UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS, COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

Artificial scene inference based on efficient parallel execution of directed acyclic tensor graphs

Andrei Ionut DAMIAN (PhD Candidate)

Nicolae Tapus (PhD Supervisor)

2019

Contents

1 Thesis abstract and objectives *(completed 90%)*

1.1 The problem of artificial scene and video stream inference

1.1.1 The intuition and the real-life problem

The initial idea behind this work has been rooted in the real-life need to convert legacy applications from old execution environments - such as legacy desktop database systems or client-server frameworks - to Cloud-based infrastructures. This particular objective has multiple roots both in terms of resource management and also from technical perspective. From the resource management motivation perspective, we can enumerate both the costs of contracting and managing the human-based team required for migration software development as well as the opportunity costs related to the time involved in this particular process *(Figure 3 depicts the classical approach for migration and associated direct costs)*. The technical aspects are mostly related to the risks associated with potential buggy modules, configuring and deployment issues and so on.

Nevertheless, following this initial real-life need identification, several other connected use-cases have been identified such as the need to employ intelligent agents that can automate user-interface human-computer interaction (Robotic Process Automation) or the use-case example of transforming a simple (even paper hand-drawn) application user interface mock-up into a functional, designed and scripted user-interface form or screen. To be even more precise, for the user-interface process automation, our target has been that of replacing the need of scripted behavior with intelligent agents. As an example, we can take the case of a scripted behavior such as the following one, given in natural language for simplicity of explanation: *"move mouse to this absolute screen rectangle area and perform click operation"*. Starting from this concrete example, our target is that of employing intelligent agents that would recognize in the above example if the user-form has been moved to another place on the interface screen or that the layout of the input areas has been revamped. Furthermore, we present a list of realistic use-cases based on actual analysis of user needs related to various scenarios:

a) Enrichment and enhancement of simple User-Experience tasks automation or even more complex ones currently provided by RPA systems based on heuristics:

arguably the first clear target area of application that is currently lacking this kind of technological advancement to the best of our knowledge is that or Robotic Process Automation (RPA). For this particular case we aim am proposing our research and experimental technology as a logical next step in the enhancement of visual interface automation agents that currently work based on visual rules and heuristics. For a clearer understanding **Error! Reference source not found.** and **Error! Reference so urce not found.** explain the real-life discovered pain-point – while in the first figure the RPA agent is able to complete the assigned task in the second image it clearly fails due to a minor change in the visual UI environment. Only recently (2019) companies in this area advertised new approaches [1] based on Computer Vision for the task of User Experience/Interface components understanding. Nevertheless, this area is still in research and experimentation phase at the moment this report is written.

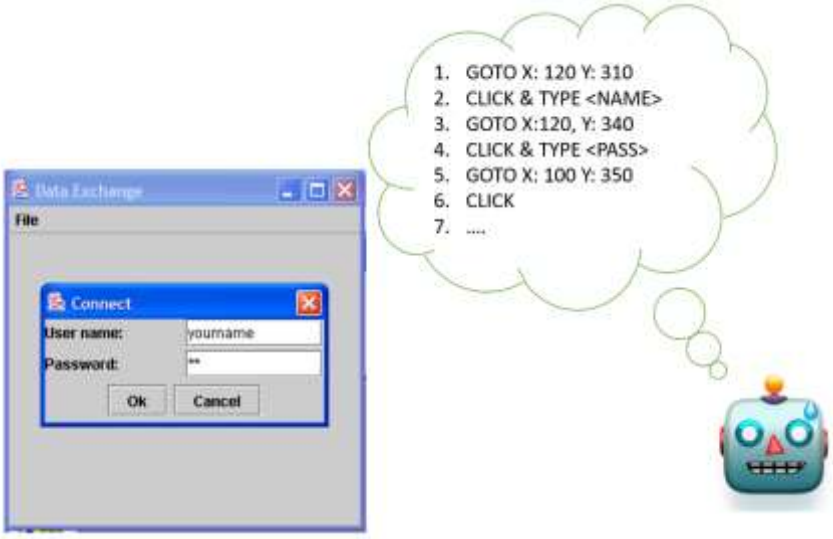


Figure 1 - RPA agent pre-programmed based on rules for a specific task

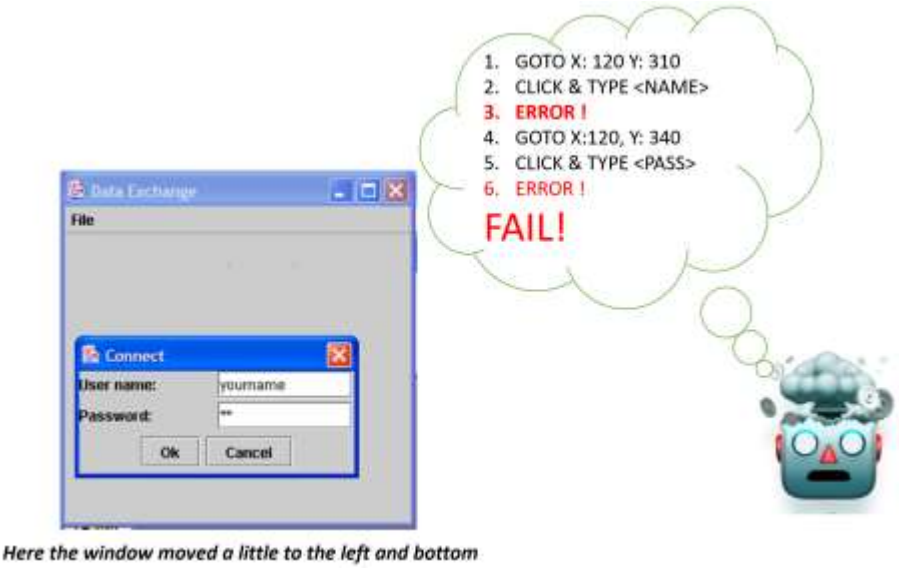


Figure 2 - RPA agent can't complete task as the environment has changed its parameters and the heuristics do not apply anymore

b) Migration, translation, and automated maintenance of legacy software systems - ranging from the need to translate a legacy system developed for a particular target operating system desktop graphical environment up to the need to quickly and on-the-spot understand user interaction and behavior within complex legacy system.

c) Fast prototyping in unknown or new user experience and user interface programming languages has been constantly an important requirement in front-end development. Although this particular task is closely related to the previous translation and migration one it is nonetheless a different scenario where the source application is already available in source code (front end source code) together with the rendered user-interface output.

d) Last but not least one of the real-life use cases relates to the advanced understanding and automation of electronic reports, forms and tables. In various both horizontal such as accounting or logistics and vertical domains such as financial sector there is an increasing need for more efficient and streamlined operations through intelligent process automation. Organizations are using either legacy systems or wide-spread systems in order to produce data in various formats and stages of a particular process. In all these scenarios there is a increasing need for advanced post-processing of the

given data without explicit data export and migration – either there is no clear and reliable way of data exporting or the process is to complex for the users that operate the system. For a clearer explanation we will quickly analyze two particular cases:

i. automated data gathering and processing based on online web-forms. This first case is a straight-forward process of user-interface analysis, inference and translation that, in this scenario, will generate a *script code* output that can range from simple JSON [2] to more complex HTML5 (https://html.spec.whatwg.org/multipage/) and up to PHP (https://www.php.net/docs.php) or other web-application script languages.

ii. digitization and structured data pre-processing of pre-printed / scanned tables and forms such as monthly financial reports or other similar unstructured data. In this particular scenario the objective is to generate a cross-platform digital representation such as a comma-separated values of a printed document such as a printed spreadsheet that is not available in the digital editable and version-able form.

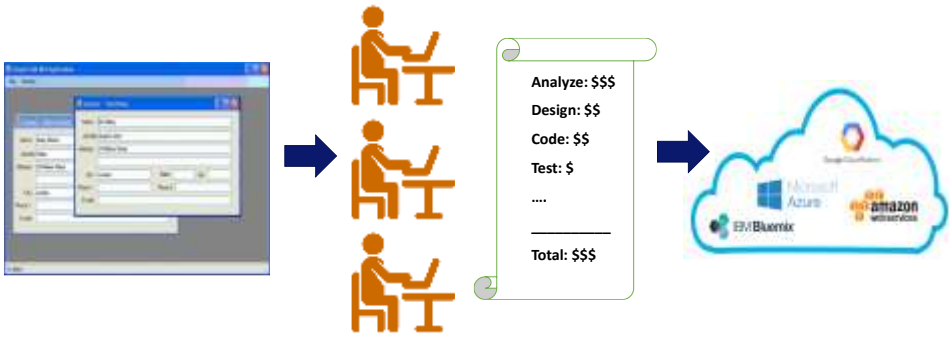


Figure 3 - The classical approach to system migration - the software development team runs a whole development cycle in order to deploy to the new target (maybe) Cloud Computing based environment

1.1.2 The overall view

The Artificial Intelligence horizontal achieved great advances [3] in latest years mainly due to the Deep Learning continuous state-of-the-art improvements and also due to the widely adoption within AI research and development community of GPU-based parallel computing [4] and the proliferation of public dataset libraries [5]. Within the multitude of existing and potential research domains of AI we have to mention the important area of systems development and maintenance automation, still considered by most a "holy-grail" area of research.

Our vision, further presented within this thesis, was to research and develop truly intelligent systems able to analyze user experience video streams from various sources and finally infer real and usable analysis including actual code-level details of those observed interfaces such as the simple example depicted in *Figure 4*. One key element of such systems is that of artificial user-interface scene inference and analysis based on deep learning computer vision systems. During a period of over 2 years we have researched and developed various experiments [6] [7] [8] that will also be referenced within the thesis with particular emphasis on the research and experiments described in the paper "*Deep Vision Models for Artificial Image Processing*" [8]. Another focus of the past research period has been to analyze and compare our research and experimental work with other similar research and other existing initiatives in this particular field [9].

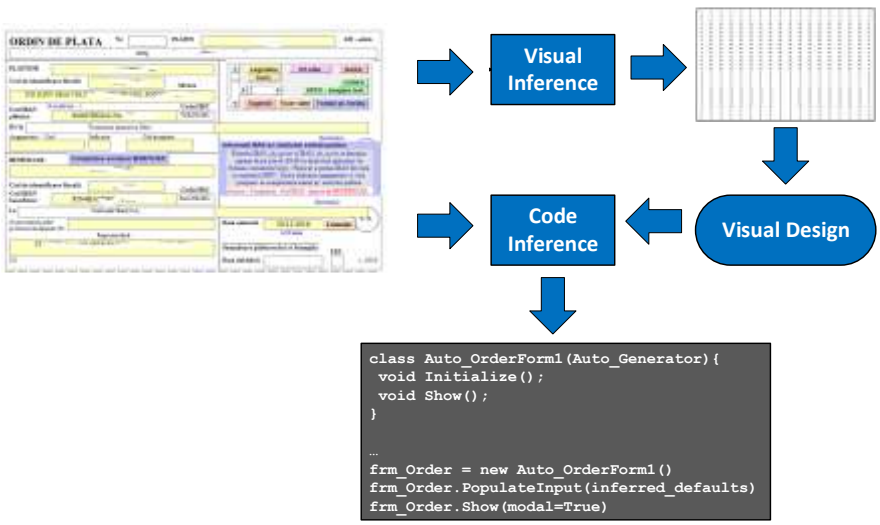


Figure 4 - From a single-form legacy desktop application to a visual design script (such as HTML, JSON,etc) up to the actual source code in a target programming language

As argued in our published work, computer vision models and particularly deep directed acyclic graphs based on convolutional modules are generally constructed and trained based on natural images datasets. Due to this fact, the convolution directed acyclic graph (DAG) models will develop during the training process natural image feature detectors with the exception of the base graph modules that will learn basic primitive visual features. As a result, one of the focus areas of our research and experimentation has been to develop DAG models specialized for synthetic image recognition and train those models on our own experimental datasets. The proposed end-to-end trained models are finally able to infer user-experience functional details of the proposed input synthetic images that can be automatically translated to target operational source-code such as HTML/JavaScript.

In our thesis we will present the current state-of-the-art in two different, yet connected, areas: that of deep learning models for computer vision and the area of GPU-based parallel computation of DAGs for efficient training and production-grade operationalization. Further this we will present the architecture of our whole end-to-end experiment including our early work [6] based on shallow model architectures and the latter Deep Learning based models [8]. A special section will be dedicated to the research and development of our artificial images dataset that we will publish

in Open Source format in order to further benefit the international research community. Following the architecture section, we will continue with the experimentation details and actual details of an online production-grade system.

Finally, we will conclude with a revision of the most important proposed contributions and the actual conclusions of the thesis that include potential multiple real-life applications – from stand-alone implementation of our models up to incorporating the CloudifierNet architectures and pipeline in external RPA (Robotic Process Automation) applications. One important observation is that in our research and experimentation cycles we designed and implemented two different versions of the CloudifierNet namely (v1 referred as *CloudifierNetV1*, v2 referred as *CloudifierNetV2*) and for each version we implemented several sub-version tuning the size of the graph and its computation requirements.

1.2 Experimental work vs. real-life application: challenges & expectations

Our final experimental results have been operationalized within a working system prototype that has been deployed live. Currently a team of data scientists and engineers are working on improving the performance of the model pipeline and the fast execution environment.

During our work on this research & development project we have encountered several issues that clearly separate our experimental work results from real-life wide application. Analyzing these individual challenges is important both from the perspective of setting the right expectations of the research ambitions as well as previewing the further work that can be done in order to improve the current results:

a) *The limited domain of application determined by the generation/production of the synthetic training dataset*: The research and experimentation task of generating the proposed Artificial Dataset (AD) – namely the images of user interface controls and full user interface screen captures – cannot, by any means, capture all the potential user interfaces variations of any previously or currently available user experience standard or

approach. In our thesis, as well as in the published papers, we emphasize the actual selection of several user interface standard such as legacy Microsoft Windows applications based on MFC, Delphi or other similar development environments. Nevertheless, a universal dataset and thus a potentially universally applicable model pipeline is beyond the scope of our work.

b) *Impractical application of the experimental project results to systems and applications with non-traditional user experience approaches (user interfaces that do not follow classic visual and functional rules):* As a complementary issue to the one previously presented we also have the one regarding the impossibility of our neural model pipeline to "understand" user-interfaces that do not follow common approaches in terms of user-experience flow.

c) *Potential need to limit the target development environment to web-based application that do not require complex client-side functionalities*: Finally, following numerous experiments and real use-cases analysis we have concluded that from the multitude of potential target environments (such as mobile-android, mobile-iOS, web, MS-windows, unity, etc.) we will focus on web applications with the classic model-view-controller approach limiting the usage of complex client-side libraries (such as jQuery) to simple and easily deployable features (such as user interaction augmentation features).

2 Related work and current state-of-the-art *(completed 80%)*

2.1 State-of-the art in Deep Learning for Computer Vision

The area of Computer Vision has known a tremendous evolution following the historical success of the AlexNet [10] developed by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton from University of Toronto. We can argue that 2012 is the particular moment in time when Deep Learning has emerged as the main direction of research and development in the area of Artificial Intelligence including, but not limited to, Deep Learning for Computer Vision. Since then, a multitude of research teams both from academic environment – mostly notable University of Toronto, Stanford University, University of Oxford – and commercial environment – Google AI research group, Microsoft Research – have been competing and continuously advancing the state-of-the-art in the area of Computer Vision.

The proposed architecture presented within our thesis relates to the most influential deep convolutional directed acyclic graphs architectures – namely the Inception [11] and ResNet [12] as well as several other architectures such as separable convolution network proposed by Xception [13], and also the fully convolutional model for end-to-end image segmentation FCN [14] based on the straight sequential VGGNet [15] deep neural network. The current state-of-the-art in Deep Learning for Computer Vision can thus be summarized by the following main elements that constitute the basis of current successful architectures:

- the residual/skip connections initially proposed by He et al [12] that allow deeper neural graphs without fearing the main concern of inefficient gradient propagation during the back-propagation based training.
- the network-in-network [11] parallel convolutional columns that allow efficient processing on GPU architectures of varied feature detectors (convolutional kernels)
- the introduction of the fully convolutional architecture with its multiple dense prediction maps [14] that allows for pixel-wise inference

- due to the average size of the proposed dataset we use strong regularization applied at multiple stages. The regularization is based on the classical dropout technique by Hinton et al [16]

2.1.1 Residual learning using skip-like connections

The main idea behind the skip connections in deep convolutional networks is based on the fact that in very deep visual models are hard to train mainly due to the inefficient propagation of the loss function gradients in lower layers. A similar idea has been proposed in the successful, and still state-of-the-art, architecture or recurrent neural networks with skip connection namely Long Short-Term Memory cells (LSTM) by Hochreiter et al [17] as early in 1997. Basically, the residual (or skip) connection is a shortcut that takes the input in a certain convolutional graph-block, no matter the content and complexity of the given graph-block and adds this input to the output of the block before the final block non-linearity. We can formalize this layer in below equation (1) where the *NL* function denotes a non-linearity (such as the *ReLU* [18] function in equation (2)), *x* denotes the input of the residual block, W_r represents the weights of the residual block *F* and finally W_t represents the transformation tensor that allows the input volume x to have the same size as the output of the *F* function.

$$ResOut = NL(F(x|\, W_r) + W_t x) \quad (1)$$

$$NL(x) = max(0, x) \quad (2)$$

Please note that in the initial paper proposed by He et al [12] the function F has been proposed as a traditional block of convolutions and nonlinearities however later work and current state-of-the-art employs multi-column convolutional graph such as the *Inception* [11] module that takes full advantage of the modern parallel computation approaches based on GPU optimized numerical computing. Such an architecture is proposed in *Figure 5* and it will be further detailed in the later sections.

An important note is that in equation (1) we can replace the addition of the input x with a concatenation operation.

Another recent state-of-the-art research that we relate to is that of Sandler et al from Google AI Research team [19]. In this recent work they propose a few new tricks for the optimization of inference and optimization of the deep visual models deployed in mobile apps with limited computation capabilities. Beside the known approaches that we will further present in our related work, the MobileNet V2 architecture does not employ any non-linearity (such as the one in equation (2)) after the addition of the input to the pre-output of the residual block (see Figure 5). This approach introduces an important decrease in the evaluation and optimization speed without hindering the overall performance of the directed convolutional acyclic graph.

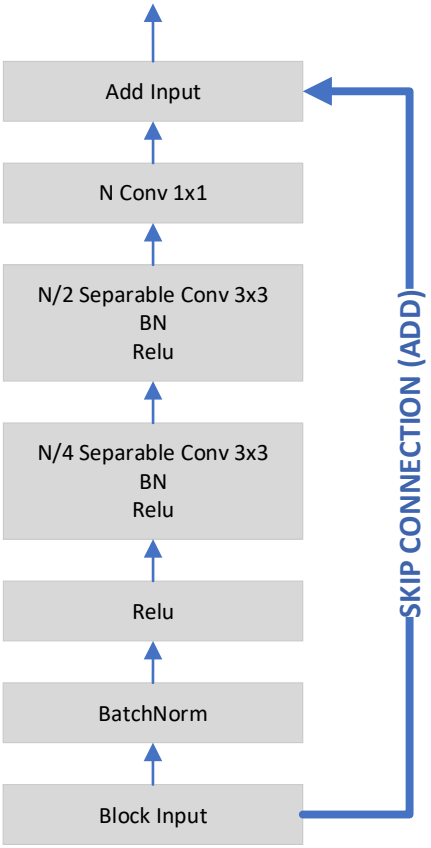


Figure 5 – Residual skip connection in a CNN

Finally, we have to mention the recent work of Zagoruyko et al [20] that proves the increase in computation efficiency both for training and inference due to reducing the number of layers and the increase of convolutional filters. As a result we use dramatically increased residual volumes

compared with the classic residual [21] or network-in-network-residual [22] approaches that will be further presented in the section 2.1.2.

2.1.2 Networks-in-networks and separable convolutions

The concept of network-in-network, or the so-called Inception module presented in Figure 6, was initially proposed by Lin et al [23] and later by Szegedy et al [11]. The purpose of this architecture is to allow the deep learning directed acyclic graph to learn/construct multiple receptive fields for the same analyzed local patch. In classic convolutional neural network architecture, the convolutional layers learn receptive fields that initially target small patches and then gradually target larger patches learning higher and more abstract concepts – nevertheless, all of this in a linear approach. The network-in-network tries to address this linearity by allowing the DAG to learn various types of receptive fields for the same level of input size granularity or more specifically for the same analyzed local input patch. Important note is that this method takes full advantage of parallel processing infrastructures due to the fact that each "column" in the network-in-network is dependent only of the module input and not of the other columns.

It is important to note that the first major graph architecture based on this approach – the *Inception v1* network – has managed to set the new state of the art for classification and detection in the *ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14)*.

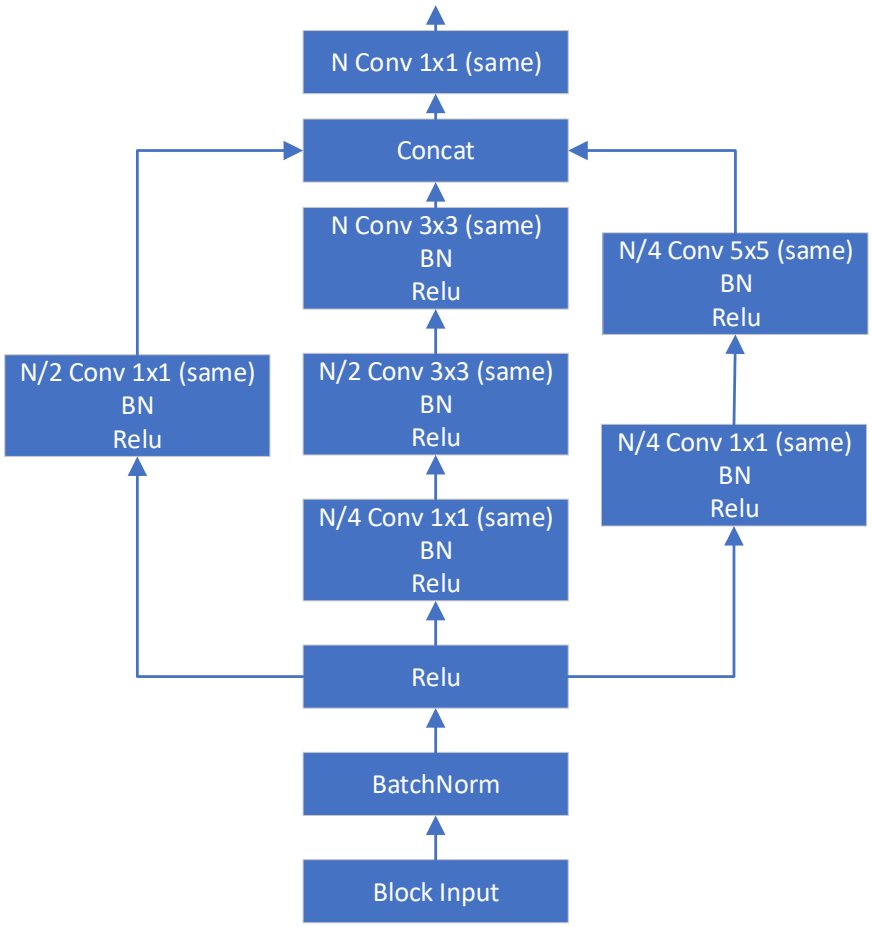


Figure 6 – Inception module example

A more recent approach to the objective of learning multiple receptive fields for the same target local patch is based on proposed Xception architecture by Chollet [13] where the ideea is to drop the variable size convolutional kernels in favor of more multiple convolutional kernels of the same size all in the same efficient module. Basically, this architecture relies on a depth-wise (spatial) convolution operation performed independently over each channel of an input, followed by a 1x1 kernel convolution on the volume generated by the depth-wise convolution and finally resulting into a new volume with a new channel space.

In terms of actual computations performed by a depth-wise separable convolution vs a classic convolution we have the following simple mathematical explanation:

a) For the case of depth-wise separable convolution we apply a bidimensional filter $1 \times k \times k$ *(multiplied* d_i *times)* to each channel of the input volume $h \times w \times d_i$ resulting in a output volume $h_d \times w_d \times d_i$ where h_d and w_d are directly dependent of the stride and padding of the convolution beside the actual size k of the kernel. The number of multiplications required for this operation is $h_d \times w_d \times k \times k \times d_i$ and in the particular case of a input volume of $8 \times 8 \times 3$ and a square kernel with size 3, no padding and stride 1, we have a output volume for the depth-wise convolution of $6 \times 6 \times 3$ based on $6 \times 6 \times 3 \times 3 \times 3 = 972$ multiplications. Following the depth-wise convolution we apply a point convolution (1×1) that has the purpose of linearly re-combining the initial depth-wise features into the final number of filters. For this final step of the depth-wise convolution we apply the point kernel with a normal convolution operation on the $h_d \times w_d \times d_i$ volume resulting in a volume of $h_d \times w_d \times d_o$ where d_o is the number of point kernels and thus the number of output filters. The computation cost of this operation is $h_d \times w_d \times 1 \times 1 \times d_o$. For our example let us say we want to obtain 32 output filters so the cost of the second operation will be $6 \times 6 \times 1 \times 1 \times 32 = 1{,}152$ multiplications summing the entire depth-wise separable convolution operation to 2,124 multiplication operations. In terms of graph size we have to store two sets of weights (without taking into consideration any other hyperparameters or biases) that of the depth-wise convolution (depth-wise kernel $3 \times 3 \times 3 = 27$ in our example) and the pointwise convolution ($3 \times 1 \times 1 \times 32 = 96$ in our example) summing a total of 123 weights.

b) Without reviewing the classic discrete convolution mathematical formulation, we ca directly compute the needed computation. In the case of the classic convolution where we apply a 3×3 kernel ($3 \times 3 \times 3$ *input channels)* with the purpose of obtaining 32 feature maps we would need to apply scalar products on each sub-volume of the full $8 \times 8 \times 3$ input volume. The actual number of operations is $6 \times 6 \times 3 \times 3 \times 3 \times 32 = 31{,}104$ multiplications – that is more 14 times bigger than the computational cost of the depth-wise separable convolution. In terms of graph size for a classic discrete convolution we have to store (without taking into consideration any other

hyperparameters) a total number of $d_i \times k \times k \times d_o$ weight-parameters – in our particular example $3 \times 3 \times 3 \times 32 = 864$ weights.

Finally, we might see this method as a simplified approach to initial Inception module architecture. This single-layer approach has yielded important performance results surpassing more advanced versions of the initial *Inception v1* architecture – such as *Inception v4* [22] - and the more advanced versions of the previously mentioned *ResNet* [21] architecture.

2.1.3 Fully convolutional architectures

Classic computational graph architectures in the field of deep learning based computer vision are mainly constructed based on either the objective of object classification or object detection/localization. This approach generates a graph architecture (presented in Figure 7) where the whole model is divided into two main sections: convolutional modules - with or without network-in-network, skips, etc. – followed by a fully connected module that takes as input the flattening of the final convolutional output volume.

This approach imposes strict restriction to the input volume size of the model and this usually demands for special tricks that replace the flattening with max-pooling or average-pooling operation performed over each channel of the final convolutional output volume.

As a result, a new approach emerged – the Fully Convolutional Neural Networks, or FCN, approach and was widely disseminated based on the works of Long et al [14]. The main intuition is simple and based on the main assumption that we can get rid of the final pre-readout block by using convolutional modules instead of the flatten and fully connected layers.

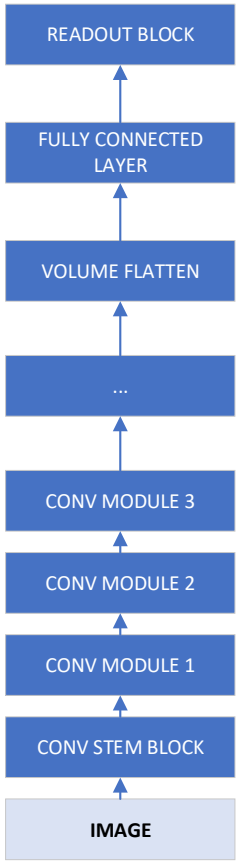


Figure 7 – Classic CNN architecture with stem, convolutional modules and fully connected modules at the top

This strategy is can be approached from multiple angles:

a) replacing the layer flattening with a pooling layer that will apply its operation over entire feature maps of the last convolutional output volume – as presented in below equation (3) where *D* is the depth of the output *HWD* volume generated by the function represented by the previous convolutional modules applied on the input *X* image.

$$FCN_{pre-readout} = \underset{D}{Pool}\left(f_{c_{HWD}}(X)\middle|X, H, W, D\right) \quad (3)$$

This way we will obtain one activation per each feature map of the final conv output volume and thus make the whole architecture totally independent of the input image height and width. This

particular approach is usually useful in tasks such as image classification or image embedding generation where no local-patch information is required.

b) For the particular goal of obtaining local patch information we have to use a different strategy that will give us the option of having per-patch or per-pixel inference capabilities. Instead of flattening or the previous mentioned strategy of introducing a "global" pooling operation we will use transposed convolution operation (or sometimes called deconvolution operation) that is basically a fractionally-strided convolutional operation that upscales the input volume to a desired output size as presented in equation (4) where *s* is the stride, *p* is the padding, *h* is the input volume height (assuming *h*=*w*).

$$O_{deconv} = s(h_{in} - 1) + s + k - 2p \tag{4}$$

Using this approach, we can generate an output volume of the same height and width size as the input image with a desired number of channels. In the case of per-pixel classification for the task of image semantic segmentation we will use the same number of channels as the potential classes we have for each pixel. As a result, the output - *readout* volume in this case – contains a "fiber" of information for each pixel of the input volume that can used in a usual Softmax function as described in equation (5)

$$h_{softmax}\left(x_i^{(j)} \middle| \theta, j \in M, i \in N\right) = \frac{e^{\theta^T x_i^{(j)}}}{\sum_k^N e^{\theta^T x_k^{(j)}}} \tag{5}$$

Finally, this model can be trained as usual in classification tasks with a negative log-likelihood loss function as described in equation (6)

$$\underset{\theta}{argmin} -\frac{1}{N*W*H}\sum_{i=1}^{N}\sum_{x=1,y=1}^{W,H} log\, p_\theta(\hat{y}_{H,W} = y_{H,W}|X,Y) \quad (6)$$

We can consider the equation (6) optimization objective as a *pixel-wise categorical cross-entropy*. To further explain the above gradient based optimization objective, we can refer to one of the most well-known image recognition datasets that includes the segmentation tasks – the COCO dataset, based on the work of Lin et al [24], currently copyright (c) 2015, COCO Consortium and available at http://cocodataset.org. A few of the images used in the COCO dataset and their segmentation annotations is presented in Figure 8.

Figure 8 – Segmentation examples from the COCO dataset http://cocodataset.org

As depicted by Figure 8 and based on the presented architecture and optimization technique we are able to obtain an actual semantic segmentation of the objects and subjects that are presented in the input image by classifying each and every pixel.

2.1.4 Loss behavior in dense pixel prediction

One of the main issues in the settings where we aim to generate dense predictions on a certain input image such as in our case is the overwhelming loss signal generated by the easy to predict classes such as background. The semantic segmentation aims to generate dense pixel classification for target objects in images, as discussed in chapter 2.1.3, however we still have to take into account other classes such as the environment or background. The actual optimization process based on the loss function described by equation (6) will have to factor every pixel of the input image into account not just the zones-of-interest. Recently Lin et all from *Facebook AI Research* described this pitfall of training dense predictive directed acyclical graphs based on back propagation in the paper their paper proposing a custom modification [25] of the classical cross-entropy loss. The proposed modification of the cross-entropy loss, namely the introduction of the Focal loss is depicted by equations (7) and . The main intuition of the approach proposed by the authors is that for the particular cases when the loss is very small for well classified examples the γ hyperparameter will allow us to further decrease the loss signal for those inferences and thus minimize its contribution in the cases where a batch or a particular observation is overwhelmed with such signals. The secondary hyperparameter α is nothing more than a weighting factor that allows us to control the potential class imbalance.

$$FL(p_t) = -\alpha_t(1-p_t)^{\gamma} * \log(p_t) \tag{7}$$

$$p_t = \begin{cases} p \; if \;\; y = 1 \\ 1-p \; if \; y = 0 \end{cases} \tag{8}$$

Further information regarding this subject is tackled in the experiment analysis and directed acyclical graph training procedure described in chapter 3.6.1 where we can observe the actual dynamic of the proposed loss function for different values of the γ hyperparameter including the case where we set γ=0 for standard cross-entropy loss or weighted cross-entropy if γ=0 and α<>1.

2.2 Optimizing computational graphs on massive computing architectures

One of the most recent work from Google Research/Brain team leverages GPU-based computation architectures to a new level: employing massive parallel computing infrastructures in order to determine the most efficient graph architecture focusing on all perspective of computation efficiency. In the work of Tan et al [26] is emphasized the need to construct and deploy computational graphs specialized on various scenarios based on available memory, target images resolution and last but not least actual numerical computation capacity. Although the initial 2019 paper "*EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*" the research team focused entirely on backbone graph architectures with the target of multi-nominal classification and underlying feature extraction a second paper published by the research team at the end of 2019 and revised in 2020 – "*EfficientDet: Scalable and Efficient Object Detection*" also by Tan et al [27]. This follow-up paper focuses on the more advanced problems of detection and segmentation albeit by employing the same range of backbone graphs as in the previous work and proving that the proposed approaches for graph efficiency optimization deliver beyond state-of-the-art results.

2.2.1 Nuts and bolts of *Efficient* graphs

In terms of proposed architecture the Tan et al Google team rely on previous proposed convolutional graph architectures and propose a *MBConv* module – inverted residual graph module using depth-wise convolutions as presented in Chapter 2.1.2 with the important addition of the *self-gating* mechanism "*Squeeze and excite*" proposed by Hu et al [28]. While we will not analyze

the overall information flow (forward and back propagation) within the proposed module as the residual and skip-like connections are presented in *Chapter 2.1.1* we will analyze the intuition behind the depth-wise convolution and *squeeze-and-excite* pairing.

The main intuition of this approach is that a depth-wise convolution by itself, although extremely efficient in terms of computation, cannot capture all the important information in the input volume particularly due to the way the computation is done: each individual channel of the input volume is individually analyzed by a dedicated bidimensional kernel resulting in a volume that has the same number of independently processed filter-maps that do not have inter-channel combined features. As presented previously in Chapter 2.1.2 for the case of depth-wise separable convolutions we further apply a "*combine*" operation based on a discrete convolution using a *1x1* kernel (actually a *1 x 1 x f* tensor where *f* is the number of channels of the volume that is processed by the initial depth-wise operation). The authors of the *EfficientNet* architecture argue that by replacing the discrete *1x1* discrete convolution operation with an operation that will recalibrate the information in each individual channel of the volume resulted from the depth-wise convolution a greater degree of performance is obtained. To be more precise the proposed feature-map-wise processing is actually interposed between the depth-wise convolution and the classic *1x1* discrete convolution feature re-combination. This information recalibration is basically a self-gating mechanism performed by multiplying an individual scalar activation – obtained based on the depth-wise convolution output passed through a small sub-graph with learned weights - with each individual channel as depicted in the block diagram from Figure 9 proposed in the original paper by Hu et al [28].

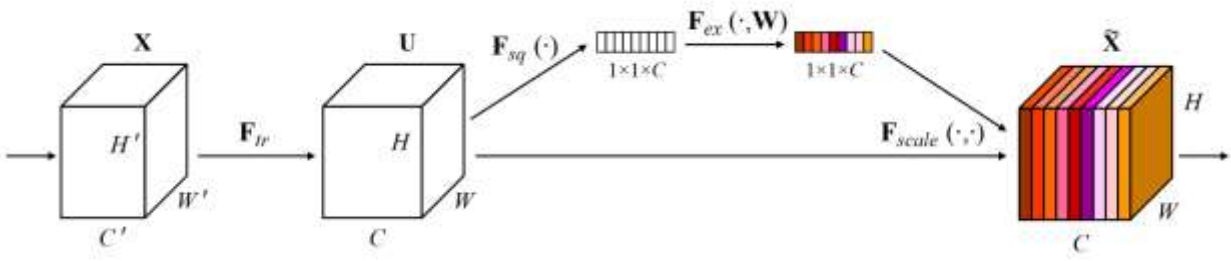


Figure 9 - Squeeze-and-excite block as original presented in the paper "Squeeze-and-Excitation Networks" by Hu et al

Finally, following the recalibration of each individual depth-wise convolved filter-map a 1x1 discrete convolution is applied in order to finally re-combine all the feature maps simillary with the approach of the separable depth-wise convolution presented in Chapter 2.1.2. In terms of non-linear activation function the authors of the *EfficientNet* architecture employ *Swish* activation function (Ramachandran et al 2017 [29]) that scales the input with its sigmoid activation $f(x) = x * sigmoid(x)$.

2.2.2 Performance through AutoML massive parallel grid search

Probably the most important finding of this related work is fact that the authors proposed a new scalable approach to graph architectures or simply put instead of using fixed architectures with fixed input size as all of the past state-of-the-art architectures they propose various (namely 8 architecture *B0-B7*) for various graph input volumes scaling the depth (the total number of *convolutional*-like operations) as well as the width (the number of feature maps generated by each *MBConv* module). The authors argue that all past approaches focused on specific computational graph architecture improvement: either increasing the number of the modules output feature maps, adding more modules (layers) to the graph or increase the graph standard input image resolution (and adapting the graph in the process).

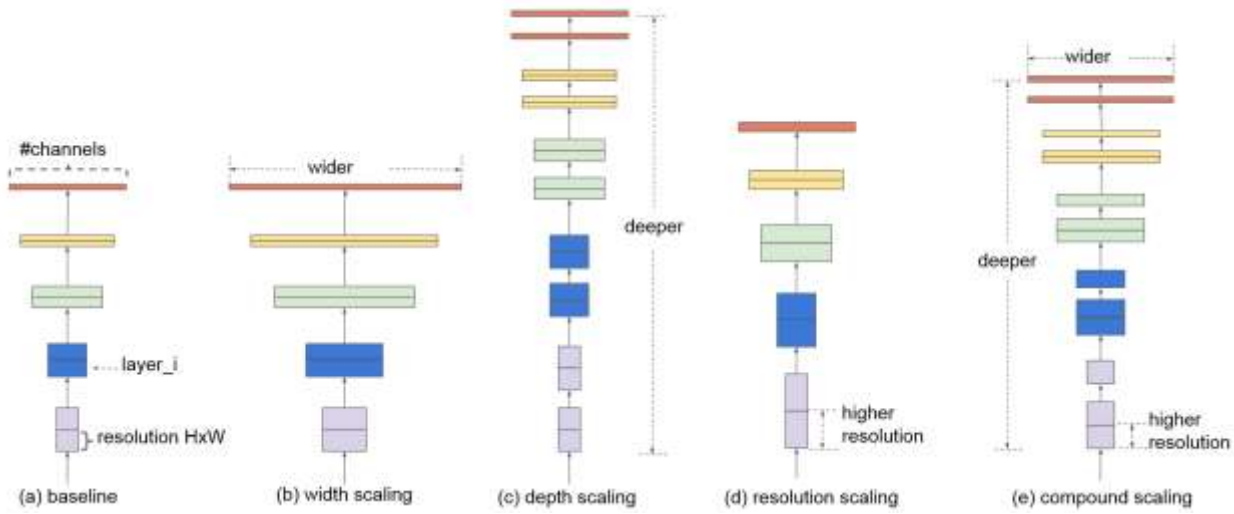


Figure 10 - Analysis of the varios model scaling approaches including the proposed compound scaling. Figure from the original "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networ" paper by Tan et al.

As presented in the original paper – *Figure 10* - the authors propose a compound graph scaling architecture that will gradually increase input resolution together with the number of nodes in the graph and the number of feature-maps generated by each *MBConv* module. The Google research team settled for 8 different *standard* architectures that trade gradually computation efficiency for feature richness and thus end-result quality as presented by the original paper overall graph performance/efficiency comparison (Figure 11). What is probably more important than the actual comparison between number of parameters and performance is the comparison not depicted in Figure 11 regarding the FLOPS of individual models – as a single example the proposed B3 architecture requires approximatively **18x fewer FLOPS** than the ResNeXt-101 architecture for a forward pass at marginally similar feature extraction capacity and accuracy performance. As previously mentioned, the actual scaling approach and proposed architecture have been obtained using *Google AutoML* model architecture grid-search engine.

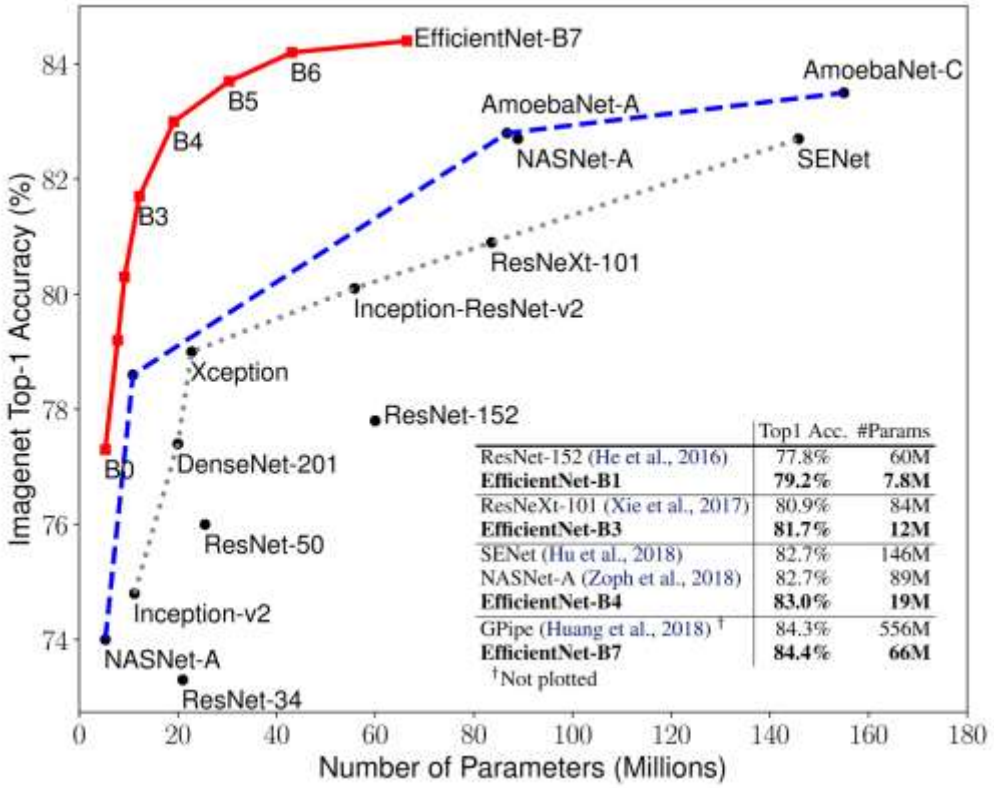


Figure 11 - Various EfficientNet instances compared by Tan et al in the paper "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". The comparison also includes various previous state-of-the-art architectures

2.3 State-of-the art in GPU based scientific computation

In this project we are strongly relying on the current state-of-the-art in GPU based numerical massive parallel computation. One of the most computationally parallelizable operation in DAGs in general is that of the "discrete convolution" and the whole family of the "*convolutional*" graph modules. Within this family of operations we assume that the input of the graph module is a tensor that has a symbolic "*depth*" dimension – be it the feature size for *uni-dimensional* data streams such as time-series, the color channels for image processing or a actual set of images with their respective channels for video processing. The intuition is that by applying a convolutional kernel (that can be a R^2,R^3,R^4 tensor for the mentioned cases) we can compute in parallel various

representations (the so called *filter maps* in deep computer vision) of the module input data and thus constructing with each additional convolutional module higher level of features for our initial input information.

Nowadays, GPU devices are no longer used only for their initial purpose of powering gaming engines and 3D multimedia applications. Actually, one of the most important key factors that enabled the last years fast development of Artificial Intelligence and Deep Learning in particular is the mainstream availability of strong parallel numerical computing through the usage of graphical card multi-core processors. A simple chart showing ca comparison of floating-point operations per second for the CPU and GPU (curtesy of Nvidia CUDA Development Toolkit documentation https://developer.nvidia.com/cuda-toolkit) is presented in Figure 12.

As an observation regarding current state of research and development for GPU computing required by Deep Learning in particular, at the moment of this writing, we have to mention that out of the two major GPU technology providers, that is Nvidia and AMD, the first one provides a clear advantage versus the competition both in terms of support as well in terms of performance in specific numerical computation tasks.

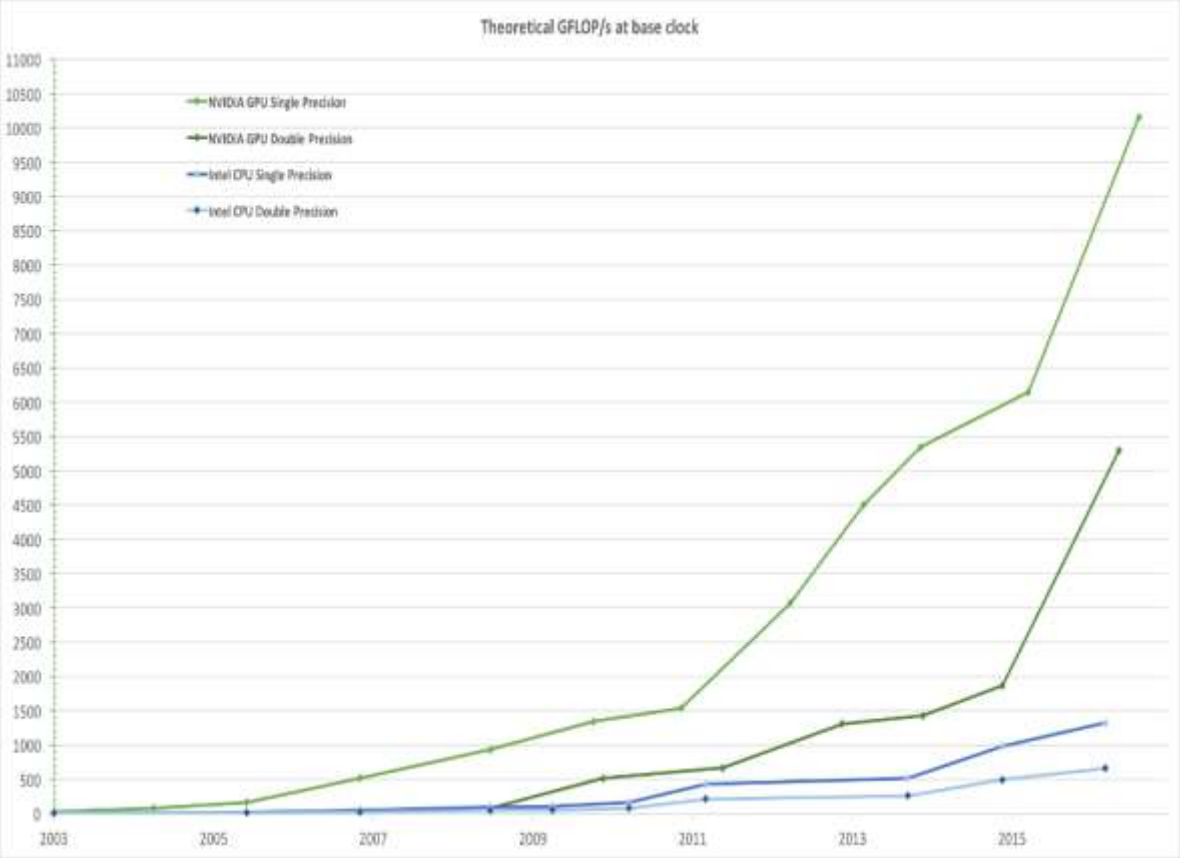


Figure 12 – Comparison between CPU and GPU processing

The current project research and experimental development is entirely based on the latest 2017-2018 state-of-the-art Nvidia Pascal and Volta technology that offer multi-core and fast GPU device memory presented in various recent papers [26] [27] [28]. Although it is beyond of the scope of this thesis, we will mention that Nvidia CUDA technology, similarly with other architectures, allows the developer to define, offload and execute highly parallel numerical computation code directly on the GPU device. In our initial stages of the current work we also explored the possibility of using OpenCL [29] as a platform for developing GPU kernels for parallel numeric computing, however we dropped the advantage of having cross-platform deployment (AMD GPUs for example) by employing OpenCL to that of higher speed and better support offered by Nvidia CUDA technology.

A short example is presented in Figure 13 where we have the CUDA C kernel code that performs the pair-wise addition of two M-dimensional vectors using a core for each particular dimension.

```
// CUDA Kernel definition
__global__ void _VecAddKernel(float* S1, float* S2, float* D)
{
    int i = threadIdx.x; // use thread ID to index the vectors
    D[i] = S1[i] + S2[i];
}

int main()
{
    M = sizeof(A)
    // above CUDA Kernel invocation with M threads
    _VecAddKernel <<<1, M>>> (A, B, C);
    ...
}
```

Figure 13 – CUDA Kernel example

The presented code is not necessarily a highly efficient parallelization of the given task, being more an example to better understand the capabilities of modern GPU infrastructure when it comes to numerical parallel computing. In modern tensor graph optimization and execution settings it is common to employ efficient and highly-optimized tensor computation engines such as Theano [30] or TensorFlow [31]. These frameworks offer high-level tensor manipulation methods that "hide" the actual GPU kernel preparation and execution tasks from the programmer.

For our whole experimentation process, we decided to use TensorFlow [31] [32], probably the most advanced and widely used tensor-graph computation engine, both in academia and in commercial environment. Besides many other engineering-related reasons, TensorFlow was chosen due to its ability to handle and scale very well GPU based parallel numerical computations. In particular, TensorFlow is able to offer both in-GPU parallel execution of multiple sub-graph operations and also multi-GPU parallel execution of one or multiple computational graphs. Finally, TensorFlow is able to deploy graph inference or optimization jobs on multiple computational nodes that are either able to use CPU resources or both CPU and GPU resources.

2.4 Image captioning and sequence decoding

For the particular feature of our architecture – that of generating human & machine readable source code such as JSON, HTML or other UX-definition script – we have to analyze the current state-of-the-art in the area of image captioning and sequence decoding both for the areas of caption generation [33] as well as classic neural language generation approaches [34].

Current state-of-the art already includes similar recent work mentioned previously [9] that employs encoder-decoder DAG architecture *(Figure 14*) generically based on the work of Karpathy et al [33] and other similar papers in the area of image caption/description generation such as [35] [36] [37].

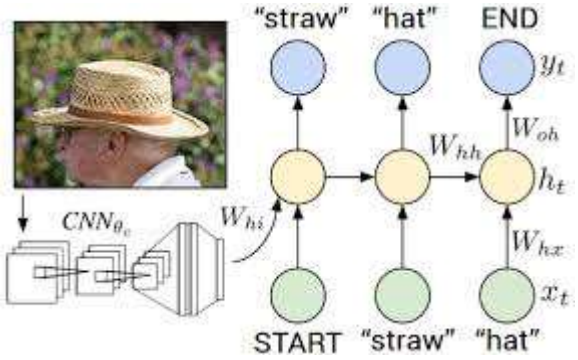


Figure 14 - From image to text - from "Deep Visual-Semantic Alignments for Generating Image Descriptions" by Karpathy et al

2.4.1 Attention based encoder-decoders

The general approach involves a two-step process that starts with the *encoding*, using a graph based on convolutional modules, of the graphical context *state* of the image and then use a RNN, such as LSTM or GRU, in order to *decode* the target source code using classic auto-regressive approach. The DAG is trained end-to-end with the classic encoder-decoder [38] approach using both the image and the target source code. In Figure 15 we can view the actual end-to-end model

for encoding-decoding with post recurrent cell attention mechanisms in the decoder module as described in [39].

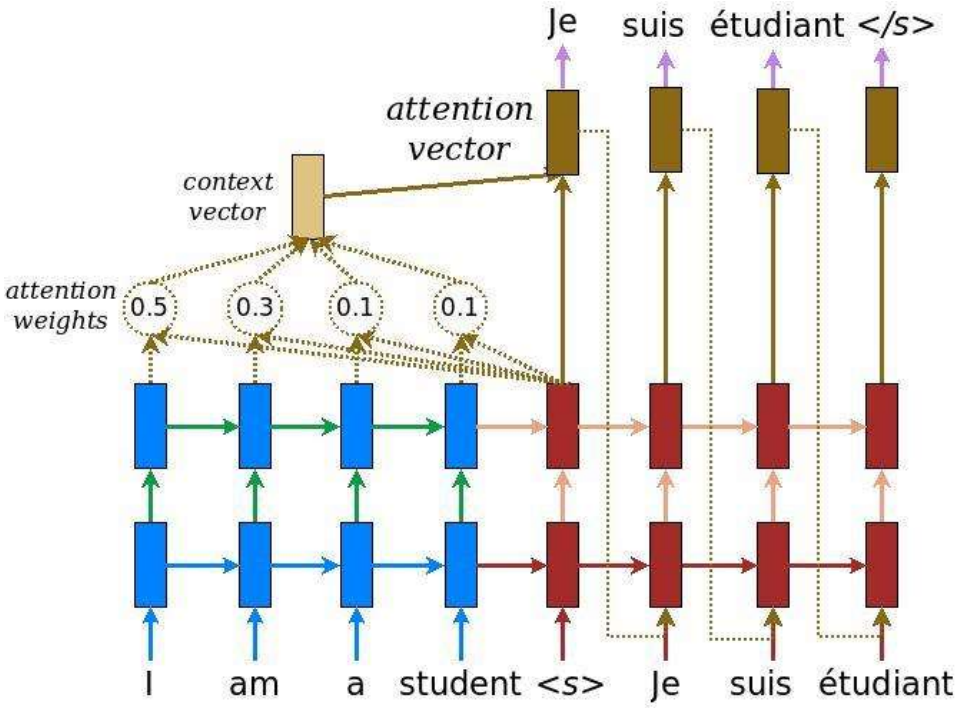


Figure 15 - Seq2Seq with attention mechanisms for end-to-end model encoding-decoding as described by Luong et al

The overall approach used in Neural Machine Translation is based on the verified hypothesis that the attention mechanisms allows the decoded to efficiently reuse parts of the encoded information – in the particular case of NMT the actual encoded source text while in our case the fine-grained *softmax* map generated by *CloudifierNet.* The *attention* taxonomy can be analyzed from multiple perspectives: general mechanism ranging from classic recurrent attention up to non-recurrent self-attention mechanisms, location and computation approach. In our case we will tackle only the recurrent-based attention where we can have either the attention module before the recurrent cell or multi-cell (such as LSTM or GRU) or we can have the attention module after the recurrent layers. The final perspective of the attention mechanisms taxonomy is related to the calculation approach of the attention distribution over the input (values or the encoded signal). Below is a formalization of the various methods for computing attention mechanisms for the particular case of applying attention in a recurrent decoder.

$$\alpha_{ts} = \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \quad \text{[Attention weights]} \quad (1)$$

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s \quad \text{[Context vector]} \quad (2)$$

$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W_c}[\boldsymbol{c}_t; \boldsymbol{h}_t]) \quad \text{[Attention vector]} \quad (3)$$

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \boldsymbol{W} \bar{\boldsymbol{h}}_s & \text{[Luong's multiplicative style]} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_1} \boldsymbol{h}_t + \boldsymbol{W_2} \bar{\boldsymbol{h}}_s\right) & \text{[Bahdanau's additive style]} \end{cases} \quad (4)$$

Figure 16 - Formalization attention mechanisms in recurrent decoder graphs

Quickly summarized, in Figure 16 we have a scoring function *score(h_t,h_s)* (4) that basically computes the un-normalized logits of the attention based on the encoder full signal h_s and the current state h_t at timestep *t* of the decoder graph. The final *softmax* normalized attention (1) is applied over the encoded signal h_s in order to select the most important parts of that signal and thus obtain the *context* vector c_t of the current state of the decoding process (2). This *context* is finally combined with the needed features of the current decoding stage (such as the state of the decoding graph and/or the inputs, previous outputs, etc). In our particular case the h_s encoder state is, as previously mentioned, the actual readout of the *CloudifierNet* models. Our approach and design details for this *code* decoder sub-graph module can be found in section 3.5.3.

2.5 Our published scientific and experimental work

During the project research and development, we have approached our objectives from various angles, did several iterations of research and experimentation and disseminated the results within research papers [6] [7] [8]. In this work we have tackled several main issues, ranging from the evaluation of the hypothesis that our proposed final goal of artificial UX scene inference could be performed by shallow machine learning models optimized for highly parallel numerical computing environments, to preparation of our training data due to the fact that little to none

research dataset exists. Finally, our research and development arrived at the point where we created a highly-optimized state-of-the-art architecture for our proposed task based on Deep Learning approach. Summarized our activities and disseminated results can be reviewed in the following points:

1. We have tested the hypothesis of applying shallow model ensembles developed and optimized for GPU execution for the problem of artificial scene inference in order to generate an intermediate (ITDL) JSON output and finally generate the user-experience real-application source code [6]

2. We have experimentally developed an initial basic of single-label images containing user interface basic controls [6] obtaining a starting point for our training data

3. We have developed highly-parallelizable version of the proposed shallow models using low-level tensor graph evaluation and optimization engine [7] obtaining improvement over existing CPU-based libraries

4. We have developed an online experimental job execution environment for GPU-optimized graph-based machine learning shallow models [7] that we will further develop for production grade projects

5. We have prepared a single-label and multi-label dataset of artificial user-interface image observation containing both information for classification and segmentation tasks [8]

6. Finally, we have researched and developed advanced deep directed acyclical graphs for end-to-end analysis and inference of visual artificial scenes [8] obtaining state-of-the-art results for our task of identifying user-interface control, primitives and overall layout of the inferred scene. One of the final model architectures generated by our research and experimentation and recently published [8] can be analyzed in

Figure 17. Further information regarding this particular architecture details and other more advanced ones will be presented in section 3 of the current thesis.

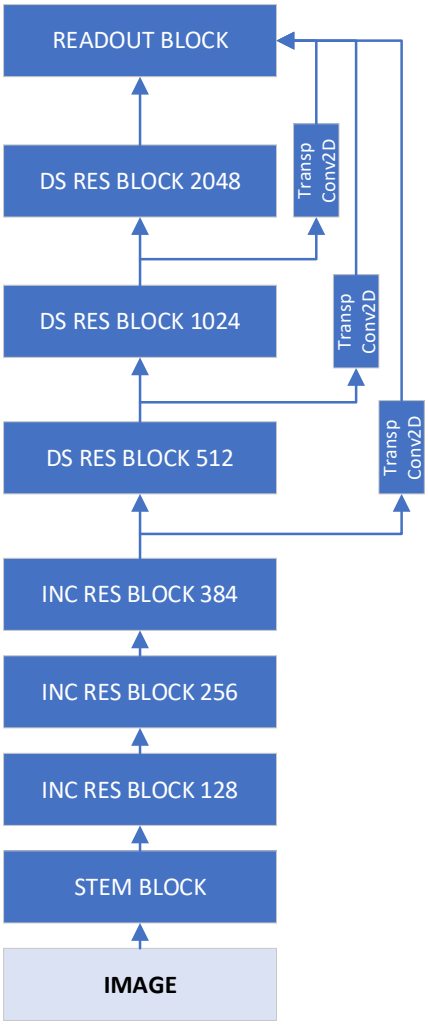


Figure 17 – CloudifierNetV1 DAG architecture as proposed by [8]

3 The architecture *(completed 80%)*

In the following chapter we will present the architectures of all our experiments, ranging from the preparation of the first experimental dataset and the initial approach of employing shallow models on highly parallelized execution environments, up to the final proposed architecture. All the mentioned stages of research and development will be approached from a comparison perspective based on the challenges we encountered and the architectural decisions we had to make along the way.

3.1 Parallelized shallow architecture vs Deep Learning

Within the project research and development lifecycle one of the decisions we had to make was related to the employment of simple shallow machine learning model ensembles versus the careful design and implementation of complex deep directed acyclical graphs based on convolutions. Our initial research and experimentation led to a potential hypothesis stating that an ensemble of shallow weak models, albeit using models that have a good potential for efficient numerical optimization in parallel GPU-based environments, could address with near state-of-the-art efficiency and accuracy our specific problem of artificial scene inference. After thoroughly testing this hypothesis and preparing in the process the research paper "*Model Architecture for Automatic Translation and Migration of Legacy Applications to Cloud Computing Environments*" [6] we obtained a list of results and conclusions. One the important findings was that our goal of proposing highly GPU-optimized versions of the selected shallow models has been achieved. This result has been reached based on the fact that we designed our proposed models as highly numerical-efficient and well-tuned tensorial graphs architectures that can be executed in a low-level tensor graph running environment – such as the one used by use, namely TensorFlow. Nevertheless, we arrived at the conclusion that the near-real-time achieved speed and resource allocation efficiency provided by employing this method has the drawback of providing below acceptance-threshold performance in terms of accuracy, recall and precision. Although we did employ ensemble methods such as boosting [40], the final results in terms of accuracy have been unsatisfactory. Another identified minor drawback of this architecture was that each of the

ensembles or models responsible for recognizing a certain user-interface visual element had to be used with a sliding-windows approach. Certainly, the sliding windows algorithm have been particularly designed for GPU-based parallel numerical computing however the complexity of the process and thus the memory and processing requirements increased linearly with the complexity of the proposed artificial visual scene due to various reasons such as the space variance dependability of the shallow models.

We concluded that the most logical hypothesis we have to pursue is the one that will test the employment of space-invariant deep directed acyclical graphs. This approach, due to the nature of that could also have the capacity of performing the proposed task in an end-to-end manner. To this end, the initial tests of this hypothesis have been based on simple deep dense-classification sequential directed acyclical graphs that performed well in terms of achieving a proof-of-concept result for the objective of end-to-end artificial scene inference. Following this initial experiment, we then conducted a series of experiments based on various state-of-the-art architectures in order to obtain our Deep Learning based architectures such as the one depicted in *Figure 17*, namely the *CloudifierNetV1* model architecture. As with the particular case of the previously mentioned shallow models, this final graph architecture has been designed with the particular goal of maximizing the parallel numerical computing capabilities of GPU-based infrastructures.

This whole approach has been adopted in order to perform an exhaustive research and experimentation that will analyze both the possibility of employing simple self-explainable models such as shallow SoftMax classifiers and also the option of using state-of-the-art and beyond deep directed acyclical graph architectures. Finally, we opted of the deep graph architecture based on all the previously presented performance evidence. Further information regarding model comparison is presented in section 4.3.

3.2 A new approach to graph module architecture – Multi Gated Units

The most important result of our graph architecture research consists in the proposal of a novel architecture form convolution module encapsulation that eliminates the need for extensive grid-search hyper-parameter tuning. We argue that our proposed novel approach not only

drastically decreases the need for grid-search of graph hyper-parameters but also allows each module in our graph to have a specific unique configuration. Although the proposed Multi Gated Unit (*MGU*) can be used with any kind of graph module (linear, convolutional 2D, convolutional 1D, recurrent modules, etc) we used it in our experiments applied to 2D convolutional modules.

The main intuition behind the proposed *MGU* architecture is that we can use gating mechanisms similar to those of [41] and [42] in order to allow our graph to learn what kind of feature processing at the level of each individual module is required to reach the optimization goal.

3.2.1 The problem

In order to formalize our approach, we will describe the *MGU* applied on fully connected (FC) graph modules and then expand this formalization to convolutional modules. As a starting point let us assume we have a FC activated by a *ReLU* function (Rectified Linear Unit [18]) defined by the equations (9), (10) and (11). Now, we have several options in potentially improving the learning process and generating better features out of *f(x)* function-module. One of the most obvious options and usual improvements is that of employing batch-normalization – setting mean to 0 and standard deviation to 1 for a batch of processed features - based on the work of Ioffe et al [43]. We note the batch-oriented normalization function with *BN(x)* and without going into the details of the actual normalization process we have to mention that we have to options: that of applying batch-normalization on the linear transformation based on *l(x)* as described in equation (12) and thus obtaining a new version of the initial proposed *f(x)* function, noted as f_{bb} or apply the *BN(x)* function after the non-linearity function *σ(x)* as denoted by equation (13) - f_{ba}. We can also replace the batch-oriented normalization of features with other learned-normalization techniques such as Layer Normalization [44] but for now let us assume we have a directed acyclical graph that has multiple modules where we could use one of the three different options denoted by equations (9), (12) and (13). Considering these options as actual hyper-parameters of the proposed graph we should then proceed with grid-searching the optimal configuration for each module.

$$f(x) = \sigma\big(l(x)\big) \tag{9}$$

$$l(x) = W * x + b; \tag{10}$$

$$\sigma(x) = \max(0, x) \tag{11}$$

$$f_{bb} = \sigma\big(BN(l(x))\big) \tag{12}$$

$$f_{ba} = BN\big(\sigma(l(x))\big) \tag{13}$$

At this point we can easily observe that the search space grows exponentially with the number of modules (we need to find the optimal setting for each module) as well as with the number of options for each module. Although there are options for efficient search in hyper-parameters space not method can ensure that the actual best/stable configuration is found – other than the exhaustive search.

3.2.2 The solution - MGUs

The main intuition of our proposed solution to the previous described problem of hyperparameter-search is to give the graph the power to learn the best configuration for each module. We argue that this can be accomplished with the introduction of multiple gating mechanisms that allow the module to find its own best configuration. Basically, for each individual processing step of the module we add a gate-activation module composed by a linear transformation and a squash-function (such as *sigmoid*) that will constrain the gate-activation module output values between *0* and *1*. Let us see how the previously described equations are modified in this new setting. First of all, the basic *f(x)*, *σ(x)* and *l(x)* as well as the batch-norming functions f_{bb} and f_{ba} remain unchanged, however we add the new gate-activation functions as well as the actual gating mechanisms.

$$g_f(x) = sigmoid\left(l_f(x)\right) \tag{14}$$

$$g_{bba}(x) = sigmoid\big(l_{bba}(x)\big) \tag{15}$$

$$g_{fb}(x) = sigmoid\left(l_{fb}(x)\right) \tag{16}$$

$$f_{bba} = g_{bba}(x) * f_{bb} + \left(1 - g_{bba}(x)\right) * f_{ba} \tag{17}$$

$$f_{fb} = g_{fb}(x) * f_{bba} + \left(1 - g_{fb}(x)\right) * f(x) \tag{18}$$

$$f_f = g_f(x) * f_{fb} + \left(1 - g_f(x)\right) * l_s(x) \tag{19}$$

Each of the proposed gate-activation functions has the purpose of choosing between one feature processing method or another. For example, the gate g_{fb} uses the linear transformation function l_{bba} followed by a *sigmoid* and enables the module to choose between using the straight *f(x)* function or the results of f_{bba} – that in turn choses either f_{bb} or f_{ba} path. The final function f_f is actually *the highway-network* proposed by Srivastava et al [42] where the gate-activation function g_f allows the module to choose between passing forward the input features using a linear transformation l_s function or processing them with the linearity followed by the non-linearity activation. As a side note for this final gate, we can use $l_s = x$ if the dimensionality of the input features is the same as that of the output features. A descriptive picture of the above proposed sub-graph can be observed in *Figure 18*.

Intuitively the process of learning all the needed gates within the *MultiGatedUnit* can truly be seen as an advanced hyperparameter grid search where the instead of learning discrete options – {0, 1} values of the gates. The *MGU* learns continuous weights that find the optimal hyperparameter configuration based on the graph input data and can potentially adapt based on this input data to complex configurations where gates are opened or closed to certain degrees for specific features within the input feature space. To this end the *MGU* offers the possibility analyzing the learned gate pathways: using simple heuristics such as determining the degree of near-zero coefficients for a certain gate we can infer which gates will be opened with certainty as well as which gates are mostly dependent of the input feature vectors.

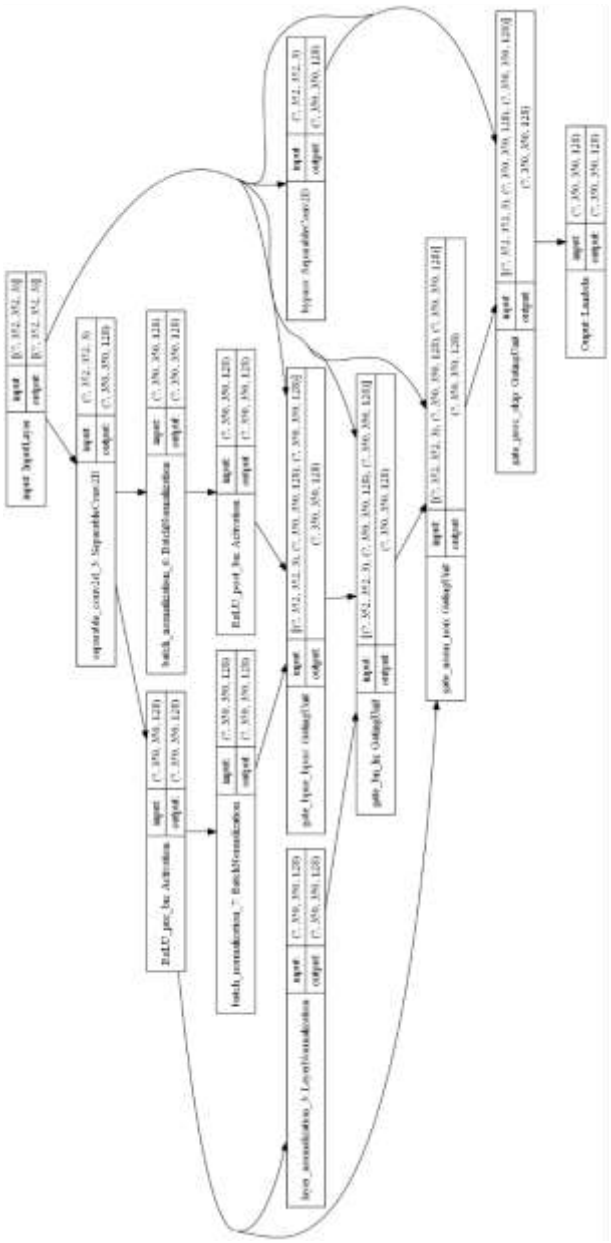


Figure 18 - The MGU sub-graph modules that is able to learn the optimal configuration of post & pre-processing of features

Our proposed implementation of the *MultiGateUnit* that is publicly available on GitHub at https://github.com/andreiionutdamian/phd/tree/master/model contains a basic implementation of the self-diagnosis that offers a high degree of self explain-ability. Basically by "expanding" a series of MGU internal gating units we can discover the actual path of information flow that resulted

from the process of graph optimization. This optimal path of feature information flow in the graph can certainly show us what nodes can pe pruned from the computational graph and thus greatly help in generating a custom optimized version of the graph. In this optimized version that does not use the MGU but rather bits and pieces of the MGU components we can find out that no two blocks have similar structure (as it happens in the classic architectures where blocks are repeated at various stages of the graph and each block in a repetitive zone has identical or almost identical hyperparameters)

3.3 Pretrained vs cold-started models

Computer Vision is undoubtedly one of the first machine learning and Deep Learning in particular fields where Transfer Learning, initially proposed by Pratt et al [45] in the paper "*Discriminability-based transfer between neural networks*", has been employed with great results. However, only later with the publication of ImageNet [46] by Deng et al in 2009, the research domain of Transfer Learning began to reach its true potential of providing a feasible way of storing gained training knowledge and applying this knowledge to a different, yet related, problem.

One of our research and experimentation challenges has been to accurately evaluate the possibility of applying transfer learning to our models due to the nature of existing pre-trained state-of-the-art modules. Although we are employing various subgraph architectures based on current state-of-the-art in Deep Vision, all of them are pre-trained on natural image datasets such as *ImageNet* dataset. As a result, we have concentrated our efforts to find a way of injecting prior knowledge into our models by exclusively focusing on lower convolutional modules. Thus, we have pre-trained our stem-block, that will be further described in the section related to the detail-oriented presentation of the final architecture. These initial graph serial convolutional blocks have been injected with basic knowledge of identifying simple visual primitives such as dots and lines of various shapes and sizes.

Finally, as we have expected, we have experimentally discovered that cold-training the models, without any prior pre-training natural-imaging knowledge injection, greatly increases convergence time as opposed to applying transfer learning to stem module. In our experimentation

we have approached this subject both by allowing the gradients to further propagate in the stem module and also blocking the training of these particular convolutional modules.

3.4 Natural vs artificial datasets

3.4.1 Natural vs artificial datasets

As mentioned in the previous section, one of the challenges that we have to face has been related to the nature of our target observation space – that of artificial computer graphical screen images. Although Deep Computer Vision is one of the most prolific and active research areas, all of the resulting state-of-the-art deep vision directed acyclical graphs are exclusively trained on natural images provided by various dataset resulted from commercial and academic environment [24] [10] [47] [48] [49] [50]. During our research and experimentation, we decided to design our architecture and plan our experiments in order to address both observations spaces "realms" – both natural scene and artificial ones. For this particular challenge we decided to use the same graph architectures albeit using different graph optimization process for each of the two types of observation space. Certainly, for the particular task of natural scene analysis we have made heavily use of transfer learning, especially in the lower modules of the computational graph, whereas for the artificial scene inference we had to research, experiment and generate a brand-new dataset.

One particular, yet important, task of our research and experimentation has been the one related to the inference of artificial scenes based on natural images. The whole idea was based on the following real-life use-case: a user needs to transform a hand-drawn user-interface mockup into a real user-interface in order to minimize the experimentation time. This particular feature will enable non-programmers, non-designers and other users with no user-interface design computer skills to design and experiment with full user interfaces that can be deployed in online scenarios or in small desktop applications. The real-life application is currently in the stage of Proof of Concept (POC) development with further roadmap to a Minimum-Viable-Product preparation for a commercial grade system deployment.

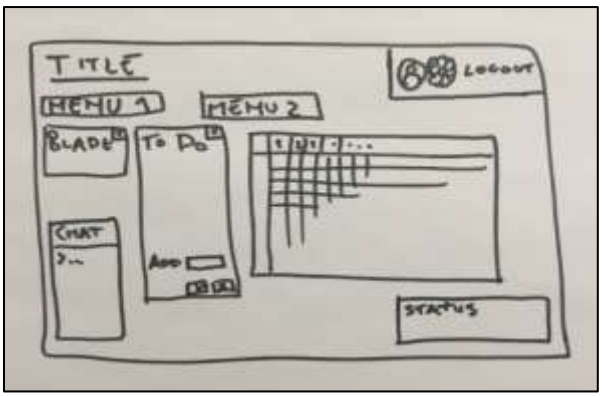


Figure 19 – Naïve user interface hand drawing example

Starting from the fact that a hand-drawn mockup can range from naïve (such as the one presented in Figure 19) up to an elaborate hand-drawn sketch based on doodling techniques such as the hand-drawn controls prepared for our project and exemplified in Figure 20.

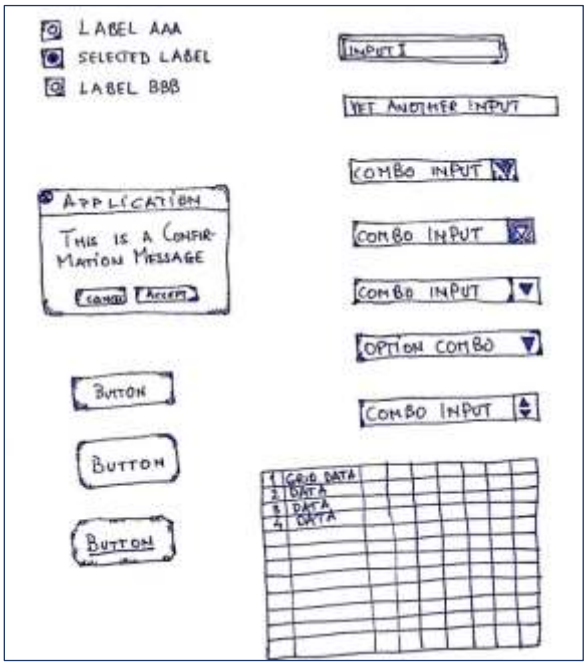


Figure 20 – Elaborated user interface hand-drawn controls

3.4.2 Dataset experimentation and generation

The artificial scene training dataset has been designed from the very beginning with the purpose of training graphs based on convolutional modules in order to classify, localize and segment all the known visual controls within a user interface screen and generate dense prediction that can be translated either in a intermediate language or into a final deployment language script such as HTML5. From our early stages of research and experimentation we have defined two particular goals in terms of dataset experimentation: (a) the goal of defining an output inference format (not necessarily directly generated by the DAG) and (b) the goal of preparing a dataset for training DAGs for artificial scene inference.

As mentioned, in our early initial experiments we have created an *Intermediate Translation Definition Language* (ITDL) [6] based on JSON script structure as described briefly in Figure 21.

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

Figure 21 – IDLT example JSON code

The artificial dataset preparation process mentioned also in the 2018 paper [8] consists in various software generated user interface controls and full screens. The actual general algorithm for generating the single-control dataset observations, described in python-like language, is presented in Figure 22 where we have ***C*** as the space of possible visual controls, ***S*** the space of

possible styles for each type of visual control and ***render*** is a function that generates an image and its dense pixel map classification. Each observation is a 352x352 image with 3 RGB color channels. The 352 by 352 size has been chosen in order to easily accommodate a series of down-sampling operation, based on convolutions that reduce the input size by half and finally obtain a 11x11 volume that can be divided in 11 by 11 patch-like regions. Each observation size is 371,712 bytes resulting, based on our experiments, in an optimal dataset batch size of 3072 observation summing approximatively 1GB of data – generating a good-sized meta-batch for the graph optimization process. As a result, each dataset batch is processes individually and in-memory during DAG optimization in various mini-batch sizes, taking a subset of the 3072 observations for each forward and backward propagation, as it will be further explained in the model architecture section.

```
GetControlObservation:
  for c in C:
    for s in S[c]:
      o_i, o_m = render(c,s)
      x = (o_i, o_m)
      y = c
      add_to_batch_of_yield(x,y)
```

Figure 22 – User interface control generation based on "render(c,s)" function that takes the UI control type "c" and style "s" and generates the control image and the corresponding dense pixel classification map

A second, slightly more complex algorithm is employed for the generation of artificial scene observations. For scene observations, that mainly serve as test data, the size of each observation and the actual dataset size is arbitrary. This means that we do not follow a strict pattern of input image size for each observation nor an information or label density rule within the test dataset. The purpose of the algorithm for artificial scene generation presented in Figure 23 is to generate observations that, although do not necessarily have a real user-interface look-and-feel, resemble

actual UI screen-shots in terms of number of visual interface controls, positioning, alignment, and so on.

```
GetSceneObservation:
  n = random(N)
  h = random(H)
  w = random(W)
  scene_list = []
  for i in n:
    c = C[random(C)]
    l = random(h) + delta_x
    t = random(h) + delta_y
    l, t = check_for_overlap(scene_list, l,t)
    render_to_scene(scene_list, c, l, t)
  scene_map = scene_to_map(scene_list)
  return_or_yield(scene_map)
```

Figure 23 – The user interface screen (scene) generation algorithm

The hand-drawn dataset has been prepared based on hand-drawing each individual visual control and user-interface mockups and then followed by scanning and image adjusting process. Following the scanning process, we have employed various image dataset augmentation (enlargement) procedures such as random rotation, random shifting, random channel shifting, random flipping, random rescaling, random cropping. Due to the complicated nature of the natural dataset generation procedure we required the dataset augmentation procedures in order to increase the observation number of the actual hand-drawn images. One important aspect we have to mention is that most of the hand-drawn observation have a limited spectrum of colors being mostly based (as presented in Figure 20) on one color.

For further explanation of our expected results for a hand-drawn mockup scene inference, we will present an actual experimentation output in Figure 24 based on the hand-drawn user interface mockup in Figure 19.

```
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" : {

                                   ....
```

Figure 24 – Output of a scene inference process in the intermediate JSON-based script

3.5 Final proposed architecture

The main building blocks of our proposed architecture are based on current state-of-the-art presented within section 2. The overall architecture, briefly presented in *Figure 17*, combines elements from current state-of-the-art Deep Vision architectures all together in a final optimized architecture that aims at minimizing the required resources and inference time, reduce the required training time and training data and perform the task of visual inference with near-human inference capability. A more detailed presentation of the proposed architecture can be observed in Figure

25 where we have the basic version of the CloudifierNet architecture without the addition of our proposed MGU sub-graphs.

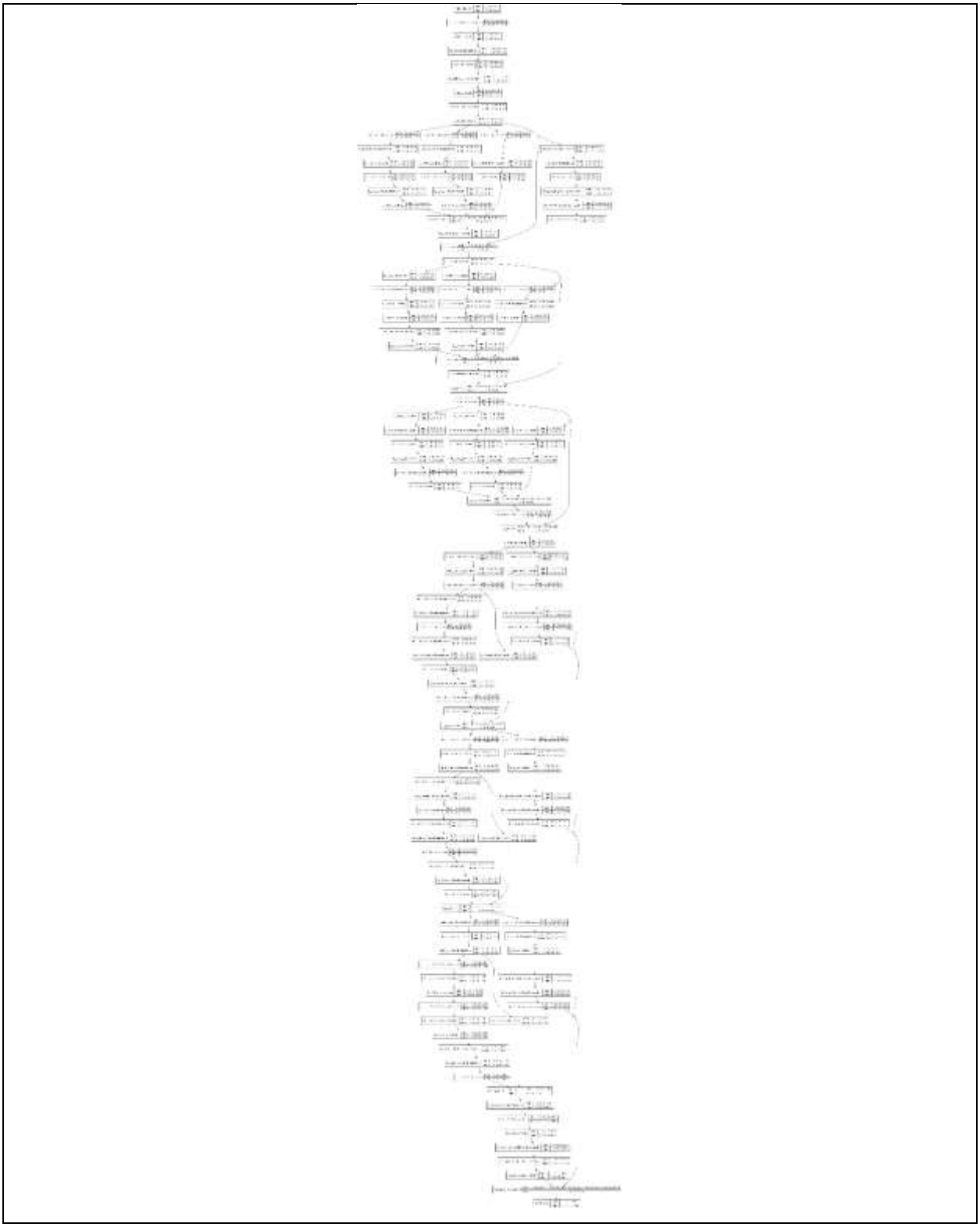


Figure 25 - In-depth presentation of the CloudifierNet architecture

Due to the size of the graph we will break it down in components and analyze each one. The DAG architecture has two main sections, namely the "*Low*" and the "*High*" section with each section containing multiple sub-sections as it will be further presented.

3.5.1 Low section

The so-called *Low* section has the purpose of performing the traditional convolutional neural network pipeline. The initial sub-section of the *Low* is basically the stem sub-network – a sequence of CNN modules – presented in *Figure 26* - that have the sole purpose of reducing the size of the input and learn lowest level convolutional kernels (such as simple visual primitives like lines and dots).

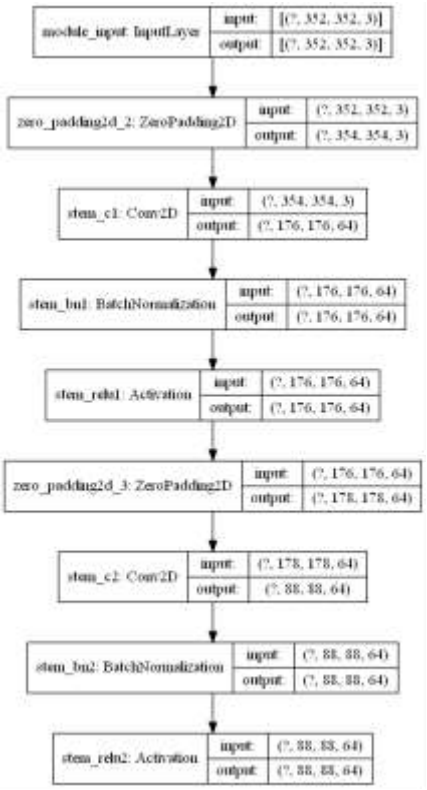


Figure 26 - The "stem" block of the CloudifierNet architecture

Following this initial sub-section follows the second and more complex one composed of a series of modified *Inception* modules with residual/skip connections as presented in *Figure 27*. As it will be presented, we will forgo this module architecture in the later stages of the network in

favor of more resource-efficient ones. Finally, at the end of the *Low* section our output is both sent to the readout module of the *High* section and to input of the *High* section. As we will see, the signal that goes straight to the readout module will feed the final output SoftMax with a finer grained feature map generated by the *Low* section. Nevertheless, this skip-signal is processed using a transposed convolution in order to match height and width of the input volume without any non-linearity.

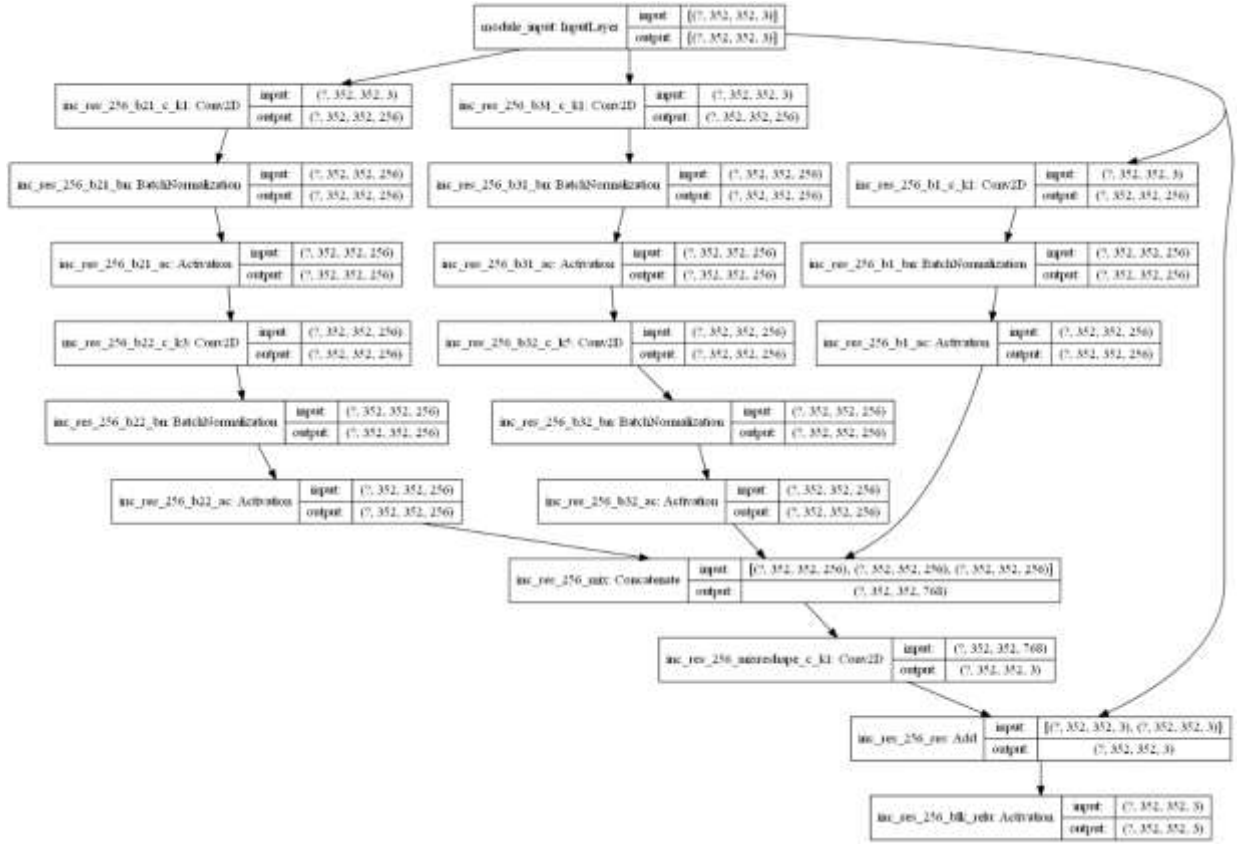


Figure 27 - InceptionResNet block used in CloudifierNet architecture "lower" section

An important aspect of the lower section is the the addition of a special graph module responsible with the reduction of the height and width of the feature maps – the so called "*ShrinkModule*"

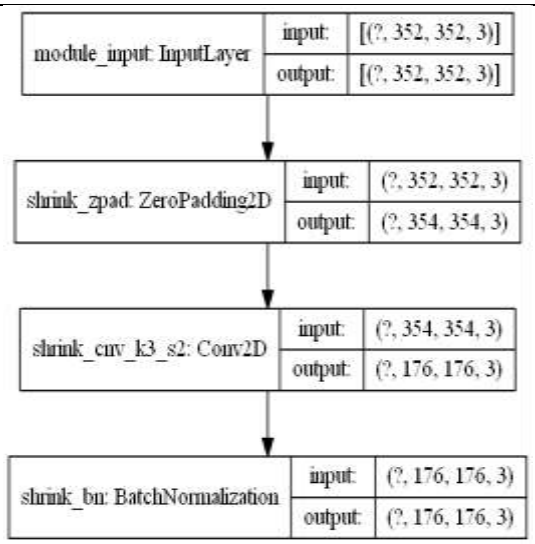


Figure 28 - ShrinkBlock graph module that has the purpose of reducing the size of the input feature maps

3.5.2 High section

At this point in our DAG pipeline we have arrived at a point where the dimensionality of the initial input has been drastically reduced in height and width and the depth has been enlarged. Due to the large volume that we have to further process in order to infer more higher-level features and the fact that we determined the requirement to further increase the depth of the processes volume we must make a change in strategy. We employ in this section separable convolutional modules – presented in *Figure 29* - similar to the ones described by Chollet et al in *Xception* architecture presented in our 2.1.2 sub-chapter, that allow us to use more efficiently the model parameters.

Figure 29 - The Higher section main workhorse that uses depthwise separable bi-dimensional convolutions

Within this particular section each separable convolution module generates a signal that is both directly fed into the readout module (via a transposed convolution with similar function as the ones in the *Low* section) and also fed as input for the next higher module in the series as presented in *Figure 30*. This technique is similar with the technique that we used at the end of the previous DAG section, namely the *Low* section. Certainly, we have to mention that beside the benefit of the fact that our SoftMax readout will "*see*" various types of feature map granularities, our gradient signal will easily flow directly from the loss and readout module to each of the separable convolution modules – as well as within the top module of the *Low* section.

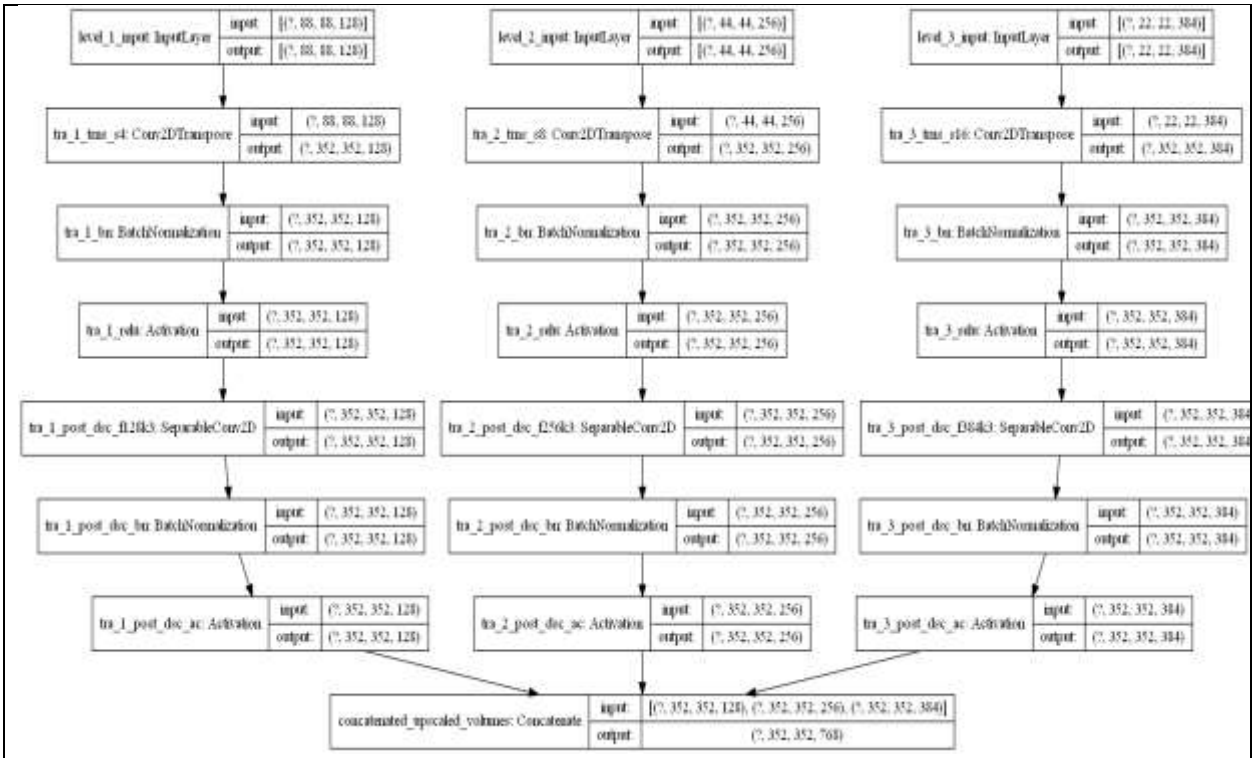


Figure 30 - The structure of the UpscaleBlock where each input comes from a different level of our graph - lower levels have lower feature map depths while higher ones have bigger feature map depths

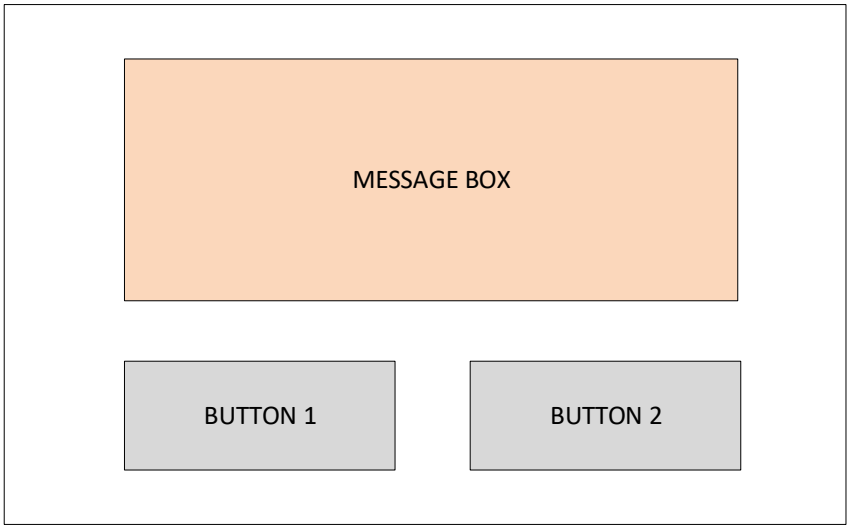


Figure 31 - Simple mockup of a visual screen where we have a message box and two buttons

Finally, the readout section is nothing more than a concatenation of the multiple inputs coming from the High section with the one coming from the Low section. This dense output volume that basically has the size of *H*W*S* where *H, W* are the height and width of the input and *S* is the number of output signals (*1* from *Low* section and *S-1* from *High* section), is then activated with the dense SoftMax function as described in equation (5). This basically generates a SoftMax prediction over the target classes for each and every pixel in the input image that has a corresponding "fiber" in the final output volume. The actual output of the dense prediction map based on the simple mockup scene in Figure 31 can be observed in Figure 32 where *0* is the background target class, *91* is message box target class and finally *12 and 13* are the classes of the buttons. In this particular case each button is assigned a particular class as the model is capable of inferring multiple types of buttons (such as "*Cancel", "Ok", "Accept", "Retry", "Close"*)

The loss is then calculated over each of the target pixel classes with negative log-likelihood described in equation (6).

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	12	12	12	12	12	12	12	0	0	13	13	13	13	13	13	13	0
0	12	12	12	12	12	12	12	0	0	13	13	13	13	13	13	13	0
0	12	12	12	12	12	12	12	0	0	13	13	13	13	13	13	13	0
0	12	12	12	12	12	12	12	0	0	13	13	13	13	13	13	13	0
0	12	12	12	12	12	12	12	0	0	13	13	13	13	13	13	13	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32 - Example of a dense output inference results based on a input scene with one UI message box and two UI buttons

3.5.3 Script decoder graph

As previously mentioned the proposed directed acyclical graph based on convolutional modules has been designed with the purpose of generating dense maps of the inferred User Interface scene (either artificially generated or hand-sketched by a designer or a simple user).

In our past and current experiments, we have discovered that employing the *CloudifierNet* architecture and generating a dense classification map of the input image shortens dramatically the training time of a potential decoder for source code. Basically, instead of forcing our end-to-end model of creating correlations between the actual image and the source code – with or without the aid of visual attention mechanism as described in [37] - we make the task easier for the training process by *translating* the image into the actual *dense pixel level map* of UX controls and primitives. Finally, using this high-level information as the decoder context as presented in *Figure 33* we can perform the auto-regressive process of generating the code. As a result, this approach will allow us to train our model in near-real time in order to adopt any target UX programming language.

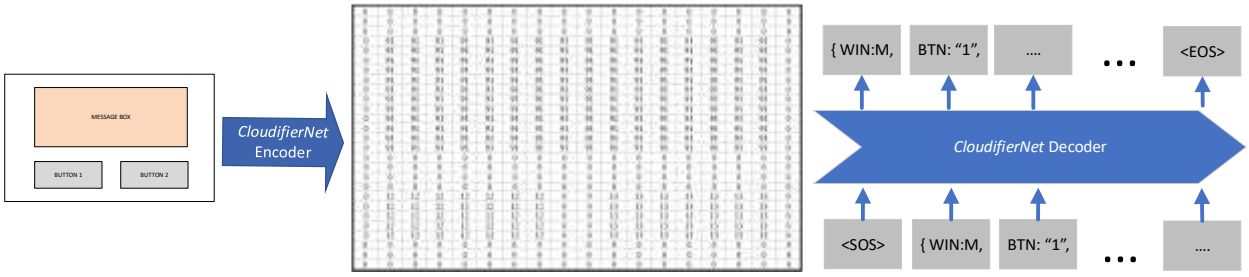


Figure 33 - Generic inference process using proposed end-to-end model. Instead of receiving a context state the decoder receives the pixel-level inference of the input image

At this moment we already implemented a simple recurrent-cell based graph using stacked LSTM cells that follows the classic approach of decoding the source code based on teacher-forcing training. We are also experimenting with various soft-attention mechanisms to further improve what we consider as the baseline for our future work in this area as it will be presented in the future research sections.

3.6 Model optimization process

The optimization process of the proposed directed acyclical graph architecture is basically divided in two main stages:

1. the initial convolutional based directed acyclical graph training that generates the model capable of transforming a input image into a dense map of pixel labels where each individual point belongs to a certain visual control class defined by the CloudifierNet dataset
2. the code generation model training stage that is mainly focused on training the recurrent directed acyclical decoding graph

3.6.1 The convolutional graph training

The convolutional graph training is straight forward based on the dense logits of the final normalized probabilities generated by the last layer in conjunction with each individual pixel label.

The negative log-likelihood objective function (6) can thus be used as in all image segmentation classic approaches. Nevertheless, due to the nature of some of the observations consisting mostly of background information and few visual-interface controls we observed a natural tendency of the models to be focused on well-inferring background pixels. As a result, we decided to employ in our experiments a modified version of cross-entropy that will enable a down-scaling of the gradients for the well-classified pixels and thus lower the "power" of the background classes. This particular approach has been based on the focal loss [25] proposed by Lin et al. In *Figure 34* we can see that by adjusting the γ hyperparameter to higher values we can drastically reduce the *cross-entropy* loss contribution of well classified examples. This balancing operation has a major impact on the gradient for the cases when a scene has the majority of pixels composed of background, thus resulting in easy classification of background that will overwhelm the gradient with the easy classification information.

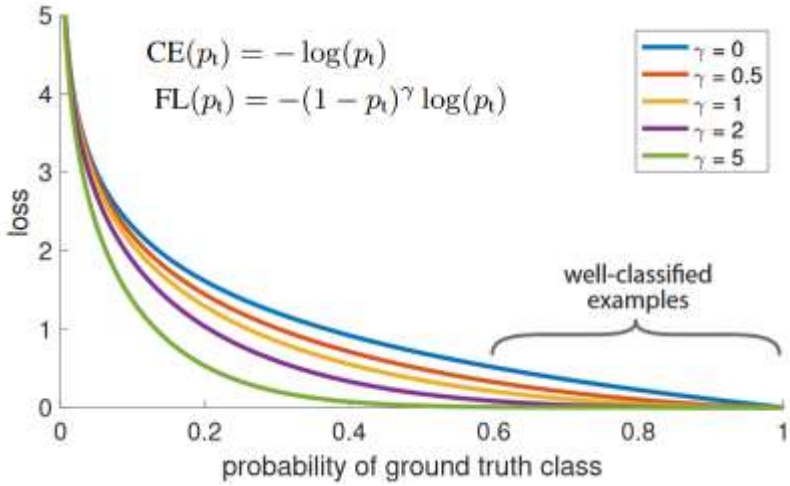


Figure 34 - Focal Loss behaviour with different values of the hyperparameter showing how well classified examples have lower impact on the overall loss and thus gradient. Image based on original paper by Lin et al https://arxiv.org/abs/1708.02002

We trained the proposed models end-to-end using *Adam* [51] optimizer on batches of variable size in multiple training experiments. We used an initial learning rate of 0.01 and applied learning rate decay based on monitoring the dev-dataset loss plateauing behavior.

3.6.2 The recurrent graph training

The training process of the final decoder model is using a discrete subset of source code examples that are one-to-one matched with a discrete subset of training observations from the base CloudifierNet. This subset contains beside the fine-grained dense pixel-wise control labels also a source code translation of the image based on a specific target language - HTML for example. The main difference of this CloudifierNet dataset subset from the bulk of the dataset is that it only contains full-scene images of user interfaces and no simple fragments or single control image cuts. This approach ensures that the decoded source code has a minimal user experience functional meaning that would not be capture by simple one-control images and their source code (for example HTML). The actual training procedure assumes that the decoder will tokenize each individual character in the target source code and will feed the character-level RNN decoder with the tokens which in turn generates character level probabilities. As a result, the decoder can be trained with negative log-likelihood cost function in supervised manner.

3.6.3 Attention always pays off

As described in previous chapters our proposed directed acyclical graphs is optimized end-to-end with the single purpose of generating a dense pixel level classification map of the input user interface. This approach has a certain pitfall similar with other cases where semantic segmentation is the solution to the given problem, that of generating wrong classification for a certain area of pixels. In order to be more explicit and have a good intuition about this issue let us take the output map presented in a “idealistic” case in *Figure 32* and see it in a more realistic scenario presented in *Figure 35* where a certain degree of pixels are wrong classified with classes that maybe are similar to the real gold target. In this case a simple decoder such as a shallow sliding window based analyzer of the input map or a more complex sequence-to-sequence script code generator will have problems decoding the input image into the target script language due to the ambiguity of the two inferred user interface zones (the visual control class 91 and the bottom left visual control class 12). In this particular case the proposed learned 2D attention mechanism will come to the rescue by generating a high-density attention map overlapped over the dense-prediction map. This end-to-end graph optimization approach will enable the teacher-forced decoder to drop the minority

class in favor of the majority class when having ground-truth target tokens that are assigned for control class 92 or respectively 12 as in our example.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	**92**	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	**92**	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	**92**	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	**92**	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	**92**	91	91	91	91	91	91	91	91	91	91	91	91	91	91	91	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	12	12	12	**13**	**13**	**13**	**13**	0	0	13	13	13	13	13	13	13	0
0	12	12	12	12	12	12	12	0	0	13	13	13	13	13	13	13	0
0	12	12	12	12	12	12	12	0	0	13	13	13	13	13	13	13	0
0	12	12	12	12	12	12	12	0	0	13	13	13	13	13	13	13	0
0	12	12	12	12	12	12	12	0	0	13	13	13	13	13	13	13	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35 - Example of a more realistic dense output inference where certain pixels (emphasized in the image) are not correctly classified. The grayed area in the output map represents a 2D attention map that is overlapped on the dense inference map and helps the sequence-to-sequence decoder focus on the correct type of user interface control.

4 Experimental implementation, execution and evaluation *(completed 75%)*

In the following paragraphs we will analyze the whole experiment, initially from a engineering point-of-view and finally from a scientific point-of-view. Our objective is to analyze the whole environment infrastructure that has been used during experimentation for various model training as well as the approach for model operationalization and finally the analysis of model performance and the related key findings.

4.1 Experiment execution environment

Our experiment has been mainly conducted on a on a GPU-based parallel numerical computing infrastructure however for inference we used both CPU and GPU in order to asses the performance of our models in various infrastructure settings. As far as the optimization process is concerned - optimization process presented in section 3.6 - we used the GPU CUDA environment due to our heavily GPU-optimized architecture of our models. The training computing infrastructure consisted in 2 Xeon CPUs each with 8 physical cores and a total of 16 logical cores (threads) summing 32 parallel thread capabilities and a total of 32 GB RAM and 512 GB fast SSD storage. The parallel numerical computing infrastructure based on CUDA GPU has been based on two parallel Nvidia Quadro with Pascal architecture – P4000 and P500 – summing 4224 CUDA cores and 24 GB or VRAM.

4.2 Operationalization approach

One of our goals during operationalization of our experiment has been that of evaluating the potential target platforms of our experiment and potential future product. As a result, for the operationalization of our experiment we used a wide variety of experimental environments ranging from high-end mobile computing devices with CUDA GPU capabilities down to low-end mobile computing devices with CPU-only numeric processing option.

A particular focus in our operationalization experimentation has been that of deploying our models and the inference setup on embedded devices such as NVidia Jetson TX2. This particular objective has been based on the fact that one of our future goals is to deliver a minimum viable product that can be deployed using embedded compact specialized hardware.

The final proposed approach for our experimental operationalization environment has the following basic characteristics:

- Cross-platform hardware. We are using at least three different scenarios in order to ensure that our proposed architecture and the overall experimental process can scale with acceptable loos of performance on all potential architectures
- Special configuration of pretrained DAGs. For CPU-based environments we deploy a different DAG architecture that uses a minimal set of required parallel computations as

for the GPU-based environments we use more complex and deep DAGs – even if we deploy on embedded GPU devices such as Nvidia Jetson TX2.

- Mobility. Due to the nature of our target domain we have identified the potential industrial (market) need for fast deployable proof-of-concept applications via application marketplaces (such as Apple AppStore or Android Google Play marketplace). Nevertheless for this particular area we have a limited capability to deploy our experimental operationalization environment and we are still researching for best viable DAG architecture and execution (inference) setting. . . . . .

4.3 Raw results analysis

The main output of our experimental work consisted in the comparison of inference tests applied to our test data using the various model architectures. The proposed model architectures, as presented in previous sections, have been based both on simple models with complex scene-inference algorithms and also on advanced deep directed acyclical graphs with the capability of end-to-end scene inference.

Model	Single image accuracy	Single image recall	Single image precision	Scene accuracy	Scene recall	Scene precision
ShallowC	…	…	…	…	…	…
CloudifierNet v1	…	…	…	…	…	…
CloudifierNet v2	…	…	…	…	…	…

Figure 36 - Shallow vs Deep model accuracy, recall and precision on artificial images

Although, in terms of processing complexity of this final proposed computation graph, the complexity of *CloudifierNet* [8] is much greater that of the initial shallow models [6], we have discovered that for the end-to-end task of analyzing a full visual scene the total inference time if actually comparable between the two approaches. Nevertheless, in terms of inference capacity the deep graph approach is far better than the initial shallow one as it can be observed in Figure 36.

4.3.1 Real-life applicable results

One of the most important goals of our research and experimentation has been that of paving the way for a future potential minimum viable product that will fully take advantage of the present work. To this particular end our proposed potential minimum viable product has been already planned as a cross-platform system that will be able to run infercences on various target devices. A minimal proof-of-concept product has been designed with a minimal web-based user-interface and a set of already optimized DAG models. Furthermore, we have proposed the following set of minimal features within our production grade solution:

- Uploading of various sizes of images and on-shot inference resulting in dense classification of the image in matrix-like format (for demo purposes).
    - Artificial image input
    - Hand-drawn image input

- Advanced feature of uploading of various sizes of images followed by on-shot inference of the artificial scene and the generation of the resulting JSON that defines the scene attributes as presented in Figure 24.
    - Artificial image input
    - Hand-drawn image input will be further analyzed/experimented
- The final Proof-of-Concept feature is that of the end-to-end conversion where we give input an artificial scene and we generate an actual HTML5 output with full source code. As inputs we have:
    - Artificial image input
    - Hand-drawn image input

- . . . . . .

4.3.2 Research results

The main research results . . . . . .(architecture, optimization, bechmarking, usage of hand-drawn training data)

5 Personal contribution areas and final conclusions *(completed 75%)*

5.1 Multi-Gated Units

Probably the most important innovation proposed in our work is the introduction of the Multi-Gated Units (MGU) presented in Section 3.2. Our hypothesis is that the MGU can perform equally well in any kind of experiments with almost any kind of graph modules.

. . . . . .
. . . . . .
. . . . . .
. . . .

5.2 Convolutional rchitectures

During the research and experimentation process we have explored the possibilities of directly employing various state-of-the-art architectures such as the ones mentioned in previous sections [13] [12] [52] [22]. However due to the specific "artificially engineered" nature of our observation space, its limitations in terms of available training data and also due to the nature of our target experimentation and deployment environments, the classic state-of-the-art DAG architectures have either optimization or performance issues. For these particular reasons we designed and finally developed various custom graph model architectures. Our final model *CloudifierNet*, presented in section 3, has been designed within multiple research and experimentation iterations using the various architectural tools and techniques described in related work section 2 together with our own incremental scientific improvements.

It is important to note that in our quest for obtaining the most viable model architecture we even tackled simple and more naïve solutions such as employing ensembles of shallow linear models and performing sliding-window [6] over the input data volumes. We also analyzed

advanced GPU-based parallel programming approaches, with actual applicability in real-life environments for similar scientific computation, that could maximize the potential and the overall performance of the proposed shallow-model solutions [7]. As an example, such an approach allowed us to run multiple parallel shallow classifiers, each parallel classifier examining multiple windows generated by the sliding windows algorithm in parallel, all based on CUDA GPU computing environment. This overall strategy allowed us to gradually explore all the potential solutions of our research and real-life experimentation problem of machine-learning based artificial visual scene inference. As previously presented, we finally arrived at the current milestone point of our research and development, proposing our custom DAG architecture.

. . . . . .
. . . . . .
. . . . . .
. . . .

5.3 Real-life experiments and advances beyond current SotA

The second important contribution of our experiments has been the creation of the *artificial dataset* that can be considered novelty. Although other experiments such as [9] have been conducted in recent years in this particular domain of user-interface inference and reconstruction for migration purposes, we have no knowledge that a dataset such as ours exists. As mentioned before, our dataset consists of a wide variety of user-interface controls and "*random*" UI scenes that have been carefully labeled in order to allow both observation-wise classification and also pixel-wise classification (segmentation).

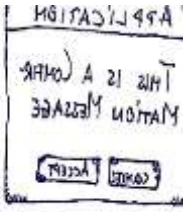
During our research and in particular during the research and experimentation of the *artificial dataset*, we analyzed a multitude of different user interfaces styles and visual approaches. Nevertheless, the research has been conducted taking into consideration the range of target user interfaces: legacy graphical user-interface based applications developed since the early start of GUI-based operating systems such as Windows 3.1 (1995-2005) or simple user-interfaces that comply with the classic user-interface rules. By user-interfaces rules we imply the actual strategy for designing, selecting color schemes, placing and thus interacting with the visual controls. Our final target has been that of obtaining a visual control artefacts distribution that will allow us to

safely consider that by using the proposed training dataset we have a high probability of obtaining a trained model that generalizes very well.

A particular case of our artificial dataset research and experimentation process has been that of creating a hand-drawn simulation of user-interface controls as previously mentioned in section 3.4.1 and section 3.4.2. This has been a particularly challenging task requiring two stage experimentation: first stage consisted in the actual sketching of the hand drawn examples with various levels of hand-drawing expertise and second stage has been that of preprocessing the given images with various automated techniques. The two-stage experimentation has been required due to the limited number of sketches that could be prepared and proposed resulting in insufficient original sketches for a minimally viable training dataset. To be more specific in below Figure 37 we present an actual example from the dataset augmentation experimentation process.

original app message sketch

Derived training observation



Derived training observation

Derived training observation

Figure 37 - Sketch training dataset augmentation

As it can be inferred from Figure 37 we applied multiple image transformation techniques such as:

- shearing (deformation by shifting and cutting)
- zca whitening (decorrelation of pixel features)

- random brightness modification,
- various levels of zooming
- random image shifting horizontally and vertically
- flipping the image vertically (turning upside-down).

Based on these techniques we obtain images that have a more varied spectrum of colors than the original (as it can be observed in Figure 37) and thus solving the issue related to single-color in original data (hand-drawn source) presented in section 3.4.2.

Another important observation is that, in our experiments, we did not employ image transformation techniques that are known to create difficulties for the convolutional kernels in known convolutional deep graph architectures – such as vertical flipping on above 45 degrees image rotations. Although it is known the recent work related to Capsule Networks [53] [54] done by Professor Geoffrey Hinton regarding this particular issue, all our deep neural graph components and the resulting state-of-the-art architecture is based on convolutional modules and techniques presented in Section 2.1.

. . . . . .
. . . . . .
. . . . . .

5.4 Cross-domain applications

One particular direction where our research has pivoted to has been that of applying the *CloudifierNet* architecture in other domains and particularly in skin lesion diagnosis. Dermatology is one of the important areas where Computer Vision has a great potential to shine and we already have important results and even benchmarks against human radiologists/diagnostics. For this particular area we already used our architecture against the HAM10000 dataset [55] in conjunction with analyzing and benchmarking both work from past years [56] and more recent [57]. This work is currently still in development stage and we plan to publish in the next period our research and experimentation conclusions.


POLITEHNICA


Computer Science
& Engineering
Department

6 Proposed future research and development *(completed 85%)*

In the following paragraphs we will tackle the limits of our current research in order to find the issues and opportunities that will lead to further incremental improvements of our experimental research and development work. Up to this moment we have identified two major areas where improvement is further needed in order to achieve an optimal potential for a proof of concept experimental solution that could be further developed and deployed into a Minimal Viable Product. The proposed areas for further research and experimentation are:

- ✓ Augmentation of naturally generated (hand-drawn) datasets in order to achieve a better coverage of this particular domain of observation that will lead to a trained DAG capable to recognizing and inferring complex hand-drawn designs;
- ✓ Programming language agnostic end-to-end generative model for source code output that will be able to generate functional source code based on the input image(s);
- ✓ Introduction of rotation and horizontal flipping invariant DAG architectures based on Capsule Networks by further adding incremental research on top of existing Capsule Network research [53] [54];
- ✓ Appling our research and experimentation in the area of Robotic Process Automation (RPA) applications and finally propose an integration approach;
- ✓ Process flow logic inference end-to-end model research and experimentation that will allow us to infer both the general intent of a particular sequence of actions and interface screens and also the process logic behind the intent;
- ✓ Last but not least we plan to extend our research in the area of Multi Gate Units – our own innovation that applies to multiple domains where Deep Learning can be employed.

6.1 Advanced hand-sketching inference

One of the main proposed future research activities is related to the enhancement of our models capabilities for advanced inferring of hand-drawn sketches. We aim at obtaining an actual functional and viable solution for this task that can be summarized as “from hand-drawing

mockups to functional online applications". In this area we have the following proposed objectives that will further advance our current research and experimental development:

- ✓ Further augmentation of the natural dataset (hand-drawn) with multiple observations for each potential user interface primitive
- ✓ Augment the user-base that generates the proposed natural dataset observations

6.1.1 From story-boards to online applications

Based on the future research in the area of advanced hand-sketched scene inference we have the objective of introducing the concept of hand-drawn storyboard observation(s). This particular type of observation (or multiple observations – to be decided within the future research stages) will define one or multiple processes that describe user experience flows. This new observation type (that could consist in an actual time-series of multiple sub-observations or a video stream) will describe in a natural, visual and common graphical way what the end-user would require from the proposed computer application. The concept of storyboards, taken from the movie industry has been long adopted within the software development/engineering horizontal with actual formalizations embedded within UML (Use Cases scenarios, etc) or simple hand-drawn approaches in quick-sprint, agile-based projects.

Finally, our aim is to develop an intelligent solution based on our further research that will offer - based on a specific set of target user-experience functionality subdomains - the possibility shortcut the whole software design, software development, implementation and deployment process of the UX.

More information regarding this future proposed advancement is presented in the section 6.3 that analyses the tools and approaches needed to generated reliable UX script code directly from the analyzed image (natural or artificial) and section 6.6 that gives more intuition regarding the process of inferring and analyzing whole process flows and the underlying logic.

6.2 Reward-based continuous learning

One of the main areas of further research and development that has been planned from the early inception of our project is that of introducing Reinforcement Learning to our ecosystem. The final proposed system can be formalized as an interaction between an *agent* (the model and its operationalization infrastructure) and the actual user, all based on an observed *environment* that consist in the actual scene to be inferred. Our goal is to transform our model and underlining system from the supervised learning setting to the two-stage trainable system. The first stage will consist in the initial supervised optimization of the proposed DAG and the second stage will employ further Reinforcement Learning based tuning of the model based on the *reward* signal generated by a qualified user that reviews the model output. Currently we are exploring both the idea of applying policy gradients based update to the core DAG and even the possibility of creating a secondary policy-function DAG that will support the main generative DAG. Due to the nature of the problem and its optimization process we think that applying the raw *REINFORCE* [58] method will not yield improved results over the standard supervised-learning setting and thus we will experiment with actor-critic approaches that combine the policy gradients with the employment of value-based critics.

To be more precise, we are pursuing the idea of adapting our architectural pipeline and approaching this subject using *Deep Deterministic Policy Gradient (DDPG)* [59]. Even more precisely we will try both the recent advances in actor-critic methods such as *Soft Actor Critic (SAC)* [60] and the simpler *TD3 (Twin Delayed)* [61] version of the DDPG.

. . . . . .
. . . . . .
. . . . . .

6.3 From image to source-code generation

A particular field of research and development currently tackled within our work is that of direct end-to-end model-based source code generation. Our current architecture supports the addition of a decoder module that is trained end-to-end with a pre-trained *CloudifierNet*. The exact details of this particular module are presented in section 3.5.3. Nevertheless, we are currently analyzing the application of more advanced neural language models approaches such as the *Transformer* [62] family of models. Our final target is to create an end-to-end trainable graph that would include both the *CloudifierNet* architecture and the source code sequence-based decoder module, based on multi-head dot-product attention presented in *Figure 38*. This approach will enable us to construct a multi-language and multi-purpose architecture capable of generating deployable source code in a variate of languages.

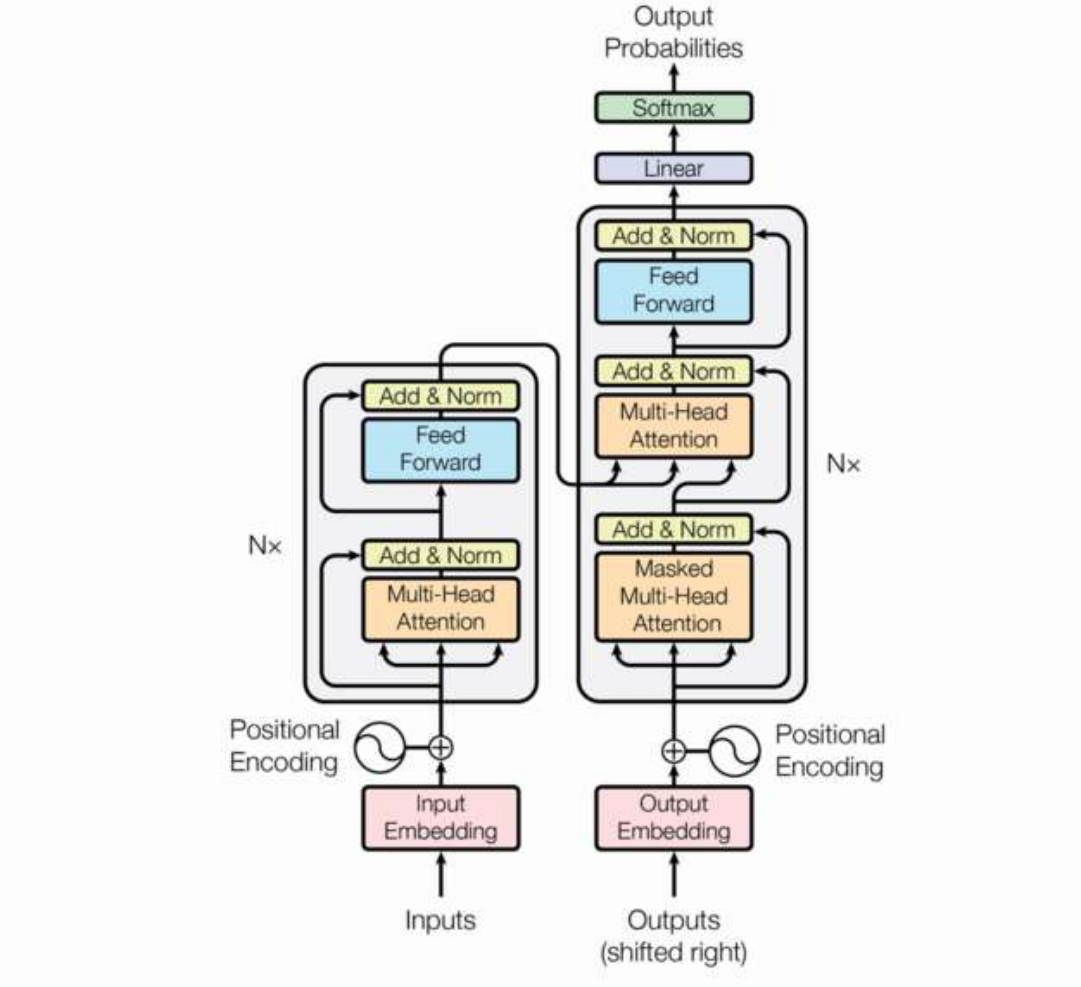


Figure 38 The Transformer architecture based on "Attention is all you need" by Vaswani et al

For this particular area of further research and experimentation we plan the following activities and objectives:

- ✓ Experiment with advanced versions of the *Transformer* architecture such as *BERT* [63] and *GPT-2* [64]
- ✓ Develop end-to-end DAG - with or without the pre-training of the encoder – using specific objective function to tackle both the dense prediction map training and the actual decoding

- ✓ Programming language agnostic system: Experiment decoupling the programing language knowledge from the model by injecting additional information in decoder and transforming the language semantics into actual model input
- ✓ Apply the end-to-end code generation approach to advanced hand-drawing inference features

The final goal of this process will be that of architecting and implementing, as previously mentioned, is two-fold as follows:

- Obtain a graph model capable of code-language script generation with minimal adaptation and fine-tuning to new script languages
- Introduce what is probably the most ambitious goal: that of generating application user-interfaces and process logic scripts for entire processes (section 6.6)

6.4 Rotation and vertical flip invariant models

It is well known that one of the major flaws of current state-of-the-art end-to-end computer vision DAGs is their inability to capture vertical flip (turning of images upside-down) and above 45° rotations. In order to address this particular issue in our future iterations of the proposed system, we plan to continue our research based on alternative approaches to the classical convolutional neural models. The first proposed path is the research and integration of the proposed work by Hiton et al [53] [54]. We plan to explore the full use of directed acyclic graphs based on Capsule modules and potentially contribute to the state-of-the-art in the area of Capsule based models.

6.5 Robotic Process Automation (RPA) experimentation

Beside the proposed research and experimentation work presented in sections 6.1 and 6.4 we plan to explore the integration of our proposed models and overall pipeline in applications from the domain or Robotic Process Automation.

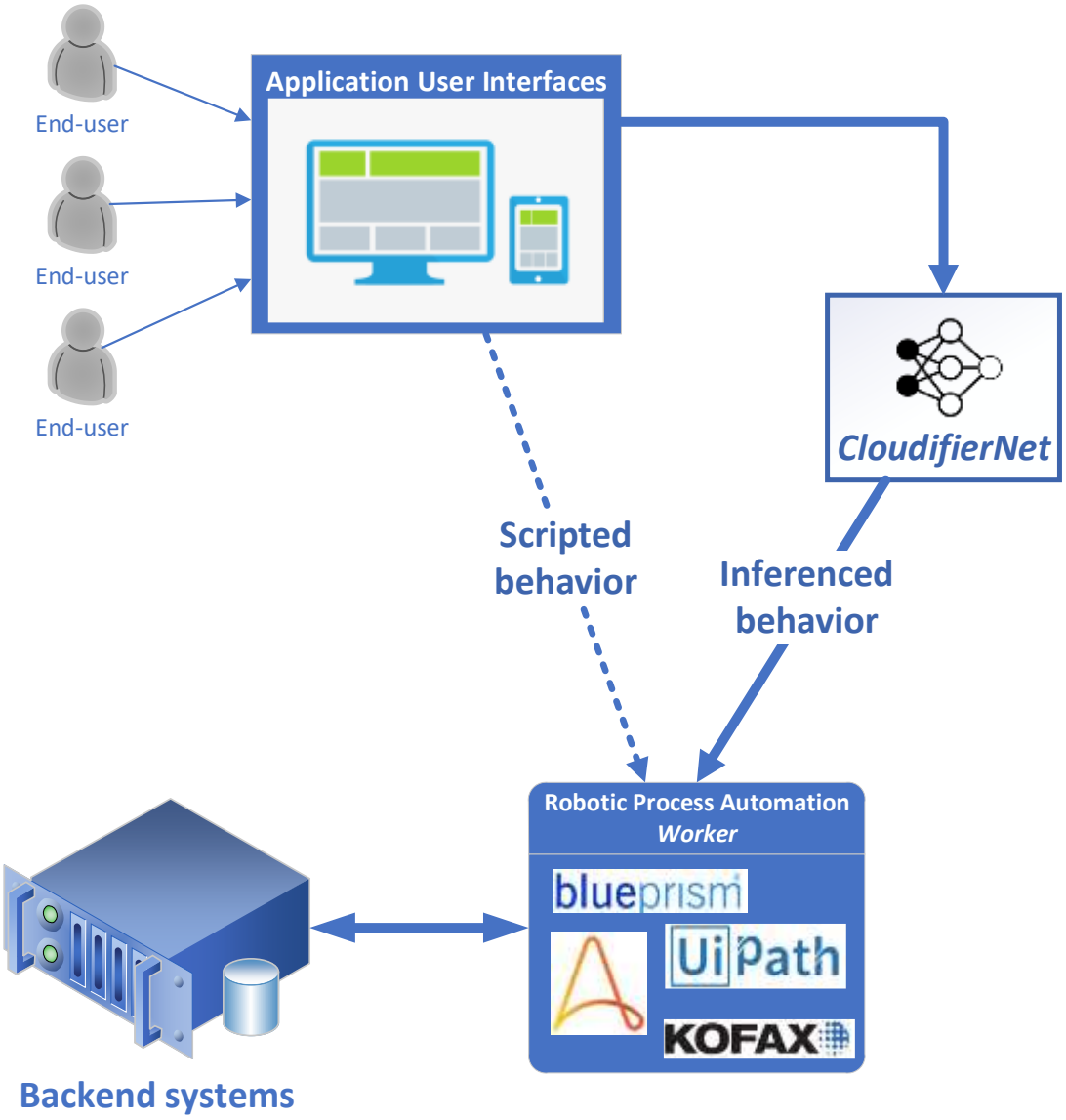


Figure 39 Inferenced vs classic scripted behavior analysis

To our knowledge several companies from the RPA industry are currently as of 2018 doing extensive research in the area of user interface recognition in order to further enhance the user-interface automation processes. To be more specific, one of the key areas the RPA industry is targeting is that of understanding the actual interaction between user and user-interfaces without having prior knowledge of the specific application or of the overall application user-interface ecosystem. This basically allows the creation of application-agnostic user-interface RPA agents that are able to fully "understand" the behavior of the user-interface and its interaction with the

final user and emulate the whole user-to-UI process. Based on our state-of-the-art research and experimentation we can deploy our models directly within the RPA systems giving them the ability to infer the "semantic structure" of the analyzed UI vs the existing approaches that use scripting, manual tagging and manual adnotation of the user-application interaction as presented in the *Figure 39 Inferenced vs classic scripted behavior analysis*.

6.6 Process flow intent and logic – from UX to backend

Closely coupled with our goal of researching and developing an approach able of understanding the actual intent of a series of sketches and mockups, previously mentioned in 6.1.1, we aim at researching and developing an end-to-end trainable DAG that will be to infer the actual process flow intent of a series of actions and screens. This will be the first step on a longer road of actually researching and developing methods of inferring the whole logic behind a given application and crossing the gap between current state of UX analysis and translation and the actual capability of porting whole applications. The final goal on this path is to research and develop a system truly capable of whole automated application translation/migration for a certain range of application types - targeting application relying heavy on the user-experience for multiple industrial horizontals (such as retail applications, CRM applications, and so on)

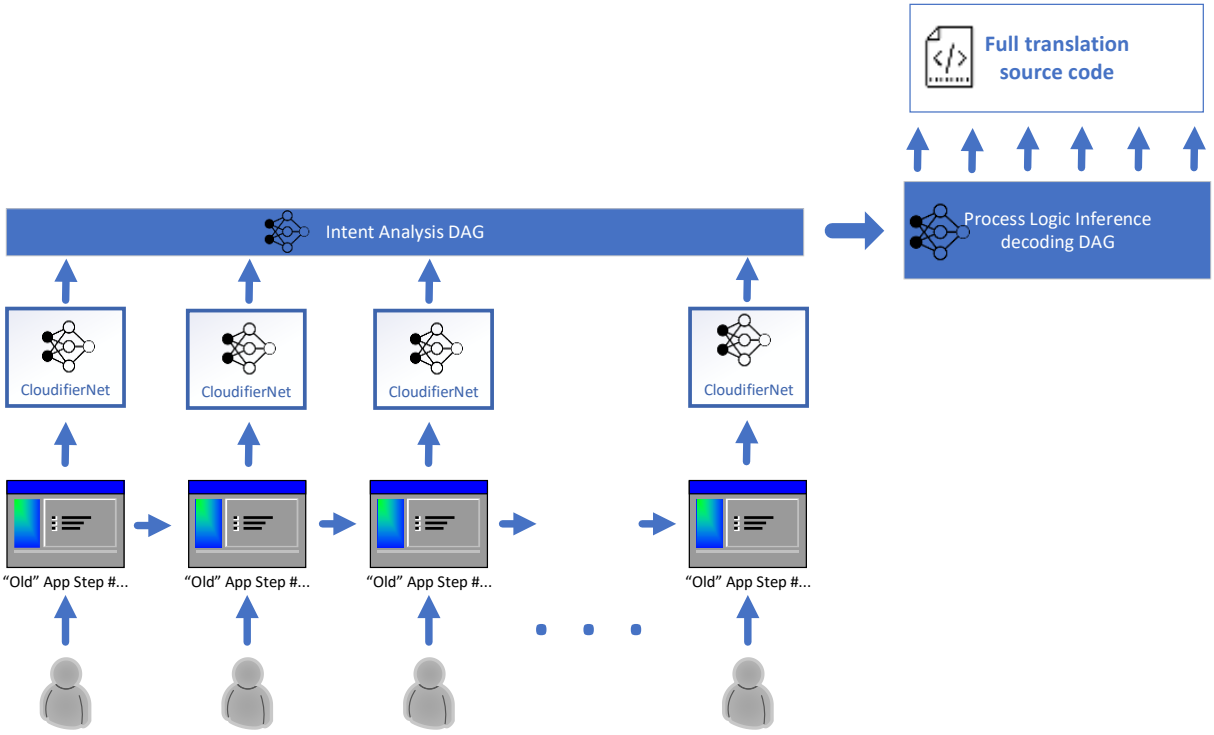


Figure 40 - Proposal for an end-to-end trainable architecture for process logic inference and source code generation for both UX and application logic flow

This approach presented in Figure 40 will allow the achievement of the following objectives:

- ✓ Inference of the overall needs addressed by the analyzed application including
  - Individual process description up to a certain detail level
  - External data-source requirements inference
  - Actual feature/process purpose-inference
- ✓ Inference of each user-interface screen
  - "In-screen processes" that will allow to understand what transitions of the process and the user-interface refer to the same UX stage (UI screen)
  - "Screen transitions" that will allow the connections of the various stages of the UX process flow and the underlying user interface screens
- ✓ User experience flow inference will be achieved by putting together the inferred information from each individual capture user experience stage

- ✓ Source code generation for the business flow that will generate, up to a certain degree, the whole business logic beside the actual user interface definition and execution scripts

6.7 Energy efficiency and environment considerations

One of the important areas of concern in the area of Deep Learning is that of energy efficiency vs training time and processing requirements. Taking into consideration that a Transformer [62] type of graph can require resources such as more than 100 GPU and more than two weeks of training summing a energy consumption and carbon (CO2) footprint similar to that of six (6) average domestic cars throughout their lifetimes. Recent work such as that of Strubell et al [65] reveal a concerning picture related to these aspects and raise the need to develop more efficient architectures and methods for graph optimization. As a result one of our areas for further research and experimentation is that related to optimizing the actual graph digital footprint by lowering the GPU floating point graph weights size [66] [67] combined with the application of distillation [68] methods.

6.8 Research on Multi Gate Units

Several directions of research have been identified:

- experiment with various biases for each individual gate in order to control the starting point of information flow

- introduce more drastical measures to impose feature selection on the gate units such as dropout on inputs might force the GateLayer to learn relevant features. Apply different dropouts for different gates

- experiment with different gating FC and different gating inputs (maybe gate dependent more than just inputs)

- Allow more direct flow of basic features and gradients such as replacing the last gate with residual connection for the complex MGUs

- add layer self-analysis (explain each gate learned features)

7 Bibliography

[1] UiPath, "https://www.uipath.com/product/platform/ai-computer-vision-for-rpa," https://www.uipath.com/product/platform/ai-computer-vision-for-rpa, 2019.

[2] E. International, "The JSON Data Interchange Format," https://www.ecma-international.org/publications/standards/Ecma, 2013.

[3] A. Agrawal, J. Gans and A. Goldfarb, "Managing the Machines," https://store.hbr.org/product/managing-the-machines-the-challenge-ahead/ROT333, 2016.

[4] J. Ghorpade, J. Parande and M. Kulkarni, "GPGPU PROCESSING IN CUDA ARCHITECTURE," *Advanced Computing: An International Journal (ACIJ); https://arxiv.org/ftp/arxiv/papers/1202/1202.4347.pdf*, vol. 3, 2012.

[5] A. Halevy, P. Norvig and F. Pereira, "The Unreasonable Effectiveness of Data," *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8-12, 2009; https://research.google/pubs/pub35179/.

[6] A. Damian and N. Tapus, "Model Architecture for Automatic Translation and Migration of Legacy Applications to Cloud Computing Environments," in *21st International Conference on Control Systems and Computer Science (CSCS)*, Bucharest; https://ieeexplore.ieee.org/document/7968616/, 2017.

[7] A. I. Damian, A. Purdila and N. Tapus, "Cloudifier virtual apps: Virtual desktop predictive analytics apps environment based on GPU computing framework," in *13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, Bucharest; https://ieeexplore.ieee.org/document/8116994, 2017.

[8] A. Damian, N. Tapus, L. Piciu and A. Purdila, "CloudifierNET - Deep Vision Models for Artificial Image Processing," *https://arxiv.org/abs/1911.01346*, 2019.

[9] T. Beltramelli, "pix2code: Generating Code from a Graphical User," *https://arxiv.org/pdf/1705.07962v2.pdf,* 2017.

[10] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, https://dl.acm.org/doi/10.1145/3065386, 2012.

[11] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions," *eprint arXiv:1409.4842; https://arxiv.org/abs/1409.4842,* 2014.

[12] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *eprint arXiv:1512.03385; https://arxiv.org/abs/1512.03385,* 2015.

[13] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *eprint arXiv:1610.02357, https://arxiv.org/abs/1610.02357,* 2016.

[14] J. Long, E. Shelhamer and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *arXiv:1411.4038; https://arxiv.org/abs/1411.4038,* 2015.

[15] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556, Computer Vision and Pattern Recognition,* no. https://arxiv.org/abs/1409.1556, 2015.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research; http://jmlr.org/papers/v15/srivastava14a.html*, vol. 15, pp. 1929-1958, 2014.

[17] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation; https://www.bioinf.jku.at/publications/older/2604.pdf,* vol. 8, pp. 1735-1780, 1997.

[18] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning*, https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf, 2010.

[19] M. Sandler, A. Z. M. Howard, A. Zhmoginov and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," arXiv:1801.04381 [cs.CV], https://arxiv.org/abs/1801.04381, 2018.

[20] S. Zagoruyko and N. Komodakis, "Wide Residual Networks," *arXiv:1605.07146; https://arxiv.org/abs/1605.07146*, 2017.

[21] K. He, X. Zhang, S. Ren and J. Sun, "Identity Mappings in Deep Residual Networks," *arXiv:1603.05027; https://arxiv.org/abs/1603.05027*, 2016.

[22] C. Szegedy, S. Ioffe, V. Vanhoucke and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," *arXiv:1602.07261, Computer Vision and Pattern Recognition; https://arxiv.org/abs/1602.07261*, 2016.

[23] M. Lin, Q. Chen and S. Ya, "Network in network," *CoRR, abs/1312.4400; https://arxiv.org/abs/1312.4400*, 2013.

[24] T. Lin, J. Hays, M. Maire, P. Perona, S. Belongie, D. Ramanan, L. Bourdev, L. Zitnick, R. Girshick and P. Dollar, "Microsoft COCO: Common Objects in Context," arxiv.org/pdf/1405.0312.pdf, 2015.

[25] T. Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, "Focal loss for dense object detection," in *IEEE international conference on computer vision*, https://arxiv.org/abs/1708.02002, 2017.

[26] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson and A. Agrawal, "Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems," *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 50); https://dl.acm.org/doi/10.1145/3123939.3124545*, 2017.

[27] NVidia, "NVIDIA Tesla P100," https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf, 2017.

[28] NVidia, "NVIDIA TESLA V100 GPU ARCHITECTURE," http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf, 2018.

[29] J. Stone, D. Gohara and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science & Engineering; https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2964860/,* vol. 12, no. 3, 2010.

[30] J. Bergstra, O. Breuleux, P. L. R. Pascanu, O. Delalleau, G. Desjardins, I. Goodfellow, A. Bergeron, Y. Bengio and P. Kaelbling, "Theano: A CPU and GPU Math Compiler in Python," in *PROC. OF THE 9th PYTHON IN SCIENCE CONF. (SCIPY 2010)*, Austin, Texas; https://hgpu.org/?p=6556, 2010.

[31] Abadi, Barham, Chen, Davis, Dean, Devin, Ghemawat, Irving, Isard, Kudlur, Levenberg, Monga, Moore, Murray, Steiner, Tucker, Vasudevan, Warden, Wicke, Yu and Zheng, "TensorFlow: A System for Large-Scale Machine Learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, Savannah, https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf, 2016.

[32] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia and R. Jozefowicz, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," Corenell University Library; arXiv:1603.04467; , https://arxiv.org/abs/1603.04467, 2016.

[33] A. Karpathy and F.-F. Li, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, https://pdfs.semanticscholar.org/84f6/f2e1ec5a2f1a1b5efe9dc65d938db1d0f0a0.pdf, 2015.

[34] O. Vinyals and Q. Le, "A Neural Conversational Model," in *ICML Deep Learning Workshop 2015*, https://arxiv.org/abs/1506.05869, 2015.

[35] J. Mao, W. Xu, Y. Yang, J. Wang and A. L. Yuille, "Explain images with multimodal recurrent neural networks.," arXiv preprint arXiv:1410.1090, https://arxiv.org/abs/1410.1090, 2014.

[36] O. Vinyals, T. A. S. Bengio and Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, https://arxiv.org/abs/1411.4555, 2015.

[37] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *nternational conference on machine learning; https://arxiv.org/abs/1502.03044*, pp. 2048-2057, 2015.

[38] D. Bahdanau, K. Cho and Y. Bengio, "Neural machine translation by jointly learning to align and translate," arXiv preprint arXiv:1409.0473, https://arxiv.org/abs/1409.0473, 2014.

[39] M.-T. Luong, H. Pham and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," in *EMNLP 2015*, https://arxiv.org/abs/1508.04025, 2015.

[40] E. Schapire, "Explaining AdaBoost," http://rob.schapire.net/papers/explaining-adaboost.pdf.

[41] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, S. H. and &. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation.," arXiv preprint arXiv:1406.1078, https://arxiv.org/abs/1412.3555, 2014.

[42] R. K. Srivastava, K. Greff and J. Schmidhuber, "Highway networks," arXiv preprint arXiv:1505.00387, https://arxiv.org/abs/1505.00387, 2015.

[43] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," arXiv preprint arXiv:1502.03167, https://arxiv.org/abs/1502.03167, 2015.

[44] J. L. Ba, J. R. Kiros and G. E. Hinton, "Layer normalization," arXiv preprint arXiv:1607.06450, https://arxiv.org/abs/1607.06450, 2016.

[45] L. Y. Pratt, "Discriminability-based transfer between neural networks," in *NIPS Conference: Advances in Neural Information Processing Systems 5.*, https://dl.acm.org/doi/10.5555/645753.668046, 1993.

[46] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on (pp. 248-255)*, http://www.image-net.org/papers/imagenet_cvpr09.pdf, 2009.

[47] G. B. Huang, M. Mattar, T. Berg and E. Learned-Miller, "Labeled faces in the wild: A database forstudying face recognition in unconstrained environments," in *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, http://vis-www.cs.umass.edu/lfw/lfw.pdf, 2008.

[48] M. Everingham, L. Van Gool, C. K. Williams, J. Winn and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision; http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.157.5766&rep=rep1&type=pdf,* vol. 88(2), pp. 303-338, 2010.

[49] R. Goyal, S. E. Kahou, V. Michalski, J. Materzyńska, S. Westphal, H. Kim, V. Haenel, I. Fruend, P. Yianilos, M. Mueller-Freitag, F. Hoppe, C. Thurau, I. Bax and R. Memisevic, "The "something something" video database for learning and evaluating visual common sense," https://arxiv.org/abs/1706.04261, 2017.

[50] F. Mahdisoltani, G. Berger, W. Gharbieh, D. Fleet and R. Memisevic, "Fine-grained Video Classification and Captioning," https://arxiv.org/abs/1804.09235, 2018.

[51] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* vol. https://arxiv.org/abs/1412.6980, 2014.

[52] L. Lai, N. Suda and V. Chandra, "Deep Convolutional Neural Network Inference with Floating-point Weights and Fixed-point Activations," *https://arxiv.org/abs/1703.03073*, 2017.

[53] G. E. Hinton, S. Sabour and N. Frosst, "Dynamic Routing between Capsules," in *NIPS-2017*, https://arxiv.org/abs/1710.09829, 2017.

[54] G. E. Hinton, S. Sabour and N. Frosst, "Matrix Capsules with EM Routing," in *ICLR-2018*, https://openreview.net/pdf?id=HJWLfGWRb, 2018.

[55] P. Tschandl, C. Rosendahl and H. Kittler, "The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions," https://arxiv.org/abs/1803.10417, 2018-2019.

[56] C. V, Q. B. Nguyen and S. Pankanti, "Deep Learning Ensembles for Melanoma Recognition in Dermoscopy Images," *Journal of Research and Development; https://arxiv.org/abs/1610.04662,* 2016.

[57] A. Rezvantalab, H. Safigholi and S. Karimijeshni, "Dermatologist Level Dermoscopy Skin Cancer Classification Using Different Deep Learning Convolutional Neural Networks Algorithms," https://arxiv.org/abs/1810.10348, 2018-2019.

[58] R. S. Sutton, McAllester, Satinder and Mansou, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems; https://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf,* 2000.

[59] T. P. Lillicrap, J. J. Hunt and A. Pritzel, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971; https://arxiv.org/abs/1509.02971,* 2015.

[60] T. Haarnoja, A. Zhou, P. Abbeel and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.," *arXiv preprint arXiv:1801.01290.; https://arxiv.org/abs/1801.01290,* 2018.

[61] S. Fujimoto, H. van Hoof and D. Meger, "Addressing function approximation error in actor-critic methods.," *arXiv preprint arXiv:1802.09477; https://arxiv.org/abs/1802.09477,* 2018.

[62] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser and I. Polosukhin, "Attention Is All You Need," *Advances in neural information processing systems; https://arxiv.org/abs/1706.03762*, pp. 5998-6008, 2017.

[63] J. Devlin and e. al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv 1810.04805; https://arxiv.org/abs/1810.04805*, 2018.

[64] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei and I. Sutskever, "Language Models are Unsupervised Multitask Learners," *OpenAI blob; https://github.com/openai/gpt-2*, 2019.

[65] E. Strubell, A. Ganesh and A. McCallum, "Energy and Policy Considerations for Deep Learning in NLP," https://arxiv.org/pdf/1906.02243.pdf, 2019.

[66] S. Vogel, C. Schorn, A. Guntoro and G. Ascheid, "Efficient Stochastic Inference of Bitwise Deep Neural Networks," *Workshop on Efficient Methods for Deep Neural Networks at Neural Information Processing Systems Conference 2016, NIPS 2016, EMDNN 2016*, https://arxiv.org/abs/1611.06539; 2016.

[67] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *https://arxiv.org/abs/1703.03073*, 2017.

[68] G. Hinton, O. Vinyals and J. Dean, "Distilling the Knowledge in a Neural Network," https://arxiv.org/abs/1503.02531, 2015.

8 Anexes

8.1 Terms

8.2 Main model architectures and algorithms