#!/bin/sh

pname=LogicTrain

# this script outputs the jaccard index using various parameters

# utilize an adaptive proportional kernel for filtering

# syntax: LogicTrain <fname> <ftype> <xkernel> <ykernel> <okernel> <ckernel> <fmethod> <mergemethod>

fname=$1 # Image filename

ftype=$2 # Filter type (Mean, Gaussian, or Median)

xkernel=$3 # x-kernel proportion (1 for no filtering) Note: kernel=xdim/xkernel

ykernel=$4 # y-kernel proportion (1 for no filtering)

okernel=$5 # Region opening kernel (1 for no opening)

ckernel=$6 # Region closing kernel (1 for no closing)

fmethod=$7 # Fill method (Single or Multiple)

mergemethod=$8 # Merging method (NotHighorLow, LowUHigh, BlueOnly, HighUHighNLow)

gaussx=$xkernel # Gaussian x-standard deviation

gaussy=$ykernel # Gaussian y-standard deviation

r=0

g=0

b=0

jac=0

## file conversion ##

# convert jpeg color image to .vx byte image

vfmt if="$fname.jpg" of="$fname.vx" -jpeg

## color channel extraction ##

# red

vchan if="$fname.vx" of="r$fname.vx" c=1

# green

vchan if="$fname.vx" of="g$fname.vx" c=2

# blue

vchan if="$fname.vx" of="b$fname.vx" c=3

## get x and y dimension ##

vcc xDim.c -o xDim

vcc yDim.c -o yDim

xdim=$(xDim if="r$fname.vx")

ydim=$(yDim if="r$fname.vx")

## create proportional kernels ##

# find proportional x-kernel

xkernel=$(expr $xdim / $xkernel)

# verify that the resulting kernel is odd

rem=$(( $xkernel % 2 ))

if [ $rem -eq 0 ]

then

xkernel=$xkernel - 1

fi

# find proportional y-kernel

ykernel=$(expr $ydim / $ykernel)

# verify that the resulting kernel is odd

rem=$(( $ykernel % 2 ))

if [ $rem -eq 0 ]

then

ykernel=$ykernel - 1

fi

# assign kernels

meanx=$xkernel # Mean x-kernel dimension

meany=$ykernel # Mean y-kernel dimension

medianx=$xkernel # Median x-kernel dimension

mediany=$ykernel # Median y-kernel dimension

case $ftype in

Mean)

## perform mean filtering ##

# red

cp "r$fname.vx" temp.vx

vmean if=temp.vx of="r$fname.vx" xs=$meanx xy=$meany

# green

cp "g$fname.vx" temp.vx

vmean if=temp.vx of="g$fname.vx" xs=$meanx xy=$meany

# blue

cp "b$fname.vx" temp.vx

vmean if=temp.vx of="b$fname.vx" xs=$meanx xy=$meany;;

Gaussian)

## perform gaussian filtering ##

# red

cp "r$fname.vx" temp.vx

vgfilt if=temp.vx of="r$fname.vx" xs=$gaussx ys=$gaussy zs=0

# green

cp "g$fname.vx" temp.vx

vgfilt if=temp.vx of="g$fname.vx" xs=$gaussx ys=$gaussy zs=0

# blue

cp "b$fname.vx" temp.vx

vgfilt if=temp.vx of="b$fname.vx" xs=$gaussx ys=$gaussy zs=0;;

Median)

## perform median filtering ##

# red

cp "r$fname.vx" temp.vx

vmedian if=temp.vx of="r$fname.vx" xs=$medianx xy=$mediany

# green

cp "g$fname.vx" temp.vx

vmedian if=temp.vx of="g$fname.vx" xs=$medianx xy=$mediany

# blue

cp "b$fname.vx" temp.vx

vmedian if=temp.vx of="b$fname.vx" xs=$medianx xy=$mediany;;

esac

## histogram equalization ##

# red

cp "r$fname.vx" temp.vx

vhisteq if=temp.vx of="r$fname.vx"

# green

cp "g$fname.vx" temp.vx

vhisteq if=temp.vx of="g$fname.vx"

# blue

cp "b$fname.vx" temp.vx

vhisteq if=temp.vx of="b$fname.vx"

## perform otsu's method thresholding ##

# red

vthresh if="r$fname.vx" of="or$fname.vx"

# green

vthresh if="g$fname.vx" of="og$fname.vx"

# blue

vthresh if="b$fname.vx" of="ob$fname.vx"

## color mask merging ##

case $mergemethod in

NotHighOrLow)

# merge the mask with the lowest value with the inverse of the mask with the highest value

vcc nMask.c -o nMask

r=$(nMask if="or$fname.vx")

b=$(nMask if="ob$fname.vx")

g=$(nMask if="og$fname.vx")

if [ $r -ge $b -a $r -ge $g ]

then

# inverse r

vpix -neg if="or$fname.vx" of=temp.vx

if [ $g -ge $b ]

then

echo not r or b

# find union

vembed if=temp.vx ig="ob$fname.vx" of="lo$fname.vx" -merge

else

echo not r or g

# find union

vembed if=temp.vx ig="og$fname.vx" of="lo$fname.vx" -merge

fi

elif [ $b -ge $r -a $b -ge $g ]

then

# inverse b

vpix -neg if="ob$fname.vx" of=temp.vx

if [ $g -ge $r ]

then

echo not b or r

# find union

vembed if=temp.vx ig="or$fname.vx" of="lo$fname.vx" -merge

else

echo not b or g

# find union

vembed if=temp.vx ig="og$fname.vx" of="lo$fname.vx" -merge

fi

else

# inverse g

vpix -neg if="og$fname.vx" of=temp.vx

if [ $b -ge $r ]

then

echo not g or r

# find union

vembed if=temp.vx ig="or$fname.vx" of="lo$fname.vx" -merge

else

echo not g or b

# find union

vembed if=temp.vx ig="ob$fname.vx" of="lo$fname.vx" -merge

fi

fi

## inverse result ##

cp "lo$fname.vx" temp.vx

vpix -neg if=temp.vx of="lo$fname.vx";;

LowUHigh)

# interesect the mask with the highest value with the union of the two masks with the lowest values

vcc nMask.c -o nMask

r=$(nMask if="or$fname.vx")

b=$(nMask if="ob$fname.vx")

g=$(nMask if="og$fname.vx")

if [ $r -ge $b -a $r -ge $g ]

then

echo b or g and r

# find union of b and g

vembed if="ob$fname.vx" ig="og$fname.vx" of=temp2.vx -merge

# take negative

vpix -neg if=temp2.vx of=temp.vx

# find intersection with r

vembed if=temp.vx ig="or$fname.vx" of="lo$fname.vx" -and

elif [ $b -ge $r -a $b -ge $g ]

then

echo r union g or b

# find union of r and g

vembed if="or$fname.vx" ig="og$fname.vx" of=temp2.vx -merge

# take negative

vpix -neg if=temp2.vx of=temp.vx

# find intersection with b

vembed if=temp.vx ig="ob$fname.vx" of="lo$fname.vx" -and

else

echo rUbNg

# find union of r and b

vembed if="or$fname.vx" ig="ob$fname.vx" of=temp2.vx -merge

# take negative

vpix -neg if=temp2.vx of=temp.vx

# find intersection with g

vembed if=temp.vx ig="og$fname.vx" of="lo$fname.vx" -and

fi;;

BlueOnly)

# utilize only the blue channel for analysis

echo blue channel isolation

cp "ob$fname.vx" temp.vx

# inverse channel

vpix -neg if=temp.vx of="lo$fname.vx";;

HighUHighNLow)

# negative union of the two highest channels intersected with the inverse lowest channel

vcc nMask.c -o nMask

r=$(nMask if="or$fname.vx")

b=$(nMask if="ob$fname.vx")

g=$(nMask if="og$fname.vx")

if [ $r -le $b -a $r -le $g ]

then

# take negative of b and g

vpix -neg if="ob$fname.vx" of=temp.vx

vpix -neg if="og$fname.vx" of="lo$fname.vx"

# find union of not b and not g

vembed if=temp.vx ig="lo$fname.vx" of=temp2.vx -merge

# find intersection with not r

vpix -neg if="or$fname.vx" of=temp.vx

vembed if=temp2.vx ig=temp.vx of="lo$fname.vx" -and

elif [ $b -le $r -a $b -le $g ]

then

# take negative r and g

vpix -neg if="or$fname.vx" of=temp.vx

vpix -neg if="og$fname.vx" of="lo$fname.vx"

# find union of not r and not g

vembed if=temp.vx ig="lo$fname.vx" of=temp2.vx -merge

# find intersection with not b

vpix -neg if="ob$fname.vx" of=temp.vx

vembed if=temp2.vx ig=temp.vx of="lo$fname.vx" -and

else

# take negative r and b

vpix -neg if="or$fname.vx" of=temp.vx

vpix -neg if="ob$fname.vx" of="lo$fname.vx"

# find union of not r and not b

vembed if=temp.vx ig="lo$fname.vx" of=temp2.vx -merge

# find intersection with not g

vpix -neg if="og$fname.vx" of=temp.vx

vembed if=temp2.vx ig=temp.vx of="lo$fname.vx" -and

fi;;

esac

## perform opening ##

# create kernel

vgenim of=temp.vx c= x=$okernel

# erode

vsf if="lo$fname.vx" of=temp2.vx k=temp.vx -e

# create kernel

vgenim of=temp.vx c= x=$okernel

# dilate

vsf if=temp2.vx of="lo$fname.vx" k=temp.vx -d

## perform closing ##

# create kernel

vgenim of=temp.vx c= x=$ckernel

# dilate

vsf if="lo$fname.vx" of=temp2.vx k=temp.vx -d

# create kernel

vgenim of=temp.vx c= x=$ckernel

# erode

vsf if=temp2.vx of="lo$fname.vx" k=temp.vx -e

## fill enclosed voids ##

cp "lo$fname.vx" temp.vx

case $fmethod in

Single)

vbfilt if=temp.vx of="lo$fname.vx" -d;;

Multiple)

v3bfill if=temp.vx of="lo$fname.vx" -2d -d;;

esac

## display parameters ##

echo "Image file = $fname"

echo "Filter type = $ftype"

case $ftype in

Mean)

echo "Mean x-kernel dimmension = $meanx"

echo "Mean y-kernel dimmension = $meany";;

Median)

echo "Median x-kernel dimmension = $medianx"

echo "Median y-kernel dimmension = $mediany";;

Gaussian)

echo "Gaussian x-standard deviation = $gaussx"

echo "Gaussian y-standard deviation = $gaussy";;

esac

echo "Region opening kernel = $okernel"

echo "Region closing kernel = $ckernel"

echo "Fill method = $fmethod"

echo "Merge method = $mergemethod"

## find Jaccard index ##

vcc jacind.c -o jacind

vfmt if="$fname""\_Segmentation.png" of="tru$fname.vx" -png

jacind if="lo$fname.vx" tf="tru$fname.vx"

jac=$(jacind if="lo$fname.vx" tf="tru$fname.vx")

echo "$jac" >> output\_file.txt

## convert final image to gif ##

vxto gif "lo$fname.vx"