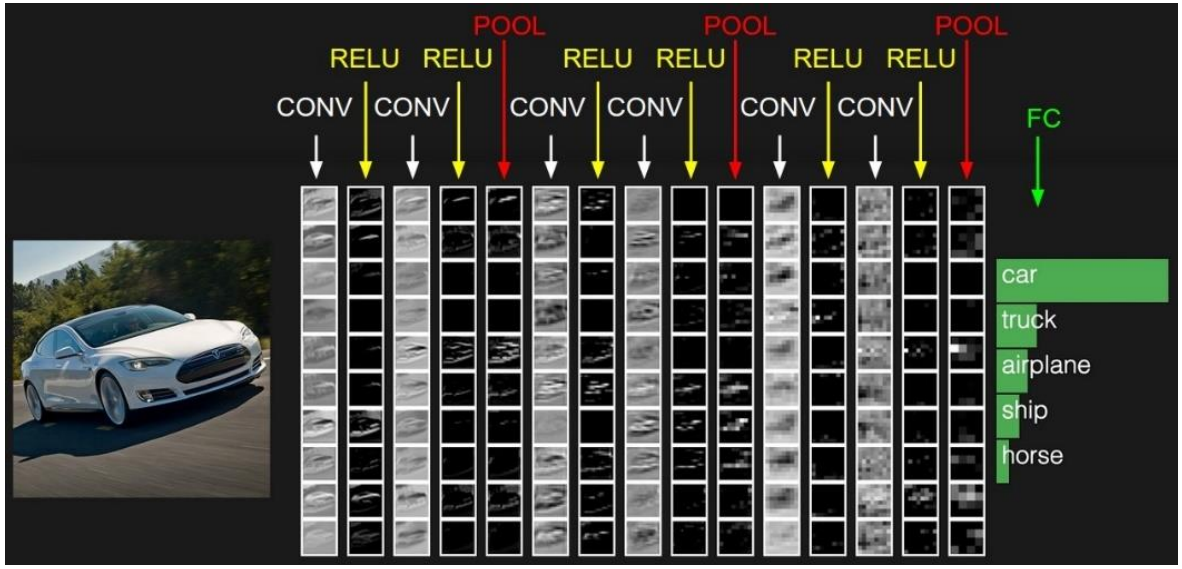
Why Deep Learing?

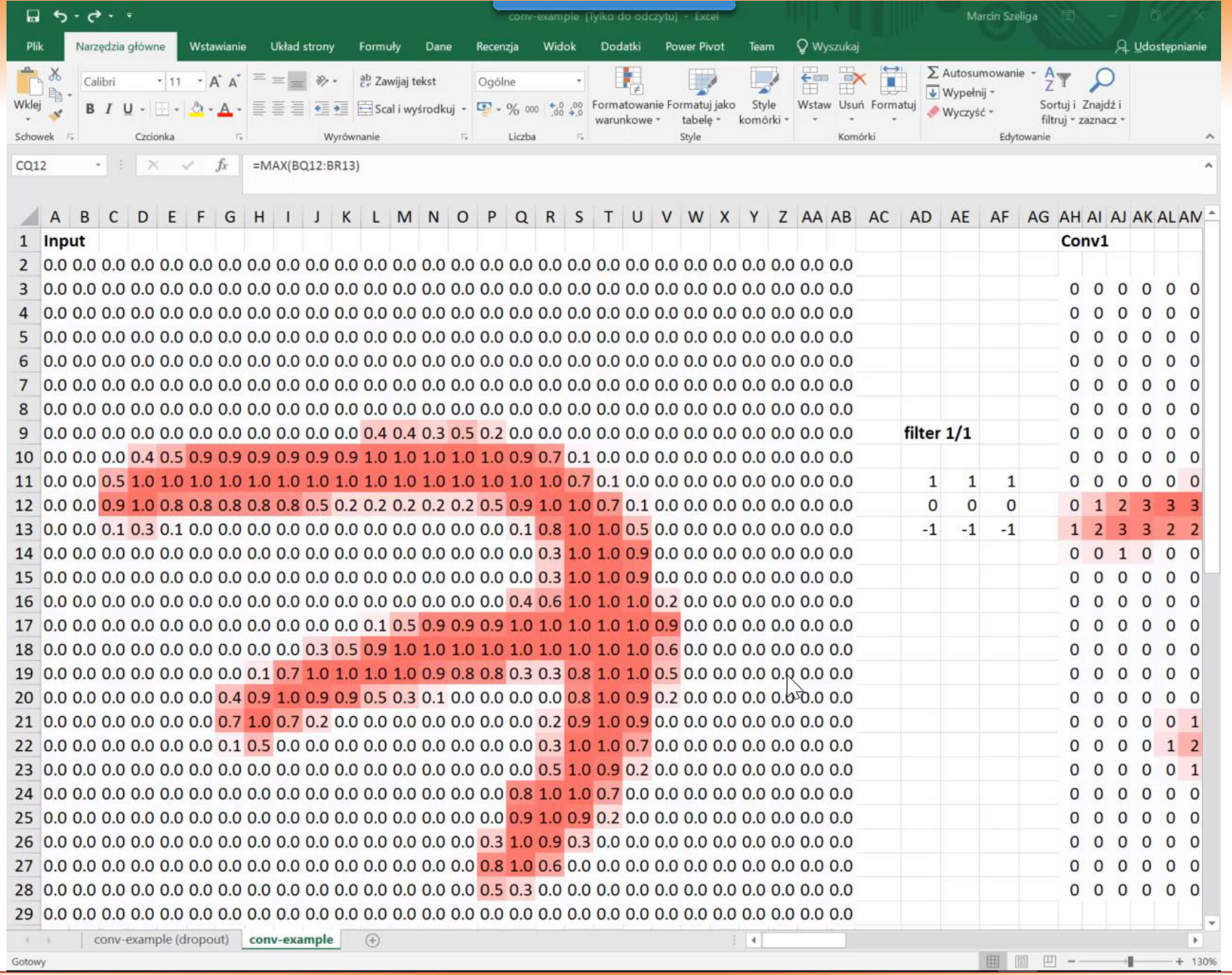
- Hand engineered features are time consuming, brittle and not scalable in practice
- Can we learn the underlying features directly from data?



Convolutional Neural Networks

- Layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth
- Neurons in a layer are connected only to a small region of the layer before it
- Three main types of layers
  - Convolutional Layer
  - Pooling Layer
  - Fully-Connected Layer

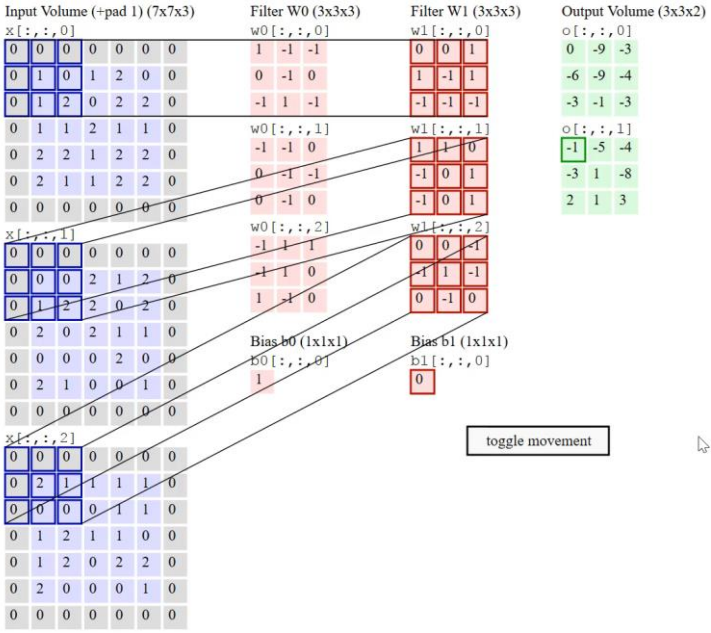


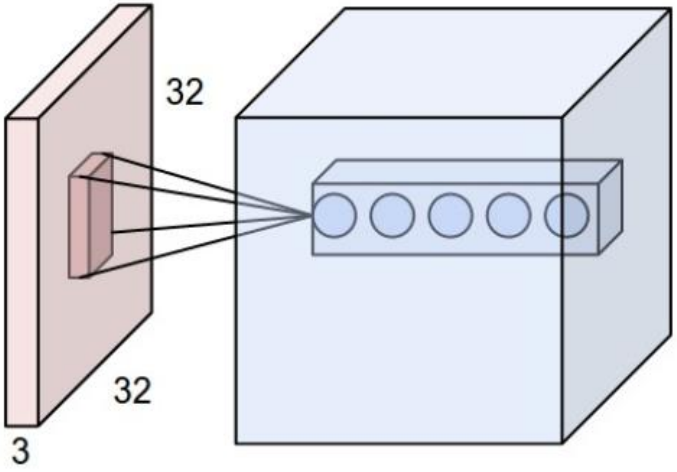


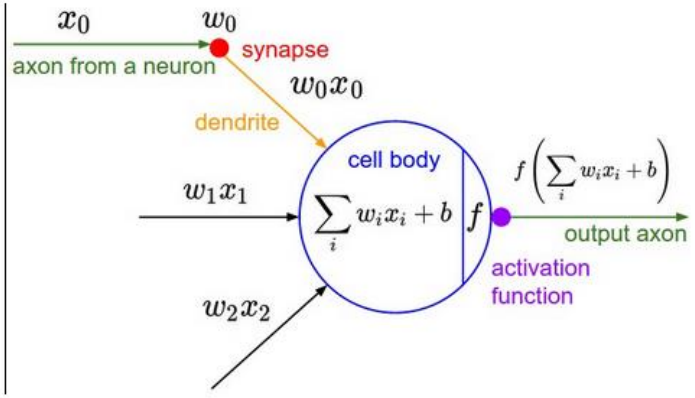
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM
1	**Input**																																	**Conv1**					
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0											
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.4	0.3	0.5	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		**filter 1/1**				0	0	0	0	0	0
10	0.0	0.0	0.0	0.4	0.5	0.9	0.9	0.9	0.9	0.9	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.7	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
11	0.0	0.0	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		1	1	1		0	0	0	0	0	0
12	0.0	0.0	0.9	1.0	0.8	0.8	0.8	0.8	0.8	0.5	0.2	0.2	0.2	0.2	0.2	0.5	0.9	1.0	1.0	0.7	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0	0	0		0	1	2	3	3	3
13	0.0	0.0	0.1	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.8	1.0	1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0		-1	-1	-1		1	2	3	3	2	2
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	1	0	0	0
15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.6	1.0	1.0	1.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
17	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.5	0.9	0.9	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
18	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.5	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
19	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.7	1.0	1.0	1.0	1.0	0.9	0.8	0.8	0.3	0.3	0.8	1.0	1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
20	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.9	1.0	0.9	0.9	0.5	0.3	0.1	0.0	0.0	0.0	0.0	0.8	1.0	0.9	0.2	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
21	0.0	0.0	0.0	0.0	0.0	0.0	0.7	1.0	0.7	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.9	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	1
22	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	1	2
23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	0.9	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	1
24	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	1.0	1.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9	1.0	0.9	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
26	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	0.9	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
27	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	1.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
28	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0						0	0	0	0	0	0
29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0											

Convolutional layer

- Set of learnable filters
- Every filter is small spatially, but extends through the full depth of the input volume
- Network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color

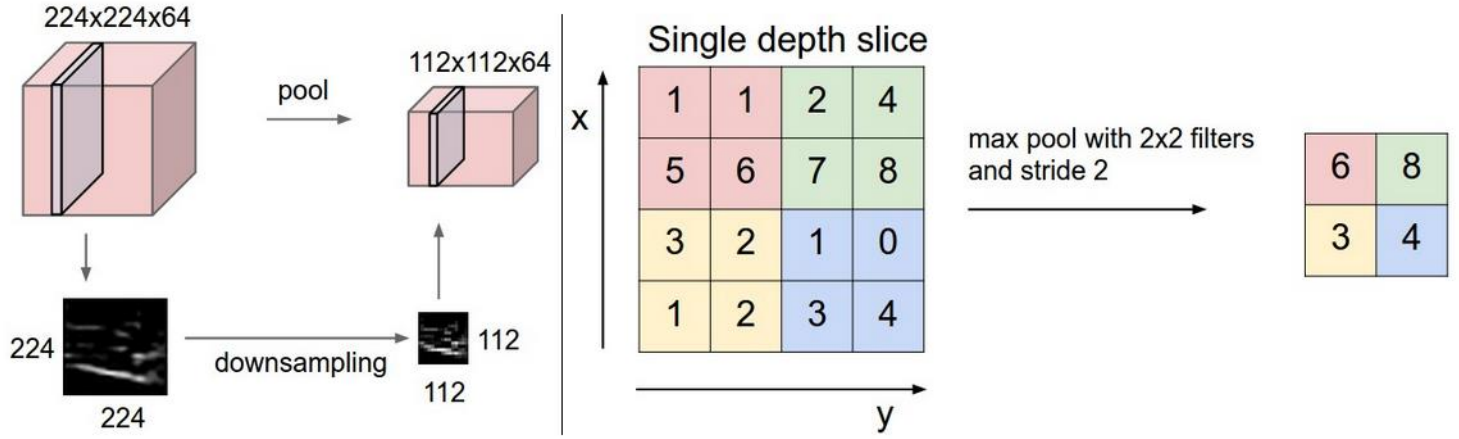






Pooling layer

- Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting
- Pooling layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation



Home | Image classification with Micro

localhost:8888/notebooks/Image classification with Micr 140% Szukaj

jupyter Image classification with Microsoft Cognitive Toolkit (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

Minibatch number

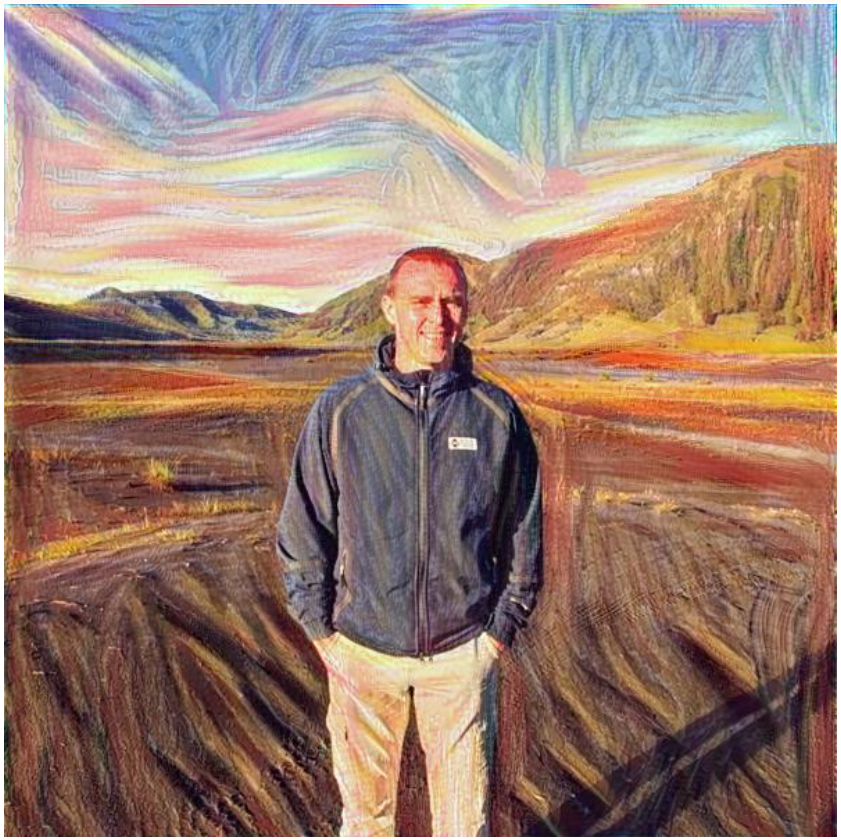
In [7]:
```
test_file = os.path.join(data_path, "gtr.png")
Image(filename=test_file)
```

Out[7]:

In [8]:
```
from PIL import Image as PILImage
myimg = PILImage.open(test_file)
myimg = myimg.resize((32,32), PILImage.ANTIALIAS)
myimg = np.array(myimg, dtype=np.float32)
```

In [9]:
```
def eval(pred_op, image_data):
    label_lookup = ["airplane", "automobile", "bird", "cat", "deer", "dog",
    image_mean = 128.0
    image_data -= image_mean
    image_data = np.ascontiguousarray(np.transpose(image_data, (2, 0, 1)))

    result = np.squeeze(pred_op.eval({pred_op.arguments[0]:[image_data]}))

    # Return top 3 results:
    top_count = 3
    result_indices = (-np.array(result)).argsort()[:top_count]

    print("Top 3 predictions:")
```

Image classification with Microsoft Cognitive Toolkit



Q&A

marcin.szeliga@datacommunity.pl
https://www.linkedin.com/in/marcinszeliga

- Get Andrew Ng course http://www.coursera.org/learn/neural-networks-deep-learning
- Practice using TensorFlow http://www.kaggle.com/learn/deep-learning
- Read „Deep Learning” by Bengio Yoshua, Courville Aaron, Goodfellow Ian
- Watch MIT course 6.S191: Introduction to Deep Learning http://introtodeeplearning.com/