Cloudifier.NET: Automated Software Migration, Translation and Development based on Massively Parallelized Augmented Shallow Predictive Models

Nr Ver	Date	Authors	Company	Project
1	08.01.2017	A.I.Damian, O. Bulie	Cloudifier SRL	Platforma de migrare automatizata in cloud a aplicatiilor si sistemelor informatice clasice Cloudifier.NET Analiza modelelor euristice
2	31.02.2017	A.I.Damian, O. Bulie	Cloudifier SRL	Platforma de migrare automatizata in cloud a aplicatiilor si sistemelor informatice clasice Cloudifier.NET Analiza modelelor euristice – machine learning

Contents

Abstract

The field of automated and generative software development is one of the fields that received generous attention from both academic and commercial areas computer scientists for the past 40 years. With the recent advances in the areas of predictive machine learning models and massive parallel computing a new set of resources is now potentially available for the computer science community in order to research and develop new truly intelligent and innovative ways of automatic software development. In or paper we show that computer software full user interface development task can be fully automated. We are proposing a pipeline machine learning model augmented with GPU parallel computing for the development of user interface with related views, models and controllers. Finally we show that the entire process of developing rich online applications can be fully automated and delivered through web services. Our proposed approach will be used in a scientific experiment for the purpose of generating fully functional user experiences including models and controllers beside the actual views.

Introduction

State-of-the-art and beyond research topics such as image classification [1] [2] and image semantic segmentation [3] provide new opportunities for various machine learning related tasks ranging from recognition of a cat in an image, its actual position in the scene and the definition of the whole scene attributes such as all objects with dimensions and positions. Prestigious image recognitions competitions such as ImageNet [4] demonstrate am amazing rate of improving of over past 10 years both related to the usage of Deep Learning and particularly deep convolutional neural networks and also to the use of GPU based massive parallel computing training of the machine learning models – from winner of 2010 competition that obtained a accuracy of 72% up to the winner of 2015 competition with a accuracy of over 96%.

Nevertheless, the advancement of GPU based hardware infrastructures capable of delivering massive parallel computing for machine learning tasks at very low costs has only been exploited by the area of deep learning and deep neural networks in particular.

In our paper we will propose several approaches: (i) a web-services based model for the deployment of model training and prediction on GPU massive parallel computing backend and (ii) an approach to automatic software design, development and deployment for user experience modules based on a pipeline shallow machine learning model. We will argue that our proposed pipeline model will be able to analyze from human drawn naive interface sketches to raster or vector images and up to captured screens for re-engineering or software translation tasks. Basically the ensemble model will be capable of performing several tasks such as (a) automated software translation of user experience modules from legacy applications to cloud environments, (b) automatic development and deployment of UX modules based on digital designer sketches, (c) automatic development of UX prototypes based on direct user input (raw hand drawings)

Related work and approach

Our work relates to the most recent advances both in the field of High Performance Computing and Machine Learning. The entire field of Artificial Intelligence tends to become more and more about improving predictions and predictive models as Goldfarb et al [5] argue in the paper "Managing the Machines". Predictive modelling can now be used for extremely varied tasks ranging from constructing expert system chat-bots [6] based on Long Short Term Memory deep recurrent neural networks that would use natural language capability (reading and writing) up to melanoma recognition in dermoscopy images [7].

As previously mentioned recent advances since 2010-2011 in computer processing, storage and communication power – and particularly in the area of GPU based numeric/scientific massive parallel processing [8] - brought a tremendous improvement to existing or ongoing approaches such as deep convolutional neural models. Currently GPU technologies such as NVIDIA Pascal architecture offer at acceptable price levels for high-performance computing power able to deliver over 10 TFLOPS of parallel computing on more than 3500 numeric computational cores. The usage of GPU based computing is currently mostly applied in deep learning – both research experiments and production development – however little to no attention is given to the employment of GPU parallel computing resources in shallow machine learning techniques such as decision trees, multi-variate linear and logistic regression, Naive-Bayes models. Nonetheless the current advances and the state-of-the-art prediction scores achieved by the deep learning scientific experiments give little room to further scientific development and advances of known shallow models techniques. Even more, companies such as Microsoft, Google or IBM currently propose online framework-engines that allow on-demand web-based prediction model creation and experimentation [9] that are entirely used for off-line training and prediction experiments, running on CPU based infrastructures.

Taking previous presented aspects into account we are proposing a set of new modern approaches that will potentially empower shallow machine learning models with the capability using GPU based parallel computing resources. Also another of the supporting aspects that we will use in our approach is based on the most important features of shallow machine learning models: the ability of the model to self-explain itself (albeit at the expense of prediction power shown by deep models with massive hidden and less than explanatory weight spaces).

In the rest of the paper we will address the previous mentioned subjects from three perspectives: (A) a simple approach for the efficient provisioning of GPU based parallel computing resources based on REST services and (B) the potential innovations of augmenting with massive parallel computing the exiting shallow machine learning techniques especially in the area of on-line learning and real-time prediction applications with and (C) an innovative approach to real-time automated programming of UX modules based on augmented shallow machine that will take advantage of both the REST based computing provisioning approach and the augmented shallow machine learning models.

REST based massive parallel computation services

Currently several programming approach options are available for programming GPU parallel computation engines both for scientific purposes and for engineering/production and probably the most widely used are OpenCL [10] and CUDA [8]. Basically both mentioned models use the same technique with only slight differences. The two most fundamental aspect in regard to the computational approach consists in the two central entities consisting in *kernel* – that is the function containing the actual code to be run in parallel - and the *devices* – the actual computational devices that will run the *kernel* in parallel. Traditionally the *kernel* is a function described by the the following generic algorithm:

Algorithm 1 *Kernel*(*Data*, *BlockInformation*)

```
        cID ⇐ get coordinates tuple from BlockInformation
        Modify Data[cID] with hard coded <operation>
Return
```

Note that in above example *BlockInformation* defines the actual data segment that a particular *device* is processing within *Data* and <*operation*> denotes the hard-coded (compiled) operation that the *kernel* is computing for the particular block of *Data* it receives. For the particular case of matrix dot product we can have each *device* compute with its *kernel* the dot product of two vectors (the row vector of 1st matrix and the column vector of 2nd matrix).

$$A * B = A^T B = \sum a_i b_i \qquad (1)$$

Finally we have the following two code snippets for computing (1) in OpenCL and CUDA based *kernels*:

Code 1 OpenCL Kernel for Matrix-Matrix dot product

```
__kernel void OpenCLMatrixDotKernel(const int A_ROWS,
                                 const int BC_COLUMNS,
                                 const int A_COLUMNS,
                                 const __global float* A,
                                 const __global float* B,
                  __global float* C)
{
    const int Row = get_global_id(0);
    const int Col = get_global_id(1);

    float acc = 0.0f;
    for (int c=0; c<A_COLUMNS; c++) {
        acc += A[c*A_ROWS + Row] * B[Col*A_COLUMNS + c];
    }
    C[Col*A_ROWS + Row] = acc;
}
```

Code 2 CUDA Kernel for Matrix-Matrix dot product

```
__global__ void CUDAMatrixDotKernel(Matrix A, Matrix B, Matrix C)
{
       float acc = 0;
       int row = blockIdx.y * blockDim.y + threadIdx.y;
       int col = blockIdx.x * blockDim.x + threadIdx.x;
       if(row > A.height || col > B.width)
              return;
       for (int e = 0; e < A.width; ++e)
              acc += A.elements[row * A.width + e] *
                     B.elements[e * B.width + col];

       C.elements[row * C.width + col] = acc;
}
```

As previously mentioned, in the current paper we will present a REST [11] approach for cloud computing based provisioning of parallel computation services. This approach will enable the implementation and consumption of a web based service that will have the following generic algorithm based on FIFO allocation

Algorithm 2 *ParallelProcessingREST*(*Request*)

```
For each Kernel, Data pair in Request
    Kernel ⇐ get kernel source code from Request
    Data ⇐ get dataset from Request by value or by reference from external source
    Available ⇐ get available devices from existing cluster based on adaptive Algorithm 3
    If Available is valid then
        Off-load Kernel and Data on Available devices
        Get off-load Result and append to Response
        Else
                $Response \Leftarrow$ Overhead time exceeds computing time
Return $Response$
```

The actual GPU execution strategy of the off-loaded kernels will be based on an adaptive algorithm that will take into consideration server-load, task complexity and GPU load/preparation overhead.

```
Algorithm 3 ResourceAllocation($Kernel$, $Data$)
    $cFLOPS \Leftarrow$ evaluate needed FLOPS for $Kernel$ and $Data$
    $Resources \Leftarrow$ Compute needed devices based on $cFLOPS$
    $ComputeTime \Leftarrow$ Aproximate total execution time based on $Resources$, $cFLOPS$
    $WaitTime \Leftarrow$ Off-load overhead time + time until devices are available
    If $ComputeTime > WaitTime$
        $Result \Leftarrow$ Queue position or actual allocated $devices$
    else
        $Result \Leftarrow$ allocation invalidated
Return $Result$
```

Augmented shallow machine learning pipeline

In our proposed approach for advancing shallow machine learning by GPU parallel computing augmentation we will argue that shallow models pipelines can achieve comparable results with deep learning models for various applications. Although most of the state-of-the-art research in the area of machine learning is focused on deep learning models the classic machine learning models with shallow hidden layer structure what two main characteristics that deep models lack: self-explain ability and algorithmic robustness for online and real-time environments. In order to continue we will briefly analyze the general case of supervised learning.

In supervised learning we have the following to main components in the learning process:

$$X \epsilon S^{MxN} \qquad (2)$$

$$y \epsilon L^{M} \qquad (3)$$

$$h(X) = hypotesis_{model}(x_i | \theta \in \Theta) = \hat{y} \qquad (4)$$

$$\Lambda(\theta) = L_{model}(h, X, y) \qquad (5)$$

$$R(\theta) = Regularization_{model}\ (h(x|\theta)) \qquad (6)$$

$$\mathit{\Omega}(\theta) = \mathit{\Lambda}(\theta) + R(\theta)\,, \qquad \theta \in \mathit{\Theta} \qquad (7)$$

$$\theta^{*} = \underset{\theta \in \mathit{\Theta}}{\operatorname{argmin}}\, \mathit{\Omega}(\theta) \qquad (8)$$

Additionally we will use the following equations in order to exemplify the generalized cases:

$$L_{cross\ entropy} = -\sum y_i \ln(\,h(x_i)) \qquad (5’)$$

$$h_{linear\ regression}\big(x^{(j)}\big) = \sum_{i=1}^{n} x_i^{(j)}\theta_i = \theta^T x^{(j)} \qquad (4’)$$

$$X_{transformed} = \frac{X - \min X}{\max X} \qquad (9)$$

Where *X* in (2) is the training dataset with *M* samples and *N* features (predictors), *y* in equation (3) is the labels dataset with *M* samples corresponding to the *X* dataset, *h(X)* in equation 4) is the model defined hypothesis function that based on a input predictor values dataset and a given weight set θ from a weight space $\mathit{\Theta}$ computes a prediction $\hat{y}$ within the label space $\hat{y}\ \epsilon\ L^M$, $\mathit{\Lambda}(\mathit{\Theta})$ in (5) is a loss function defined by the model as in equation (5’) example of the Softmax cross-entropy loss function, $R(\theta)$ is a regularization function and finally $\mathit{\Omega}(\mathit{\Theta})$ is the actual objective function. As a result the goal of the learning process in any supervised learning model is the minimization of the objective function (or the maximization if $L_{model}(h, X, y)$ if is actually a gain function instead of a loss function) as defined by equation (8). Although in the case of deep models the complexity of the weight space surpasses by far the complexity of weight spaces within shallow models, the set of weights θ^* that are found to be minimizing the objective function contain easily and self-explanatory properties – such as in the case of linear regression (4’) where each weight θ_i directly defines the actual importance of the predictor feature $x_i^{(j)}$ given that the input data is normalized / standardized as in equation (9). Even more self-explanatory is the case of training decision trees models where the actual view of the trained model gives full and easy to understand information for any person no matter the background.

Up until now the shallow models, due to their nature have been totally inefficient in accomplishing complex tasks such as image classification training and prediction in real time for images where the size of the predictors and weight spaces is within millions or tens of millions of features. Such is the case of a predictive model that is supposed to analyze a screen image of a user interface and determine the controls, labels, buttons, etc. together with actual locations, sizes and any other inferable attributes. For the particular case of automated UX recognition the predictive model must analyze a stream of such images delivered (as a movie/stream for example) from an actual interaction between a user and the system –reading a stream of high-definition screens at over $3 * 10^6$predictor features per screen and at least 10 frames per second several hours and generating in real time outputs as described by ***Output** 1*.

Output 1 JSON/NoSQL data structure for pipeline output *(actual example in **Output** 2)*

```
Result = {
             [      <string element name> : {
                          "TYPE": <value>,
                          "LABEL": <value>,
                          "COORD": { <values> }
                          "SIZE": { <values> }
                          "ACTION": {<value> }
                           [ <string subelement name> { [<definition>] }
                                ....
             ]
}
```

Also, in terms of actual image scene inference we have to analyze from small areas such as 16x16x3 pixel matrices for simple one-glyph controls, to 64x16x3 pixel matrices or more for interface buttons, to even more complex user experience controls such as grids and tables, up to windows or workspace areas of the same size as the actual frame resolution.

Our proposed shallow machine learning pipeline architecture will be composed of: (a) *RESTSPC* layer – a execution model for scientific calculations based on GPU parallel computing as described by previous chapter in our paper (b) *RESTOLAS* layer - a shallow machine learning model based on a modified online Softmax regression with the hypothesis function defined in equation (10) that will use the *RESTSPC* layer for all calculations both for the online gradient descent based optimization and the predictions; (c) *RESTWIND* data pre-preprocessing and post-processing model that will handle the actual basic analysis of raw input information based on *RESTSPC*, feed the pre-processed data to the *RESTOLAS* and then prepare the pipeline output as described in ***Output** 1* and ***Output** 2*;

$$h_{softmax}\left(x_i^{(j)}\middle|\,\theta, j\ \in M, i\ \in N\right) = \frac{e^{\theta^T x_i^{(j)}}}{\sum_k^N e^{\theta^T x_k^{(j)}}} \qquad (10)$$

For the pre-processing layer *RESTWIND* of our pipeline we decided to take a simple sliding window algorithm and apply a modified approach for our parallel computing setting based on

RESTSPC layer. The original form of the sliding window algorithm presented by Algorithm 4 has been modified by the algorithm presented in Algorithm 5. The presented form of the Algorithm 5 is actually inspired by the convolution layers within the existing state-of-the-art CNN [1].

Algorithm 4 *SlidingWindows*(*ImageFrame*, *Model*)

```
    $Width \Leftarrow$ width of $ImageFrame$
    $Height \Leftarrow$ height of $ImageFrame$
    $ModelWidth \Leftarrow$ width of image to be analyzed $Model$
    $ModelHeight \Leftarrow$ height of image to be analysed by $Model$
    $StepSize \Leftarrow$ pixels skipped by the sliding window, proposed by the $Model$
    For each $Y$ in range 1 to $Height-ModelHeight$ with step $StepSize$
        For each $X$ in range 1 to $Width-ModelWidth$ with step $StepSize$
            $ModelImage \Leftarrow$ Extract subimage (X,$Y$,$X+ModelWidth$,$Y+ModelHeight$)
            Execute $Model.Train$ on $Model.Predict$ on $ModelImage$
Return
```

Algorithm 5 *SlidingWindows*(*ImageFrame*, *Model*, *RESTSPC*)

```
    $W \Leftarrow$ width of $ImageFrame$
    $H \Leftarrow$ height of $ImageFrame$
    $MW \Leftarrow$ width of image to be analyzed $Model$
    $MH \Leftarrow$ height of image to be analysed by $Model$
    $SS \Leftarrow$ pixels skipped by the sliding window, proposed by the $Model$
    $Tasks \Leftarrow ((W-MW) / SS) * ((H-MH) / SS)$
    $Allocation \Leftarrow$ Request resource allocation from $RESTSPC$ based on $Tasks$, $Model$
    Request $RESTSPC$ run in parallel total number $Tasks$ jobs with $Model$ and $ImageFrame$
Return
```

As mentioned, the main processing required by the *RESTOLAS* layer of our experimental pipeline is mainly composed of the optimization required by the cost function $L_{cross\ entropy}$ defined by equation (5'). The use of the Softmax approach ensures that for each scene element we will have a probabilistic inference – each scene element will receive a maximum likelihood probabilistic score together with a few other potential options (for example a "label" inferred with a probability of 60% could be also proposed as a "flattened button" with a probability of

35%). For the particular case of online learning we assume that each training case is fed to our machine learning model by the pre-processing layer and a simple gradient descent step is calculated based on the equation ...

Gradient descent for softmax regression..

Automatic UX programming based on image semantic segmentation

The main proposed innovation of our paper in the area of automated programming is the introduction of a machine learning pipeline that will be able to accomplish real time user interface analysis and reconstruction for the case of legacy system porting to Cloud computing environments or UX prediction for computer or hand drawn mockups.

The main reason behind the employment of a shallow machine learning model instead of a classic deep convolutional neural network for image recognition is based on the sparsity of training data for each individual user interface element at the moment of system experimentation. In order to train a deep convolutional network as described by [1] [3] [2] we would need either a pre-trained model or a massive training dataset. Due to the nature of our target scenes that we have to infer we will not be able to use pre-trained models on pictures/photographs. As a result a small variable dimension training set is provided as described by the sample data and figures below.

Nr	Nr Feats	Feat 1	Feat 2	...	Feat 500	Feat 501	...	Feat 9900	Feat 9901	...	Class
...	...	...	...	...	...	...	...	...	...	...	...
21	500	214	121	...	410	null	...	null	null	...	button
...	...	...	...	...	...	...	...	...	...	...	...
151	9900	0	250	...	17	17	...	0	null	...	label
...	...	...	...	...	...	...	...	...	...	...	...

Table 1 - Sample training data

Our machine learning pipeline will use several processing and will entirely rely on a REST model and GPU parallel computation engine. The simplest model proposed will be a augmented Softmax regression engine optimized for on-line learning and real-time prediction that will be presented in the following algorithm

Algorithm 6 GPUSoftmax

- *Short presentation of the goal and approach*

- *Why a pipeline and it is parallelized:*
    - *On-line real-time learning for model self-adaptation*
    - *Real-time predictions for REST service*
- *Presentation of pipeline model engine based on*
    - *Sliding window*
    - *Image recognition*
    - *UX View inference based on image recognition*
    - *UX Model inference based on UX view inference*
    - *UX Controller inference based on UX View/Model inference*

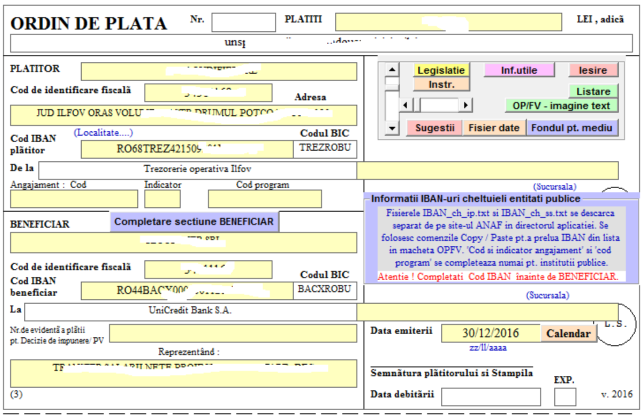
Automated application migration

Automated translation of legacy applications has always been an important research area for the computer science community. With each arrival of a new programming languages, frameworks or formal modelling languages computer scientist pursued the research and development of various code translators that would automatically migrate from legacy application source code to the new programming language. Most successful research and production projects over the years have been in the areas of automated code generation from formal modelling languages and in the area of automated code translation. Nevertheless the process of source code migration has been always accompanied by re-engineering processes that were supposed to optimize the internal mechanics of the newly migrated software systems. Yet another objective of the software translation has been and still is the migration of the legacy application from a deprecated environment to a more modern, robust and reliable one – such as migrating from desktop based applications to client-server and then to Cloud Computing environments. However on multiple occasions due to the missing of legacy source code the actual migration has been forced by employing terminals or virtual desktops.

In our paper we propose an innovative model architecture designed particularly for the challenges posed particularly by the user interface and data flow re-engineering of legacy applications where the source code is not available. Our model will employ machine learning techniques and in doing so will be able to “observe” and infer the functional behavior of the legacy application. At the end of the inferential process our pipeline model will be able to generate an intermediate scripted definition of the observed legacy application. Finally, the inferred and scripted definitions, defined by our own Intermediate Translation Definition Language (ITDL) script, will be used by the last layer of our pipeline model to generate target platform code.

From design mockup to functional UX software

A presentation of a real-life legacy application use case and the actual ITDL output of the model will better clarify the actual process described in Algorithm (1)-(5). For this purpose we will use a real life windows desktop application presented below in Figure (1). For obvious confidentiality reasons in Figure (1) the details/data within some of the fields has been obfuscated.



ORDIN DE PLATA Nr. PLATITI [redacted] LEI , adică

unsp [redacted]

PLATITOR [redacted]

Cod de identificare fiscalã [redacted]

Adresa

JUD ILFOV ORAS VOLU [redacted] DRUMUL POTCO [redacted]

(Localitate....)

Cod IBAN plãtitor RO68TREZ421509 [redacted]

Codul BIC TREZROBU

De la Trezorerie operativa Ilfov

(Sucursala)

Angajament : Cod Indicator Cod program

BENEFICIAR

[redacted]

Cod de identificare fiscalã [redacted]

Cod IBAN beneficiar RO44BAC [redacted]

Codul BIC BACXROBU

La UniCredit Bank S.A.

(Sucursala)

Nr.de evidentã a plãtii pt. Decizie de impunere/ PV

Reprezentând :

TF [redacted]

(3)

Informatii IBAN-uri cheltuieli entitati publice

Fisierele IBAN_ch_ip.txt si IBAN_ch_ss.txt se descarca separat de pe site-ul ANAF in directorul aplicatiei. Se folosesc comenzile Copy / Paste pt.a prelua IBAN din lista in macheta OPFV. 'Cod si indicator angajament' si 'cod program' se completeaza numai pt. institutii publice.

Atentie ! Completati Cod IBAN înainte de BENEFICIAR.

Data emiterii 30/12/2016 zz/ll/aaaa

L.S.

Semnãtura plãtitorului si Stampila

EXP.

Data debitãrii

v. 2016

Figure 1 Desktop legacy application

In the following output (2) we present the one of scene inferences generated by our pipeline model:

Output 1 *Use Case ITDL Output*

```
Result = {
"lblORDIN" : {
               "TYPE": {
                              "CAPTION" : 0.97
                              },
               "LABEL": "ORDIN DE PLATA",
               "COORD": { X: 5, Y: 5},
               "SIZE": {X: 150, Y: 20},
               "ACTION": "None",
               "BOUNDDATA": "None",
               "CHILDREN": "None"
                              },
"lblNr" : {
               "TYPE": {
                       "AUTOEDIT" : 0.55,
                       "EDIT": 0.41,
               },
               “LABEL”: “Nr.”,
               “COORD”: { X: 165, Y: 4},
               “SIZE”: {X: 40, Y: 20},
               “ACTION”: “None“,
               “BOUNDDATA”: “lblNr_Data”,
               “CHILDREN”: “None”
                              },
“lblPLATITI” : {
               “TYPE”: {
                       “EDIT”: 0.89
                       },
               “LABEL”: “PLATITI:”,
               “COORD”: { X: 215, Y: 4},
               “SIZE”: {X: 180, Y: 20},
               “ACTION”: “on_lblPLATITI_Change“
               “BOUNDDATA”: “None”,
               “CHILDREN”: “None”
                              },
“lblLEIadica” : {
               “TYPE”: {
                        “LABEL”:0.91
                       },
               “LABEL”: “LEI, adica”,
               “COORD”: { X: 395, Y: 5},
               “SIZE”: {X: 35, Y: 20},
               “ACTION”: “None“,
               “BOUNDDATA”: “None”,
               “CHILDREN”: “None”
                              },
“gidUnnameGridArea”:
{
               “TYPE”: {
                       “VIEWGRID”,:0.99
```

```
                      },
               “LABEL”: “None”,
               “COORD”: { X: 5, Y: 35},
               “SIZE”: {X: 450, Y: 350},
               “ACTION”: “None‟,
               “BOUNDDATA”: “None”,
               “CHILDREN”: {
        “lblPLATITOR” : {
               “TYPE”: {
                      “COMBOEDIT” : 0.59,
                      “AUTOEDIT”: 0.30,
                      “EDIT”: 0.10
                      },
               “LABEL”: “PLATITOR:”,
               “COORD”: { X: 3, Y: 3},
               “SIZE”: {X: 185, Y: 20},
               “ACTION”: “on_lblPLATITOR_Change‟
               “BOUNDDATA”: “lblPLATITOR_Data”,
                              },

. . .
}
```

As presented in *Output 1* our pipeline model generates a full scene inference for the desktop based application with the screen captured in Figure 2. Based on the actual user interaction with the legacy system the pipeline generates a maximum likelihood probability for each user interface control class. The inferred user interface control class is presented together with other highly likely potential candidates if such exists.

UX prototyping based on naive input

We will not continue with a short presentation of a potential experiment with naive hand-drawn user interface as presented in Figure 2 - Hand drawn interface mockup.

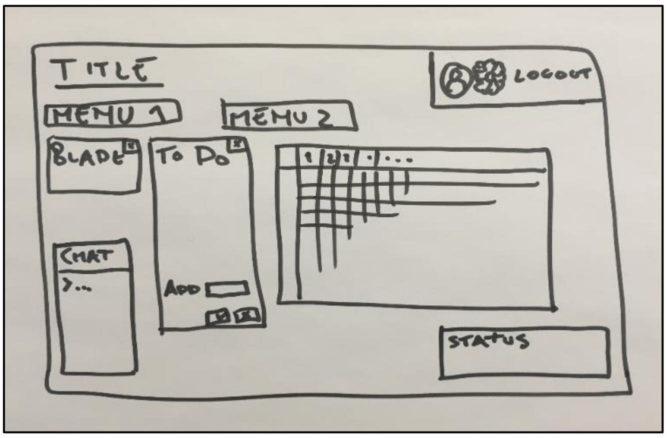


Figure 2 - Hand drawn interface mockup

Our model pipeline for automated UX engineering will be able to apply both semantic segmentation together with the basic visual recognition of given in order to output descriptive information that can easily be translated in a target programming language. Below is a fragment of the actual output of above naive hand drawn example:

Output 2 HandDrawnInterface

```
Result = {
             “lblTITLE” : {
                           “TYPE”: “CAPTION”,
                           “LABEL”: “TITLE”,
                           “COORD”: { X: 10, Y: 10}
                           “SIZE”: {X: 70, Y: 20}
                           “ACTION”: “None“
                           }
             “mnuMenu1”: {
                           “TYPE”: “MENU”,
                           “LABEL”: “MENU 1”,
                           “COORD”: { X: 10, Y: 50}
                           “SIZE”: {X: 75, Y: 15}
                           “ACTION”: “mnuMenu_Route1“
                           }
             “mnuMenu2”: {
                           “TYPE”: “MENU”,
                           “LABEL”: “MENU 2”,
                           “COORD”: { X: 110, Y: 50}
                           “SIZE”: {X: 75, Y: 15}
                           “ACTION”: “mnuMenu2_Route1“
                           }
             “frmTodo”: {
                           “TYPE”: {“FRAME”: 0.53, “BLADE”:0.47}
                           “LABEL”: “To Do”,
                           “COORD”: { X: 40, Y: 50}
                           “SIZE”: {X: 60, Y: 100}
                           “ACTION”: “stdActionOkCancel“
                           “CONTROLS”:{
                                  “edAdd” : {
                                         “TYPE”: “EDIT”,
                                         “LABEL”: “ADD”,
                                         “COORD”: { X: 5, Y: 80}
                                         “SIZE”: {X: 50, Y: 15}
                                         “ACTION”: “edValidate“
                                         “MODEL” : {

                                     .....
```

References

[1] A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet Classification with Deep Convolutional Networks," *Advances in neural information processing systems 1097-1105,* 2012.

[2] L. e. a. Szegedy, "Going Deeper with Convolutions," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) ,* 2015.

[3] J. Long, E. Shelhamer and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *IEEE Pattern Analysis and Machine Intelligence ISSN: 0162-8828,* no. May 2016, 2016.

[4] ImageNet, "ImageNet Large Scale Visual Recognition Competition (ILSVRC)," in *www.image-net.org/challenges/LSVRC/.*

[5] A. Agrawal, J. Gans and A. Goldfarb, "Managing the Machines," *HBR,* 2016.

[6] Vinyals, "A Neural Conversational Model," in *ICML Deep Learning Workshop 2015,* 2015.

[7] C. V, Q.-B. Nguyen and S. Pankanti, "Deep Learning Ensembles for Melanoma Recognition in Dermoscopy Images," *Journal of Research and Development,* 2016.

[8] J. Ghorpade, J. Parande and M. Kulkarni, "GPGPU PROCESSING IN CUDA ARCHITECTURE," *Advanced Computing: An International Journal (ACIJ),* vol. 3, 2012.

[9] M. Bihis and S. Roychowdhury, "A generalized flow for multi-class and binary classification tasks: An Azure ML approach," in *Big Data (Big Data), 2015 IEEE International Conference on,* 2015.

[10] J. Stone, D. Gohara and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science & Engineering*, vol. 12, no. 3, 2010.

[11] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California, Irvine, 2000.