Workflow Languages for Convolutional Neural Networks

Jony Van Puymbroeck

jony.vanpuymbroeck@student.uantwerpen.be

Abstract

Creating a Convolutional Neural Network can be very complex, especially for people who have no programming experience. A tool to generate the project code would make it more likely for non-experts to give Deep Learning a try. Having a Visual Workflow Language provides the users to easily create their model and tweak the parameters in a plug-and-play environment. These different parameters determine the viability of your project, so thoughtfully setting these is advised. A user will often have a dataset that is too small to generate usefull models. Therefore, Data Augmentation code could be provided by the tool if parameters concerning the data are given. This paper discusses a possible solution and assesses the risks.

Keywords: Workflow Languages, Convolutional Neural Networks, CNN, Machine Learning, KERAS, Python, Tools, Visual Workflow Languages, Data Augmentation, AToMPM, metaDepth, Persimmon

1. Introduction

Working on my Masters' Thesis - *Computer-aided diagnosis: Determining the immune phenotype using Deep Learning* - I noticed the complexity of learning how to work with CNN's is high. It involves learning to work with the **KERAS**[9] library which requires decent knowledge of the **Python**[8] programming language. Furthermore, one is required to learn all about the different layers of a **CNN**. Setting up the entire project directory with the working code, a GUI, ... takes quite some time, even for someone with a decent amount of programming knowledge.

Since I'm doing my research for a company with no expertise on Machine Learning (and in this case more specifically Deep Learning), I will need to provide them with a workflow document giving them a way of (re-)generating the project while all they would need to do is set a minimal amount of parameters. This would optimize their experience with creating deep learning projects. Therefore, in this paper I aim to justify **the need of a workflow language** which would allow **code generation** of the model and **GUI** and possible extra's that might be needed in a project.

A Workflow is an **MDE-based** solution where the user defines a workflow that can be parametrized at run-time and executed. This Workflow is a **DSL** for defining activities that can be performed in MDE tools. It consists of an orchestrated and repeatable pattern of business activity enabled by the systematic organization of resources into processes that provide services, or process information. It can be depicted as a sequence of operations or one or more simple or complex mechanisms. From a more abstract or higher-level perspective, workflow may be considered a view or representation of **real work**. The flow being described may refer to a service or product that is being transferred from one step to another[6].

Building a **Convolutional Neural Network** requires programming skills and decent insights in the relevant Deep Learning algorithms. Creation of the project code using a **Visual Workflow Language** (VWL) could make the process easier for the user. In this paper, we will look at a possible solution to create a model, capable of generating the project code to train a CNN as well as discussing why this is relevant.

Very important to note is that by using a Visual Workflow Language, the user would no longer need to have any programming skills. Code is generated as is and can be executed using the command line. This provides users a way of creating CNN's without the need of knowledge of **KERAS** or **Python**.

Using a VWL allows us to perform analysis on the created model. Our model would include the possibility of generating Data Augmentation code, Model generation code, ... Data Augmentation is very dependent on the data we have: Are all images of the same size, how many pixels should the width/height-shift at least be, ... If these parameters are not set correctly, data augmentation might generate erroneous data (duplicate images, stretched images, ...).

Section 2 clearly explains the objectives and the things we can't guarantee, Section 3 gives more information about CNN's, Section 4 gives a brief introduction into how the (training) data should look, Section 5 talks about what Transfer Learning is, and how we will use it, Section 6 explains how the workflow will look and its different components, Section 7 gives a short introduction to the tools that will be used, Section 8 has a Risk Analysis, Section 9 gives an introduction to related work and Section 10 concludes the research project.

2. Objectives

I will attempt to identify **the commonalities** in the development of a CNN and summarise those in the form of a **specialised workflow language**. This language would consist of a pipeline of elements which have settable parameters. After the creation of the model, it would then be possible to use **metaDepth**[4] to generate the needed and requested files. You could then create your entire project without programming expertise. The parameters you should set require knowledge of Convolutional Neural Network layers though, but these could be briefly explained in an additional file.

This workflow language **will not ensure viable model creation** after generating the project. That still depends on the parameters used, the data you are using and any preprocessing you might do or have done with the data. Certain **analysis** of the model will ensure your data is usable though. Training a model still requires decent insight in how Deep Learning works. The Visual Workflow Language can just be used as a tool for people with no expertise in programming.

3. Convolutional neural networks

A Convolutional Neural Network (CNN) is a biologically-inspired variant of **multilayer perceptron**[1] [1]. It is a class of deep, feed-forward artificial neural networks that has been successfully applied to analyzing visual imagery. It allows us to generate a neural network which has been proven effective in areas such as image recognition and classification. The CNN assumes the inputs to be images and has an architecture designed to take

[1]Multilayer perceptron: A network of simple neurons, called perceptrons. [3]

advantage of the structure of images. Such a CNN is a sequence of layers, such as: *The convolutional layer, the ReLU layer, the Pooling layer* and *the FC layer*[2]. An example of these layers, applied on an example image, can be found in figure 1.

A very useful example of how a Convolutional Neural Network is applied can be found in the tutorial for a **Kaggle** competition which aims to classify pictures of cats and dogs.[12]

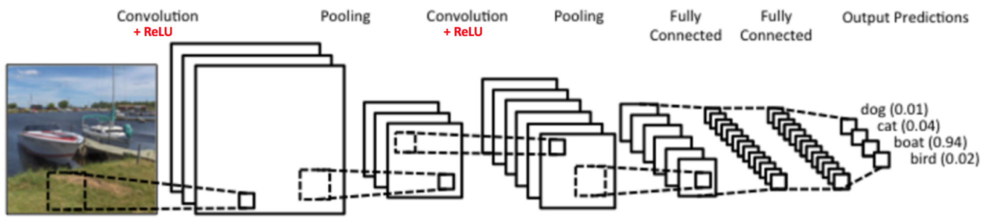

Figure 1: The layers of a CNN applied on a picture.[11]

4. Data

Any data the model would be trained on has to be an image. These images need to be labeled by their correct class (supervised learning[5]) and put into a folder hierarchy for training purposes. The model can then be trained using this folder structure.

5. Transfer Learning

Transfer learning or inductive transfer is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks[13]. In practice, **very few people train an entire Convolutional Network from scratch**, because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pretrain a ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the ConvNet either as an initialization or a fixed feature extractor for the task of interest [14].

Using transfer learning, non-experts are enabled to **generate very useful results without the need of creating a Convolutional Neural Network from scratch**. Instead, they are allowed to use a pretrained network. We will use InceptionV3[16] and possibly NASnet[15], which are pretrained models on the ImageNet[17] database. To illustrate the complexity of such a Convolutional Neural Network, figure 2 depicts the layers of the inception model.

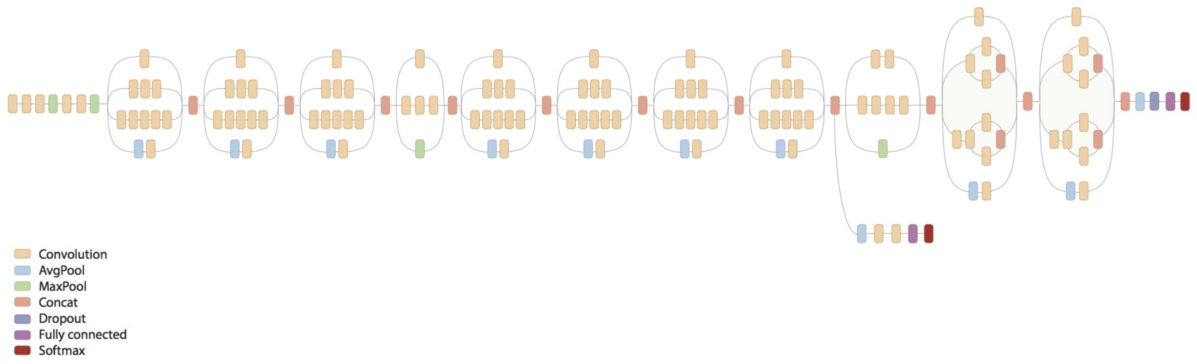

Figure 2: The layers of the CNN of the Inception model.[16]

6. Workflows

The workflow can consist of different blocks, which can be set by the user in order to generate the desired project code.

6.1. Data Augmentation

6.1.1. Required Folder Hierarchy

The data 4 will need to have been put in a specific folder hierarchy. For the kaggle example[12], the hierarchy would look like:

```
/data/
> dogs/
 —— dog.1.png
 —— dog.2.png
   ...
> cats/
 —— cat.1.png
 —— cat.2.png
   ...
```

This is not automatable and requires input from the user. Possibly, the model could generate python code which would make command line execution of a script possible which could generate all filenames of the images (dog.1.png, ...). This is automatable and can divert hours of work into a simple script.

6.1.2. Expanding the data set size

The Data Augmentation block would enable augmentations like:

- Generating smaller images from our original images. Take for example a 2000x2000 image. You would then generate 900 images out of a single image by taking an image of size 1700x1700 out of this 2000x2000 image. This by shifting 10 pixels in width/height in each iteration. This results in images with dimensions [0,0,1700,1700], [10, 0, 1710, 1700], ..., [0, 10, 1700, 1710], ... Further augmentations would **only be performed on these 1700x1700 images** in order to avoid generating duplicate images.
- Rotate the image 90 degrees, up to three times (resulting images would be 0, 90, 180 and 270 degrees rotated to the right)
- Flip the resulting images vertically.

For the first kind of augmentation, we must note that the amount of pixels to shift with (in the example: 10 pixels) has to be bigger than a certain value. To determine this value, we must notice that the pretrained model of which we will **transfer-learn**5, has a certain input image resolution, which means the input data will be scaled up/down to that resolution. For InceptionV3[16], this resolution is **299 x 299**. In our example of 1700x1700 images, this would mean we need to shift at least

$$\frac{299}{1700} < 6 pixels$$

since every 6 pixels will be used to become 1 pixel in the case we downscale. The model could perform analysis on the parameters to check for such necessities.

6.1.3. Future perspectives

Color correction could be added to the VWL, but as it is not required for the work I perform for my thesis, this will be considered a future perspective.

6.1.4. Dividing into Training, Validation and Testing set

When the actual augmentation is finished, we would like the dataset to be split up into a Training, Validation and a Testing set. Normal percentages would be 60% of the data would go to the Training set, 20% to the Validation set and 20% to the Testing set. The resulting folder for the example of the dogs and cats structure would be:

```
/augmented_data/
> train_dir/
-> dogs/
-> cats/

> val_dir/
-> dogs/
-> cats/

> test_dir/
-> dogs/
-> cats/
```

6.2. Model generation

The model generation block takes a **variety of parameters**. These include:

- Amount of epochs: The amount of times to iterate over the entire dataset. An additional epoch usually increases the accuracy of the models' predictions.

- The transferlearning optimizers[10]. The code uses two optimizers.
- The optimizer learning rate.
- Directory of training data.
- Directory of validation data.
- Directory of model creation plots.

The generated code can then be executed to generate the model.

6.3. GUI

I created a **GUI** using **PyQT** which can be used on the testing set, to manually test the accuracy of the model. You can use the GUI to test a **single images' accuracy**, or execute the prediction algorithm on **an entire directory full of images**. An image of this GUI can be found in figure 3.

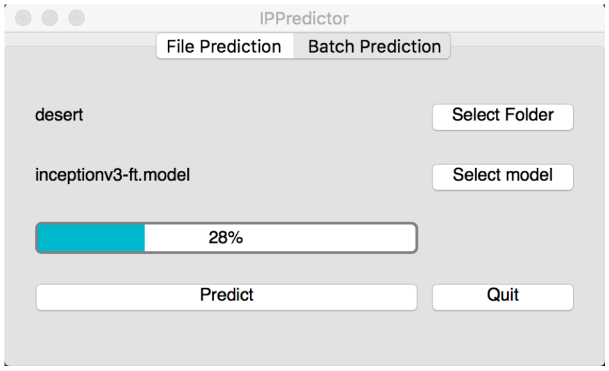

Figure 3: The gui for prediction model execution.

7. Tools

7.1. AToMPM

AToMPM[18] is a research framework from which you can generate domain-specific modeling tools. It is an open-source framework for designing DSML environements, performing model transformations, and manipulating and managing models. In this case, it will be used to generate the Visual Workflow Language entirely.

7.2. metaDepth

metaDepth[4] is a framework for deep meta-modelling. **AToMPM** allows for exporting generated models to metaDepth. metaDepth in turn allows for this exported model to be used as input for template coding, which can generate our needed python code.

7.3. Extra

Furthermore, we will use:

- Python: The programming language[8]
- KERAS: Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.

8. Risk analysis

8.1. Usability

One of the more significant risks is usability. Will the user know what to do with the Visual Workflow Language? Will they understand what they are actually working with? To answer these questions, you need to talk to your users. You need to discuss with them how they would see such a language. Luckily, **I have access to a lot of non-experts who would be potential users of the VWL**. A decent study would reduce this risk drastically. Having the users test the software would be a meaningful study for the evaluation as well.

8.2. Ignorance

A lot of people hear about Machine Learning or Deep Learning, but actually have no clue about how to use it. It takes a gigantic leap of faith to start such a project. They don't know what to expect and failure is immenent at all times. Often, one can not find time to start their research into this due to pressure at their work or research. Therefore, a lot of people hear about these algorithms that can classify data, but it still sounds like science-fiction. Every tool you create for potential users is therefore subject to skepticism. Providing people with the correct tools that are easy to use and just plug-and-play, might trigger them to actually try it. Different use cases could be worked out and published, like my exploratory Thesis, to get the attention it deserves and needs.

9. Related work

Persimmon is a visual programming interface that leverages scikit-learn to provide a drag and drop interface for developing Machine Learning and Data Mining pipelines. It is based on the dataflow programming principles, giving the user a functional visual language with a type safety system that checks connections at write time, non-strict evaluation, task parallelization, and execution visualization. It had been evaluated by participants on a three-task form, overall receiving good reviews, being praised by the use of colors to indicate types, consistent design, easy to navigate and shallow learning curve[7].

The motivation for Persimmon is the lack of programming skills of users of the Machine Learning algorithms. These users are mostly experts on Maths, Physics, Electric Engineering, Statistics, ... They aim to provide the user with feasibility study functionality, ease to use in the form of a drag and drop interface and a useful learning tool (for programming and Machine Learning algorithms).

10. Conclusion

We can conclude that we want to create a tool which can generate CNN models for a user without the need of programming skills, capable of doing analysis where possible. The user experience is of fundamental importance. Optimizing this will make tools like this more accessible for the public. The models can be tweaked in many ways using the different parameters of the model. Users will still be expected to learn about the way Deep Learning and Convolutional Neural Network optimizers work in order to produce a viable model.

References

[1] Deeplearning.net *Convolutional Neural Networks (LeNet).* http://deeplearning.net/tutorial/lenet.html deeplearning.net, LeNet, accessed: 2017-09-19.

[2] cs231n *CS231n Convolutional Neural Networks for Visual Recognition.* http://cs231n.github.io/convolutional-networks/#conv cs231n, accessed: 2017-09-19.

[3] Antti Honkela *Multilayer perceptrons.* https://www.hiit.fi/u/ahonkela/dippa/node41.html Antti Honkela, 2001-05-30, accessed: 2017-09-19.

[4] metaDepth *a framework for multi-level meta-modelling* http://metadepth.org/

[5] Jason Brownlee *Supervised and Unsupervised Machine Learning Algorithms* https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ Jason Brownlee, 2016-03-16, accessed: 2017-12-11.

[6] Gamboa M., Syriani E. *Using Workflows to Automate Activities in MDE Tools* http://www.springer.com/cda/content/document/cda_downloaddocument/9783319663012-c2.pdf?SGWID=0-0-45-1616886-p181095235 Universite de Montreal, Montreal, Canada, accessed: 2017-12-11.

[7] Garcia A. *Persimmon, A Visual Dataflow Language for Machine LEarling* http://eprints.ucm.es/44618/1/Persimmon.pdf Complutense University of Madrid, Alvaro Bermejo Garcia, 2017-06-16, accessed: 2017-12-11.

[8] Python Software Foundation (2001) https://www.python.org/ , accessed: 2017-12-13.

[9] Keras (2015) https://keras.io/ , accessed: 2017-12-13.

[10] Keras (2015) *Usage of optimizers* https://keras.io/optimizers/ , accessed: 2017-12-13.

[11] ujjwalkarn *An Intuitive Explanation of Convolutional Neural Networks* https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ The Data Science Blog, ujjwalkarn, 2016-08-11, accessed: 2017-12-13.

[12] sub-subroutine *Cats and dogs and convolutional neural networks* http://www.subsubroutine.com/sub-subroutine/2016/9/30/cats-and-dogs-and-convolutional-neural-networks Subsubroutine, 2016-09-30, accessed: 2017-12-13.

[13] West J., Ventura D., Warnick S. (2007) *Spring Research Presentation: A Theoretical Foundation for Inductive Transfer* https://web.archive.org/web/20070801120743/http://cpms.byu.edu/springresearch/abstract-entry?id=861 Brigham Young University, College of Physical and Mathematical Sciences. Archived from the original on 2007-08-01. Retrieved 2007-08-05. Accessed: 2017-12-13.

[14] Transfer Learning http://cs231n.github.io/transfer-learning/ cs231n, accessed: 22017-12-13.

[15] Nealwu *TensorFlow-Slim NASNet-A* https://github.com/tensorflow/models/tree/master/research/slim/nets/nasnet github, accessed: 2017-12-13.

[16] Niyazpk *Inception in TensorFlow* https://github.com/tensorflow/models/tree/master/research/inception github, accessed: 2017-12-13.

[17] *image-net* http://www.image-net.org/ Stanford Vision Lab, Stanford University, Princeton University, 2016, accessed: 2017-12-13.

[18] Syriani E., Vangheluwe H., Mannadiar R., Hansen C., Van Mierlo S., Ergin H., Corley J. *AToMPM Documentation* https://msdl.uantwerpen.be/documentation/AToMPM/ Accessed: 2017-12-13.