Macro files

Each file needs to be saved separately by the name mentioned in the first line of the macro

```
//file "1. lif2tif.txt"
/*
 * Extract images from a lif file
 * Images are RGB but for the current analysis, only the green channel is saved
 * New insights also require the blue channel.
 * Enter starting folder, process recursively all lif files, creating a subfolder for each
 * Name of subfolder reflects the name of the lif file
 * Name of the image files are created from series names in lif files.

 * Blue is in the first channel, green in the second and red in the third channel
 * (as opposed to the usual RGB ordering of pixel data).
 */
//last changes (for publication) 20191105 13:55
//by a.jonker@amsterdamumc.nl

//modified version of "lif2tif 20190418"

macro "Extract Timelapse from Lif as Tif"{
    //avoiding the Ext functions
    startTime = getTime();
    close("*");
    print("\\Clear");
    run("Bio-Formats Macro Extensions");
    startPath = getDirectory("Choose a Directory");
setBatchMode(true);//change to true for speeding up, should be false for debugging or you won't see images!
    nLifFiles = processFolder(startPath);
setBatchMode(false);
    close("*");
    endTime = getTime();deltaTime=floor((endTime-startTime)/60);
    showMessage("Processed "+nLifFiles+" Lif files in " + deltaTime + " seconds");
}

    function processFolder(inPath){
        processedFiles=0;
        arrayOfNames = getFileList( inPath );
        for(i=0;i<arrayOfNames.length;i++){
            thisName=arrayOfNames[i];
            if(File.isDirectory(inPath+thisName)){
                processedFiles = processedFiles + processFolder(inPath + thisName);
            }
        }
        for(i=0;i<arrayOfNames.length;i++){
            thisName = arrayOfNames[i];//shows up in Debug window
            if(endsWith(thisName,".lif")){
                thisFolder=replace(arrayOfNames[i],".lif","");
                outPath = inPath+thisFolder+File.separator;
                bluePath = outPath+"blue"+File.separator;
                //having a symmetrical solution in bluePath and greenPath would have been better
                File.makeDirectory(outPath);
                File.makeDirectory(bluePath);
                if (!File.exists(outPath) || !File.exists(bluePath)){
                    errorMessage="Unable to create directory " + outPath;
                    exit(errorMessage);
                }
                processedFiles = processedFiles + extractGreen(thisFolder,inPath,outPath);
            }
        }
```

```
		return	processedFiles;
	}

	function extractGreen(fileName,inpath, outpath){//fileName has no extension, fileName is same as tail of outpath
		close("*");//remove remaining windows
		pf=0;//number of processed files
		print("processing "+fileName+ " into "+outpath);//keep track of progress in big batches
		run("Bio-Formats Macro Extensions");
		//import headers in Original Metadata Window
		//get an array with series names
		//for each series
		//could use FluoCubeName property as indicator of wavelength used
		id = inpath + fileName+".lif";
		Ext.setId(id);//use EXT functions of Bio-Formats extensions to read .lif files
		Ext.getCurrentFile(file);
		Ext.getImageCount(n);
		Ext.getSeriesCount(seriesCount);

		Ext.getFormat(inpath+fileName+".lif", format);
		Ext.getSeriesCount(seriesCount);
		for(s=0; s < seriesCount; s++){
			Ext.setSeries(s);st=s+1;
			Ext.getSeriesName(seriesName);
			//string, format " Series X Name\t StringValue"
			//string, format "Image name\tStringValue"
			//a slash is sometimes found in the seriesName, as subfolder indicator
			//slashes are a directory indicator that we don't want to handle; replace slash by undersocre
			if(indexOf(seriesName,"/")!=-1){seriesName=replace(seriesName,"/","_");}
			destFile = fileName + "/" + seriesName;
			Ext.getImageCount(imageCount);//if 0 then this is a text file
			if(imageCount > 0){
				ext = ".tif";
			}else{
				ext="";
			}
			testname = seriesName + ext;
			if(!File.exists(outpath + testname) || true){//force re-saving of blue channel; save time by removing ||true and skip old files.
				print("\\Update:trying to load "+testname);
				run("Bio-Formats Importer", "open=["+inpath+fileName+".lif"+"] use_virtual_stack autoscale color_mode=Default rois_import=[ROI manager] view=Hyperstack stack_order=XYCZT series_"+st);
				print("\\Update:Successfully loaded "+testname);
				outFileName = outpath + testname;
				print("\\Update:Saving "+outFileName);
				if(ext == ".tif"){
					Stack.setDisplayMode("color");
					run("Stack to Images");
					tws=getList("image.titles");
					for(i=tws.length;i>0;i--){//close images that are not needed, save other images
						thisTitle=tws[i-1];
						selectImage(thisTitle);
						foundPos=indexOf(thisTitle,"3/3");//image containing red (remember: order is blue, green, red)
						if(foundPos!=-1){
							close;//run("Close");
						}

						foundPos=indexOf(thisTitle,"2/3");
						if(foundPos!=-1){
							saveAs("tiff", outpath+seriesName+".tif");//save the green image. Not a symmetrical solution (would be outpath/green/seriesName.tif).
```

```
                            close;//run("Close");
                        }
                        foundPos=indexOf(thisTitle,"1/3");
                        if(foundPos!=-1){
                            saveAs("tiff", outpath+"blue/"+seriesName+".tif");
                            close;//run("Close");
                        }
                    }
                }
                print("\\Update:Saving "+outFileName );
                pf++;
            }else{
                print ("skipping already existing "+testname);
            }
        }
        return pf;
    }
    //file "2. generate and clean up rois.txt"

/*
 * Generate a polyline region of interest that follows the edge of tissue in the image
 * The edge starts at an image's border, crosses the point farthest away from any border and exits on another border.
 *
 * These ROIs are stored in a zip file within the folder where tiff files are found
 * The macro recursively follows any folder from the chosen start path
 *
 * Last change (for publication) dd 20191105 12:36
 * By a.jonker@amsterdamumc.nl
 *
 * Outline of the macro:
 * Tissue is considered land, background is considered water. Try to find the shoreline.
 * Assumption is that the edge of the tissue is near the middle of the image
 * Therefore, any cracks, holes and other artifacts should mostly be distregarded when finding the shore
 *
 * If shoreline cannot be detected for any reason, draw a diagonal ROI, as a visual alert
 *
 * Failures occur and users must be aware of, or manually corrected if
 * -foreground and background are too similar (noisy background, sporadic signal in foreground)
 * -tissue has cracks touching the edge (shoreline enters and exits through the same image edge)
 * -tissue has cracks, which will be followed, but which do not represent edge tissue
 * -multiple parallel areas of low signal are present (the wrong edge may be chosen)
 *
 * Because of these shortcomings, resulting ROIs should be inspected manually.
 * To this end, a built-in switch can be used, the functions can be called in two ways:
 * -function argument = 1 -> tissue edge ROIs will be generated without user intervention
 *  Input: a folder with images containing tissue that touches at least one image edge
 *  Output: a zip file containing one ROI for each image

 * -function argument = 0 -> ROIs will be loaded onto images for inspecition and correction
 *  Input: a folder with a zip file containing named ROIs, and correspondingly named image files
 * (possible user interaction to amend or re-create the proper tissue edge)
 *  Output: a zip file containing one ROI for each image, indicating the tissue edge
 *
 * User intervention can take place in two ways after ROIs are loaded onto images
 * -Rois can be re-drawn/re-created using the (already active) broken-line tool
 * -Rois can be amended by deleting (alt-click) or relocating equidistant handles on the ROI
 *
 * If the user accepts the ROI, the file remains as is
 * If the user amends a ROI the roi is resampled and the original ROI.zip overwritten
 *
 * User inspection and intervention is slow.
 * Re-examining is prevented, and inspection can be sped up by skipping folders
 * Any folder name containing an asterix (*) will be skipped.
 * Bug fix 20191018, identified potential bug in 'restore selection'
 */
    requires("1.52p");
    generate=1;//can be changed to 0 for inspection
    inspect=1-generate;//complementary boolean value

    var logLineNr = 0;//vertical indent in log file allows for concise log
    print("\\Clear");
    if(nResults!=0){
        IJ.deleteRows(0,nResults); //any previous result should be wiped
    }
    startPath = getDirectory("Choose a Directory");
    close("*");
    pf = processFolder(startPath,generate);
    //pf = processFolder(startPath,inspect);
    print ("analysed "+pf+" files");

function processFolder(inPath,generateOrInspect){
    if(indexOf(inPath,"*")!= -1){//skip completed folders; the asterix can be, upon completion, manually added to a folder
name
        return 0;
    }
    generateload=generateOrInspect;
    //(=1)-> generate without user intervention
    //(=0)-> load, inspect, alter, accept and save
    processedFiles=0;
    arrayOfNames = getFileList( inPath );
    for(i=0;i<arrayOfNames.length;i++){
        thisName=arrayOfNames[i];//using a separate variable allows for inspection in debugger
        if(File.isDirectory(inPath+thisName)){ //enter a depth-first search
            print("\\Update"+logLineNr+": processing "+thisName);//logLineNr allows selective clearing
            processedFiles = processedFiles + processFolder(inPath + thisName,generateload);
        }
    }
    folderName=File.getName(inPath);
    logLineNr++;
    print("\\Update"+logLineNr+": processing "+folderName);//logLineNr allows selective clearing
    if(generateload ==1){ //generate outlines without further intervention
        processedFiles = processedFiles + generateRois(inPath);
        nrois=roiManager("count");
        if(checkProperRoiNames()!=0){ //Weird cases create exclamation marks in the ROI name
            debug;
        }
        if(nrois>0){
            if(File.exists(inPath+"roiSet.zip")){
                //do not overwrite but move to date stamped copy
                prevFile=inPath+"roiSet_before_"+ymdhs()+".zip";
                File.rename(outFile, prevFile);
                //or the more rude way: File.delete(inPath+"roiSet.zip");
            }
            if(checkProperRoiNames()!=0){
                //sometimes an exclamation mark is preceding the roi/file name
                debug;
            }
            roiManager("save",inPath+"roiSet.zip");
        }

    }else{ //load images and display for inspection with ROIs active and image contrast maximised
        processedFiles = processedFiles + loadRois(inPath);
        if (nImages() > 1){ //images have been found
            run("Tile");
            for(i=1;i<=nImages;i++){
                selectImage(i);
                run("Enhance Contrast", "saturated=0.35");
            }
            //ask user to accept the ROIs (with numbered begin-end points?) and store ROI's
            updateDisplay();
            accept= getBoolean("Do you need to change these outlines?\n"+thisName,"accept as is","change");

            if (accept == 1) {
                print("\\Update"+logLineNr+":accepted " + inPath); // no  need to re-save the roi
                logLineNr++;
                updateDisplay() ;
            }else{
                changeRois();//wait for the user to have modified ROIs and clicked OK
                nrois=roiManager("count");//if rois were found
                if(nrois>0){//overwrite the old ones
                    outFile = inPath+"roiSet.zip";
                    if(File.exists(outFile)){
                        prevFile=inPath+"roiSet_before_"+ymdhs()+".zip";
                        File.rename(outFile, prevFile);
                        // or the more rude way: File.delete(outFileName);
                    }
                    roiManager("save",outFileName);
                }else{//no rois were found, delete the old roi set
                    if(File.exists(inPath+"roiSet.zip"))File.delete(inPath+"roiSet.zip");

                }
            }
        }
    }
    if(nResults!=0)IJ.deleteRows(0,nResults);
    close("*");
    return    processedFiles;
}

function loadRois(path){
    //open the first encountered zip file in a folder and use that as ROI set
    nRois = 0;
    zipfiles=getFileList(path);
    if(zipfiles.length == 0){
        return 0;
    }
    i=0;found = 0;
    do{
        thisname=zipfiles[i];
        fpn=path+zipfiles[i];
        if(endsWith(fpn,"zip")){
            found = 1;
        }else{
            i++;
        }
    }while (i<zipfiles.length&&found==0);
    if(!endsWith(fpn,"zip")) return;
    print("\\Clear");
    if(nResults!=0)IJ.deleteRows(0,nResults);
    close("*");
    if(roiManager("count")>0){
            roiManager("Deselect");
            roiManager("Delete");
        }
    //path = File.openDialog("Select a roi (zip) File");
    roiZip=File.getName(fpn);
    dir = replace(path,roiZip,"");//this leaves the directory separator in place

    roiManager("Open", fpn);
    for(iRoi=0;iRoi<roiManager("count");iRoi++){
        //get the name of the ROI, this is the same as the name of the image file the ROI belongs to
        imName=call("ij.plugin.frame.RoiManager.getName", iRoi);
        //test for names of images that we generated ourselves; exclude these usual suspects
        if((   !isOpen(imName)
            && indexOf(imName, "montage")==-1
            && indexOf(imName, "strip")==-1
            && indexOf(imName, "normalised")==-1
        ) !=0)
        {
            //the image is not (yet) open so attempt to importimage by that name
            importImageName=dir+imName;
            if(File.exists(importImageName)){
                open(importImageName);
                selectImage(imName);
                roiManager("select",iRoi);
                Roi.getCoordinates(xpoints,ypoints);
                makeSelection("freeline",xpoints,ypoints);
                run("Interpolate", "interval=10  adjust"); //put in adjustment handles
                nRois++;
            }else{
                print("\\Update"+logLineNr+": Failed to locate image " + importImageName);
                logLineNr++;
            }

        }
    }
    return nRois;
}
function generateRois(path){
//debug;
    if(roiManager("count")>0){
        roiManager("Deselect");
        roiManager("Delete");
    }
    nRois = 0;
    files=getFileList( path );
    for(i=0;i<files.length;i++){
        thisName=files[i];
        if(    indexOf(files[i],"tif")!=-1
            &&indexOf(files[i],"Snapshot")==-1
            &&indexOf(files[i],"strip")==-1
            &&indexOf(files[i],"montage")==-1
        ){//only process Mark_and_find files
            open(path+files[i]);
            inImgID=getImageID();
            findCoastline(inImgID);
            nRois=roiManager("count");
            print(nRois+" rois found ; i= "+i);
        }
    }
    if (nRois == 0 ) {
        //sometimes things go wrong, then offer debug option or fail silently
        //debug;
    }
```

```
        return nRois;
}
function changeRois(){
        //for every ROI in manager, get the image name, set roi, set ROI as editable
        // this has been done already if changeROIs() is called after loading images and ROIs
        setTool("polyline");
        //      wait for changes to be complete
        waitForUser("Change ROIs to your satisfaction\nRemember to work from bio top to bio bottom\nThen click OK");
        imagesTitlesList = getList("image.titles");
        nImgs=imagesTitlesList.length; //now walk all open images
        for(iImg=0;iImg<nImgs;iImg++){
                imTitle=imagesTitlesList[iImg];
                selectImage(imTitle);//get their title
                st=selectionType() ;
                if(st==6 || st == 7){
                        for(iRoi=0;iRoi<roiManager("count");iRoi++){
                                imName=call("ij.plugin.frame.RoiManager.getName", iRoi);
                                if(imName==imTitle){
                                        //you can't just call 'update' as the roi manager is not aware of the new coordinates
                                        //first harvest the current roi (this differs from the roiManager's current Roi!!) from the image
                                        Roi.getCoordinates(xpoints,ypoints);
                                        //select iRoi in roimanager to amend
                                        roiManager("select",iRoi);
                                        // re-create roi in image so roiManager is aware of changes
                                        makeSelection("freeline",xpoints,ypoints);
                                        //then do update the roi in the roiManager's list
                                        roiManager("update");
                                }
                        }
                }
        }
        return;
}
function findCoastline(imgID){
        getDimensions(width, height, channels, slices, frames);
        inTitle=getTitle();
        //Tissue in the image covers at least 30% of the area.
        aPart = 0.3 * width * height;
        run("Select All");
        run("Duplicate...", "title=coastline");//keep origial intensity data and work on copy
        dupID=getImageID();
        run ("Enhance Contrast", "saturated=0.35");//some images have very low signal
        run("Apply LUT");//now image is bright, also to the eye. Data is no longer quantitative
        run("Set Scale...", "distance=0 known=0 pixel=1 unit=pixel");//remove Âµm scale, eventually we want pixels anyway
        rotateNeed=0;//might be changed to 1 by code further down
        run("Gaussian Blur...", "sigma=2");//get a global intensity so small features disappear
        setAutoThreshold("Default dark");
        setOption("BlackBackground", true);
        run("Convert to Mask");
        run("Fill Holes");//remove any blobs and blebs in both the tissue...
        run("Invert");
        run("Fill Holes");//...and in the background
        run("Invert");

        run("Set Measurements...", "area mean standard modal min centroid center bounding shape integrated skewness
area_fraction limit redirect=None decimal=5");
        setAutoThreshold("Default dark");
        List.setMeasurements;
        xm=List.getValue("XM");//get the center of mass of what is hopefully the tissue
        ym=List.getValue("YM");
        doWand(xm,ym);//create a ROI around this
```

```
run("Measure");
//the following code tries to figure out if selection is background or tissue
//and swaps found outlines if necessary
amean=getResult("Mean");//the current selection's mean
aarea=getResult("Area");
selectImage(imgID);//mesure in the original image
run("Restore Selection");//re-create the same area in the original image
run("Measure");
m1=getResult("Mean");
run("Make Inverse");
run("Measure");
m2=getResult("Mean");
if(m2<m1){//try alternative
    run("Make Inverse");
}
selectImage(dupID);//continue to work with the duplicate.

//we already have a selection present here //run("Restore Selection");
if(m2<m1){//so also have the inverse in the duplicate, if necessary.
    run("Make Inverse");
}

//aarea should now delineate tissue and run along the 'beach'
//amean should now be 255
//we are not interested in the entire ROI enclosing the tissue
//but only the area that touches the outer edge of the tissue
//
//find the roi coordinates farthest away from edge as
//this is a good starting point to find the shore line-part of the entire ROI
run("Interpolate", "interval=1");//walk pixel by pixel
Roi.getCoordinates(xpoints, ypoints);
dmax=0;//maximum distance to all edges
nmax=0;//ordinal of coordinate that is farthest away from all edges

for(i=0;i<xpoints.length;i++){
    x=xpoints[i];
    y=ypoints[i];
    makePoint(x, y, "cross");//wait(delay); //if you want to SEE it happening, remove first two slashes
    dx=min(x,width-x);//x-distance away from edges
    dy=min(y,height-y);//y-distance away from edges
    farthest=min(dx,dy);//should also check for convexity here?
    if(farthest>dmax){//the current pixel is further away, keep this info
        nmax=i;
        dmax=farthest;
    }
}
makeOval(xpoints[nmax]-2,ypoints[nmax]-2,5,5);//indicate the point found as the farthest from any image side
//waitForUser("Farthest");//uncomment if you want to SEE it happening
nmid=nmax;
i=nmid+1;//mod xpoints.length; go and inspect the next point of the (closed contour) ROI.
if(i==xpoints.length){i=0;}
do{ //ordinally walk forward in coordinates, starting at the coordinate farthest away from any image edge,
    //till we reach the image border
    x=xpoints[i];
    y=ypoints[i];
    makePoint(x, y, "cross");//wait(delay);
    i=(i+1);
    if(i==xpoints.length){//oops, the beach is not [first..start______end..last] but [first___end...start___last]
        i=0;
        rotateNeed=1;//rotating is pushing out values on one side, moving them in on the other side of the array
    }
    }while(i<xpoints.length && x>0 && y>0 && x<width-1 && y<height-1 && i!=nmax);//rounding gives x=-0.00
which is false for x==0
    // that is: wile we have not reached the end of the array,
    //do not hit a wall left or right(x=0 or x=width-1)
    //or top or bottom (y=0 or y=height-1)
    //makeOval(x-2,y-2,5,5);//uncomment if you want to SEE it happening
    //waitForUser("Forward");//uncomment if you want to SEE it happening
    end=i;
    if(rotateNeed!=0){
        //from the polygon ROI we have selected part of it that describes the coastline
        //index i now indicates how far the coastline runs; from 0-i and from ?? to nmax
        //we rotate the array so the last point of the coastline is the last point of the array.
        //elements that are 'rotated out' at the end, enter in the head of the array
        xpoints=Array.rotate(xpoints,-i);
        ypoints=Array.rotate(ypoints,-i);
        nmax=nmax-i;
        end=xpoints.length-1;
    }else{
        end=i;
    }
    i=nmax;
    //
    do{//walk backward in coordinates till we reach the image border
        i=(i-1);
        if(i==-1){//oops, we backwar we ran out of the front of the array
            i=xpoints.length -1;//so we enter in the end of the array
        }
        x=xpoints[i];
        y=ypoints[i];
        print("\\Update"+logLineNr+":x = "+x+" y = "+y+" i = "+i);
        makePoint(x, y, "cross");//wait(2);
        //if(i==0) debug;//complex expression; save in variable for debugging purposes:
        finished=!(i<xpoints.length && x>0 && y>0 && x<width-1 && y<height-1 && i!=nmid);
    }while(!finished);

    makeOval(x-2,y-2,5,5);
    //waitForUser("Backward");//if you want to SEE it happening halt until the user has viewed the image

    start=i;

    //nmin is first coordinate nearest to edge of image and at start of coastline, nmax the last of the coastline
    if(start>end){
        ror=xpoints.length-start;//rotate over right distance
        Array.rotate(xpoints,ror);//elements that shift out right, shift in left.
        Array.rotate(ypoints,ror);
        start=0; end=end+ror;
    }
    s=min(start,end)+1;
    e=max(start,end)-1;
    makeArrow(xpoints[s],ypoints[s],xpoints[e],ypoints[e],"notched"); //quicklook visualisation
    //waitForUser("Direction");//uncomment if you want to SEE the direction the beach was travelled
    xpoints=Array.slice(xpoints,s,e);
    ypoints=Array.slice(ypoints,s,e);
    makeSelection("freeline",xpoints,ypoints);//the actual visualisation of the beach
//
    if(xpoints.length < 2){//generate diagonal if no roi has been found.
        makeLine(0, 0, width/2, height/2,width,height);
        //waitForUser("No beach");//uncomment if you want to SEE in case no beach was found

    }
    //store the shoreline ROI in the roiManager and name it after the image it was obtained from
    run("Interpolate", "interval=5 smooth adjust");//iron out small deviations
    roiManager("add");
    roiManager("select",roiManager("count")-1);
    roiManager("rename",inTitle);
    Roi.getCoordinates(xpoints,ypoints);
    close;
    makeSelection("freeline",xpoints,ypoints);//draw the outline in the original image
    return;
}
function checkProperRoiNames(){
    //if an exclamation mark or a dash is present in any ROI name
    nRois=roiManager("count");
    OK=0;
    for(i=0;i<nRois;i++){
        imName=call("ij.plugin.frame.RoiManager.getName", i);
        if(indexOf(imName,"!")!=-1 || indexOf(imName,"-")!=-1){
            print("Roi name of roi number "+i+" contains ! or - ");
            return 1;
        }
    }
    return 0;
}

function min(a,b){
    if(a>b)return b; else return a;
}
function max(a,b){
    if(a<b) return b;else return a;
}
function abs(a){
    if(a<0)return a*-1;else return a;
}
function ymdhs(){
    getDateAndTime(year, month, dayOfWeek, dayOfMonth, hour, minute, second, msec);
    yes=IJ.pad(year,4);
    mos=IJ.pad(month,2);
    das=IJ.pad(dayOfMonth,2);
    hos=IJ.pad(hour,2);
    mis=IJ.pad(minute,2);
    ses=IJ.pad(second,2);
    ymdhsString=yes+mos+das+"_"+hos+"."+mis+"."+ses;
    return ymdhsString;
}
// file "3. straighten ROIs.txt"
/* The current macro is the third (3.) step in the process of measuring brain tissue
 * Input:
 * -a roiSet.zip file containing ROIs
 * -image files containing quantitative fluorescent signal
 * -a text file containing information on the relative image locations
 *
 * Output:
 * - strip files, one strip per image, containing the straightened tissue edge stripe
 * - a montage, containing all pixels in a stripHeight-pixel wide neighbourhood of the tissue edge
 * - a scaled back image of 1000 and of 100 pixel wide, stripHeight pixels high
 * - numeric measurements representing the total fluorescence in the 0-100% length of the tissue edge
 *
 * Method
 * Non-overlapping mosaÃ¯c images of brain tissue were recorded, for details see macro (1.)
 * Foreground and background of each recorded image was found using a separate macro (2.)
 *
 * The placement of the tissue section on the object glass necessitated a certain direction of recording
 * The edge of the tissue was followed either from Top to bottom or Bottom to top
 * Likewise, Left to right or Right to left indicates direction of recording
 * The directionality of the set of tiles was indicated by one of these capital four letters.
 * The directionality string contains the appropriate set from the letters BLRT, max length 4.
 * The letters indicating the order of borders to inspect in the image for a border-crossing of tissue edge
 *
 * For each image of the mosaÃ¯c the tissue border was outlined by a Region Of Interest (ROI)
 * this was done in the macro in file "2. generate and clean up rois"
 * In the current macro, these ROIs are straightened into strips, one strip per image
 * Strips are annealed to one long strip, the montage image
 * Pixels in the montage image representing 0-100% of the tissue edge found in the images.
 * Perpendicular, on either side of the tissue edge, 40 pixels were taken into consideration
 *
 * From the resulting arbitrary wide and stripHeight pixels high montage, fluorescence was measured
 *
 * Last change 20190930 16:59
 *
 */

var debugger=false;//set to true for inspection of code execution
var stripHeight = 80;//column of pixels perpendicular to the tissue edge, area that is extracted into strips
//strip is divided into four sections, each stripHeight/4 high. Two of them do not contain tissue, the other two do.
//Because cells are polarised, the nucleii will be far furhter from tissue edge, so the Blue channel is sampled 40 pixel-wide
//(green) lipid droplets are closer to edge, there only 20 pixel wide would do. For now, we consider a 40-wide pixel column.

//Select the root folder of files to process
directionsFile=File.openDialog("Select the file sampledirections.txt");
//get folderlist
//open directionality file
//debugging runs into the Main function automatically.
macro "Main"{
//      setBatchMode(true);//for faster execution, images will not be updated during processing
        g=Main("");
        setBatchMode(false);//we want to see the execution of the application for blue channel images
        b=Main("blue/");
        setBatchMode(false);
        if(g!=b){//g should be equal to b
                showMessage("Found "+g+" green edges and "+b+" blue edges");
        }
}
macro "Open Zipped Rois"{
        close("*");print("\\Clear");
        if(roiManager("count")!=0){roiManager("deselect");roiManager("delete");}
        path = File.openDialog("Select a roi (zip) File");
        openRoiZipImages(path,"",true);
        run("Tile");

}

function Main(blue){
        fn=File.getName(directionsFile);
        path=replace(directionsFile,fn,"");
        if(!File.exists(directionsFile)){
                return;//without information on the direction of the border we can't reconstruct
        }

        directions = File.openAsString(directionsFile);
        lines = split(directions,"\n");

        //loop through directionality list
        for(lineNo=0;lineNo<lines.length;lineNo++){
```

```
    line = lines[lineNo];
    chunks = split(line,"\t");
    fileNamePart = chunks[0];
    direction = chunks[1];
    //loop through folders in folder list
    arrayOfNames = getFileList(path );

    for(i=0;i<arrayOfNames.length;i++){
        thisName=arrayOfNames[i];
        print("Analysing "+ thisName);
        //if directionality name is in folder name
        partInFile=indexOf(thisName,fileNamePart);
        if(File.isDirectory(path+thisName) && partInFile>-1){
            if(nResults!=0)IJ.deleteRows(0,nResults);//start with clean Results slat
            close("*");//close any remaining open images
            if(roiManager("count")>0){//remove any remaining ROIs in the manager
                roiManager("Deselect");
                roiManager("Delete");
            }
            //let zip file open all associated images and activate their rois
            //will disable the selection of images later on so do not use setBatchMode(true);
            pf=processZipfile(path + thisName ,blue );
            if(nImages()!=0){
                run("Tile");//show an overview of all images with their ROI
                processedFiles = processedFiles + pf;
                //loop through roi names (they equal (now opened) image window names!)
                totalWidth = 0;
                nRois=roiManager("count");
                stripIDs=newArray(nRois);//bookkeeping for fast access of images
                stripWidths = newArray(nRois)//bookkeeping for total width and starting positions
                for(roiNr = 0; roiNr<nRois;roiNr++){
                    imgName=call("ij.plugin.frame.RoiManager.getName", roiNr);//get image title from ROI
name

                    selectImage(imgName);//select the appropriate image
                    roiManager("select",roiNr);//get its ROI into the image display
                    run("Interpolate", "interval=1");//also with sparse ROIs we need to pick up all points.
                    myFlip=flipNeeded(direction,roiNr); //see function for explanation
                    //straighten curved ROIs into a straight strip(e) of pixels
                    stripName = "strip_"+IJ.pad(stripHeight,2) +"_"+ IJ.pad(roiNr+1,2)+".tif";
                    run("Properties... ", "  width="+stripHeight);//set line width to desired size
                    run("Straighten...", "line="+stripHeight+" title="+stripName);
                    //flip if necessary
                    if(myFlip!=0){
                        run("Flip Horizontally");
                    }
                    stripIDs[roiNr]=getImageID();//for numeric reference
                    //save roi at full size with name stripX, where X is [01,..]
                    setOption("ScaleConversions", false);//we don't want scaling of grey values
                    run("8-bit");//we want raw 8-bit values!
                    fn=path+thisName+blue+stripName;
                    saveAs("Tiff", fn);
                    getDimensions(width, height, channels, slices, frames);
                    //(keep roi open)
                    stripWidths[roiNr] = width;
                    totalWidth = totalWidth + width;
                }
                //anneal all rois to one big strip @height=stripHeight

                newImage("strip", "8-bit black", totalWidth, stripHeight, 1);
                montageID=getImageID();
                leftPos=0;
```

```
                    for(stripNr = 0;stripNr<nRois;stripNr++){
                        //select strip
                        selectImage(stripIDs[stripNr]);
                        run("Select All");
                        run("Copy");
                        //select montage
                        selectImage(montageID);
                        //create paste rectangle
                        sw=stripWidths[stripNr];
                        makeRectangle(leftPos,0,sw,height);
                        run("Paste");
                        leftPos=leftPos + sw;
                    }
                    //save annealed image
                    thisUName = replace(thisName,"/","_");
                    fn = thisName +blue+ thisUName + "montage_"+IJ.pad(stripHeight,2) + ".tif";
                    fn = path + fn;
                    saveAs("Tiff",fn);
                }
            }
        }
    }
    close("*");
    return 0;
}
function processZipfile(inPath,blue){
        processedFiles=0;
        arrayOfNames = getFileList( inPath );
        for(i=0;i<arrayOfNames.length;i++){
            thisName = arrayOfNames[i];
            if(endsWith(thisName,"roiSet.zip")){
                //make total intensity fluorescence in the column
                //save as 1000 values
                // use the sum of fluorescence per 1/100 and per 1/1000 of the total length of the edges.
                processName = inPath + thisName; //shows up in Debug window
                processedFiles = processedFiles + openRoiZipImages(processName,blue,0);
                a=5;//debugger decoy, useless assignment needed for variable inspection during single stepping
                //now all images belonging to this roiManager zip file are open with the ROIs selected
            }
        }
        return    1;
    }

function openRoiZipImages(path,blue,makeHandles){
    //open a zip file (point at the zip file using Finder dialog)
    //zip file contains rois, one roi per image, with roi name is image(file) name
    //open every image by name, in same folder as zip file, name taken from roiManager entry

    roiZip=File.getName(path);
    dir = replace(path,roiZip,"");//this leaves the directory separator in place
    roiManager("Open", path);
    for(iRoi=0;iRoi<roiManager("count");iRoi++){
        imName=call("ij.plugin.frame.RoiManager.getName", iRoi);
        if(!isOpen(imName)){
            //attempt to importimage by that name
            importImageName=dir+blue+imName;
            if(File.exists(importImageName)){
                open(importImageName);
                selectImage(imName);
                roiManager("select",iRoi);
                if(makeHandles){//convert the possible fine grained, or very coarse ROI to smooth handles
```

```
                    Roi.getCoordinates(xpoints,ypoints);
                    makeSelection("freeline",xpoints,ypoints);
                    run("Interpolate", "interval=10  adjust"); //put in adjustment handles
                }
                run("Enhance Contrast", "saturated=0.35");//in case of faint images
            }
        }
    }
    return 1; //one zip file processed
}
function flipNeeded(direction,roiNr){
    Roi.getCoordinates(xpts,ypts);
    getDimensions(width, height, channels, slices, frames);
    dtet = 10;//distance-to-edge tolerance
    l=xpts.length-1;//last coordinate
    determined = false;
    index = 0;
    sc=0;
    toofar=false;
    direction = toUpperCase(direction);
    do{
        pd = substring(direction,index,index+1);//primary direction
        if( pd == "T"){
            if(ypts[0]<ypts[l] ){ //y[0] is nearest tot top
                if(ypts[0] < dtet) { //y[0] is close enough to image's top edge
                    sc = 0;
                    determined = true;
                }else{//not close enough, try again
                    toofar=true;
                }
            }else{     //ypts[l] is close to the top
                if(ypts[l] < dtet) { //y[l] is close enough to image's top edge
                    sc = l;
                    determined = true;
                }else{//not close enough, try again
                    toofar=true;
                }
            }
        }
        if(pd == "B"){
            if(ypts[0]>ypts[l] ){ //y[0] is nearest tot bottom
                if(ypts[0] > (height - dtet)) { //y[0] is close enough to image's bottom edgee
                    sc = 0;
                    determined = true;
                }else{//not close enough, try again
                    toofar=true;
                }
            }else{     //ypts[l] is close to the bottom
                if(ypts[l] > (height - dtet)) { //y[l] is close enough to image's bottom edge
                    sc = l;
                    determined = true;
                }else{//not close enough, try again
                    toofar=true;
                }
            }
        }
        if(pd == "L"){
            if(xpts[0] < xpts[l] ){ //x[0] is nearest tot left edge
                if(xpts[0] < dtet) { //y[0] is close enough to image's left edgee
                    sc = 0;
                    determined = true;
                }else{//not close enough, try again
                    toofar=true;
                }
            }else{    //xpts[l] is nearest to the left edge
                if(xpts[l] < dtet) { //x[l] is close enough to image's left edge
                    sc = l;
                    determined = true;
                }else{//not close enough, try again
                    toofar=true;
                }
            }
        }
        if(pd == "R"){
            if(xpts[0] > xpts[l] ){ //x[0] is nearest tot right edge
                if(xpts[0] > (width - dtet)) { //y[0] is close enough to image's right edge
                    sc = 0;
                    determined = true;
                }else{//not close enough, try again
                    toofar=true;
                }
            }else{    //xpts[l] is nearest to the left edge
                if(xpts[l] > (width - dtet)) { //x[l] is close enough to image's rightt edge
                    sc = l;
                    determined = true;
                }else{//not close enough, try again
                    toofar=true;
                }
            }
        }
        index++;
    }while(index < lengthOf(direction) && determined == false);
    if(!determined && toofar){
        //use [0] or [l] based on first character of direction, disregard distance to edge
        pd = substring(direction,0,1);flipstring = "flip the direction";flip=0;
        if(pd == "T"){if(ypts[0]<ypts[l])flip = 0;else flip = 1;}
        if(pd == "B"){if(ypts[0]>ypts[l])flip = 0;else flip = 1;}
        if(pd == "L"){if(xpts[0]<ypts[l])flip = 0;else flip = 1;}
        if(pd == "R"){if(xpts[0]>ypts[l])flip = 0;else flip = 1;}
        if(flip ==1)flipstring = "not "+flipstring;
        //waitForUser("far away","based on "+pd+" we chose to "+flipstring);
        return flip * l;
    }
    return sc != 0;
}

//analyse strips.txt
requires("1.52p");
var debugger=true;

/* Last change 20190930 11:25
 * written by a.jonker@amsterdamumc.nl
 *
 * Input:
 * image representing (green) fluorescent lipid droplets
 * image representing (blue) nuclear signal
 * Output: csv file
 *
 * Images contain straightened ROIs with 4 zones of equal size (usually 10 pixels high, arbitrary wide)
 * Images blue and green contain in 4 zones respectively tissue background, droplet signal, background: 1 near and 1 far from tissue edge
 * Analyse Strips macro detects if tissue is in upper or lower part.
 * Green signal in tissue background is considered background, mean is measured and used as lower threshold in tissue edge.
 * Same procedure is followed for measuring blue tissue background and tissue edge.
 * Results of threshold value, net signal, and area (expressed in pixels) are collected for green and blue.
 * Measurement results are saved to Excel-readable comma separated file
 *
 * For visual inspection, the variable 'debugger' can be set to 'true', after which the macro is executed slowly
 * If debugger is set to false, processing takes a few minutes.
 *
 *
 *
 */




macro "Analyse Strips"{
	close("*");
	pattern = "montage_80.tif";//later: montage_"+IJ.pad(stripHeight,2);
	normaliseTo=100;
	startPath = getDirectory("Choose a Directory");
	nFiles=analyseStrips(startPath,pattern);
	print("Processed "+nFiles+" files");
}



function analyseStrips(path,pat){
	arrayOfNames = getFileList(path );
	nSubstrips=4;
	processedFiles=0;
	substripVals=newArray(nSubstrips);
	run("Set Measurements...", "mean modal min integrated limit redirect=None decimal=0");
	for(i=0;i<arrayOfNames.length;i++){
		thisName=arrayOfNames[i];
		if(File.isDirectory(path+thisName) && indexOf(thisName,"blue")==-1){
			processedFiles = processedFiles + analyseStrips(path + thisName,pat);
		}
	}
	for(iNames=0;iNames<arrayOfNames.length;iNames++){
		thisName=arrayOfNames[iNames];
		updateResults();
		blueDaughter=path+"blue/"+thisName;
		parentName=File.getName(path);
		if(indexOf(thisName,pat)!= -1 && indexOf(thisName,".tif")!=-1 && File.exists(blueDaughter)){
			print("\\Clear");
			print("\\Update0:processing " + parentName);
			run("Clear Results");
			close("*");
			//open strip
			pn="blue/"+thisName;
			pn=path+pn;
			open(pn);
			blueImageID=getImageID();
			rename("Blue");
			run("Select All");;
			run("Duplicate...", "title=bluegauss"); //if we can't make nice outlines
			dupGauss=getImageID();
			getDimensions(width, height, channels, slices, frames);
			//these arrays hold coordinates for a polyLine ROI that separates cells in foreground from tissue background
			var edgeArrayY = newArray(width);//add 2 for corners of enclosed area
			var edgeArrayX = newArray(width);
```

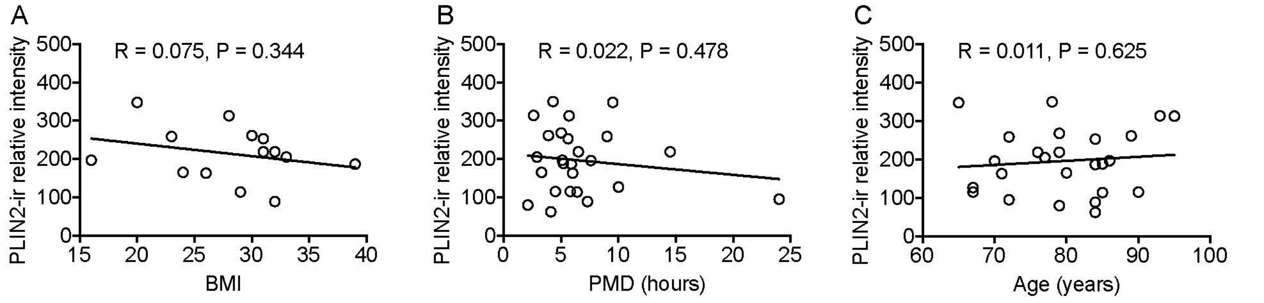
```
    var sigAreaArray = newArray(width); //area for blue and green is the same.
    var blueSigValueArray = newArray(width);//background area is heigt/2 -SigArea
    var bgAreaArray = newArray(width); //is not strictly necessary, can be calculated from sigAreaArray
    var blueBgValueArray = newArray(width);

    var greenSigValueArray = newArray(width);//area is same as blue
    var greenBgValueArray = newArray(width); //area is same as blue


    open(path+thisName);
    greenImageID=getImageID();
    rename("Green");
    run("Tile");
    substripHeight = height / nSubstrips;
    //measure fat droplet strip
    selectImage(greenImageID);//best find tissue using green channel
    for(iStrips=0;iStrips<nSubstrips;iStrips++){
        makeRectangle(0,substripHeight*iStrips,width,substripHeight);
        meanVal=getValue("Mean");
        substripVals[iStrips]=meanVal;

    }
    //four substrips: either upper 2 or lower 2 represent the tissue
    topVal = substripVals[0]+substripVals[1] ;
    bottomVal =  substripVals[2]+substripVals[3];
    halfHeight = height/2;
    yStart=halfHeight;
    //decide if tissue is in the upper half or lower half of the image
    //if in the lower half,  flip lower and top half
    if(topVal < bottomVal){ //tissue is in the lower part, flip that part vertical, so tissue is in upper part
        selectImage(greenImageID);
        run("Select All");
        run("Flip Vertically");

        selectImage(blueImageID);
        run("Select All");
        run("Flip Vertically");

        selectImage(dupGauss);
        run("Select All");
        run("Flip Vertically");
    }
    //the tissue edge is near the middle of the image.
    //flip top half (containing tissue now) so tissue edge is at top of image; easier for for-loops.

    selectImage(greenImageID);
    makeRectangle(0,0,width,height/2);
    run("Flip Vertically");

    selectImage(blueImageID);
    makeRectangle(0,0,width,height/2);
    run("Flip Vertically");

    selectImage(dupGauss);
    makeRectangle(0,0,width,height/2);
    run("Flip Vertically");

    //first find the division line based on the blue signal. (Then measure foreground and background signals in
both original images.)
    //as selectImageID is very slow, don't measure whiledetecting the division line.
    selectImage(dupGauss);
```

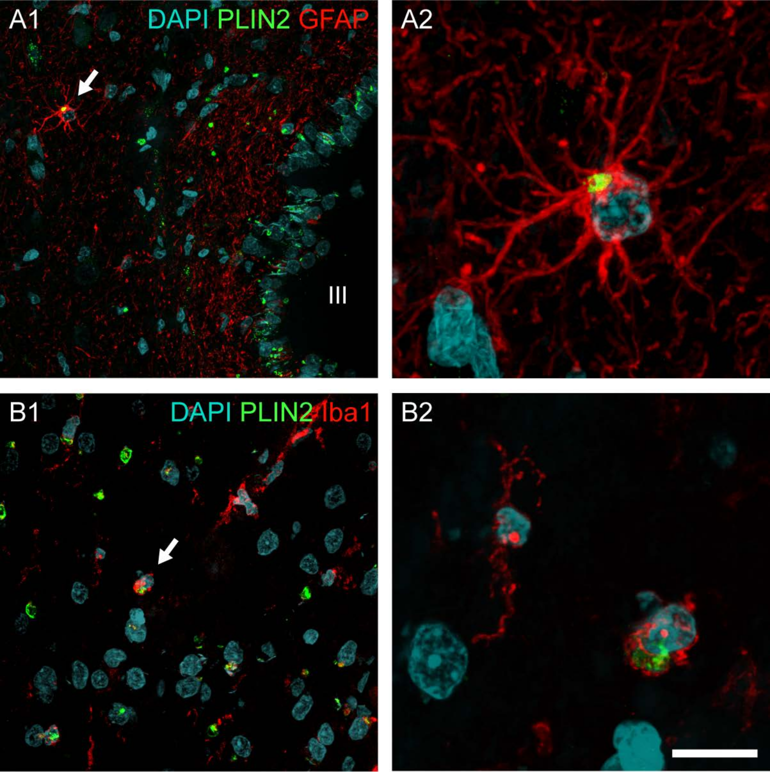
```
            setBatchMode(true);
            for(x = 0; x < width; x++){//walk the strip
                makeRectangle(x,0,1,halfHeight);
                max=getValue("Max");//highest value of the current strip
                halfMax=max/2;
                pv=0; //pixel value
                y=-1; //want increment in loop, want exit value to be pointing at half max location
                do{//find max position of column, there might be more?
                    y++;
                    pv = getPixel(x,y);

                }while(y < halfHeight && pv < max);
                while(y < halfHeight && pv > halfMax){ //walk downhill till half max in column
                    y++;
                    pv = getPixel(x,y);
                }
                //y points to the first position of the background
                //as gathered from the blue channel (= DNA intensity)
                edgeArrayY[x] = y-1; //the foreground was left already, y points to first pixel of background, so edge
should be one bak

                while (y < halfHeight){//while loop needs to be here, if y == halfHeight skip this loop
                    //walk down hill until end of image
                    pv = getPixel(x,y);
                    y++;
                }
                edgeArrayX[x]=x;
            }


            for(x = 0; x < width; x++){//set ROI to shape in original image
                sigAreaArray[x] = edgeArrayY[x];//easiest moment to extract signal area
                if(topVal > bottomVal){//flip  coordinates of signal rectangle wrt. top half
                    edgeArrayY[x] = halfHeight - edgeArrayY[x];
                }else{//flip  coordinates of signal rectangle wrt. bottom half
                    edgeArrayY[x] = halfHeight + edgeArrayY[x];
                }
                edgeArrayX[x]=x;
            }

            selectImage(blueImageID);
//warning: using sigArea!
            makeSelection("polyline",edgeArrayX,sigAreaArray);
//warning: used sigArea!
            for(x = 0; x < width; x++){//set ROI to shape of original image
                makeRectangle(x,sigAreaArray[x],1,halfHeight-sigAreaArray[x]);
                blueBgValue=getValue("IntDen");
                blueBgValueArray[x] = blueBgValue;

                makeRectangle(x,0,1,sigAreaArray[x]);
                blueSigValue=getValue("IntDen");
                blueSigValueArray [x] = blueSigValue;
            }

            selectImage(greenImageID);
            for(x = 0; x < width; x++){//set ROI to shape of original image
                makeRectangle(x,sigAreaArray[x],1,halfHeight-sigAreaArray[x]);
                greenBgValue=getValue("IntDen");
                greenBgValueArray[x] = greenBgValue;

                makeRectangle(x,0,1,sigAreaArray[x]);
                greenSigVal=getValue("IntDen");
                greenSigValueArray[x] = greenSigVal;
            }
            if(debugger){setBatchMode(false);debug;}

            //flip back tissue edge to middle of image, so it resembles original again
            selectImage(greenImageID);
            makeRectangle(0,0,width,height/2);
            run("Flip Vertically");

            selectImage(blueImageID);
            makeRectangle(0,0,width,height/2);
            run("Flip Vertically");

            selectImage(dupGauss);
            makeRectangle(0,0,width,height/2);
            run("Flip Vertically");
            //if tissue was in lower half of image, restore by flipping entire image vertically
            if(topVal < bottomVal){ //tissue is in the lower part, flip that part vertical, so tissue is in upper part
                selectImage(greenImageID);
                run("Select All");
                run("Flip Vertically");

                selectImage(blueImageID);
                run("Select All");
                run("Flip Vertically");

                selectImage(dupGauss);
                run("Select All");
                run("Flip Vertically");
            }
            //draw edge of signal
            selectImage(blueImageID);
            makeSelection("polyline",edgeArrayX,edgeArrayY);
            selectImage(greenImageID);
            makeSelection("polyline",edgeArrayX,edgeArrayY);
            if(roiManager("count")>0){
                roiManager("Deselect");
                roiManager("Delete");
            }
            parent = File.getName(path);
            roiManager("add");
            roiManager("select",roiManager("count")-1);
            roiManager("rename",parent);

            roiManager("save",path+parent+"edge.roi");


            fn = parent + "_normalised_"+greenSigValueArray.length+".csv";fn =  path + fn;
            l=greenSigValueArray.length;

reportResults(fn,greenSigValueArray,greenBgValueArray,blueSigValueArray,blueBgValueArray,sigAreaArray,l);
            fn = parent + "_normalised_"+"100"+".csv";fn =  path + fn;

reportResults(fn,greenSigValueArray,greenBgValueArray,blueSigValueArray,blueBgValueArray,sigAreaArray,100);
            fn = parent + "_normalised_"+"10"+".csv";fn =  path + fn;

reportResults(fn,greenSigValueArray,greenBgValueArray,blueSigValueArray,blueBgValueArray,sigAreaArray,10);
            close("*");
            processedFiles++;
        }
```

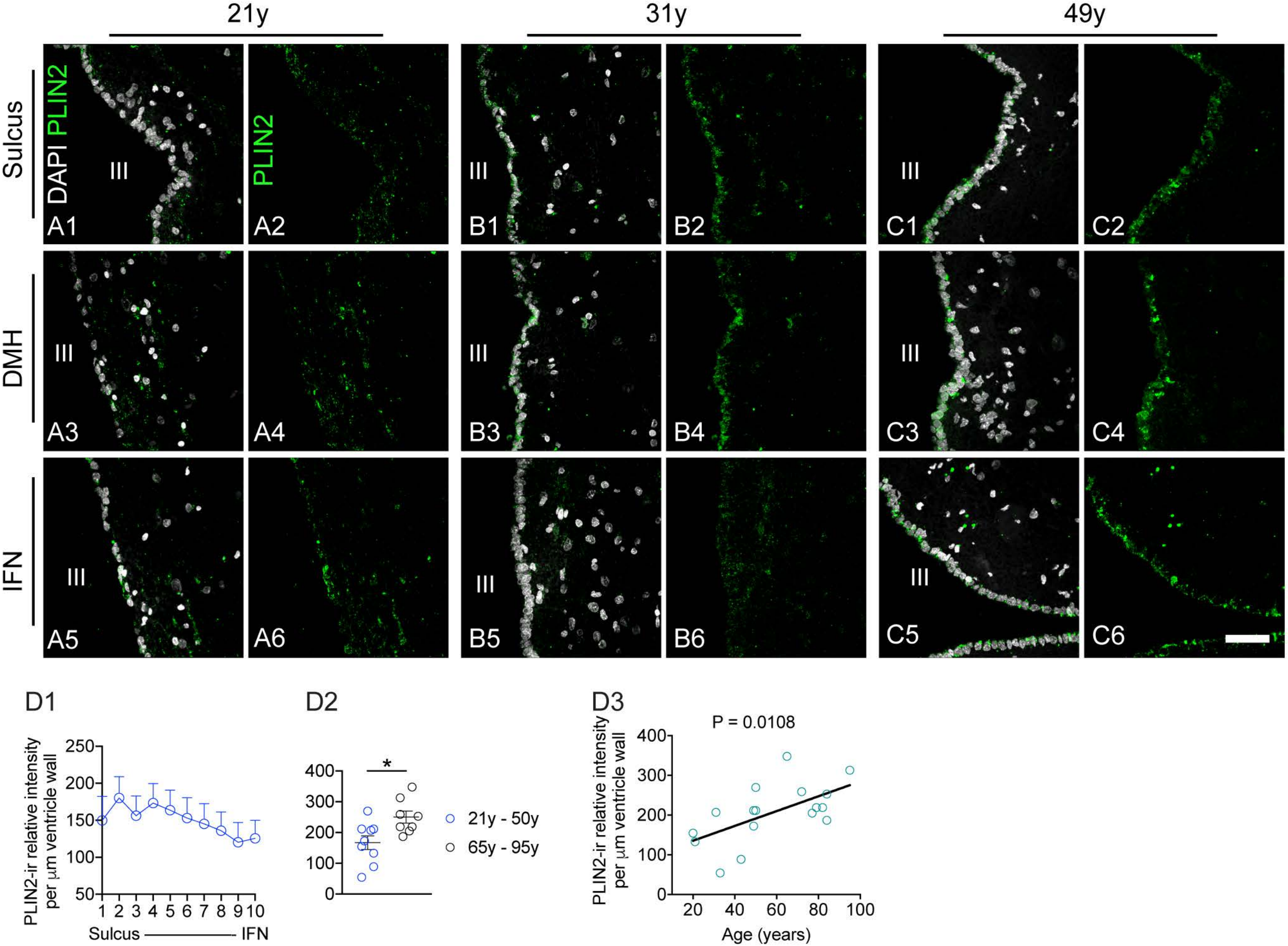
```
    }
    return processedFiles;
}
function reportResults(pfn,gs,gb,bs,bb,sa,normaliser){
    chunk = floor(gs.length/normaliser);counter = 0;
    gsx=0;gbx=0;bbx=0;sax=0;bsx=0;//running counters for normalising to normaliser values in output
    run("Clear Results");
    row = 0;
    for(x=0;x<gs.length;x++){
        gsx=gsx+gs[x];
        gbx=gbx+gb[x];
        bbx=bbx+bb[x];
        sax=sax+sa[x];
        bsx=bsx+bs[x];
        counter ++;
        if(counter >= chunk){
            setResult("Green Signal",row,gsx);
            setResult("Green Background",row,gbx);
            setResult("Blue Background",row,bbx);
            setResult("Signal Area",row,sax);
            setResult("Blue Signal",row,bsx);
            updateResults();
            counter = 0;gsx=0;gbx=0;bbx=0;sax=0;bsx=0;
            row++;
        }
    }
    //export results
    saveAs("results", pfn);
```

Supplementary Table. Clinico-pathological data of younger subjects.

NBB Number	Sex	Age	Braak	PM	BMI	Cause of death and clinical diagnosis
1994-040	m	20	0	8.0	25	Heart failure, B cell lymphoma, viral pneumonia
2001-009	f	21	/	19.5	/	Myocard infart
1991-005	m	31	/	34	23	Necrotic tumor, adenocarcinoma
1984-026	f	33	/	24.75	22	Anoxia, lung oedema, tracheobronchitis, brain death
1980-006	f	43	/	50	/	Renal metastatic carcinoma
1994-118	m	49	0	/	20	Sepsis (E.coli), coloncarcinoma with metastasis
2011-069	m	49	0	6.5	24	Legal euthanasia, Hodgkin`s lymphoma, urolithiasis
1982-008	f	50	/	52.75	23	Coma; bronchopneumonia, carcinoma
1998-091	f	50	0	41	19	Legal euthanasia, metastatic Pancoust tumor


A
R = 0.075, P = 0.344
PLIN2-ir relative intensity
500
400
300
200
100
0
15
20
25
30
35
40
BMI
B
R = 0.022, P = 0.478
PLIN2-ir relative intensity
0
5
10
15
20
25
PMD (hours)
C
R = 0.011, P = 0.625
PLIN2-ir relative intensity
60
70
80
90
100
Age (years)


A1
DAPI PLIN2 GFAP
III
A2
B1
DAPI PLIN2 Iba1
B2


21y
31y
49y
Sulcus
DMH
IFN
DAPI PLIN2
PLIN2
III
A1
A2
A3
A4
A5
A6
B1
B2
B3
B4
B5
B6
C1
C2
C3
C4
C5
C6
D1
PLIN2-ir relative intensity per μm ventricle wall
250
200
150
100
50
1 2 3 4 5 6 7 8 9 10
Sulcus — IFN
D2
*
400
300
200
100
0
21y - 50y
65y - 95y
D3
P = 0.0108
PLIN2-ir relative intensity per μm ventricle wall
400
300
200
100
0
20 40 60 80 100
Age (years)