Single-frame Regularization for Temporally Stable CNNs

Supplementary material

Gabriel Eilertsen[1] Rafał K. Mantiuk[2] Jonas Unger[1]
[1]Dept. of Science and Technology, Linköping University, Sweden
[2]Dept. of Computer Science and Technology, University of Cambridge, UK
{gabriel.eilertsen, jonas.unger}@liu.se rafal.mantiuk@cl.cam.ac.uk

A. Loss formulation

In order to make this document more self-contained, we start by listing the different regularization loss formulations that we consider. The total loss is:

$$\mathcal{L} = (1-\alpha)\mathcal{L}_{rec} + \alpha\mathcal{L}_{reg}, \tag{1}$$

where α is the regularization strength, and where we use three different definitions of $\mathcal{L}_{reg}$:

Stability regularization:

$$\mathcal{L}_{stability} = ||f(x) - f(T(x))||_2. \tag{2}$$

Transform invariance regularization:

$$\mathcal{L}_{trans\text{-}inv} = ||f(T(x)) - T(f(x))||_2\,. \tag{3}$$

Sparse Jacobian regularization:

$$\mathcal{L}_{jacobian} = ||\,(f(T(x)) - f(x)) - (T(y) - y)\,||_2 \tag{4}$$
$$= ||\,(f(T(x)) - T(y)) - (f(x) - y)\,||_2\,. \tag{5}$$

B. Transformations

The image perturbations $T(\cdot)$ are performed by means of a linear transformation of the pixel indices i and j,

$$\begin{bmatrix} i' \\ j' \end{bmatrix} = \begin{bmatrix} T_{1,1} & T_{1,2} & T_{1,3} \\ T_{2,1} & T_{2,2} & T_{2,3} \end{bmatrix} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix}, \tag{6}$$

where (i', j') are the transformed indices, such that $T(x)_{i,j} = x_{i',j'}$. The transformation matrix elements are defined as follows:

$$\begin{aligned}
T_{1,1} &= \frac{z\cos a}{\cos(h_x)}, \\
T_{1,2} &= \frac{z\sin(a)}{\cos(h_x)}, \\
T_{1,3} &= \frac{s_x\cos(h_x) - s_x z\cos a}{2\cos(h_x)} \\
&\quad + \frac{2t_x z\cos a - s_y z\sin a + 2t_y z\sin a}{2\cos(h_x)}, \\
T_{2,1} &= \frac{z\sin b}{\cos(h_y)}, \\
T_{2,2} &= \frac{z\cos b}{\cos(h_y)}, \\
T_{2,3} &= \frac{s_y\cos(h_y) - s_y z\cos b}{2\cos(h_y)} \\
&\quad + \frac{2t_y z\cos b - s_x z\sin b + 2t_x z\sin b}{2\cos(h_y)}.
\end{aligned} \tag{7}$$

Here, we have $a = h_x - r$ and $b = h_y + r$, and (s_x, s_y) is the image size. The formulation assumes that the image origin is in the corner of the image, thus incorporating a translation of the image center to the origin before performing the image transformations and translating back afterwards. (t_x, t_y), r, z, and (h_x, h_y) are translation offset, rotation angle, zoom factor, and shearing angles, respectively. All the transformation parameters are drawn from uniform distributions, in a selected range of values as specified in Table 1.

Table 1. Ranges of transformation parameters.

Parameter	Min	Max
Translation	-2 px	2 px
Rotation	-1°	1°
Zoom	0.97×	1.03×
Shearing	-1°	1°

C. Implementation

The transformations in Section B and the loss formulations in Section A can be implemented with little modification of an existing CNN training script. An example implementation is provided in Listing 1, using Tensorflow. It evaluates the CNN on the input image x and the transformed image $T(x)$ by means of a weight-sharing network.

```
import numpy as np
import tensorflow as tf

# Other initialization stuff
...

# The ground truth (bs is batch size)
y = tf.placeholder(tf.float32, [bs, sx, sy])

# Random transformations
ang = np.deg2rad(1.0)
tx = tf.random_uniform(shape=[bs,1], minval=-2.0, maxval=2.0, dtype=tf.float32)
ty = tf.random_uniform(shape=[bs,1], minval=-2.0, maxval=2.0, dtype=tf.float32)
r = tf.random_uniform(shape=[bs,1], minval=-ang, maxval=ang, dtype=tf.float32)
z = tf.random_uniform(shape=[bs,1], minval=0.97, maxval=1.03, dtype=tf.float32)
hx = tf.random_uniform(shape=[bs,1], minval=-ang, maxval=ang, dtype=tf.float32)
hy = tf.random_uniform(shape=[bs,1], minval=-ang, maxval=ang, dtype=tf.float32)

# Transformation matrix
a = hx - r
b = hy + r
T1 = tf.divide(z*tf.cos(a), tf.cos(hx))
T2 = tf.divide(z*tf.sin(a), tf.cos(hx))
T3 = tf.divide(sx*tf.cos(hx)-sx*z*tf.cos(a)+2*tx*z*tf.cos(a)-sy*z*tf.sin(a)+2*
      ty*z*tf.sin(a), 2*tf.cos(hx))
T4 = tf.divide(z*tf.sin(b), tf.cos(hy))
T5 = tf.divide(z*tf.cos(b), tf.cos(hy))
T6 = tf.divide(sy*tf.cos(hy)-sy*z*tf.cos(b)+2*ty*z*tf.cos(b)-sx*z*tf.sin(b)+2*
      tx*z*tf.sin(b), 2*tf.cos(hy))
T7 = tf.zeros([bs,2], 'float32')
T = tf.concat([T1, T2, T3, T4, T5, T6, T7], 1)

# Perform transformation
Ty = tf.contrib.image.transform(y, T, interpolation='BILINEAR')

# Prepare input x from ground truth y
x = prepare_data(y)
Tx = prepare_data(Ty)

# Model
with tf.variable_scope("siamese") as scope:
  fx = cnn_model(x)

  # Weight-sharing
  scope.reuse_variables()
  fTx = cnn_model(Tx)

# Transformation on prediction
Tfx = tf.contrib.image.transform(fx, T, interpolation='BILINEAR')

# Reconstruction loss
loss = (1.0-alpha)*tf.reduce_mean(tf.square(fx-y))

# Regularization loss
if stability:
  loss += alpha*tf.reduce_mean(tf.square(fx-fTx))
elif transform_invariance:
  loss += alpha*tf.reduce_mean(tf.square(fTx-Tfx))
elif sparse_jacobian:
  loss += alpha*tf.reduce_mean(tf.square((fTx-fx)-(Ty-y)))

# Train model using the regularized loss
...
```

Listing 1. Tensorflow example for formulating regularized loss.

D. Training time

The regularized losses take approximately 2 times longer to evaluate as compared to training with only the loss $\mathcal{L}_{rec} = ||f(x) - y||_2$. For the HDR reconstruction application, the Sparse Jacobian formulation took on average 1.92 times longer, whereas the transform invariance took 1.99 times longer. The latter is slightly slower since it requires running the transformation $T(f(x))$ on the reconstructed image $f(x)$.

E. Experimental setup

The two different applications used for the experiments are evaluated in the following way:

- In the total loss in Equation 1, we use three different formulations of $\mathcal{L}_{reg}$: stability (2), transform invariance (3), and sparse Jacobian (5) regularization.
- The regularization strength is sampled at 12 locations, $\alpha_i = \frac{l_i}{l_i+1}$, $i = 1, ..., 12$, where $l_i = 2^{i-3}$. This means that the relative regularization strength, or ratio $\frac{\mathcal{L}_{reg}}{\mathcal{L}_{rec}}$, will double for each point.
- For the perturbed sample $T(x)$, we use the geometric transformation specifying the warping from coordinate transformations according to Equation 6. For the stability regularization we also add one setting with noise perturbations, $T(x) = x + \Delta x$, where $\Delta x \sim \mathcal{N}(0, \sigma)$, and σ is randomly selected for each image, $\sigma \sim \mathcal{U}(0.01, 0.04)$.
- We complement with a training run using $T(x)$ for specifying naïve augmentation, increasing the training dataset size from N to $2N$.
- For each combination of the above, we run 10 individual trainings, in order to estimate a proper mean and standard deviation of each datapoint.

In total, the combinations and repeated runs means that for each of the two applications we perform 500 optimization runs.