Criterion A: Planning

Definition of the problem and justification:
The problem I'm attempting to solve is the ability to train a software program to find specific objects in images. The product would be a software program or module that could be used for high speed, high quality object identification and tracking from video. This would allow it to be used in surveillance systems for detecting people, cars, and all sorts of things that surveillance cameras see every day.

Today this kind of problem can be solved more accurately with convolutional neural networks, but training such networks typically requires many thousands of sample images. My product will use keyword based object detection, which is a system that allows for easier training with less data and quick detection of generic objects like cars and faces. This makes it a better solution for surveillance applications, where you typically would only have one picture of the object that you need to find within millions of video images.

The idea for this project, finding pre-trained objects in images, is not a new idea nor is the technique I use. However, the implementation is fully my own, and it's unique in that it makes heavy use of existing code libraries to solve a very challenging problem. To implement this project I have selected to use the OpenCV library because OpenCV is a very robust and useful open source library for image processing and analysis. I use C++ as my programming language because I have the most experience using C++ with the OpenCV libraries.

Words 248

The proposed product:
The scenario for this product is finding objects in images using pre-generated training keywords. The program works in two modules, a Trainer and a Detector.

Trainer: This module will take an input of multiple similar images of the object in the same scenario. The program starts with the first image and splits it into boxes of features (using a gradient image to find these boxes). It then repeats this step for all of the other images giving a lot of features of the images. Then it looks through each of the images to find if any of the features are similar and then adds those to a list in a stack (although I'm just going to use a list). When computing similarity, the scale of the "Keyword" is varied to compensate for different image sizes. When the lists are made, each image is combined with each other image at the point where the best correlation is found relative to the first image. This will make a "blurred" image ready for use as a "keyword." The range for the vector is given by the range at which each "keyword" lies within the image.

Detector: In an earlier project I used edges to find circles and lines in an image. Instead of using edges this program will use common image samples (the "keywords") to identify where and what the object is. Each "keyword" will be like a blurry key portion of the full image as to aid the correlation process. The "keywords" are stored in a database along with what it represents and a vector that points to the center of the object. Using this technique, the program finds the equivalent images (all that pass a correlation threshold) of what the "keyword" represents. This is done for every "keyword" and with variations like rotation and scale. When each "keyword" is correlated the vector is read, and the possible object location is given from the image's location plus the vector. It is important to note that each vector in the list is really a range, so that the program can account for errors in scaling and other discrete variations.

Additionally, to allow for the program to be adapted to different platforms it will be made into a console application that can be called from other programs.

Words 383

Software tools used:
The OpenCV libraries offer a matrix/image codex that is very good at storing image data and doing mathematical operations with them. OpenCV also has a lot of built-in methods such as convolutional template matching and computation of Sobel images (for finding object edges). This, along with other library features, is why OpenCV is a great tool for implementing this product. The reason C++ was used is because of how versatile and useful the language is, also because I know how it works with OpenCV. I'm using MS Visual Studio because it supports C++ 2014 and because OpenCV was developed to work best with Visual Studio. I used Visual Studio 2017, which offers a community edition for free and works well with OpenCV.

Words 123

Success criteria:

- Have a Program capable of creating image “keywords”
- These keywords must have vectors pointing to the center of the object and the keyword itself
- Having the ability to store and retrieve these keywords
- Have a program that can use the keywords to find the object
- Indicate where the object is
- Ability to find faces in images, when trained
- Ability to find inanimate objects such as a car, when trained

Words 69

Total words 848