CloudifierNet

Deep Vision Models for Artificial Image Processing

Andrei Ionut DAMIAN (*Author*)
Cloudifier SRL, Bucharest, Romania

Laurentiu Gheorghe PICIU (*Author*)
University "Politehnica" Bucharest, Romania

Nicolae TAPUS (*Author*)
University "Politehnica" Bucharest, Romania

Alexandru PURDILA (*Author*)
Cloudifier SRL, Bucharest, Romania

***Abstract* — With the advancement of Artificial Intelligence and Deep Learning with its multitude of applications in particular, a new area of research is emerging – that of automated systems development and maintenance. With this particular area of application of Artificial Intelligence, a broad range of computer users – from IT maintenance personnel to software developers – will be enabled to use automated inference and prediction tools. These automation tools will be in future available for a multitude of tasks such as general purpose automated maintenance of custom applications and operating system issues. Our vision is to research and develop truly intelligent systems able to analyze user interfaces from various sources and generate real and usable inferences ranging from architecture analysis to actual code generation. One key element of such systems is that of artificial scene detection and analysis based on deep learning computer vision systems. 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 models will develop during the training process natural image feature detectors with the exception of the base graph modules that will learn basic primitive features. In the current paper we will present the base principles of a deep neural pipeline for computer vision applied to artificial scenes (scenes generated by user interfaces or similar). Finally, we will present our conclusions based on experimental development and benchmarking against state-of-the-art transfer-learning implemented deep vision models.**

Keywords— artificial intelligence, deep learning, computer vision, automated programming

I. Introduction

Artificial Intelligence and Deep Learning in particular rapid advancement in the past years generates an almost infinite multitude of potential applications with strong impact in our lives [1]. From business predictive analytics to computer vision, from data security to healthcare it is hard to imagine a particular field where Artificial Intelligence impact will not be felt in coming future. One particular area of interest is the area of computer systems and software development and maintenance. In this particular area we foresee a number of potential applications some of which are already in research development in various prestigious artificial intelligence laboratories. Among these applications there are two particular directions that we are focused on:

- Automated legacy application translation using advanced visual inference and automated programming based on user interface activity. Within this research direction the main objective is construct advanced visual recognition systems for artificial scene instances segmentation coupled with sequence-to-sequence translation of user actions and visual flow to finally output actual intermediary source code. This intermediary source code must address both the user experience graphical interface and the actual basic functionalities of the user interface control behavior. A particular use case would be the translation of a simple financial management application written for windows or even MS-DOS operating systems in '90s to a modern web-based online system that would be uploaded within a cloud computing infrastructure.
- Intelligent inference of systems maintenance use cases is the second area where our main objective is to advance the state-of-the-art in the area of automated maintenance tools for 3rd party software systems. For this particular area we focus on the need to produce intelligent virtual agents capable to replacing the need for remote analysis currently done by software engineering and system administrators. As a particular use case, we could imagine the maintenance procedure of a client-server system where the end-user interacts with a thin-client user-interface and requires the systems/software engineers assistance for a potential identified bug. In this case our end-to-end pipeline could understand the basic behavior and flow of the given user interface (and the potential buggy functionalities) and provide the maintenance team with advanced debugging information.

Both the above mentioned areas of intervention will be further detailed in following sections however then main focus of our paper is to present our research findings in the area of artificial (*synthetically generated via computer graphics methods*) scene inference. In this particular area we started with the most well-known architectures – presented within the *Related Work* section of our paper – we have developed our own training and validations datasets and finally we have identified optimal network architecture for end-to-end artificial scene inference. Aside from the artificial scene inference task we also have included within our scope of work the inference of natural

Research sponsored by Cloudifier SRL

scenes generated by actual hand-drawing of user interfaces mock-ups thus increasing the real and commercial application of our research and experimentation work.

II. Related work

Our work relates to the most influential deep convolutional directed acyclic graphs architectures – namely the Inception [2] and ResNet [3] as well as several other architectures such as separable convolution network proposed by Xception [4], and also the fully convolutional model for end-to-end image segmentation FCN [5] based on the straight sequential VGGNet [6] deep neural network.

A. Inceptions, residual and skip connections

Our work is strongly related with de *Inception* architecture developed by Szegedy et al [2] combined with the residual connections proposed by He et al [3] and finally introduced by Szegedy et al in the 4th version of the *Inception* architecture [7]. As it will be presented within the architectural section we are using a custom version of a *Inception*-residual module interlinked together with modules based on separable depth-wise convolutions fully augmented by residual connections for efficient gradient back-propagation.

B. End-to-end image semantic segmentation

The proposed architecture is based on the basic principles of dropping all pooling layers within the convolutional network and inserting larger step convolutions for map width/height reduction and also replacing the dense layers with convolutional. Finally, this results in transforming the entire computational graph into one big fully convolutional directed acyclic graph. As described in the related work by Long et at [5] we replace the final fully connected dense top layers with transposed convolution layers in order to learn upscaling kernels that will convert the reduced activation volumes from the final convolutions to the initial size and depth of the image. We also use the skip-and-merge strategy in order to fuse lower-level upscaled maps with later, and thus higher-level, upscaled maps.

C. Other similar work in this area

The field of automatic program generation that strongly relates to our work has known many attempts and approaches over the years ranging from systems designed for automatic code generation based on (near) natural language specifications up to source code generation based on an interface mockup (computer aided drawing of user-interface mockup). The closest and newest similar work to our knowledge is the pix2code [8] proposed by Beltramelli T. In relation to this proposed approach we argue that our work is more generalized as follows:

- in terms of target platform as we are proposing cross-platform approach similar to our early work [9]
- in terms of source input our advanced neural model accepts both artificial data (such as screen snapshots) and hand-drawn natural images (mock-ups)
- our model generates a dense prediction of the actual observed UI scene (artificial or natural) excluding the need of a RNN-based source code generator and inherited problems such as the potentially erroneous generated code or the need for soft/hard attention (proposed as a future improvement in referenced work)

III. Proposed Architecture

A. End-to-end trained portable model justification

One of the main goals of our chosen architecture design is to be able to deploy the production models on different devices in inference mode the same version of the trained computational graph. The initial challenge consisted in obtaining a architecture that will run with acceptable performance results both on GPU-augmented embedded devices such as Nvidia Jetson family and on mobile devices that do not have GPU parallel computing capabilities (either Android or similar devices).

One of the use cases that we envisioned from the very beginning was a mobile device app that would enable user-interface inference and refactoring. More specifically, the app would enable users with no prior software engineering experience to take a snapshot of a legacy application screen, a website or a simple UI mockup hand-drawing. Following this initial step, the deployed pre-trained model in the smart-phone device would generate an inference of the snapshot image and produce a UI functional design including multiple visual templates based on inferred functionalities. Finally, the app, using an intermediary module/engine, would generate a simple runtime environment, including minimal views and controllers, for a target web-server and with one or multiple cloud publishing mechanisms would push the functional website where it could be accessed.

B. Model achitecture

The main building blocks of our proposed architecture are the custom directed acyclic graph blocks presented in *Figure 1* and *Figure 2*. The first basic building block based on the Inception-ResNet [7] concept is basically a three-column directed acyclic graph with a skip connection that starts with a 1x1 convolution without non-linearity that prepares the input volume and connects it to the final bottleneck final that takes as input the concatenated 3-columns. Within the 3 columns infrastructure we use simple 3x3 and 5x5 same convolution together with the 1x1 bottleneck convolutions. All convolutions are followed by batch-normalization before the non-linearity. We also employ pre-activation skip connection – a network architecture that do not introduce any non-linearity between any two skip-connections as proposed by He et al in [10]. The main objective of this block architecture is to create a local network topology as proposed by the network-in-network architecture of Inception coupled with latest research findings in employing skip-connections. In order to decrease our network size, we use the wide residual network principles described by Zagoruyko et al [11] and increase the volumes of our residual blocks.

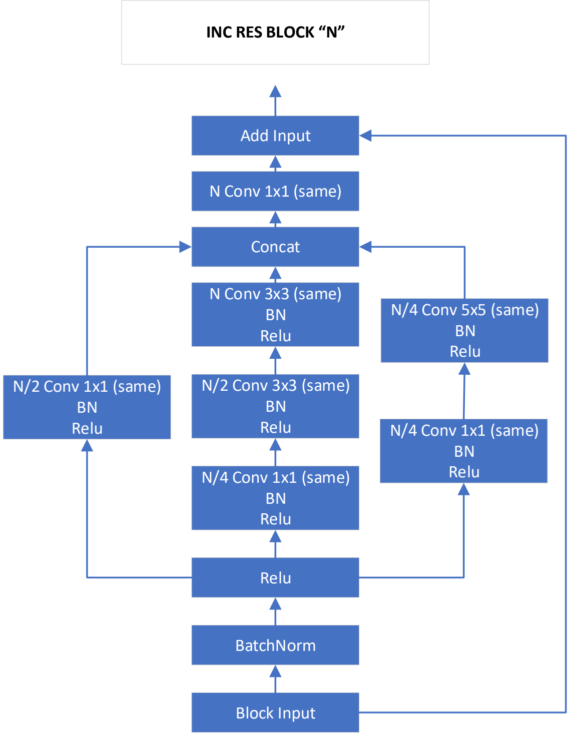

Figure 1– INC RES BLOCK with N output filters

The second building block "*DS RES Block*" is based on the concept of separable depth-wise convolution proposed by Chollet [4] in the Xception architecture together with the already mentioned skip connections required for the gradient flowing optimization. Each of the *DS RES Blocks* is then upscaled using a transposed convolution 2D (or a so called *backwards convolution*) using a specific fractional stride in order to upscale from respective volume to the initial input volume height and width.

Within the *Readout Block* all the upscaled volumes are merged in a *H*W*D* volume where *H* and *W* are respectively the height and width of input volume and the *D* is the depth of the concatenated volumes from the *DS RES Blocks.* Finally, a dense output map of size *H*W*C* is generated with a simple fully connected layer applied on each of the *H*W fibers* in the *H*W*D* volume, where *C* is the number of inferred classes. This particular architecture allows the fully convolutional DAG to be *input volume size agnostic* and to have the capacity to accept any kind of input volume without height or/or width restriction.

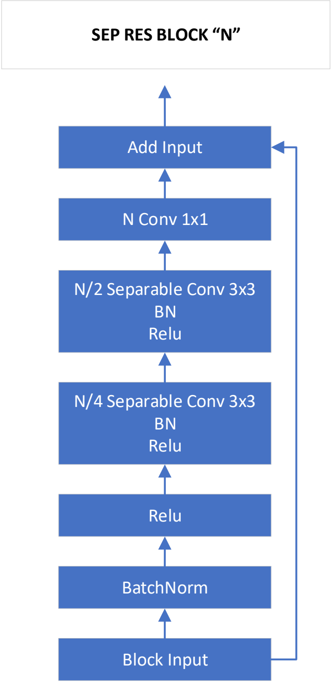

Figure 2– DS RES Block with N output filters

As previously mentioned, in order to achieve our proposed goal, we use a alternation of the two main building blocks as presented in Figure 3. The architecture has been determined experimentally and has been tested both on natural and artificial images.

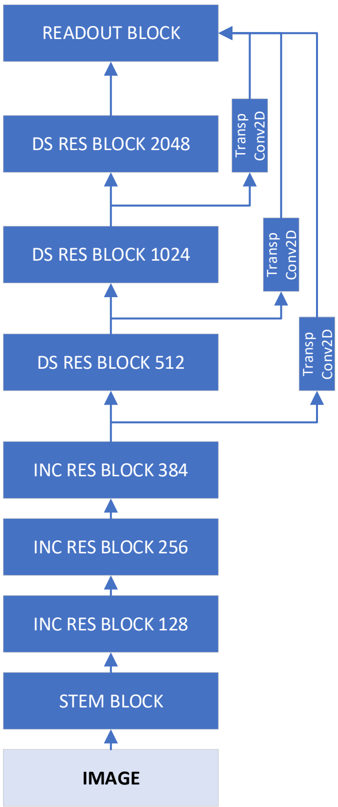

Figure 3– Overall architecture of CloudifierNet

The main objective of this architecture is to obtain the good balance between powerful feature detection – based on inception modules with skip connections - and model complexity. Also, a very important objective is to maximize the operation parallelism within out computational graph by reducing as much as possible the sequential operations and transferring the complexity to the parallelizable operations. This is achieved effectively through reducing the length of the computational graph while increasing the so-called depth of each computational node, all this by increasing the number of convolutional kernels in each convolutional operation – either depth-wise separable or classic Conv2D operation. As a result our computational graph can maximize the use of GPU numerical core parallelism both at training time and inference time. For training we use P5000 family NVidia GPU – as it will be further described within the *Training and Experimentation* section - while at inference time we use both Nvidia Jetson TK1 and the newer TX2 generation.

In terms of complexity we have a average number of layers compared with various state-of-the-art architectures totaling 109 layers that sum a little over 10^6 weights for the CloudifierNet model. For initial experiments we used 32bit floats but we plan on further experimenting with reduced size weights (16bit, 8 bit and even extreme 1 bit for inference) based on [12] [13] and with combined float/integer for weights/activations such as the ones proposed by Lai L et al [14].

Finally, our proposed readout layer is a dense prediction SoftMax layers where each input pixel is given a probability over all known classes similar to the fully convolutional architecture for image segmentation proposed by Long et al [5]. This structure allows us to further train our models with negative log likelihood objective function applied to the entire dense prediction map for all images within a minibatch.

C. Datasets preparation and augmentation

Our models have been trained using two different kinds of datasets that we are planning on publishing open-source for further research use:

- the "artificial" dataset consisting of software generated user interfaces controls and actual scenes (full user interfaces)
- the "natural" dataset consisting of hand-drawn mockups

The "artificial" dataset preparation and augmentation process has been done in multiple iterations. The main aspects that have been taken into account have been:

a) Operating systems dependent visual aspects of user interface (OS-theme visual controls). For this particular issue we have targeted legacy operating systems and their user-interfaces themes such as Windows 95, 98 and XP.
b) Compiler and development environment dependent visual control primitives. Narrowing the search to the proposed target operating systems, we have researched several different legacy development environments (such as Borland Delphi 1-3, Visual Basic, FoxPro, etc.) and extracted visual themes and customary user interface graphical primitives.

Finally, all "artificial" dataset observations have been generated using automated tools that performed the following tasks: (a) visual control generation; (b) automatic image labeling; (c) automatic visual control instance segmentation. The final "artificial" dataset is composed of multiple meta-batches of 3072 observations of 352x352 cropped images with 3 channels, based on the fact that we train our models with various mini-batch size within 32-128 range. As previously mentioned the labels dataset contains both class-per-observation as well as dense pixel-level classes – each image observation has an associated dense pixel-class map and an overall label. We use both coarse labels based on well-known major UI control groups and fine labels where each type of UI visual has multiple sub-classes.

The second dataset, the so called "natural" dataset consists in hand-drawn examples of user interface primitives or actual user-interfaces mockups. It is important to mention that this dataset is limited in size due to the complicated nature of the hand-drawing and scanning process. Nevertheless, we are using varied image dataset augmentation approaches such as random rotation, random shifting, random channel shifting, random flipping, random rescaling, random cropping in order to dramatically increase the size of our proposed natural image dataset (up to 7 times the original size of the natural dataset). It is important to mention that most of these proposed image data augmentation methods are not required for the artificial dataset due to the actual used image/data acquisition process.

IV. Training and Experiments

A. Models training setup

Given the previously presented architecture and datasets with dense-labels per image and considering each image has a *H* x *W* size, for *N* mini-batch images our training objective is to minimize the dense output cross-entropy or more specifically the negative log likelihood objective function w.r.t. the model parameters *Θ*.

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

The dense output of the DAG is nothing more than a Softmax function applied to each *1x1xC* "*fiber*" of the final output volume.

$$h_{softmax}\left(x_i^{(j)} \middle| \theta, j \in M, i \in N\right) = \frac{e^{\theta^T x_i^{(j)}}}{\sum_k^N e^{\theta^T x_k^{(j)}}} \quad (2)$$

We trained our models end-to-end using Adam 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.

B. *Training hardware infrastructure*

As a training environment infrastructure, we used both GPU and CPU. The training machine CPU capabilities consisted of 32 GB RAM and 2 Xeon processors each with 8 physical cores for total logical core count of 32 cores. Nevertheless, the main training of our models has been done on the machine GPUs summing a total of 4224 CUDA cores provided by a 16 GB Nvidia Pascal 5000 GPU with 2560 CUDA cores and an additional 8GB Nvidia Maxwell 4000 GPU with 1664 CUDA cores.

The optimization process is presented in below figure

Figure loss/epochs

In our pipeline validation experiments we have targeted both types of proposed scenes: artificial images - that is actual user interface screen-shots – and also hand-drawn user interface mockups. For the experiments we used several versions of our models that varied both the model required memory/computation capacity and the model training time. We considered this approach in order to cover the potential usage of our model pipeline in smart mobile devices as previously mentioned in our architecture section.

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%

Table 1 – Experimentation results for test dataset

The final results of our experiments presented in *Table 1* are based on two different models both following previously described architecture. More specifically, *Cloudifier109* is the 109 layers model presented in Proposed Architecture section and *Cloudifier50* is a reduced version in terms of modules of the 109 layers version. The benchmarking tests have been performed both on a reduced performance mobile computer using a Nvidia GeForce 940MX with only 2GB RAM and a total of 384 CUDA cores. We have also considered running the inference task and on a Nvidia Jetson TX2 device with 256 core Pascal GPU and 8GB of RAM. The inference environment for our models has been based on TensorFlow [15] and also on a specialized inference engine, namely TensorRT - a library that facilitates high performance inference on NVIDIA GPUs. Final results have been generated by cross-validation approach and averaged over target environments.

For training/validation/testing split we decided to retain 7% of the whole joined dataset for validation/testing purposes and 93% has been used for actual model training. Out of the 7% retained dataset we used 4% for testing and 3% for validation. During the initial training experiments, we additionally used in-training random validation dataset of 5% of proposed training data.

The tests and the results have been divided in two categories – the artificial scene inference and the natural hand-drawn mockup scenes inference. In our experimentation results table *ArtAcc* and *ArtRec* represent the accuracy and the recall for the artificial dataset tests, while *NatAcc* and *NatRec* represent the accuracy and the recall for the natural hand-drawn scenes.

As it can be observed from the above presented performance indicators we achieved results that clearly demonstrate the capability of our model to infer user interface functionalities based on static scenes.

V. Conclusions

Our proposed models have achieved beyond state-of-the-art results in the area of user-interface scene inference and a new state-of-the-art status in the area of natural UX mockups inference where we did not identify any particular current state-of-the-art. Although it has proven a powerful approach for the proposed tasks, our end-to-end pipeline is not yet capable of inferring actual high-level functionalities. Thus, one area of further research and improvement of the end-to-end pipeline model is the addition of high-end application functionality inference, albeit only for artificial user-interface video streams. This future proposed work will augment our models with the ability to analyze a video stream presenting a actual user-application interaction and infer actual user experience including high-level process functionality. This will further lead to the potential employment of sequence to sequence models that will generate actual high-level process functionality – rather than the basic one generated by our existing models.

VI. Bibliography

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

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

[3] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *eprint arXiv:1512.03385*, 2015.

[4] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *eprint arXiv:1610.02357*, 2016.

[5] J. Long, E. Shelhamer and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *arXiv:1411.4038,* 2015.

[6] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556, Computer Vision and Pattern Recognition,* 2015.

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

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

[9] A. Damian and N. Tapus, "Model Architecture for Automatic Translation and Migration of Legacy Applications to Cloud Computing Environments," in *CSCS,* Bucharest, 2017.

[10] K. He, X. Zhang, S. Ren and J. Sun, "Identity Mappings in Deep Residual Networks," *arXiv:1603.05027,* 2016.

[11] S. Zagoruyko and N. Komodakis, "Wide Residual Networks," *arXiv:1605.07146,* 2017.

[12] S. Vogel, C. Schorn, A. Guntoro and G. Ascheid, "Efficient Stochastic Inference of Bitwise Deep Neural Networks," *Workshop on Efficient Methods for Deep Neural Networks at Neural Information Processing Systems Conference 2016, NIPS 2016, EMDNN 2016,* 2016.

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

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

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