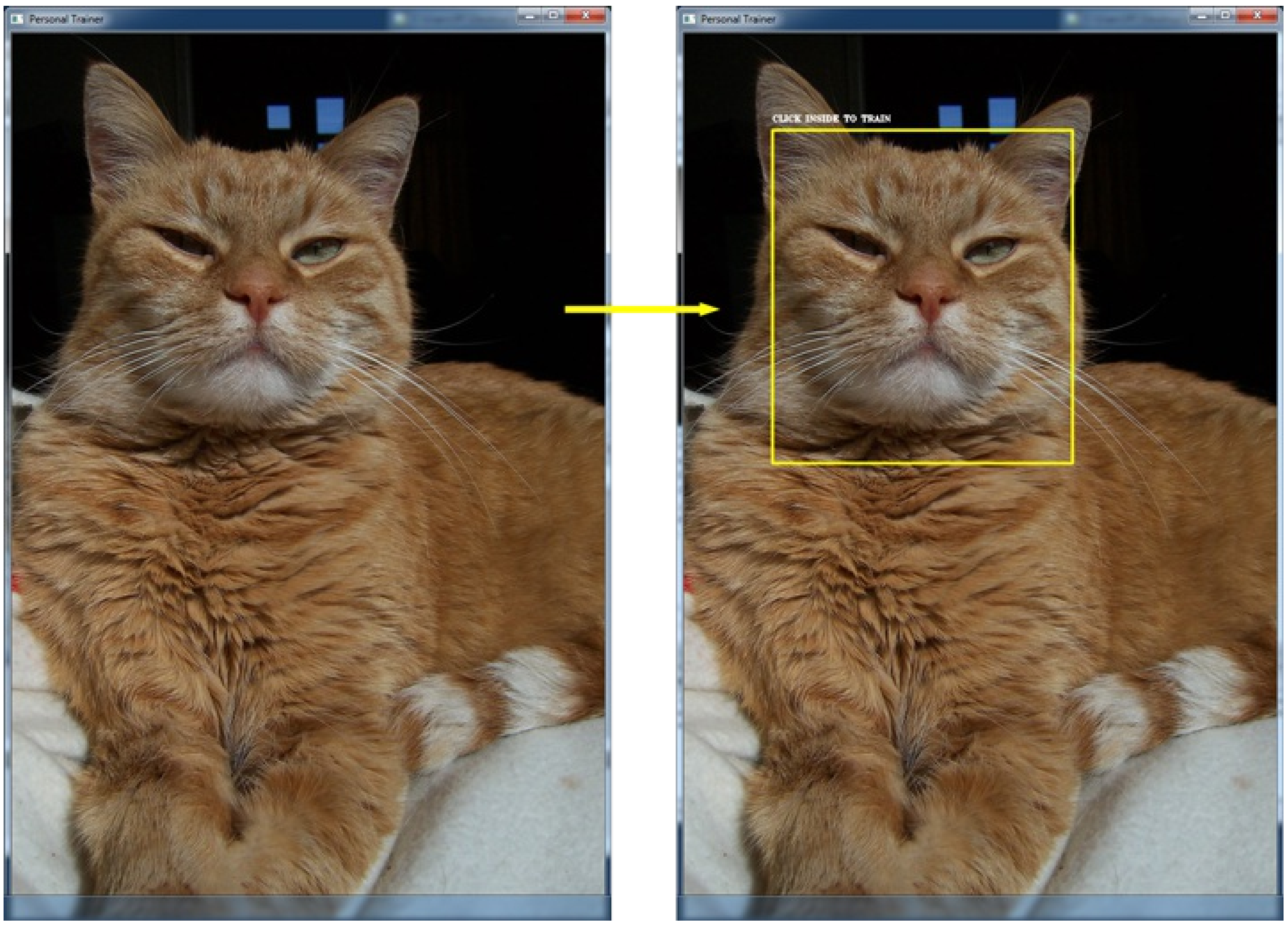
Personal Trainer

1370838614

When training a custom cascade classifier with opencv we need to provide a log (or info) file for the positive entry, these are the files which the objects we want to train the classifier to identify actually exist in, along with one for the negative entry, the challenge with generating the first file is that it needs to contain the information of where the object is placed in the image – the rect which bound the image, something like the following:

```
pos/00000899_022.jpg  1  143 119 386 315
pos/00000888_005.jpg  2  112 103 448 450
pos/00000889_002.jpg  3  156 80 406 324
pos/00000889_006.jpg  4  140 74 384 284
pos/00000889_008.jpg  5  76 70 331 296
pos/00000889_012.jpg  6  178 110 289 208
pos/00000889_013.jpg  7  228 77 395 225
pos/00000889_016.jpg  8  401 126 695 418
pos/00000889_018.jpg  9  286 120 439 265
pos/00000889_022.jpg  10  63 41 361 305
pos/00000889_023.jpg  11  89 59 243 202
pos/00000889_026.jpg  12  426 281 746 613
```

so if you want to write a classifier that will learn how to identify if someone is eating a ice cream in an image (or if there is a ice cream in the picture) you'll need first to have a database of ice cream and people eating ice cream images and then you'll need to save an entry for each file in your collection. obviously since I am more concern with cats rather than ice cream images I chose to go ahead and download this cat images database of more than 2gig of pure cat images, designed for image training, the great thing about this database is that it also comes with a map file for each cat that define the shape of it face. but lets ignore that for now and go back to the original problem, what if we want to create a new database for our cats.



So I wrote a simple c++ utility which I call personal trainer, you'll see why when you use it, it gets very personal after the 400 image or so. the app lets you define an images directory, and then run through the file in side it one by one – allowing you to crop the area which you'll like to save an object and sort it in a log file, you can grab the full code from this git page – https://github.com/screename/Personal-Trainer.git

here is what the trainer is based on, first we read the content of the directory using the tinydir lib which makes it very easy to handle directories in c++ (for me at least)

C++

```
//list all files in a directory using tinydir
//we'll use this to read all the files in the pos images folder
void listDir(const char *destination, vector<string> &labels ) {

    tinydir_dir dir;
    //check if the directory exist or return an error
    if (tinydir_open(&dir, destination) == -1){
        printf("%s", destination);
        perror("Error opening file");
        goto bail;
    }

    //loop through what we have found in the dir
    while (dir.has_next) {
        tinydir_file file;
        if (tinydir_readfile(&dir, &file) == -1){
            perror("Error getting file");
            goto bail;
        }

        if (file.is_dir){
//          ignore sub folders
//            printf("/");
        } else {
//            printf("%s", file.name);
//            push the file name into the labels array
            labels.push_back( file.name );
        }
//        printf("\n");
//        check the next file
        tinydir_next(&dir);
    }

//    we'll triger this in case of errors
    bail:
        tinydir_close(&dir);

}
```

Then we run a call to load the next – first image and assign the mouse callbacks for it (which will draw the rect on the image)

C++

```
//load an image and set it as the current one
void loadImage(String filename) {
   sourceMat = imread(filename, 1);
   sourceMat.copyTo(currentImage);
   //display the image in our app window
   imshow(WINDOW_NAME, currentImage);
   //and set the mouse callbacks
   setMouseCallback(WINDOW_NAME, onMouse, 0);
}

//load the next item on the list
//or shutdown the app if the training is completed
void loadNext() {

   int len = posLabels.size();
   cout << currentLogIndex << " " << currentIndex << " - " << len << endl;
   if ( currentIndex <= len - 1 ) {
      //up the indexes
      currentLogIndex++;
      currentIndex++;
      cout << "loading image " <<  currentIndex + 1  << " of " << len << endl;
      stringstream  stream(posLabels.at(currentIndex));
      stream >> currentImageFile;
      loadImage(POS_IMAGES_DIR + currentImageFile);
   } else if ( currentIndex > len ) {
      //done with training, lets shutdown.
      cvDestroyWindow( WINDOW_NAME );
      cout << "--> Training completed "  << currentIndex + 1 << " of " << len << endl;
   }
}
```

The cat and mouse game

This is where opencv makes it fun to handle images, with the built in events for any of the mouse gestures. we'll define 3 variable which will handle the drawing and clicking.

drawing_box – a bool that will notify us if we are currently drawing or not.

mouseBox – a rect which we'll use to hold the mouse drawing position

train_box_on - a bool which will tell us if there is a rectangle already drawn on the image

C++

```
//a rectangle to measure the mouse movement
Rect mouseBox;
//are we drawing or not?
bool drawing_box = false;
//do we have a rect on the image already?
bool train_box_on = false;
```

we'll switch between 3 different mouse events: down, up and move.

CV_EVENT_LBUTTONDOWN - when the mouse is down we'll either reset the rectangle or if it is clicked inside an existing rectangle we'll use that as an indication to save the current position.

CV_EVENT_MOUSEMOVE – when the mouse is moving, if we are currently in drawing mode, we'll add the mouse pos to the mouseBox rect.

CV_EVENT_LBUTTONUP – when the mouse is up, we'll find out where our rect ends and draw it over the image, if we had only a click, we’ll use that as an indication to reset any existing rectangle.

C++

```
//handle the mouse events
void onMouse(int event, int x, int y, int, void*) {

    switch( event ){
        //draw the rectangle if the mouse was clicked befor the move
        case CV_EVENT_MOUSEMOVE:
            if( drawing_box ){
                mouseBox.width = x-mouseBox.x;
                mouseBox.height = y-mouseBox.y;
            }
            break;

        //on mouse down:
        //either define a new rectangle
        //or save the current one
        case CV_EVENT_LBUTTONDOWN:
             if (train_box_on) {
                //check if the click was inside the train box
                if ( x >= mouseBox.x && x <= mouseBox.x + mouseBox.width ) {
                    if ( y >= mouseBox.y && y <= mouseBox.y + mouseBox.height  ) {
//                          snap the pictuer and load the next image
                        saveCurrentImage();
//                          and load the next one
                        loadNext();
                    }
                }
            }
            //if clicked out side the current rectangle,
            //start drawing a new one instead
            drawing_box = true;
            mouseBox = Rect( x, y, 0, 0 );
//              reset the current image to the original
            sourceMat.copyTo(currentImage);
            break;

        //on mouse up
        //finish the rect and draw it on the image
        case CV_EVENT_LBUTTONUP:
            drawing_box = false;
            if( mouseBox.width < 0 ){
                mouseBox.x += mouseBox.width;
                mouseBox.width *= -1;
            }
            if( mouseBox.height < 0 ){
                mouseBox.y += mouseBox.height;
                mouseBox.height *= -1;
            }

//              if we have a valid rect lets draw it on the image
            if (mouseBox.width > 0 && mouseBox.height > 0 ) {
                train_box_on = true;
                highlight( currentImage, mouseBox );
            } else {
                //or set the mouse rect to invald
                mouseBox  = Rect(-1,-1,-1,-1);
                train_box_on = false;
            }
            imshow(WINDOW_NAME, currentImage);
            break;
    }

}
```

When we highlight the object we'll also add a call to action to click inside to save the current position.

C++

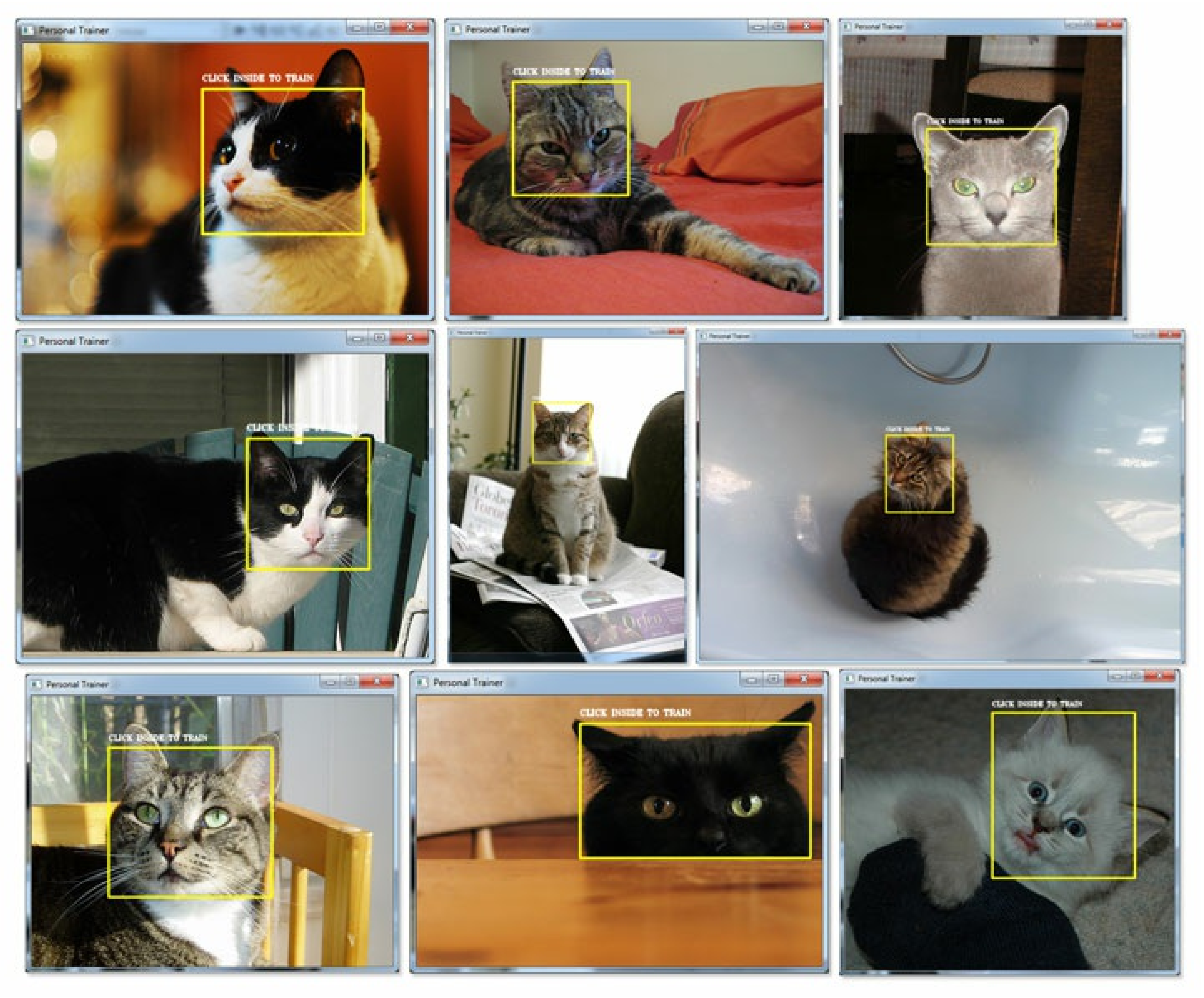
```cpp
//highlight the selected area and display the cta on top
void highlight( Mat img, Rect mouseRect){
   rectangle(img, mouseRect, CV_RGB(255,255,0), 2, CV_AA);
   putText(img, CAPTURE_CTA, Point( mouseRect.x, mouseRect.y - 10 ),
   CV_FONT_HERSHEY_TRIPLEX, 0.35, cvScalar(255,255,255), 0, CV_AA);
}
```

lets save the rect into our log file and we are close to done.

C++

```cpp
// save the current image in the following format
// dir/filename.ext index x y x+w y+h
// pos/img1.jpg  1  140 100 45 45
void saveCurrentImage() {
//construct a string based on the current file name, index and the mouse box rect
  string out = toString(POS_IMAGES_FOLDER) + currentImageFile +"  " + toString(currentLogIndex + 1 )+
"  " + toString(mouseBox.x) + " " + toString(mouseBox.y) + " " + toString(mouseBox.x+mouseBox.width) +
" " + toString(mouseBox.y+mouseBox.height);
  ofstream myfile;
  myfile.open (LOG_FILE_NAME, fstream::in | fstream::out | fstream::app);
  myfile << out + "\n";
  myfile.close();
}
```

done? well – now we have time to go through another 1000 cat images and build our data, here are some of the highlights, taken from the cat images database.



Cascade Training

now you should have a log file with a similar structure to the one outlined above, you'll also need a log file for the negative sample, the one that don't contain your object, this could be a simple file that list the images in the directory you sort them, something like this:

```
neg/001.jpg
neg/002.jpg
neg/003.jpg
neg/004.jpg
neg/005.jpg
neg/006.jpg
neg/007.jpg
...
```

now you'll need to generate a .vec file which will include the results of both files, we'll use the opencv_createsamples.exe and the opencv_traincascade.exe to build this file and train the classifier (you can find the both in the bin dir - C:\opencv\release\bin), you can see the full documentation and specification of the samples and the trainer at the opencv cascade training page.

createsamples.exe will take the following arguments

```
C:\opencv\release\bin>opencv_createsamples.exe
Usage: opencv_createsamples.exe
  [-info <collection_file_name>]
  [-img <image_file_name>]
  [-vec <vec_file_name>]
  [-bg <background_file_name>]
  [-num <number_of_samples = 1000>]
  [-bgcolor <background_color = 0>]
  [-inv] [-randinv] [-bgthresh <background_color_threshold = 80>]
  [-maxidev <max_intensity_deviation = 40>]
  [-maxxangle <max_x_rotation_angle = 1.100000>]
  [-maxyangle <max_y_rotation_angle = 1.100000>]
  [-maxzangle <max_z_rotation_angle = 0.500000>]
  [-show [<scale = 4.000000>]]
  [-w <sample_width = 24>]
  [-h <sample_height = 24>]
```

running the following command will generate the cats.vec file

```
opencv_createsamples.exe -vec "C:\Path\to\your\files\cats.vec" -info "C:\Path\to\your\files\log.txt" -bg
"C:\Path\to\your\files\neg.txt" -w 24 -h 24 -num 1000
```

now we'll use the data in the vec file to train our classifier and save an xml file with the results, if you had a chance to check out the opencv docs, you'll see that there are two type of training you can do. Haar training which is much slower yet much more accurate, it could be based on the viola jones algorithm approach or the LBP - Local Binary Patterns which uses integers as features for faster processing, note that you can get great results with LBP it is mainly depend on the quality of the training you do, the number of sources you use and how well they are mapped. if you run the opencv_traincascade.exe path into your cmd/terminal window you'll see the following specs

```
C:\opencv\release\bin>opencv_traincascade.exe
Usage: opencv_traincascade.exe
  -data <cascade_dir_name>
  -vec <vec_file_name>
  -bg <background_file_name>
  [-numPos <number_of_positive_samples = 2000>]
  [-numNeg <number_of_negative_samples = 1000>]
  [-numStages <number_of_stages = 20>]
  [-precalcValBufSize <precalculated_vals_buffer_size_in_Mb = 256>]
  [-precalcIdxBufSize <precalculated_idxs_buffer_size_in_Mb = 256>]
  [-baseFormatSave]
--cascadeParams--
  [-stageType <BOOST(default)>]
  [-featureType <{HAAR(default), LBP, HOG}>]
  [-w <sampleWidth = 24>]
  [-h <sampleHeight = 24>]
--boostParams--
  [-bt <{DAB, RAB, LB, GAB(default)}>]
  [-minHitRate <min_hit_rate> = 0.995>]
  [-maxFalseAlarmRate <max_false_alarm_rate = 0.5>]
  [-weightTrimRate <weight_trim_rate = 0.95>]
  [-maxDepth <max_depth_of_weak_tree = 1>]
  [-maxWeakCount <max_weak_tree_count = 100>]
--haarFeatureParams--
  [-mode <BASIC(default) | CORE | ALL
--lbpFeatureParams--
--HOGFeatureParams--
```

The following will generate cascade training xml file using the data we collected and LBP trainer.

```
opencv_traincascade.exe -data "C:\Path\to\your\files\" -vec "C:\Path\to\your\files\cats.vec"
-bg "C:\Path\to\your\files\neg.txt" -numPos 1000 numNeg 1000 -w 24 -h 24 -featureType LBP
```

The structure of the LBP cascade xml file will look somewhat similar to this

```
<_>
    <maxWeakCount>4</maxWeakCount>
    <stageThreshold>-1.1592328548431396</stageThreshold>
    <weakClassifiers>
      <!-- tree 0 -->
      <_>
        <internalNodes>
          0 -1 47 -21585 -20549 -100818262 -738254174 -20561 -36865
          -151016790 -134238549</internalNodes>
        <leafValues>
          -0.5601882934570313 0.7743113040924072</leafValues></_>
      <!-- tree 1 -->
      <_>
        <internalNodes>
          0 -1 12 -286003217 183435247 -268994614 -421330945
          -402686081 1090387966 -286785545 -402653185</internalNodes>
        <leafValues>
          -0.6124526262283325 0.6978127956390381</leafValues></_>
      <!-- tree 2 -->
      <_>
        <internalNodes>
          0 -1 26 -50347012 970882927 -50463492 -1253377 -134218251
          -50364513 -33619992 -172490753</internalNodes>
        <leafValues>
          -0.6114496588706970 0.6537628173828125</leafValues></_>
      <!-- tree 3 -->
      <_>
        <internalNodes>
          0 -1 8 -273 -135266321 1877977738 -2088243418 -134217987
          2146926575 -18910642 1095231247</internalNodes>
        <leafValues>
          -0.6854077577590942 0.5403239130973816</leafValues></_></weakClassifiers></_>
```

Finally you are ready to test the cascade training file and see if your object appear in an image or not. first we’ll define a cascadeclassifier and load the cascade xml file we just trained to it.

C++

```
//we'll use this to size down the images we process
const int DETECTION_IMAGE_WIDTH = 320;
const char *catsCascadeFilename = "C:/opencv/data/lbpcascades/cats.xml";
CascadeClassifier catsCascade;

void loadCascadeClassifier(CascadeClassifier &classifier, string &filename, bool exitOnError = true) {

   try {
      classifier.load(filename);
   } catch (cv::Exception &e) {}
   if ( classifier.empty() ) {
      cerr << "ERROR: Couldn't load face detector (";
      cerr << filename <<")!" << endl;
      if (exitOnError) exit(1);
   }

}

std::string ccfilename(catsCascadeFilename);
loadCascadeClassifier( catsCascade, ccfilename, true);
```

next we'll load an image we'll like to test (or a camera feed), re-size it and convert it into a gray mat, this is better for performance to work with smaller images with minimal amount of channels.

C++

```
//resize the image so it is faster to process
//it is better to use an image larger than 240x240 pixels
//but there is no use in using a large image, specially when
//we deal with live camera feed where this will take place over
//a loop, so lets set it to someplace around the 300px
//using the DETECTION_IMAGE_WIDTH var we defined earlier
Mat resizeImage(Mat src) {
   Mat small;
   float scale = src.cols / (float) DETECTION_IMAGE_WIDTH;
   if (src.cols > DETECTION_IMAGE_WIDTH) {
//       resize the image while keeping the proportions
      int h = cvRound( src.rows / scale );
      resize(src, small, Size(DETECTION_IMAGE_WIDTH, h));
   } else {
      small = src;
   }
   return small;
}

//convert the image to a grayscale image for faster processing
Mat getGrayscale(Mat src) {
   Mat gray;
   if ( src.channels() == 3 ) {
//       3 channels to grayscale conversion
      cvtColor(src,gray, CV_BGR2GRAY);
   } else if ( src.channels() == 4 ) {
//       4 channels to grayscalce conversion
      cvtColor(src,gray,CV_BGRA2GRAY);
   } else {
//       the image is prob already a grayscale image
      gray = src;
   }
   return gray;
}


//load the image as a Mat object
Mat src = imread("mycat.jpg", 1);
Mat dst;
//we'll copy the image so we can draw on to it
src.copyTo(dst)

//resize and convert to grayscale
Mat clean = resizeImage( getGrayscale( dst ) );
```

we’ll also equalize the histogram of the image so the calculation happen faster, this is less important when we deal with one image, but when we have a live camera feed that we want to detect, reducing the effort will make a significant difference. I apply a single equalizeHist filter here, though it is recommended when you have a cropped version of an image to divided the image into two since many times a light hits the image from a specific direction resulting with bright and dark side of the image.

C++

```
//equlize the contrast of the histogram,
//in detection of a cropped object it is
//recommanded to equalize the hist on half of the object
//at a time, since many times a light hits the object from
//one angle, so half of it is dark and half of it is bright
Mat equalized;
equalizeHist(clean, equalized);
```

we are ready to call the catsCascade.detectMultiScale and find out if we found any cats in the image.

C++

```
//find the biggest object in an image
vector<Rect> objects;

//the min size for an object
Size minSize(20,20);

float scaleFactor = 1.1f;
int minNeighbors = 4;

//detect the objects
catsCascade.detectMultiScale(
                    equalized, //note that we pass an image which have
                       // been resized, converted to grayscale and equalized
                    objects,
                    searchScaleFactor,
                    minNeighbors,
                    flags,
                    minSize
                    );
```

lets set the results to the original size of the image and verify that it is within the image frame

C++

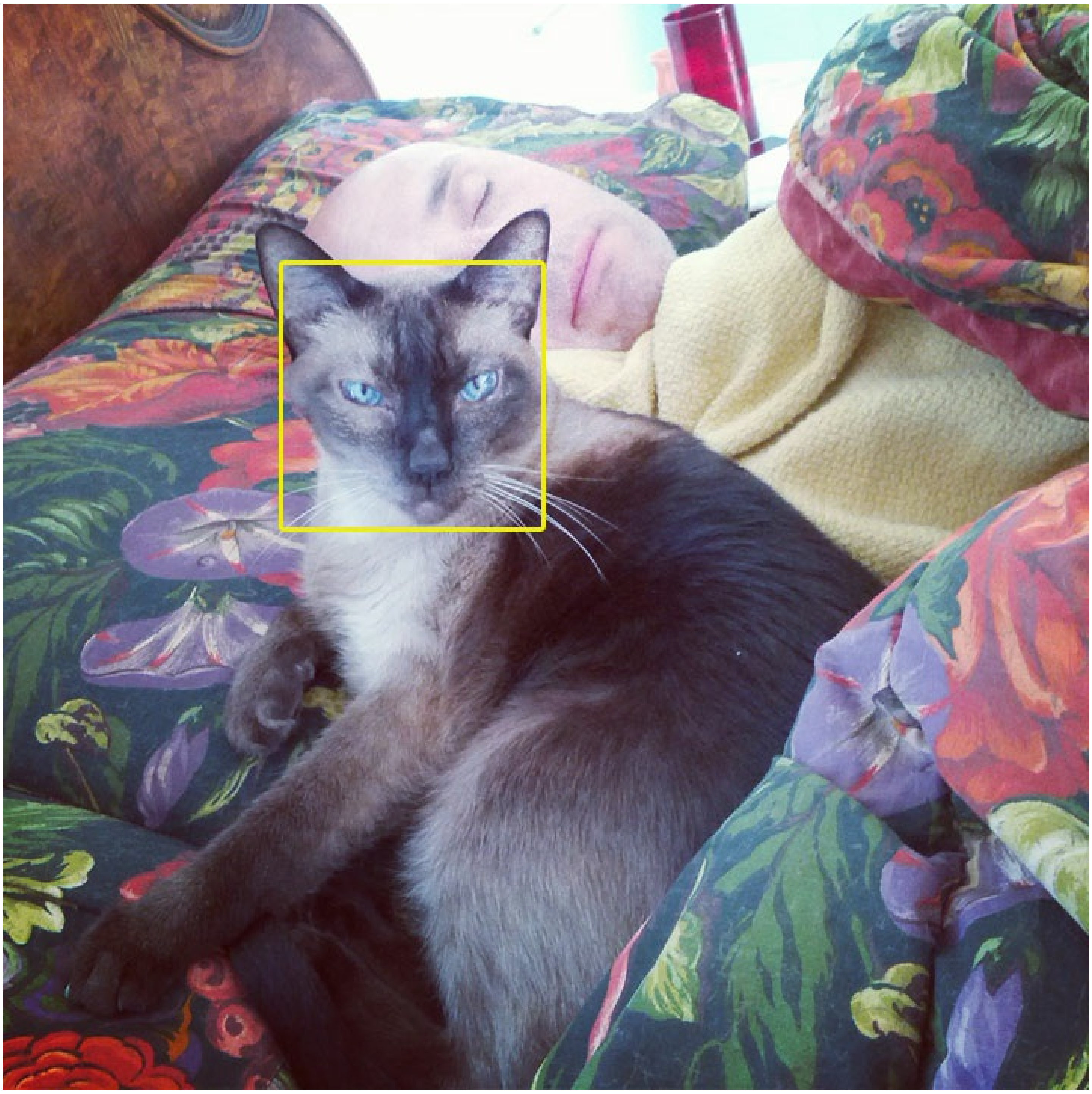
```
//    set the results to the original size of the image
    if (dst.cols > DETECTION_IMAGE_WIDTH) {
        float scale = dst.cols / (float) DETECTION_IMAGE_WIDTH;
        for(int i = 0; i<(int)objects.size(); i++) {
            objects[i].x = cvRound(objects[i].x * scale);
            objects[i].y = cvRound(objects[i].y * scale);
            objects[i].width = cvRound(objects[i].width * scale);
            objects[i].height = cvRound(objects[i].height * scale);
        }
    }

//    verify that the object is within the image borders
    for(int i = 0; i < (int)objects.size(); i++) {
        if (objects[i].x < 0) objects[i].x = 0;
        if (objects[i].y < 0) objects[i].y = 0;
        if (objects[i].x + objects[i].width > dst.cols) {
            objects[i].x = dst.cols - objects[i].width;
        }
        if (objects[i].y + objects[i].height > dst.rows) {
            objects[i].y = dst.rows - objects[i].height;
        }
    }
```

now if we have found an object, lets frame it and show the image

```
if (objects.size() > 0) {
    //draw the detected object
     catRect = (Rect)objects.at(0);
     rectangle(dst, catRect, CV_RGB(255,255,0), 2, CV_AA);
     imshow("Cat", dst);
} else {
    //I guess there is no cat in this image
}
```

here is a test of me and my cat chachi, which was not apart of the original training set.



I hope you find this useful in detecting your own objects with opencv, here are some of the resources I came across/mentioned through my learning of how to detect objects.

personal trainer – https://github.com/screename/Personal-Trainer.git
tinydir file utils – https://github.com/cxong/tinydir
haar training tutorial – http://note.sonots.com/SciSoftware/haartraining.html
opencv train cascade documentation – http://docs.opencv.org/doc/user_guide/ug_traincascade.html
cascade classifiers – http://docs.opencv.org/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html
cats database – http://137.189.35.203/WebUI/CatDatabase/catData.html
http://docs.opencv.org/doc/user_guide/ug_traincascade.html
http://docs.opencv.org/modules/contrib/doc/facerec/facerec_tutorial.html
viola jones – http://en.wikipedia.org/wiki/Viola%E2%80%93Jones_object_detection_framework
Local binary patterns (LBP) – http://en.wikipedia.org/wiki/Local_binary_patterns