Abstract

BeerIt is an application for beer lovers created during a Team Programming Project proseminar. It provides necessary information and allows its users to create a personalized collection of consumed beers. Its distinct feature is beer recognition based on label images. It was realized with machine learning algorithms, specifically convolutional neural networks. This work presents the created system, its architecture and building process. It is strongly focused on the image recognition problem including its theoretical background.

Keywords

machine learning, convolutional neural network, image recognition, mobile application, Android, Django, iOS, Theano

Erasmus Subject Area Codes

11.3 Informatics

ACM Classification Codes

I. Computing Methodologies
I.5 Pattern recognition
I.5.1 Models

English title

Machine learning algorithms for image recognition in application for beer lovers

Contents

Chapter 1

Introduction

1.1. Motivation

Beer market is one of the fastest growing sectors of the Polish economy[1]. In the last 5-6 years about 40 local breweries have been built in Poland (only in 2013, 20 new breweries were opened). In 2015 some breweries recorded sales increase of up to 50 percent[2]. Furthermore, we can observe rapid growth of new, trendy pubs in Warsaw (and Poland) with large selection of local beers. All of that makes beer lovers overwhelmed with kinds and types of rare beers. On the other hand, there is lack of information about newly created beers. Often there are no indicators, such as IBU or EBC[3] written on beer labels, which results in wrong choices and, in consequence, wasted money. In this state of things we create BeerIt[4] - a mobile application that allows users to recognize beer by label and display detailed information about it.

1.2. Project vision

Our aim was to create a mobile application for beer lovers. Together with our Client — strategy and creative agency Kalicińscy[5] — we planned a perfect application for fans of the amber brew. The core functionality lies in providing detailed information about beers. With this application users can quickly and easily find necessary things about beer. They can do this in one of two ways: take a photo of beer label and immediately get the profile of the recognized beer, or find beer manually by a search bar. Additionally, users are allowed to rate beers (on the scale of 1 to 5), share comments about them and add them to *cellar* or *wishlist* - places where lists of drunk or desirable beers are stored. Alternatively one can use feed - often updated list of the newest and rare beers. Obviously mobile application would be useless without an up-to-date base with beer information. For that purpose, a web application for owners of breweries has been launched. Every single brewmaster, after verification, has access to beer profiles of his own brewery, where beer descriptions can be updated. The required information about beer include: type, brewery, country, EBC, IBU, alcohol by volume, etc.

[1] `https://pl.wikipedia.org/wiki/Browary_w_Polsce`, (Retrieved: Thursday 26th May, 2016)
[2] `http://inwestycje.pl/firma/jonzee;280409;0.html`, (Retrieved: Thursday 26th May, 2016)
[3] `https://en.wikipedia.org/wiki/Beer_measurement`, (Retrieved: Thursday 26th May, 2016)
[4] `https://beerit.co`, (Retrieved: Saturday 28th May, 2016)
[5] `http://www.kalicinscy.com/`, (Retrieved: Sunday 12th June, 2016)

1.3. Image recognition

1.3.1. Overview

From the functionalities described above, the most challenging one is recognizing a beer present in a photo taken by an application user. The problem of recognizing an image content has received great attention throughout many years of computer science. There are many possible ways of approaching this task and most of them are studied by the field called computer vision. Yet, there is another growing area of computer science, machine learning, which provides different (and equally effective) techniques for object classification.

One of the projects of our Client's company also involves object recognition task. Their goal is to distinguish some specific group of products visible in pictures taken by customers. They have built and implemented a system based on some computer vision algorithms. Without going into details, their idea was to extract the product's name (present in textual format somewhere in the picture) and then find the closest match in the training set.

1.3.2. Existing solutions

There are many existing applications of visual recognition presenting different approaches to this task. The first example is a mobile application Vivino[6]. It offers a feature of recognizing a wine given a photo of its label. It resembles the functionality covered by our project, but is realized very differently — it uses a combination of pattern recognition and OCR[7] to match unlabeled queried photos to the already collected and labeled ones.

On the other hand, there is also a variety of machine learning based solutions. First of all, the Google Translate app has recently incorporated the neural networks into its visual translation feature[8]. It is used to recognize letters and digits on small image patches, which is a typical problem in image processing field.

There is another successful application of neural networks in object recognition task presented by Clarifai[9]. Their image content classifier has won the ImageNet competition in 2013[10]. Not only is it a theoretically beneficial solution, but it also serves as an engine in many real-life applications, listed in the Clarifai blog[11]. One of them is Forevery[12], a mobile app capable of recognizing and indexing images depicting people, places, and even emotions.

1.3.3. Our approach

The Client's requirements for our system involved the labels recognition process, but they did not propose or impose any specific solution. Finally, it has been decided to pursue the machine learning approach due to its arising popularity and prospective properties. Using the idea of convolutional neural networks, a standalone classifier capable of beer label recognition has been built and trained.

Contrary to the computer vision based solutions, it does not need to refer to the set of model images during classification of the new ones. Instead, training the model on a large

[6] `https://www.vivino.com/`, (Retrieved: Monday 13th June, 2016)

[7] `https://en.wikipedia.org/wiki/Optical_character_recognition`, (Retrieved: Monday 13th June, 2016)

[8] `https://research.googleblog.com/2015/07/how-google-translate-squeezes-deep.html`, (Retrieved: Monday 13th June, 2016)

[9] `https://www.clarifai.com`, (Retrieved: Monday 13th June, 2016)

[10] `http://www.image-net.org/challenges/LSVRC/2013/`, (Retrieved: Tuesday 5th April, 2016)

[11] `http://blog.clarifai.com/built-with-clarifai`, (Retrieved: Monday 13th June, 2016)

[12] `http://blog.clarifai.com/meet-forevery`, (Retrieved: Monday 13th June, 2016)

dataset of images encodes the domain specific knowledge (which is captured in the dataset and may be formally thought of as a function mapping beer images to beer names) in the prediction model. This intuition is developed and formalized in chapter 2.

Chapter 2

Theoretical background

2.1. Machine learning

According to Kevin P. Murphy in [Mur12], *"we define machine learning as a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data (...)"*. In our project we concentrate only on a narrow subset of machine learning, specifically neural networks. They belong to *supervised learning* type of algorithms, which means the following: given a training set of some objects with associated labels (decisions) we want to be able to predict a decision for future unlabeled examples. In case of image recognition task, each image can be represented as a vector $x \in \textsc{Imag} \subset \mathbb{R}^n$ (where n is the dimension of an image, i.e. number of pixels) and labels are integers, $d \in \textsc{Lab} \subset \mathbb{N}$. We treat images Imag as a subset of $\mathbb{R}^n$ and labels Lab as a finite subset of $\mathbb{N}$. Hence, we define:

$$\mathcal{D} = \{(x_i, d_i) : x_i \in \textsc{Imag}, d_i \in \textsc{Lab}, 1 \leqslant i \leqslant N\}$$

as a *training set* of size N.

There exists a labeling $dec : \textsc{Imag} \to \textsc{Lab}$ which assigns a decision to each image of a beer (this decision corresponds to a beer name), such that:

$$\forall_{(x,d)\in\mathcal{D}} : dec(x) = d$$

We use neural networks to find the best possible approximation of this function.

2.2. Neural networks

2.2.1. Neurons

The concept of neural networks is based on a human brain structure. The building block of a neural network is a neuron - a unit with several inputs and an output. There are a few different types of neurons, each determined by its so called *activation function*. An output of each neuron can be obtained by calculating a weighted sum of its inputs and a *bias* (weights and a bias are neuron's parameters) and applying the activation function to the sum[1]. So, if a neuron has inputs x, weights w, bias b ($x, w \in \mathbb{R}^t$, $b \in \mathbb{R}$) and an activation function ϕ, then its output y is given by a formula:

$$y = \phi(w^T x + b)$$

Each neuron is therefore a function ($\mathbb{R}^t \to \mathbb{R}$) mapping its input to an output.

[1] Some of the most frequently used activation functions ϕ are: sigmoid ($\phi(x) = (1 + e^{-x})^{-1}$), hyperbolic tangent ($\phi(x) = tanh(x)$) and ReLU (Rectified Linear Unit, $\phi(x) = max\{x, 0\}$).

2.2.2. Layers

Neurons are usually combined into layers (neurons in a single layer are not interconnected). A layer's input usually becomes an input of all its neurons. A layer's output consists of outputs of all its neurons. Thus, a single layer with s inputs and t outputs can be thought of as a function $\mathbb{R}^s \to \mathbb{R}^t$. A neural network arises from composition of many consecutive layers.

When it comes to visual recognition, a particularly good performance in this task is achieved by using specific types of neuron layers, such as convolutional, pooling, fully-connected or softmax layer. All of them are described in [Conv] in detail, we will briefly discuss their purpose below.

- Fully-connected layer
  This is the standard and the most frequently used type of layer, in which all neurons are connected with all layer's inputs. The output of such layer (with s inputs and t outputs) can be described with a formula

  $$Y = \phi(W^T x + B)$$

  where $Y \in \mathbb{R}^t$ is a layer's output vector, $x \in \mathbb{R}^s$ is a layer's input vector, $W \in \mathbb{R}^{s \times t}$ is a matrix of weights of each t neurons in a layer, $B \in \mathbb{R}^t$ is a vector of neurons' biases and ϕ is an activation function[2].

- Softmax layer
  This is a layer performing a logistic regression task[3]. Let $Z = (Z_1, \ldots, Z_t) = W^T x + B$ with notation as before. Additionally, we apply a *softmax* function S (where $S = (S_1, \ldots, S_t) \in \mathbb{R}^t \to \mathbb{R}^t$) defined as:

  $$S_i(Z) = \frac{e^{Z_i}}{\sum_j e^{Z_j}}$$

  to Z, obtaining the output $Y = (Y_1, \ldots, Y_t)$ of the layer:

  $$Y_i = S_i(Z)$$

  When performing a classification task[4], this gives an output vector Y easy to interpret. From the definition of S we see, that it outputs a vector of numbers in range $[0, 1]$ summing to 1. Hence, we understand Y_i as a probability that input should belong to the i-th class, represented by i-th neuron.

  Furthermore, if we want to predict a class for an unlabeled example, it is as simple as choosing the class with the highest probability:

  $$dec(x) = \operatorname*{argmax}_i S_i(Z)$$

  for $Z = W^T x + B$ as before.

- Convolutional layer
  This is a type of layer specialized for visual recognition task. Because of this specialization, we arrange neurons spatially and create a specific semantic interpretation. That

[2] here the activation function $\mathbb{R} \to \mathbb{R}$ is applied element-wise to a vector $(W^T x + B) \in \mathbb{R}^t$

[3] `https://en.wikipedia.org/wiki/Logistic_regression`(Retrieved: Monday 13th June, 2016)

[4] in which we assign one of t classes to an input example

said, we form the neurons of a single layer to a three dimensional grid, with the dimensions: depth, height, width. Each neuron of a convolutional layer has weights connected with a small region of a previous layer (e.g., 5x5x3) and shares those weights with all neurons along the second and third dimension (height and width).

This can be presented from another perspective. The shared weights of neurons in a depth slice can be viewed as a single filter. Activation of all those neurons is the same as a convolution of this filter with the outputs of a previous layer - hence the layer name.

There are two main reasons behind using this type of layer. Thanks to weights sharing between some neurons, the number of layer parameters is significantly reduced compared with the use of e.g., fully connected layers. Secondly, there are many neurons with the same weights connected to different areas of the input (or a previous layer). This gives us some kind of translation invariance - the neural network will respond to similar inputs in different regions of the input image.

- Pooling layer
  The main function of this layer is to reduce the layers' size. Commonly, for all disjoint input regions of size 2x2x1 it outputs a maximum of those four numbers[5]. It corresponds to image downsampling and reduces its height and width twice.

2.2.3. Convolutional neural networks architecture

Convolutional neural networks usually use all types of layers described above. Obviously, the most significant one is the convolutional layer. The state-of-the-art networks for visual recognition usually consist of several such layers interleaved with pooling layers (an example of such network is presented in [Deep]). They are followed by a few fully connected layers terminated with the softmax layer (in case of a classification task).

2.3. Learning

In terms of artificial intelligence, *learning* means changing the system so that it can perform same or similar tasks more excellently or effectively in the future.

2.3.1. Loss function

We want to adapt the neural network to label beer images more accurately. For this purpose we will treat learning as an optimization problem - we want to find a labeling function $f : \textsc{Imag} \to \textsc{Lab}$ which minimizes some fixed objective function (also called a *loss function*). By using neural networks, we limit the space of possible solutions to functions representable by them. Let then $o(\theta, x) = (o_1(\theta, x), \ldots, o_t(\theta, x))$ be an output of a neural network with parameters (weights and biases) θ given an input $x \in \textsc{Imag}$ and let's assume that $o_i(\theta, x)$ is a probability that the input image x should be labeled as an i-th class[6].

A typical choice for the loss function in such classification problems is cross-entropy. To determine the error of one training example $x \in \textsc{Imag}$ in this case, we define:

$$L_{x,d}(\theta, x) = -\log(o_d(\theta, x))$$

[5] Maximum can also be replaced by another operation, e.g., an average; the size of regions can also be altered.

[6] this is the case if the last layer of a neural network is a softmax one - see softmax layer description in section 2.2

where d is a correct label for $x \in \text{IMAG}$. A total loss for the dataset $\mathcal{D}$ and a neural network parametrized by θ is then given by:

$$L(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(x,d)\in\mathcal{D}} L_{x,d}(x)$$

2.3.2. Regularization

The loss function suggested above can lead to finding the best (globally or locally) labeling function in terms of cost L. It will optimally choose decisions for data from training dataset $\mathcal{D}$. However, our goal is to find a function performing well on all objects from IMAG, especially the yet unseen ones. Here is where the Ockham's razor principle[7] can be applied - we state that a simple model is better than a more sophisticated one. In this context we interpret *simplicity* as a measure of neurons weights' magnitude.

Hence, we introduce a regularization penalty - a function that will penalize network for having too much weight values. Let R_1 (sometimes called *lasso regularization*) and R_2 (*ridge regularization*) be functions as follows:

$$R_1(\theta) = \sum_{w\in\theta} |w|$$

$$R_2(\theta) = \sum_{w\in\theta} w^2$$

Now we can redefine the loss function to include regularization (with regularization factors λ_1, λ_2):

$$L(\theta) = \left(\frac{1}{|\mathcal{D}|} \sum_{(x,d)\in\mathcal{D}} L_{x,d}(\theta) \right) + \lambda_1 R_1(\theta) + \lambda_2 R_2(\theta)$$

2.3.3. Optimizing loss function

To find the loss functions' optima we use a Mini-Batch Gradient Descent algorithm. In Gradient Descent, at each step of algorithm, we compute gradients of the loss function L and update weights in the direction of steepest descent, i.e. proportional to the negative of the gradient with a proportionality factor α, called the *learning rate*:

$$w := w - \alpha \frac{\partial L}{\partial w}(\theta) \tag{2.1}$$

for each weight $w \in \theta$.

In Mini-Batch Gradient Descent, we do not compute the loss $L(\theta)$ over the entire training set, but only over some subset of training examples (a *batch*) and repeat the weights update (2.1) for all batches in the dataset.

2.3.4. Learning extensions

All steps presented above describe the whole process of training a neural network. Still, there are a few more details that play a significant role in the quality and the effectiveness of learning (see [Learn]).

[7] *Entities must not be multiplied beyond necessity* (from Latin: "Non sunt multiplicanda entia sine necessitate")

After many training iterations, the initial learning rate may turn out to be too high for the final parameters fine tuning. Therefore, it may be helpful to gradually lower the learning rate. Usually the decay is chosen to be exponential, for example the learning rate may be multiplied by some factor (e.g., 0.99) in each training iteration.

Another technique, called RMSprop, was suggested by Geoffrey Hinton in [Hint][8]. It normalizes weights' updates by computing a running average of squared gradients for each weight. The weights' update formula becomes then:

$$\begin{cases} \mathrm{MS}_w := (1-\gamma)\mathrm{MS}_w + \gamma \left(\dfrac{\partial L}{\partial w}(\theta)\right)^2 \\ w := w - \dfrac{\alpha}{\sqrt{\mathrm{MS}_w}} \dfrac{\partial L}{\partial w}(\theta) \end{cases}$$

for each weight w, where γ is a fixed parameter (preferably $\gamma = 0.1$).

[8] also available at `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf` (slide 30)

Chapter 3

Project requirements

3.1. Scope changes

During the academic year the scope of the functionality has been changed several times. All modifications were discussed with the Client and the thesis supervisor. Nevertheless, the core functionality and vision of the application remained intact. In order to provide these basic functionalities, an Android and web applications have been planned. However, after several meetings with the Client, it was decided to change the scope of the project and add an iOS application at the expense of some of the functionality of the web application. Another feature, which was repeatedly discussed, was connected with a social dimension of the application. Either social buttons (for Facebook or Google+ sharing) or possibility of rating and commenting beers had to be chosen for development. After a consultation with the thesis supervisor we decided to implement the second feature.

3.2. Functional requirements

At the beginning of the academic year, we decided to work in agile development methodologies ([Agile]), therefore the functional requirements were formulated in the form of user stories.

3.2.1. User stories

In the initial phase of the project (October - November) a core set of user stories was created. We worked together with the Client to specify all functional requirements as far as mobile applications and web applications are concerned. During the development process we identified new necessary functionalities and decided to add them to the backlog (see 5.3).

Several user roles were introduced:

- *user* - mobile user, who can, e.g., take photos of beers, add beers to cellar, read information about beers,
- *brewery* - web application user, who can, e.g., add new beers or view present beers.

Final list of user stories is as follows:

1. Mobile application
    1.1. As a user, I want to sign in the application so that my personal data can be linked to my account.

1.2. As a user, I want to take photo of a beer label so that I can receive information about this beer.

1.3. As a user, I want to read details about a beer so that I would increase my knowledge about it.

1.4. As a user, I want to rate a beer so that other users will know my opinion about this beer.

1.5. As a user, I want to read beer ratings from other people so that I can make my decision regarding choosing this beer more accurate.

1.6. As a user, I want to add a beer to the cellar so that I can have this beer in my own collection.

1.7. As a user, I want to add a beer to the wishlist so that I will remember to try this beer later.

1.8. As a user, I want to search beers so that I can get information about beers I am interested in.

1.9. As a user, I want to have my own beer photo displayed on beer profile so that I have more personalized experience.

1.10. As a user, I want to browse through available beers so that I can discover new ones.

1.11. As a user, I want to sign out of the application so that another user could use it.

2. Web application

2.1. As a brewery, I want to create an account so that I will have access to the web application.

2.2. As a brewery, I want to add beer produced in my brewery so that users can read information about this beer.

2.3. As a brewery, I want to display previously added beers and their details so that I can check accuracy of previously added data.

3.3. Non-functional requirements

At the initial meeting with the Client we jointly agreed that, from our future user's point of view, the most important non-functional requirement is the look and feel[1] of mobile applications. Therefore, we decided to entrust a professional User Experience Designer and Graphic Designer with the work on graphic projects. Moreover, all graphics should be compatible with guidelines for Android — Material Design[2], and for iOS — iOS Human Interface Guidelines[3]. While planning other non-functional requirements, we mainly focused on user comfort and safety, realized in such values as:

- Security - all users' private information about drunk or wanted beer must be safe and protected, a secure HTTPS protocol should be used for communication via the Internet,

[1] `https://en.wikipedia.org/wiki/Look_and_feel` (Retrieved: Sunday 12th June, 2016)

[2] `https://developer.android.com/design/index.html` (Retrieved: Sunday 12th June, 2016)

[3] `http://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/` (Retrieved: Sunday 12th June, 2016)

- Accessibility - the application has to be easy to read and understand, the color contrast should be high enough to enable visually impaired to use the application,
- Usability - basic functionality should be available offline,
- Simplicity - authentication should not require creating a new account, users should be able to authenticate using popular third party websites (like Facebook or Google),
- Reliability - since the main functionalities are based on communication with the server, the server should be stable and highly available,
- Data Integrity - data updates on server should be delivered to the user side as soon as possible.

Moreover, at the beginning of the academic year, we established some ways to raise the quality and effectiveness of our work:

- Documentation - all major processes should be documented, especially communication with the server and JSONs' format,
- Quality - all written code should be reviewed, vulnerable parts should be covered with tests,
- Performance - communication with server (especially synchronization mechanism) should minimize a consumption of Internet transfer
- Manageability - the process of testing and deployment should be automatic,
- Backup - the created code should have a backup copy at a safe platform.

Chapter 4

Realization

4.1. Machine learning

4.1.1. Prediction model

The prediction model purpose is to classify beer images, which means the model takes a beer image as an input and gives a beer name as an output. The prediction itself consists of two steps.

Firstly, the input image is preprocessed with the OpenCV library. The colorspace is turned from RGB to HSV[1] and the colors are normalized using the Contrast Limited Adaptive Histogram Equalization algorithm[2]. The image is then resized to proper dimensions. For the time being, the scaled height should equal to 230 pixels and width - 170 pixels. If the image's ratio does not match the 230:170 ratio, it is padded with black pixels.

Secondly, the preprocessed input image is fed to the core of the model - a neural network. The neural network mechanics is described in chapter 2.

Neural network realization

To model a convolutional neural network, all layers listed in 2.2 must be implemented. The essential part of each layer implementation is presented below.

- Fully-connected layer

  ```
  lin_output = T.dot(input, W) + b
  output = lin_output if activation is None else activation(lin_output)
  ```

  Here, `W` is a matrix of neurons' weights, `b` is a vector of biases and `activation` is an activation function (sometimes it can be omitted). `T.dot` is a Theano matrix multiplication function.

- Softmax layer

[1] description and rationale available at `https://en.wikipedia.org/wiki/HSL_and_HSV#Use_in_image_analysis` (Retrieved: Friday 27th May, 2016)

[2] description available at `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_histograms/py_histogram_equalization/py_histogram_equalization.html` (Retrieved: Friday 27th May, 2016)

```
y = T.nnet.softmax(T.dot(input, W) + b)
y_pred = T.argmax(y, axis=1)
```

After multiplying the input by a matrix of weights `W`, a softmax function is applied, giving a vector of probabilities as a result. The highest one becomes a final prediction for the given input.

- Convolutional layer

```
conv_out = conv.conv2d(
  filters=W,
  filter_shape=filter_shape,
  image_shape=image_shape,
  subsample=conv_stride
)
lin_output = conv_out + b.dimshuffle('x', 0, 'x', 'x')
output = lin_output if activation is None else activation(lin_output)
```

The convolution of an input with a filter (a matrix `W`) is realized by a Theano library function `conv2d`[3]. Afterwards, the activation function is applied similarly to the case of a fully-connected layer.

- Pooling layer

```
output = downsample.max_pool_2d(
  input=input,
  ds=poolsize,
  ignore_border=True
)
```

Here, the pooling operation is realized with a library function `max_pool_2d` with pooling size as a parameter. It is worth noticing that there are no individual neuron parameters (like a matrix `W`) in this layer.

- Normalization layer

```
mean = T.mean(input, keepdims=True)
var = T.var(input, keepdims=True)
norm_output = (input - mean) / (T.sqrt(var) + epsilon)
output = norm_output * gamma + beta
```

This layer is not mentioned in theoretical analysis as a not typically used one. It is thoroughly described in [Norm], but the idea is to keep the input and output values of all layers in a reasonable range, i.e. with a mean close to 0 and a variance close to 1. Therefore, this layer computes a mean (`mean`) and a variance (`var`) of an input and after normalization the values are shifted again using gradually learned parameters `gamma` and `beta` (initialized to 1 and 0 respectively).

[3] with parameters `filter_shape`, `image_shape`, `subsample` describing the convolution properties

Neural network architecture

As said before, a neural network is a composition of single layers. The chosen network structure (architecture) for beer images labeling is the following (the layer dimensions are given in order: depth, height, width, as in section 2.2):

- Input - an image of dimensions 3x230x170
- Convolutional layer - 30x76x56 neurons
- Pooling layer - 30x38x28 neurons
- Normalization layer - 30x38x28 neurons
- Convolutional layer - 50x17x12 neurons
- Pooling layer - 50x8x6 neurons
- Normalization layer - 50x8x6 neurons
- Fully-connected layer - 50 neurons
- Softmax (output) layer - 25 neurons

A ReLU function[4] was chosen as an activation function in all layers requiring it.

Moreover, the size of the last layer implies the number of beers recognized by the prediction model. Currently this number is equal to 25.

4.1.2. Machine learning unit

The main purpose of this system is to generate a prediction model with a Mini-Batch Gradient Descent algorithm, as described in section 2.3. This part also uses Theano, mainly due to its symbolic differentiation feature. This enables one to automatically calculate differentials with respect to weights, as stated in weights update equation 2.1[5].

Training data

In order to train a neural network, a great amount of training data is needed. This data was collected by all members of the team by taking photos of actual beers. Those images have been labeled using the moderator panel (described in section 4.1.2 below), a tool developed as a part of the project to simplify and speed up the data labeling process. This resulted in collecting a database with 2700 images, containing more than 100 different beers. Then, the most frequent 25 ones were chosen as an actual training set of size 1300.

A dataset of this size is still not enough for a neural network to be trained properly. For this reason, this data had to be artificially extended. To avoid storing a lot of static files, the augmentation took place dynamically, before each iteration of the learning algorithm. The images were randomly (with samples drawn from a normal distribution):

- cropped - this changed an image size from 240x180 to 230x170
- lightened (or darkened) - the intensity of colors was multiplied by a random factor (with mean 1 and a standard deviation of 0.1)

[4] given by a formula $\phi(x) = max\{x, 0\}$.

[5] it allows to avoid complicated computation of differentials, classically handled with a backpropagation algorithm

- rotated - an image was rotated by a random angle (with mean 0 and a standard deviation of 3 degrees)

Those operations increased data diversity and enabled us to effectively apply the learning algorithm. On the other hand, they complicated the realization and required custom implementation of the algorithms.

Mini-Batch Gradient Descent parameters

The learning algorithm results depend on the choice of many algorithm parameters. To properly estimate a quality of a generated model, a train-validation split was used. It implies a division of a training set into two parts: training data, on which the prediction model is trained, and validation data, on which the trained model is evaluated. It allows one to choose the best algorithm parameters according to model performance on the validation set.

Throughout many parameters combinations, the best one included the following parameter values (the meaning of all of them is described in chapter 2):

- batch size - 100 training examples
- number of algorithm iterations (epochs) - 50
- initial learning rate - 0.01
- decay learning rate factor - 0.93
- lasso regularization factor λ_1 - 0 (not used)
- ridge regularization factor λ_2 - 0.005

Moderator panel

In order to prepare the training data needed for the neural network learning process, all collected 2700 images had to be properly labeled. The information required for prediction model training assigned to each beer image includes:

- beer name,
- coordinates of the beer label borders present in the image.

To acquire this kind of data, a specially designed tool was needed to efficiently label all images. This so called *moderator panel* was created in Javascript[6] language as a part of web application (described in section 4.2.2). It means it is available online and it operates on images stored in the main database. The images can be gradually and independently labeled by moderators - people responsible for this task.

The panel itself offers a user-friendly graphical interface which allows the user to mark the label borders in an image and choose an appropriate beer name from a given list of available beers. The results are saved in a database and can be extracted before training the neural network.

[6] `https://developer.mozilla.org/en-US/docs/Web/JavaScript` (Retrieved: Thursday 2nd June, 2016)

4.2. Logical architecture

4.2.1. Overview

The whole system is divided into several components. They form a REST-based[7] architecture and communicate with each other with a simple protocol.

4.2.2. Web application

The web application is composed of two parts: the moderator panel and the brewery panel. Since the former is also a part of the machine learning unit, its details are described in 4.1.2. Therefore, this section will focus entirely on the latter.

Brewery panel

The first version of the system includes data (i.e. beer information) delivered by the Client. However, this data is not adequate for production use of the application due to the insufficient number of beers included. The purpose of the brewery panel is to provide a way for beer companies to populate and manage information about their beers.

To use this panel, the user (i.e., a brewery company) needs to register using a form available on the website. After successful email verification, the user can log into his account, but he cannot edit or add new beers to the database until his identity is manually verified by an admin user. Once that is complete, the brewery user can easily manage information about his beers by using the web interface.

4.2.3. Mobile clients

Mobile clients for Android and iOS have been implemented, thus covering about 98.4% of the market[8]. According to the Client's expertise, users usually have constant Internet connection, therefore most part of functionality is based on this assumption. Details of backend communication are described below in section 4.2.4. It has been decided, for the users' convenience, to keep beers from *cellar* and *wishlist* (along with users' photos) in a local database. The synchronization mechanism has been implemented: by default synchronization starts when Wi-Fi is turned on, but one can change this in the application settings. Moreover, access to the camera of the device is required to send beer label photo to server.

4.2.4. Server

System Overview

There are a few purposes of the system, including the following:

- provide API to resources for mobile devices and web application,
- manage the database and image storage,
- store a prediction model for image recognition.

[7] `https://en.wikipedia.org/wiki/Representational_state_transfer` (Retrieved: Wednesday 8th June, 2016)

[8] `http://www.macrumors.com/2016/02/18/ios-android-market-share-q4-15-gartner/` (Retrieved: Saturday 11th June, 2016)

The system should enable mobile applications to access the database and start image recognition cycle. The recognition process needs to include a way of accessing prediction model and immediately presenting the user with results. Furthermore, the web application must be able to access the same database as mobile devices.

The following sub-parts provide the details on the architecture imposed by contextual factors described above.

Authentication

In order to ensure users' data privacy, secure authentication and authorization system was needed. In an attempt to make it also user friendly, OAuth 2.0 protocol (described in the RFC 6749[9] document) was implemented, which enables users to authenticate with social backends (e.g., Facebook, Google). However, in order to create a scalable solution with a possibility of implementing one's own authentication system in the future, authentication process with social services was combined with Django OAuth Toolkit[10] using Django REST framework Social OAuth2[11]. This solution, as well as the section describing it, were strongly based on the RFC 6749 document.

This approach is different from the standard OAuth 2.0 implementation mainly because it combines two resource servers: the external server, which is one of the social backends' servers, and the internal server, which contains all application related data. The OAuth 2.0 protocol defines four roles. These roles, along with the context described above, lead to the following division:

- server

  Has two roles in the system:

  - the internal resource server,
  - a client making requests to the external resource server on behalf of the resource owner in order to confirm his identity.

- mobile client

  Acts as an intermediary between the server and external resource server, as well as a client making requests to the internal resource server.

- user

  Resource owner, a person that uses a mobile application.

The authentication flow illustrated in figure 4.1 describes the interaction between system components and includes the following steps:

1. The mobile client requests authorization from the resource owner. This step is performed indirectly via the authorization server as an intermediary (the intermediary is not included in the figure 4.1).

2. The mobile client receives an authorization grant, which is a credential representing the resource owner's authorization.

[9] `http://tools.ietf.org/html/rfc6749` (Retrieved: Monday 13th June, 2016)

[10] `http://django-oauth-toolkit.readthedocs.io` (Retrieved: Monday 13th June, 2016)

[11] `https://github.com/PhilipGarnero/django-rest-framework-social-oauth2` (Retrieved: Monday 13th June, 2016)

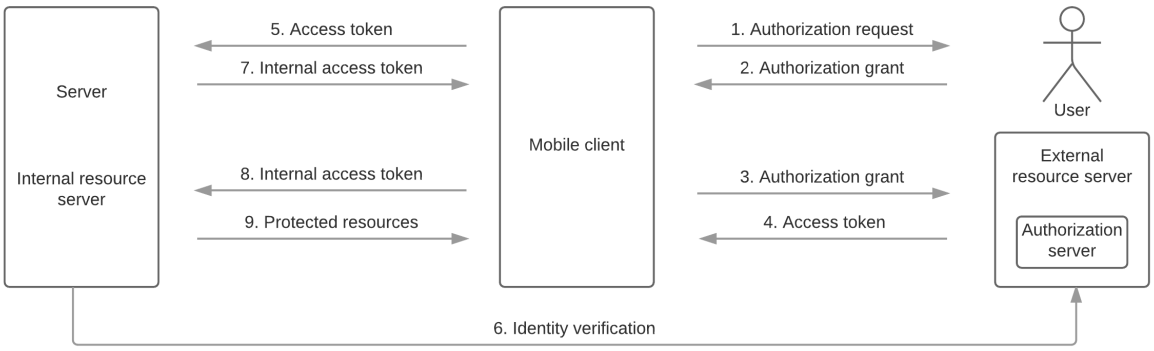

Figure 4.1: Authentication flow

3. The mobile client requests an access token by authenticating with the authorization server and presenting the authorization grant.
4. The authorization server authenticates the client, validates the authorization grant, and if valid, issues an access token.
5. The mobile client presents the access token to the server.
6. The server issues a request to the external resource server in order to get owner's identity.
7. The server authenticates the user using information received from the external resource server and issues an internal access token to the mobile client.
8. The mobile client requests the protected resource from the internal resource server and authenticates by presenting internal access token.
9. The internal resource server validates the access token, and if valid, serves the request.

Data synchronization

Synchronization system should resolve the following problems:

- users should be able to use the application from different devices,
- data stored on mobile devices should be up-to-date with data in the central database.

One must also take into consideration the fact that some resources include large images, so in order not to slow down the application, the number of requests should be optimized.

All user data is stored both on a mobile device and in the central database. Every time a mobile client makes an API request, data is updated on the server. In addition, the mobile client makes asynchronous calls to update resources. In order to reduce data transfer, the client sends only appropriate identifiers along with version labels indicating the last version received from server. Server sends back only those resources in which the version label is different.

However, there is no full synchronization between the server and mobile clients. It means that whenever user logs in from a different device, his resources are not downloaded from server. This feature was not implemented due to its low priority, but since it was carefully considered during architecture designing, it can be easily included in the future versions of the application.

Image recognition process

The recognition process defines a way of invoking prediction model and presenting the user with the results. Its details are described in the image classification mechanism document included in appendix A.

Upon a successful image upload from a mobile client, the server labels it with appropriate status and stores it in Amazon S3[12] bucket. The image is then sent to prediction model, where its status is updated according to recognition results and those results are presented back to the mobile client. The processed image is stored for future use in model training.

Prediction model

Since prediction model training is carried out remotely, a way of placing it on server was needed. This is achieved by providing an endpoint for uploading the model to the Amazon S3 bucket. After training process is complete, the model is placed in the bucket, where it is accessible by server.

4.3. Technology

4.3.1. Web application

Web application was created with Django web framework, which was a natural choice, as other server's components were developed using this technology. For front-end parts, HTML5 and Bootstrap were used for creating the brewery interface and Javascript was used for the moderator panel.

4.3.2. Machine learning unit

The software responsible for machine learning algorithms and for image processing was written using Python[13]. The difficult task of creating and training a neural network was achieved with the use of a Theano[14] library. It offers many arithmetic operations and functions implementation and therefore is suitable for a neural network realization. Besides, Theano features symbolic differentiation, which is needed for computing loss function gradients.

In order to process images before training and during classification of new beer images, an OpenCV[15] library was used (specifically, its Python interface). It offers implementation of many computer vision algorithms necessary for image manipulation.

4.3.3. Mobile clients

iOS

Due to the fact that none of the members of our team had significant experience in programming in Objective-C for iOS, it has been decided to use Swift[16] - new and highly praised, multi-paradigm language for app developers. CoreData[17], persistence framework provided by

[12] `https://aws.amazon.com/s3/` (Retrieved: Friday 27th May, 2016)
[13] `https://www.python.org/` (Retrieved: Friday 27th May, 2016)
[14] `http://deeplearning.net/software/theano/` (Retrieved: Wednesday 8th June, 2016)
[15] `http://opencv.org/` (Retrieved: Wednesday 8th June, 2016)
[16] `https://developer.apple.com/swift/` (Retrieved: Thursday 26th May, 2016)
[17] `https://en.wikipedia.org/wiki/Core_Data` (Retrieved: Thursday 26th May, 2016)

Apple, has been used to keep database on the client side. For communication with backend and some basic view features well known open-source libraries have been applied:

- Alamofire[18] - used for communication with backend,
- SwiftyJSON[19] - used for JSON handling,
- Reachability[20] - used for checking network connection,
- Toast-Swift[21] - used for displaying popups.

Android

A mobile application for Android operation system was created using Android Software Development Kit[22]. It was decided that the app will require minimum API level 15, which means that our application is compatible with devices running Android 4.0.3 (Ice Cream Sandwich) or newer (approximately 97,3% of devices). Several open-source libraries have been used, including:

- Retrofit[23] - used for communication with the server: creating queries and parsing response,
- Picasso[24] - used for downloading and displaying images,
- Espresso[25] - used as a testing framework.

4.3.4. Server

Server software was written using Python, specifically Django REST Framework[26], which comes with everything needed to build an API server right out-of-the-box. Furthermore, Django provides an easy way to create web applications and thus integrate the server with another system component described in section 4.2.2.

The API server is deployed using Heroku[27] with Gunicorn[28] as a HTTP server, which is the recommended production web server for Django applications. Using Gunicorn instead of Django build-in web server allows one to run application concurrently by running multiple Python processes within a single dyno[29].

For data storage relational database is used, specifically PostgreSQL[30]. However, due to the fact that images need to be shared with mobile clients, image data is stored using Amazon Simple Storage Service[31]

[18] `https://github.com/Alamofire/Alamofire` (Retrieved: Thursday 26th May, 2016)
[19] `https://github.com/SwiftyJSON/SwiftyJSON` (Retrieved: Thursday 26th May, 2016)
[20] `https://github.com/ashleymills/Reachability.swift` (Retrieved: Thursday 26th May, 2016)
[21] `https://github.com/scalessec/Toast-Swift` (Retrieved: Thursday 26th May, 2016)
[22] `https://developer.android.com/develop/index.html` (Retrieved: Thursday 9th June, 2016)
[23] `http://square.github.io/retrofit/` (Retrieved: Thursday 9th June, 2016)
[24] `http://square.github.io/picasso/` (Retrieved: Thursday 9th June, 2016)
[25] `https://developer.android.com/training/testing/ui-testing/espresso-testing.html` (Retrieved: Thursday 9th June, 2016)
[26] `http://www.django-rest-framework.org/` (Retrieved: Friday 27th May, 2016)
[27] `https://devcenter.heroku.com/` (Retrieved: Friday 27th May, 2016)
[28] `http://gunicorn.org/` (Retrieved: Friday 27th May, 2016)
[29] more information: `https://devcenter.heroku.com/articles/python-gunicorn` (Retrieved: Friday 27th May, 2016)
[30] `https://www.postgresql.org/` (Retrieved: Friday 27th May, 2016)
[31] `https://aws.amazon.com/s3/` (Retrieved: Friday 27th May, 2016)

4.4. Physical architecture

4.4.1. Overview

The communication of the system components is realized with the HTTPS protocol. A secure version of HTTP is used to ensure data privacy and security. This solution creates modularity, thus the subsystems can be analyzed independently.

The physical architecture of all components excluding the machine learning part is rather self-explanatory.

4.4.2. Server

Since the server is hosted on Heroku platform, its architecture is imposed by Heroku and is described on their official website[32]. There is also no information on Amazon S3 architecture, as details of its design are not made public by Amazon.

4.4.3. Mobile clients

The mobile applications require a typical architecture to be run. The Android application is run in the runtime environment provided by the Android operating system. Correspondingly, the iOS application uses the Objective-C runtime which allows the Swift code to run within a single program.

4.4.4. Machine learning unit

The only part requiring more extensive explanation is the machine learning unit. The neural network training is a resource-intensive task. However, the project low budget implied the decision to use one of the project member's portable computer as a computational unit. The computation was performed on an Intel Core i5-2450M CPU with four 2.50GHz cores. The model and the training data consumed 2GB of RAM memory altogether. Therefore, after installing additional 8GB of RAM in the computer, up to three neural networks could be trained simultaneously.

When it comes to machine learning computations, the typical solutions use graphics processing units (GPU) for calculations. Nevertheless, the Nvidia GeForce GT 525M GPU available on the computer failed to outperform the CPU mentioned above and there are two reasons behind it. Firstly, the 1GB of the GPU's embedded memory was insufficient to store the training data and the prediction model. In this case the data must be constantly transferred from CPU to GPU (and from GPU to CPU) which is not efficient and time consuming. Secondly, the data was dynamically generated (as described in 4.1.2 section), so the images had to be processed by the CPU on every learning iteration.

Finally, it is worth mentioning that an extra care was taken to provide independence between the project subsystems. The machine learning unit is a module that can be easily replaced or moved to another physical device. This allows one to provide additional computing resources for the neural network training in the future.

[32] `https://devcenter.heroku.com/categories/heroku-architecture` (Retrieved: Thursday 9th June, 2016)

Chapter 5

Project management

At the beginning of the project we decided to work using the start-up[1] approach. This was possible because our Client was very open and willing to cooperate. We also decided to use Lean Startup methodology ([Lean]) and commercialize our product in the future.

5.1. Methodology

Start-up technology projects are characterized by a number of factors including constantly changing market, not well defined requirements and strong pressure for fast results. While traditional software development methodologies, such as waterfall cycle model, are strongly criticized in the literature due to their inability to verify and correct previously made assumptions ([Man]) and to their failure to offer any solution for start-up teams, the agile development methodologies ([Agile]) have recently been widely promoted as simple and effective approach to creating software ([Usage]). Due to some uncertainty of our project, we decided to work in the Scrum ([Scrum]) methodology. This enabled us to be flexible and receive feedback from our client on regular basis.

5.1.1. Roles

We assigned the following roles:

- Product Owner - Jan Horubała,
- Scrum Master - Aleksander Matusiak,
- development team - the whole team.

Unfortunately, due to the small number of people in the project, Product Owner and Scrum Master had to be also development team members. Traditional solution, when product owner and scrum master do not code, would dramatically decrease our efficiency and would make it impossible to create a functioning product during the given time.

5.1.2. Sprints length

Clear sprint length has not been set at the beginning, but we rather wanted to experiment with different durations of the sprint. In the initial phase of the project, sprints lasted from 2 to 4 weeks. However, later on we decided to shorten the duration of the sprint to 1 week, which enabled us to deliver new functionalities very often and motivated us to work.

[1] `https://en.wikipedia.org/wiki/Startup_company` (Retrieved: Sunday 29th May, 2016)

5.1.3. Meetings

Due to working on the project only part-time, we were unable to organize daily meetings, so called stand-ups. We decided to have sprint planning, review and retrospective during Thursday classes and short meetings once a week. Additional meetings (sometimes remote, i.e. via the Internet) were organized based on current needs.

5.1.4. User stories

During the sprint planning meetings the product owner proposed user stories on which the development team should work. Later, these stories were decomposed into smaller tasks and added to our issue tracking system. All user stories are described in subsection 3.2.1.

5.1.5. Workflow of issues

At the beginning of the project strict workflow of issues was developed. Atlassian's Jira[2] was used as an issue tracking system. Atlassian's Bitbucket[3] was used as a code repository.

1. Create an issue with an appropriate label indicating which area this issue is connected with (*Android*, *iOS* - for developing mobile apps, *server* - for developing functionalities on cloud server, *ML* - for creating machine learning algorithms).
2. Place the issue in the current or the forthcoming sprint and assign a team member responsible for it (Jira status: **To Do**).
3. Create corresponding branch in the Bitbucket repository (using mechanism provided by Jira).
4. Work to solve the issue, create unit tests which test developed functionality (Jira status: **In Progress**).
5. Finish work when the issue is solved, appropriate documentation and tests are created.
6. Create a pull request for this issue in the Bitbucket web interface.
7. Assign the issue to the team member responsible for testing issues from this area (Jira status: **To Be Tested**).
8. Check whether the functionality has been implemented/bug has been fixed. Conduct code review and check test coverage. Write recommendations or questions in the form of comments under the pull request.
9. Depending on the necessity to make improvements:
   - Mark the pull request as *Accepted* if there are no significant remarks (Jira status: **To Merge**).
   - Do not change the status of pull request if there are small shortcomings that the person responsible for this issue has to correct.
   - Mark the pull request as *Declined* if there are significant and disqualifying flaws. Return to **4**. (Jira status: **In Progress**).
10. Merge into an appropriate master branch (Jira status: **Done**).

[2] `https://www.atlassian.com/software/jira` (Retrieved: Wednesday 25th May, 2016)
[3] `https://bitbucket.org` (Retrieved: Wednesday 25th May, 2016)

5.2. Quality management

To assure a high quality of our product we have followed strict procedures developed during the initial phase of the project. We have conducted code reviews and used pull requests mechanism (see 5.1.5).

What is more, we set up a Continuous Integration[4] (CI) platform for backend server. Semaphore[5] was chosen as a CI platform. Previously created tests were executed before every deploy and it was a precondition for the newest version to be installed on the server. The whole deployment process was fully automized, which enabled us to provide fast and secure (i.e. faultless) way of deploying our application.

Furthermore, CI platform Greenhouse[6] was set up for deploying Android application. When the tests pass, the application is deployed to TestFairy[7]. TestFairy is a system for beta testing mobile applications. Using it enabled us to test our application on different devices, monitor application and system performance, manage application crashes and receive feedback from testers.

5.3. Change management

During the development process some new ideas about our product arose. Whenever one of the team member had an idea of a new functionality or a product improvement, we discussed it during the next meeting. If we found the idea worth pursuing, we added it to our backlog. During the next meeting with the Client we presented our new ideas. They were later thoroughly discussed and the priorities were assigned to them. Keeping this new ideas in the form of tasks in our backlog enabled us to remember them and implement them in due course.

[4] `https://en.wikipedia.org/wiki/Continuous_integration` (Retrieved: Sunday 29th May, 2016)
[5] `https://semaphoreci.com/` (Retrieved: Wednesday 4th May, 2016)
[6] `https://greenhouseci.com` (Retrieved: Sunday 29th May, 2016)
[7] `https://www.testfairy.com` (Retrieved: Sunday 29th May, 2016)

Chapter 6

Division of labour

Every team member participated actively in the initial phase of the project, i.e. created use cases and non-functional requirements together with the Client or designed an appropriate logical and physical architecture. Later, the whole team actively participated in regular team meetings and meetings with the Client. Everyone also participated in writing the Bachelor's thesis and collected training set for machine learning.

The implementation of the system was divided into following areas: machine learning algorithms development, server development, Android development and iOS development. One team member was assigned to each of these areas. He was then responsible for developing functionalities in this area.

Contribution of each member to the project is described below.

6.1. Mikołaj Błaż

1. Implementation - machine learning algorithms development
    - prediction model,
    - panel for moderators.
2. Documentation
    - logical and technical architecture.
3. Bachelor's thesis
    - coordination of work on the thesis,
    - section 1.3 Image recognition,
    - chapter 2 Theoretical background,
    - section 4.1 Machine learning,
    - subsection 4.3.2 Machine learning unit,
    - section 4.4 Physical architecture.
4. Presentation
    - participation in the end-of-semester presentation,
    - participation in preparing the final presentation.

6.2. Jan Horubała

1. Implementation - iOS development
    - client application for devices with iOS operating system.
2. Documentation
    - documentation for use cases,
    - functionalities of mobile application.
3. Bachelor's thesis
    - section 1.1 Motivation,
    - section 1.2 Project vision,
    - chapter 3 Project requirements,
    - section 4.3.3 iOS.
4. Presentation
    - participation in the end-of-semester presentation,
    - participation in preparing the final presentation.
5. Role in the team - Product Owner
    - maintaining backlog,
    - responsibility for the communication with the Client.

6.3. Aleksander Matusiak

1. Implementation - Android development
    - client application for devices with Android operating system.
2. Documentation
    - project vision,
    - class scheme,
    - the outline of how mobile application functions,
    - work methodology,
    - documentation for synchronization mechanism.
3. Bachelor's thesis
    - section 4.3.3 Android,
    - chapter 5 Project management,
    - chapter 6 Division of labour.
4. Presentation
    - *Project vision* presentation,

- participation in the end-of-semester presentation,
- coordination of work on the final presentation,
- delivering a final presentation.

5. Role in the team - Scrum Master
    - ensuring that the Scrum process is followed,
    - creating procedures used during software development (e.g., issues workflow, pull requests mechanism, deployment procedure).

6.4. Michał Moskal

1. Implementation - server development
    - API to be used by mobile applications,
    - web application for breweries.
2. Documentation
    - documentation for server API.
3. Bachelor's thesis
    - section 4.2 Logical architecture,
    - subsection 4.3.1 Web application,
    - subsection 4.3.4 Server.
4. Presentation
    - participation in the end-of-semester presentation,
    - participation in preparing the final presentation.

Appendix A

DVD disc

The DVD disc included contains the following:

- folder android
  - source files with implementation of Android application
- folder artefatcs
  - server API documentation
  - system architecture
  - functional requirements
  - image classification mechanism
  - mobile clients synchronization mechanism
  - product development strategy
  - use cases
  - database schema
  - mobile application features
  - project vision
- folder graphics
  - graphics used by mobile applications
- folder ios
  - source files with implementation of iOS application
- folder poster
  - project promotional poster
- folder prediction model
  - source files with prediction model implementation
  - training data
- folder presentations

  - end-of-semester presentation
  - final presention
  - project vision presentation
- folder server
  - source files with API implementation
  - source files with implementation of web application

Bibliography

[Mur12] Kevin P. Murphy, *Machine learning: a probabilistic perspective*, The MIT Press, pp. 1-24, London 2012.

[Conv] *Convolutional Neural Networks for Visual Recognition* (ConvNets), Stanford CS class, `http://cs231n.github.io/convolutional-networks/` (Retrieved: Wednesday 4th May, 2016).

[Learn] *Convolutional Neural Networks for Visual Recognition* (Learning), Stanford CS class, `http://cs231n.github.io/neural-networks-3/` (Retrieved: Sunday 15th May, 2016).

[Hint] Geoffrey Hinton, *Neural Networks for Machine Learning* course, `https://www.coursera.org/course/neuralnets` (Retrieved: Thursday 1st November, 2012).

[Deep] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, *Going Deeper With Convolutions*, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1-9, 2015

[Norm] Sergey Ioffe, Christian Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, Proceedings of The 32nd International Conference on Machine Learning, pp. 448–456, 2015.

[Lean] E. Ries, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*, Crown Business, 2011.

[Man] M. Flasiński, *Zarządzanie projektami informatycznymi* (Polish) [*IT project management*], Wydawnictwo Naukowe PWN, Warsaw 2006.

[Agile] Manifesto for Agile Software Development, `http://www.agilemanifesto.org/principles.html` (Retrieved: Wednesday 25th May, 2016).

[Usage] A. Begel, N. Nagappan, *Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study*, ESEM '07: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, 2007.

[Scrum] M. Lacey, *Scrum. Praktyczny przewodnik dla początkujących* (Polish) [*The Scrum Field Guide. Practical Advice for Your First Year*], Helion, 2014.