Detecting Motion Blur in Images

Karan Varindani **Wenyang Zhang** **Sibo Zhu**

Abstract

Our project aims to estimate motion blur from a single, blurry image. We propose a deep learning approach to predict the probabilistic distribution of motion blur at the patch level using a Convolutional Neural Network (CNN).

1 Our approach

We approached the problem by slicing 100 images into 30x30 patches, and applied our own motion blur algorithm to them (with a random rate of 0.50). We then labeled the blurry and non-blurry patches with 0s and 1s (0 for still, 1 for blurry), and loaded the modified images in as our training data.

1.1 Generating the training data

We generated the training data using images from the Pascal Visual Object Classes Challenge 2010 (VOC2010) data set. Our work was done in Python using the PIL, numpy, opency, and os libraries.

Once we had the original images from Pascal, we had to modify them to fit our needs. We needed to have 100 images, each partially blurred and with a corresponding matrix indicating which part of the image is blurred.

We achieved this by:

1. Making a blurred copy of the original image.
2. Cutting both images (original and blurry) into 30×30 patches.
3. Creating a 2D List in Python of size 30×30, to represent each image patch We initialize each element to 0 (to represent non-blurry).
4. Picking half the patches from the list and marking them as 1 (to represent blurry).
5. Putting the final image together to get a partially-blurred, qualifying image (and its corresponding matrix).
6. Saving the image as "n.jpg" (where n is the serial number of the image), and adding the matrix to a list (to form a 3D 'list of lists') containing the matrices of all the image.

We repeat the above for all 100 images, until we end up with a folder containing partially-blurred images "0.jpg" through "100.jpg", and a 3D list (named "labels") that contains 100 matrices. This lets us access the matrix for image "31.jpg", for example, by querying for "labels[31]".

2 Learning the Convolutional Neural Network (CNN)

Once we had the prepared images, **we loaded them into our training set.** We ran into a problem loading the images into a numpy array, where our images were for the form (30,30,3), while the Keras.Conv2D layer required input to be of the form (3,30,30). We solved this by using the numpy.swapexes() function to alter the images' shape in order to fit the convolutional layer.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

We then apply the CNN learning model. First, we apply a Convolution2D layer with 7×7 filters, followed by a ReLU function. The Conv layer's parameters consiste of a set of learnable filters. Each filter is small spatially, but extends through the depth of the input volume.

During the forward pass, we slide each filter across the width and height of the input volume and compute the dot products between the entries of the filter and the input at any position. ReLU is the rectifier function- an activation function that can be used by neurons, just like any other activation function. A node using the rectifier activation function is called a ReLU node. ReLU sets all negative values in the matrix x to 0, and all other values are kept constant. ReLU us computed after the convolution, and thus a nonlinear activation function (like tanh or sigmoid).

After that, **we add a MaxPooling2D layer** with a pool size of 2×2. MaxPooling is a sample-based discretization process. The objective is to down-sample an input representation, reducing its dimensionality and allowing for assumptions to be made about features contained in the binned sub-regions.

We then **add a Dropout layer** with dropout rate of 0.2, which makes out learning process faster. Dropout randomly ignoring nodes is useful in CNN models because it prevents interdependencies from emerging between nodes. This allows the network to learn more and form a more robust relationship. We then do the 'Conv2D, ReLU, MaxPooling2D, Dropout' circle again. Finally, we add a fully-connected layer with ReLU, and then softmax the result. Softmax is a classifier at the end of the neural network — a logistic regression to regularize outputs to a value between 0 and 1.

We set our model's learning rate to be 0.01**.** This might generally be too big, but we made this decision for the sake of brevity - it was the fastest way to show a result. We chose a batch size of 126 (because we had large training data). We also chose Adam as our optimizer as it's the most efficient optimizer for our model.

After training with 100 epochs, **we had testing accuracy of 92%**, which is a very optimal rate for our model. Our training model is saved in an HDF5 file, "motionblur.h5".

References

[1] Jian Sun, Wenfei Cao, Zongben Xu, Jean Ponce. Learning a convolutional neural network for non-uniform motion blur removal. CVPR 2015 - IEEE Conference on Computer Vision and Pattern Recognition 2015, Jun 2015, Boston, United States. IEEE, 2015, .

[2] "Visual Object Classes Challenge 2010 (VOC2010)." The PASCAL Visual Object Classes Challenge 2010 (VOC2010), PASCAL, 2010, host.robots.ox.ac.uk/pascal/VOC/voc2010/.