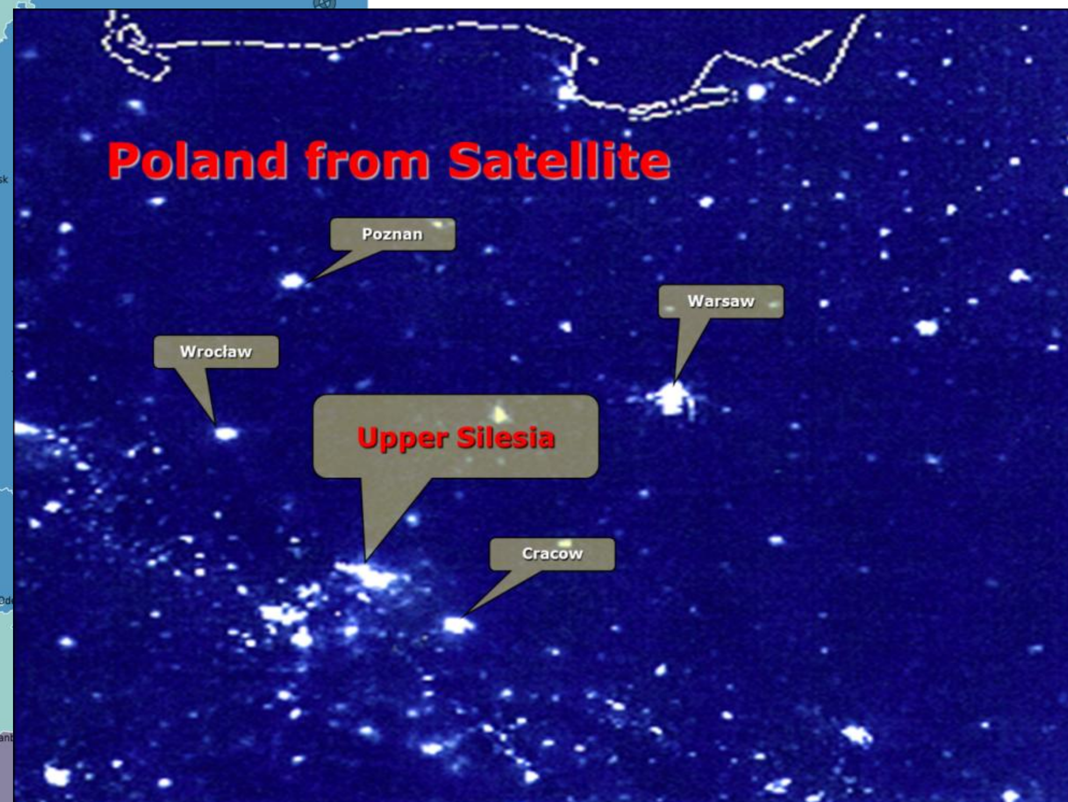
Deep Learning
in the
Eye Tracking World

Paweł Kasprowski
Silesian University of Technology
Gliwice, Poland

ETRA • 2019

About my home

Silesian University of Technology, Gliwice, Poland...?



https://wikitravel.org/en/Europe

Upper Silesian Industrial Region

14 cities, population about 3 mln

Politechnika Śląska

Silesian University of Technology
(Politechnika Śląska)

- Rankings among all universities in Poland
  - 2nd place in educating top managers
  - 4th place whose graduates are most desired by employers
  - 6th place among all higher education technology schools in Poland
- **13** Faculties
- More than:
  - **15,000** students
  - **600** PhD students
  - **3,150** employees
  - **150** full-professors

ETRA • 2019

My Faculty

Faculty of Automatic Control, Electronics and Computer Science

(AEI - "vowel faculty")

Majors:
Automatic Control
Electronics and Teleinformatics
Computer Science
Computer Science (English)
Macrofaculty (English)
Bioinformatics

Staff:
40 professors
160 associate professors and lecturers
>2000 students

Tutorial content

- Not so many formulas (only a few)
  - no "gory details" – just the ideas
- Very simple examples (max 100 lines of code)
  - just the working code (in Python/Keras)
- Simple datasets to show that it works (hopefully)

- *If you don't have any experience with Machine Learning you will be able to start your own experiments with your own data right after the tutorial*

Evironment preparation

- Install Python 3.x (64 bit)
  - add to path (installer gives this option)
- Install prerequisites:
  - pip install -r http://www.kasprowski.pl/tutorial/requirements.txt
- Download and unzip the tutorial code from

  http://www.kasprowski.pl/tutorial/code.zip
- or just use git (if you have it installed):

  git clone https://github.com/kasprowski/tutorial2019

Requirements

- scikit-learn
- tensorflow
- numpy
- opencv-contrib-python
- pygame
- thorpy
- matplotlib
- pandas

Code disclaimers

- Python is not my native language...
- KISS[1] over DRY[2] principle
  - a lot of code duplications
  - separate examples in separate files/folders
  - hopefully the code is self explanatory
  - not more than 120 lines per file

1) KISS – Keep It Simple Stupid
2) DRY – Don't Repeat Yourself

Tutorial agenda

- Introduction to Machine Learning
  - problems, algorithms, measures, examples
- Neural Networks
  - architectures, implementations, Keras/Tensorflow, examples
- Convolutional Neural Networks
  - idea, advantages, examples
- Recurrent Neural Networks
  - sequences, architectures, examples

Problems to be solved

- Classification
  - assign samples to predefined classes
  - [scan-path -> novice/expert]
- Regression
  - calculate a value for each sample
  - [eye image, glint -> gaze coordinates]
- Conversion
  - convert object into another object
  - [sequence of gazes -> sequence of events]

Task and method

- Task: find a function Y = f(X) where
  - X – **sample** (input)
  - Y – result – class, value (output) – **label**
- Method (learning by example):
  - take some number of samples X with known output Y (**examples**, labeled samples)
  - build the function based on these examples (learning process)
  - use the function to predict the label for unknown samples

Problem of "generality"

- It is relatively easy to create a function that correctly classifies all examples
- But is this function "general" enough?
  - is it able to correctly classify unknown samples?
- The answer is not trivial and that is why we have a lot of different classification methods
- "No free lunch" theorem

Over-fitting

- If the function (model) is optimized for the given data (given examples) it may have poor generality
- This problem is called over-fitting
- Solution is to simplify the model, e.g. stop learning even when it is not perfect for examples

The simplest algorithms

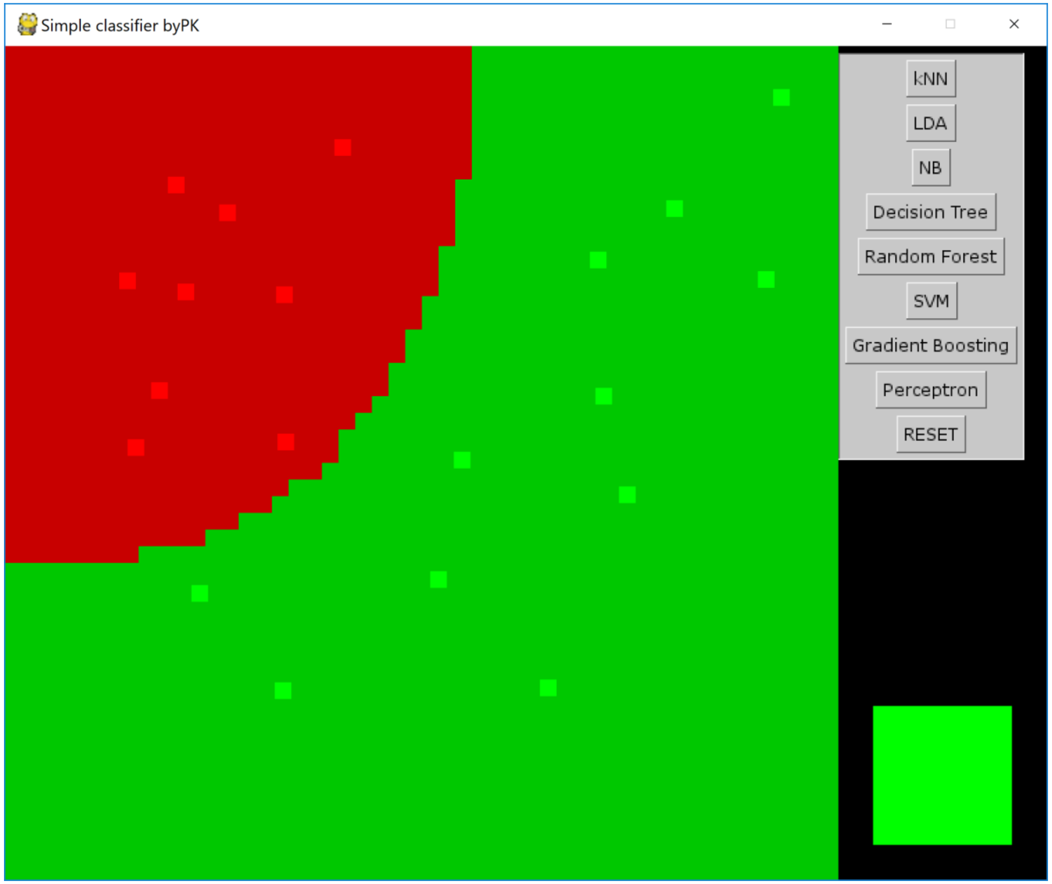
- K-Nearest Neighbors (kNN)
  - a new sample is classified as majority of its K nearest neighboring examples
- Linear Discriminant Analysis (LDA)
  - finds a linear equation separating positive and negative samples
- Naive Bayes
  - the class of a new sample is calculated based on examples using Bayes equation
- Decision Tree
  - the tree is built using examples by seaching for the most discriminative features
  - new samples are going through the tree reaching a leaf with the predicted class

More sophisticated algorithms

- Random Forest
  - a set of randomly generated decision trees voting for the final results
- Support Vector Machines (SVM)
  - recalculates examples into another space where they are easier to separate lineary (kernel trick)
- Neural Networks
  - based on neurons organized into layers

Example

Run: *classifier/main.py*



Hyper parameters

- Classification algorithms have parameters
  - so called hyper parameters
- kNN – the number of neighbors
- Random Forest – the number of trees
- SVM – kernel function, parameters of the function, learning rate etc.
- Good choice of hyper parameters influences the quality of the model!
- Bad news: deep learning methods have A LOT hyper parameters

Evaluating results

- Hyper parameters may be tuned
  - it is important to be able to check if the model is general
- First rule: never check the model using the same data that you used for training (examples)
  - over-fitting!
- The dataset should always be divided into:
  - training set (used to build the model)
  - test set (to check how it works for unknown samples)

Evaluation

- Dividing into training set and test set may be not enough!
  - we optimize the model (by tuning hyper parameters) for the given test set
- The safest way:
  - training set (for learning)
  - validation set (for hyper parameters tuning)
  - test set (for final evaluation)
    - the "holdout" dataset

Cross-validation

- Dynamic division into training and test
  - the same examples are sometimes training and sometimes test samples
- 10-fold cross validation:
  - divide the whole dataset into 10 subsets
  - for each subset
    - train the model using the remaining 9 subsets
    - test the results using the chosen subset
  - average the results
- Folds are randomized but may be stratified
  - the same distribution of classes in each fold

Multinomial classification

- More than two classes
  - E.g. fixations, saccades, smooth-pursuits and glissades
- The model returns a probability for each class
- We choose the class with the highest probability
- Typical output:
  - binary class matrix (one-hot encoding)
    - 1 -> [1,0,0,0,0]
    - 2 -> [0,1,0,0,0]
    - 3 -> [0,0,1,0,0]
    - ...

Measures

- Accuracy – the number of correctly classified samples to all samples
  - a problem with unbalanced sets
  - if 90% of test samples is positive the blind classifier achieves 90% accuracy
- Precision – the cost of false positives
  - not belonging to class predicted as belonging
- Recall – the cost of false negatives
  - belonging to class predicted as not belonging
- F1-Score – combination of precision and recall
- Cohen's kappa – compares prediction with random prediction

Measures interpretation

- Example: We have built the model that diagnoses autism based on eye movements
- For every eye movement sample the model returns:
  - ***H*** - if a person is healthy
  - ***S*** - when a person is sick with autism
- We test the model using
  - 18 samples from sick people (***S***)
  - 82 samples from healthy people (***H***)
- We achieve 82% accuracy
  - is it bad or good?
  - it depends: we must examine a confusion matrix

ETRA • 2019

Confusion matrix

Predicted as >> Real class	H	S
H	81	1
S	17	1

P	N
TP	FN
FP	TN

Accuracy 0.82

H:
precision: 0.83
recall: 0.99
F1-score: 0.90

S:
precision: 0.50
recall: 0.06
F1-score: 0.1

Cohen's kappa: 0.07

Measures

Predicted as >> Real class	H	S
H	80	2
S	15	3

P	N
TP	FN
FP	TN

Accuracy 0.83

H:
precision: 0.84
recall: 0.98
F1-score: 0.90

S:
precision: 0.60
recall: 0.17
F1-score: 0.26

Cohen's kappa: 0.20

Measures

Predicted as >> Real class	H	S
H	66	16
S	2	16

P	N
TP	FN
FP	TN

Accuracy 0.82

H:
precision: 0.97
recall: 0.80
F1-score: 0.88

S:
precision: 0.50
recall: 0.89
F1-score: 0.64

Cohen's kappa: 0.53

Measures

Predicted as >> Real class	H	S
H	64	18
S	0	18

P	N
TP	FN
FP	TN

Accuracy 0.82

H:
precision: 1.00
recall: 0.78
F1-score: 0.88

S:
precision: 0.50
recall: 1.00
F1-score: 0.67

Cohen's kappa: 0.56

Example for three classes

Confusion matrix

[[24 3 9]
 [3 24 24]
 [2 15 49]]

class	precision	recall	f1-score	support
0	0.83	0.67	0.74	36
1	0.57	0.47	0.52	51
2	0.60	0.74	0.66	66
micro avg	0.63	0.63	0.63	153
macro avg	0.67	0.63	0.64	153
weighted avg	0.64	0.63	0.63	153

Accuracy: 0.63

Summary

- Accuracy is often not enough
  - esp. for unbalanced datasets
- Cohen's kappa is the easiest way to evaluate results
  - but has some drawbacks as well!
- The evaluation depends on the target we want to achieve
  - False Acceptance Rate (FAR=0)
    - no incorrectly classified healthy people
  - False Rejection Rate (FRR=0)
    - no incorrectly classified sick people

Regression

- The model does not search for one of N classes but for a value
  - E.g. gaze coordinates
- Examples:
  - Linear Regression
  - Support Vector Regression
  - Decision Tree Regression
  - Neural Network Regression

Evaluation of regression

- Mean Absolute Error
- Mean Squared Error
  - penalizes predictions differing greatly (outliers)
- Root Mean Squared Error
  - comparable to real values
- Coefficient of Determination (R^2)
  - value 0-1

Example: Linear Regression

- Given a training set of pairs (X,Y) find coefficients of a linear function

$$y = a * x + b$$

- Error function Mean Squared Error

$$E = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i}^{n}(y_i - (ax_i + b))^2$$

- Goal: minimize this function

Linear Regression using Gradient Descent

- Gradient Descent algorithm:

  initialize randomly a and b

  repeat:

  calculate error

  change a and b in such way that error is smaller

  until error is small enough or it is not decreasing

- Question: how to find the direction of change for ***a*** and ***b***?

Gradient Descent

- Gradient dE/da – a function describing how ***E*** changes when ***a*** changes

$$\frac{\delta E}{\delta a} = \frac{-2}{n}\sum_{i}^{n}(\hat{y}_i - y_i)x_i$$

- Gradient dE/db - function describing how ***E*** changes when ***b*** changes

$$\frac{\delta E}{\delta b} = \frac{-2}{n}\sum_{i}^{n}(\hat{y}_i - y_i)$$

Gradient Descent algorithm

- for each step (iteration)
  - calculate error E
  - calculate gradients dE/da and dE/db
  - a = a - learning rate * dE/da
  - b = b - learning rate * dE/db
- Every iteration gives better values of a and b
  - error (**loss**) is smaller
- Learning rate controls how fast the parameters are updated
  - too fast – we may pass the minimum
  - too slow – we may wait forever

Analytical solution

- For linear regression the best parameters may be calculated using equations:

$$a = \frac{\sum_i^N (x_i - \bar{x})(y_i - \bar{y})}{\sum_i^N (x_i - \bar{x})^2}$$

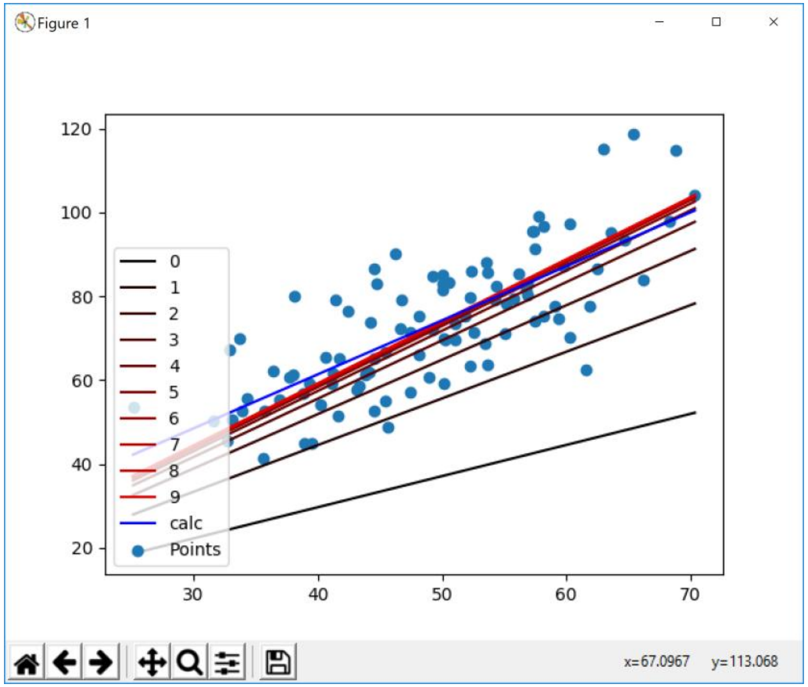
$$b = \bar{y} - a\bar{x}$$

- but the presented example shows how Gradient Descent algorithm works (and we will use this algorithm later!)

Example

Run: *linear/linear_descent.py*

- Simple code calculating linear regression using gradient descent for a set of points



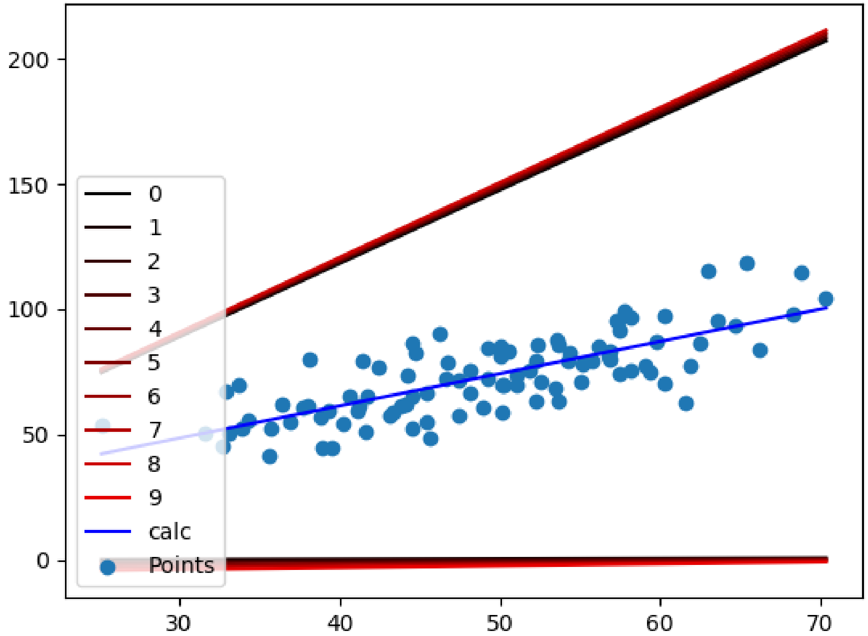
Observations

- If we change initial values of a and b we have different results
  - b is reluctant to change
  - kind of "vanishing gradient" problem
- (1) Change the learning rate for b:
  - nb = b - L*100 * D_b # Update b
- (2) Change the number of iterations
  - epochs = 100 (1000?)
- The results are better!

ETRA • 2019

Learning rate

- Change to:
  - L = 0.0004 # The learning rate
- Results jump over the solution
  - too big step!



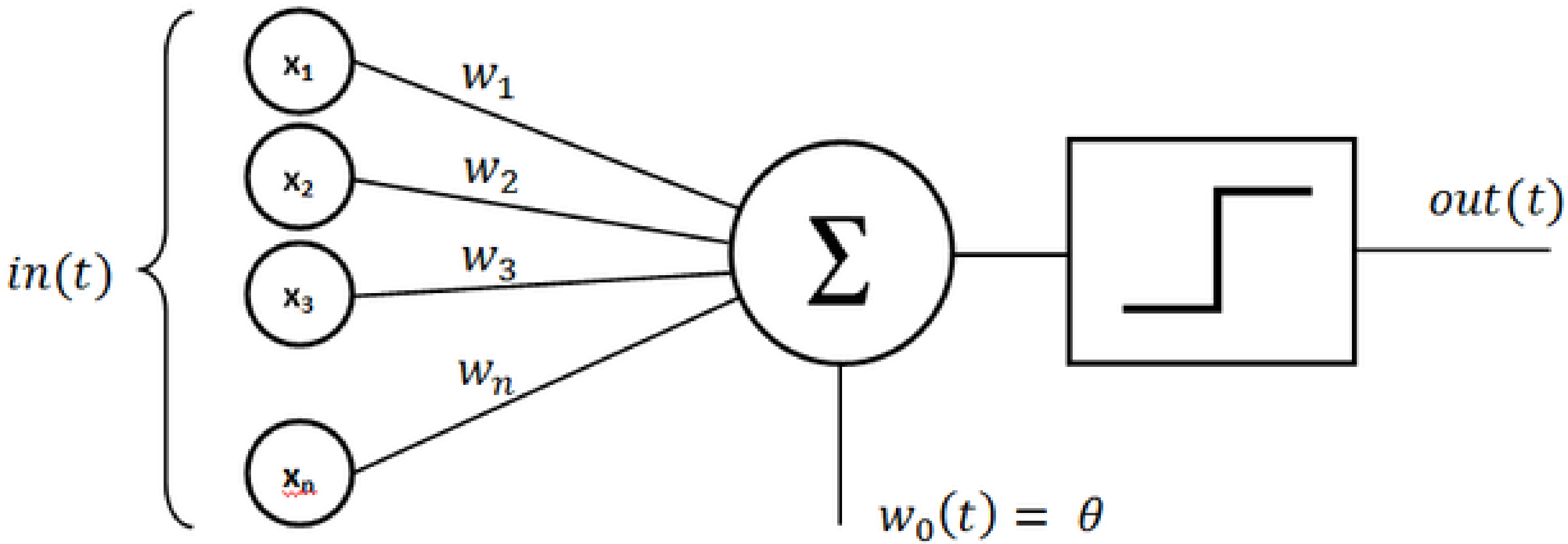
Summary

- There are many different classification algorithms
- There are many ways to tune these algorithms
- There are many measures to estimate the quality of classification/regression
- So what is so special about the deep learning methods?
  - Wait for the following sections!

Neural Networks

Introduction and implementation (Keras)

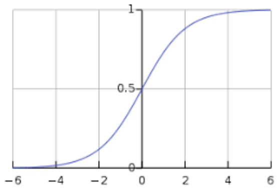
Perceptron (Rosenblatt 1957)

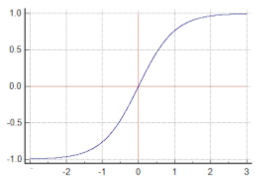


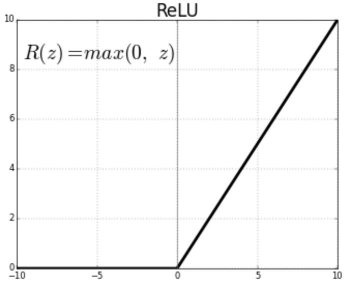
https://pl.wikipedia.org/wiki/Perceptron

The simplest form

- Linear combination of inputs
- Activation functions may be different
  - threshold function
  - sigmoid function
  - tanh function
  - RELU

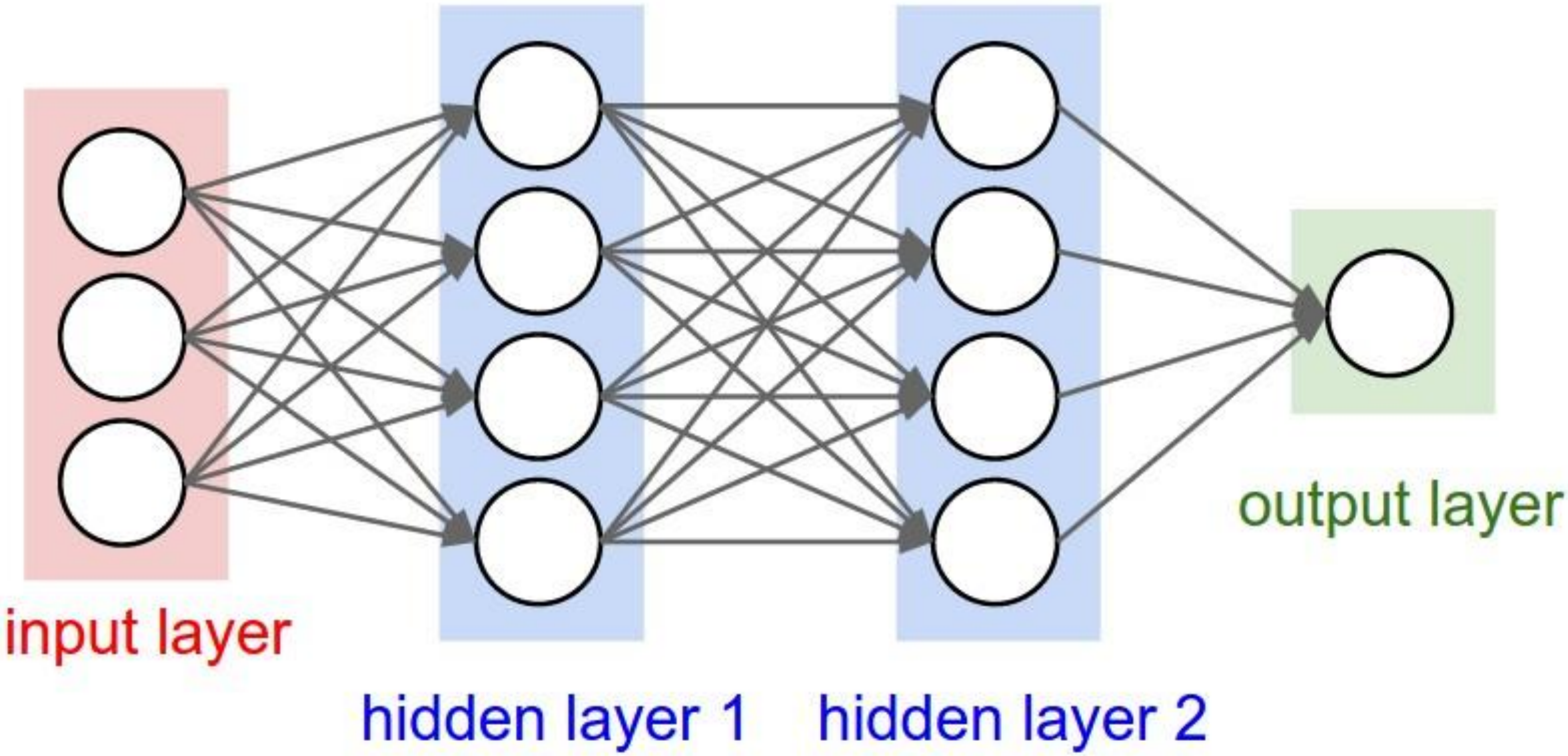






Neural Network

- input layer > hidden layers > output layer



http://cs231n.github.io/convolutional-networks/

Training the network

- Back propagation
  - the state-of-the art method to train the network
  - utilizes the already mentioned Gradient Descent algorithm
- The algorithm:

  initialize weights

  repeat many times:

    use network to calculate output (Y_{pred}) for some examples (X)

    calculate error (loss) using Y_{pred} and Y (real)

    update weights to minimize loss

Tuning - hyper parameters

- Network structure
  - Number of layers
  - Number of neurons for layer
  - Connections
  - Activation functions for layers
- Loss function
- Optimization algorithm
  - how to change the weights
  - learning rate (how big changes of weights)

Implementations

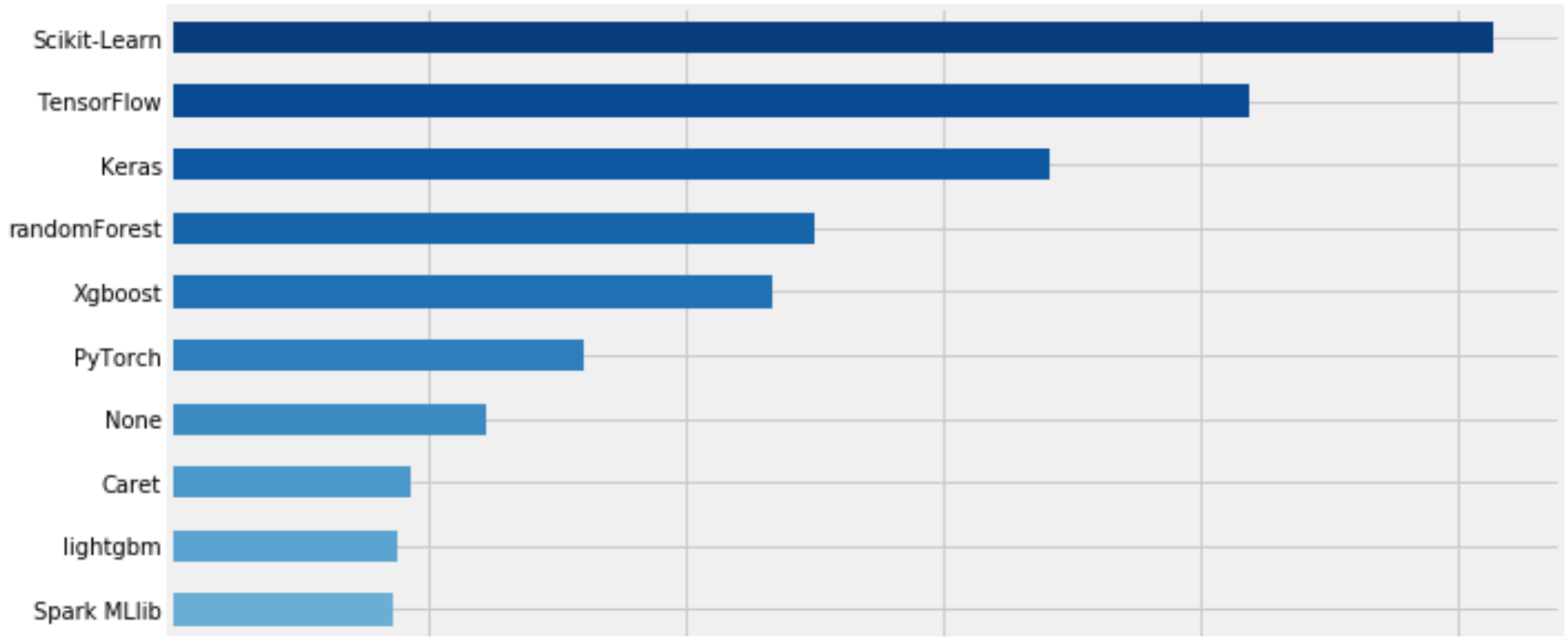
- Many classification libraries like scikit-learn or WEKA implement the neural networks
  – but only the simplest models
- For Deep Learning there are many libraries developed by leading companies:
  – Tensorflow, Google
  – PyTorch, Facebook
  – CNTK, Microsoft
  – ...
- We will use the simplest to use: KERAS

Keras

- General interface to use deep networks
- Works with Tensorflow, Theano, CNTK...
- User-friendly, modular, and extensible
- Created in Google
- Works in Python
- Tensorflow library includes Keras by default
  - that is why we installed Tensorflow and we use Keras

Kaggle 2018 survey

- What machine learning frameworks have you used in the past 5 years?



https://www.kaggle.com/antgoldbloom/2018-kaggle-survey-eda-platforms-tools-and-more

KDNuggets 2018 poll

- Increase in usage 2017-2018

Table 2: Major Analytics/Data Science/ML Tools with the largest increase in usage

Tool	% change	2018 % share	2017 % share
Keras	108%	22.2%	10.7%
PyTorch	92%	6.4%	3.4%
Amazon Machine Learning	74%	3.3%	1.9%
RapidMiner	65%	52.7%	31.9%
Other free analytics/data mining tools	53%	8.3%	5.4%
DeepLearning4J	39%	3.4%	2.4%
Anaconda	37%	33.4%	24.3%
PyCharm	33%	13.5%	10.1%
Tensorflow	32%	29.9%	22.7%
Excel	24%	39.1%	31.5%
Tableau	21%	26.4%	21.8%

https://www.kdnuggets.com/2018/05/poll-tools-analytics-data-science-machine-learning-results.html

Keras basics

- Build the model (network)
  - contains layers with neurons and connections
- Compile the model (model.compile)
  - define loss function and optimizer
- Train the model (model.fit)
  - provide samples with known labels
- Use the model for predictions (model.predict)
  - predict labels of unknown samples

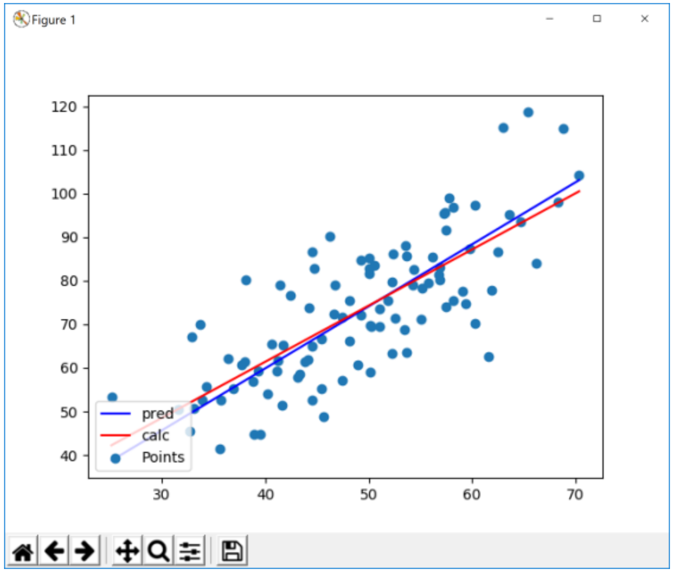
The simplest example

```
# import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# build model
model = Sequential()
model.add(Dense(1, activation='sigmoid'))
# compile model
model.compile(loss='binary_crossentropy',
    optimizer="adam",metrics=['accuracy'])
# train model
model.fit(samples, labels, epochs=100, batch_size=10)
# use model
predicted = model.predict(sample)
```

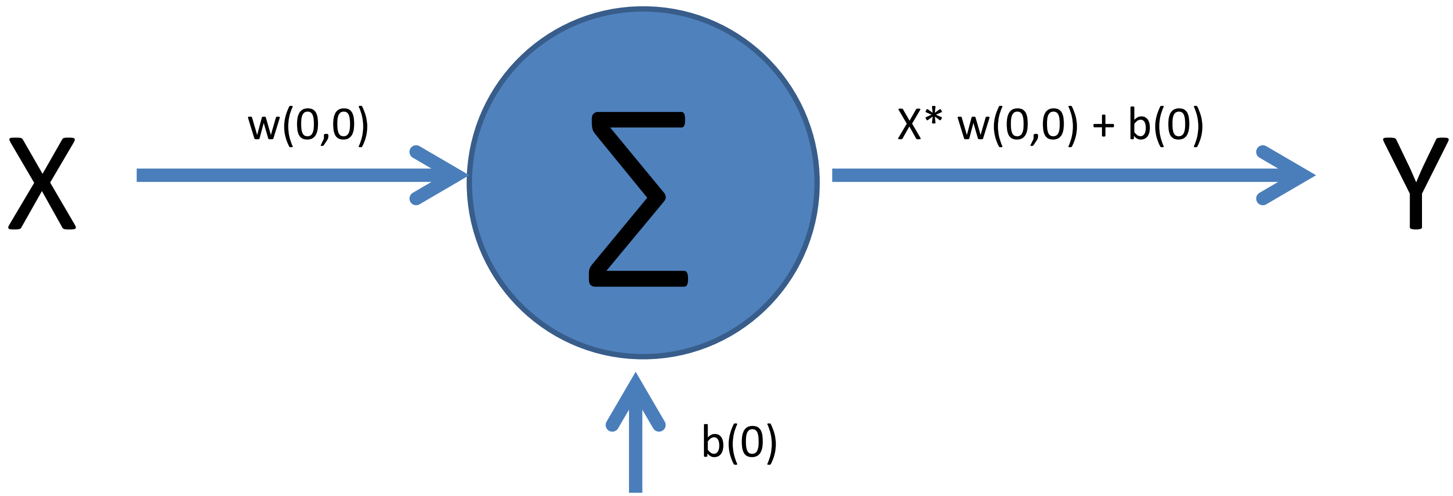
Backprop example

Run: *linear / linear_keras.py*

- Linear regression using back prop
- One input, one output
- Two weights: w_{00} and b_0 (as a and b)



Our VERY simple network...



Model architecture

- Add a hidden layer

```
model = Sequential()
model.add(Dense(50, input_dim=2))
model.add(Dense(1, activation='sigmoid'))
```

- Add a nonlinear activation

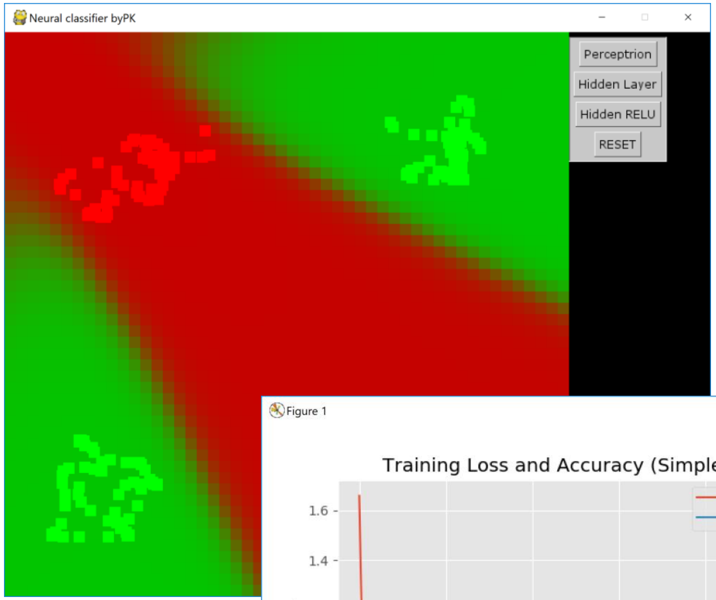
```
model = Sequential()
model.add(Dense(50, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

ETRA • 2019

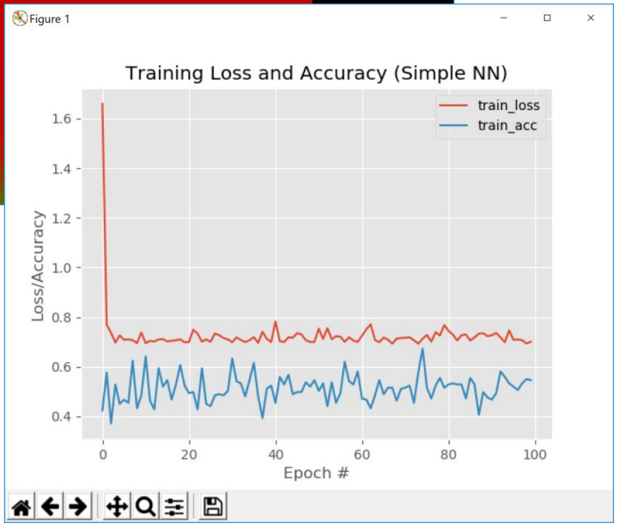
Example

Run: *neural/main.py*

- Several architectures
- Works with XOR
- Perceptron network
  - only linear
- RELU – much better



- Plot:
  - number of iterations
  - change of loss and accuracy



model.compile parameters

- loss – loss function
  - *binary_crossentropy* for 2-class problem
  - *categorical_crossentropy* for many classess (one-hot encoded)
  - *mean_squared_error* for regression
- optimizer – algorithm for back propagation
  - adam
  - SGD
- metrics – metrics to measure after each iteration
  - *accuracy* for classification
  - *mean squared error* for regression

ETRA • 2019

model.fit parameters

- samples – array of samples
- labels – array of labels (one for each sample)
- epochs – number of iterations
- batch_size – number of samples per batch (each back propagation pass)
  - stochastic (1 – backprop after every sample)
  - mini-batch (from 2 to N-1)
  - batch (N – backprop after all samples – one per epoch)
- validation_split – percent of validation samples
- validation_data = (samples, labels)

Input parameters

- samples – a list of samples
  - every sample may be multidimensional (e.g. 3D for images)
- labels – a list of correct labels for each sample
  - a list of values [0,1,4,2,1,4,2...]
  - one-hot encoded:
    - 0 > [1,0,0,0,0]
    - 1 > [0,1,0,0,0]
    - 4 > [0,0,0,0,1]

Real data at last!

- Identification of people based on their eye movements
- Part of EMVIC 2012[1] dataset:
  - 155 samples
  - 3 classes (persons)
  - 16384 attributes (position-vel-acc)
- unbalanced distribution
  - {'a25': 104, 'a41': 39, 'a37': 11}

[1] Kasprowski, Komogortsev, Karpov: First eye movement verification and identification competition at BTAS 2012,
IEEE Fifth International Conference on Biometrics: Theory, Applications and Systems (BTAS)

Loading and preparing data

- Loading

```
dataframe = pandas.read_csv(file)
dataset = dataframe.values
samples = dataset[:,1:]
labels = dataset[:,0]
```

- Preparing – "one hot" encoding

```
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
classesNum= labels.shape[1]
```

The model

Run: *mlp/main.py*

- Simple MLP model:

```
model = Sequential()
model.add(Dense(250, activation='sigmoid'))
model.add(Dense(250, activation='sigmoid'))
model.add(Dense(250, activation='sigmoid'))
model.add(Dense(classesNum, activation='softmax'))
model.compile(loss= 'categorical_crossentropy',
    optimizer="adam", metrics=['accuracy'])
```

Training, testing, reporting

- Training:

```
H = model.fit(trainSamples, trainLabels, batch_size=BATCH,
    epochs=EPOCHS)
```

- Testing:

```
mlpResults = model.predict(testSamples)
```

- Reporting (using methods from sklearn):

```
print(confusion_matrix(testLabels.argmax(axis=1),
    mlpResults.argmax(axis=1)))
print(classification_report(testLabels.argmax(axis=1),
    mlpResults.argmax(axis=1),target_names=lb.classes_))
```

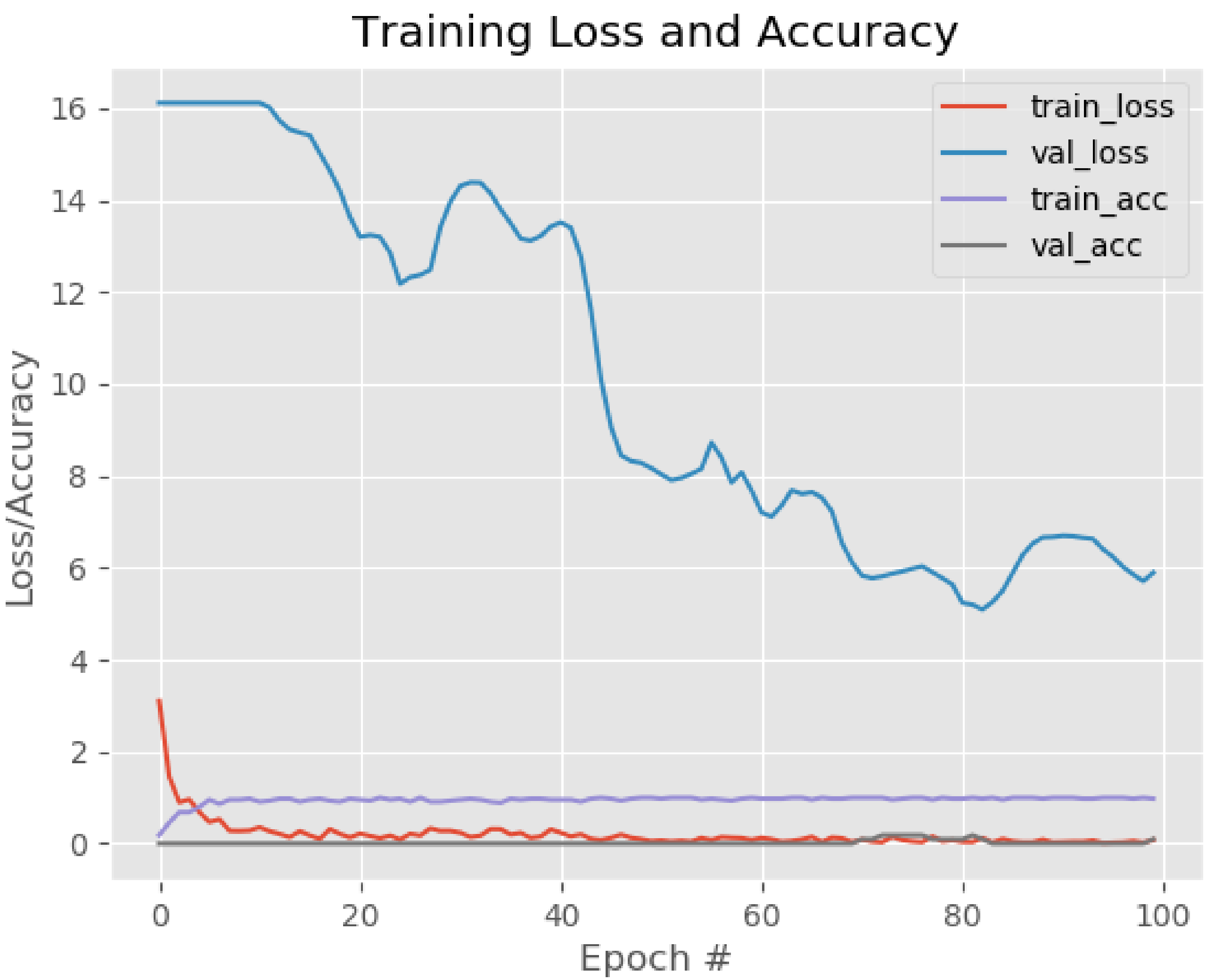
Validation

- Prevents from over-fitting
- Parameters of model.fit
  - validation_split=0.1
  - validation_data=(testSamples,testLabels)
- Division to training and testing
  - (trainSamples, testSamples, trainLabels, testLabels) = train_test_split(samples, labels, test_size=0.25)
- Test only using samples not used for training!

Training results

ETRA • 2019

It may be worse...



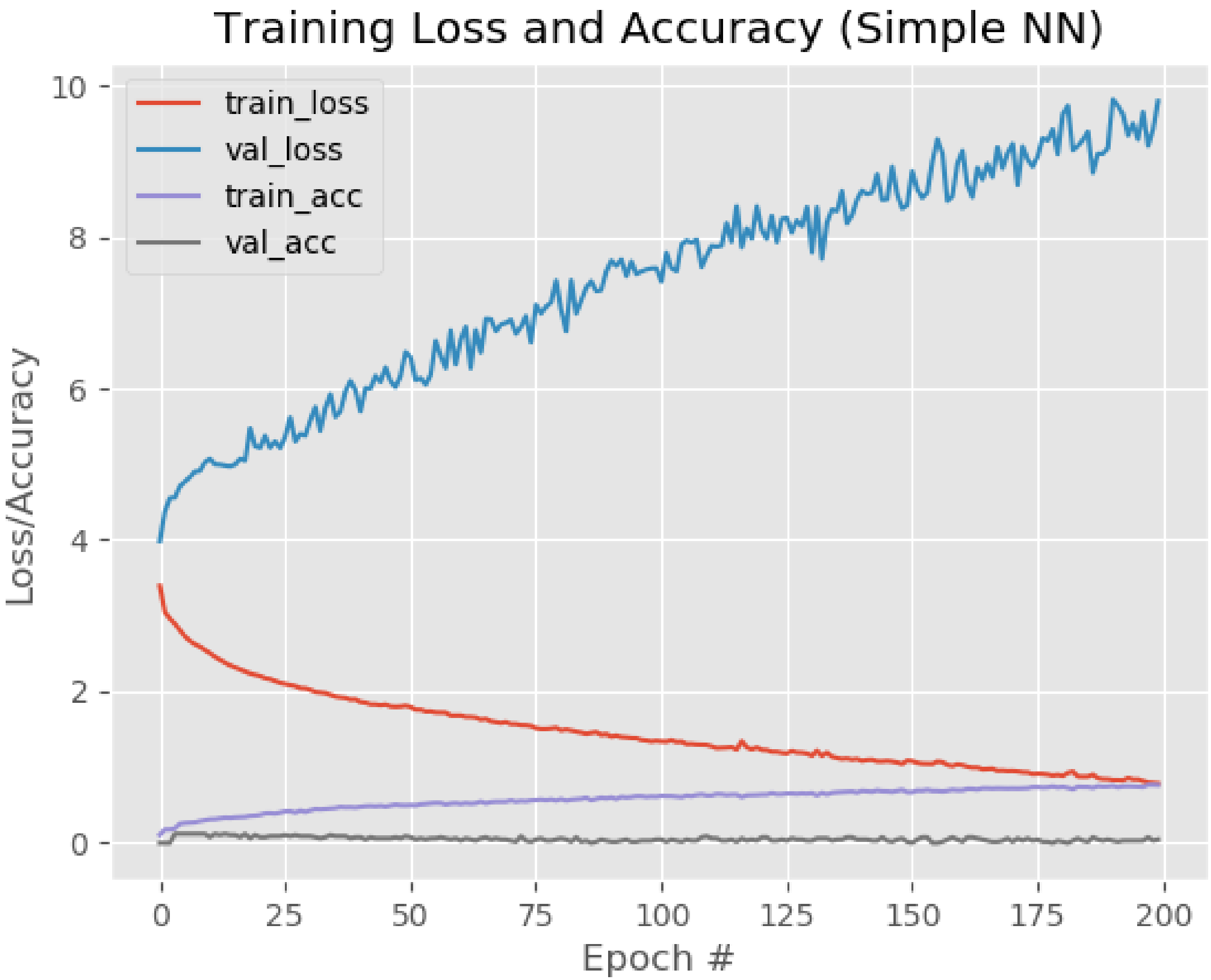
ETRA • 2019

It may be worse...

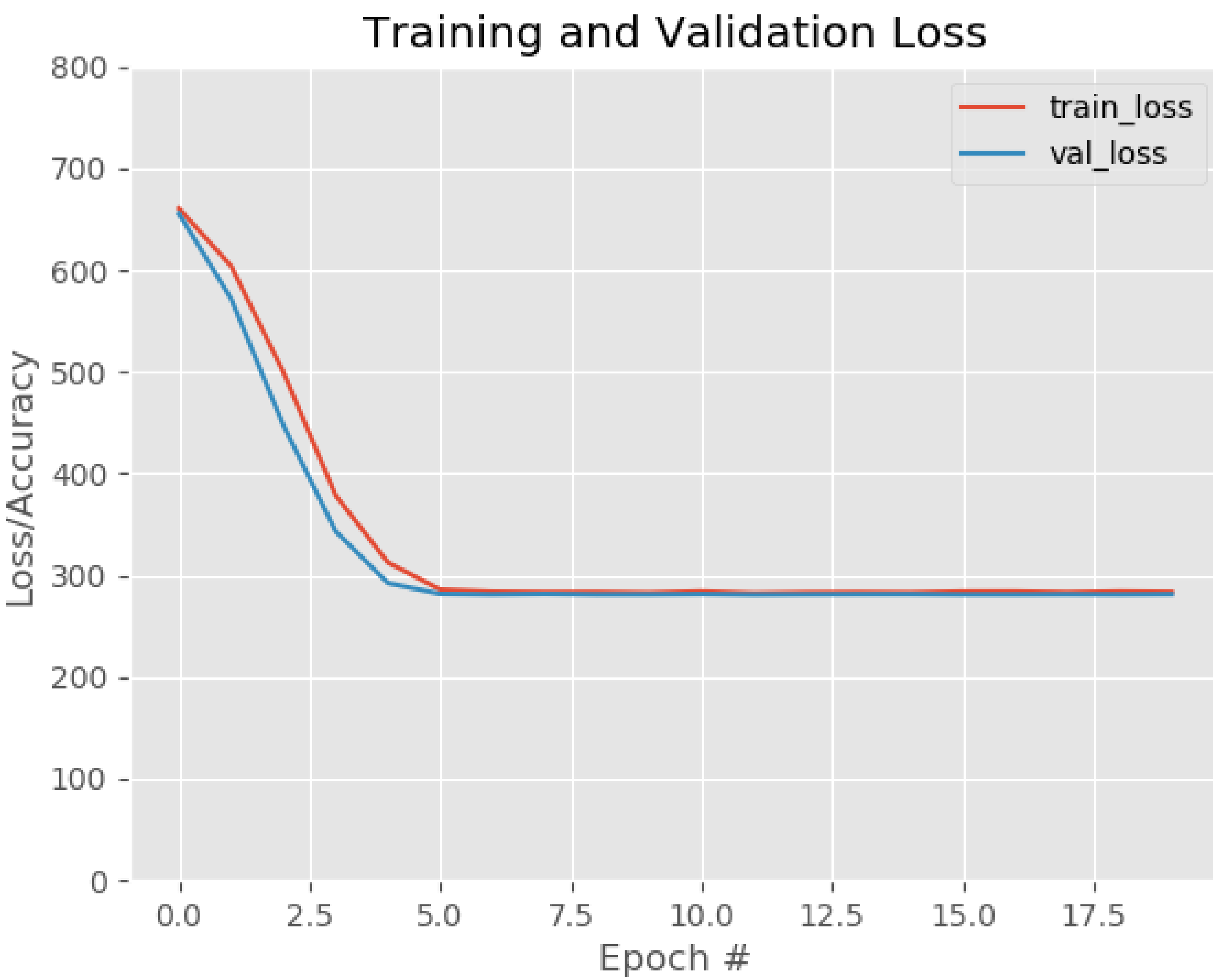
It may be worse...

It may be even worse...

It may be even worse...



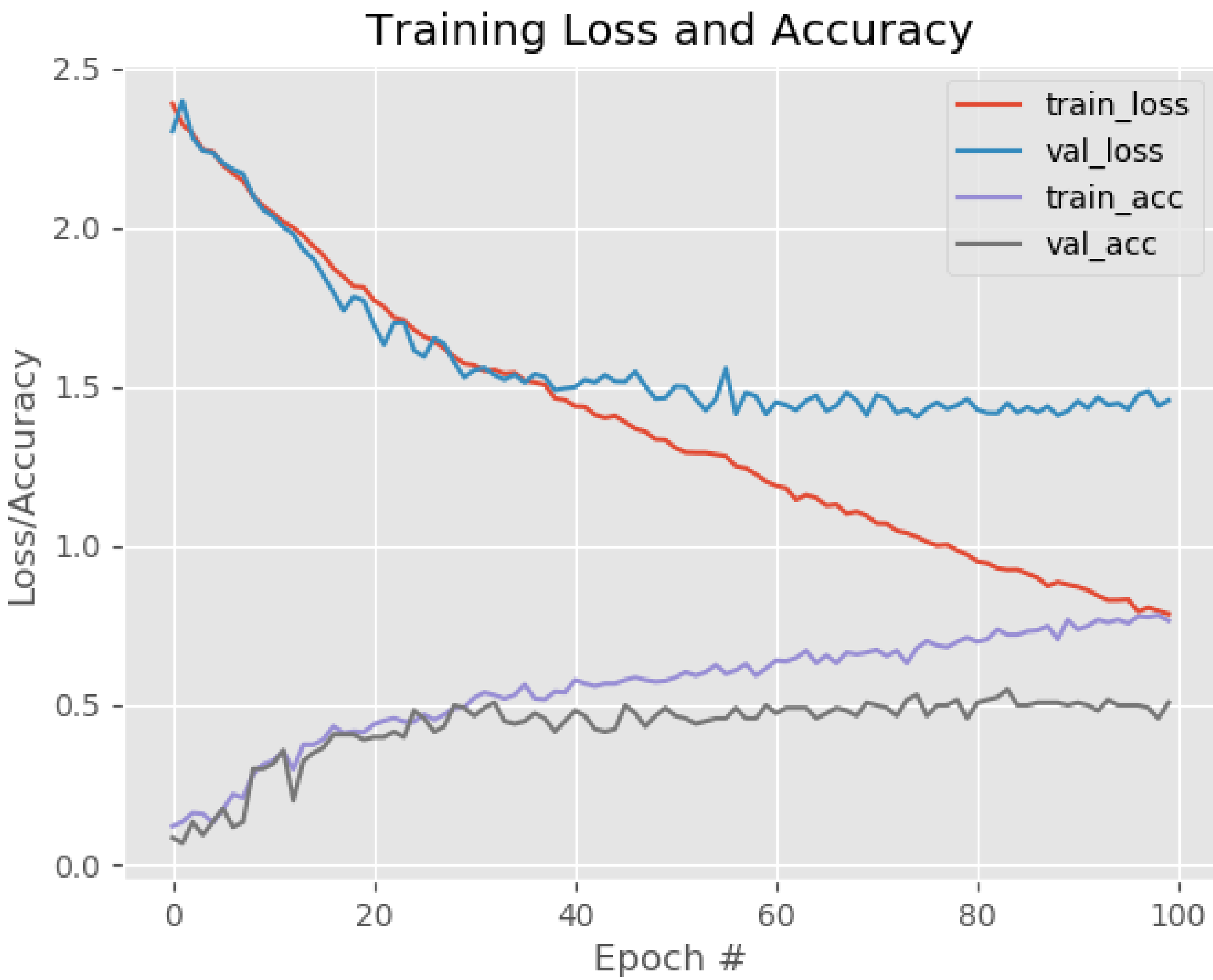
Sometimes even loss is not decreasing...



So ,this is quite OK

ETRA·2019

This is worse



pawel@kasprowski.pl

Possible additions (sklearn)

- Choose 100 best attributes

```
newSamples = SelectKBest(k=100).fit_transform(samples, labels)
```

- Add weights to classes

```
class_weights = class_weight.compute_class_weight('balanced',np.unique(labels),labels)
d_class_weights = dict(enumerate(class_weights))
```

- Normalize the dataset

```
normalize(samples)
```

Comparison with Decision Tree

- Code for Decision Tree:

```
from sklearn.tree import DecisionTreeClassifier
treemodel = DecisionTreeClassifier()
treemodel.fit(trainSamples, trainLabels)
treeResults = treemodel.predict(testSamples)
```

- Results:
  - Comparable...

Problems with neural networks

- Configuration is complicated (many hyper parameters)
  - layers, number of neurons
  - activation functions
  - optimizers
- Training is challenging
  - A lot of computations – a lot of weights
  - A lot of examples needed
- Training of many layers may fail
  - Vanishing gradient problem
  - Exploding gradient problem

Question for today

- So why Deep Learning has become so popular?
- Next part is about it!

Convolutional Neural Networks

Going really deep...

Curse of dimensionality

- If the number of features is too big we have the 'curse of dimensionality' problem
  - number of possible 'states' is too big to find any similarities in data
- Typical dataset:
  - A lot of irrelevevant features
  - Correlations between features
- For example:
  - Images and pixel values as features
  - 10000 features for 100x100 images!

Feature extraction

- All previously mentioned algrorithms treat input as a set of independent features
  - preferably not correlated
- So the first and most important part of classification is **feature extraction** from real data
- For instance in image classification it could be [1]:
  - **Color** - Color Channel Statistics (Mean, Standard Deviation) and Color Histogram
  - **Shape** - Hu Moments, Zernike Moments
  - **Texture** - Haralick Texture, Local Binary Patterns (LBP)
  - **Others** - Histogram of Oriented Gradients (HOG), Threshold Adjancency Statistics (TAS)

[1] https://gogul09.github.io/software/image-classification-python

Convolutional Neural Networks

- Network that is not "dense"
  - neuron from layer N+1 is not connected to every neuron in layer N
- It preserves "local connectivity"
  - Features (pixels) close to each other are processed together
- Such a network is in fact doing automatic feature extraction in input layers
- Two key properties:
  - not all neurons are connected
  - there are common weights for many connections

Brief history

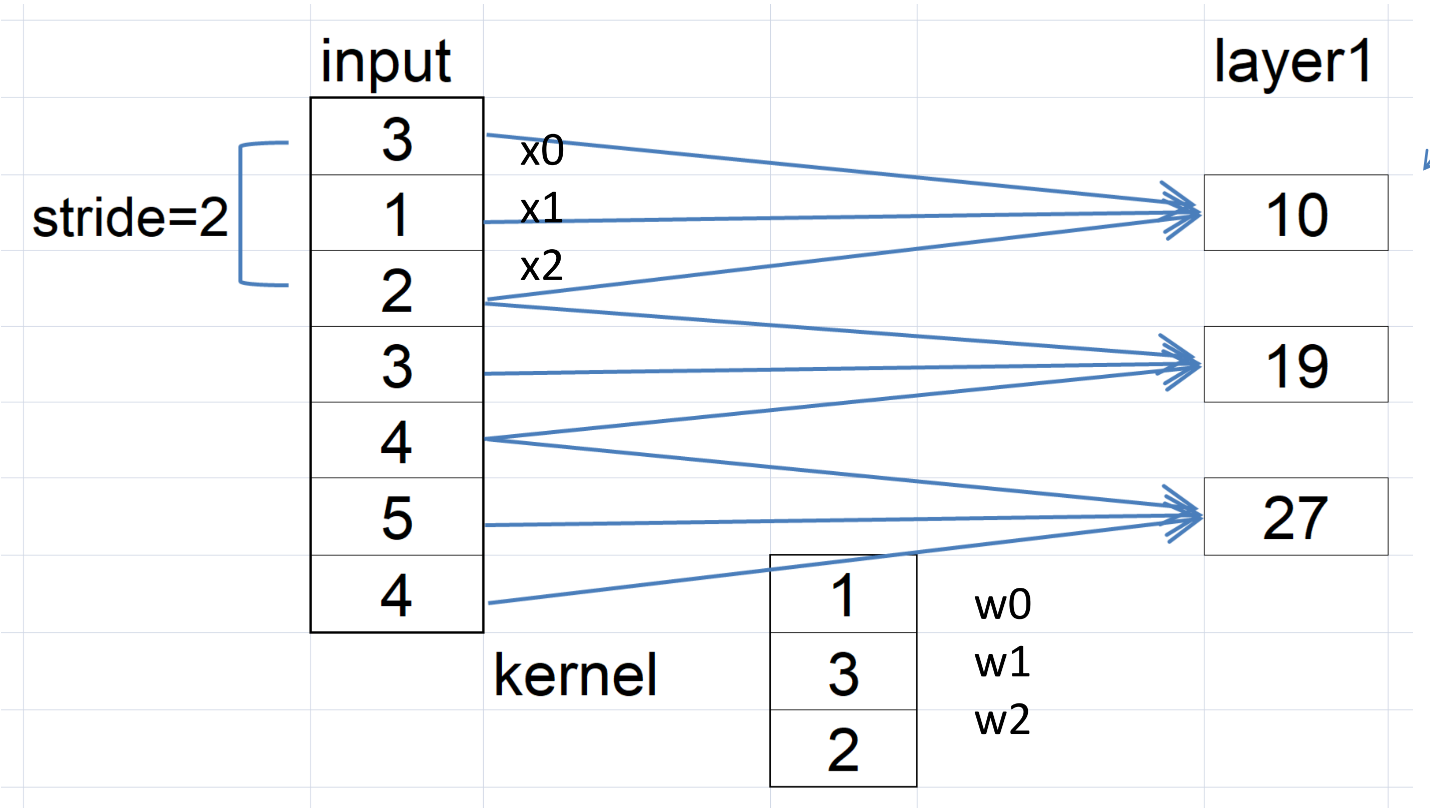
- 1982 – Kunihiko Fukushima, Neocognitron
  - pattern recognition
- 1989 – Yann LeCunn, LeNet-5
- 2010 – ImageNet Competition
  - 1000 classes, over million of images
- 2012 – Alex Krirzevsky, AlexNet
  - 15% error rate for ImageNet (runner-up: 26%)
  - 8 layers
- 2015 – Deep Residual Nets wins ImageNet
  - over 100 layers!

Why now?

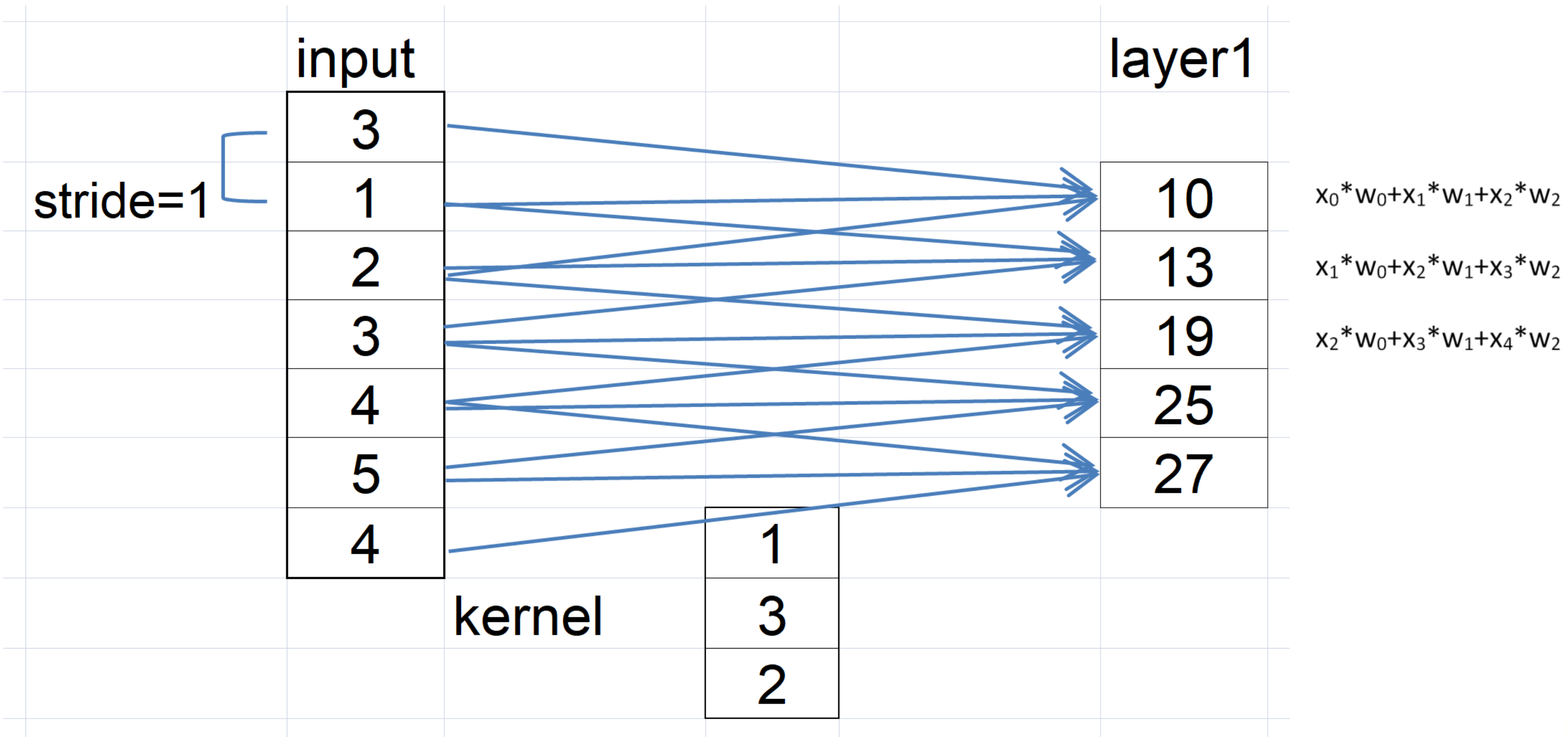
- Neural Network training and predicting involves a lot of calculations
  - Only new computers are capable to calculate it in reasonable time (esp. with CUDA/GPU)
- Neural Network training requires a lot of training data
  - Huge datasets like ImageNet were not available before
  - Everybody may easily create their own dataset using internet resources
- Some advancement in algorithms

CNN - 1D example

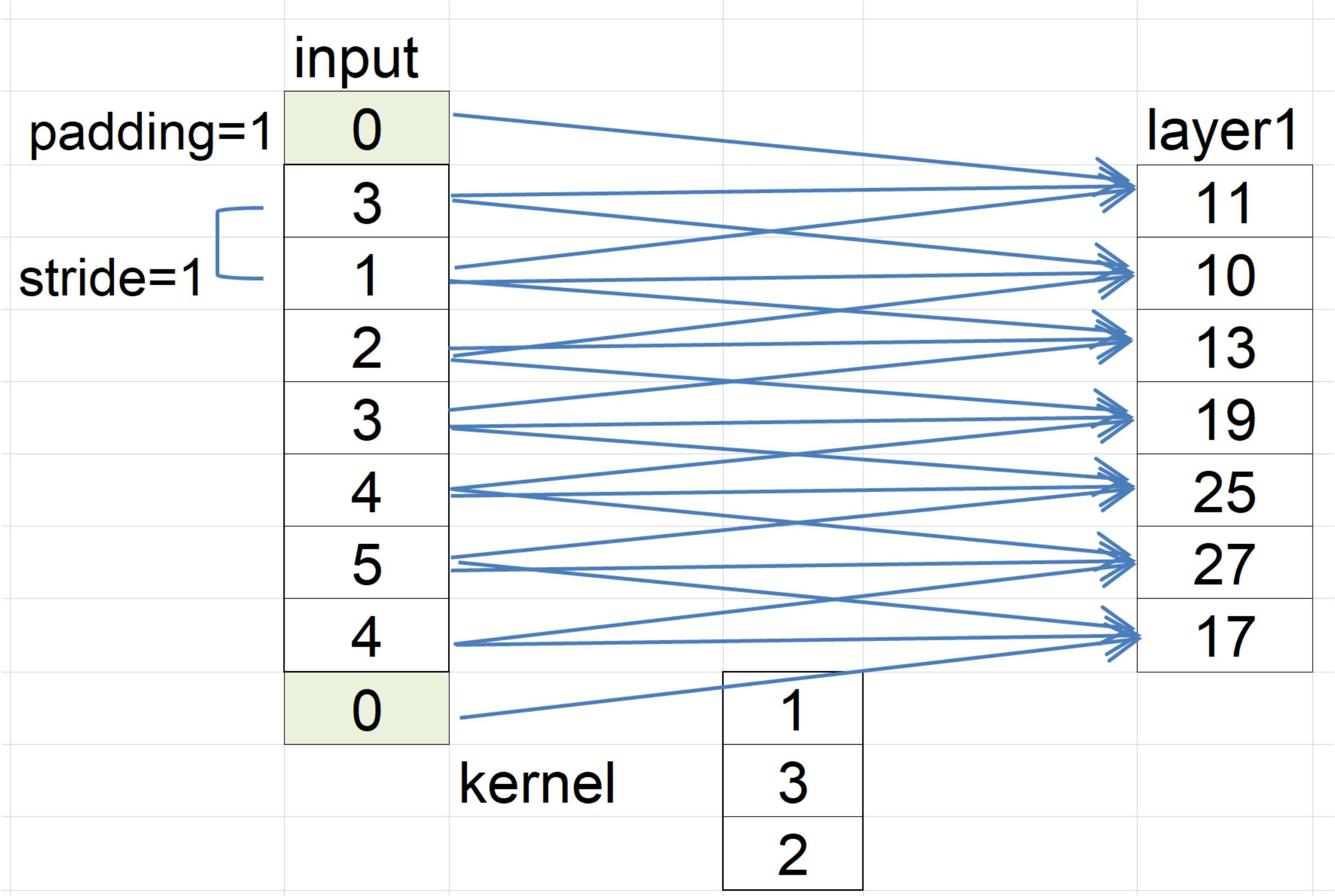
$x_0 * w_0 + x_1 * w_1 + x_2 * w_2$



CNN - stride=1

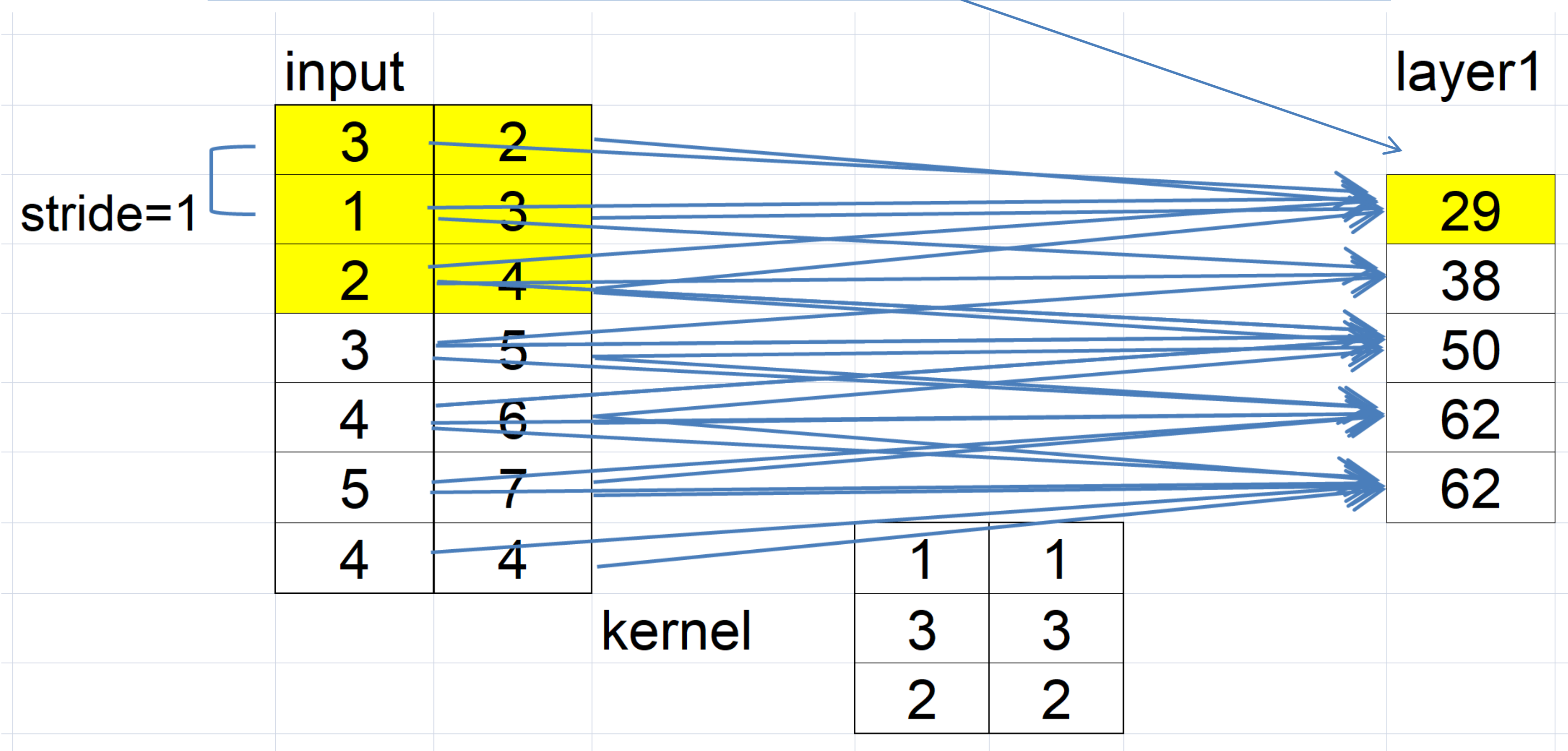

input
layer1
stride=1
3
1
2
3
4
5
4
10
13
19
25
27
kernel
1
3
2
x0*w0+x1*w1+x2*w2
x1*w0+x2*w1+x3*w2
x2*w0+x3*w1+x4*w2

CNN - stride=1, padding=1



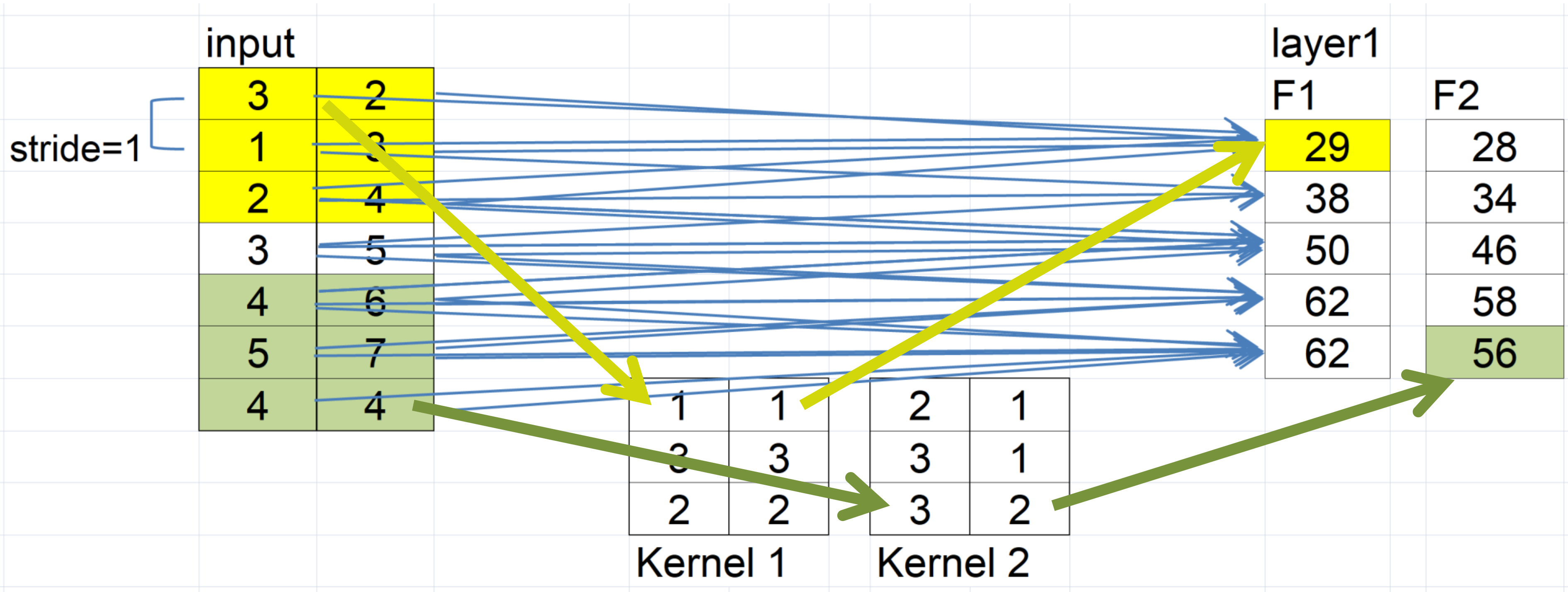
CNN - 2D

$$x_{00}*w_{00}+x_{01}*w_{01}+x_{02}*w_{02}+x_{10}*w_{10}+x_{11}*w_{11}+x_{12}*w_{12}$$



ETRA · 2019

CNN – 2D, many filters



ETRA·2019

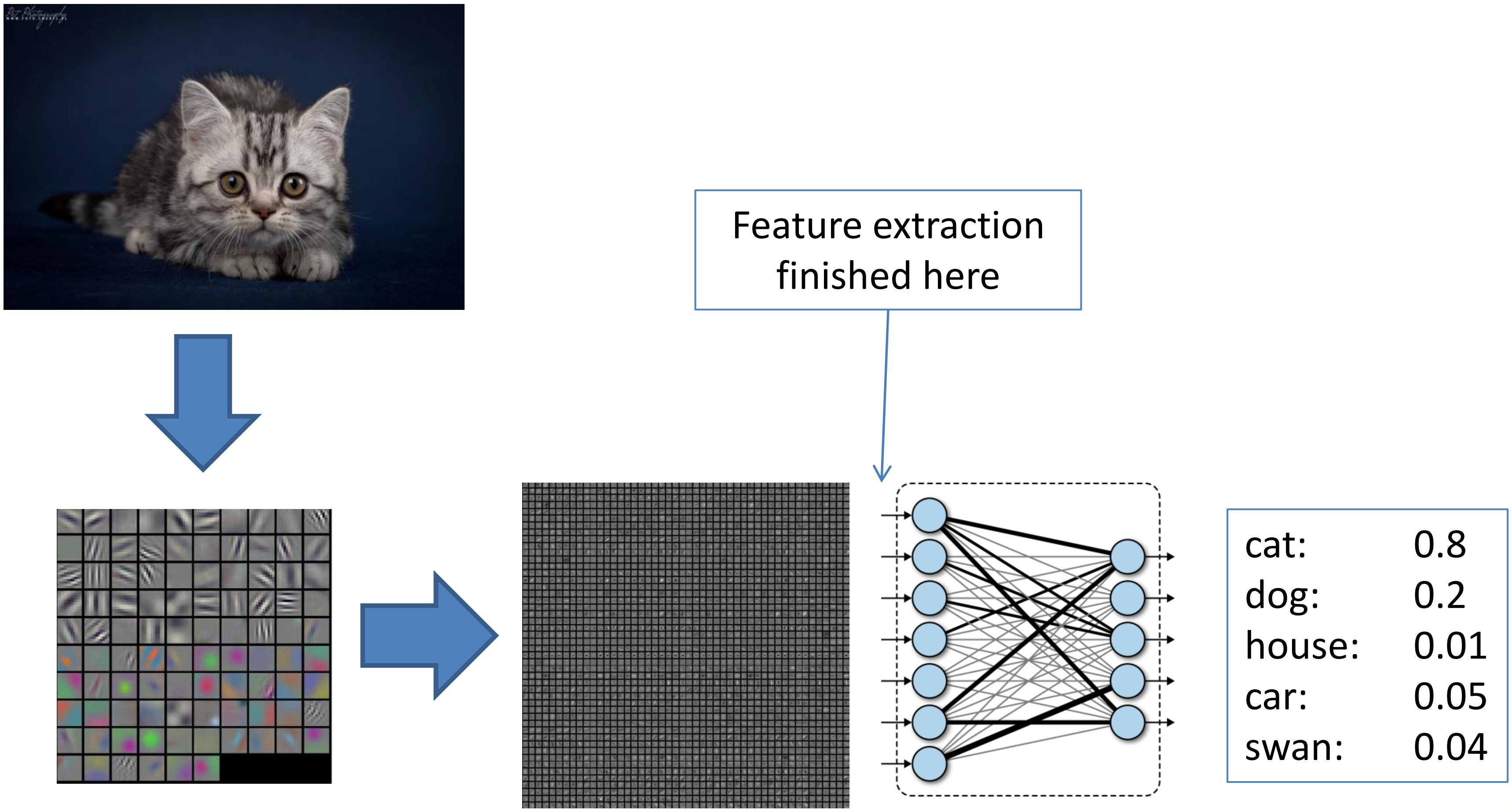
Convolutional Layer

- Parameters:
  - filters – how many kernels
  - kernel size – 1D or 2D
  - stride – step for applying the kernel
  - padding – add borders with zeroes
    - VALID – no padding
    - SAME – padding to preserve size (if STRIDE=1)
- Keras:
  - Conv2D(filters=64, kernel_size=(3,3), padding=SAME, input_shape=(32,32))

Advantages of CNN

- Takes into account spatial/temporal relationships between features
- May find patterns in data
- Builds own filters that extract useful information
- The output from convolutional layer is a set of filters representing various properties of the image
  - i.e. features! – which are automatically created

Example of CNN



 pawel@kasprowski.pl

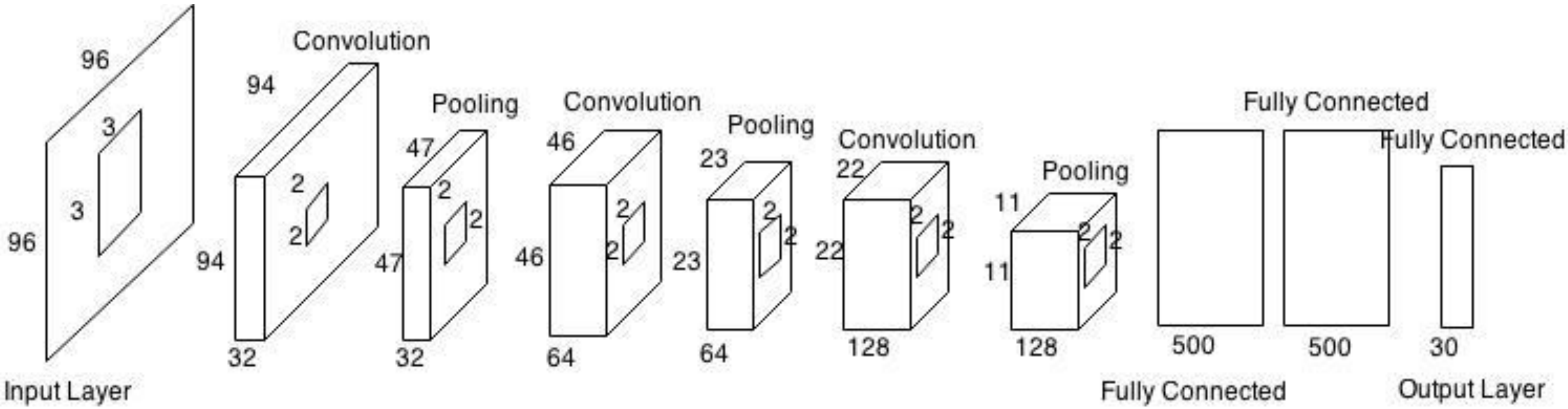
CNN Layers in Keras

- Conv2D(filters=16, kernel_size=(3, 3), padding="same",input_shape=(120,120))
  - classic layer
- MaxPooling2D(pool_size=(2, 2))
  - calculate max for given area
  - reduces size
- Dropout(rate=0.25)
  - randomly set rate percent of weights to zero
  - helps to prevent overfitting
- BatchNormalization(axes=-1)
  - normalizes output from the layer

ETRA • 2019

Cascade of layers

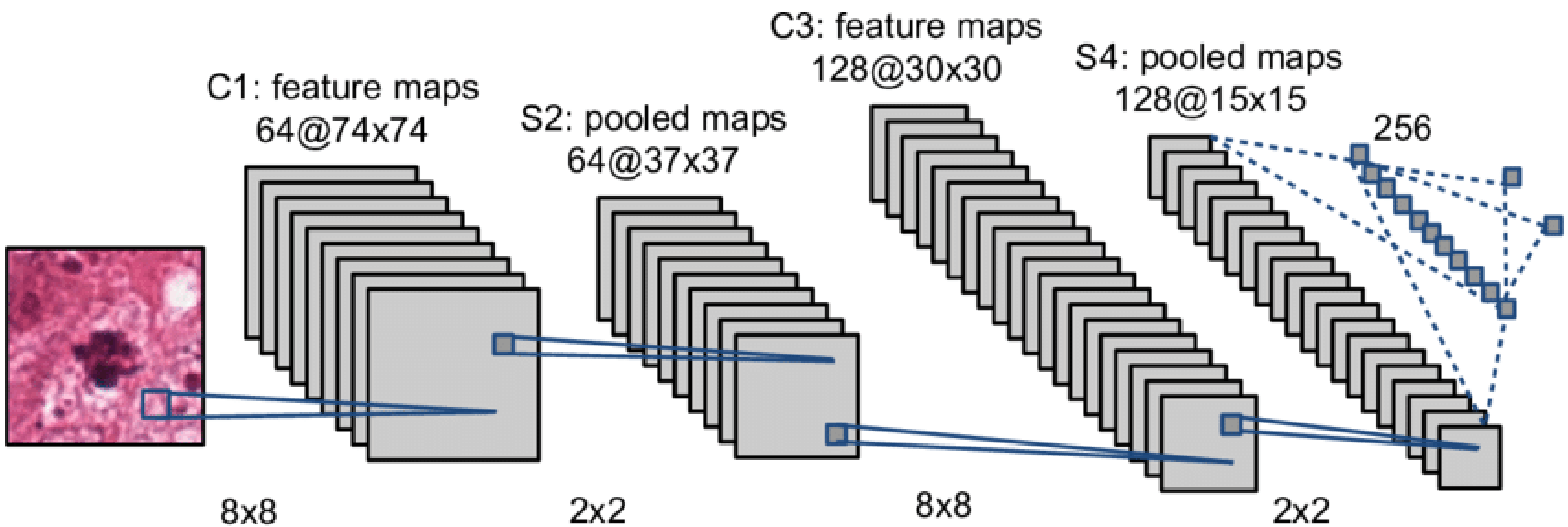
- Conv2D>MaxPooling>Conv2D>MaxPooling>Dropout
- Example:



https://www.hackerearth.com/practice/notes/

 pawel@kasprowski.pl

Another example



https://www.researchgate.net/publication/266734716

Artificial example

- Classification of artificial images
  - black images with white dashes
- Three algorithms:
  - Decision Tree
  - Classic Neural Networks (flat - MLP)
  - Convolutional Neural Network (CNN)
- 200 generated images for each class

Run: *cnn1/images_test.py*

Run: *cnn1/images_test_4classes.py*

Flat models

- Each pixel is treated as a separate feature
  - 10.000 features for 100x100 images
- Two applications:
  - Classic Decision Tree algorithm

```
model = sklearn.tree.DecisionTreeClassifier()
```

  - MLP network

```
model = Sequential()
model.add(Flatten())
model.add(Dense(50, activation='sigmoid'))
model.add(Dense(classes_number, activation='softmax'))
```

Standard CNN Model

- Two parts:
  - Convolutional
    - extracting different image features using filters
  - Dense
    - classic neural network with all layers connected
- Convolutional part:
  - Conv2D > MaxPooling2D > Conv2D > MaxPooling2D > Dropout
- Dense part
  - Flatten > Dense > Dense

CNN architecture

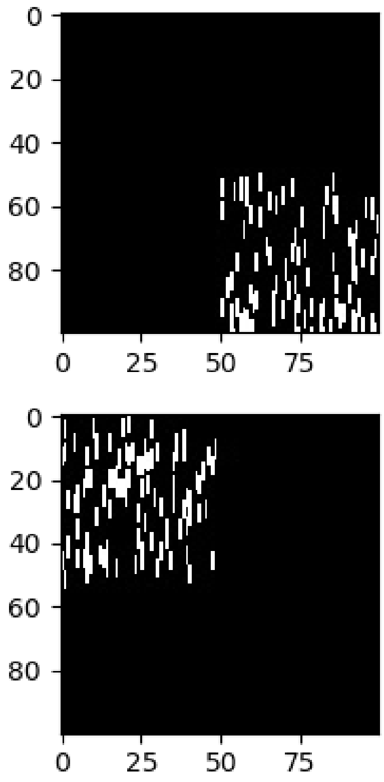
- Convolutional part

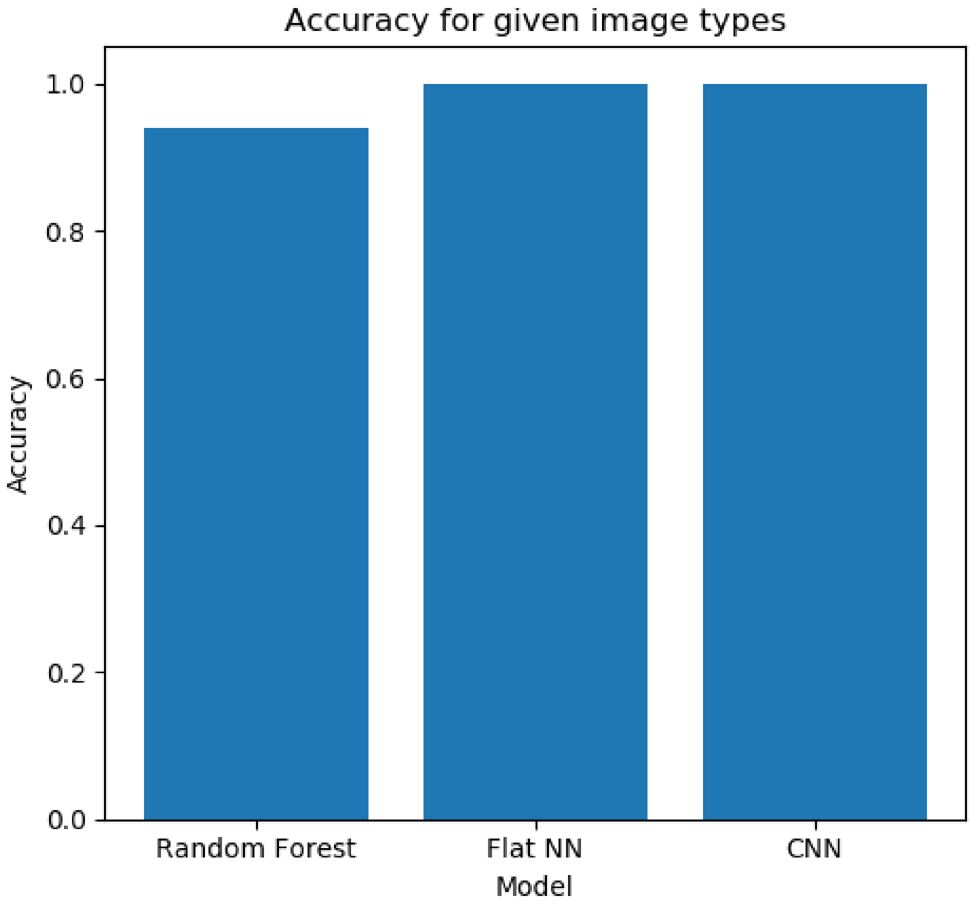
```
model = Sequential()
model.add(Conv2D(16, (3, 3), padding="same", input_shape=inputShape))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

- Dense part

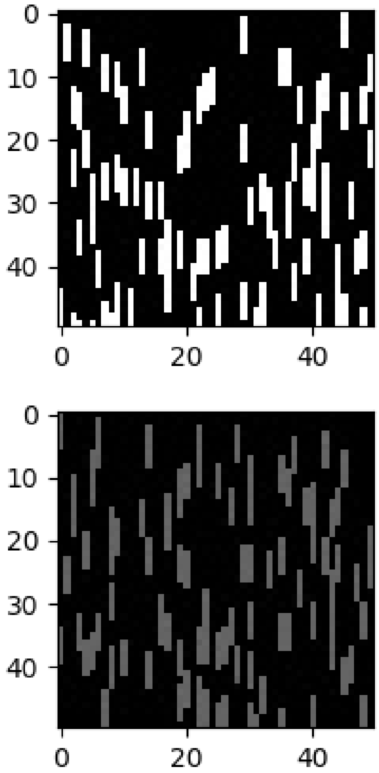
```
model.add(Flatten())
model.add(Dense(512))
model.add(Activation("sigmoid"))
model.add(Dense(512))
model.add(Activation("sigmoid"))
```

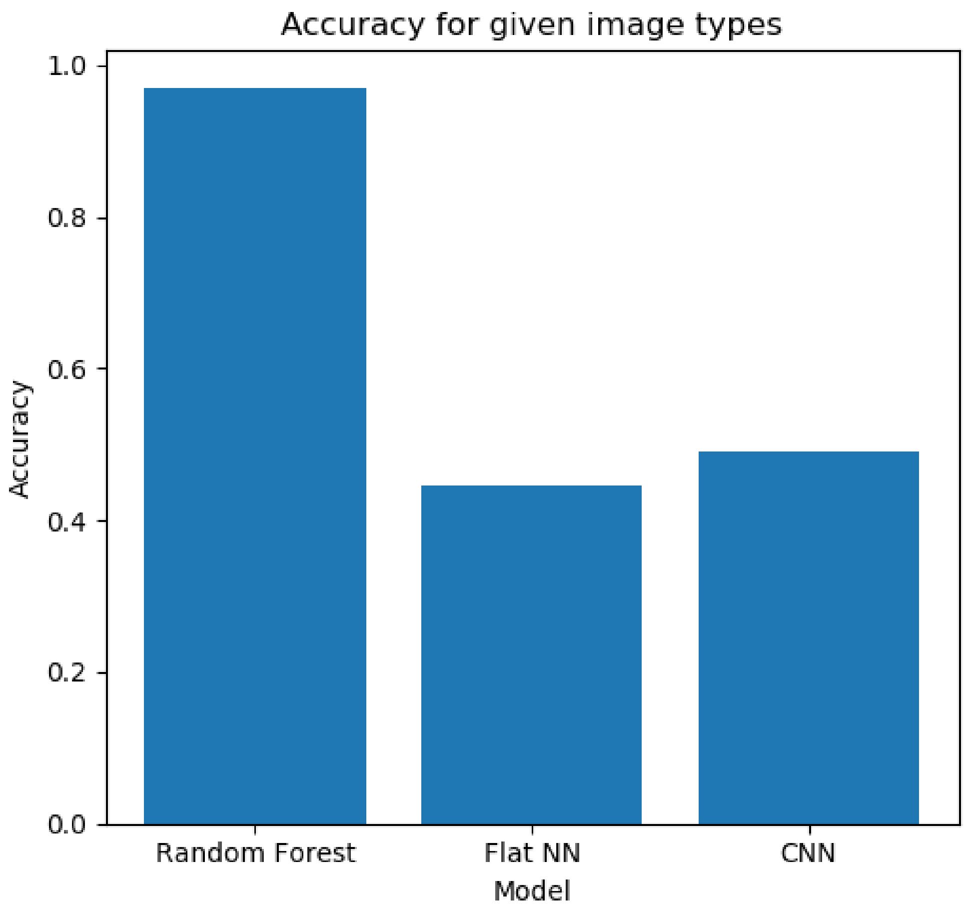
Different areas



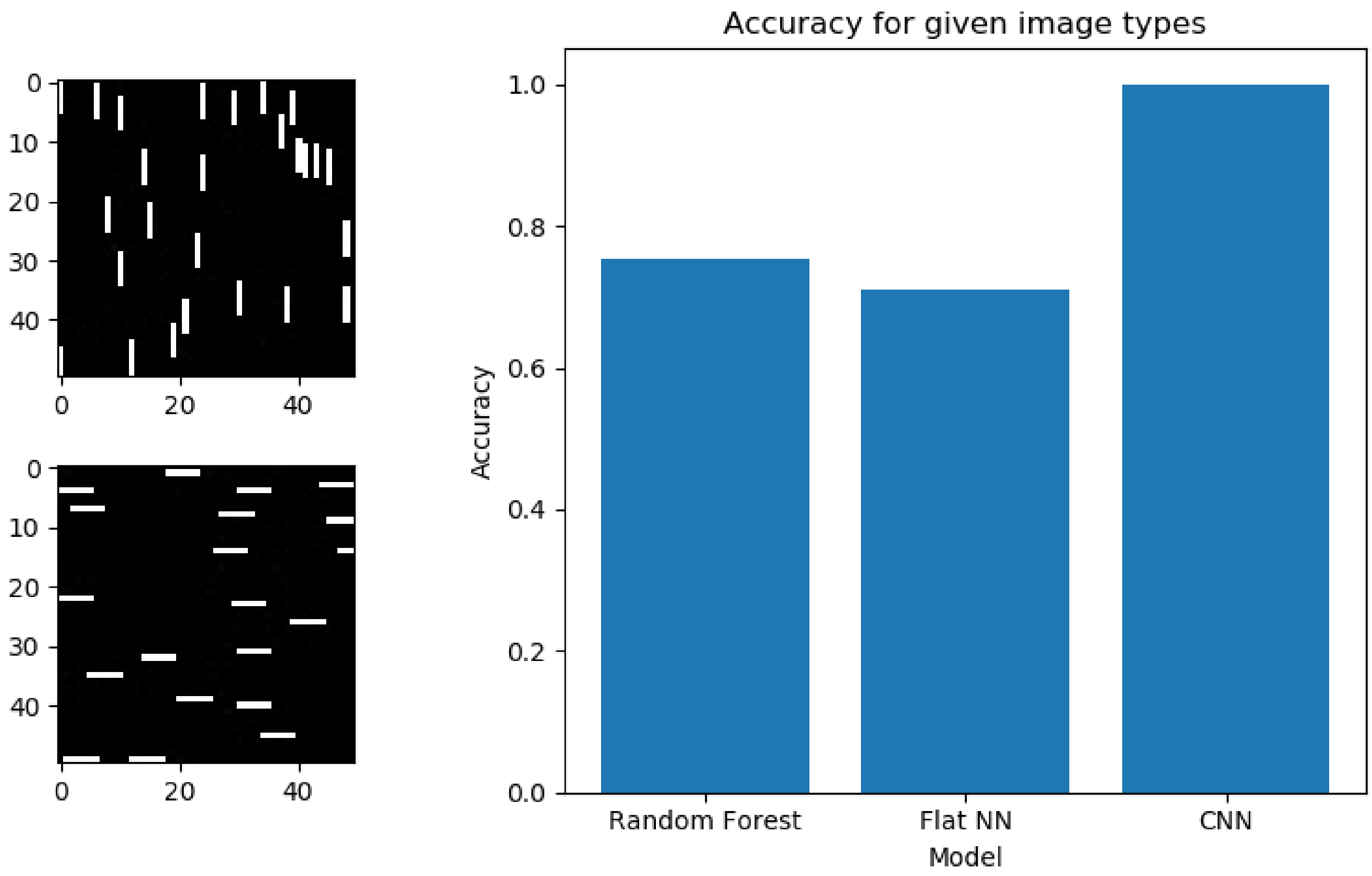


Different intensities/colors

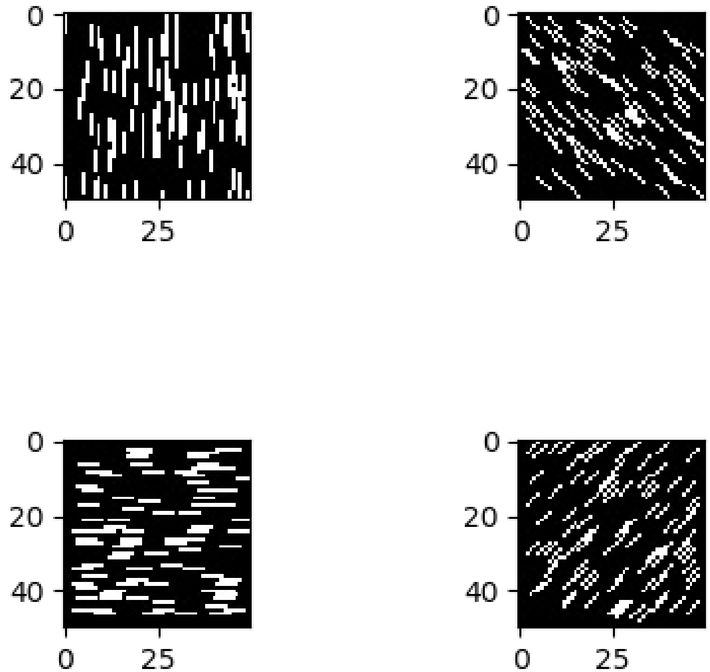


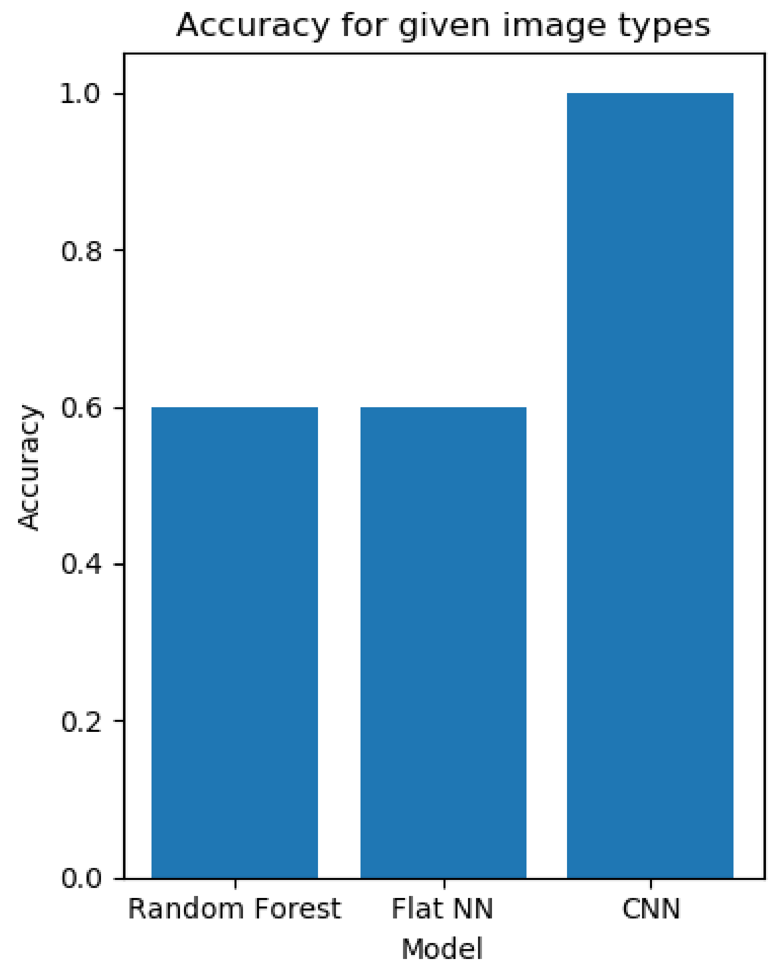


Different direction of dashes



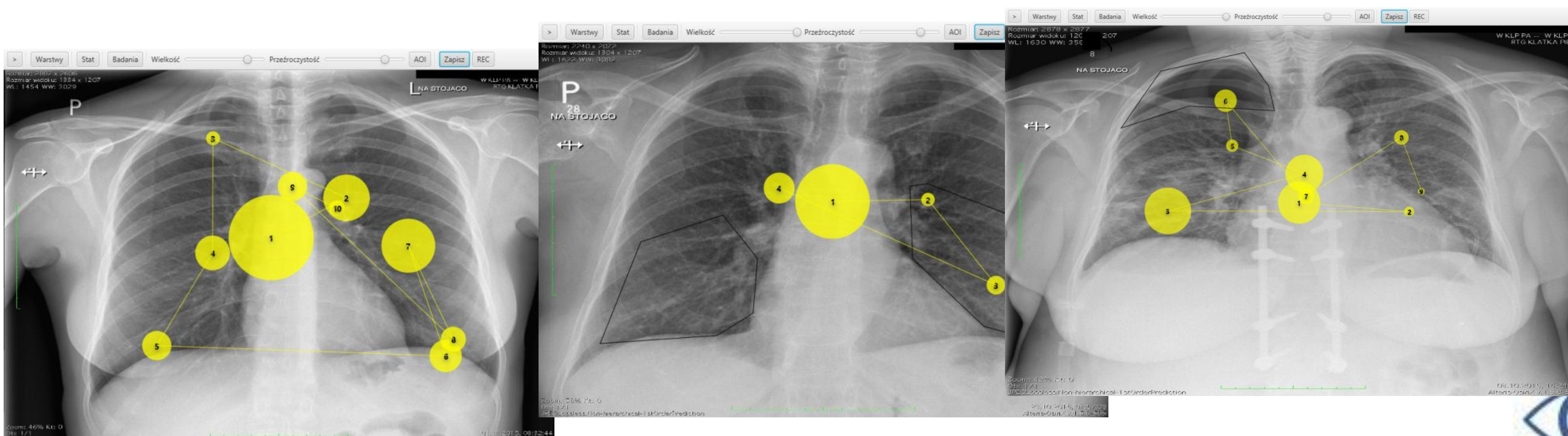
Four different directions





Example for eye tracking

- Specialists, residents and laymen observed the same set of chest radiographs
- Eye movement of a person were registered during each session
- Task: distinguish type of the observer (specialist vs resident vs layman)



ETRA • 2019

Input data [1]

- Text files in tabular format

  timstamp,smoothx,smoothy,rawx,rawy,fixation

  610288046, 913.17, 515.64, 823.76, 475.20, 1

  610288063, 913.31, 516.22, 855.16, 492.03, 1

  610288079, 913.65, 517.81, 862.49, 508.85, 1

  610288096, 913.48, 517.75, 852.26, 477.54, 1

  ...
- 611 samples
- From 200 to 660 readings (rows) per sample

[1] Kasprowski, Harezlak, Kasprowska: Development of diagnostic performance & visual processing in different types of radiological expertise. ETRA 2018

Tree

Run: *radio/classify_tree.py*

- Takes all values (without the timestamp!)
- Pads samples to 700 (filling with zeroes)
- Result:
  - Accuracy approx. 40%
  - Cohen's Kappa approx. 0.07

pawel@kasprowski.pl

Conversion to images

Run: *radio/prepare_img.py*

- Plot all points from a file on image
- Use GaussianBlur
- Result: a saliency map

CNN Model

- #Convolutional Layer 0
  - model.add(Conv2D(16, (3, 3), padding="same",input_shape=inputShape))
  - model.add(Activation("relu"))
  - model.add(MaxPooling2D(pool_size=(2, 2)))
- #Convolutional Layer 1
  - model.add(Conv2D(32, (3, 3), padding="same",input_shape=inputShape))
  - model.add(Activation("relu"))
  - model.add(MaxPooling2D(pool_size=(2, 2)))
- #Convolutional Layer 2
  - model.add(Conv2D(64, (3, 3), padding="same",input_shape=inputShape))
  - model.add(Activation("relu"))
  - model.add(MaxPooling2D(pool_size=(2, 2)))
  - model.add(Dropout(0.25))
- #Dense Layer
  - model.add(Flatten())
  - model.add(Dense(512))
  - model.add(Activation("sigmoid"))
  - model.add(Dense(256))
  - model.add(Activation("sigmoid"))
- #Final classifier
  - model.add(Dense(numClasses))
  - model.add(Activation("softmax"))

CNN

Run: *radio/classify_cnn.py*

- CNN network using images
  - Accuracy approx. 70%
  - Cohen's Kappa approx. 0.5

Callbacks

- It is possible to add a callback to the fit() function
  - executed for every epoch
- Examples:
  - History – remembers metrics for every step
  - ModelCheckpoint – saves model after step
  - EarlyStopping – stops training when no progress
  - RemoteMonitor – sends results in POST requests
  - LambdaCallback – call to your own function

ModelCheckpoint callback

- The callback that saves the models to files:

```
checkpt = ModelCheckpoint (
    filepath='model.{epoch:02d}-{val_loss:.2f}.h5',
    save_best_only=True)
```

- Using the callback:

```
model.fit(..., callbacks = [checkpt],...)
```

EarlyStopping callback

- The callback that stops training when for 3 consecutive epochs the val_loss doesn't decrease

```
chkpt = EarlyStopping(monitor='val_loss', patience=3)
```

Combined model

- Problem
  - We have both images and descriptions as a set of features
- Possible solutions
  - Classify independently images and feature sets, use voting to choose a final label

  or

  - Build a network that combines two types of inputs

Example

Run: *radio/combined_model.py*

- Loads images and a CSV file with 14 attributes
  - samplesIMG
  - samplesCSV
- Builds two networks
  - CNN for images - *cnnmodel*
  - MLP for attributes - *flatmodel*
- Concatenates both networks

Concatenation

- Build the model using cnnmodel and flatmodel

```
combined = concatenate([cnnmodel.output, flatmodel.output])
combined = Dense(16, activation="sigmoid")(combined)
combined = Dense(numClasses, activation="sigmoid")(combined)
model = Model(inputs=[cnnmodel.input, flatmodel.input],
    outputs=combined)
model.compile(loss=loss, optimizer="adam",metrics=['accuracy'])
```

- Fit using both sets

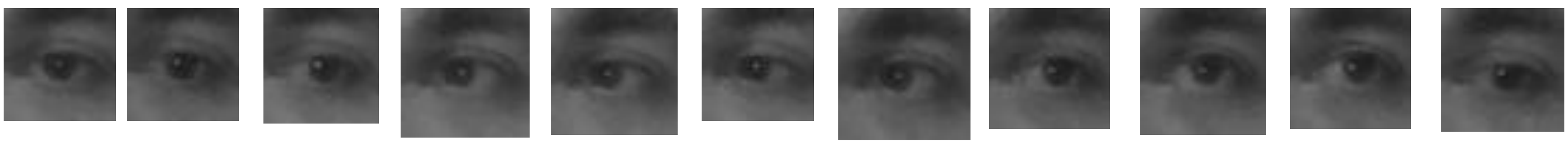
```
model.fit([samplesIMG,samplesCSV], labels, batch_size=BATCH,
    epochs=EPOCHS,validation_split=0.25)
```

- Predict using both sets

```
results = model.predict([samplesIMG,samplesCSV])
```

Regression example

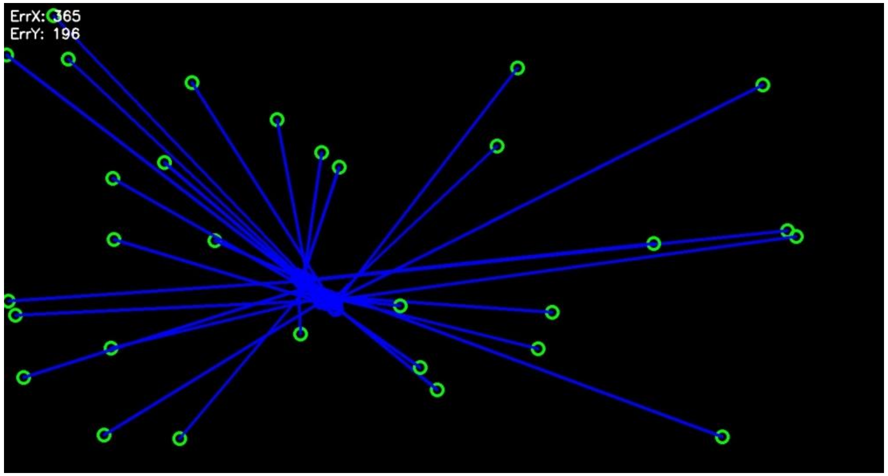
- Input: low resolution eye images
- Output: gaze location (x,y)

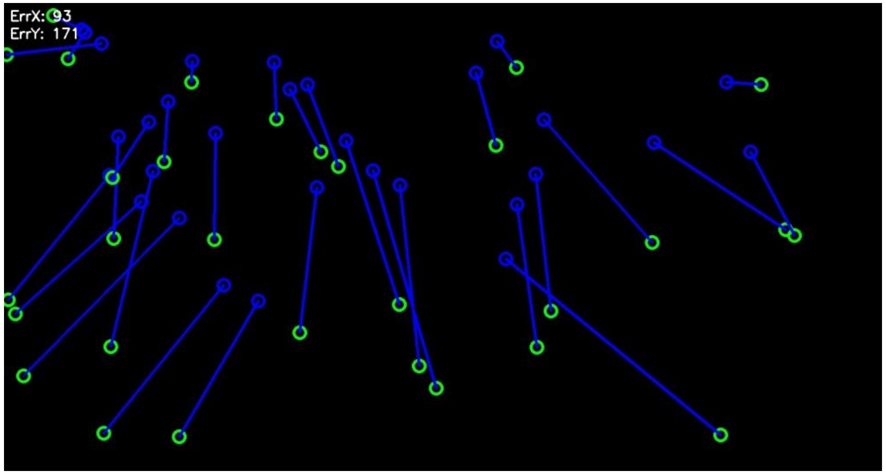


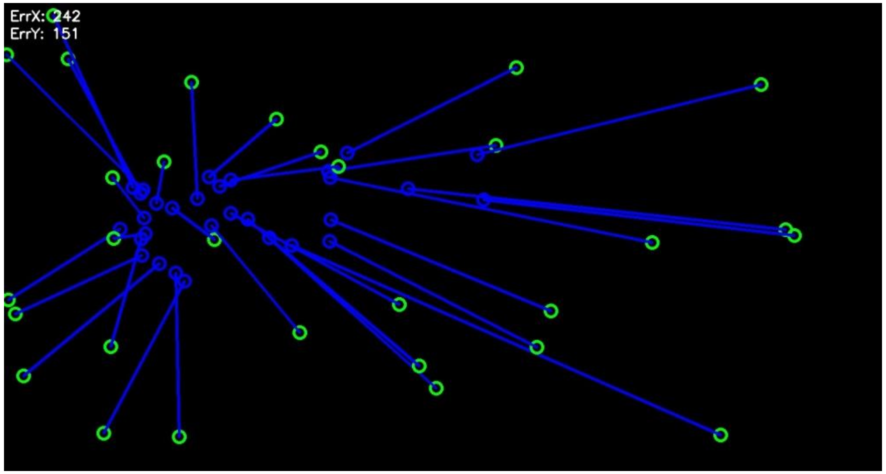
- Solution:
  - CNN model returning two values: (x,y) gaze coordinates

Run: *eyes/eye_to_gaze.py*

Training the model







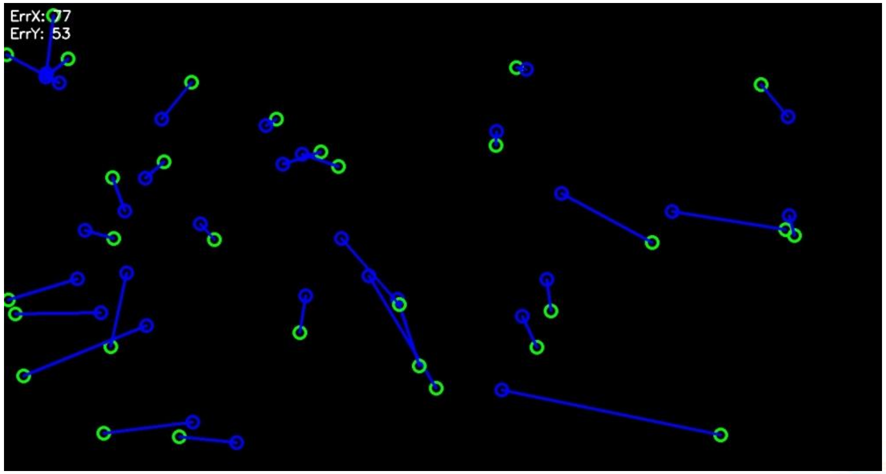
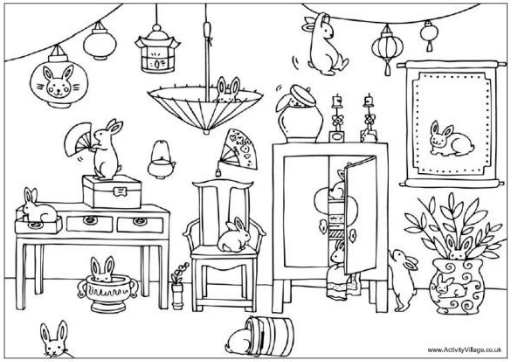


Image prediction example [1]

- Four images, 18 observations of each image
- Task: predict image observed from eye movements

Na kolejnym obrazku policz ile
widzisz króliczków.
Zapamiętaj ten wynik, bo będzie
potrzebny!
Na koniec zostaniesz zapytany
o ich liczbę.



[1] Kasprowski, Harezlak: Gaze Self-Similarity Plot-A New Visualization Technique. *Journal of Eye Movement Research*. 2017

Using Conv1D

- Ready-to-use Keras layer
  - gets 2D samples instead of 3D images
- Our samples:
  - chunks od 100 recordings, two values (X, Y) for each
- Our labels:
  - one-hot encoded four images: bus, cat, txt, rab
- Our model:
  - stack of Conv1D layers

Example code

Run: *images/images_cnn.py*

- Code using DecisionTree (with flattened samples):
  - Accuracy: 0.41, Kappa: 0.10
- CNN code:
  - almost perfect after 10 iterations!
- Note aside:
  - the experiment is not quite fair: sequences from the same file may be used in training and test set!
- But: It shows that when the same pattern repeats in data (even in different locations) CNN deals with it perfectly (and DecisionTrees does not...)

Summary

- CNNs are really powerful but it is sometimes difficult to build your own architecture that works
- So, using a ready and working architecture may be useful
- There are ready to use networks implemented in Keras
- These networks are also pretrained!

Ready to use pretrained models

- Xception
- VGG16
- VGG19
- ResNet, ResNetV2, ResNeXt
- InceptionV3
- InceptionResNetV2
- MobileNet
- MobileNetV2
- DenseNet
- NASNet

Usage of pretrained networks

- Loading is simple:
  - model = ResNet50(weights='imagenet')
- The weights may be updated for our problem
  - typically we need to update only a few top layers (trainable=False for all other)
    - for CNNs it is commonly only the "flat" part of the network
- Most architectures are quite deep
- Most are optimized to categorize images

Using a pretrained model

Run: *eyes/test_model.py*

- The script
  - loads and prepares eye images
  - loads the pretrained model "model_XXX.h5"
  - uses the model to find gaze coordinates (X,Y)
- The model was trained using more images and gives better results!

Sequence

- What if my samples come in a sequence and the result (label) should depend not only on the corresponding sample but on the samples previous in the sequence?
- The answer is:

  Recurrent Neural Network (RNN)

Recurrent Neural Networks

The network that remembers...

Problem statement

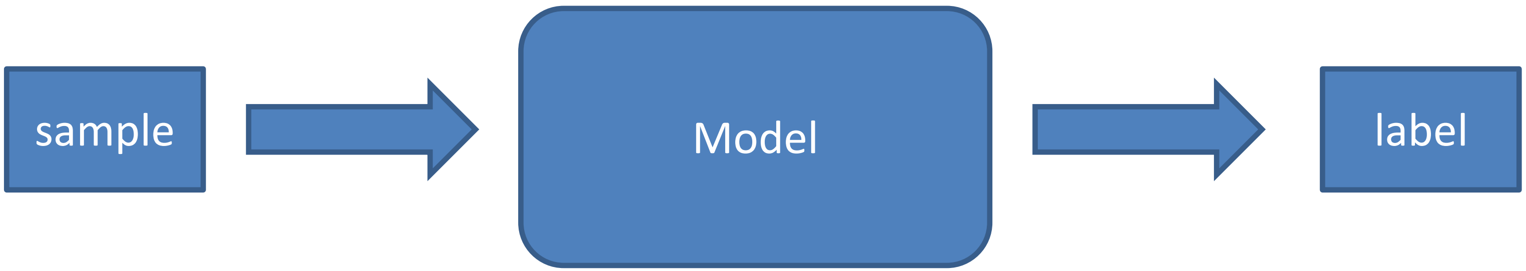
- Till now we created models (functions) that map (convert) a sample to a label
  - sample -> label
- Set of samples was converted to a set of labels
  - each sample was treated independently and the model produced one label for each sample
- There are applications were there is a sequence of samples which should be converted to a sequence of labels
  - label value depends not only on one sample but also on previous samples (and previous labels!)

Conventional NN

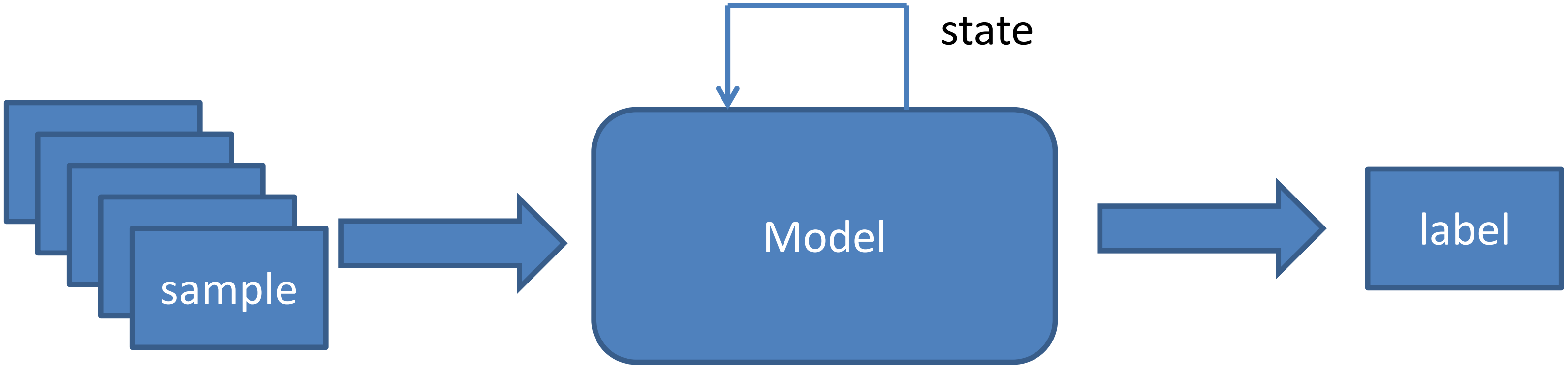
- Conventional NN
  - sample -> MODEL -> label
- Recurrent NN
  - sequence of samples -> MODEL -> label (or sequence of labels)
- MODEL has state (memory)!
  - current output depends on previous samples and outputs
- Instead of $Y_t = f(X_t)$, now we have:
  - $Y_t = f(X_t, X_{t-1}, X_{t-2}, \ldots, Y_{t-1}, Y_{t-2}, \ldots.)$

Recurrent NN

- Classic NN

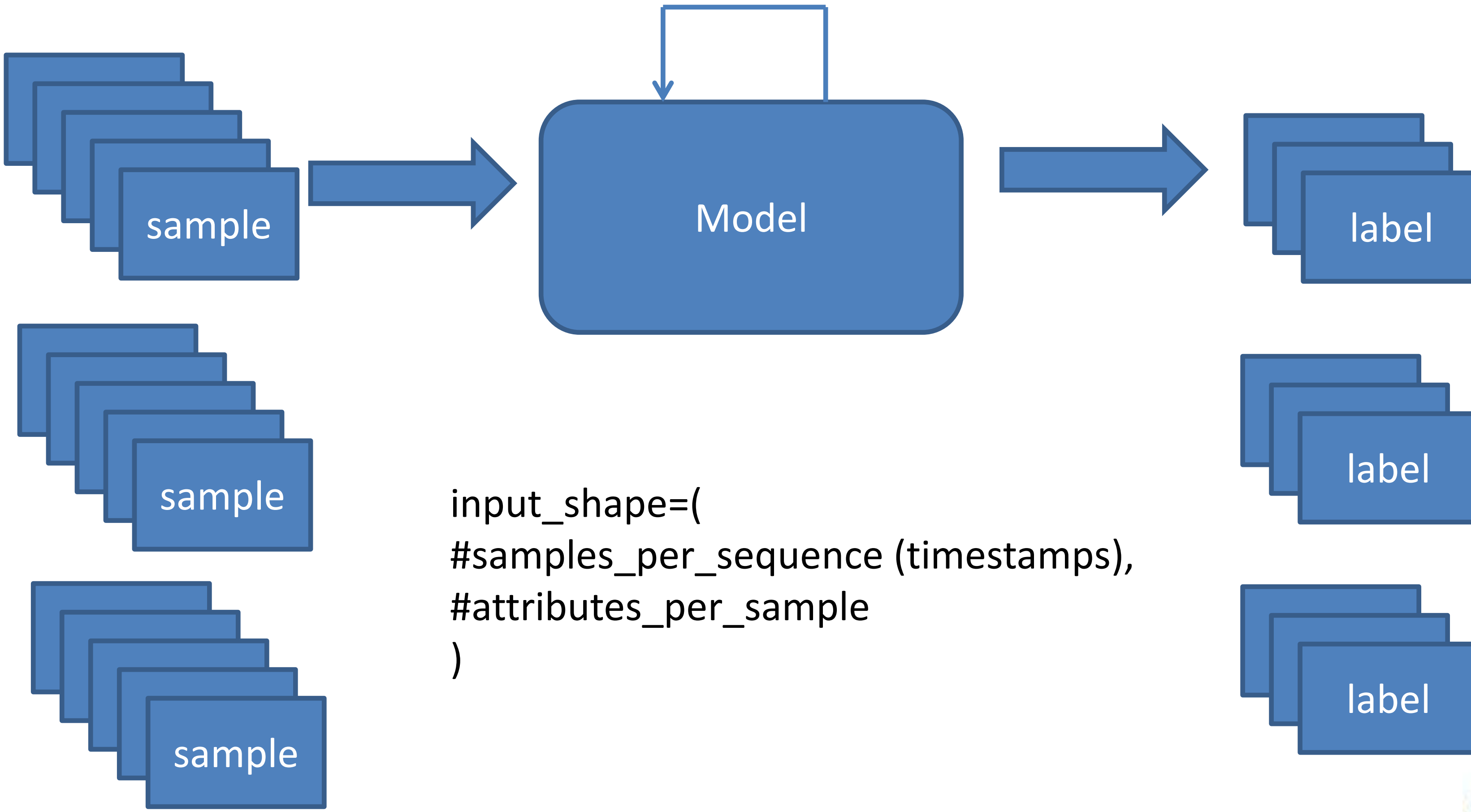


- Recurrent NN



Full RNN

state maintained across sequences



input_shape=(
#samples_per_sequence (timestamps),
#attributes_per_sample
)

Applications

- Image -> text description
  - seq_in=1, seq_out=10
- English sentence (words) -> Polish sentence
  - seq_in=20, seq_out=20
- Text (sequence of words) -> next word
  - seq_in=10, seq_out=1

RNN layers in Keras

- Layers with memory!
- RNN
- SimpleRNN
- GRU (Gated Recurrent Unit)
  - additional gate
- LSTM (Long Short-Term Memory Layer)
  - even more gates (input, output, forget)
- ConvLSTM2D

LSTM layer main parameters

- units – number of units (output size)
- input_shape=(timestamps, attributes)
  - timestamps = the number of samples in sequence
  - attribute = the number of attributes per sample
- return_sequences
  - False – return only the last state Y_t
  - True – return all states $Y_1 \ldots Y_t$ - output becomes a matrix of size: (units,timestamps)
- stateful
  - False – reset state after each batch
  - True – maintain state until model.reset_states() called

Example

- Network learning roman->arabic numbers translation
- Input: roman number
- Output: arabic number
- Examples:
  - *IN: MDCCXXXI, OUT: 1731*
  - *IN: DCXLIX, OUT: 649*
  - etc.

Run: *rnn/roman_rnn.py*

Code analysis

- Input and output
  - INPUT_TS = 14 chars from romans='MDCLXVI '
  - OUTPUT_TS = 4 chars from digits='0123456789 '
- One-hot encoding
  - 'M' becomes (F, F, F, F, F, F, T, F) – 1 of 8 possible characters
- Model
  - model = Sequential()
  - model.add(LSTM(128, input_shape=(INPUT_TS, len(romans))))
  - model.add(RepeatVector(OUTPUT_TS))
  - model.add(LSTM(128, return_sequences=True))
  - model.add(Dense(len(digits), activation='softmax'))
- Prediction
  - preds = model.predict_classes(roman_sequence, verbose=0)

Results

- 750 examples for training
- 250 (different than training!) examples for testing
- After 200 iterations – 95% of correct translations!
  - The network does not know any rules of translation – just learns from examples...

Analysis of the network's structure

- input: 'MXVII   ' encoded to (14,8)
  - 14 samples in a sequence
  - every sample one of 8 possibilities ('MDCLXVI<space>')
- LSTM(128, input_shape=(14, 8))
  - output: (128)
- RepeatVector(4) [output sequence will have length=4]
  - output: (4,128)
- LSTM(128, return_sequences=True)
  - output (4,128) – because sequences is True
- Dense(11, activation='softmax')
  - output (4, 11) – 4 chars each one from 11 possibilities ('0123456789<space>')

RNN for eye movement data

- Example: Creating artificial scanpaths
- Input data [1]:
  - 57 genuine scanpaths recorded during image observation

[1] Skurowski, Kasprowski: Evaluation of Saliency Maps in a Hard Case – Images of Camouflaged Animals
Best Paper of 2018 IEEE International Conference on Image Processing, Applications and Systems (IPAS)

Preprocessing

- The model tries to predict gaze coordinates (x,y) based on previous 20 gaze coordinates
- Each scanpath preprocessed to form:
  - sequences – 20 subsequent recordings (x,y)
  - labels – single recording directly after the 20
- 57 scanpaths -> 5757 sequences of length 20 each
- RNN architecture:

```
model = Sequential()
model.add(LSTM(128,input_shape=(20,2)
                                        ,return_sequences=True))
model.add(LSTM(128))
model.add(Dense(64))
model.add(Dense(2))
```

Training

Run: *animals/animal_rnn.py*

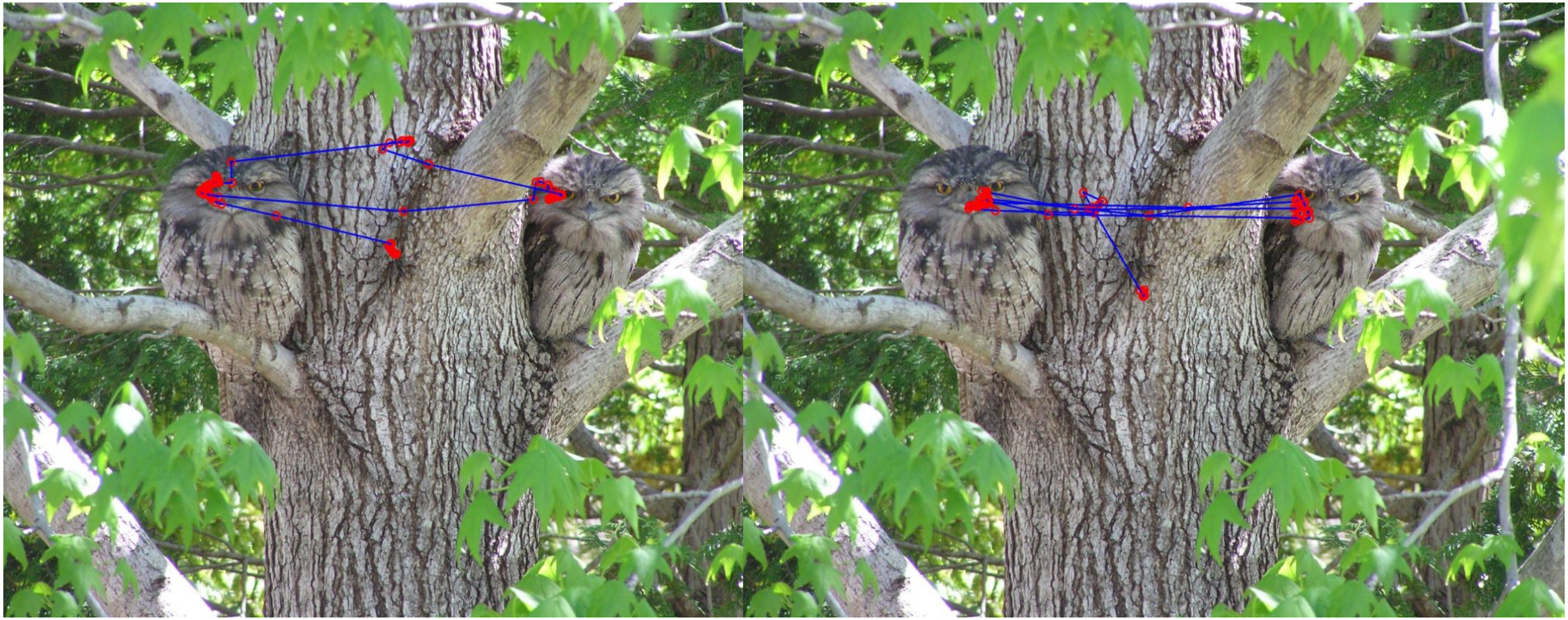
- Loss function: mean absolute error
  - model.compile(loss="mean_absolute_error", optimizer="adam", metrics=["mae"])
- Fitting in subsequent runs (loop)
- After each iteration the image with all predicted gazes is saved in /img folder:
  - animal-11_eNN.jpg
  - where NN is the epoch number
- The final model is saved:
  - model.save(*"model_rnn.h5"*)

Using the model

Run: *animals/test_model.py*

- Loads the model:
  - model = load_model(*"model_rnn.h5")*
- Uses the model to create artificial scanpaths
  - predictions = model.predict(sampleSequence)
- Creates images with scanpaths in /scanpaths folder
  - save_scanpath(sequence, name)

Real vs. generated



It is not possible to tell the difference!

Summary

- Recurrent Neural Networks
  - remember previous state
  - input is present and past (also previous decisions)
- The most popular implementation:
  - Long Short-Term Memory Units (LSTM)
- RNNs proved to be extremaly good at translating and even producing texts
- RNN works much better when the number of possible inputs and outputs is limited
  - there are better methods for time series prediction (for now!)

Conclusion

What you should remember...

Quick recap

- Classification
  - many algorithms and measures, the interpretation of results may be tricky
- Neural Networks
  - easy implementation in Keras, training requires a lot of data
- Convolutional Neural Networks
  - automatically building conv filters using examples, powerful for image classification
- Recurrent Neural Networks
  - ideal for sequences, have memory (state), powerful for text analysis

Final remarks

- It is relatively easy to implement deep learning for your own experiment
  - There are simple plug-in libraries
- Two main problems:
  - we need a lot of data to train the network
  - architecture of the network is very important
- The field is very dynamic – new algorithms are created constantly
- Unfortunately frequently we know only that it works but we don't know what happens inside...

THANK YOU FOR YOUR ATTENTION