To be presented at *IEEE Multimedia Signal Processing Workshop (MMSP)*, Vancouver, BC, Aug. 2018

De-sketching

Lior Bragilevsky
School of Engineering Science
Simon Fraser University
Burnaby, BC, Canada
lbragile@sfu.ca

Ivan V. Bajić
School of Engineering Science
Simon Fraser University
Burnaby, BC, Canada
ibajic@ensc.sfu.ca

***Abstract*—Many software applications exist for plotting graphs of mathematical functions, yet there are none (to our knowledge) that perform the inverse operation – estimating mathematical expressions from graphs. Since plotting graphs (especially by hand) is often referred to as "sketching," we refer to the inverse operation as "de-sketching." As the number of mathematical expressions that approximate a given curve can be quite large, in this demo we restrict our attention to polynomials, and present a deep model that performs de-sketching by finding the best second-degree polynomial to fit the curve in the input image. Currently, our trained model is able to provide reasonably accurate estimates of polynomial coefficients for both synthetically-generated and hand-drawn curves.**

I. Prototype Description

Determining an equation for a graph would usually involve reading off several points from the curve and then inputting those values into an appropriate software for curve fitting. In this work we developed and trained a deep neural network to allow users to automatically "de-sketch" a graph of a function and produce a mathematical expression that approximates the function. Since the number of mathematical expressions that could approximate a given curve is quite large, we restricted our attention to second-degree polynomials over $[-5, 5]$. Our model finds the best polynomial of the form $p(x) = a_2x^2 + a_1x + a_0$ to fit the curve in the input image.

To tackle this task, we developed a deep model inspired by the well-known VGG16 architecture. Our model contains 4 convolution-convolution-maxpooling (CCM) blocks, followed by a fully connected layer that reduces to 3 nodes at the output, each producing one polynomial coefficient. As a result, our model generates second-degree polynomial coefficients for a given input image. All convolution filter sizes were 3×3 and the number of filters used per layer doubled from 16 up to 128 as the depth increased. The CCM blocks used "ReLU" activation, whereas the output layers used "tanh" activation to produce real numbers in $[-1, 1]$, which are then scaled by 2 to match the range of coefficient values for which the model is trained, as described below.

The model was trained on $80,000$ synthetically generated polynomial curves, where the coefficients were drawn randomly and uniformly from $[-2, 2]$. Training images were similar to those shown in Fig. 1. The loss function J was the L^1-norm between the true polynomial that generated the curve in the input image, $p(x) = a_2x^2+a_1x+a_0$, and the polynomial induced by the estimated coefficients, $\widehat{p}(x) = \widehat{a}_2x^2+\widehat{a}_1x+\widehat{a}_0$:

$$J = \int_{-5}^{5} |p(x) - \widehat{p}(x)| \, dx. \tag{1}$$

This work was supported in part by the NSERC Grant RGPIN-2016-04590

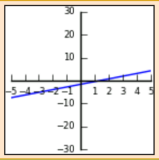
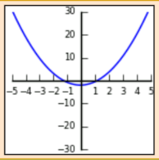
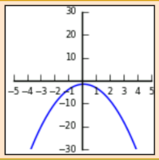
Input Image			
Ground Truth	$1.17x - 1.61$	$1.34x^2 + 0.15x - 1.96$	$-1.93x^2 + 0.41x - 1.58$
Prediction	$-0.01x^2 + 1.18x - 1.62$	$1.34x^2 + 0.16x - 1.96$	$-1.89x^2 + 0.41x - 1.59$

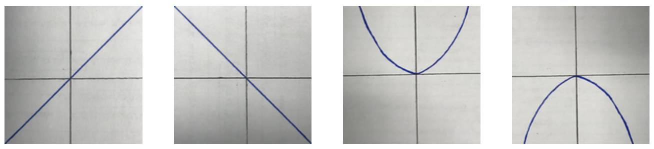
Fig. 1. Synthetic test image prediction results

The model was allowed to train for up to 250 epochs. The loss function was continuously monitored for $20,000$ randomly chosen images from the training set and the model's weights were saved once it reached a minimum value. If the loss function did not decrease from the minimum for 5 consecutive epochs, the training procedure was stopped.

Fig. 1 illustrates our model's predictions for sample input synthetic test images. The predicted polynomials are very close to the ground truth in each case, with coefficient errors mostly in the second decimal place.

Fig. 2 shows the results obtained on hand-drawn images. In these cases, both the axes and the curves were drawn by hand, and the images were taken by a smart-phone camera. In these images there are no axes ticks, which makes it impossible to accurately determine the axes intercept points. Despite these challenges and non-uniform lighting, the model gets the polynomial degree right, and even produces reasonable first- and second-degree coefficients.

We plan to demonstrate the system at the conference. The users will be able to draw a curve on a piece of paper, take a photo of the drawing and feed it to the system, which will then produce the estimates of the best-fit second-degree polynomial.



0.06, 1.60, 0.02   -0.02, -1.69, -0.13   1.61, -0.15, 0.02   -1.58, -0.69, -0.14

Fig. 2. Hand-drawn test image results (Coefficients displayed: $\widehat{a}_2, \widehat{a}_1, \widehat{a}_0$)