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)

2020

Contents

1 Thesis summarization and objectives 6
1.1 Introduction – artificial scene inference 6
1.1.1 Real life historical and motivational context 6
1.1.2 The main intuition of the real-life application 7
1.1.3 The overall view 11
1.1.4 Experimental work vs. real-life application: challenges & expectations 13
1.2 Thesis summarization 14
1.2.1 The proposed problem 14
1.2.2 Related work chapter summary 16
1.2.3 Architecture chapter summary 17
1.2.4 Important research results 19
1.2.5 Experiments 20
1.2.6 Main personal contributions summarization 21
1.2.7 Further work 22
2 Related work 24
2.1 State-of-the art in Deep Learning for Computer Vision 24
2.1.1 Residual learning using skip-like connections 25
2.1.2 Networks-in-networks and separable convolutions 27
2.1.3 Fully convolutional architectures 30
2.1.4 Loss behavior in dense pixel prediction 34
2.2 Optimizing computational graphs on massive computing architectures 35
2.2.1 Nuts and bolts of *Efficient* graphs 35

2.2.2 Performance through AutoML massive parallel grid search 37
2.3 State-of-the art in GPU based scientific computation 38
2.4 Image captioning and sequence decoding 41
2.4.1 Attention based encoder-decoders 42
2.5 Directed Acyclical Graph Architecture Search 45
2.5.1 Highway Networks review and comparison to our work 45
2.5.2 Network Architecture Search comparison and contrast to our work 46
3 Architectural elements of the proposed tensor directed graphs 48
3.1 Introduction of architectural elements 48
3.1.1 Domain oriented architectural elements 49
3.1.2 Domain agnostic elements 49
3.2 Parallelized shallow architecture vs deep directed acyclical tensor graphs 50
3.3 Multi Gated Units (MGU) - A new approach to tensor graph module architecture 51
3.3.1 Learnable gating mechanisms in tensor graphs 52
3.3.2 The "*MultiGatedUnit"* directed acyclical graph architecture 53
3.3.3 The "*MultiGatedUnit"* explained as a pseudo-code algorithm 59
3.3.4 Self-explain-ability capacity of *MGU*-based tensor graphs 61
3.4 Pretrained vs cold-started models as starting points 63
3.5 Dataset considerations - natural hand-drawn vs computer generated artificial data 64
3.5.1 The pros and cons of using human vs computer generated examples 64
3.5.2 Dataset experimentation and generation procedure 66
3.6 Proposed architecture of the *CloudifierNet* graph 71
3.6.1 Overall review of the directed acyclical graph 71
3.6.2 Considerations regarding parallel execution of the graph branches 71
3.6.3 Initial sections of the proposed directed acyclical graph 75
3.6.4 End sections of the proposed directed acyclical graph 78
3.6.5 Replacing simple repeated operations with Multi Gated Units 82

3.6.6 Script decoder DAGs 84
3.7 Model weights optimization process 86
3.7.1 The convolutional graph training 86
3.7.2 The recurrent graph training 88
3.7.3 Attention always pays off 88
4 Implementation, execution and evaluation 90
4.1 Experiment execution environment 90
4.2 Operationalization approach 91
4.3 CloudifierNet experimental results analysis 93
4.4 MultiGateUnit performance evaluation 94
4.4.1 Analysis and comparison of MGU sub-graph performance 94
4.4.2 Experiments with MGU self-explain-ability 97
4.4.3 Applicability of our experiments in production-grade systems 102
4.5 Research results summarization 104
5 Personal contribution areas and final conclusions 105
5.1 Multi-Gated Units 105
5.2 Convolutional graph architectures 106
5.3 Experimenting user-interface analysis 107
5.4 Real life cross-domain applications 109
5.4.1 Dermatology assistance system 109
5.4.2 Oncological gynecology 110
6 Proposed future research and development 112
6.1 Advanced hand-sketching inference 112
6.1.1 From story-boards to online applications 113
6.2 Reward-based continuous learning 114
6.3 From image to source-code generation 115
6.4 Rotation and vertical flip invariant models 117

6.5 Robotic Process Automation (RPA) experimentation 118
6.6 Process flow, user intent and user-experience logic – from UX to backend 119
6.7 Energy efficiency and environment-related considerations 121
6.8 Further research on Multi Gate Units 122
7 References 125

1 Thesis summarization and objectives

1.1 Introduction – artificial scene inference

1.1.1 Real life historical and motivational context

Constructing user interaction interfaces or *graphical user interfaces (GUIs)* is an important aspect of software development no matter the horizontal or vertical target domain nor the deployment environment. Various techniques have been developed since the early days of *GUIs* for the actual designing and implementation of user interfaces for all the platforms balancing both speed of implementation (through the *rapid application development* or *RAD* techniques) as well as quality of graphics and usability. As the systems evolved over the years with generation after generation a new range of needs related to the GUIs appeared: the need to re-think, re-design and re-construct the graphical user interfaces as well as the need to automate behavior - by "automated behavior" we refer the task of sequencing and automating a series of commands that are usual given by a human operator to an application user interface.

The aforementioned needs have been addressed for quite some time with classic techniques such as: re-using the code (if available) and re-designing the graphical user interfaces (*GUIs)*, or for the second case of behavior automation applying rule-based actions and events to the 2D coordinate *canvas* of the target user interface – that is "scripting" actions such as "click on coordinates X,Y on the display where there is supposed to be a button". Thus, the rapid application development software tools evolved in order to provide faster and more productive GUI development cycles and, related to automation subject, a new breed of software systems emerged: *robotic process automation* (or RPA) software systems that allow the users to construct automation scenarios where series of UI commands and operations are scripted and executed based on heuristic triggers.

Nonetheless, in all these cases several inherent issues have not been solved such as: (i) re-design of software applications where the source code is no longer available for various reasons; (ii) quickly iteration from graphical designer produced mock-ups to functional interfaces – i.e. from board-designed to actual GUI; (iii) semantic understanding of graphical user interface components (i.e. understanding actual functionality of visual elements) rather then "*blind*" 2D coordinates event generation; (iv) rapid semantic understanding of printed or graphical (on-screen) data forms and information conversion without the limitation of straight

object character recognition. As described in the following sections of the thesis these issues received active attention from both academia and industry and are still important research topics.

1.1.2 The main intuition of the real-life application

The main motivation behind our 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 1 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.

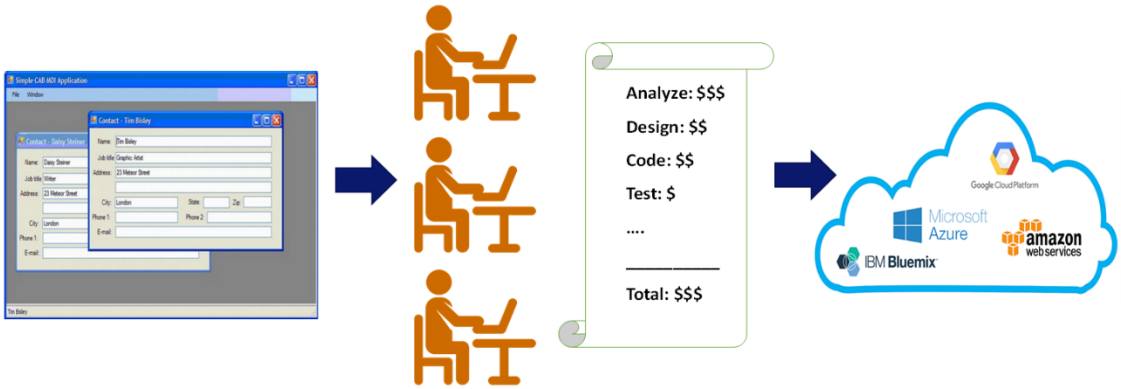

Figure 1 - The classical approach to system migration - the software development team runs a whole development cycle in order to deploy to the new target (maybe) Cloud Computing based environment

Nevertheless, following this initial real-life need identification, several other connected use-cases have been identified in the areas of automation and rapid prototyping: employ

intelligent agents that can automate user-interface human-computer interaction (Robotic Process Automation); 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 agent behavior of clicking a user interface button such as the following one. We can summarize the task in natural language as follows, for simplicity of explanation, although usually it is described in script languages: *"MOVE mouse to this absolute screen rectangle area and perform CLICK operation"* i.e command the mouse to move to a certain location and generate a "click" event on the user interface without human intervention. Now imagine that, while the absolute location remains the same, the button we need to automatically- click moves with its form due to certain circumstances as we will further explain. 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 at 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 *Figure 2* and *Figure 3* 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.

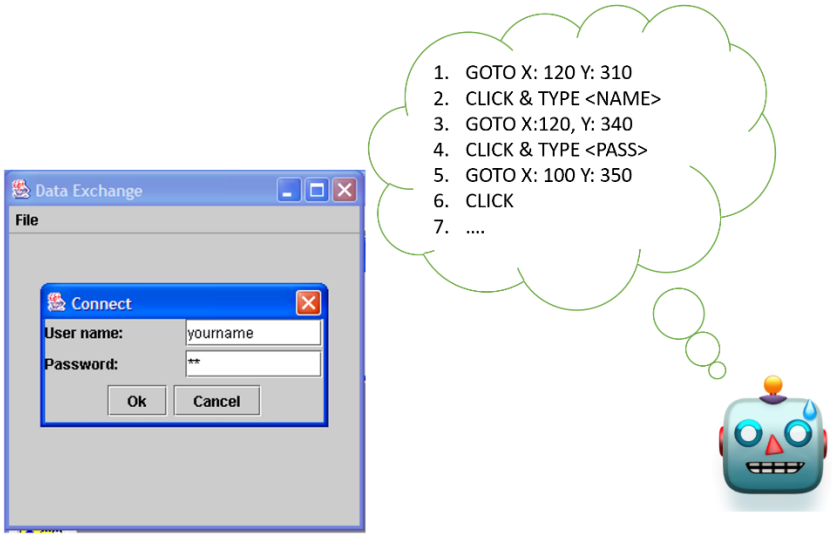

Figure 2 – simple RPA agent pre-programmed based on rules for a specific task goes to certain coordinates and completes the required information. The task is successful

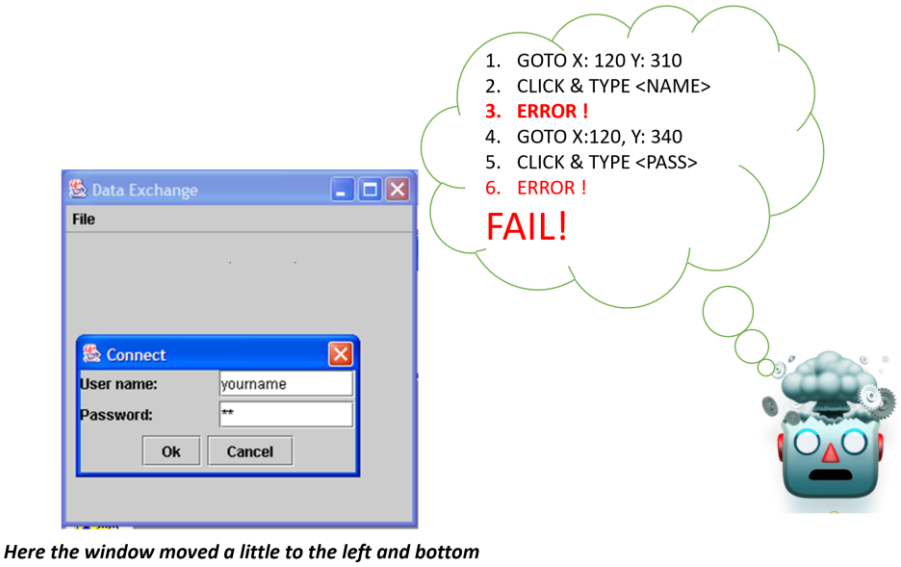

Figure 3 - RPA agent can't complete task as the environment has changed its parameters and the heuristics do not apply anymore (eq: the window has moved from the pre-configured coordinates and the "click" event generates error)

b) Migration, translation, and automated maintenance of legacy software systems such as complex client-server accounting/ERP systems or large-scale banking systems. The potential applications are ranging from the need to translate a legacy

system developed for a particular target operating system desktop graphical environment (a basic example would be develop a new version of the legacy accounting system migrating from a text-based terminal GUI to a modern graphical one) 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. In this scenario we do not have access to the any source code or already developed graphical user interface. Another potential requirement and constraint of this use case is the requirement for low-to-no-code development – i.e. develop graphical user interface without the prior knowledge of programming languages.

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. Actual example of use cases are companies that generate physical or electronic invoices or transaction reporting documents and then deliver them to their partners, who in turn have the option of manually or automatically extract structured information without the need of an in-place electronic data inter-exchange system. In all these scenarios there is an 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.

1.1.3 The overall view

The Artificial Intelligence horizontal achieved great [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]–[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].

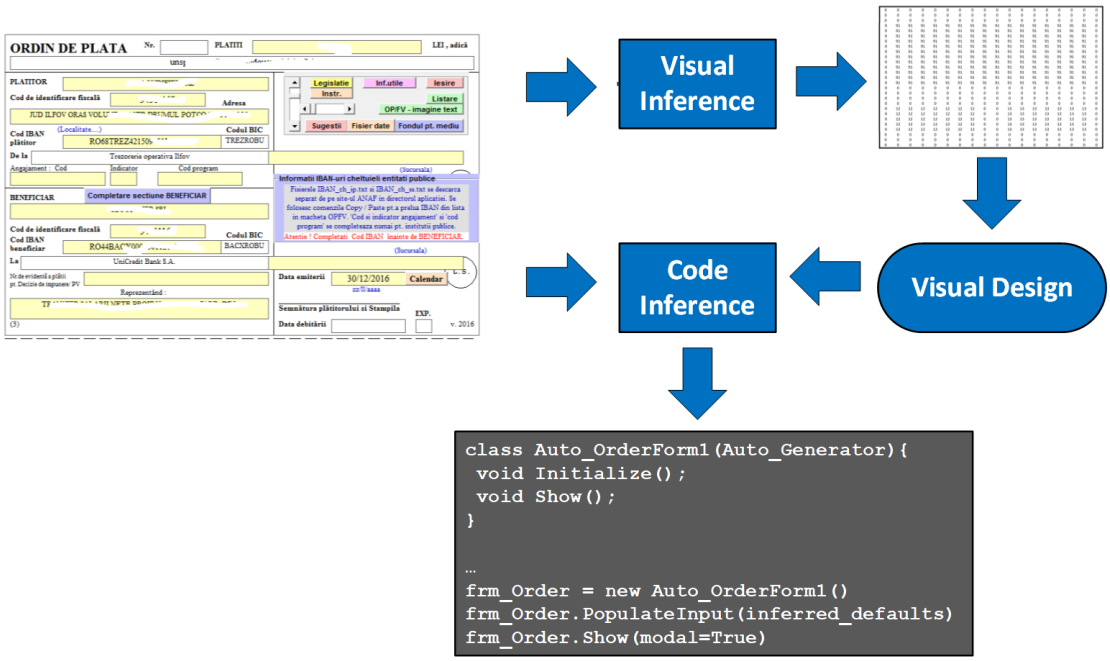

Figure 4 - From a single-form legacy desktop application to a visual design script (such as HTML, JSON,etc) up to the actual source code in a target programming language

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.

1.1.4 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 (i.e. the dataset where images/observations are not taken from real-life but rather purely computer generated)*: 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. One such example is interfaces where say, the "buttons", do not follow any visual pattern the graphical user interface buttons usually have.

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).

1.2 Thesis summarization

In this section we will briefly present the main aspects described within each of the main sections of the thesis with references to the respective individual chapters.

1.2.1 The proposed problem

The motivation behind the thesis and its research and development activities – presented with industry uses cases in ***Chapter 1.1*** - has been that of constructing an efficient and viable approach for the analysis and fine-grained recognition of application user interfaces without any kind of scriptable or programmable heuristics. Detailed elements and background

can be found in the "*Application of artificial scene inference*" chapter. The root of this motivation can be traced to various industry and academic use-cases such as the need for self-adapting robotic process automation or the need to convert legacy applications to new infrastructures such as Cloud Computing without the availability of explicit initial source code. Other real-life industry use-cases worth mentioning are recognition of completed electronic forms or electronic form templates or data extraction or data entry or fast-prototyping of modern user interfaces without the need of software design tools such as Rapid Application Development tools.

In order to have a more concrete view and better understanding we will breakdown a couple of use-cases into problem definition, proposed solution hypothesis and the actual industry application:

i. Automatic computer visual recognition of application user interfaces without access to the pre-defined rules or heuristics
   - **Problem**: Given screen captures of legacy applications user interfaces generate collection of interface artefacts, locations and purposes.
   - **Hypothesis**: Using highly optimized numerical computation tensor graphs deployed on GPU compute we detect and localize each of the user interface objects
   - **Application**: Robotic process automation benefits from this use-case by dropping the need for scripted rules and heuristics thus reducing implementation and configuration time and drastically increasing overall viability and dimension/location/aspect invariability

ii. Quick prototyping of functioning user interfaces based on old visual interfaces or simple mock-ups
   - **Problem:** Given screens of user interface mockups, legacy applications screen-shots or simple human create hand-drawn interfaces (such as the one depicted later in *Figure 24*) generate full user-interface script code
   - **Hypothesis:** Using highly optimized numerical computation tensor graphs deployed on GPU compute analyze input images and directly generate sequences of text-code (script)
   - **Application**: From-drawing-to-code quick-prototyping of user-interfaces directly by non-programmers – thus without programming language prior

knowledge - or visual designers and source code generation for a target Cloud app platform based on the old legacy user interface screens;

1.2.2 Related work chapter summary

The analysis of current state-of-the-art and how it relates to our work is presented in ***Chapter 2*** of the thesis. Multiple areas of hot research topics are strongly connected to our however two different areas stand-out as the main ones: GPU based approaches for numerical computing efficiency optimization and advanced architectures for construction of directed acyclical graphs for deep computer vision and natural language processing.

The mainstream adoption of GPU-based devices as well as the recent fast advancement and the new family additions (such as the more recent TPUs - Tensor Processing Units) has enabled both the academic and the commercial community to process large quantities of data using highly parallelized numerical algorithms. Discrete convolution operations, multi-branch numerical graph computation, *Transformer*-like [10] architectures are just few examples where GPU/TPU based massive parallel numerical computation shine. In our work, a few particular areas and individual components have received special attention with regard to parallel numerical compute optimization: the new proposed approach for multi-gated sub-graphs and the overall artificial scene recognition directed acyclical graph. Although GPU kernel construction is a subject highly abstracted by tensor computation and optimization frameworks such as *Tensorflow* [11] or *PyTorch* [12], we did mention the techniques and sample approaches in our related work.

Deep learning architectures resulted from the latest researched state-of-the-art are the second main area of related work presented in ***Chapter 0*** as well as in ***Chapter 2.2*** . First of all, we inventoried the known classic approaches in Computer Vision focusing on more recent approaches based on deep directed acyclical graphs. Although the main presented deep vision architectures directly relate to various known problems and use-cases, the main focus of our experiments and thus the presented results are focused on graphical user interface scene inference (object localization and detection) as well as pixel level segmentation.

A secondary focus has been directed in the area of generative models for source code sequence generation. This has been done both in conjunction with the research on the subject

of scene segmentation and image captioning approaches as well as analyzing the modern trends in neural natural language processing.

Last but not least, within this area of research one important focus subject has been that of finding optimal and efficient architectures for tensorial directed acyclical graphs. More specifically an important aspect of the work has been that of finding innovative approaches for self-tuning of graph architectures that leverage parallel numerical computing instead of classic approaches such as exhaustive or random search of graph architecture option space (grid-search). In this particular area, papers and past experiments on self-learning gating mechanisms have received special attention in ***Chapter 2.5*** and briefly in ***Chapter 2.2.2***.

To summarize the main areas of related work we have the following:

i. Highly efficient **tensor graph evaluation based on GPU** numerical parallel compute
ii. Deep vision architectures for **artefact localization and segmentation**
iii. Generative methods for **image-to-code-sequence** translation
iv. **Graph hyperparameter self-learning** based on actual graph parameter tuning using optimization objectives

1.2.3 Architecture chapter summary

The whole research and experimentation processes has been based on a cyclic approach and step-by-step advancement starting from simple approaches into deeper and more complex solutions for the proposed objectives. This process led to two types of results as described in ***Chapter 3.1***: the domain specific results and deliverables as well as results that can be applied cross-domain and have already been applied to several real-life industrial use-cases. The detailed design process and the architectural approach with all their details are fully described ***in Chapter 3*** of the thesis. The initial steps of the research and experimentation process have been entirely based on basic “shallow” (i.e. machine learning approaches that do not have hidden graph layers) approaches of using regression models in parallel numeric computation environment in order to gauge the option of having ensembles of simple models. Basically, the initial “baseline” approach, based on simple machine learning models, has been that of employing parallel dot-product computations of input space – the RGB image representation of the user-interface – with weight matrices representing potential user interface primitives, all

this using 2D sliding-window mechanisms. More concretely, we leveraged GPU parallel numerical compute capabilities in order to compute multiple 1-vs-all hypotheses for all potential regions of the target user interface image.

Following this rather naive approach to user-interface screen-shot scene inference, we started developing more complex approaches based on directed acyclical graphs with discrete and separable convolution modules – this architectural approach being fully presented in ***Chapter 3.6***. In this process of refining architectures our objective has been that of constructing a well optimized and balanced class of deep directed acyclical graphs in terms of performance vs costs and energy consumption. Thus, we researched the potential benefit of constructing self-learnable sub-graphs based on learnable gating mechanisms that are explained and formalized in ***Chapter 3.3***. This innovative approach would allow the directed acyclical graph to adapt its own operation and data-flow structure, based on the overall optimization objective. Several important aspects regarding the convolutional module based directed acyclical graph architecture are presented further in ***Chapter 3*** such as the analysis of transfer learning options we explored in ***Chapter 3.4***, comparison between training with the artificial data versus the hand-drawn human generated images in ***Chapter 3.5*** and details of optimization process in ***Chapter 3.7***.

In order to synthetize the main principles used in our experimental architecture design we can summarize the following:

i. A set of **basic goals and principles** has been laid-out from the verry beginning such as using efficient parallel numerical computation mass-market infrastructures based on GPU
ii. Experiments have been **started from low-level, low-complexity** approaches to setup clear baselines
iii. The **complexity has been added gradually increased** and more challenges have been tackled iteratively
iv. Finally, the self-learning graph architecture based on self-gating mechanisms allowed the generation of our DAG family

1.2.4 Notable research results

There are three different areas where our work has produced validated results with the important mention that some of them are currently used in production grade implementations of various industries uses cases. Probably the most important and industry-relevant research result is the proposal for an innovative approach of tuning directed acyclical graphs hyperparameters based on self-learning instead of classical exhaustive or random grid searching in hyperparameter spaces, in essence the **Multi** *self*-**Gating** mechanism. This particular innovation described in detail ***Chapter 3.3***, currently in use in a series of production grade systems in the area of predictive analytics - demand forecasting and event prediction – thus demonstrating cross-industry application. The domain agnosticism of this proposed innovation shows great promise and further work is planned in this area in the next period. Moreover, in this area of research particular focus has been directed towards green computation – i.e. **energy efficiency** – as well as **experiment explain-ability** - one of the hottest topics in today machine learning landscape.

The second general area of research where our results have been validated either through published research and experimentation and/or industrial experimental development is that of artificial image-scene (e.g.: user interface screen-shot) analysis and source code (script) generation. In this particular area we have developed – the third main result - a new dataset that is publicly available for experimentation as well as methodology for potential expansion of this proposed dataset. Beside the dataset our work has been focused on finding efficient directed acyclical graph architectures that would successfully reach both the objectives of segmenting user interface pictures as well as generate basic user interface design source code (or script).

As a summarization of the most important research results, we have the following three main points:

i. Proposal for the new **Multi Gated** subgraph **Unit** that enables self-learning of graph topology structure (reconfiguration of nodes and arcs)
ii. Efficient discrete convolutional module **architecture for artificial scene inference**
iii. Publicly available artificial scene dataset – a ***ImageNet* for the user interfaces**

1.2.5 Experiments

In terms of experimentation, as it will be clearly presented throughout ***Chapter 3*** and more specifically in ***Chapter 4***, we have had two different types of experiments: those that are directly related to the main objective of the thesis as well as experiments, both purely academic or actual industry systems, related to the proposed innovations that are agnostic to our proposed goals. To quickly mention the main industry applications we need to mention that we have both deep vision related use cases in medical domain as well as predictive analytics use-cases.

For the first category of experiments - those related to user interfaces screen inferences – our work, presented in the results related ***Chapter 4.3***, has been focused on correctly assessing the performance of the proposed directed acyclical graph architectures on the collected data as well as iteratively increase the quality and quantity of *CloudifierNet* dataset. Beside measuring performance in user interface artefacts detection and localization a secondary area of experiments targeted the scaling and efficiency of computation. More concretely we experimented with different graph sizes – from a minimal size up to a 2x, 3x expansion of graph nodes and arcs - as well as different compute capabilities such as scaling from a mere 256 numerical core embedded infrastructure to over to 4000-8000 numerical core infrastructures commonly found in modern GPUs.

In this particular class of experiments and particularly related to the proposed *CloudifierNet* dataset we also created a series of experimental applications with various development tools with the main purpose of generating visual user interfaces as well as user interface design source code. More concretely, we designed both *win32* based visual applications as well as portable POSIX compliant visual applications without any complex process or business logic, all with the sole purpose of generating artificial scenes for our dataset – images (screen shots) of potential user interfaces for various cases.

Aside from all previously mentioned artificial scene inference related experiments we pursued a secondary experimentation pipeline for the proposed project-agnostic innovations presented initially in ***Chapter 3.3*** and evaluated in ***Chapter 4.4***. A particular focus in this area has been that of experimenting with various directed acyclical graph architectures with or without the usage of our proposed *Multi Gated Unit*. The main objective of these experiments has been that of testing the performance improvement on simple classification tasks (such as classifying MNIST single channel images of hand written numbers) that a classical directed

acyclical graph receives after being augmented by our *Multi Gated Unit.* The secondary objective of the experiments in this area has been that of self-explain-ability: automatic extracting of gating information and performing – if possible – gate pruning operations thus decreasing the number of matrix multiplications required to process the whole computational graph. This proposed pruning approach would allow obtaining non self-gated subgraphs from modules where the gates are fully closed or opened.

1.2.6 Main personal contributions summarization

Directly related to both the experimentation results and the underlying proposed directed acyclical graph architectures and pipelines, in ***Chapter 5*** we have the following list of main research and innovation contributions as well as practical applications deployed in real-life scenarios:

i. ***Multi Gated Unit***: Proposal of a new innovative approach for finding optimal graph structure (self-learning graph hyperparameters) directly through the minimization of the task's objective function. This particular contribution is domain-agnostic and has been tested both in *deep vision* related tasks as well **as successfully implemented in production-grade predictive analytics systems**. The two main results of this particular research contribution ca be summarized as follows:
    a. Eliminating the need for grid-search approaches that would require running up to millions of experiments on parallel compute infrastructure and thus drastically reducing the **carbon footprint** of the graph architecture search procedure
    b. Self-explain-ability of the model inner structure based on the gate activations within each Multi Gated Unit for the whole graph. Another view of viewing this finding is that of having a method for defining a unique structure (hyper-parameters) for each individual node of a deep directed acyclical graph.

ii. ***CloudifierNet dataset***: Open Source publication of a novel dataset that enables research and experimentation in the area of automatic recognition of user-

interface content. The dataset can be used in a wide range of experiments such as: training agents to recognize user interface elements in order to simulate/generate application messages, generate basic functionality source code for layout design or even construct generative approaches (such as *generative adversarial networks*) that could potentially automatize the design process beside the source code creation.

iii. ***CloudifierNet architectures***: The proposed directed acyclical graphs designs are based on incremental improvements on top of several state-of-the-art architectures while also leveraging the multi-gating modules.

iv. ***Real life applications for medical domain***: Using the proposed CloudifierNet architecture, in the past 2 years, we managed to develop and propose production grade systems in the area of oncology such as:
   a. Full auto-tuned system based on proposed CloudifierNet architecture for dermatologists and inference of severe dermatology lesions
   b. Application able to perform colposcopy (or *cervigram*) analysis for oncological gynecology in order to detect potential cervix lesions and their severity

1.2.7 Further work

In ***Chapter 6*** of the thesis a series of improvements are proposed, further work that is currently in research and experimentation phase for two different areas. One of the proposed areas of further work is specific to the task artificial image scene processing while the second general direction of improvement is related to hyper-parameters self-tuning.

In the area of artificial scene recognition, we have several clear directions of planned further work as well as potential directions of exploration, all described in ***Chapter 6.2***, ***Chapter 0*** and finally ***Chapter 6.4***. The main driver of the planned-ahead work in this general direction is strongly tied both to the real-life use-cases that we target as well as the academically related objective of improving the *CloudifierNet* dataset.

In the area of hyper-parameter self-tuning our goals, described in ***Chapter 6.8***, are divided between further improving the energy efficiency gains as well as advancing the self-explain-ability and self-pruning of the proposed *Multi Gated Unit*. We strongly believe that a more reliable and universally applicable *Multi Gate Unit* will encourage both academic and industry research teams to employ it rather than apply classic hyper-parameter grid-search space exploration. No matter the target domain where the self-tuning of hyperparameters is applied, the carbon print reduction of the optimal graph architecture search will be dramatically reduced when generating a single self-learnable architecture rather than performing searching operations in hyperparameter space. Although a single classical deep graph optimization process might have a much lower energy cost than that of a self-tuned one – due to the multitude of added gating mechanisms computations in the Multi Gated Unit – using the later one involves a ***single*** full graph optimization iteration.

In order to clearly summarize our proposed further work, we can enumerate the following objectives based on the proposed Chapter 6 of the thesis:

i. Advanced user-interface artificial scene recognition including inference from videos and inference of process behavior;
ii. Further improvement iterations of the *CloudifierNet* dataset and the creation of a academia and industrial community around this dataset;
iii. Finally, the improvement of the *Multi Gated Unit* both from the perspective of successful application in various use-cases as well as advancing the self-explain-ability and computation reduction using self-pruning mechanisms.

2 Related work

The following section will describe in detail the main areas of research that our work relates to, composed of deep computer vision techniques, GPU based numerical compute optimization, model architecture search and various domain-agnostic neural graph components.

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 [13] 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 [14] and ResNet [15] as well as several other architectures such as separable convolution network proposed by Xception [16], and also the fully convolutional model for end-to-end image segmentation FCN [17] based on the straight sequential VGGNet [18] 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 [15] that allow deeper neural graphs without fearing the main concern of inefficient gradient propagation during the back-propagation based training.
- the network-in-network [14] 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 [17] 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 [19].

2.1.1 Residual learning using skip-like connections

Another basic ingredient of our models is the skip-like connection – either residual or not. 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 [20] 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* [21] 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 [15] 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* [14] 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 [22]. 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.

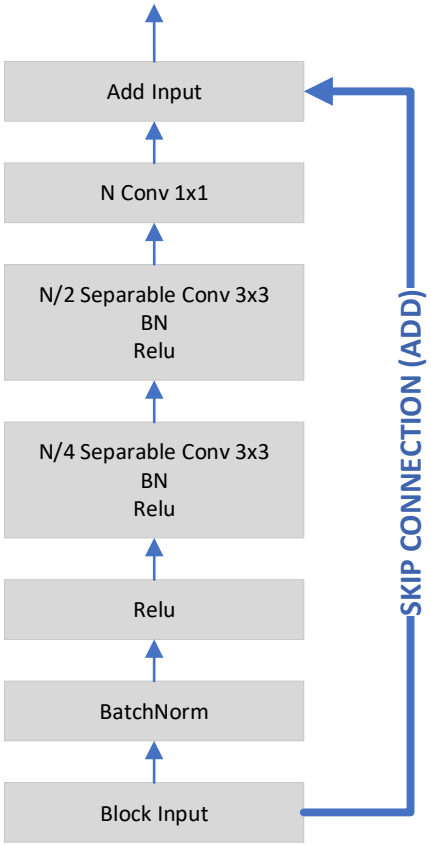

Figure 5 – Residual skip connection in a CNN

Finally, we have to mention the recent work of Zagoruyko et al [23] 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 [24] or network-in-network-residual [25] 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 [26] and later by Szegedy et al [14]. 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)*.

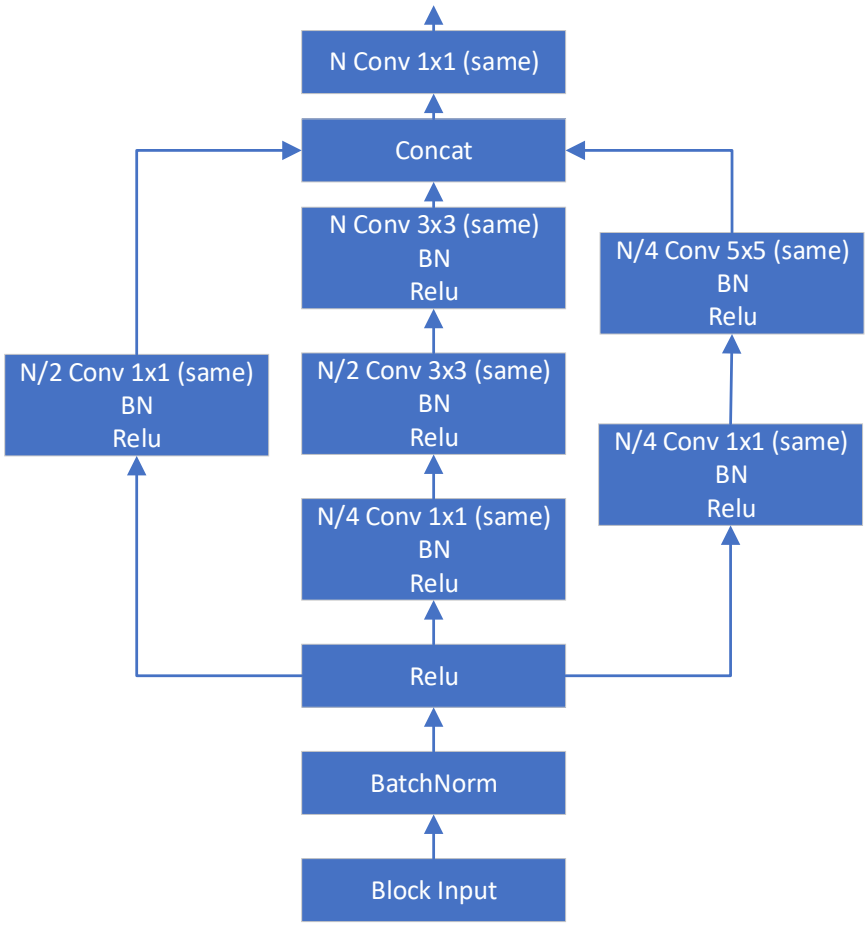

Figure 6 – Inception module example

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 [16] 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* [25] - and the more advanced versions of the previously mentioned *ResNet* [24] 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 [17]. 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.

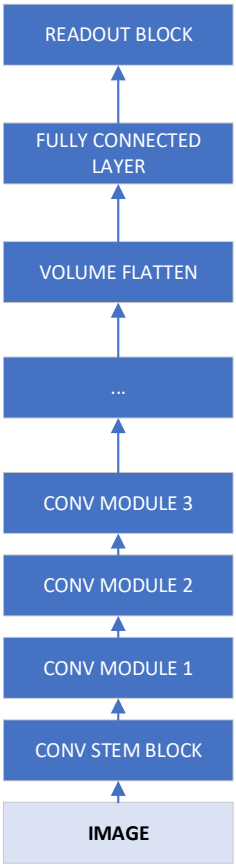

Figure 7 – Classic CNN architecture with stem, convolutional modules and fully connected modules at the top

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)

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

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 [27], 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 [28] 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 (8). 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 3.7.1 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 [29] 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 [30]. 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

We will now quickly analyze in terms of ingredients of the proposed architecture of Tan et al. The 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 [31]. 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 [31].

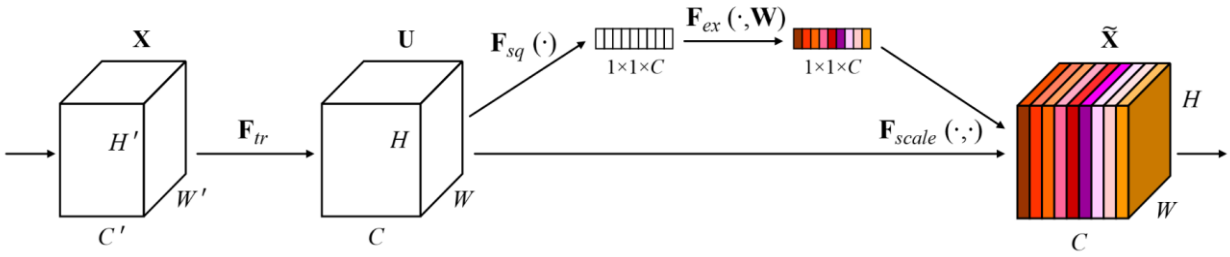

Figure 9 - Squeeze-and-excite block as original presented in the paper "Squeeze-and-Excitation Networks" by Hu et al

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 [32] 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).

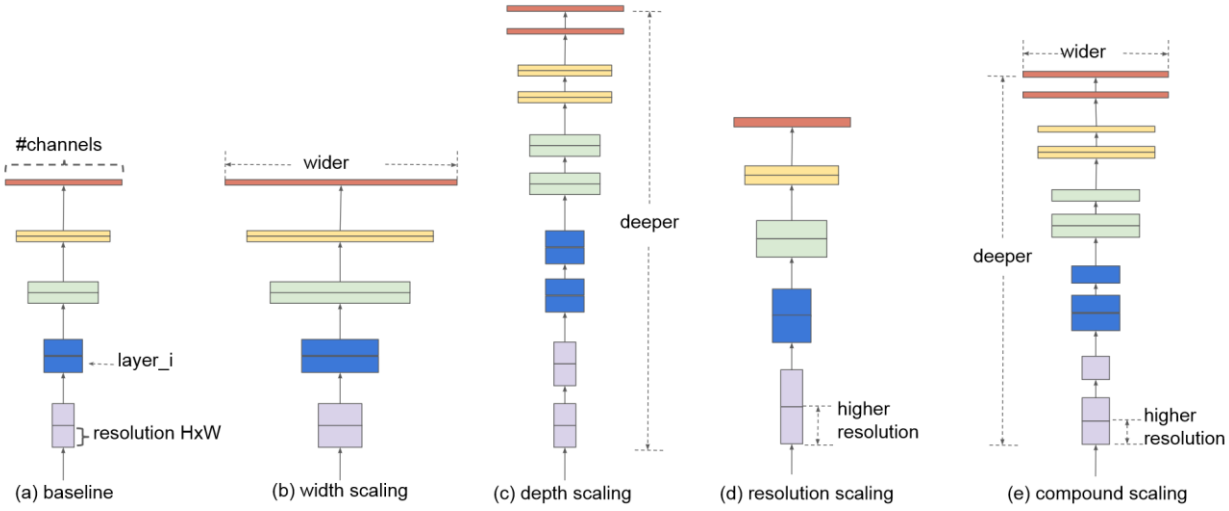

Figure 10 - Analysis of the varios model scaling approaches including the proposed compound scaling. Figure from the original "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networ" paper by Tan et al.

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.

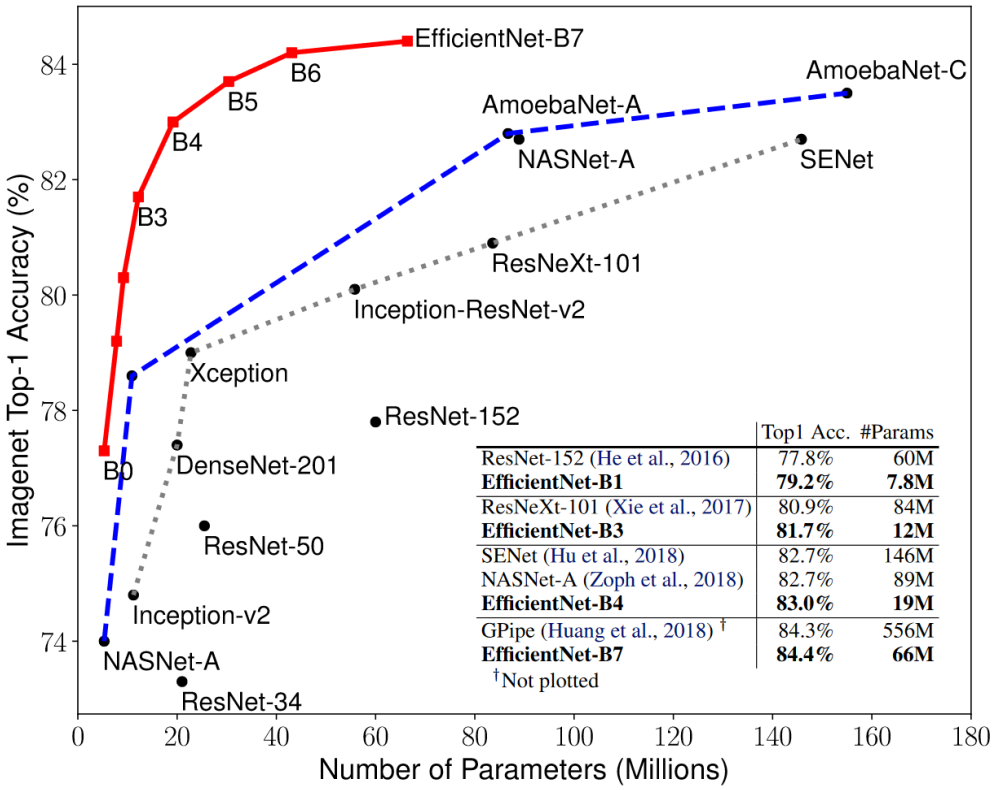

Figure 11 - Various EfficientNet instances compared by Tan et al in the paper "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". The comparison also includes various previous state-of-the-art architectures

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 the 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.

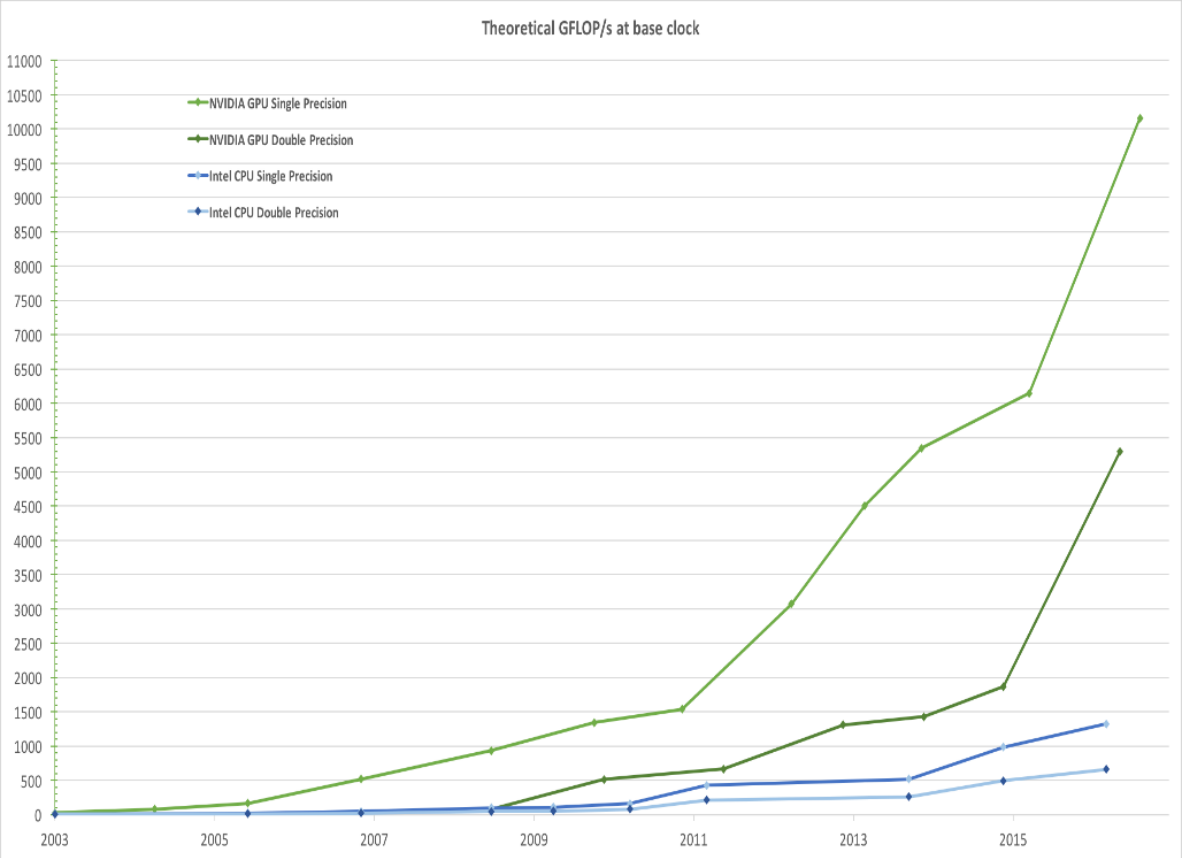

Figure 12 – Comparison between CPU and GPU processing

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 [33] [34] [35]. 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 [36] 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 [37] or TensorFlow [11]. 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 [11], 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 [38] as well as classic neural language generation approaches [39].

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 [38] and other similar papers in the area of image caption/description generation such as [40] [41] [42].

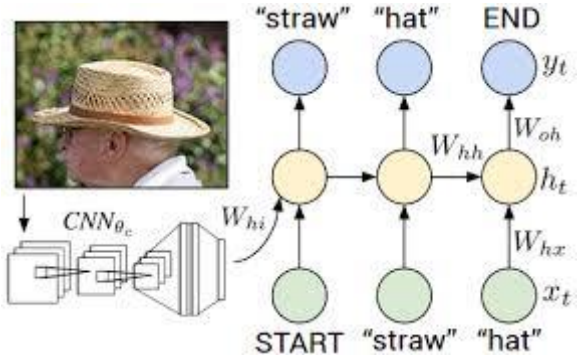

Figure 14 - From image to text - from "Deep Visual-Semantic Alignments for Generating Image Descriptions" by Karpathy et al

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 [43] 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 [44].

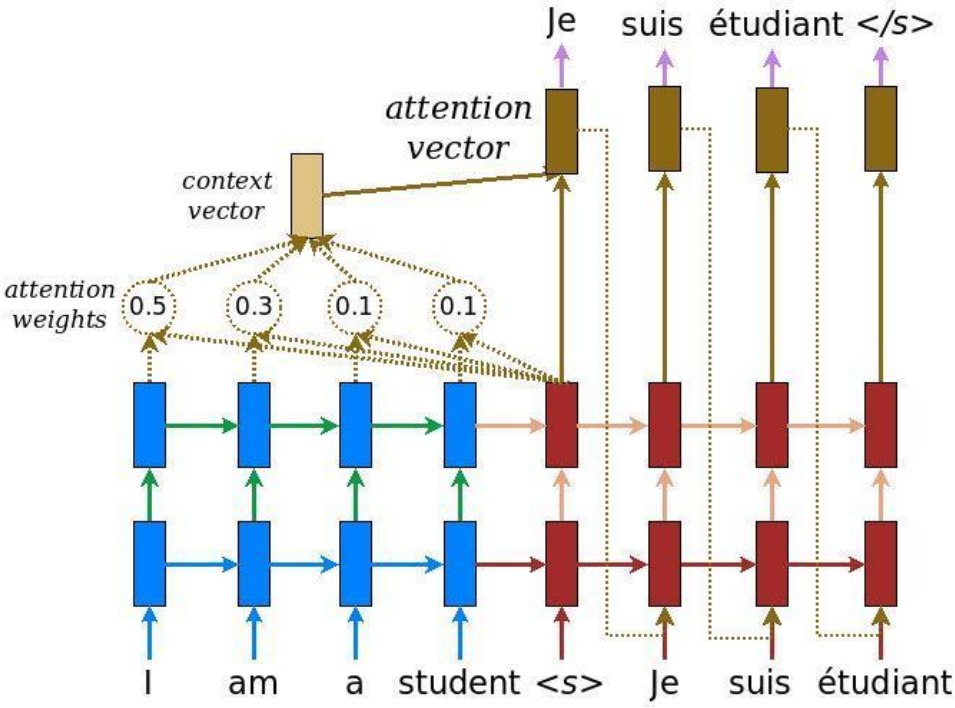

Figure 15 - Seq2Seq with attention mechanisms for end-to-end model encoding-decoding as described by Luong et al

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 3.6.6.

2.5 Directed Acyclical Graph Architecture Search

In this section we are going to briefly analyze the current main directions of research and state-of-the-art in the area of neural graph automated topology searching and we will also compare and contrast with our own approach that will be further presented in following sections.

2.5.1 Highway Networks review and comparison to our work

One of the most influencing papers for our work in the area of automatic graph topology generation, "Highway Networks" [45] by Srivastava, Greff and Schmidhuber, proposes a straight-forward layer-wise self-gating mechanism for deep neural networks that basically allows the graph to self-prune modules that are obsolete. The research team directly references the 1995 Long Short Term Memory recurrent neural network module inception paper by Hochreiter and Schmidhuber as an inspiration for the proposed self-gating mechanism, Schmidhuber being one of the former paper authors. The researchers argue that the proposed approach can alleviate the forward as well as backward information flow attenuation in very deep neural networks as well as construct information flow critical paths namely information highways.

The main intuition of the proposed Highway Network deep neural graph module can be analyzed starting from a simple case where we have a single linear layer followed by an activation function of our choosing. At this point we add a duplicate linear layer parallel to the the initial one as well as a squash activation function such as sigmoid. This additional layer basically uses the input feature information in order to compute the gate values for each individual unit of the initial layer. Finally, using the below equation we use the self-gating mechanism to "allow" initial layer information to pass forward (as well as backward) or just bypass it all together.

2.5.1.1 Comparison with our graph topology generation

While the work of Srivastava et al can be considered as the main influential idea behind the inception of our paper we went several steps forward, some of which have also been hinted in the paper – i.e. learning to route information through deep neural networks and the self-learning ability of highway networks for bypass almost entirely some layers. Based on their

experiments the authors conclude that highway networks utilize the self-gating mechanisms to pass information almost unchanged through many of the layers. Finally, our work aims to further develop more complex approaches both to information routing in deep neural graphs with the purpose of enabling self-learning architectures as well as generating automated and advanced heuristics to analyze the actual information flows and regenerate pruned graphs based on inferred critical paths.

2.5.2 Network Architecture Search comparison and contrast to our work

The area of Neural Architecture Search - or NAS in short - is a very active research and experimentation area that provided, both the academia and industry, in the last period of time with good results particularly in the area of Deep Computer Vision both for proof-of-concept as well as real-life applications. Our proposed work is directly related to the general area of deep neural graph architectures search and we will reference the work of Zoph et al [46]. However, as mentioned before, in this area we have multiple directions of research and within the multitude of approaches a particular sub-direction is dedicated to applying self-learning algorithms based on reinforcement learning for the objective intelligent search within the hyper-parameters spaces.

Intuitively the authors of [46] propose an agent that simulates a recurrent process of predicting sequential tokens representing layer parameters that, when put together, generate a functional design of a sequential neural network. In a simple example the authors consider the usual case of deep vision models where the NAS based on reinforcement learning is applied: at each step of the unrolled recurrent neural network - called controller - generates a certain type of convolutional neural network hyper-parameter such as number of filters, filter width, stride width, etc. The authors use as reinforcement reward for the agent controller a accuracy-like metric - such as accuracy, recall, etc - obtained after training the controller-generated candidate. Finally, the reward is used in conjunction with the REINFORCE algorithm in order to optimize the recurrent sequence generator controller model.

One of the inherent issues of the above approach is that each proposed candidate must be restricted to pure sequential operations without branching or even skip or residual connections that any modern graph architecture relies on. This constraint is imposed by the nature of the controller agent that is based on token-after-token and thus layer-after-layer

generation. In order to enable the controller agent to generate complex non-sequential graph structures - such as skip connection - the authors add attention-based mechanisms that allow for each timestep-generated layer to be connected to previous generated layers. The intuition is that for each new predicted token (layer) the attention mechanisms can pinpoint to a previous layer that should be added as input to the current layer input and thus create learnable skip (or residual for that matter) connections.

Finally, in terms of efficiency the authors focus mainly on reducing the time-to-optimal-solution introducing multi-agent training with multiple parallel parameter servers.

2.5.2.1 Comparison with our graph topology generation

The work of Zoph and Le is arguably one of the most influential in the area of Neural Architecture Search and a multitude of derived incremental approaches have been developed in past years. Nevertheless, the proposed approach of using reinforcement learning agents as deep neural graph designers does not drastically reduce the computational burden and the architecture search carbon footprint. Although with our proposed approach we do not aim to self-learn and bypass the need for identifying hyper-parameters such as learning rates, weight initialization strategies and such, we argue that focusing on self-learning network topologies - consisting in using Multi Gated Units in conjunction with graph pruning post-processing - can drastically reduce the computational costs and the underlying carbon footprint

3 Architectural elements of the proposed tensor directed graphs

3.1 Introduction of architectural elements

In the following chapter we will present the architectural aspects of all our experiments and research deliverables, 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 presented with a top-down approach in *Figure 17*. All the mentioned stages of research and development will be tackled from a contrasting and comparison perspective based on initial assumptions we have made, the challenges we encountered and the architectural decisions we had to make along the way.

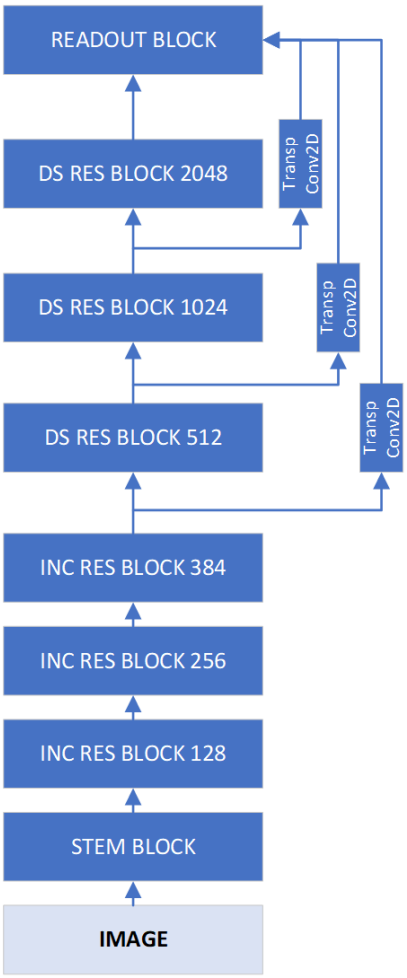

Figure 17 – Architecture of CloudifierNetV1 Directed Acyclical Graph as proposed in [8]. *Please note that the sub-graphs (INC RES and DS RES blocks) of the overall graph have own different "optional" architectures such as including or not the Multi Gated Unit proposed approach.*

Furthermore, in terms of actual main deliverables of our work we can divide the research results in two categories:

i. the domain-oriented elements, i.e. elements directly related to our deep vision tasks, such as the CloudifierNet architecture, CloudifierNet dataset
ii. domain-agnostic results, i.e. results that can be applied to non deep vision areas, consisting mainly in the Multi-Gated-Unit proposed advance in self-learned network topology mechanism

3.1.1 Domain oriented architectural elements

The two main deliverables of our research are the proposed *CloudifierNet* architecture with its multiple versions and flavors and the proposed dataset. The *CloudifierNet* architecture has multiple flavors ranging from the basic one based on depth-wise separable convolutional modules up to the advanced Multi-Gate-Unit based one that is able to learn variable topology modules within the optimization process.

The *CloudifierNet* dataset development objective has been two-fold: that of producing a dataset for our own task of artificial scene analysis as well as provide an open-source dataset for teams that are experimenting with similar tasks in areas such as user experience design or Robotic Process Automation. Thus, this deliverable can be seen as a partially domain-oriented deliverable of our research.

3.1.2 Domain agnostic elements

The main domain-agnostic element of our work is the proposed Multi-Gated-Unit graph module architecture that enables topology self-learning during optimization process without the need of basic or advanced *AutoML* techniques or *Network Architecture Search* controller-model approaches. Important to note is the fact that during the research and development of this module we have been able to employ the proposed innovation and test it in real life scenarios. One of the most important area of application – outside of *deep vision* models - has been that of predictive analytics applications. In business predictive analytics modeling we have developed several industrial cases such as demand forecasting or consumer event prediction where the application of the *Multi-Gated-Unit* allows quick generation of end-to-end trained advanced prediction models based on complex directed acyclical graph architectures without the need of *AutoML* or *NAS* approaches. Within the applications and the

pipelines developed for relevant commercial actors, in areas such as pharma retail and distribution, ground delivery services, food delivery logistics, we obtained new state-of-the-art results in comparison with other known neural architectures and approaches such as Facebook Prophet [47].

3.2 Parallelized shallow architecture vs deep directed acyclical tensor graphs

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 of the most 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 (Schapire), 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 for the deep graph architecture based on all the previously presented performance evidence. Further information regarding model comparison is presented in section 4.3.

3.3 Multi Gated Units (MGU) - A new approach to tensor graph module architecture

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. In previous related-work section, and in particular *Chapter 2.5* we already provided a first-hand view of our proposed work and the main intuitions behind it. 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 [20] and [45] 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.

A very important aspect of the MGU is its optimized parallel architecture. As we will see in the next sections all the majority of components within the MGU can be computed in parallel threads leveraging numerical compute parallelism of GPUs and TPUs.

3.3.1 Learnable gating mechanisms in tensor graphs

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 [21] 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 [48]. 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 two 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 $\sigma(x)$ 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 [49] 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.

3.3.2 The "*MultiGatedUnit*" directed acyclical graph architecture

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* in *Figure 18* - that will constrain the gate-activation module output values between *0* and *1*.

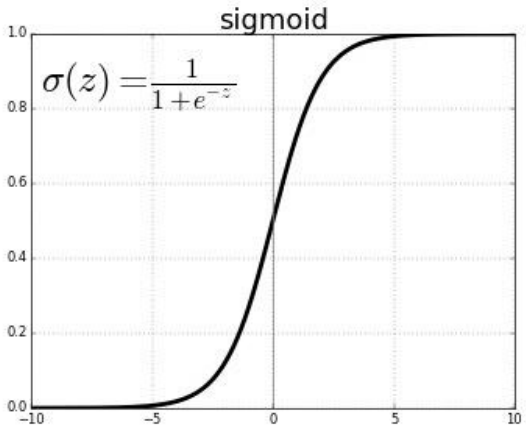

Figure 18 - The sigmoid activation function used as a gate activation-deactivation mechanism

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 from the previous equations, however we add the new gate-activation functions as well as the actual gating mechanisms. First, we have the gating operations defined by equations (15), (16), (17) and (14)

$$sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{14}$$

$$g_f(x) = sigmoid\left(l_f(x)\right) \tag{15}$$

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

$$g_{fb}(x) = sigmoid\left(l_{fb}(x)\right) \tag{17}$$

Following the gate calculation based on sigmoid we can now apply the self-gating mechanisms using a sequential line of computations as described by equations (18), (19) and (20). In this particular case we have a sequence of three successive gates based on three potential graph architectural decisions, however we can add other chained gates and their respective computations. Important to mention is that this particular subject of *dynamically* changing the *Multi Gated Unit* structure is one of our current active research topics.

$$f_{bba} = g_{bba}(x) * f_{bb} + \left(1 - g_{bba}(x)\right) * f_{ba} \quad (18)$$

$$f_{fb} = g_{fb}(x) * f_{bba} + \left(1 - g_{fb}(x)\right) * f(x) \quad (19)$$

$$f_{f} = g_{f}(x) * f_{fb} + \left(1 - g_{f}(x)\right) * l_s(x) \quad (20)$$

As mentioned, 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. Please note that the linear transformation mentioned above are not limited to simple $l(x) = W * x + b$ operations and can take the shape of much more advanced linear transformations such as discrete convolutions and other transformations. The final function f_f is actually *the highway-network* proposed by Srivastava et al [45] 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. All of the gating function being dependent only of the MGU module input can be computed in parallel as well as all of the branches that the gating functions act on. This enables a high degree of parallelism within the MGU module that leverages GPU numerical compute capabilities. 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 graph can be observed in *Figure 21*. A more refined presentation of the architecture can be observed using the *Tensorboard* [50] computational graph visualization system in *Figure 19* and its gating system based on equations (15),(16) and (17) in *Figure 20*

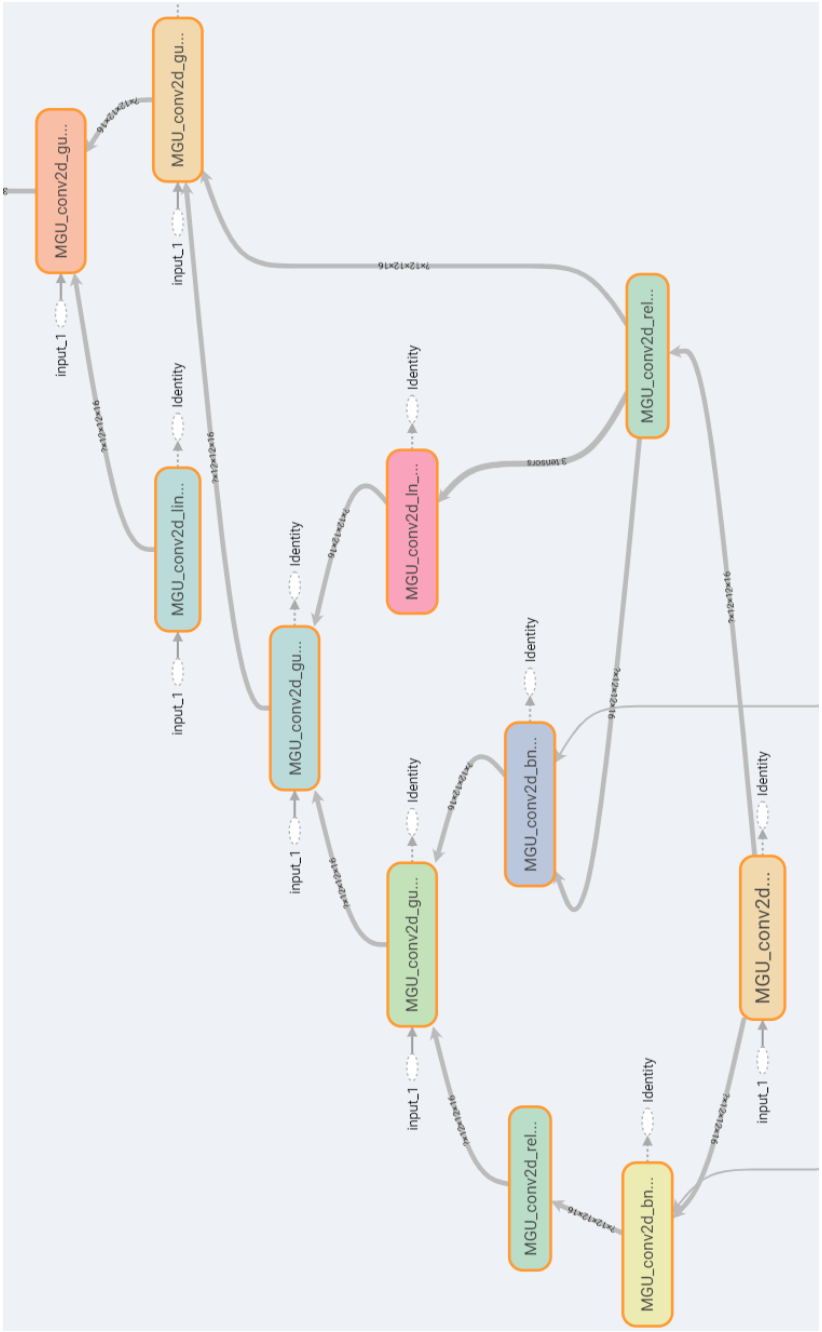

Figure 19 - MGU overall graph architecture seen by Tensorboard computational graph viewer. In this example the MGU is applied on a convolutional graph module

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.

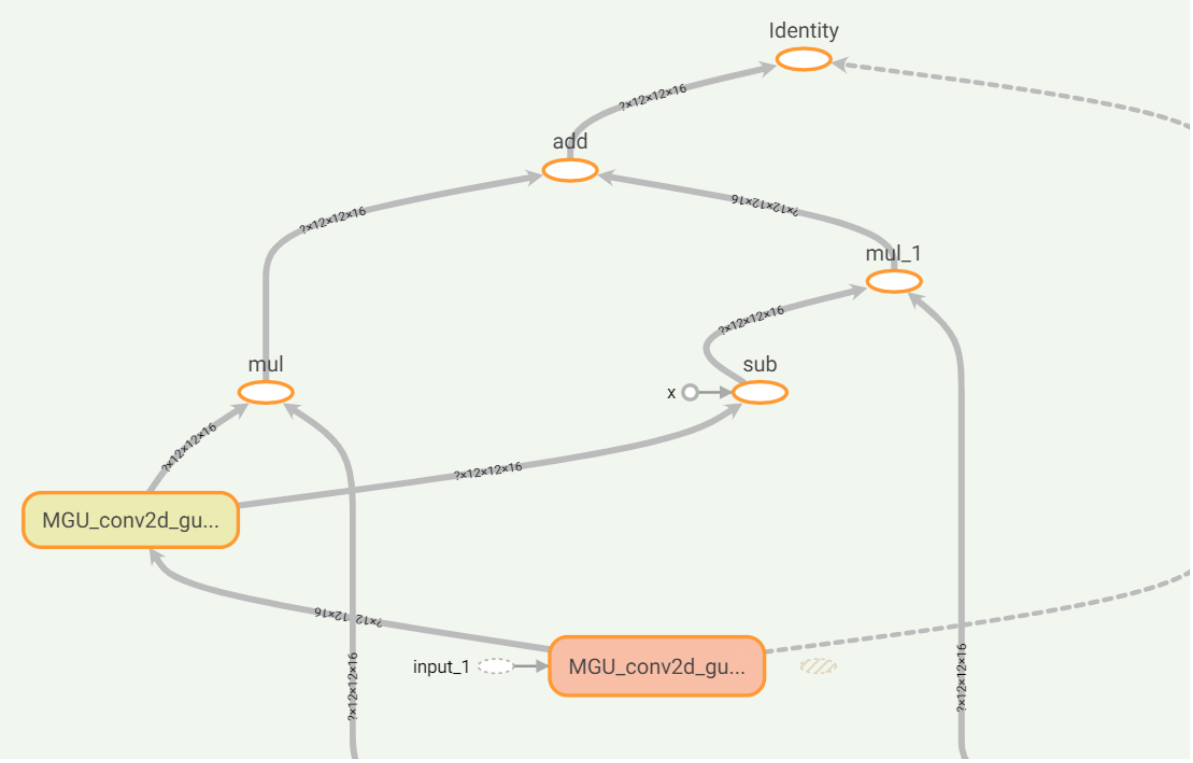

Figure 20 - The gating sub-graph of the Multi Gated Unit - in this particular case the MGU is applyed on a convolutional module

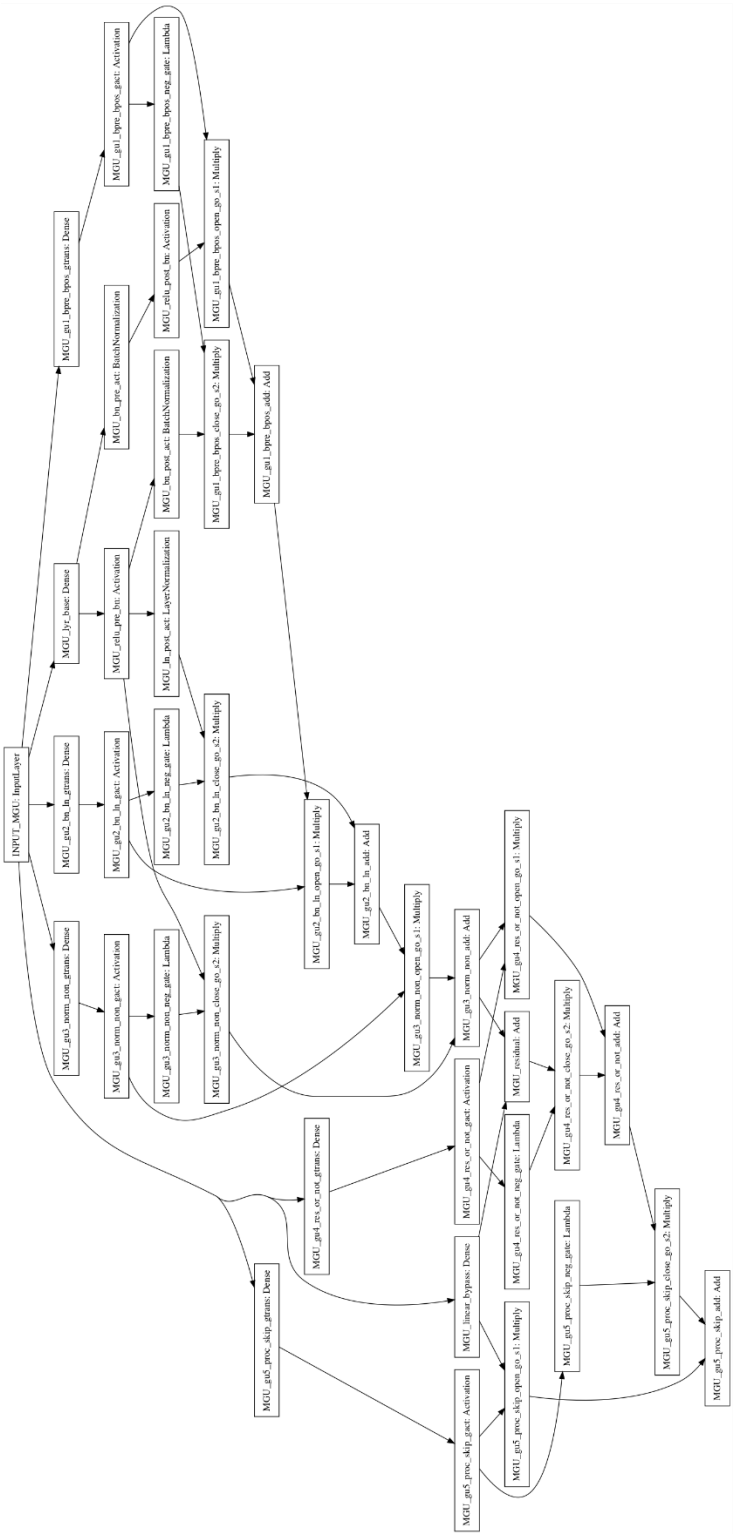

Figure 21 - The full MGU sub-graph module that is able to learn the optimal configuration of post & pre-processing of features

3.3.3 The "*MultiGatedUnit*" explained as a pseudo-code algorithm

In this section we will try to summarize the previous explanations and mathematical formalization of the MGU by presenting a pseudo-code algorithm that simulates the actual behavior of the graph. This algorithm simulates the operations within the forward propagation of the computational graph and in order to present the parallelization capabilities we added a *PARALLEL* keyword that signifies the capability to execute all similar operations using parallel numerical computational cores rather than sequentially as described below.

Algorithm MGU-PROCESS

Inputs:

Inp: Tensor with input data

Graph: Tensor graph containing MGU modules

Outputs:

Output processed tensor

Output **<-** ***Inp***

For each ***Transform*** of the ***Graph*** DO

PrevOutput **<-** ***Output***

WT, b **<-** Weights of the Transform

OP1 <- Operation on input if ***Transform*** **gate** is open

OP2 **<-** Operation on input if ***Transform*** **gate** is closed

UsePrev1 **<-** ***Op1*** uses as input the previous transformation ***PrevOutput***

UsePrev2 **<-** ***Op2*** uses as input the previous transformation ***PrevOutput***

PARALLEL ***GL*** **<-** ***WT*** **•** ***Inp*** **+** ***b***

PARALLEL ***GA*** **<-** **Sigmoid*****(GL)***

InpOp1 <- ***PrevOutput*** if ***UsePrev1*** else ***Inp***

InpOp2 <- ***PrevOutput*** if ***UsePrev2*** else ***Inp***

Output **<-** ***GA * OP1(InpOp1) + (1 – GA) * 7OP2(InpOp2)***

End For

Return ***Output***

Figure 22 - MGU pseudo-code algorithm.

A few important observations are required with regard to the MGU behavior pseudo-code algorithm presented in *Figure 22:*

i. This algorithm presents the flow of data within the MGU starting from the assumption that the weights of each transformation gate and operations have been previously optimized with back-propagation;

ii. In order to add increased gradient/information it is important that the **last** *Transform* is a simple identity-operation that will use as inputs the initial input data of the MGU and the final output of the last/previous ***Transform*** operation;

iii. For each ***Transform*** operation we can use either two operations one-vs-another with the previous output as input in order to force the graph optimization process to select the best pathway. Another strategy consists in using just one complex operation versus a identity operation (both applied on the previous output) in order to force the MGU optimization to select between using that proposed complex operation (eg. a layer normalization graph module) and the previous output bypass;

iv. In terms of computational efficiency all the gates are computed in parallel due to the fact that their inputs depend only on the initial input data of the *Multi Gated Unit* subgraph. As such, the module can compute all the gate statuses in parallel then apply them to the sequence of feature processing.

3.3.4 Self-explain-ability capacity of *MGU*-based tensor graphs

Our proposed implementation of the *MultiGatetUnit* 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).

One of the most interesting finding generated by the self-explain-ability of the MGU provided in the implementation is the capability to discover at each stage of the directed acyclical graph a different learned configuration of the gating mechanism. While in the traditional architecture the hyperparameter grid-search procedure generate such potential differentiations in our case this approach offers a continuous search space with limitless options. Regarding this particular set of experiments more information can be found in the next sections while the actual algorithm can be viewed in *Figure 23*

An important aspect of this self-explain-ability is that a potential graph pruning operation can be applied if and where the gates are significantly opened or closed. While in the case of fully opened/closed gates there is no information/processing change when pruning the gating operation from the graph for the partially opened/closed gates we have to measure the potential loss in feature quality that might lead to accuracy loss of the final results. Finally, successful pruning operations will lead to obtaining a much more efficient graph where all the gate matrix multiplications and activations are omitted from the computational graph thus greatly reducing the carbon footprint of the whole operation.

Algorithm MGU-EXPLAIN

Inputs:

Inp: Tensor with input data

Graph: Tensor graph containing MGU modules

Condition(***Gate***, ***State***): Function that generates the pathway of the graph when ***Gate*** is in ***State***

Outputs:

Critical graph path

Path <- Nil

For each ***Module***(***M***) of the ***Graph*** DO

If ***M*** is ***MGU*** then

SubGraph <- Retrieve modules from Graph up to M

ModuleInput <- ***SubGraph***(***Inp***)

For ***Gate*** in all gates of ***M***:

GateState <- ***Gate***(***ModuleInput***)

If Mean(GateState) >> 0.5 then

Path <- ***Path*** + **Condition**(***Gate***, Opened)

Else Mean(GateState) << 0.5 then

Path <- ***Path*** + **Condition**(***Gate***, Closed)

End if

End if

End For

Return ***Path***

*Figure 23 - MGU self-explain-ability algorithm. For the paticular case of gates being totally closed of totally opened we can see that the path is clear and no data is used from the "closed" branch of the gating mechanism. As a result the actual "**Path**" computed by this algorithm can be transoformed into a new pruned version of the initial and more complex computational graph*

3.4 Pretrained vs cold-started models as starting points

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 [51] in the paper "*Discriminability-based transfer between neural networks*", has been employed with great results. However, only later with the publication of ImageNet [52] 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.

3.5 Dataset considerations - natural hand-drawn vs computer generated artificial data

As already mentioned in the past sections, for the *CloudifierNet* dataset there are two radically different sources of image observations as briefly described below:

i. the computer-generated *arbitrary* user interface screens based on rapid application development frameworks and other UI/UX APIs approaches (POSIX or similar)
ii. hand-drawings depicting whole user interfaces or single UX controls (e.g. *buttons*, *scrollbars*, etc.) prepared by humans and digitized either as a 1 channel (grayscale) or 3 channel (RGB) image observation

The main objective of having a mixed distribution of images in the *CloudifierNet* dataset is that of obtaining a approach based on a deep neural directed acyclical graph that would allow automatic user-interface layout inference from simple sketches and mockups as well as automatic inference of existing legacy applications user interface screens. In the following sections we will present examples and use-cases from both categories of images contained in the *CloudifierNet* dataset that is currently available for download and direct experimentation with open source license at https://github.com/lummetry/CloudifierNet as well as within the thesis main repository at https://github.com/andreiionutdamian/phd.

3.5.1 The pros and cons of using human vs computer generated examples

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 [27] [13] [53] [54] [55] [56]. 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 (i.e. non synthetic) 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.

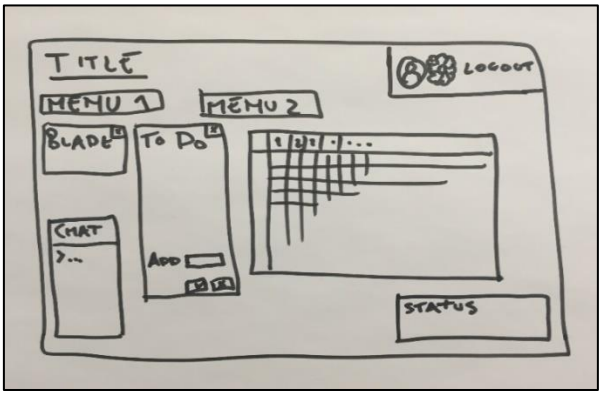

Figure 24 – Naïve user interface hand drawing example

Starting from the fact that a hand-drawn mockup can range from naïve (such as the one presented in Figure 24) 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 25, we can have a wide area of observation types in our dataset with wide range of variation.

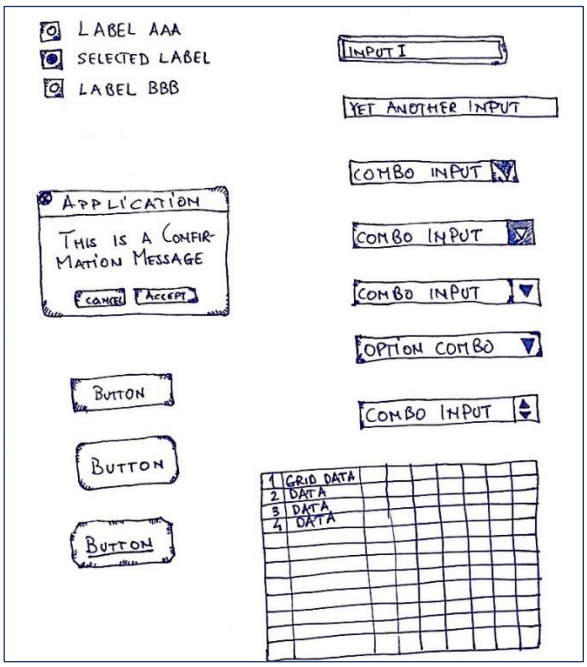

Figure 25 – Elaborated user interface hand-drawn controls. This picture is sliced in multiple primitive controls (labels, messages boxes, etc) and further augmented before added to the dataset

3.5.2 Dataset experimentation and generation procedure

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 26.

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

Figure 26 – 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 27 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 27 – 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 28 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 28 – 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 25) 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 29 based on the hand-drawn user interface mockup in Figure 24.

```
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 29 – Output of a scene inference process in the intermediate JSON-based script

3.6 Proposed architecture of the *CloudifierNet* graph

3.6.1 Considerations regarding parallel execution of the graph branches

All our proposed directed acyclical graphs architectures throughout *Chapter 3* are not using simple sequential execution but rather parallel branches in order to allow the achievement of various objectives such as: computation of parallel features based on the same input data, generating skip-concatenated and residual-summarized connections within the graph or generating various inference-maps in order to obtain multi-resolution dense outputs. While an obvious concern could be that the execution of these parallel *threads* is not optimal or even flawed in certain cases, the advancement of CUDA-based tensor computation low-level libraries – such as *CuDNN* [57]- as well as the advancement of higher-level tensor computation frameworks such as Tensorflow [58] or PyTorch [12], enables scientists to focus only on the top-level architecture design without any concerns for the actual execution at GPU or cluster level.

3.6.2 Overall review of the directed acyclical graph

The main building blocks of our proposed architecture are based on current state-of-the-art presented within section 2. The overall architecture, briefly previously presented in *Figure 17* and in more depth in multiple sections presented in *Figure 30, Figure 31, Figure 32, Figure 33*, 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 individual elements can be observed – also in multiple sections – in *Figure 34*, *Figure 35*, *Figure 36*, where we have the basic version of the CloudifierNet architecture without the addition of our proposed MGU sub-graphs.

Due to the size of the fully expanded graph, initially presented as the generic block graph in *Figure 17*, we will break it down in components and analyze each one. First, we will present the overall four-part graph in a puzzle approach based on 4 different images that concatenated together give use the clear big-picture of the overall graph. Secondly, we will *semantically* analyze the graph based on the fact that there are two main sections within its architecture and we will get a clear intuition on the purposes of each section.

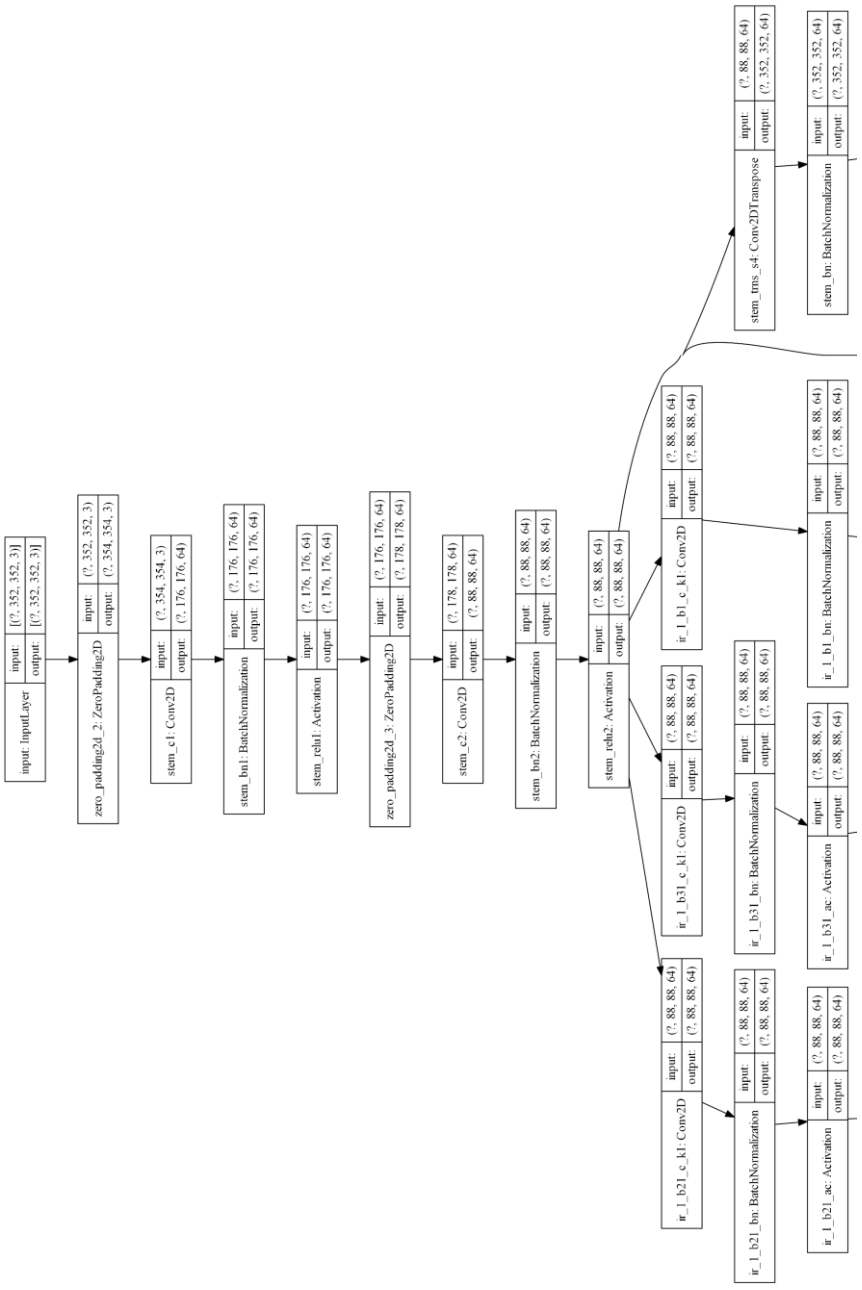

Figure 30 - Part 1 of 4 of the Cloudifier deep directed acyclical graph consisting in initial information bottlenecks and followed by the initial parallel feature computation bracnhes

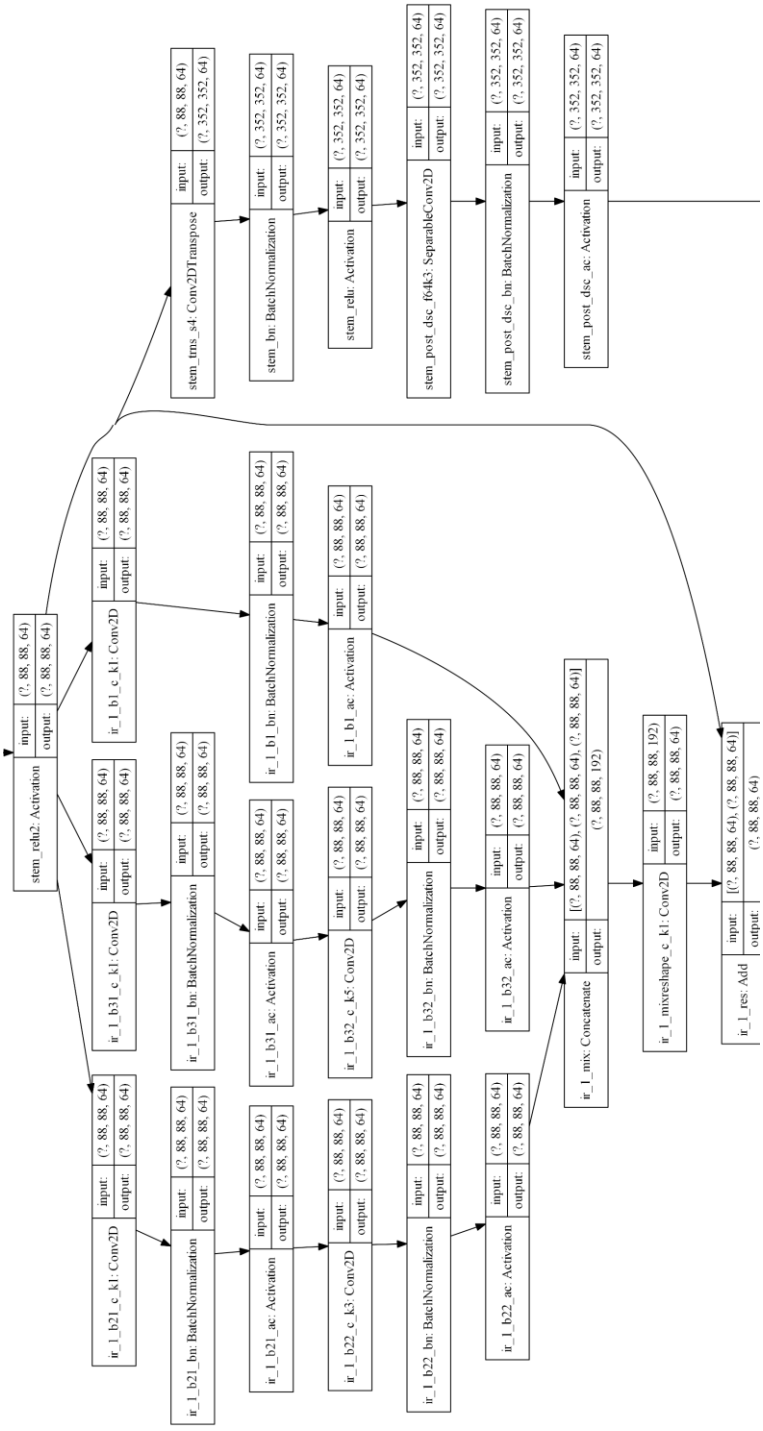

Figure 31 - Part 2 of 4 of the overall graph shows a multi-branch feature generation sub-graph as well (right side) a de-convolution operation that uses the basic features generated by the DAG

In these four slices of the overall DAG we can observe, usually on the right side – such as the case presented in *Figure 31*, a branch that basically de-convolves features in order to

create the final dense pixel-level label-map of the input image. This subject will be further explained in *Section 3.6.4*.

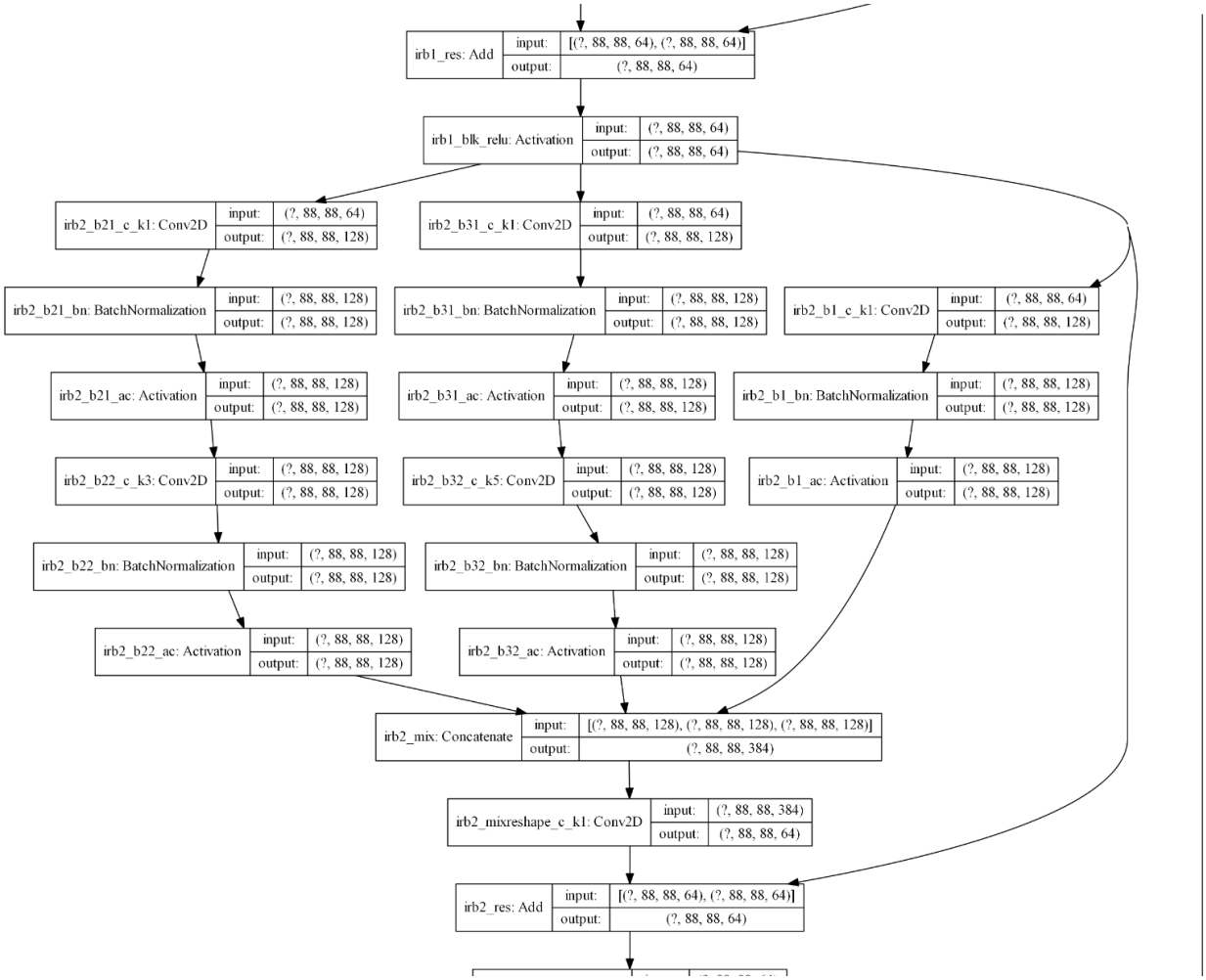

Figure 32 - Part 3 of 4 of the overall DAG - single Inception-like module. Note that the line on the right side of the figure represents the skip-connection that feeds low-level de-convolved features to the final readout layers

In the third slice image presented in *Figure 32* we have the representation of a repeated multi-branch graph module similar with the *Inception* [26] architecture. Please note that in this picture we used a simple architecture that uses classic convolutional operations followed by feature batch-oriented normalization and finally activation. In the final version of the *CloudifierNetV1* we employ the *MultiGatedUnit* that encapsulates a sub-graph with self-learned hyper-parameters as discussed in *Section 3.3*. This final important modification can be observed in *Figure 41* within *Section 3.6.5*.

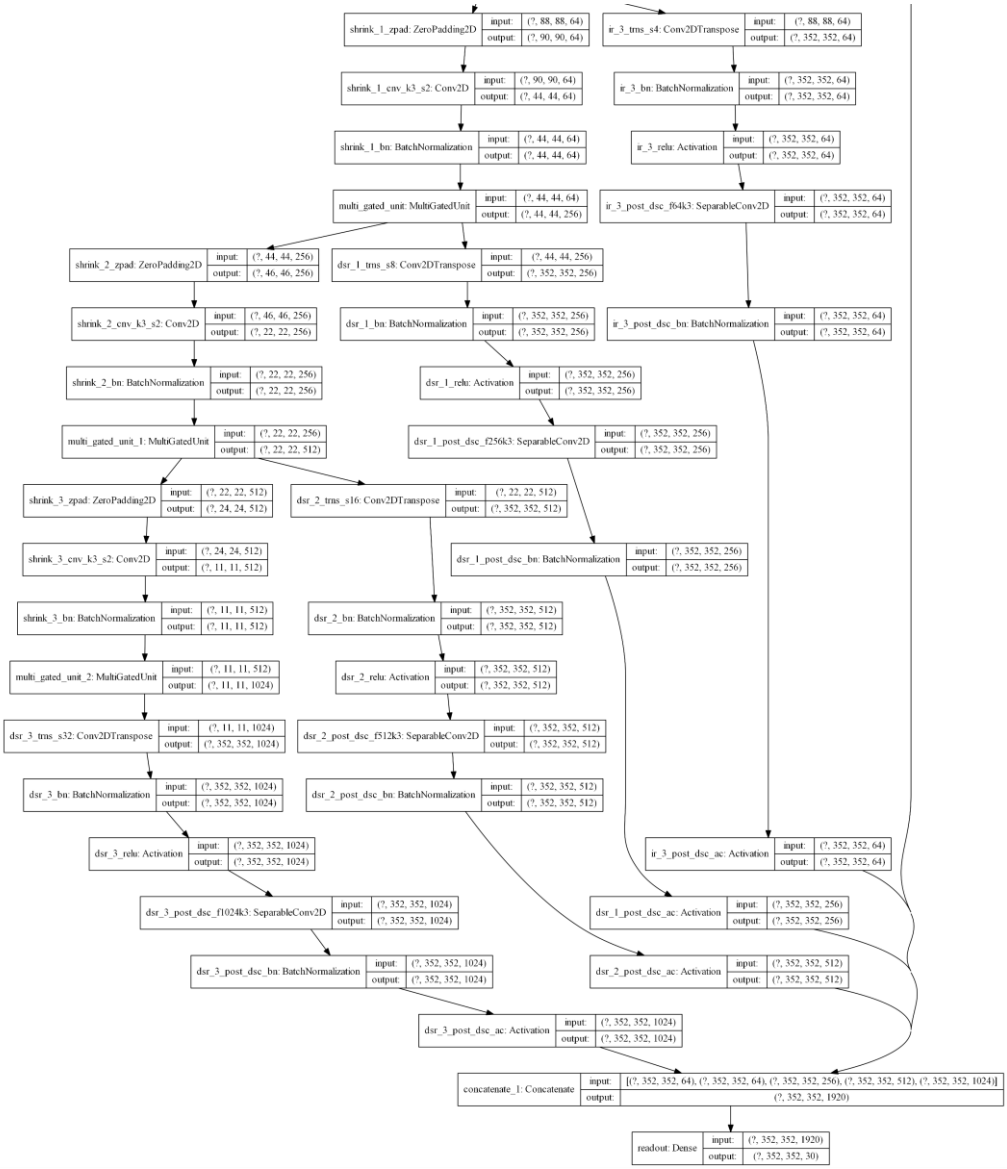

Figure 33 - Part 4 of 4 of the DAG presents the final (High) section where most of the operations have the purpose of re-constructing the input image as a dense pixel-level map of labels

3.6.3 Initial sections of the proposed directed acyclical graph

The proposed *Cloudifier* 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.

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 34* - 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).

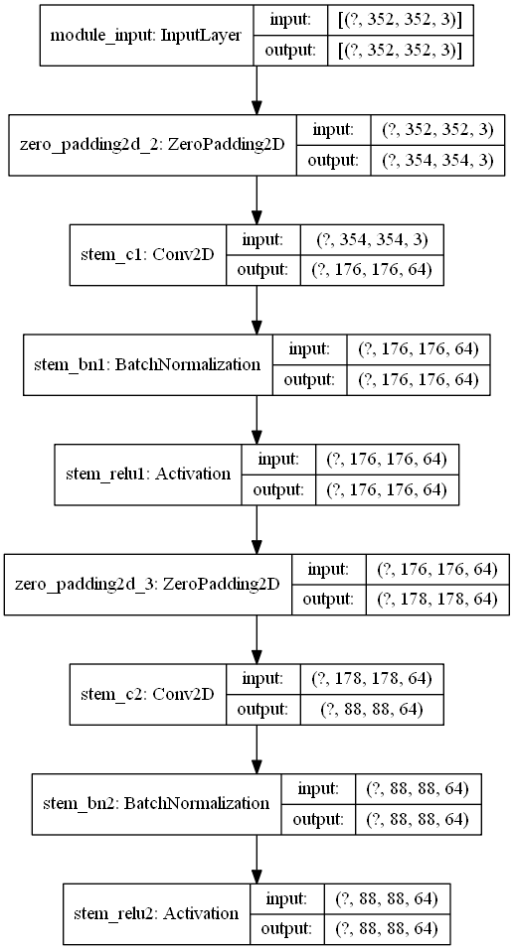

Figure 34 - The "stem" block of the CloudifierNet architecture

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 35*. 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 35 - 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*"

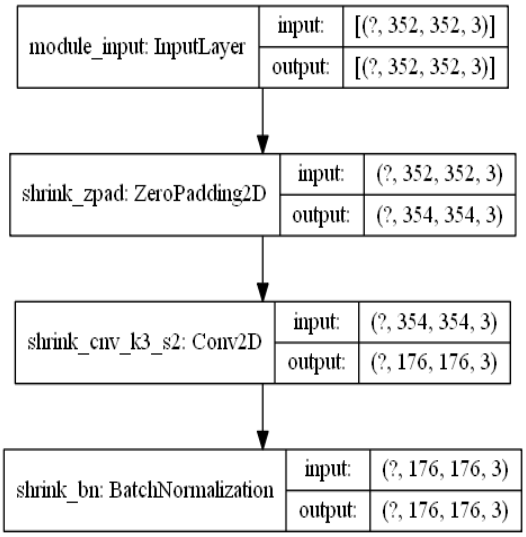

Figure 36 - ShrinkBlock graph module that has the purpose of reducing the size of the input feature maps

3.6.4 End sections of the proposed directed acyclical graph

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 37* - 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.

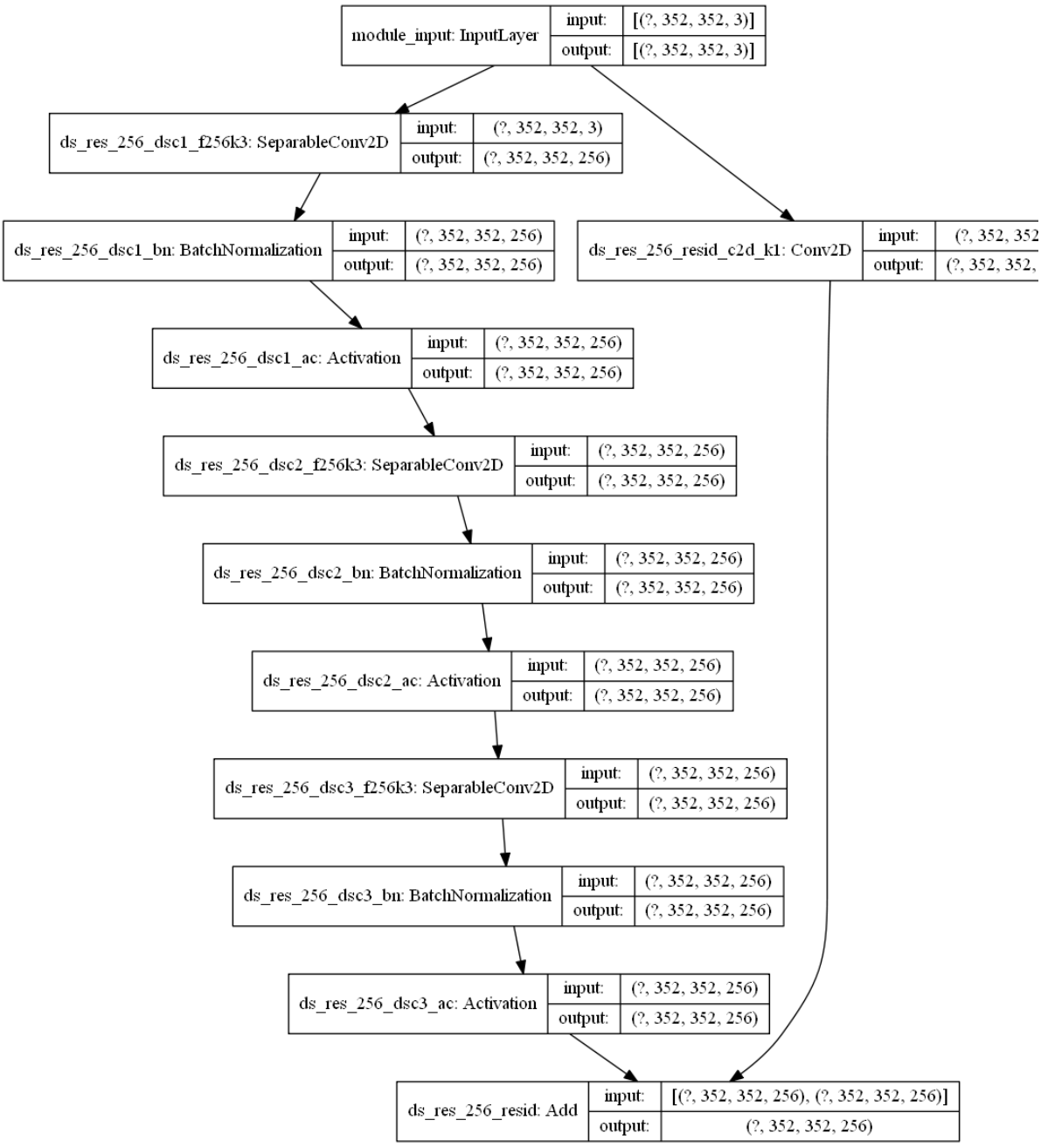

Figure 37 - The Higher section main workhorse that uses depthwise separable bi-dimensional convolutions

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 38*. 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.

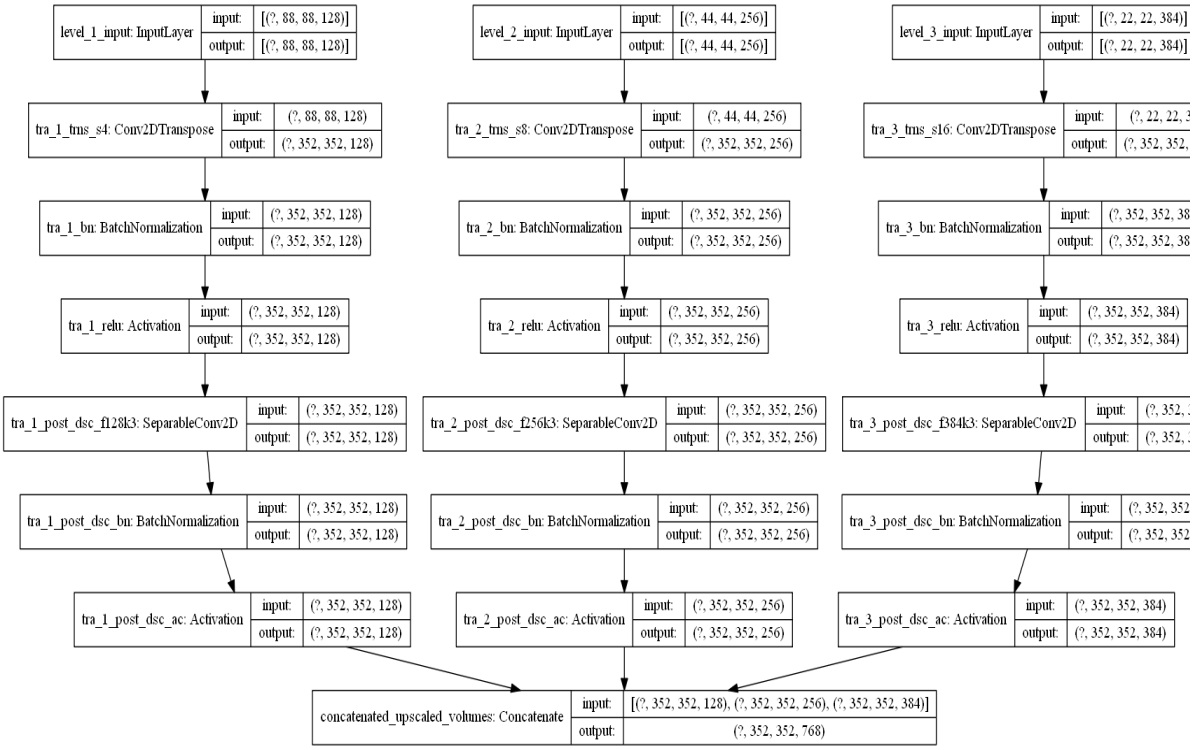

Figure 38 - The structure of the UpscaleBlock where each input comes from a different level of our graph - lower levels have lower feature map depths while higher ones have bigger feature map depths

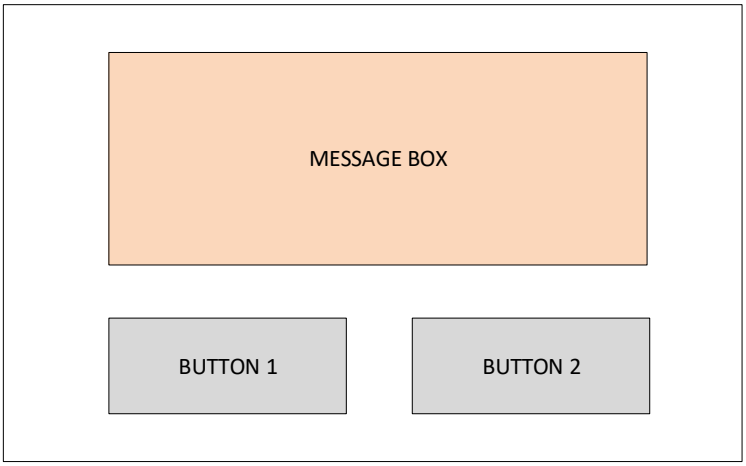

Figure 39 - Simple mockup of a visual screen where we have a message box and two buttons

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 39 can be observed in Figure 40 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 40 - Example of a dense output inference results based on a input scene with one UI message box and two UI buttons

3.6.5 Replacing simple repeated operations with Multi Gated Units

In the above sections a simple version of the graph has been presented with the purpose of understanding the basic operations within each module, however in the final version of the CloudifierNet architecture, as previously mentioned, we have employed *Multi Gated Units* for each convolutional operation – either discrete or spatially depth-wise separable. More concretely each *Conv-BatchNorm-Activation* sequence of operations – no matter the type of the Conv operation – has been directly replaced with an *MGU(Conv)* computational sub-graph with the corresponding *Conv* operation.

This replacement basically generated a unique complex architecture at the level of each individual *convolutional* computation rather than a by-section repeated architecture – considering the fact that each section of the graph contains multiple sub-graph that have basically the same repeated structure and the only actual difference is generated by the graph weights optimization process.

In order to have a clear understanding of the replacement operation we will present in below *Figure 41* the modification that are applied to the sub-graph depicted by Figure 32 with the introduction of the *MultiGatedUnit* module.

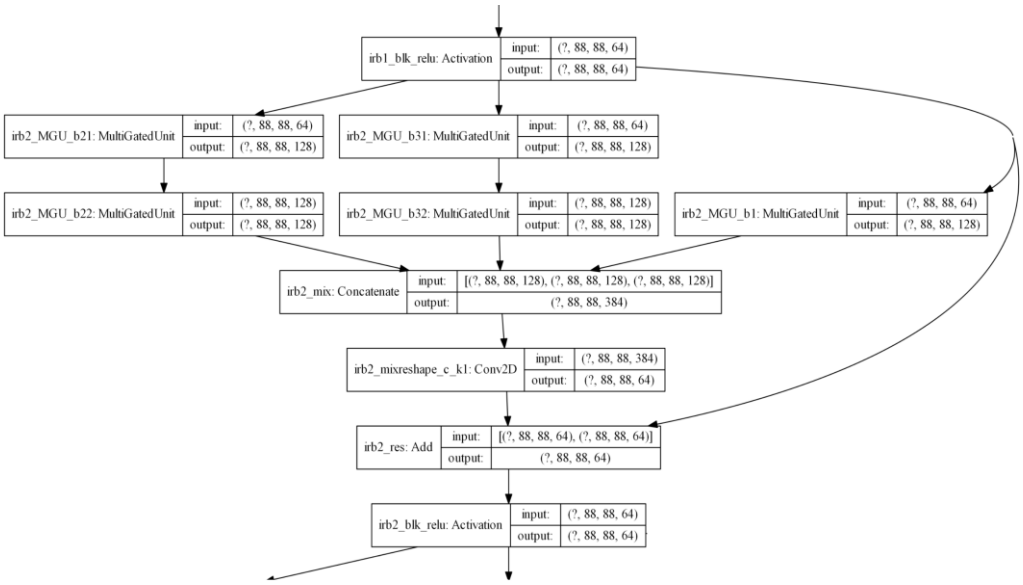

Figure 41 - Modification of a repeated multi-column sub-graph based on the replacement of classic convolution, norming and activation operations presented in Figure 32 with MultiGatedUnits

3.6.6 Script decoder DAGs

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 (Xu, et al., 2015) - 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 42* 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.

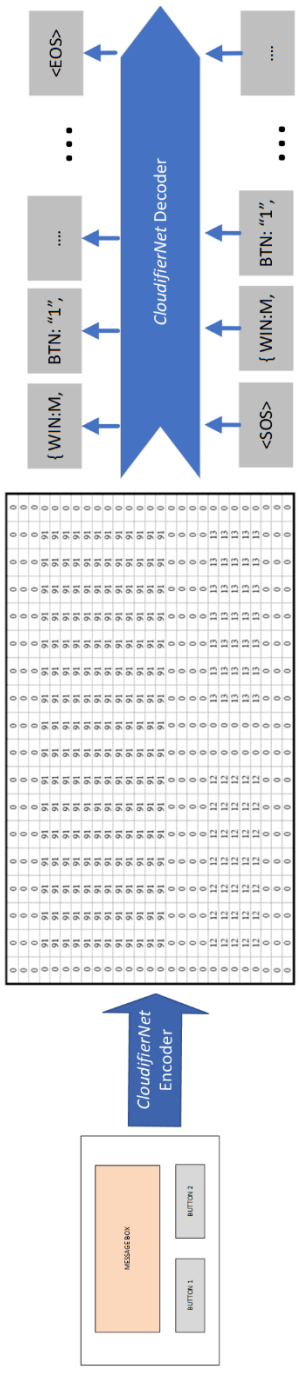

Figure 42 - Generic inference process using proposed end-to-end model. Instead of receiving a context state the decoder receives the pixel-level inference of the input image

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.

3.7 Model weights 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

3.7.1 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 [28] proposed by Lin et al. In *Figure 43* 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.

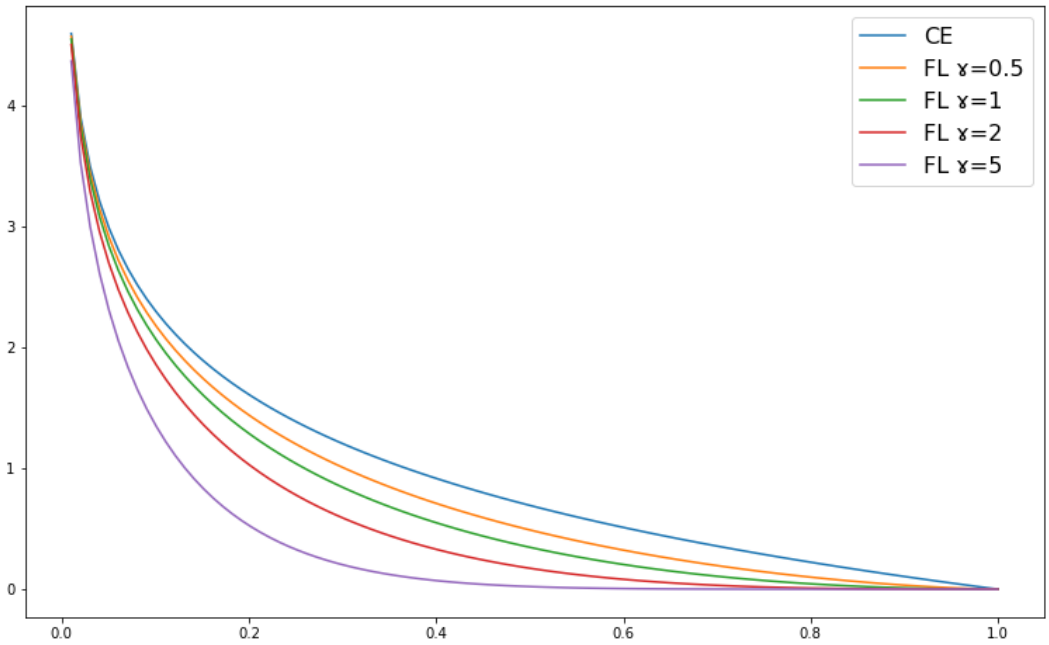

Figure 43 - Focal Loss behaviour with different values of the γ hyperparameter showing how well classified examples have lower impact on the overall loss and thus gradient. Image based on original paper by Lin et al https://arxiv.org/abs/1708.02002

We trained the proposed models end-to-end using *Adam* [59] 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.

3.7.2 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.

3.7.3 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 40* and see it in a more realistic scenario presented in *Figure 44* 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 44 - 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 Implementation, execution and evaluation

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 3.7 - 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. Nevertheless, one of our objectives, as previous mentioned, was to deploy our pipelines in various hardware environments. As it will be further presented in 4.2 this included embedded devices.

4.2 Operationalization approach

Figure 45 - Picture of our Jetson TX2 based embedded device

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.

As a result, we have developed a prototype device presented in Figure 45 that is based on the aforementioned platform.

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

i. **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

ii. **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 deeper DAGs – even if we deploy on embedded GPU compute devices such as Nvidia Jetson TX2.

iii. **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

iv. **Continuous adaptation**: One of the key elements of production grade systems based on machine learning and deep learning in particular is that of adaptability based on continuous training. One important aspect of the operationalized and *productized* environment is addition of the re-training step in the system pipeline. Intuitively, allowing the interacting users of the system to mark wrongly identified controls within a particular scene automatically generates new unseen potential training candidates for the further optimization of the DAG. Following a series of such mismatch identifications we can gather a batch of such observations and perform a local (in device) graph DAG training step based on optimization objective function gradient back-propagation even on limited compute capability devices. Another

important option of that of centralizing all the marked mismatches and generate a non-moderated candidate for *CloudifierNet* dataset extension. As such, following a review from our team we can add the new candidates to the *CloudifierNet* dataset and thus extend it via this crowd-sourcing approach.

4.3 CloudifierNet experimental 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	ArtAcc	ArtRec	NatAcc	NatRec
Cloudifier50_1	*85.1%	*91.2%	*84.3%	*88.0%
Cloudifier50_2	*88.3%	*92.1%	*86.2%	*90.7%
Cloudifier109_1	*95.2%	*97.2%	*93.1%	*95.2%
Cloudifier109_2	***98.4%**	***99.7%**	***96.1%**	***96.1%**

Figure 46 - Results of various models in different data settings - based on the results from CloudifierNet - Deep Vision Models for Artificial Image Processing

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 is actually comparable between the two approaches. Nevertheless, in terms of inference capacity the deep graph approach is far better than the initial shallow. Regarding actual results of our experiments in Figure 46, based on the published results from [8], we have divided the experiments based on the type of input dataset: either natural hand-drawn or artificial computer generated. In the above figure the *ArtAcc* and *ArtRec* represent the accuracy and the recall on the artificial test-sets, while *NatAcc* and *NatRec* represent the same performance indicators but this time for the natural hand-drawn scenes. Regarding the results our model perform in-line

with our initial hypothesis that performance is directly proportional with the model capacity for this architecture class.

Lastly, we have to mention once again that beside the main objective of inferring graphical user interface elements we also conducted experiments and developed production-grade systems in the area o oncology – all based on the *CloudifierNet* architectures.

4.4 MultiGateUnit performance evaluation

4.4.1 Analysis and comparison of MGU sub-graph performance

The secondary experimentation results we have to mention are those that present the MGU performance based on a simple dataset (MNIST) compared with the classic approach for developing the convolutional directed acyclical graphs. For this particular experiment we decided to generate a series of experiments where we used the following architectures:

Model No.	Name	Description
0	`MGU_DNS`	A two-layer MGU-based directed acyclical graph where each MGU uses a linear layer with 512 units and a rectified linear activation function
1	`MGU_CONV`	A two-layer MGU-based convolutional directed acyclical graph where the MGU uses either 16 or 32 discrete convolutional kernels of size 3x3
2	`MGU_SEP`	A two-layer MGU-based directed acyclical graph where the MGU uses either 16 or 32 kernels of size 3x3 for depth-wise separable convolutions
3	`STD_DNS_bnpre_True_bn_post_False`	A two layer graph based on linear layers with 512 units, rectified linear activation function and batch-normalization of the linearities (before applying rectified linear activation function)
4	`STD_DNS_bnpre_False_bn_post_True`	A two layer graph based on linear layers with 512 units, rectified linear activation function and batch-normalization of the activations

Model No.	Name	Description
5	`STD_DNS_bnpre_True_bn_post_False`	A two layer graph based on linear layers each with 512 units, rectified linear activation function and batch-normalization of the linearities (before applying the activation function)
6	`STD_CNV_bnpre_True_bn_post_False`	A two layer graph based on classic discrete convolutions operations with similar weights as the MGU-based graphs (two sets of kernels with size 3x3 applied 16 and 32 times on the input volumes), rectified linear activation function and batch-normalization of the linearities (before applying the activation function)
7	`STD_CNV_bnpre_False_bn_post_True`	A two layer graph based on classic discrete convolutions operations with similar weights as the MGU-based graphs (two sets of kernels with size 3x3 applied 16 and 32 times on the input volumes), rectified linear activation function and batch-normalization of the activations
8	`STD_CNV_bnpre_False_bn_post_False`	A two layer graph based on classic discrete convolutions operations with similar weights as the MGU-based graphs (two sets of kernels with size 3x3 applied 16 and 32 times on the input volumes), rectified linear activation function and no batch-normalization.
9	`STD_SEP_bnpre_True_bn_post_False`	A two layer graph based on depth-wise separable convolutions similar weights as the MGU-based convolutional graphs (two sets of kernels with size 3x3 applied 16 and 32 times on the input volumes), rectified linear activation function and batch-normalization before the non-linearities.

Model No.	Name	Description
10	**STD_SEP_bnpre_False_bn_post_True**	A two layer graph based on depth-wise separable convolutions similar weights as the MGU-based convolutional graphs (two sets of kernels with size 3x3 applied 16 and 32 times on the input volumes), rectified linear activation function and batch-normalization applied post-activation.
11	**STD_SEP_bnpre_False_bn_post_False**	A two layer graph based on depth-wise separable convolutions similar weights as the MGU-based convolutional graphs (two sets of kernels with size 3x3 applied 16 and 32 times on the input volumes), rectified linear activation function and no batch-normalization

The main objective of choosing all these 9 contra-candidates for the MGU-based graphs was to verify the hypothesis that a straight-designed graph based on classic architectures can be surpassed in terms of inference performance by the fully-gated version.

	LOSS	ACC	NAME	SZ
11	0.088498	0.9737	STD_SEP_bnpre_False_bn_post_False	6.13
10	0.073648	0.9771	STD_SEP_bnpre_False_bn_post_True	6.32
9	0.079164	0.9775	STD_SEP_bnpre_True_bn_post_False	6.32
0	0.103443	0.9779	MGU_DNS	4002.83
4	0.080561	0.9803	STD_DNS_bnpre_False_bn_post_True	673.80
3	0.071716	0.9816	STD_DNS_bnpre_True_bn_post_False	673.80
5	0.073411	0.9822	STD_DNS_bnpre_False_bn_post_False	669.71
2	0.049146	0.9838	MGU_SEP	11.62
8	0.047901	0.9855	STD_CNV_bnpre_False_bn_post_False	18.38
7	0.036386	0.9885	STD_CNV_bnpre_False_bn_post_True	18.57
6	0.034428	0.9894	STD_CNV_bnpre_True_bn_post_False	18.57
1	0.029253	**0.9909**	MGU_CNV	85.10

Figure 47 - Comparison of all 12 scenarious. The best result is obtained after multiple cross-validation cycles by the (1) MGU_CNV graph based on MGU architecture

As presented in Figure 47 the discrete convolutional MGU is outperforming all the other models verifying the hypothesis that the fully gated approach can surpass the similar architecture without any kind of learned information filtering. Basically in this results we can intuitively see the MGU_CNV model as a combination of models (6), (7) and (8) where the actual final architecture (using batch-norm before or after activation, skipping the batch-norm all-together, skipping the whole module, etc) is generated by the learned gates within the MGU module.

In terms of classic architectures, we can see that the results are following the known facts such as:

a) the convolutional architectures surpass the fully-connected ones at this particular task (computer vision) while both discrete convolutional architectures and fully connected ones might surpass the depth-wise separable convolutions in cases where the model capacity is very low. In our case the fully connected graphs have almost 60 times bigger sizes (see colum *SZ* in the figure) than the discrete convolution graphs and more than 100 times more parameters (and thus graph capacity) than the two-layer small depth-wise separable convolutional graphs.
b) Important to note is the apparent lack of performance of the MGU_DNS model – the MGU-based fully connected graph in comparison with the simple fully connected graphs. This particular aspect is due to the large size of the MGU based graph (almost 4 million parameters) with fully connected nodes and thus requiring longer training time. *Observation: for this particular experiment we limited the number of optimization epochs to 10.*
c) For the particular case of the classic discrete convolutional directed acyclical graphs we see that the best model (6) is the one where the batch-normalization is applied on the linearities rather than on the activations (7) and both surpass in all the tests the variant that has no batch-normalization whatsoever.

4.4.2 Experiments with MGU self-explain-ability

The following two sub-sections will present a short description of the self-explain ability of the proposed *MGU* sub-graph module that includes two main parts:

i. a pseudo-code presentation of the heuristics approach for understanding the results of the self-learning

ii. an actual output (a screen-shot figure) from a working and self-trained *MGU* where we can see the self-diagnosis output generated by Python code

The MultiGatedUnit explained as a pseudo-code algorithm

First we will try to summarize the previous explanations and mathematical formalization of the MGU by presenting a pseudo-code algorithm that simulates the actual behavior of the graph. This algorithm simulates the operations within the forward propagation of the computational graph and in order to present the parallelization capabilities we added a PARALLEL keyword that signifies the capability to execute all similar operations using parallel numerical computational cores rather than sequentially as described below.

Algorithm MGU-PROCESS

Inputs:

`Inp`: Tensor with input data

`Graph`: Tensor graph containing MGU modules

Outputs:

Output processed tensor

```
Output <- Inp
For each Transform  of the Graph DO
  PrevOutput <- Output
  WT, b <- Weights of the Transform
  OP1 <- Operation on input if Transform gate is open
  OP2 <- Operation on input if Transform gate is closed
  UsePrev1 <- Op1 uses as input the previous transformation PrevOutput
  UsePrev2 <- Op2 uses as input the previous transformation PrevOutput
  PARALLEL GL <- WT ● Inp + b
  PARALLEL GA <- Sigmoid(GL)
  InpOp1 <- PrevOutput if UsePrev1 else Inp
  InpOp2 <- PrevOutput if UsePrev2 else Inp
  Output <- GA * OP1(InpOp1) + (1 – GA) * OP2(InpOp2)
End For
```

```
        Return Output
```

Figure 48 - MGU pseudo-code algorithm.

A few important observations are required with regard to the *MGU* behavior pseudo-code algorithm presented in Figure 48:

a) This algorithm presents the flow of data within the MGU starting from the assumption that the weights of each transformation gate and operations have been previously optimized with back-propagation;

b) In order to add increased gradient/information it is important that the last Transform is a simple identity-operation that will use as inputs the initial input data of the MGU and the final output of the last/previous Transform operation;

c) For each Transform operation we can use either two operations one-vs-another with the previous output as input in order to force the graph optimization process to select the best pathway. Another strategy consists in using just one complex operation versus a identity operation (both applied on the previous output) in order to force the MGU optimization to select between using that proposed complex operation (eg. a layer normalization graph module) and the previous output bypass;

In terms of computational efficiency all the gates are computed in parallel due to the fact that their inputs depend only on the initial input data of the Multi Gated Unit subgraph. As such, the module can compute all the gate statuses in parallel then apply them to the sequence of feature processing.

Inspecting self-explain-ability capacity of MGU-based tensor graphs

In the following section we will use the previous benchmarking experiment of the MGU in order to *explain* the learned structure of the MGU_CNV model that achieved the best performance.

```
[MGU1][2020-08-28 14:25:14] Self-explainability of MGU layers in 6 layer graph
[MGU1][2020-08-28 14:25:14]   Layer: MGU_conv2d self-explainability analysis based on (28, 28, 1) input
[MGU1][2020-08-28 14:25:14]     Analysing 'Gate1 - BN pre or post activation'
[MGU1][2020-08-28 14:25:14]       Gate rule: `gate * BN-pre-relu  +  (1 - gate) * BN-post-relu`
[MGU1][2020-08-28 14:25:14]       Gate mean: 0.42, median: 0.44, min/max: 0.03/0.75
[MGU1][2020-08-28 14:25:14]       Gate opened for: BN-post-relu
[MGU1][2020-08-28 14:25:14]     Analysing 'Gate2 - BN or LayerNorm'
[MGU1][2020-08-28 14:25:14]       Gate rule: `gate * BN  +  (1 - gate) * LayerNorm`
[MGU1][2020-08-28 14:25:14]       Gate mean: 0.50, median: 0.50, min/max: 0.11/0.83
[MGU1][2020-08-28 14:25:14]       Gate opened for: BN
[MGU1][2020-08-28 14:25:14]     Analysing 'Gate3 - any norm or no norming at all'
[MGU1][2020-08-28 14:25:14]       Gate rule: `gate * Norming  +  (1 - gate) * No-norming`
[MGU1][2020-08-28 14:25:14]       Gate mean: 0.57, median: 0.55, min/max: 0.27/0.93
[MGU1][2020-08-28 14:25:14]       Gate opened for: Norming
[MGU1][2020-08-28 14:25:14]     Analysing 'Gate4 - direct linear bypass or processed'
[MGU1][2020-08-28 14:25:14]       Gate rule: `gate * Bypass  +  (1 - gate) * Processed`
[MGU1][2020-08-28 14:25:14]       Gate mean: 0.43, median: 0.45, min/max: 0.05/0.77
[MGU1][2020-08-28 14:25:14]       Gate opened for: Processed
[MGU1][2020-08-28 14:25:15]   Layer: MGU_conv2d_1 self-explainability analysis based on (12, 12, 16) input
[MGU1][2020-08-28 14:25:15]     Analysing 'Gate1 - BN pre or post activation'
[MGU1][2020-08-28 14:25:15]       Gate rule: `gate * BN-pre-relu  +  (1 - gate) * BN-post-relu`
[MGU1][2020-08-28 14:25:15]       Gate mean: 0.38, median: 0.30, min/max: 0.00/1.00
[MGU1][2020-08-28 14:25:15]       Gate opened for: BN-post-relu
[MGU1][2020-08-28 14:25:15]     Analysing 'Gate2 - BN or LayerNorm'
[MGU1][2020-08-28 14:25:15]       Gate rule: `gate * BN  +  (1 - gate) * LayerNorm`
[MGU1][2020-08-28 14:25:15]       Gate mean: 0.47, median: 0.46, min/max: 0.00/1.00
[MGU1][2020-08-28 14:25:15]       Gate opened for: LayerNorm
[MGU1][2020-08-28 14:25:15]     Analysing 'Gate3 - any norm or no norming at all'
[MGU1][2020-08-28 14:25:15]       Gate rule: `gate * Norming  +  (1 - gate) * No-norming`
[MGU1][2020-08-28 14:25:15]       Gate mean: 0.65, median: 0.70, min/max: 0.00/1.00
[MGU1][2020-08-28 14:25:15]       Gate opened for: Norming
[MGU1][2020-08-28 14:25:15]     Analysing 'Gate4 - direct linear bypass or processed'
[MGU1][2020-08-28 14:25:15]       Gate rule: `gate * Bypass  +  (1 - gate) * Processed`
[MGU1][2020-08-28 14:25:15]       Gate mean: 0.38, median: 0.32, min/max: 0.00/0.99
[MGU1][2020-08-28 14:25:15]       Gate opened for: Processed
```

Figure 49 - An actual output from the MGU self-explain-ability module for the best model described in the MGU benchmark

In our experiments we took the best model (namely *MGU_CNV* from our benchmark output in Figure 47) and used the MGU self-explainability approach in order to infer what is the approximative information pathway and thus the final graph architecture learned by the MGU. In Figure 49 we can see that the actual architecture is quite different from any classic approaches. In terms of actual interpretability and potentially transforming this result into a pruned and compact model with no actual gating we have the following insights:

a) for the first layer the MGU favors batch-normalization vs layer normalization while for the second module the layer normalization technique is favored vs batch-normalization the MGU finds better features in the processed information.

b) In both instances the MGU finds that batch-normalization of results post rectified linear activation yields better results that batch-normalization applied directly to linearities. This can be easily observed in Figure 50 where we can see the distribution of activations for the first analyzed module gate. The gate selects the batch-norming before ***or*** after activation – in this case as well as in

the second one not shown as a histogram here – and we can see that most activations are lower than the 0.5 threshold thus favoring *(1 – gate_activation)*.

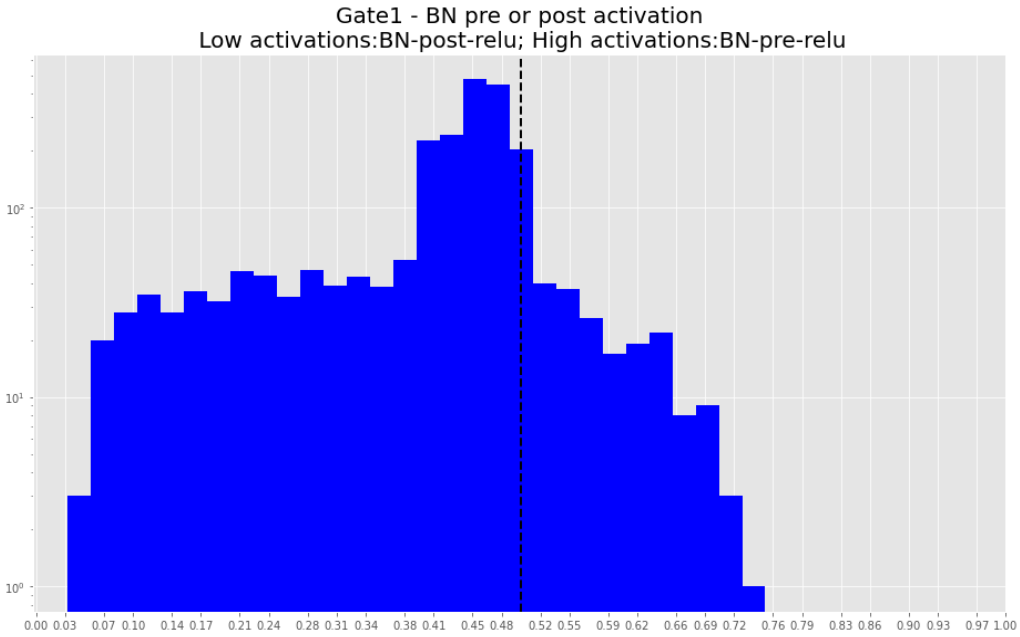

Figure 50 - Automatic gate analysis with gate activation histogram

c) Also with respect to using activated linearities vs normed activation (no matter if we are talking about layer-normalization or batch-normalization) the MGU finds that normed activations generate better features for the target of minimizing the objective function

d) Regarding the decision between direct forward bypassing the module input versus complete processing in both cases we see that the MGU finds that the best path is that with the internal processing. Beside the information available in Figure 49 mode clear information can be seen in the histogram of the activation of this specific gate that is presented in Figure 51.

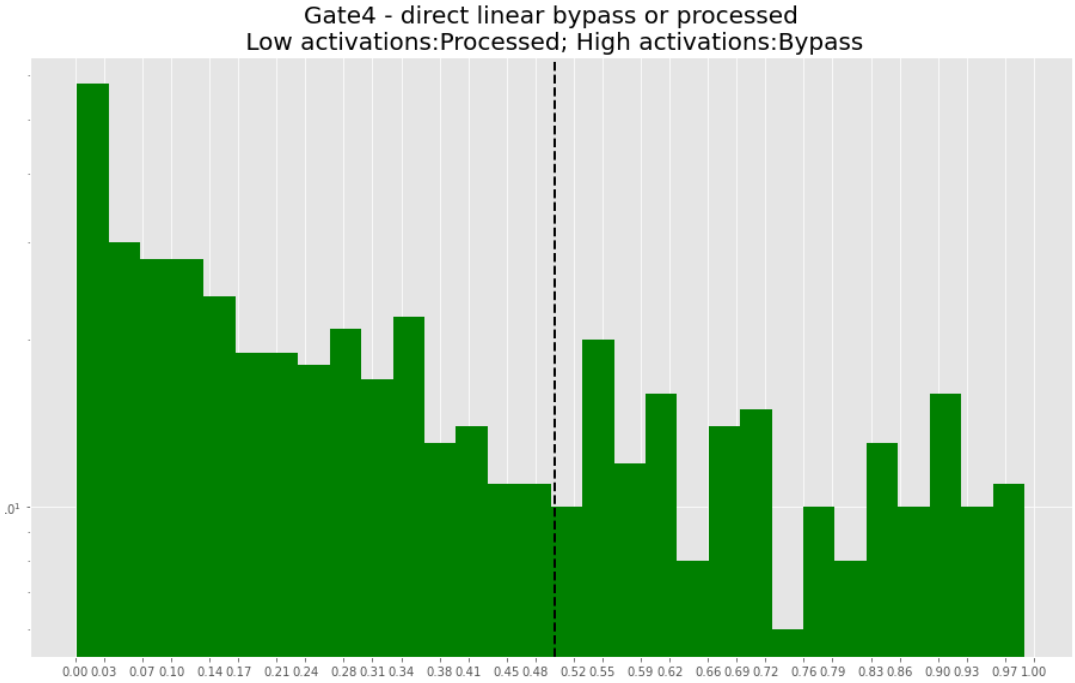

Figure 51 - Automatic gate analysis with gate activation histogram this time for a skip vs processed

4.4.3 Applicability of our experiments in production-grade systems

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 inferences 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 however more experimentation is required for an actual production-grade system for the objective of graphical user interface analysis. Nevertheless, we have proposed the following set of minimal features within our future production grade solution:

- **Single scene inference**: Uploading of various sizes of images and on-shot inference resulting in dense classification of the image in matrix-like format (for demo purposes) as in *Figure 40*.

  - Artificial image input
  - Hand-drawn image input

- **Single scene script generation**: 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 29.
  - Artificial image input
  - Hand-drawn image input will be further analyzed/experimented

- **Single scene code generation**: The final Proof-of-Concept feature is that of the end-to-end conversion where we give input an artificial scene and we will generate an actual HTML5 output with full source code. As inputs we have:
  - Artificial image input
  - Hand-drawn image input

- **Robotic Process Automation integration**: This particular feature implies the operationalization of a basic API that would allow RPA agents to request scene inference and also potentially give a reference user interface control. This way our system is able to semantically understand a initial user interface layout and identify in the second scene the target controls used by the RPA agent to generate events such as button-clicking, scrolling into a list of options and selecting one, editing a certain box with required information.

While some of the above features have already been developed there is still a long road ahead both in terms of experimental development, testing as well as finding the right production-related use-cases.

4.5 Research results summarization

Summating the main research results presented in this thesis we have aspects geared both towards the particular objective of user-interface scene inference as well towards the horizontal objective of improving the creation and generation process of deep neural directed acyclical graphs by minimization of the carbon footprint of the architecture search process. Although both areas are highly innovative and currently research-active both in academic and industrial environments, we argue that the proposed methods and techniques offer great potential for state-of-the-art advancement.

Automatic user-interface inference

In the area of user-interface automatic inference we proposed a new publicly available dataset called *CloudifierNet* that combines both artificial images – actual user interfaces images – as well as hand-drawn UI mockups. Together with the proposed dataset we have researched and experimented with a task-oriented deep neural DAG for dense pixel-level prediction of the scene content as well as script generation of the scene user interface controls layout.

Self-learning graph topology

In the more general area of DAG architecture search and hyper-parameter tuning we propose a new innovative approach to classic grid-search and auto-ml. This approach aims at generating the optimal graph structure based on a one-shot training process rather than a massive hyper-parameter space search and thus dramatically reduce the carbon footprint of the graph hyper-parameter tuning process. Last but not least we have to mention that employing the proposed *MultiGateUnit* allows the model to learn individual hyper-parameter configuration for every module in the graph eliminating the need for in-section module duplication and potentially generating better features than the classic approach of having same repeated structure.

5 Personal contribution areas and final conclusions

In this section the main personal contributions of the author will be enumerated starting with the newest and most promising one for industrial large-scale application – the *Multi Gated Unit* graph modules – and continuing with the contributions in other areas related to the thesis: the proposed open-source *CloudifierNet* dataset as well as the directed acyclical graph architectures.

While the initial purpose of the research has been directly related to artificial scene inference during the research and experimental development processes, we discovered clear and immediate cross-domain applications of our work. This is particularly relevant to the subject of network architecture search approaches.

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 3.3. Our hypothesis is that the MGU can perform equally well in any kind of experiments with almost any kind of graph modules. As presented in section "*Analysis and comparison of MGU sub-graph performance"* a series of experiments has been executed consisting various scenarios exploration.

Multi dataset experiments **-** *Comparing MGU approach with the classic hand-build module-by-module design of tensor graphs*

The whole intuition behind this comparison was to demonstrate that the top resulting models generated from a grid-search approach are similar or worse than the single *MGU* end-to-end trained directed acyclical graph. This particular comparison has been done using various datasets – both public as well as private datasets from real-life production systems – and the metrics used have been divided into two categories:

1. Comparison in terms of performance (*accuracy*) of the graphs using *MGU* versus the classic graphs.
2. Comparison of the time and carbon print required to find the optimal architecture using the classic grid-*search* approach versus the optimization time of the *MGU* graph.

Further research and experimentation have been conducted in the area of graph architecture optimization based on *MGU* and *gate pruning*. Still being an active area of interest

and research, we are confident that this direction will lead to improved more efficient versions of the *MGU* architecture.

5.2 Convolutional graph architectures

Directly related to our initial main objective – inference of artificial graphical user interfaces screens – is the area of convolutional graph architectures. 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 [16] [15] [60] [25]. 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 3.6, 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.

Finally, in the area of convolutional graph architectures we proposed the replacement of classic operations sequence – *convolutions, feature post-processing, non-linear activation* –

with the *MultiGatedUnit* module. This important addition allowed us to eliminate the need for compute and energy expensive grid-search/auto-ml operations while achieving top performance as compared to our reference models.

5.3 Experimenting user-interface analysis

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 good distribution of visual control artefacts. This would allow us in turn to safely consider that we have a high probability of obtaining a trained model that generalizes well. The secondary implication of this target was that we can provide the open source community with a started dataset than can gradually grow in size and scope.

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 3.5 and section 3.5.2 in particular. 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 52 we present an actual example from the dataset augmentation experimentation process.

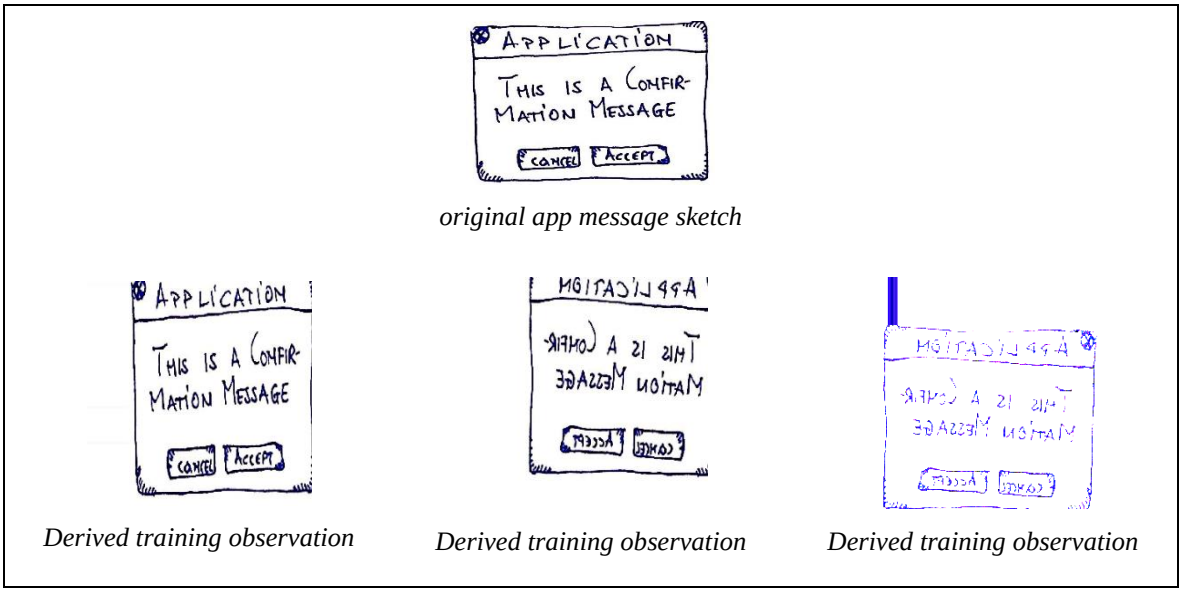

Figure 52 - Sketch training dataset augmentation

As it can be inferred from Figure 52 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 52) and thus solving the issue related to single-color in original data (hand-drawn source) presented in section 3.5.2.

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 [61] [62] 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 0.

5.4 Real life cross-domain applications

One particular direction where our research has pivoted to has been that of applying the *CloudifierNet* architecture with its *Multi Gated Unit* modules in other domains and particularly in the medical radiology domain. In this area we managed to develop and deploy both experimental as well as production-grade systems in areas of dermatology and gynecology.

5.4.1 Dermatology assistance system

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 [63] in conjunction with analyzing and benchmarking both work from past years [64] and more recent [65].

While this work is currently still in development stage and we plan to publish in the next period our research and experimentation conclusions we already managed to pilot medical industry implementation directly in dermatology clinics. More concretely we developed an MS Windows app as an experimental deployable pipeline and deployed it within one of the most prestigious dermatology clinics in Romania where dermatologists use it with on-the-spot taken *dermatographism photo* or *dermatography* - pictures taken with a camera using special dermatological lens. The application uses a *CloudifierNet* deep neural graph in order to generate an inference regarding the lesion that the medic currently analyses as well as possessing the system logic required for continuous training and fine-tuning. Intuitively when the deployed app generates a good inference on a new unseen dermatography the medic will "tell" the system it has done a good classification of the given dermatography while when the system makes a error in classifying the correct lesion the medic will give the feedback as well.

This way, the deployed pipeline using *CloudifierNet* performs inferences as well as a continuous improvement process.

The last important aspect of this real-life experiment is the actual feedback from dermatology specialists. The inferences generated by the proposed *CloudifierNet* deep neural graph received praise from the specialists with particular accent on inference for the notorious corner cases – case where a malign lesion can be mistakenly confounded with a benign and almost mundane skin formation. Worth mentioning is that one of the main specialists has been previously involved in medical research projects for designing and generating skin lesion analysis systems based on machine learning and computer vision.

5.4.2 Oncological gynecology

Another important area of application of our industrial research and experimentations where we have been able to deploy production-grade systems is the area of oncological gynecology. In cooperation with one of the most prestigious oncological institutes in Romania we performed experimental work together with gynecology surgeons specialized in the pathology of women genital area. In this particular real-life application, we have decided to use an ensemble of multiple *CloudifierNet* in order to generate per-case inferences: inferences of *cervigram* radiology images taken from a colposcope in a series of multiple visits of the same case. The starting point of this real-life experiment consisted in using a dataset of colposcopy images provided initially from US National Health Institute and published through the work of Xu et al [42]. This dataset allowed us to have from the very beginning a clear overview of the radiology medical process and protocol required for the analysis of colposcopies. Thus, both the pipeline we designed based on *CloudifierNet* deep neural acyclical graph as well as the host application and its user-interface take into consideration the following aspects of the medical diagnosis protocol:

- ✓ Each case is defined by at least 5 different colposcopies taken at different times
- ✓ Beside the acetic-acid colposcopies and their corresponding images a 6^{th} image is taken using green lens as well as a 7^{th} image is added to the case after applying iodine solution to the target colposcopy area
- ✓ The analysis is done individually on each cervigram as well as a whole

- ✓ A particular attention is geared towards the inference of the cervix transfer zone as there are three types of cervixes and particular Type B and Type C might imply extra analysis of hidden zones.

This particular system has been developed and deployed on low-compute resources in order to allow the following main objectives:

i. Allow medical personnel to use mobile colposcopy device output images

ii. Allow medical personnel to take on-the-spot *cervigrams* during cervical cancer screening missions in remote areas

iii. Finally obtain an on-the-spot *opinion* inference from the proposed application allowing a quick decision for follow-up treatment in early cases of *Human-Papilloma-Virus*-caused cervical lesions

6 Proposed future research and development

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*. While the description of this potential *MVP* is beyond the scope of this work 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 [61] [62];
- ✓ 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 0 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* [66] 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)* [67]. Even more precisely we will try both the recent advances in actor-critic methods such as *Soft Actor Critic (SAC)* [68] and the simpler *TD3 (Twin Delayed)* [69] 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 3.6.6. Nevertheless, we are currently analyzing the application of more advanced neural language models approaches such as the *Transformer* [10] 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 53*. This approach will enable us to construct a multi-language and multi-purpose architecture capable of generating deployable source code in a variate environment of coding languages.

Aside from the obvious end-to-end sequence generating architecture research, another area of further research that we have in our scope is that of constructing more advanced methods for understanding user-flow and user-experience. This basically consists in automated methods based on deep neural graphs that map the user behavior within a observed environment – such as a *legacy accounting software* for example – to an action space that can be semantically *modeled* in order to replicate the functionalities within a target translated application.

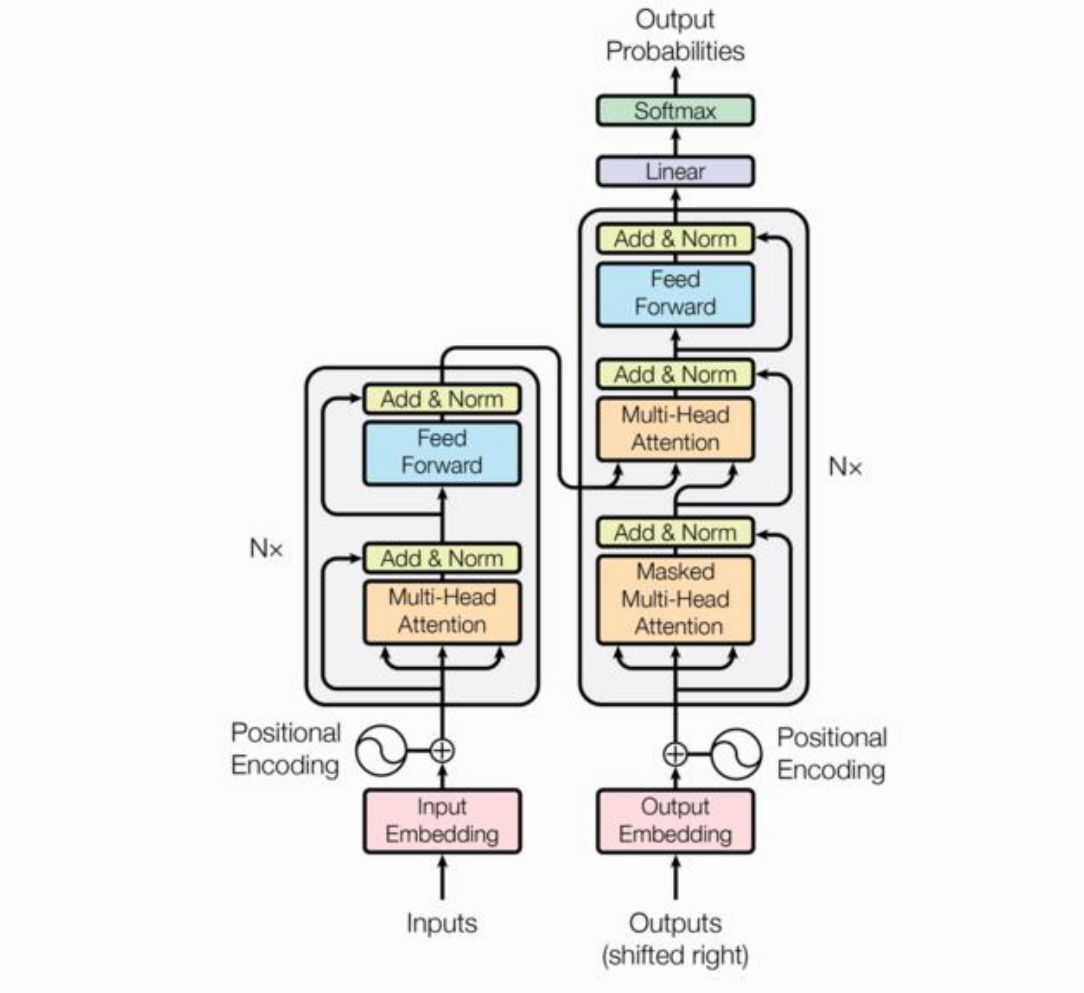

Figure 53 The Transformer architecture based on "Attention is all you need" by Vaswani et al

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* [70] and *GPT-2* [71].
- ✓ 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 [61]. 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.

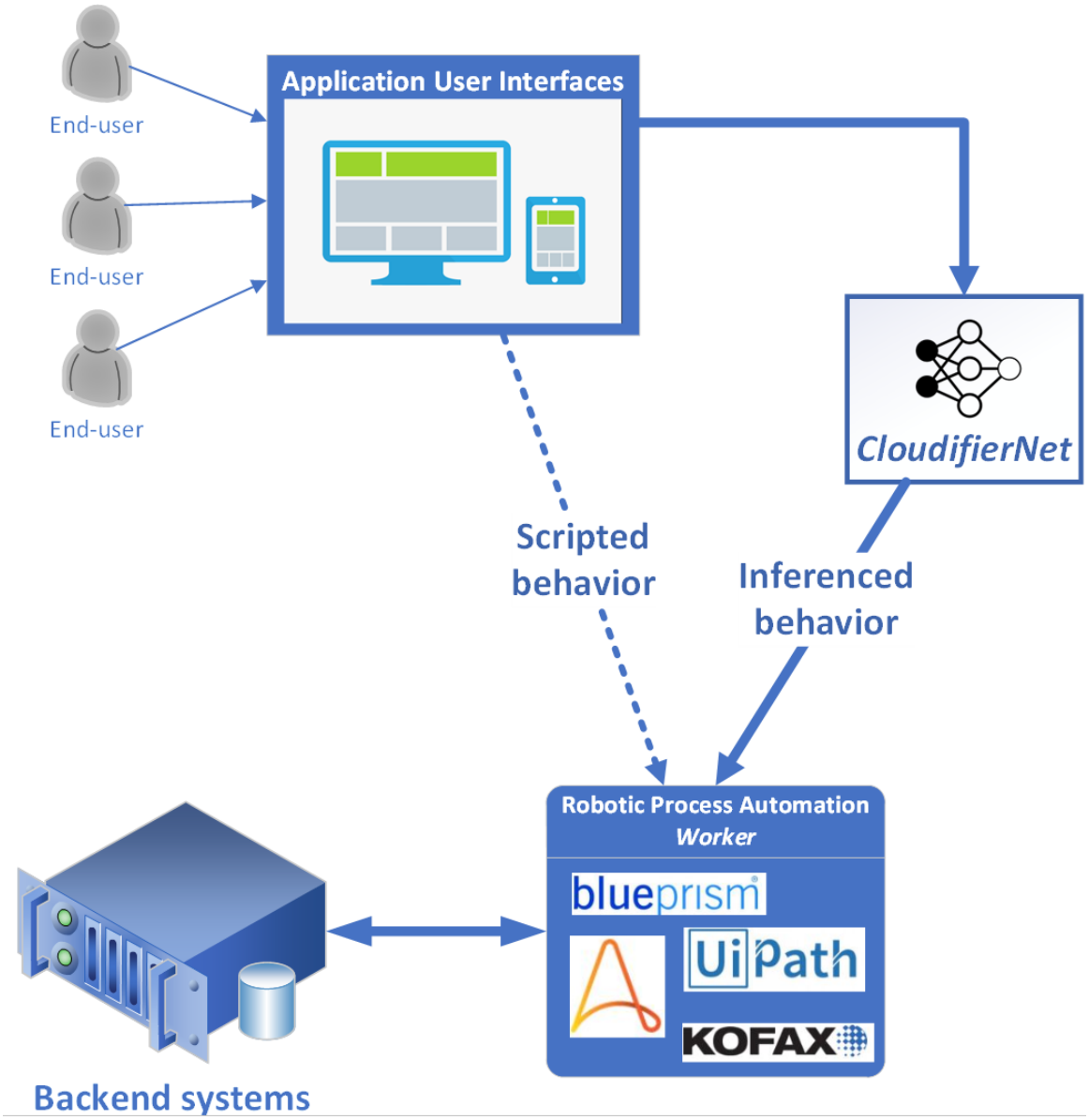

Figure 54 Model based inference vs classic scripted behavior analysis

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 54 Model based inference vs classic scripted* behavior analysis.

6.6 Process flow, user intent and user-experience 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)

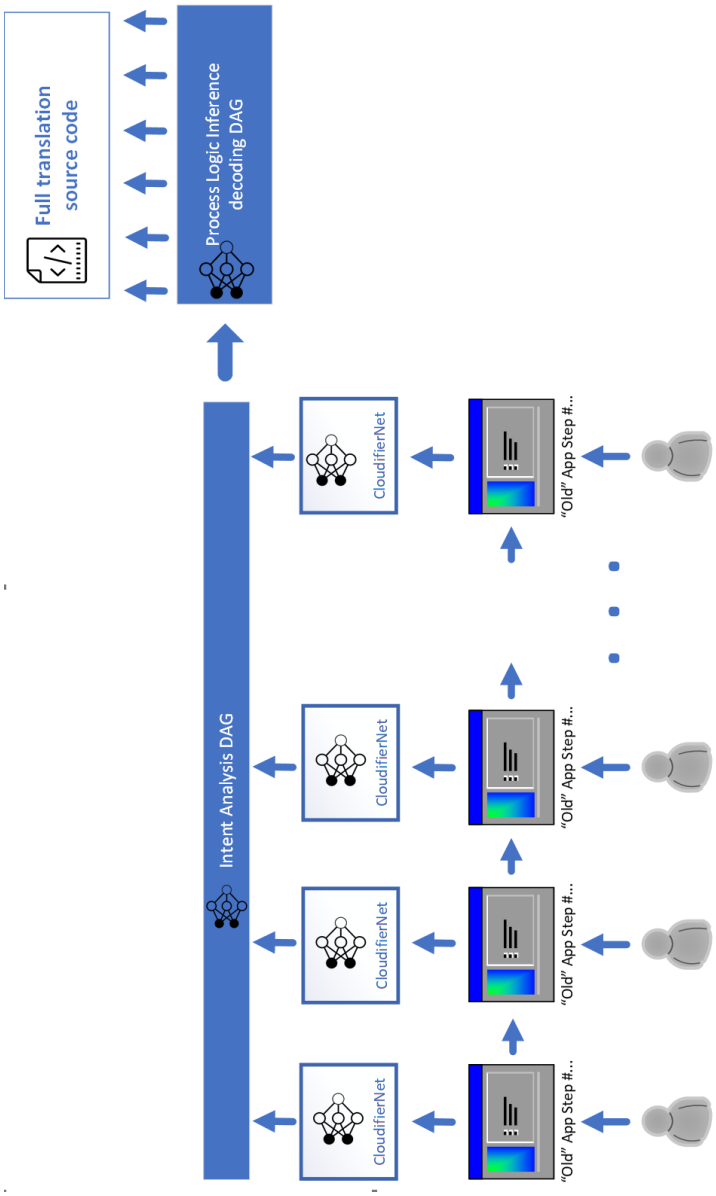

Figure 55 - Proposal for an end-to-end trainable architecture for process logic inference and source code generation for both UX and application logic flow

This approach presented in Figure 55 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-related 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 [10] 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 [72] 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 [73] [74] combined with the application of distillation [75] methods.

Carbon footprint of deep neural acyclical graphs optimization process as well as that of normal *inference-mode* computation is a clear and important concern. As such, probably the most important research within our work as well as the most important future research direction is that of further optimizing the proposed methods for graph topology creation, further presented in next section.

6.8 Further research on Multi Gate Units

Several research directions are, at the time of the current thesis preparation, currently in work for what we consider our most important and horizontally applicable research result – the *Multi Gated Unit* [76]. Also, there are two research papers in work – one position paper that it is in peer-review status and a more elaborate research paper that will be published in the next period. In order to further improve the multi-gating sub-graph mechanism and make addition steps in the direction of auto-learning of graph architecture hyperparameters we identified several clear directions of research:

i. Probably the most important aspect of our current research is the problem of MGU over-parametrization. The problem of **over-parametrization** occurs in situations where large graph modules are connected and require big transformation tensors. In this case the main issue of the *MGU* is its intrinsic need to replicate the linear transformation behavior of the encapsulated layer for each individual gate. We can imagine that for a 3x3 kernel convolutional operation that results in 2^{11} channels and takes as input a volume of 2^{10} channels a transformation tensor of $9*2^{21}$ is required and this tensor is then used with individual transformations of the module input for all the internal gates of the *MGU*. Several options are currently explored:
    a. Heuristics based - using a less complex transformation that still generates a similar tensor with the one of the *MGU* encapsulated module. For example, using a depth-based kernel discrete convolution for the case of a classic convolution encapsulated layer and thus drastically reducing the needed tensor from our above example from approximatively 2^{24} down to 2^{14} parameters. This way the total needed memory for a single *MGU* module is just a fraction of the memory needed for the classic encapsulated layer – for our example the total additional memory required for the gating mechanisms of the MGU would be approximatively 2^{17} parameters. Please note that a parameter is most likely represented as a 32-bit float number.
    b. Low dimensional embedding representations – using similar approaches such as that of Hu et al [31] we will construct lower dimensional representations of the *MGU* input that will be used as gate transformation for each self-learning gates. The lower dimensional representations will be

then expanded and used in conjunction with the output generated by the MGU encapsulated operation.

ii. Research and experimentation with **bias configuration heuristics** for each individual gate in order to control the starting point of information flow

iii. Introduce more drastically approaches to **impose feature selection** through information low-dimensional projection – and implicitly the actual information flow pathway selection within the multi-gated unit - on the gate units. Introducing mechanisms such as squeeze-and-excite [31] is one of the potential options of generating better embedding vectors for the gating units. This area objective is to improve the gating units from simple sigmoid-activated linear units to more complex ones.

iv. Experiment with **different gating activation** methods as well as different gating inputs – such as maybe gate dependent on more than just inputs

v. Allow more direct **flow of basic features and gradients** such as replacing or augment the last gate with direct residual connection for the complex MGUs

vi. Expand the range of features and degree of **explain-ability for layer self-analysis** (explain each gate learned features). This research direction is closely related to the current trend in the artificial intelligence community towards explain-ability in machine learning and more importantly in deep learning.

vii. While our approach tends to solve the problem of multi-iteration **graph topology search** in a direct end-to-end approach for a discrete number of cases there are two main inherited flaws that we currently research and require further research:

   a. Address the **complexity of the MGU units** – although the proposed module can encapsulate almost any kind of known module (linear, discrete convolutional, separable convolution, depth-wise convolution, etc.) there

still remains the problem of having even multiple options in terms of potential branching within the module

b. Connected with the previous proposed issue we already acknowledged the need for **pruning the optimized graphs** and find automated methods for dropping certain pathways within each module based on the importance give by the associated gates. While this might seem trivial for gates that tend to fully open or close the target operations, in most cases the gates have partially-clear behaviors – such as having an average of 0.6 gate activation value and thus being impossible to decide that the gate is actually opened (such as for a gate with 0.99 average activation).

7 References

[1] UiPath, "https://www.uipath.com/product/platform/ai-computer-vision-for-rpa." UiPath, 2020, [Online]. Available: https://www.uipath.com/product/platform/ai-computer-vision-for-rpa,.

[2] ECMA International, "The JSON Data Interchange Format - ECMA-404." 2017, [Online]. Available: https://www.ecma-international.org/publications/standards/Ecma,.

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

[4] J. Ghorpade, J. Parande, and M. Kulkarni, "GPGPU PROCESSING IN CUDA ARCHITECTURE," *Adv. Comput. An Int. J. (ACIJ*, vol. 3, [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1202/1202.4347.pdf,.

[5] A. Halevy, P. Norvig, and F. Pereira, "The Unreasonable Effectiveness of Data," *IEEE Intell. Syst.*, vol. 24, no. 2, pp. 8–12, [Online]. Available: https://research.google/pubs/pub35179/.

[6] A. I. Damian and N. Tapus, "Model Architecture for Automatic Translation and Migration of Legacy Applications to Cloud Computing Environments," 2017, doi: 10.1109/CSCS.2017.88.

[7] A. I. Damian, A. Purdila, and N. Tapus, "Cloudifier virtual apps: Virtual desktop predictive analytics apps environment based on GPU computing framework," 2017, doi: 10.1109/ICCP.2017.8116994.

[8] A. I. Damian, L. Piciu, A. Purdila, and N. Tapus, "Cloudifiernet - Deep vision models for artificial image processing," *Procedia Computer Science*. pp. 720–728, 2019, doi: https://doi.org/10.1016/j.procs.2019.12.043.

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

[10] A. Vaswani *et al.*, "Attention Is All You Need," in *Advances in neural information processing systems; https://arxiv.org/abs/1706.03762*, 2019, pp. 5998–6008,.

[11] M. Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.” Corenell University Library, 2015.

[12] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Advances in Neural Information Processing Systems 32*, M. F. D. Z. Paszke, Ed. 2019, pp. 8024–8035.

[13] 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*, .

[14] C. Szegedy *et al.*, “Going Deeper with Convolutions.” [Online]. Available: https://arxiv.org/abs/1409.4842,.

[15] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition.” [Online]. Available: https://arxiv.org/abs/1512.03385,.

[16] F. Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions.” [Online]. Available: https://arxiv.org/abs/1610.02357,.

[17] J. Long, E. Shelhamer, and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation.” [Online]. Available: https://arxiv.org/abs/1411.4038,.

[18] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition.” [Online]. Available: https://arxiv.org/abs/1409.1556,.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html,.

[20] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, no. ttps://www.bioinf.jku.at/publications/older/2604.pdf, vol. 8, pp. 1735–1780,.

[21] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines.”

[22] M. Sandler, A. Z. M. Howard, A. Zhmoginov, and L. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks.” [Online]. Available: https://arxiv.org/abs/1801.04381,.

[23] S. Zagoruyko and N. Komodakis, “Wide Residual Networks.” [Online]. Available: https://arxiv.org/abs/1605.07146,.

[24] K. He, X. Zhang, S. Ren, and J. Sun, “Identity Mappings in Deep Residual Networks.” [Online]. Available: https://arxiv.org/abs/1603.05027,.

[25] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning.” .

[26] M. Lin, Q. Chen, and S. Ya, “Network in network,” *CoRR*, [Online]. Available: https://arxiv.org/abs/1312.4400,.

[27] T. Lin *et al.*, “Microsoft COCO: Common Objects in Context.” .

[28] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection.”

[29] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks.” [Online]. Available: https://arxiv.org/abs/1905.11946,.

[30] M. Tan, R. Pang, and Q. Le, “Efficientdet: Scalable and efficient object detection,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10781–10790 , [Online]. Available: https://arxiv.org/abs/1911.09070,.

[31] J. HU, L. SHEN, and G. SUN, “Squeeze-and-excitation networks.”

[32] P. RAMACHANDRAN, B. ZOPH, and Q. V LE, “Searching for activation functions.” [Online]. Available: https://arxiv.org/abs/1710.05941,.

[33] M. O’Connor, N. Chatterjee, D. Lee, J. Wilson, and A. Agrawal, “Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems.”

[34] NVidia, “NVIDIA Tesla P100.” [Online]. Available: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf,.

[35] NVidia, “NVIDIA TESLA V100 GPU ARCHITECTURE.” [Online]. Available: http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf,.

[36] J. Stone, D. Gohara, and G. Shi, “OpenCL: A Parallel Programming Standard for

Heterogeneous Computing Systems," *Comput. Sci. Eng.*, no. ttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC2964860/, vol. 12, number 3.

[37] J. Bergstra *et al.*, "Theano: A CPU and GPU Math Compiler in Python," [Online]. Available: https://hgpu.org/?p=6556,.

[38] A. Karpathy and F.-F. Li, "Deep visual-semantic alignments for generating image descriptions," [Online]. Available: https://pdfs.semanticscholar.org/84f6/f2e1ec5a2f1a1b5efe9dc65d938db1d0f0a0.pdf,.

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

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

[41] O. Vinyals, T. A. S. Bengio, and Erhan, "Show and tell: A neural image caption generator."

[42] K. Xu *et al.*, "Show, attend and tell: Neural image caption generation with visual attention," in *nternational conference on machine learning; https://arxiv.org/abs/1502.03044*, pp. 2048–2057,.

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

[44] 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*, .

[45] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks." [Online]. Available: https://arxiv.org/abs/1505.00387,.

[46] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.*, Nov. 2016, Accessed: Jun. 28, 2021. [Online]. Available: http://arxiv.org/abs/1611.01578.

[47] S. J. Taylor and B. Letham, "Forecasting at Scale," *Am. Stat.*, vol. 72, no. 1, pp. 37–45, Jan. 2018, doi: 10.1080/00031305.2017.1380080.

[48] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by

reducing internal covariate shift.” [Online]. Available: https://arxiv.org/abs/1502.03167,.

[49] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization.” [Online]. Available: https://arxiv.org/abs/1607.06450,.

[50] D. Mané, “TensorBoard: TensorFlow’s visualization toolkit,” *Tensorflow.org*, [Online]. Available: https://www.tensorflow.org/tensorboard,.

[51] L. Y. Pratt, “Discriminability-based transfer between neural networks,” [Online]. Available: https://dl.acm.org/doi/10.5555/645753.668046,.

[52] 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 , [Online]. Available: http://www.image-net.org/papers/imagenet_cvpr09.pdf,.

[53] 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*, .

[54] M. Everingham, L. V. Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.157.5766&rep=rep1&type=pdf,.

[55] R. Goyal *et al.*, “The ‘something something’ video database for learning and evaluating visual common sense.” [Online]. Available: https://arxiv.org/abs/1706.04261,.

[56] F. Mahdisoltani, G. Berger, W. Gharbieh, D. Fleet, and R. Memisevic, “Fine-grained Video Classification and Captioning.” [Online]. Available: https://arxiv.org/abs/1804.09235,.

[57] S. Chetlur *et al.*, “cuDNN: Efficient Primitives for Deep Learning.” Accessed: Mar. 03, 2021. [Online]. Available: https://arxiv.org/abs/1410.0759.

[58] B. Abadi *et al.*, “TensorFlow: A System for Large-Scale Machine Learning.”

[59] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization.” [Online].

Available: https://arxiv.org/abs/1412.6980,.

[60] L. Lai, N. Suda, and V. Chandra, “Deep Convolutional Neural Network Inference with Floating-point Weights and Fixed-point Activations.” [Online]. Available: https://arxiv.org/abs/1703.03073,.

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

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

[63] P. Tschandl, C. Rosendahl, and H. Kittler, “The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions.” [Online]. Available: https://arxiv.org/abs/1803.10417,.

[64] C. V, Q. B. Nguyen, and S. Pankanti, “Deep Learning Ensembles for Melanoma Recognition in Dermoscopy Images,” *J. Res. Dev.*, [Online]. Available: https://arxiv.org/abs/1610.04662,.

[65] A. Rezvantalab, H. Safigholi, and S. Karimijeshni, “Dermatologist Level Dermoscopy Skin Cancer Classification Using Different Deep Learning Convolutional Neural Networks Algorithms.” [Online]. Available: https://arxiv.org/abs/1810.10348,.

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

[67] T. P. Lillicrap, J. J. Hunt, and A. Pritzel, “Continuous control with deep reinforcement learning.” [Online]. Available: https://arxiv.org/abs/1509.02971,.

[68] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.” [Online]. Available: https://arxiv.org/abs/1801.01290,.

[69] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods.” [Online]. Available: https://arxiv.org/abs/1802.09477,.

[70] J. Devlin and al, “BERT: Pre-training of Deep Bidirectional Transformers for Language

Understanding.” [Online]. Available: https://arxiv.org/abs/1810.04805,.

[71] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language Models are Unsupervised Multitask Learners,” *OpenAI blob*, [Online]. Available: https://github.com/openai/gpt-2,.

[72] E. Strubell, A. Ganesh, and A. McCallum, “Energy and Policy Considerations for Deep Learning in NLP.” [Online]. Available: https://arxiv.org/pdf/1906.02243.pdf,.

[73] S. Vogel, C. Schorn, A. Guntoro, and G. Ascheid, “Efficient Stochastic Inference of Bitwise Deep Neural Networks.”

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

[75] G. Hinton, O. Vinyals, and J. Dean, “Distilling the Knowledge in a Neural Network,” in *https://arxiv.org/abs/1503.02531*, .

[76] A. I. Damian, L. Piciu, and N. Tapus, “A view on automated neural graph topology generation and a viable direction of innovation,” in *Proceedings - RoEduNet IEEE International Conference*, 2020, vol. 2020-Decem, doi: 10.1109/RoEduNet51892.2020.9324853.