Endbericht zum Berufspraktikum

Marius-Constantin Dinu

BACHELORARBEIT

Nr. 1310307054-B

eingereicht am
Fachhochschul-Bachelorstudiengang

Software Engineering

in Hagenberg

im September 2016

Praktikumsstelle:

Siemens Corporation Corporate Technology
SCCT
755 College Rd E, Princeton, NJ 08540

609-734-6500
www.siemens.com

Kontaktperson:

Dr. Erhan Batuhan Arisoy
Siemens Research Scientist
FH-Prof. DI Dr. Werner Christian Kurschl
University of Applied Sciences Upper Austria Professor

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, September 1, 2016

Marius-Constantin Dinu

Contents

Abstract

This thesis covers the implementation of a mobile application using the Xamarin framework and modern machine learning techniques to recognize handwritten mathematical characters. It focuses on the Android version of the Xamarin framework and refers to the Android *NDK* to enable native C++ code support for the backend implementation. It also includes the required interoperability between C# and C++ using InteropServices. Furthermore the classification of characters is formally described, which is based on support vector machines and artificial neural networks.

Kurzfassung

Diese Bachelorarbeit beinhaltet die Implementierung einer mobilen Applikation unter Zuhilfenahme des Xamarin Frameworks zur Erkennung von mathematischen Zeichen. Hierbei wird der Fokus auf die Android Variante des Xamarin Frameworks gesetzt, um mittels des Android *NDK* das Backend in nativem C++ Code programmieren zu können. Im Zuge dessen, werden auch die InteropServices für die C# und C++ Interoperabilität beschrieben. Ein weiterer Aspekt dieser Arbeit ist die Klassifizierung von Zeichen, welche mittels moderner Machine Learning Methoden umgesetzt wird. Konkret handelt es sich hierbei um die formale Definition von neuronalen Netzen und Support Vector Machines.

Chapter 1

Introduction

1.1 Siemens Corporation Corporate Technology

Siemens was founded 1847 by Werner von Siemens. Its principal divisions are Industry, Energy, Healthineers and Infrastructure & Cities. Corporate Technology is Siemens' central research and development unit. It develops Siemens' technology and innovation strategy, creates and industrializes basic technologies, promotes business excellence through consulting and engineering services, and protects the company's intellectual property.
The Research & Development Center located in Princeton, New Jersey, USA was founded in 1977 and comprises around 200 of the worlds most talented researchers for digital technologies and industrial automation. This covers topics such as additive manufacturing (3D printing), machine learning algorithmic perspectives and PLM software solutions for computer-aided design and engineering.
It is also interesting to know, that not all developed solutions are targeting a direct end-user. They may also be integrated as subparts to existing solutions or be the foundational research base of innovative Siemens products.

1.2 Background

Mobile applications have become one of the most important business models for modern companies. Not only do they provide on-hand information to their customers at any given time, they also emphasize their brand recognition value, retrieve user data and may become an additional source of income [Sta15].
But developing an app might not be an easy task for every *SME*. To cover the variety of operating systems, handle multiple development platforms and operate with different programming languages may become a very challenging project. Alternatively it would be possible to use a web based app approach, embedding browser controls into an app for each native platform to save time and money on development. But in reality, this results in compromising with the user expe-

rience and may not only end up with a negative evaluation of an app, but also with a bad reputation in terms of software quality.
This emerged framework developers to create solutions offering cross-platform toolsets. Xamarin, alongside with Appcelerator Titanium, Embarcardero Fire-Monkey and Apache Cordova offer state of the art cross-platform development frameworks, which focus on building applications for the most common platforms, such as Windows, Android and iOS [Opt15].
These new toolsets are speeding up productivity for application prototyping, shortening the release cycles of an app solution.

1.3 Motivation

Of great interest with regard to these opportunities Siemens has assigned me to develop a mobile application, enabling users to draw numerical characters via touch input and recognition of the digitally hand-written characters. To achieve this task, several operations are required:

- Building a user interface for the input strokes using a cross-platform toolset
- Performing analysis of the stroke paths to determine the character boundaries
- Classifying the characters using machine learning approaches

1.4 Goal

The goal of this thesis is to develop a mobile application prototype, which recognizes touch input and classifies the strokes to defined characters. The prototype app MathQ must normalize the drawn images by resizing and splitting them into separate data strokes. This is followed by feeding the normalized data into a Support Vector Machine (SVM) or Artificial Neural Network (ANN) for classification. The resulting output is determining which type of character an individual stroke represented. Furthermore, according to Siemens the backend has to be implemented in C++, which is requiring the Android *NDK* for interoperability between C# and C++.

1.5 Target

This thesis targets software developers with a profound expertise in C#, C++ and are aware of the common software patterns and paradigms. Basic knowledge of a native App development on Android or iOS is also recommended. In addition it may also require some basic knowledge of applied mathematics.

1.6 Structure

Chapter Xamarin will describe the structure of the Xamarin framework and give an overview of cross-platform development using different approaches.
Chapter Classification offers an introduction to image pre-processing and classification. It also provides an overview of the principles applied in Support Vector Machines and Neural Networks.
Chapter MathQ defines the App architecture and illustrates the major implementation points of the software.
Chapter Conclusion reviews the development results and gives an overview about possible future work.

Chapter 2

Xamarin

Xamarin was founded in May 2011 by the engineers who created the *Mono* projects based on the Common Language Infrastructure (CLI). On the 24th of February 2016, Microsoft signed an agreement to acquire Xamarin, improving their mobile development tools and services. Since then, integrations for Visual Studio, Microsoft Azure or Office 365 provide developers with an end-to-end workflow for native cross-platform app development [Gut16].
The Xamarin extension is available for manual download for Visual Studio 2012, 2013 and integrated in the setup since 2015. In addition to the Visual Studio *IDE*, the Xamarin Studio *IDE* provides opportunities to develop cross-platform apps on Mac and also on Windows. Although the Xamarin Studio version for Mac only offers development support for Android and iOS. Its Windows representation only offers profound Android app development support. The recommended *IDE* is still Visual Studio enabling development not only for iOS and Android Apps, but also for Universal Windows Platform (UWP) Apps and Windows Phone 8, 8.1 Apps or Microsoft Silverlight applications.
The Xamarin SDK offers portable libraries supporting *UI* flow control, notification based services, device specific sensors, etc. across various platforms, which is continuously extended. The framework generally distinguishes between two types of development approaches. The Xamarin.Forms development approach and the classic Xamarin native platform approach. Xamarin.Forms uses Portable Class Libraries (PCL) to enable code compatibility on multiple platforms. This enables possibilities to write code once and deploy to various devices. This includes the *UI* development by giving support to Microsoft's *MVVM* pattern using *XAML* for declarative *UI* designing.
In addition to its advantages, there are also limitations when using portable code. The developed code base has to be supported on all targeted platforms, excluding special features such as File.IO support, Runtime *API*, HTTPClient, etc. It also includes limitations to the use of the ViewModel and *XAML* approach. The included Xamarin.Forms classes differ mainly from classic *XAML* code, offering only a subset of abilities or even restricting the binding abilities.

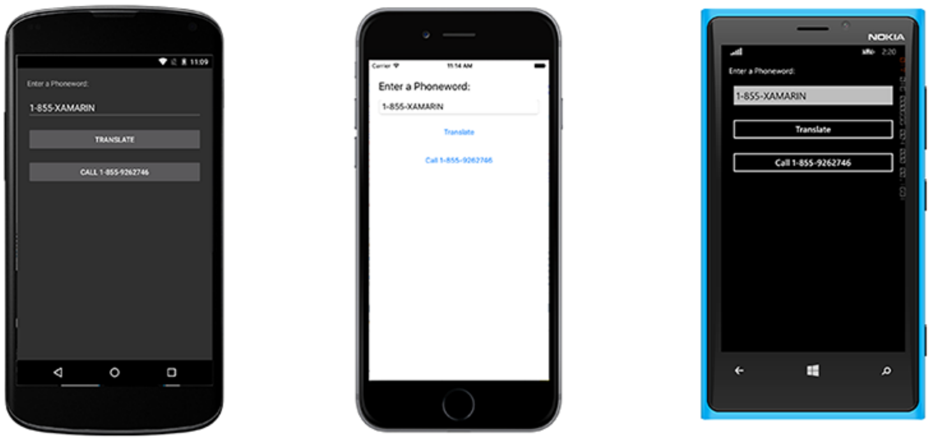


Figure 2.1: Xamarin.Forms App example (left to right: Android, iOS, Windows Phone) [edx16]

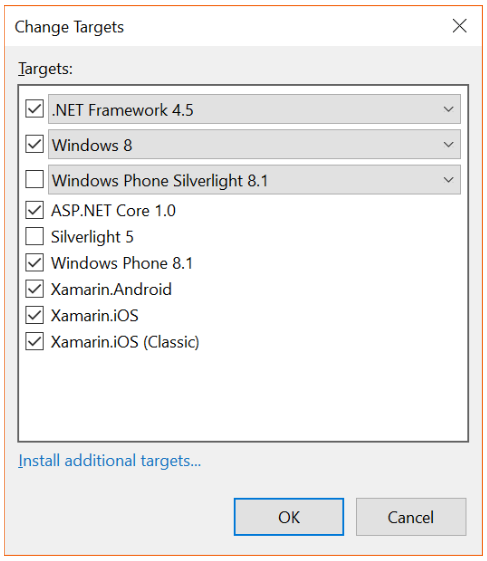
To enable platform-specific features, which are not available in Xamarin.Forms, Xamarin offers either the use of runtime callable wrappers or the translation of native language code to a .NET compliant language.
Whether to use the classic Xamarin approach or write code using Xamarin.Forms *PCL* depends on factors such as:

- testability
- responsiveness
- achievable user experience
- extensiveness of device specific requirements
- time related investments

Xamarin.Form is best used for fast prototyping and offers good testability due to portable code but does not give support for device specific sensors or navigation idiosyncrasies, which may contribute to the overall user experience [Xam16]. Figure 2.1 shows an example of the Xamarin.Forms user experience transition for each native platform after deployment.

2.1 Portable Class Library vs Shared Projects

Due to past license restrictions *PCL* were only available for the Microsoft platform, but since 2013 this has changed and now the standalone releases of *PCL* can be shared and developed across a wide variety of platforms to build even more profound .NET applications. Developing and testing code this way, may not only save time but also money. Xamarin extends the usage of *PCL* by including new supported target platforms such as Android and iOS. This means when developing applications, the target *API* settings influence directly the available


Change Targets

Targets:

- ✓ .NET Framework 4.5
- ✓ Windows 8
- ☐ Windows Phone Silverlight 8.1
- ✓ ASP.NET Core 1.0
- ☐ Silverlight 5
- ✓ Windows Phone 8.1
- ✓ Xamarin.Android
- ✓ Xamarin.iOS
- ✓ Xamarin.iOS (Classic)

Figure 2.2: Target settings in Visual Studio

features for the *PCL* code and have to be the same for all used *PCL* projects. Figure 2.2 illustrates the Visual Studio target settings of a *PCL*.
As an alternative for *PCL* it is possible to use Shared Projects. Shared Projects enable the use of platform specific code by implementing a compiler directive around that code. The following code fragment shows the usage of such compiler directives:

```
public string GetPlatform() {
  var platform = "unknown";
  #if WINDOWS_PHONE
  platform = "windowsphone";
  #else
  #if __ANDROID__
  platform = "android";
  #else

  ...

  #if __IOS__
  platform = "iOS";
  #endif
  return platform;
}
```

This might be a good choice for prototyping or developing small projects using partial classes, although when dealing with multiple developers and more profound architectural structures, developing with Shared Projects ends up with struggling to find the individual settings for each platform. This concludes to a boilerplate code, creating a hardly maintainable application.

2.2 Xamarin.Forms

At first sight, developing with Xamarin.Forms may seem very familiar for *WPF* experienced developers. This initial template structure is similar organized as in *WPF*, when creating a new project. The *App.cs* file represents the main entry point of the Xamarin.Forms application, inheriting from a base class *Application*, but the using declarative are referencing different namespaces.
The life cycle methods offered by *WPF* are based on the *System.Windows* namespace, while the Xamarin.Forms framework uses the Xamarin.Forms namespace. Also in comparison to the generated *WPF App* output class, the Xamarin output does not contain a main entry method. Because of its *PCL* characteristics it is meant to be instantiated by a platform specific implementation via the provided *LoadApplication* method.
In addition both are using different *CodeDom GeneratedCodeAttribute* task parameter for processing the *XAML* markup.
Xamarin.Forms generated class (App.g.i.cs):

```
namespace matheqrec.App {
  using System;
  using Xamarin.Forms;
  using Xamarin.Forms.Xaml;

  public partial class App : Application {
    [System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build
    .Tasks.XamlG", "0.0.0.0")]
    private void InitializeComponent() {
        this.LoadFromXaml(typeof(App));
    }
  }
}
```

WPF generated class (App.xaml.g.cs):

```
public partial class App : System.Windows.Application {
  [System.Diagnostics.DebuggerNonUserCodeAttribute()]
  [System.CodeDom.Compiler.GeneratedCodeAttribute("
    PresentationBuildTasks", "4.0.0.0")]
  public void InitializeComponent() {
    #line 5 "..\..\App.xaml"
    this.StartupUri = new System.Uri("MainWindow.xaml", System.UriKind.
    Relative);
    #line default
    #line hidden
  }

  [System.STAThreadAttribute()]
  [System.Diagnostics.DebuggerNonUserCodeAttribute()]
  [System.CodeDom.Compiler.GeneratedCodeAttribute("
    PresentationBuildTasks", "4.0.0.0")]
  public static void Main() {
    MyApp.App app = new MyApp.App();
    app.InitializeComponent();
```

```
      app.Run();
    }
  }
```

Both *WPF* and Xamarin.Forms can use a code or *XAML* based approach for developing *UI* elements, but unlike *WPF* at the current state the *IDE* support for Xamarin.Forms may seems sometimes unstable and buggy. Also many online tutorials offer only a code based approach, which also is related to some *XAML* Binding disabilities of many Xamarin.Forms classes.

2.3 Xamarin.Android

The native Android support of Xamarin operates with the Android SDK and uses mainly runtime callable wrappers. An experienced Android developer has to familiarize himself with the new *IDE* environment and the C# programming language and is then able to anticipate mostly the available *API*.
Derived from the preset project template structure the main application entry point is the *MainActivity* class. This class is based on the Activity class known from the Android SDK, providing important features such as the life cycle methods *OnCreate*, *OnStop* or *OnPause* and the *FindViewById* method to select *UI* elements.
Xamarin.Android also offers support for working with Android Resource structures. Within the Resources directory of the project sub-directories for drawable, layout and value objects are made available. This enables the usage of the Android designed library for parsing *AXML UI* files. To programmatically access the declared resources, ID entries are declared within the partial resource class (*Resources.designer.cs*) and categorized within partial inner classes of type Attribute, Drawable, Id, Layout and String.
The following code sample illustrates the partial resource class with a declared id:

```
[System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Android.Build.
    Tasks", "1.0.0.0")]
public partial class Resource {
  public partial class Id {
    public const int myButton = 2131034112;
    static Id() {
      global::Android.Runtime.ResourceIdManager.UpdateIdValues();
    }
  }
  ...
}
```

The following sample illustrates the usage of the Id resource via *AXML*.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
<Button
  android:id="@+id/myButton"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="@string/hello" />
</LinearLayout>
```

2.4 Xamarin.iOS

Xamarin allows to also build native iOS Apps by binding all the available Apple *API* framework. There are two possibilities for development. Either directly use a Mac and download the Xamarin Studio or developing on a Windows machine using network access via ssh providing the build and deployment service.
Since Xcode 7 free provisioning terms apply in terms of App development, canceling out the necessity of having an Apple Developer membership. Although this eases the development for iOS platforms, there are still some preconditions required:

- Mac running OS X Yosemite (10.10) or higher
- including iOS SDK (latest version recommended)
- including Xcode (latest version recommended)
- including Xamarin iOS SDK and
- for Windows developers: Windows 7 or higher
- for Mac developers: Xamarin Studio

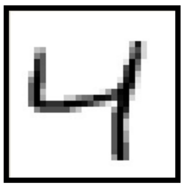
2.5 Xamarin.UWP

Developing for the *UWP* is only possible with Visual Studio 2015, using Windows 10 and with Xamarin.Forms 2.1 or later. This includes the usage of the entire .NET framework including also the Wind32 *API*. The *UWP* offers a guaranteed *API* layer available for all the deployable devices using the Windows 10 architecture and also offering individual features available only on the targeted device family. This enables to build Apps for Desktop, Mobile, Xbox and IoT devices.

Chapter 3

Classification

In computer vision image classification is a very crucial task. Recognizing written digits, letters, characters or drawn pictures has found a wide range of applications from security enhancement, geographic remote sensing techniques to autonomous car control and biological pattern analytics [Liu08].
For humans it seems to be a very easy task to extract information from an image or object, but for machines it has proven to be a fairly complex problem. Developing classification methods can be a very exhausting job and the resulting outcome may request a lot of computational power to work efficiently.
The intent of the classification is to assign each image to a defined category, which then can be used for matching the expected output element. A simple example would be to use a grayscale image 28x28 pixels large and to examine each pixel value (0x00 to 0xFF => 256 values) to create classes based on the overall pattern. Because these patterns are very complex to describe, due to their wide range of free variables (784), it is hard to determine a rule based approach for classification. When trying to match these images with digits classes from 0 to 9, functions have to be declared, describing the characteristics of each class and including the consideration of drawing scatterings. Figure 3.1 on the following page shows a drawn grayscale image of the digit four from the *MNIST* database.
To solve these kind of problems, machine learning algorithms are preferred due to their self-learning feature behavior from the provided data. These operations use different weights for their data transformation and offer a normalized output depending on the used classification type. The computational outcome is the representation of a class type, which can be further processed.
The most common approaches of image classification techniques are separated into two categories, supervised and unsupervised classification **KaSaAg13** To explain the differences of those two techniques it is easier to project the features on a two dimensional scatter diagram. A feature is a measurable property observed from a phenomenon within the dataset. Each dot shown in figure 3.2 on the next page is representing a feature vector based on the given input val-



```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 43 E8 27 00 00 00 00 00
00 00 00 00 3E 51 00 00 00 00 00 00 00 00 00 00 00 00 00 00 78 B4 27 00 00 00 00 00
00 00 00 00 7E A3 00 00 00 00 00 00 00 00 00 00 00 00 00 02 99 D2 28 00 00 00 00 00
00 00 00 00 DC A3 00 00 00 00 00 00 00 00 00 00 00 00 00 1B FE A2 00 00 00 00 00 00
00 00 00 00 DE A3 00 00 00 00 00 00 00 00 00 00 00 00 00 B7 FE 7D 00 00 00 00 00 00
00 00 00 2E F5 A3 00 00 00 00 00 00 00 00 00 00 00 00 00 C6 FE 38 00 00 00 00 00 00
00 00 00 78 FE A3 00 00 00 00 00 00 00 00 00 00 00 00 17 E7 FE 1D 00 00 00 00 00 00
00 00 00 9F FE 78 00 00 00 00 00 00 00 00 00 00 00 00 A3 FE D8 10 00 00 00 00 00 00
00 00 00 9F FE 43 00 00 00 00 00 00 00 00 00 0E 56 B2 F8 FE 5B 00 00 00 00 00 00 00
00 00 00 9F FE 55 00 00 00 2F 31 74 90 96 F1 F3 EA B3 F1 FC 28 00 00 00 00 00 00 00
00 00 00 96 FD ED CF CF CF FD FE FA F0 C6 8F 5B 1C 05 E9 FA 00 00 00 00 00 00 00 00
00 00 00 00 77 B1 B1 B1 B1 B1 62 38 00 00 00 00 00 66 FE DC 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 A9 FE 89 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 A9 FE 39 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 A9 FE 39 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 A9 FF 5E 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 A9 FE 60 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 A9 FE 99 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 A9 FF 99 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 60 FE 99 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 3.1: *MNIST* data sample illustrating the grayscale values for the number four [McC14]

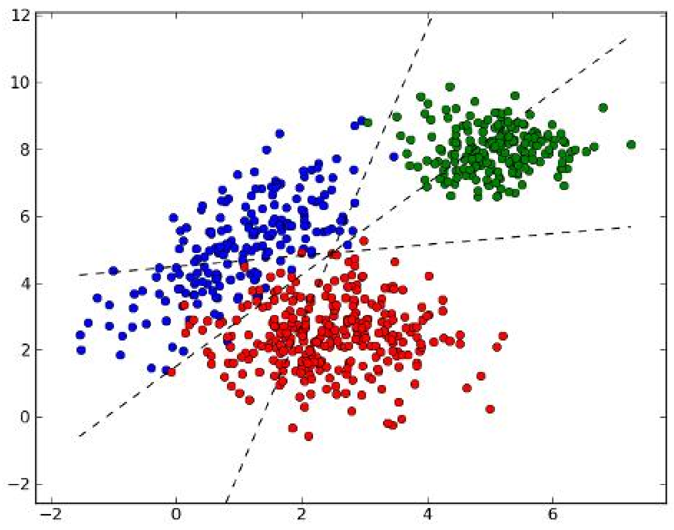


Figure 3.2: Schematic illustration of a classification in a two dimensional space, where each color represents a separate class and the dotted lines mark the segmentation borders created by the algorithm [Alb12]

ues.
Unsupervised classifications do not offer feedback on the predictions and derive structures from input values into clusters, offering a more flexible processing range of data. Expectation-maximization algorithms are used to find the maximum likelihood of the structures within the committed dataset. The created clusters are then used for reference based matches, such as provided by

Samples	Classes
	0
	1
	2

Table 3.1: Example of supervised classification based on figure 3.2 on the previous page

the Amazon services when buying products. Suggestions may appear offering similar products based on the items placed within the shopping cart. Due to unpredictable cluster manifestation, it is hard to post-process the classified results and assign defined semantics for solving handwritten equations. This makes this type of classification unsuitable for the required app tasks.
Supervised classifications are based on having preset output samples matching the defined input values (known as training datasets). *SVM* and Artificial Neural Networks (ANN) are typical examples for supervised classifications. This is achieved by initializing the weights in advance and processing the input values with functions using the chosen parameters. If the calculated result mismatches the expected output, the offset error is measured and the weights are readjusted to better approximate the expected output data samples. This is also known as the backpropagation algorithm. In terms of supervised classification, each colored region from figure 3.2 on the preceding page represents a defined class; e.g. shown in table 3.1 red could stand for the digit zero, green for the digit one and blue for the digit two. If a new image is classified the calculated result will be matched to the class closest to its values. Because each class is clearly defined in advance, it is also possible to assign the expected semantics after classification, making this type of classification suitable for the MathQ app.
Currently the preset classes are:

- digits from zero to nine
- symbols: = + - . ( )

3.1 Image Pre-Processing

According to the previous example of figure 3.1 on the preceding page an image of 28x28 pixels can be represented as a 784-dimensional column vector of variables, which can be used as an input value for a determined machine learning algorithm. Using the entire image size this way has the disadvantage of storing many zero value pixels and can offer slow response time when using remote web based services with higher resolutions. Also when considering the user interactions on the mobile app, the canvas touch inputs received from the device offers the exact positioning of the user drawing movements. This enables to life track the input strokes and save only the entered pixel paths for further computation. Because devices differ in their screen resolution, the size of the canvas also varies. To be able to classify data from multiple devices, their stroke information has to be normalized. A good approach is also to store the screen resolution to help improve the downscaling, considering the given screen ratio. The data normalization is achieved by the following steps:

1. Remove impurities from the stroke collection (largely offset or insignificantly small dots)
2. Downscale the pixel coordinate values from the originally entered strokes
3. Resample the pixel density to fit the unified image size
4. Normalize axis offsets for each stroke
5. Dilate and blur the stroke images to increase the help signify the corresponding drawing for classification

The accuracy of the character recognition may vary according to the following main factors:

- Image resolution of the user input
- The algorithm detecting the bounding boxes for each split character from the equation
- The chosen classification algorithm
- Amount of training data samples for the classification

3.2 Support Vector Machine

An *SVM* uses labeled training data to determine n-dimensional hyperplanes separating the different classes for classifications (finding the support vectors). The support vectors are determined by two vectors representing the hyperplane and one extra vector used for determining the maximum distance. The algorithm used for optimizing the hyperplanes is trying to find the maximum margins between the used support vectors by minimizing the weight vectors used for computation. This optimization problem is nonlinear and can be solved by using the Karush-Kuhn-Tucker (KKT) conditions based on Lagrange multipliers (finding the local maxima and minima for constrained regions [Lus15]). To provide a visualizing example figure 3.3 on the next page illustrates the maximum margin of a hyperplane for a two dimensional feature space.

The hyperplane $g(\vec{x})$ from figure 3.3 can be represented as stated in equation 3.1 on the following page, where $\vec{\omega}^T$ represents the weight vector, $\vec{x}$ the feature vector and ω_0 the weight offset.

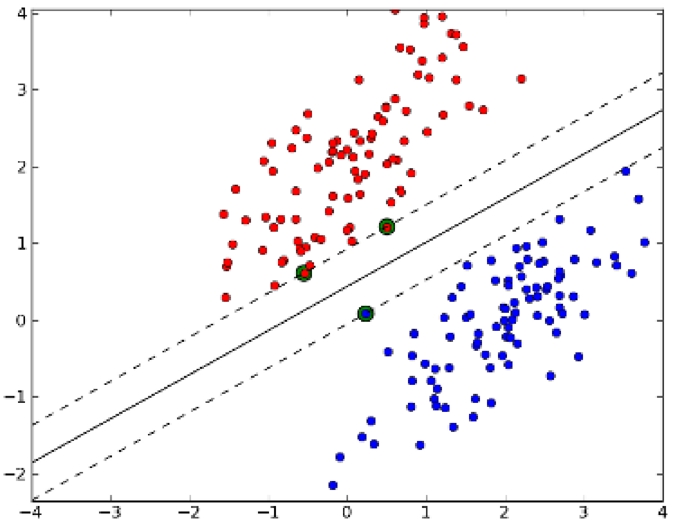


Figure 3.3: *SVM* showing the maximum margin of a hyperplane for linear binary sets [Blo10]

$$g(\vec{x}) = \vec{\omega}^T \vec{x} + \omega_0 \tag{3.1}$$

The optimization problem for finding the minimum weight vector for linear separable classes is shown in equation 3.2

$$\begin{aligned} \underset{\vec{\omega}}{\text{minimize}} \quad & \frac{1}{2}||\vec{\omega}||_2^2 \\ \text{subject to} \quad & \forall i \quad y_i(\vec{\omega}^T \vec{x} + \omega_0) - 1 \geq 0 \end{aligned} \tag{3.2}$$

where y_i represents the according labels of the training examples. This type of linear decision boundary is known as hard margin, clearly separating one class from another.
However to be able to classify features which include distortions and do not always offer a clear separation, slack variables can be added to the equation tolerating small classification errors. This is creating so-called soft margins as shown in the following equation:

$$\begin{aligned} \underset{\vec{\omega}}{\text{minimize}} \quad & \frac{1}{2}||\vec{\omega}||_2^2 + \mu \sum_i \xi_i \\ \text{subject to} \quad & \forall i \quad -(y_i(\vec{\omega}^T \vec{x} + \omega_0) - 1 + \xi_i) \leq 0, \\ & \forall i \quad -\xi_i \leq 0 \end{aligned} \tag{3.3}$$

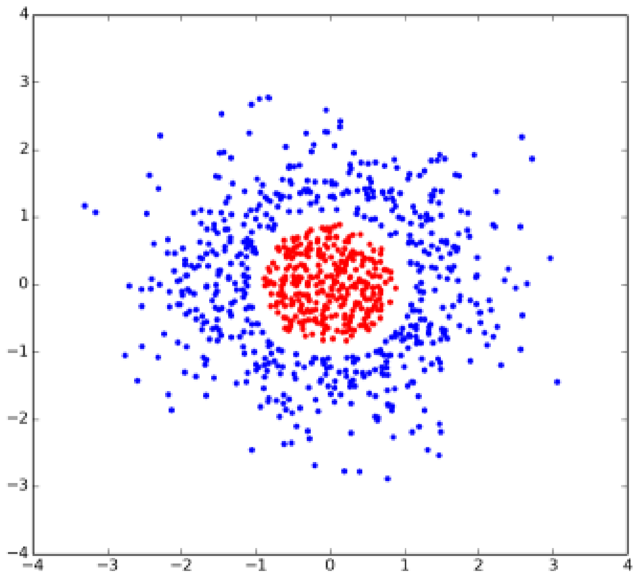


Figure 3.4: Non-linear separable classification problem [Sta09]

To be able to apply this method also for non-linear separable classification problems, such as shown in figure 3.4, a so-called kernel trick can be applied, performing data transformations by operating in a high-dimensional feature space. Creating the higher-dimensional feature vector $\phi(x)$ requires to define the individual operations for each parameter and may require some effort. The following listing offers an example how to create the new feature vector [Nöt15]:

$$\phi : \mathbb{R}^d \rightarrow \mathbb{R}^D, \quad D \geq d$$

$$\phi(x) = \begin{pmatrix} 1 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_1 \\ x_2 \end{pmatrix} \tag{3.4}$$

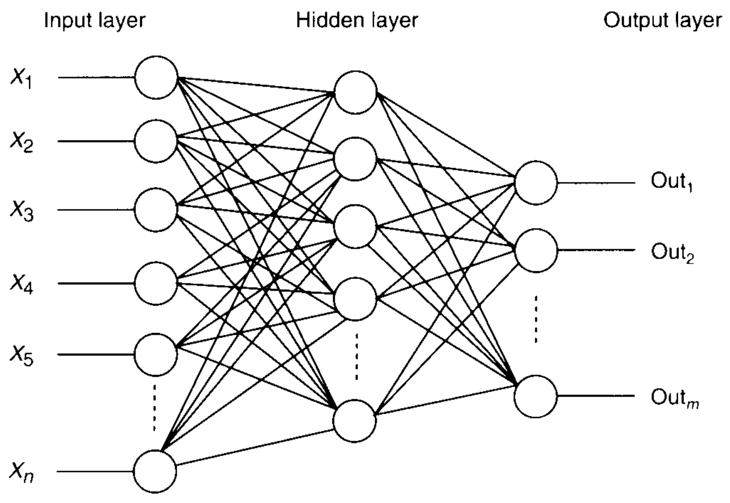


Figure 3.5: Fully connected *ANN* [Mec11]

3.3 Neural Network

Artificial Neural Networks or simply Neural Networks are derived from biological neural networks which define how the brain functions. An *ANN* consists of multiple Neurons which are organized in three layer types. The first layer is called the input layer, the last layer is called the output layer and all other layers in between are denoted as the hidden layers. Each layer, except of the input layer, performs calculations based on its input values received from the neurons of its previous layer. In a fully connected *NN* each neuron layer receives all the neuron output values from its previous layer. Figure 3.5 shows a fully connected neural network.

The input layer is usually only a representative layer for mapping the input values of the feature vector (e.g. mapping the pixels of an image). The output layer neurons represent the determined classes, i.e. the figure 3.5 output neurons could be mapped to different characters or digits such as $textOut_1 = 0, textOut_2 = 2, ...textOut_m = x$. In the hidden and output layers weighted calculations are performed on their input values from their previous layers and compute one output value for each neuron in their layer.

When artificial neurons where developed they where called Perceptrons (see figure 3.6 on the following page), receiving several binary inputs and producing a single binary output. To determine the binary output of a perceptron a threshold function is required as shown in equation 3.5 on the next page.

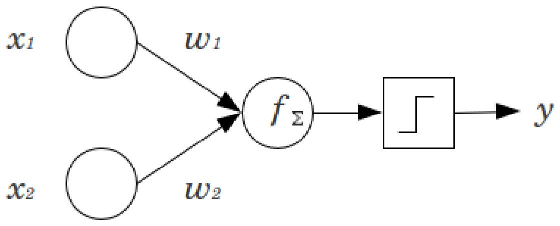


Figure 3.6: Perceptron example with two input values x_1 and x_2

[The11]

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j + b \leq 0 \\ 1 & \text{otherwise} \end{cases} \tag{3.5}$$

In the equation (3.5) $\vec{w}_j$ represents the weight vector (previously self-trained values based on training samples) which is multiplied on the feature vector $\vec{x}_j$ added with a negated threshold value, known as the bias b. Perceptrons where used to implement logical gates, such as the NAND gate and by connecting them in series more complex problems could be solved. Depending on the neurons which have been activated, the overall behavior of the neural network shifts and different output results are computed. The disadvantage of the simple perceptron model is if only one input value is too large or too low the other weighted values may not be significantly large enough to balance the computational result and the entire network can become adulterated always calculating the same result.
Modern *NN* for example use a sigmoid function for the neuron calculations (sigmoid neuron), computing not only binary results, but also values between 0 and 1. Figure 3.7 on the next page shows the characteristics of the sigmoid function.
The sigmoid function $\sigma(z)$ is used to calculate the values of the y axis shown in figure 3.7. The equation for the calculation is declared below:

$$\begin{aligned} z &= \sum_j w_j x_j + b \\ \sigma(z) &\equiv \frac{1}{1+e^{-z}} \end{aligned} . \tag{3.6}$$

The sigmoid function makes the neuron more resistant against extremities from the input values when performing slight changes to its weights or bias. This avoids discontinuity at the point of unsteadiness when switching between 0 and 1 as shown in figure 3.7 on the following page.
It is necessary to fit the parameters in the network (weights, biases) such that the actual network output is close to the expected output, which means the obtained

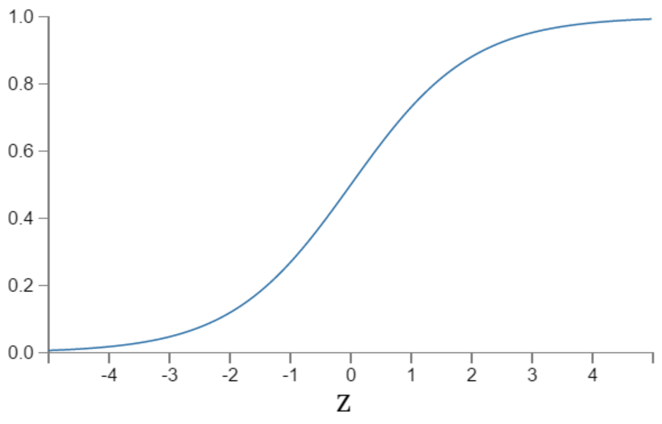


Figure 3.7: Sigmoid function example [Nie16]

error should be small. To achieve optimal network performance the error has to be minimized. The typical minimization technique which is used is the so-called stochastic gradient descent [Oha13]. For this, the error is quantized using a cost function that has to be designed problem specifically. In order to enable the usage of the stochastic gradient descent the cost function C must fulfill the condition shown in the equation 3.7, allowing it to be written as an average over the cost functions C_x for individual training samples x and it must not depend on any activation values of the neural network other than the output values a^L of the previous layer.

$$C = \frac{1}{n} \sum_x C_x \tag{3.7}$$

To adjust the parameters it is necessary to calculate the gradient of the cost function C with respect to the weights w^l and biases b^l in all layers. This parameter optimization is called training. The training is achieved by randomly initializing the weights at the beginning, then feeding each training sample into the network to obtain the predictions. The cost function is then derived with respect to the parameters in the output layer and because each layer depends on the previous layers the gradient has to be backpropagated. This is done by using the gradient from the current layer as shown in equation 3.8, where a^{l-1} marks the output values of the neurons in the previous layer.

$$\begin{aligned} \frac{\partial C}{\partial w^l_{jk}} &= a^{l-1}_k \delta^l_j \\ \frac{\partial C}{\partial b^l_j} &= \delta^l_j \end{aligned} \tag{3.8}$$

The calculated gradient is then used to determine the direction of the steepest descent of the corresponding cost function in dependence of w^l and b^l for their update.
The following listing gives an overview of how the weights and biases are updated based on the previous definitions [Nie16] :

1. **Define a set of training examples by determining the input and output values.**
2. **For each training example x:** Set the corresponding input activation $a^{x,1}$, and perform the following steps:

Feedforward:	For each layer $l = 2, 3, \ldots, L$ compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$.
Output error $\delta^{x,L}$:	Compute the vector $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$.
Backpropagate the error:	For each layer $l = L-1, L-2, \ldots, 2$ compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$.

3. **Gradient descent:** For each $l = L, L-1, \ldots, 2$ update the weights according to the rule $w^l \to w^l - \frac{\eta}{m} \sum_x \delta^{x,l}(a^{x,l-1})^T$ and the biases according to the rule $b^l \to b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$.

Example

The example below shows a very simple neural network written in Python with one input layer (three neurons) and one output layer (one neuron having three weights; bias is zero). It computes the weights according to the prediction table 3.2 and prints the result for each input value after the training is complete (based on [Tra15]):

```
# adding support for multi-dimensional arrays and
# matrices, along with high-level mathematical functions
import numpy as np

# sigmoid function
def sigmoid(x):
    return 1/(1+np.exp(-x))

# derivative / quadratic cost function
def deriv(x):
    return x*(1-x)

# input dataset
X = np.array([ [0,0,1],
               [0,1,1],
               [1,0,1],
               [1,1,1] ])
# output dataset
y = np.array([[0,0,1,1]]).T
```

Inputs			Output
0	0	1	0
1	1	1	1
1	0	1	1
0	1	1	0

Table 3.2: Simple *NN* prediction table example

```
# seed random numbers to make calculation deterministic
np.random.seed(1)
# initialize weights randomly
weights = 2*np.random.random((3,1)) - 1

# iterate 10000 epochs over the training samples
for iter in range(10000):

    # forward propagation of the input values
    inputLayer = X
    # calculate the output result
    outputLayer = sigmoid(np.dot(inputLayer, weights))

    # determine the error
    error = y - outputLayer

    # multiply the error with the
    # slope of the sigmoid at the values in hiddenLayerError
    delta = error * deriv(outputLayer)

    # update the weights of the hidden layer
    weights += np.dot(inputLayer.T, delta)

print("Training output:")
print(outputLayer)
```

The output may be similar to the following listing:

```
Training output:
[[0.00966449]
 [0.00786506]
 [0.99358898]
 [0.99211957]]
```

Hyper-Parameters

The training process is influenced by so-called hyper-parameters, such as learning rate, epochs of repetition, mini-batch size of training samples, etc. These have a great influence on the success of the training process and thus the quality of the *NN*. It may also happen that wrong configurations or too little training samples cause the *NN* to overfit or get stuck on a local minima. Suitable hyper-parameters can sometimes only be found by empirically adjusting the values and experimenting with the training samples.

Chapter 4

MathQ

This chapter will provide an overview of the MathQ app architecture, also including the core implementation patterns used for the frontend, backend and the interoperability in between. In addition it will also focus on a demo implementation regarding an object oriented neural network design.

4.1 Problem

According to the requirements the mobile application has to be implemented using the Xamarin framework with enhanced focus on the Android platform. It should offer touch based user interactions, allowing the user to draw mathematical equations on a canvas. The input coordinates received from the device are then stored and forwarded for further computation to the C++ based backend, which is performing the classification. The classified labels are returned to the frontend to visualize the results. In addition a different functionality is required to train the used machine learning algorithm and collect data provided by users.

4.2 Architecture

The MathQ app is a two-tiered application, offering a visual frontend for the user interactions and a computational backend. The solution structure is split into three main packages. The Frontend package, including the Xamarin related projects, the Backend package, including the business logic projects and the Common package, including the domain model and service projects, which are shared across the Frontend and Backend.
For fast prototyping and to support multiple platforms the Xamarin.Forms approach was chosen for the mobile app frontend development. This results in having multiple projects created when setting up a new environment. A *PCL* project is created which includes the Xamarin.Forms libraries to create shared

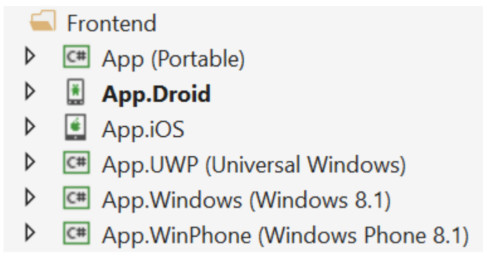

Frontend
App (Portable)
App.Droid
App.iOS
App.UWP (Universal Windows)
App.Windows (Windows 8.1)
App.WinPhone (Windows Phone 8.1)

Figure 4.1: Xamarin frontend project structure

code across all supported platforms. For each supported platform a separate native representational project is also created. The project structure may look similar to the example shown in figure 4.1, splitting the projects into App (Portable), App.Droid (Android), App.iOS, App.UWP, etc.
The backend mainly consists of native C++ projects (Dynamic Shared Libraries) using the Android *NDK* to compile for the target platform. The link between the C# and C++ codebase is a marshalling project on both ends. This also concludes that the domain model classes have to be written in both languages. To share objects between C# and C++ it is required to use *InteropServices* provided by the .NET framework. The C# backend projects are basically build as *PCL* to create a common codebase for all targets.
Figure 4.2 on the following page provides an overview of the main components of the MathQ app. It also shows how these components are related to each other.

4.3 Design

The application design focuses on the patterns used within those modules and provide some implementation examples.

4.3.1 Common

The common projects include the offered services and domain model classes for the C# environment. These projects are also *PCL* enabling them to be used of multiple target platforms. The domain model objects instantiated for solving equations use a stroke based approach, storing the coordinates of the device touch input. This is determined by creating objects of a Stroke type and adding Point type elements to it. The collection of Stroke objects represents a drawing of the corresponding equation. The service classes offer methods to train the classification, collect data and solve equations and have a representational implementation functioning as a delegate for the native code.

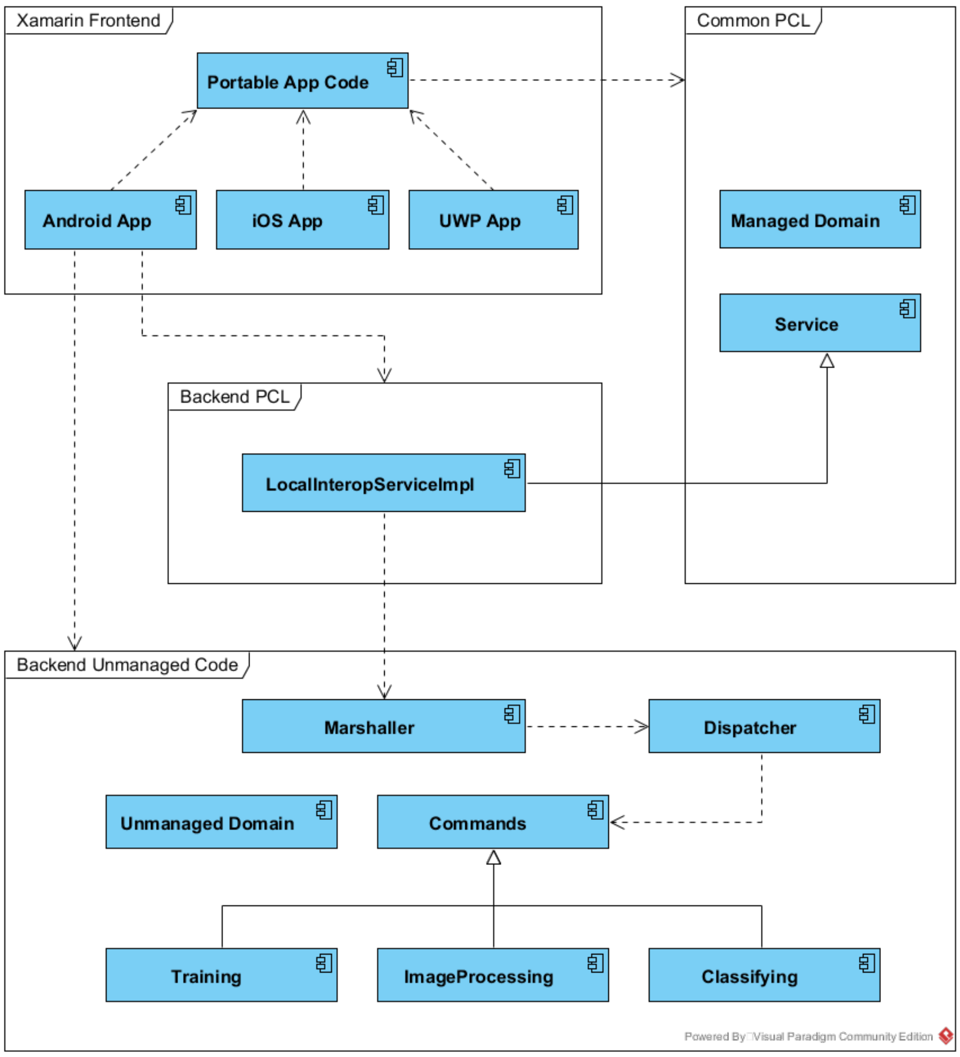


Figure 4.2: Architecture schematics

4.3.2 Frontend

The frontend is based on Xamarin and implemented in C#. The main implementation is written within the portable codebase project used by all native platforms. It uses the *MVVM* pattern to separate *UI* (View) related code from the core functionality, which also enhances the testability of the workflow related classes (ViewController). The Services project represents the Model and is instantiated by using Inversion of Control (IoC) to ensure loose coupling from the implementation projects. This is achieved by dynamically registering and retrieving instances for each concrete platform implementation projects (Droid, iOS, UWP). The currently used *IoC* provider is offered by the MVVM Light framework. To also abstract from the MVVM Light related code an own class *IoCProvider* has been created based on the facade pattern, easing the possibility to replace the MVVM Light framework with other frameworks. The Android project uses the IoCProvider class to register the targeted service implementations. The code fragment below shows the possible usage:

```
[Activity(Label = "App Name", MainLauncher = true)]
public class MainActivity : FormsApplicationActivity {
  protected override void OnCreate(Bundle bundle) {
    base.OnCreate(bundle);
    // register services
    InitializeServices();
    // instanciate portable codebase main class
    LoadApplication(new App());
  }

  private void InitializeServices() {
    // register Android specific services
    IocProvider.Register<IService, DroidNativeService>();

    ...
  }
}
```

4.3.3 Interoperability

PInvoke (Platform Invoke in .NET) offers support for interoperation between managed code running inside the *CLR* and unmanaged (native) code such as used in the Win32 *API* or *Android NDK*. Managed code enables support for garbage collection, code access security to prevent destructive actions when loading code, machine independence due to the *CIL* etc. Unmanaged code runs outside of the *CLR* environment as natively compiled according to the supported *CPU* architecture.
Although *PCL* do not allow references to unportable libraries, the *DllImport* attribute from the InteropServices is still available. This enables to write a C# *API* which is calling native C++ code. The declaration of the interoperation call from C# may look similar to the code in the listing below. The example defines a class *Point* which has two properties x and y. It represents a domain model object and may be used for C# based service interfaces. When used for a native call, it has to be mapped to a data type, which is known by both languages. This means that the C# *Point* class has to be converted into a struct object. Structs can be marshalled and referenced via pointer types and therefore are suitable as parameters for interoperability calls. Also some value types such as int, float, bool, etc. are suitable for interoperability with C++. The resulting IntPtr object has is also marshalled back to a *t_point* object and can then be converted back to a C# *Point* type.

```
using System;
using System.Runtime.InteropServices;

namespace App {
  public class Point {
    public int X { get; set; }
    public int Y { get; set; }
  }

  public class MarshallingExample {
    // The Attribute specifies that the fields layout in the memory
    // is in the same order as at its declarations
    [StructLayout(LayoutKind.Sequential)]
    struct t_point {
      public int x;
      public int y;
    }

    [DllImport("NativeCodeExample")]
    public static extern IntPtr add(IntPtr pointA, IntPtr pointB);

    [DllImport("NativeCodeExample")]
    public static extern void free_alloc_memory();

    public Point Add(Point pointA, Point pointB) {
      // map the Point class to the t_point struct
      var t_pa = new t_point {x = pointA.X, y = pointA.Y};
      var t_pb = new t_point {x = pointB.X, y = pointB.Y};
      // allocate memory for the struct objects
      var paPtr = Marshal.AllocHGlobal(Marshal.SizeOf(t_pa));
      var pbPtr = Marshal.AllocHGlobal(Marshal.SizeOf(t_pb));
      try {
        // marshal the struct pointer with the actual data struct
        Marshal.StructureToPtr(t_pa, paPtr, false);
        Marshal.StructureToPtr(t_pb, pbPtr, false);
        // call the native API
        IntPtr resultPtr = add(paPtr, pbPtr);
        // marshal the pointer result back to the data struct
        var result =
          (t_point)Marshal.PtrToStructure(resultPtr, typeof (t_point));
        // map the t_point struct to the Point class
        return new Point {X = result.x, Y = result.y};
      } finally {
        // free allocated memory from C#
        if (paPtr != IntPtr.Zero)
          Marshal.FreeHGlobal(paPtr);
        if (pbPtr != IntPtr.Zero)
          Marshal.FreeHGlobal(pbPtr);
        // free allocated memory from C++
        free_alloc_memory();
      }
    }
  }
}
```

The DllImport attribute references the name of the assembly containing the native *API*. The *add* method must be declared as *static*, *extern* and have a type suitable signature with the unmanaged code assembly. The input and return parameters of the method are of type IntPtr, which represent a generic address pointer. This interface of the C++ code is declared as stated below:

```
#pragma once

struct t_point {
  int x;
  int y;
};

extern "C" t_point* add(t_point *pointA, t_point *pointB);
extern "C" void free_alloc_memory();
```

The IntPtr address from the C# code is used by the InteropServices to reference at the corresponding *t_point* values in memory, allowing to access the data within the structures.

```
#include "Interface.h"
#include <vector>

using namespace std;

// store the pointers ready to be removed
static vector<void*> alloc_mem_ptrs;

t_point* add(t_point *pointA, t_point *pointB) {
  auto result = new t_point;
  result->x = pointA->x + pointB->x;
  result->y = pointA->y + pointB->y;
  // safe the object reference to call the free_alloc_memory
  // method after the usage is done
  alloc_mem_ptrs.push_back(result);
  return result;
}

void free_alloc_memory() {
  // delete all added objects
  for (auto it = alloc_mem_ptrs.rbegin();
        it != alloc_mem_ptrs.rend();
        ++it) {
    delete *it;
  }
  // remove pointers from vector
  alloc_mem_ptrs.clear();
}
```

It is very important to safe references to allocated objects and to delete them from the memory, otherwise the application will create memory leaks.
The usage of the interoperability *API* may look similar to the following example:

```
var marshal = new MarshallingExample();
```

```
  var result = marshal.Add(
    new Point {X = 1, Y = 2},
    new Point {X = 3, Y = 1});
  // Prints the following to the console: X: 4, Y: 3
  Console.WriteLine(\$"X: {result.X}, Y: {result.Y}");
```

Conditions

Using PInvoke with the Android *NDK* requires to create C++ based Cross-platform Android projects in Visual Studio. In addition some project settings have to be modified, which includes using the *GCC* for compilation, setting some linker options, defining the target Android *API* level and exporting shared object (.so) files instead of dynamically linked library (.dll) files. Shared object files are required to be included in the linking list at compile time.
Using PInvoke with Win32 libraries requires to declare *__declspec(dllimport)* and *__stdcall* to the method signatures in the header files:

```
  ...
  extern "C" __declspec(dllimport) t_point* __stdcall add(t_point *pointA,
       t_point *pointB);
  ...
```

4.3.4 Backend

The backend includes a marshaller converting the interoperability struct objects to the native domain model objects. Furthermore a dispatcher is implemented using the command pattern to create commands steering the classification process. This also allows a more flexible implementation, when adding multiple workflow sequences to the existing implementation. The classification may contain a *SVM* or *NN* implementation based on chapter 3 and retrieves the predicted result after the command execution back to the marshaller.

Chapter 5

Conclusion

5.1 Evaluation

Xamarin

Cross-platform development with Xamarin is extremely beneficial, saving time and costs when prototyping a new software solution for multiple target platforms. Also the achieved performances at runtime do not compromise in comparison to natively developed applications. The downside is that there are still some instabilities with the development environment and that not all *XAML* features are supported when using Xamarin Forms compared to the *WPF*. Also the overall architectural complexity grows rapidly and it may become a great challenge to handle errors at compile time or runtime. Furthermore, when using native C++ code it is fairly hard to debug and determine the origins of the occurring issues.

SVM vs Neural Network

Currently SVM*s* are more often used due to fewer hyper-parameters and because they are guaranteed to find the global optimum, not only the local one. But *NN* are gaining more and more importance due to their extreme flexible and adaptive feature learning of basically any data structure and because they do not require to manually derive features. In addition a well trained *NN* can learn much faster and achieve a higher classification accuracy with less training datasets in comparison to a *SVM*.

5.2 Future Work

Currently the backend only splits the drawn equation into associated stroke parts, which then are classified to the determined characters. The resulting values are then concatenated and returned as a string representation of the equation for visualization. To be able to compute a drawn equation it is necessary to determine the semantics of each symbol. This requires structural analysis of the drawing and building up a *AST*, which then evaluates the detected statements. In addition also the set of character can be increased to support more characters. The implementation of the backend may also be ported to the C# environment as a *PCL* to support all target platforms and avoid marshalling between managed and unmanaged code.

5.3 Experiences

Xamarin

Although there are still some issues, it was a great experience to develop an App with Xamarin and I definitively can recommend everyone to give it a try. I also believe it will become more and more widespread, due to its great integration in a very profound development environment and it will improve the functionality of its framework.

Machine Learning

Machine learning is a very interesting topic and enables a huge field for researching and experiments with very complex data models. In my opinion it seems to become even more adopted not only by large companies, such as Siemens, Facebook, Microsoft or Google, but also by medium-sized companies. The data that can be processed using learning algorithms is very important not only for predicting the overall user behavior to determine market trends, but also to develop more sophisticated solutions for customer needs. On the other hand it is also very difficult to determine which problems can and cannot be solved using machine learning techniques, such as *SVM* or *NN*. Not all problems are suitable and not all solutions may compute a satisfactory result in an appropriate time-frame. In addition it may also be a very time consuming task to implement a customized and very performing solution, requiring a lot of knowledge and experience. I considered the hardest part not related to understanding or using a functioning *ANN* or *SVM*, but it is very challenging to perform the transition from a theoretical model to a customized implementation and finding suitable parameters to achieve an optimized result. Nevertheless this thesis was an great experience and I will definitely continue studying this field.

Development Process

The MathQ project was developed using Scrum. Every Sprint was scheduled for a two weeks period and included Daily Scrum meetings to keep track of the current development progress. Because the tasks were well defined at the beginning, it was straightforward to keep up with the estimated time schedule. To avoid higher learning curves and keep in direct sync with the other teammates, no electronic means were used for the Product Backlog.

References

Literature

[Alb12] Davide Albanese. *High-performance library built using the Python programming language*. Softpedia. 2012. URL: http://www.softpedia.com/get/Programming/Components-Libraries/mlpy.shtml.

[Blo10] Mathieu Blondel. *Machine Learning, Data Mining, Natural Language Processing*. Mathieu's log. 2010. URL: http://www.mblondel.org/journal/2010/09/19/support-vector-machines-in-python/.

[edx16] edx Inc. *Introduction to Xamarin.Forms*. Online Courses. 2016. URL: https://courses.edx.org/courses/course-v1:Microsoft+DEV215x+1T2016/info.

[Gut16] Scott Guthrie. *Microsoft to acquire Xamarin and empower more developers to build on any device*. Tech. rep. 2016. URL: http://blogs.microsoft.com/blog/2016/02/24/microsoft-to-acquire-xamarin-and-empower-more-developers-to-build-apps-on-any-device/#sm.00019758clfs1drgxwh1qe9i9232j.

[Liu08] Cheng-Lin Liu. *Classification and Learning Methods for Character Recognition: Advances and Remaining Problems, Chapter Machine Learning in Document Analysis and Recognition Volume 90 of the series Studies in Computational Intelligence pp 139-161, Springer*. 2008.

[Lus15] Richard Lusby. *Karush-Kuhn-Tucker Conditions, DTU Management Engineering*. University Lecture: 42111 Static and Dynamic Optimization. 2015. URL: http://www.kurser.dtu.dk/2015-2016/42111.aspx?menulanguage=en-gb.

[McC14] James D. McCaffrey. *Working with the MNIST Image Recognition Data Set*. James D. McCaffrey Blog. 2014. URL: https://jamesmccaffrey.wordpress.com/2014/06/10/working-with-the-mnist-image-recognition-data-set/.

[Mec11] Mechanical Forex. *Neural Networks in Trading*. Mechanical Forex. 2011. URL: http://mechanicalforex.com/2011/06/neural-networks-in-trading-how-to-design-a-network-for-financial-forecasting-in-eight-simple-steps.html.

[Nie16] Michael A. Nielsen. "Neural Networks and Deep Learning". In: (2016). URL: http://neuralnetworksanddeeplearning.com/.

[Nöt15] Elmar Nöth. *Pattern Recognition*. University Lecture: Friedrich-Alexander-Universität Erlangen-Nürnberg, Chair of Computer Science 5. 2015. URL: http://univis.uni-erlangen.de/formbot/dsc_3Danew_2Fresrep_26dir_3Dtech_2FIMMD_2FIMMD5_26ref_3Dresrep_26lang_3Den.

[Oha13] Tong Zhang Ohad Shamir. *Stochastic Gradient Descent for Non-smooth Optimization: Convergence Results and Optimal Averaging Schemes, JMLR Workshop and Conference Proceedings Volume 28: Proceedings of The 30th International Conference on Machine Learning: 71-79, ICML*. 2013.

[Opt15] Optimus Information Inc. *Cross-Platform Framework Comparison*. Tech. rep. 2015. URL: http://www.optimusinfo.com/blog/cross-platform-framework-comparison-xamarin-vs-titanium-vs-phonegap/.

[Sta09] StackOverflow. *Difference between a linear problem and a non-linear problem*. StackOverflow. 2009. URL: http://stackoverflow.com/questions/1148513/difference-between-a-linear-problem-and-a-non-linear-problem-essence-of-dot-pro.

[Sta15] Statista GmbH. *Income for mobile apps in Germany*. Tech. rep. 2015. URL: http://de.statista.com/statistik/daten/studie/173810/umfrage/umsatz-mit-mobilen-apps-in-deutschland-seit-2009/.

[The11] The Glowing Python. *The Perceptron*. The Glowing Python. 2011. URL: http://glowingpython.blogspot.com/2011/10/perceptron.html.

[Tra15] Trask. *A Neural Network in 11 lines of Python*. iamtrask Github. 2015. URL: http://iamtrask.github.io/2015/07/12/basic-python-network/.

[Xam16] Xamarin Inc. *Xamarin Documentation*. Tech. rep. 2016. URL: https://developer.xamarin.com/guides.

Acronyms

ANN	Artificial Neural Network.
API	Application Program Interface.
AST	*Abstract Syntax Tree*.
ATG	*Attributed Grammar*.
AXML	Article XML Markup Language.
CIL	Common Intermediate Language.
CLI	Common Language Infrastructure.
CLR	Common Language Runtime.
CPU	Central Processing Unit.
GCC	GNU Compiler Collection.
IDE	Integrated Development Environment.
IoC	Inversion of Control.
KKT	Karush-Kuhn-Tucker.
MER	Mathematical Equation Recognition.
MNIST	Mixed National Institute of Standards and Technology.
MVVM	Model-View-ViewModel.
NDK	Native Development Kit.
NN	Neural Network.
PCL	Portable Class Library.
SDK	Software Development Kit.
SME	Small and Medium-Sized Enterprises.
SVM	Support Vector Machine.
UI	User Interface.
UWP	Universal Windows Platform.
WPF	Windows Presentation Foundation.
XAML	Extensible Application Markup Language.

Glossary

Abstract Syntax Tree A tree representation of the syntactic source code structure.

Attributed Grammar Formal grammar description of a specified programming language with attached attributes, which are evaluated within the *AST* nodes by a parser or compiler.

CodeDom Microsoft's language independent Object-model used for automated source code generators.

Mono Open source implementation of the ECMA-335 standard, based on the *CLI*.

Xamarin Framework for cross-platform application development.

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —

Breite = 100 mm
Höhe = 50 mm

— Diese Seite nach dem Druck entfernen! —