Transforming hierarchical images to program expressions using deep networks

Křen, Tomáš
2018

Institute of Computer Science
The Czech Academy of Sciences

Transforming Hierarchical Images to Program Expressions Using Deep Networks

Tomáš Křen

Technical report No. V-1263

December 2018

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: +420 266 053 201, fax: +420 286 585 789,
e-mail: kren@cs.cas.cz

Institute of Computer Science
The Czech Academy of Sciences

Transforming Hierarchical Images to Program Expressions Using Deep Networks

Tomáš Křen[1,2]

Technical report No. V-1263

December 2018

Abstract:

We present a technique describing how to effectively train a neural network given an image to produce a formal description of the given image. The basic motivation of the proposed technique is an intention to design a new tool for automatic program synthesis capable of transforming sensory data (in our case static image, but generally a phenotype) to a formal code expression (i.e. syntactic tree of a program), such that the code (from evolutionary perspective a genotype) evaluates to a value that is similar to the input data, ideally identical. Our approach is partially based on our technique for generating program expressions in the context of typed functional genetic programming. We present promising results evaluating a simple image description language achieved with a deep network combining convolution encoder of images and recurrent decoder for generating program expressions in the sequential prefix notation and propose possible future applications.

Keywords:
deep networks, automatic program synthesis, image processing

[1]Research was supported by the grant of the Czech Science Foundation GA 18-23827S.

[2]Institute of Computer Science, The Czech Academy of Sciences, Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic, E-mail: kren@cs.cas.cz.

1 Introduction

We present a technique describing how to effectively train a neural network given an image to produce a formal description of the given image.

Since the output of the network is a program code, the task falls into the broader category of algorithms performing *automatic program synthesis*. The basic motivation of the proposed technique is an intention to design a new tool for automatic program synthesis capable of transforming sensory data (in our case static image, but generally a "phenotype") to a formal code expression (i.e. syntactic tree of a program), such that the code (from evolutionary perspective a "genotype") evaluates to a value that is similar to the input data, ideally identical.

Unlike other methods which can be used for automatic program synthesis (genetic programming, inductive programming, Monte Carlo tree search techniques, constraint satisfaction problem solvers) which perform a search on the individual space, the neural network approach is unique in that it produces the code directly. We may interpret this as a kind of "artificial intuition".

Our approach is partially based on our technique for generating program expressions in the context of typed functional genetic programming. We present promising results evaluating a simple image description language achieved with a deep network combining convolution encoder of images and recurrent decoder for generating program expressions in the sequential prefix notation.

Input domain I : Inputs are *Images*. We use convolution encoder to parse them. So, if $i \in I$, then i is an *image*.

Ouput domain C : Our neural networ will output a code c. A code $c \in C$ is a syntactic tree of a program expression describing a specific image i. We can evaluate a code c by our hand crafted render function R, such that:

$$c \in C \Rightarrow R(c) = i \in I$$

Render function R : A render function R may be given by a Γ set, we denote this fact by writing R_Γ. Γ is a *library* represented as a collection of symbols, where each symbol s has:

- an implemetation R_s,
- a type τ_s.

What we really want to do is to compute an approximation of R^{-1}: We want to construct an *inverse* of the render function.

Dataset D : D is a collection of (i, c) pairs such that $i = R(c)$, where i is input image and c is the desired output code for the input image i. Colecttion D serves as training and testing dataset for the network. We obtain these data

samples by automatically generating c codes. The generated learning dataset D should obey one of the following *"Occam Razor"* constraints.

Sharp Razor:

- (a) A specific image i is at most once in D,
- (b2) For each $(i, c) \in D$ holds that c is the shortest possible code such that $R(c) = i$.

Rough Razor:
A practical approximation of the *Sharp Razor*:

- (a) same as *(a)* in the *Sharp Razor*,
- (b1) If two codes c_1, c_2 such that $R(c_1) = i = R(c_2)$ are generated for D, keep the *shorter* one (i.e. keep the one described with *less symbols*).

Our current implementation follows the *Rough Razor*.

2 Related Work

Similar task to our task is image captioning to the domain of natural language [1], we use similar network achitecture inspired by theirs deep network with convolution encoder for feature extraction and recurrent decoder (LSTM as recurrent cell) with use of attention mechanism [2]. Our network takes same input (image) but instead of outputing to natural language, our output is to formal language of programming code.

Task of generating programming code by neural network is performed for example by [3] which is addressed from a using different perspective and different output domain of SQL-like programs to query a table.

Very similar image description language is used in [4], but in the context of context-free grammars.

3 Image description language

One of the key features of formal languages is their hierarchical nature. They are perfectly suited for describing objects with hierarchical structure. Hierarchical structure provides natural ways to describe visual self-similarty as we know it from mathematics (fractals) and nature (trees, leaves, snowflakes, feathers, rivers, lightning, lungs, coastlines, mountains, clouds, Nautilus shells, Romanesco broccoli).

An exemplary piece of a hierarchicaly structured object is a tree (the living structure in a forest), with a benefit of being a visually stunning example. Visually speaking, simplified trees are composed of simple parts: branches and leaves. These parts are composed into a hierarchy. Hierarchy is naturally describable

as a tree (the syntactic graph structure). A living tree can by desscribed by a syntactic tree. Their structures overlap each other.

In our experiments we use simple image description language with symbol set capable of describing visually hierarchical structures. It is an intentionally simple, yet general language.

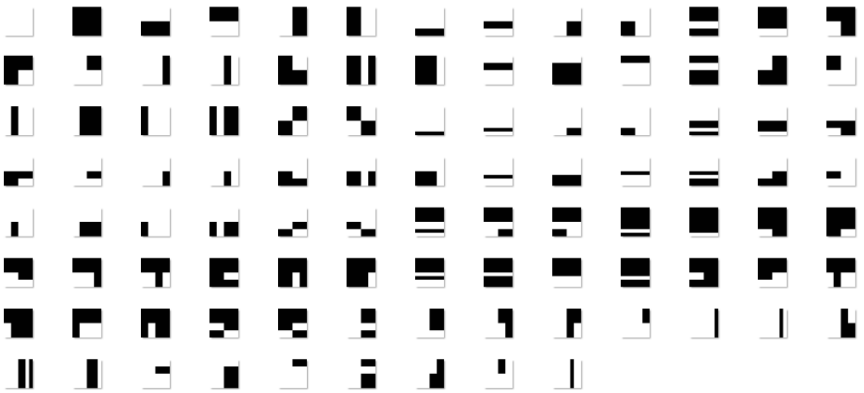


Figure 1: First 100 simplest images in our simple image description language.

If an image canvas size used for rendering is power of two, then this language can describe any B/W image. Fig. 1 shows first 100 simplest images. Fig. 2 illustrates the image to code correspondence of our simple image description language.

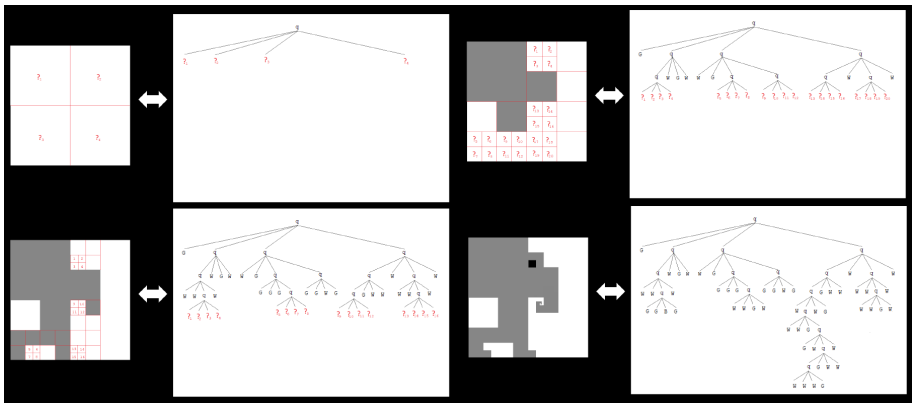


Figure 2: Illustration of image to code correspondence of our simple image description language.

Our symbols are:

- **q** : split to 4 quadrants,
- **h** : horizontal split,
- **v** : vertical split,
- **B** : fill black,
- **W** : fill white.

4 Dataset generation method

We generate program codes for each syntax tree size (i.e. number of program symbols) from 1 to some maxTreeSize (e.g. 25, 13). For each tree size we compute the total number of trees:

- Below some total number of trees limit we generate all trees exhaustively,
- above the limit we use uniform sampling utilizing our tree generating method [5] capable of uniform tree sampling for a given tree size.

Our generating method is ready for typed languages with parametric polymorphism (types with type variables). While generating, we utilize a kind of "Occam's razor"; we keep only the smallest representation of a given image in the dataset. To achive this we use visual image hashing method *pHash* [6].

5 Results

From preliminary experiments we evaluated as a best choice for model a deep network with convolution encoder and recurrent decoder with Gated Recurrent Unit (GRU) used as a recurrent cell. In the decoder we used attention mechanism [1]. The output of the network is a sequence of symbols interpreted as program code in prefix notation.

We trained the nework by a simple method of treating the output code as ordinary text, using BLEU-4 as our training metric metric. But for final evaluation of the result we render the output codes back to images and compare them to the original images using portion of pixels with correct color as the final evaluation metric.

The source code of the project is available at (the core script is located in the *i2c.py* file): *https://github.com/tomkren/TFGPy*

In the project implemetation we utilized *Tensorflow* [7] as a back-end and *NeuralMonkey toolkit* [8] as experiment front-end.

Here we briefly present results of two experiments with this architecture, first with images of size 32×32, second with images of size 64×64.

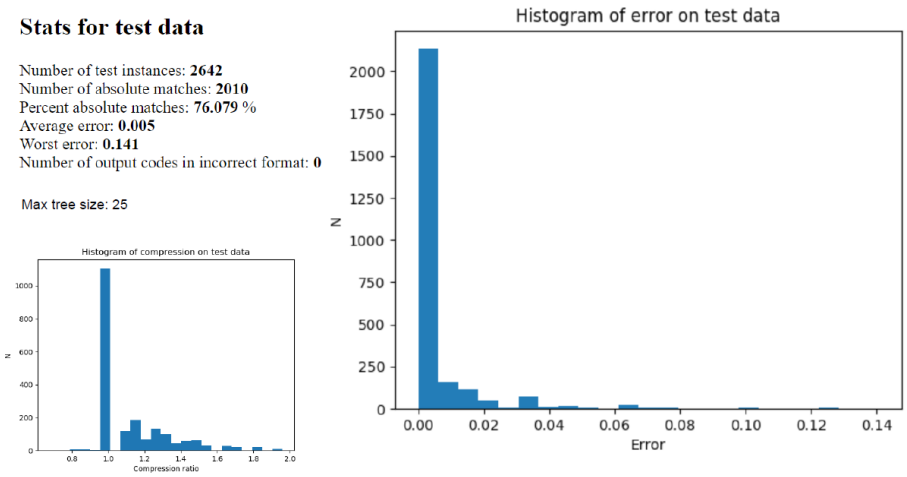


Figure 3: Results summary for test data in experiment with images of size 32×32.

In the first experiment we generated 80642 images of size 32×32 (with max tree size 25), using 78000 images as training data set and 2642 images as test data set for evaluation of the results. From 2642 images, 2010 images (76%) were fully correct matches. Average error on the final evaluation metric was 0.005, worst error was 0.141. Nice property of this model is that it grasps the output format; in the first experiment all outputs for the test set instances had correct format (i.e. represented correct prefix program). Fig. 3 summarizes results for test data set of the first experiment. Fig. 7 shows random examples of original test data and their result codes and images for this experiment. Fig. 9 shows same information for test data images with maximum error. Another nice property of the model is that it performs compression of the output codes when compared with the original input codes (see fig. 3), even though the generating method used the *"Rough Razor"* to enhance original codes shortness.

In the second experiment we generated 52525 images of size 64×64 (with max tree size 13), using 42020 images as training data set and 10505 images as test data set for evaluation of the results. From 10505 images, 8478 images (80.7%) were fully correct matches. Average error on the final evaluation metric was 0.015, worst error was 0.625. Number of output codes in incorrect format is 29 (thus not zero but very low). Fig. 4 summarizes results for test data set of the second experiment. Fig. 8 shows random examples of original test data and their result codes and images for this experiment. Fig. 10 shows same information for test data images with maximum error. In this experiment the compression property was absent, we think this is because of the smaller max tree size, so that majorit of the generated images was already very compressed.

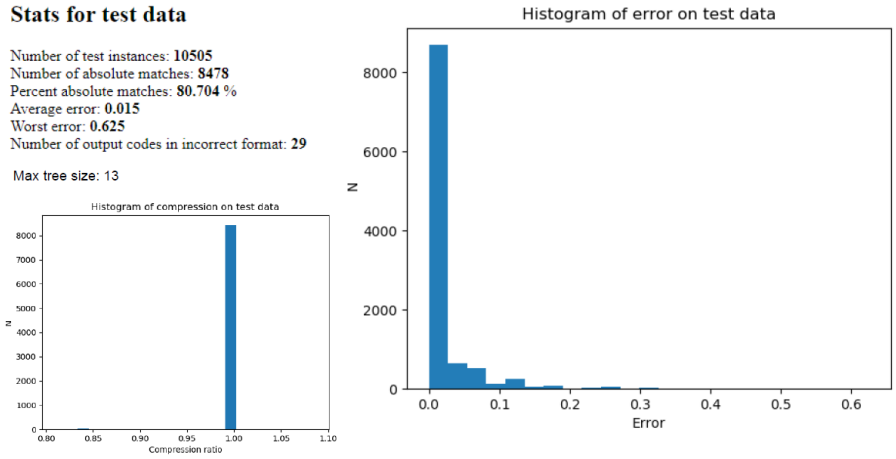


Figure 4: Results summary for test data in experiment with images of size 64×64.

6 Conclusion and Future work

The presented experiments on the simple image description language have promising results. However, the language is very primitive, so experiments on more complex and interesting languages must be perform to make the results more conclusive. We plan to significantly expand the set of symbols in the language with:

- *Symmetry* operations,
- operations to support easy creation of *fractal structures* by functional folding,
- *Shader* approach to canvas filling.

And we also plan to add more interesting image description languages which we have been working on, namely:

- Terrain Generating (fig. 5 shows some examples),
- Celullar automata ornaments (fig. 6 shows an example).

Currently we are working on implementation of *Recursive Zoomer*, which will enable to use a trained model for a specific image size on much bigger images, exploiting the hierarchical nature of the description language.

The source code of the project is available online at:

https://github.com/tomkren/TFGPy

The core script is located in the *i2c.py* file.

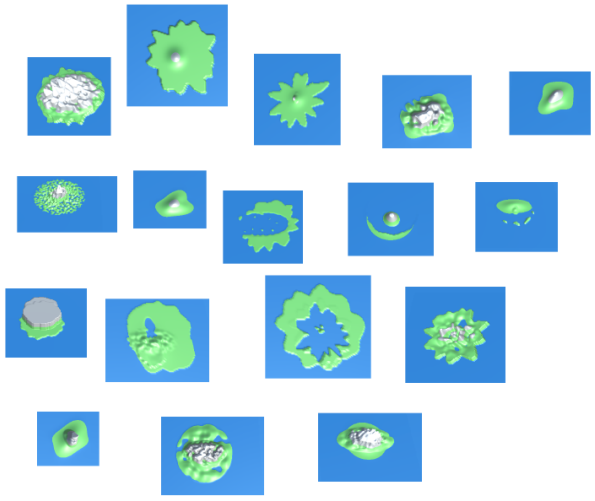


Figure 5: Examples of *"Islands"* image description language we are working on.

References

[1] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[3] Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. *CoRR*, abs/1511.04834, 2015.

[4] Michail I Schlesinger and Václav Hlaváč. Context-free languages, their two-dimensional generalisation, related tasks. In *Ten Lectures on Statistical and Structural Pattern Recognition*, pages 479–505. Springer, 2002.

[5] Tomáš Křen, Martin Pilát, and Roman Neruda. Automatic creation of machine learning workflows with strongly typed genetic programming. *International Journal on Artificial Intelligence Tools*, 26(05):1760020, 2017.

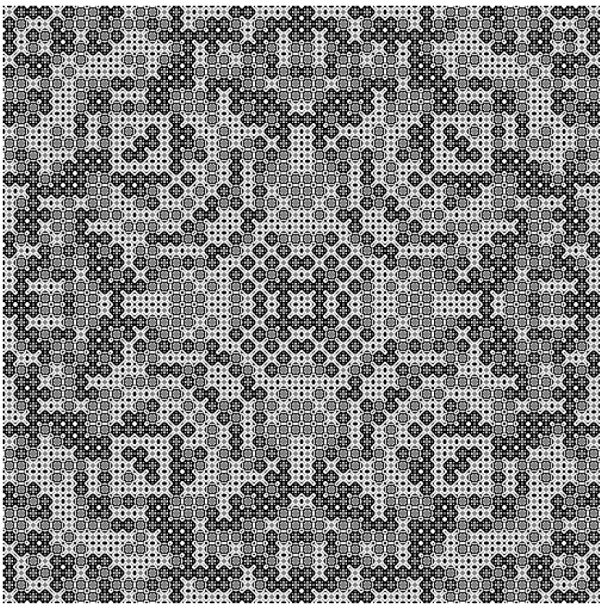


Figure 6: Example of an image generated by the *Celullar automata ornaments* image description language we are working on.

[6] EVAN Klinger and DAVID Starkweather. phash–the open source perceptual hash library. Technical report, accessed 2016-05-19.[Online]. Available: http://www. phash. org/apps, 2010.

[7] M Abadi, A Agarwal, P Barham, E Brevdo, Z Chen, C Citro, GS Corrado, A Davis, J Dean, M Devin, et al. Tensorflow: large-scale machine learning on heterogeneous distributed systems. arxiv preprint (2016). *arXiv preprint arXiv:1603.04467*.

[8] Jindřich Helcl and Jindřich Libovický. Neural Monkey: An Open-source Tool for Sequence Learning. *The Prague Bulletin of Mathematical Linguistics*, (107):5–17, 2017.

file	in	out	raw output / input prefix	error
00065313.png			v v W h W v v W h v W B B W h v W W v v W v q W B h h h h B W W W B B B W	0.015625
00051089.png			v B h q v B h W B B B W B q B q q B W h B B v B h B B B B W B B	0
00010961.png			v B h q W h B W B W W v B h q W h B W B W W	0
00011386.png			v B q B W v W v W B B v B q B W v W v W B B	0
00006861.png			h B v h h B W h W B B h B v h h B W h W B B	0
00035148.png			h h B h B h B h B v W B W h h B h B h B h q B W B W B W	0.03125
00019001.png			h h v v W B W h B W W h h v v W B W h B W W	0
00038128.png			h B h W h B q v B W W W B h B h W h h B B q v B W W W h B B	0
00015228.png			h h h B W v W B h B W h h h B W v W B h B W	0
00017012.png			q v B v B W B B h B W q v B v B W B B h B W	0
00031831.png			v B v B h h v h B W W W B q B v B q h B W W W h W W B B	0
00002717.png			h W h W h v W B h W B h W h W h v W B h W B	0
00036468.png			h v q B B W v B h B W v B W W q h B q W B W v B v W W v B W W W	0
00073335.png			h v W v B h h B q W v B W B h B W B B h v W h q B B v h B B B h v W v v W B B v B W B B	0.009765625
00074295.png			q q B q h B W W W B W B B W B q q B q h B W W W h B h B v B q B B B W W B B W B	0
00018614.png			h h v W B h h B W B B h h v W B h h B W B B	0

Figure 7: Random examples of original and result test data images for experiment with images of size 32×32.

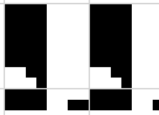
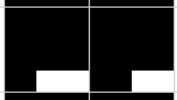
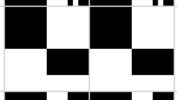
file	in	out	raw output / original input	error
00051705.png			output : h W v v B v B W h W v B W original : q W W v B v B W v h W B W	0
00037973.png			output : v W q h W q B W W B W W W original : v W h h W v q B W W B W W	0
00042623.png			output : v h B h B v W q B B W B W original : q B W h B v W h B v W B W	0
00019608.png			output : h h v B h W v W B W B original : h h v B h W v W B W B	0
00005215.png			output : h B h B h W v B h W B original : h B h B h W v B h W B	0
00024679.png			output : v v q h W B B B W W W original : v v q h W B B B W W W	0
00042708.png			output : v B h h W h h B W h B W B original : q B h W h h B W h B W B B	0
00005392.png			output : h B h B v B W original : h B h B v v B v B W W	0.03125
00034496.png			output : v B h B v W v h W v B W B original : q B B B q W W W v v B W B	0.03125
00013229.png			output : q B W W h B h h B W W original : q B W W h B h h B W W	0
00010821.png			output : v B h h h v W B B B W original : v B h h h v W B B B W	0
00027914.png			output : q h h B h W B W B B W original : q h h B h W B W B B W	0
00028328.png			output : v q q W W v B W W W B B B original : q v v v W h W B W W B B B	0.03125
00041319.png			output : h h W v B h h B W B v B W original : h h W v B h h B W B v B W	0

Figure 8: Random examples of original and result test data images for experiment with images of size 64×64.

file	in	out	raw output / input prefix	error
00066843.png			v B v W v v q B W B W B h W B v v B B v W v v h v q v B B B B B W B h W W B	0.140625
00045041.png			v q h W B B B W v v v W B W W v v h h W B B h B W v v h W W v B W W	0.125
00035537.png			h q h v B W B W W B W h q q B W h W W W h B W W B W	0.125
00047024.png			v q B v B W h B W q W B W W B v q B v B W h B W q W B W W h h W B B	0.125
00026414.png			q W v B v B W B B q W v B v B W h W B B	0.125
00037066.png			h W h v h W h W v h B h W B W B B q W W h h W h W q B W W W B h W B	0.125
00072860.png			v q B W q W W v B v W B B v W B W v q v B B W v h W q B v B W W W h W B v B v W B W	0.11328125
00079253.png			h q v B h W B B W h B W h B h W q B W v B W B h h v h v B W h B B B W h B q W W q W h B B B W B	0.109375
00043935.png			h h B h q B h W B v B W h W B W W h h B q v h h B B W B h h W B W B W W	0.109375
00068310.png			v B h q h q W v W q B W W B W B W B W B B v B h q h v W h h v B B W v v W B W B W W B B	0.103515625
00072402.png			q h W B v v h B h W B W v W h B B W B h v h W B q v B W v B W v v h W B W W v B B v W B	0.1015625
00074992.png			v v h h q W B W h W v B W W W v W v W B B v h v q W q h W W B h W B W W W v v B W W W h B B	0.1015625
00080455.png			h h W h q B h W B B h W B q v h W B B W B W W h h W h q W h v B W B B W h h v q B B W B W B W W	0.1015625
00074721.png			v v v W h h B W h W v h W v h W B W W W B v q W h B W v h W v W v W q h W W W B W W W v B B	0.099609375
00054394.png			v q h h B W B v B W W B W v q h h B v v W W W W v B h v B B W W B W	0.09375
00065947.png			v B v v B v W h h B W h h B W W B v B v v B h v W h B v B W W v h W h B v W B B	0.09375
00024515.png			v v v h W B h B W W B v v v h W h B W B W B	0.09375
00040370.png			h B h W h h B B h B h v B W B h B h W h h W B q B B q B W W B B	0.09375

Figure 9: Result test data images with maximum error for experiment with images of size 32×32.

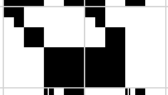
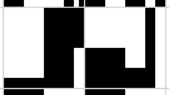
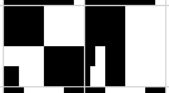
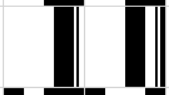
file	in	out	raw output / original input	error
00034118.png			output : v v h h v W B B B W v B original : v h v q W B B B W B v W B	0.625
00035879.png			output : q q q B B W B W W B W B original : q q q B B W B W W B W W B	0.5
00048766.png			output : v W v v v v B v W B W B original : v W v v v v B v B W B W B	0.46875
00044452.png			output : v h W v B W v B v W B original : v h W v B W v W v B v W B	0.4375
00039738.png			output : v h W B v h W h W B v B W original : q W B h W h W B v B v B W	0.4375
00034790.png			output : q B W v v h B v B W W B original : q B W h W v v B v B W W B	0.4375
00043187.png			output : h q v B W v W B B v W B B original : h q v B W v W B W B h W B	0.4375
00047423.png			output : q h q W B h W B W B W W B original : v h h q W B h W B W W W B	0.375
00038183.png			output : v W v B v W v v B W B original : v W v v W B v B v v W B W	0.375
00047075.png			output : v v B h W B q W B B v W B original : v v B h W h W B q B B W B	0.375
00052093.png			output : h h W v B v B W v W B original : h h W v B v B W v B v W B	0.375
00035360.png			output : h q h W B q W B B B W B B original : h q v W h W B h B W B W B	0.375
00050400.png			output : h v B W h h W h B W h W B original : h v B W h h h B W W h B W	0.375
00012691.png			output : v v W B v v v B W W B original : v v W B v v B v B W W	0.375

Figure 10: Result test data images with maximum error for experiment with images of size 64×64.