Airplane Detection

Robert Flack
Computer Science,
Brock University,
Ontario,
Canada
rf04nc@brocku.ca

October 26, 2008

Contents

List of Figures

List of Tables

1 Introduction

The goal of this assignment is to evolve a computer program that is capable of learning an image analysis algorithm or filter that is capable of recognizing pixels which are likely to be airplanes as opposed to those which are not likely to be airplanes.

The genetic programming package used to complete this test is the *bstlilgp*[1] package which was originally developed by Dr. Bill Punch and Douglas Zongker. It was then patched to support strong typing by Sean Luke, and finally further patched by Adam Hewgill and Cale Fairchild at Brock University.

2 Experiment Setup

2.1 Airplane Imagery Data

The data for the experiment comes from satellite imagery provided by Dr. Brian Ross containing hundreds of parked aircraft. The image is 4000x4000 pixels and has not been classified. Airplanes are roughly between 10 pixels to 30 pixels wide. The airplanes are various colours, parked in various orientations, and some are even covered by cloud cover. The goal will be to provide a good training set by which the GP system can derive a good rule of thumb for classifying airplanes.

Provided to the GP will be several image filters to aid in the classification. These image filters are useful for things such as edge detection and noise elimination. The filtered values will be provided for every pixel and the GP can use the filtered values from any nearby pixel to the one it is evaluating. I considered using other special filters such as moment, but it was infeasible to do so. This is because moment must be especially calculated for each directionality you wish to test, but as all of the filters are precomputed for the sake of speed I would have had to precompute a multitude of moment values. However, using non-directional filters means that my evolved GP results are not biased to any particular rotation of airplane.

intensity The raw intensity of the current pixel. This is simply the average of the red, green and blue channels.

min The minimum intensity of a certain range of pixels. This range is either 3x3, 5x5 or 9x9.

max The maximum intensity of a certain range of pixels. This range is either 3x3, 5x5 or 9x9.

avg The average of a range of pixels. The available ranges are 3x3, 5x5 and 9x9.

diff The summed difference of the pixels compared to the average pixel colour in a region (either 3x3, 5x5, or 9x9).

sqdiff The square of differences of pixel intensities compared to the average intensity for a region (3x3, 5x5 or 9x9).

In order to train the GP to find airplanes, a truth mask will be created for training images. This truth mask will consist of green pixels that are trained as positives, red pixels which are trained as negatives, and black pixels which are considered unidentified. Allowing a unidentified colour in the truth mask means that pixels which are questionable as to whether they should be considered to be plane or not do not need to be trained on. Not training on these cases allows us to ignore parts of the image that we know are going to be difficult, and we don't really care to worry about the added complexity of trying to come up with a rule for them too as even we might have difficulty classifying them. Examples of these might be cases where the edge of the plane is questionable (whether it extends out another pixel or not) as in Figure 1(a), or cloud covers part of a plane and the plane is partly or almost completely invisible under the cloud as in Figure 1(b). What it really comes down to is not forcing the genetic program to worry about pixels that we're not sure of ourselves.

Of course if the genetic program simply used all of the positive and negative examples in the training mask it would very quickly determine that saying that all pixels are not planes would give it a very good fitness (as most of the image is not planes!). This is not desireable, hence we will force it to have an equal sized set of positive and negative examples. However, to ensure that the training does

(a) Example of hazy plane edges – how certain can even a human be of the actual truth for every pixel.

(b) Example of cloud cover – some planes are partially or almost completely covered by clouds, why should we force a particular arbitrary boundary?

Figure 1: Examples of images where the classification for some pixels is arbitrary.

not become biased towards the sample set of pixels, the GP will periodically replace a percentage of the training set with examples that were previously falsely classified. This forces it to focus on the worst cases from before.

2.2 GP Parameters

The parameters for the GP runs are listed in Table 1. These seemed to work well in general, although the function set and the maximum offset need some tweaking to improve the performance.

2.3 GP Language

The GP language primitives were chosen to best represent the problem of classifying a given pixel. The required return type of the root is the boolean classification of plane or not plane. There are no boolean terminals which forces the GP to choose a decision at the root-level which will ultimately decide the classification.

Logic Operators

if: If the first given value (boolean) is true, then the second value is returned, otherwise the third is returned.

and: Returns true only if both of its given boolean values are true.

(a) Base GP Parameters used in Run 1

Parameter	Value
Random seed	1 for run #1, 2 for run #2, etc.
Population size	1000
Number of generations	200
Fitness cases	2000
Retraining Interval	20 generations
Retraining percentage	20%
Training weight	1.0
Max Offset	20
Test cases	>1,000,000
Initial tree method	half_and_half
Initial tree depth	2 – 6
Max tree depth	17
Crossover	95%
Mutation	5%
Selection	tournament (size 5)
Training images	sat-col1.jpg, sat-col2.jpg, sat-col3.jpg
Test-only images	satsmall.png, area51.png, dubai.jpg

(b) Changed parameters for Run 2

Parameter	Value
Max Offset	0

(c) Changed parameters for Run 3

Parameter	Value
Max Offset	0
Training images	satsmall.png
Test-only images	N/A

(d) Changed parameters for Run 4

Parameter	Value
Max Offset	0
Removed Grammar	DIFFnXn, SQDIFFnXn

(e) Changed parameters for Run 5

Parameter	Value
Crossover	85%
Selection	tournament (size 3)

(f) Changed parameters for baseline run

Parameter	Value
Selection	random

Table 1: GP Parameters used for runs

or: Returns true if either one of its given boolean values are true.

>: Returns true if the first given value is greater than the second.

<: Returns true if the first given value is less than the second.

Mathematical Operators

***:** Multiplies the two given numbers.

/: Divides the first number by the second, returning 0 if the second number is 0.

+: Adds the two given numbers.

-: Subtracts the second number from the first.

exp: Returns e raised to the power of the number.

rlog: Returns the natural logarithm of the absolute value of the number.

Image Functions

The area image functions operate on nXn areas of 3x3, 5x5, and 9x9 around the target pixel.

(INTENSITY,x,y): Returns the intensity of the pixel at offset (x,y) from the current pixel.

(MAXnXn,x,y): Returns the maximum pixel intensity of a nxn range around the pixel at offset (x,y) from the current pixel.

(MINnXn,x,y): Returns the minimum pixel intensity of a nxn range around the pixel at offset (x,y) from the current pixel.

(AVGnXn,x,y): Returns the average pixel intensity of a nxn range around the pixel at offset (x,y) from the current pixel.

(DIFFnXn,x,y): Returns the sum of differences of the pixels in the nxn range around the pixel at offset(x,y) from the average of those pixels.

(SQDIFFnXn,x,y): Returns the squared sum of differences of the pixels in the nxn range around the pixel at offset(x,y) from the average of those pixels.

Terminal Values

R: Ephemeral random floating point number initialized in the range $[-1.0 - -1.0]$

intR: Ephemeral random integer number in the range $[-x - -x]$ where x is read in from the variable "app.max_offset" from the input file.

Image	Used for	# positive	# negative	# unknown
satcol-1.jpg	Training	1560	611817	386623
satcol-2.jpg	Training	295	584894	414811
satcol-3.jpg	Training	134	531495	468371
satsmall.png	Testing	1150	18318	0
area51.png	Testing	0	0	1148395
dubai.jpg	Testing	0	0	548000

Table 2: Summary of training and testing images

2.4 Training and Testing Sets

The number of fitness cases is the number of pixels selected for training. These pixels are selected from the classified examples in the masks provided for learning. The pixels are selected on a first-come first-serve basis but no more than half of the fitness cases can be negative or positive. This way the average fitness of a random solution or the all negative/positive solution is in theory 50%.

The testing set consists of all classified pixels and testing fitnesses are calculated on a per image basis. If the testing image is also a learning image than the current testing pixel may replace a current learning pixel if it is falsely classified by the current best algorithm. This forces the GP to train on the most difficult to classify examples as they will be added to the learning set on a regular basis. A parameter in the input file "app.test_train_interval" sets how often the program tests (and adds to the training set) in terms of number of generations.

The training file and its painted truth mask are shown in Figure 2.4. As you can see most of the training samples are negative as more pixels are not planes than planes. This can be seen in Table **??**, which summarizes the set of images provided to the GP.

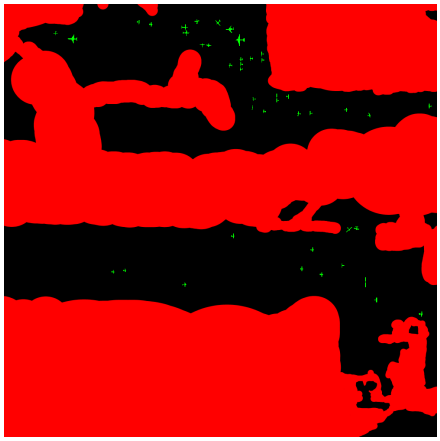
2.5 Fitness calculation

The original raw fitness f_r is the number of pixels incorrectly classified. The standardized fitness f_s is the percentage of incorrectly classified pixels out of total pixels classified. Finally, the adjusted fitness f_a is $1 - f_s$. While this standardized fitness worked well, and did produce some impressive classifications, I felt that more significance was needed for particularly hard to classify cases.

To resolve the issue of difficult cases, I introduced the concept of a weighted fitness. In each generation, I keep track of how many times each training case is misclassified. In the following generation, it uses the number of misclassifications in the previous generation as a weight on how much an incorrect classification of this case will cost it in the fitness function. The resulting fitness functions are:

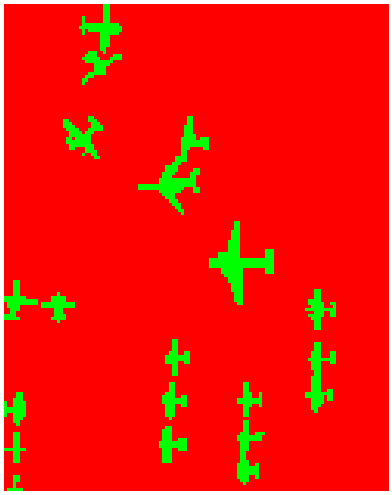
$$f_r = \sum_{x \in E} 1 + k \cdot w_x$$

(a) Sample of large satellite training image.



(b) Truth mask for large satellite image.

(c) Standard test image.



(d) Truth mask for testing image.

Figure 2: Example of training images provided to airplane GP.

$$w_x = 1 + E'x$$

Where E is the set of examples that were misclassified and w_x is the the number of times example x was misclassified in the last generation. k is a multiplier constant defined in the input file to control weighting of difficult examples.

3 Results

The first run (Run 1) was run exactly as specified in the base parameters in Table 1. The remaining tests were changed slightly to see how various parameters affect the ability of the GP to solve the problem. Run 2 used the same parameters but removed the max offset, setting it to 0. Run 3 tested the effect of training the GP on the standard test image and its testing results should not be considered "untrained". Run 4 tests the effects of removing most of the image operations, restricting the GP to using averages, intensities, max and min.

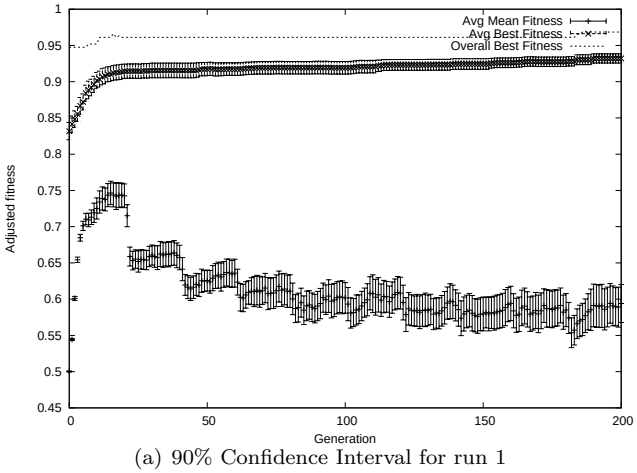
Figures 3 and 3 summarize the resulting performances of the various runs. Each run was performed 30 times such that a 90% confidence interval could be given. Unfortunately, the results are not exactly what we would like to see with repect to outperforming a random search. Namely, the best results almost immediately approach the best percentage, and the average results do not even steadily increase. However, the genetic algorithm selection pressure is keeping the fitness from dropping altogether as in the random selection baseline test in Figure 4(c).

The testing percentages shown in Table 3 and Table 3 show the testing results summarized across the various images. A very interesting observation is that in the tests where the GP was trained on other images it actually outperformed being trained directly on the test image "satsmall.png". This doesn't intuitively make much sense, although it could perhaps be the case that some of the fuzzy training examples (blurry plane edges) confused the GP algorithm more than it did it good to train on. Another thing to note about the results is how the maximum offset affected the results. While the best of run 1 was comparable to the rest of the runs, on average it did much worse. Compare the confidence intervals presented in Table 3 and it is clear that the language and parameters used in the other runs outperformed run 1 by far.

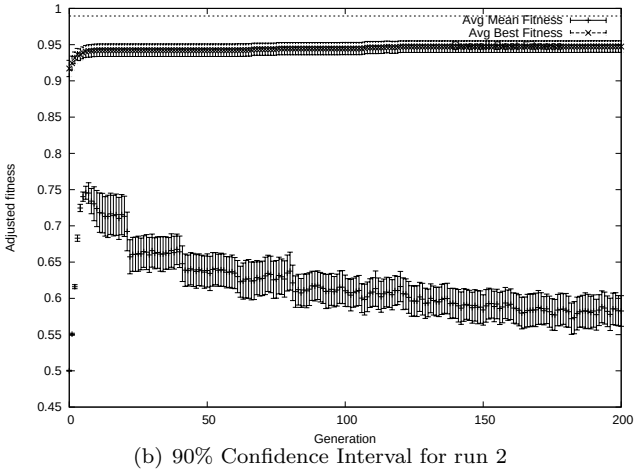
Run configuration 2 was probably the optimal configuration for this problem, even though run 5 on average did better recognizing positives. Run 2 did better in general recognizing most positives and negatives. You can see the negative effect of removing the filters as was done in run 5, it reduced the performance significantly producing more false positives.

3.1 Best Results

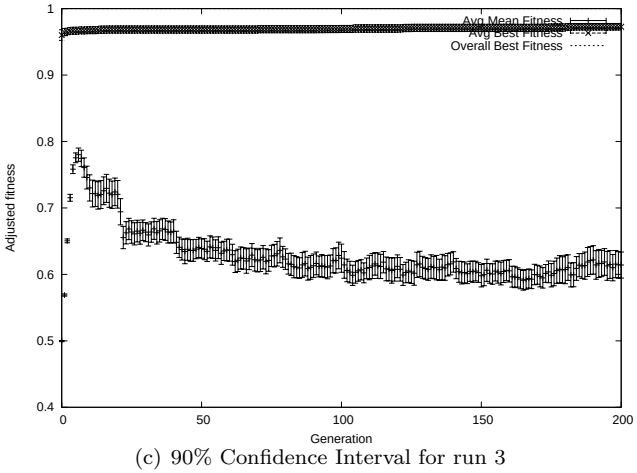
The best results are tabularized in Table 3, however we can see the actual classifications in Figure 3.1. Due to the size of these images, I chose just to include the class test images, however, all of the classifications can be seen in



(a) 90% Confidence Interval for run 1

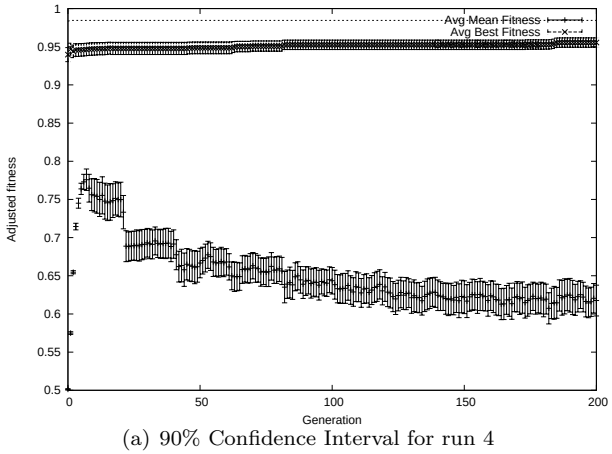


(b) 90% Confidence Interval for run 2

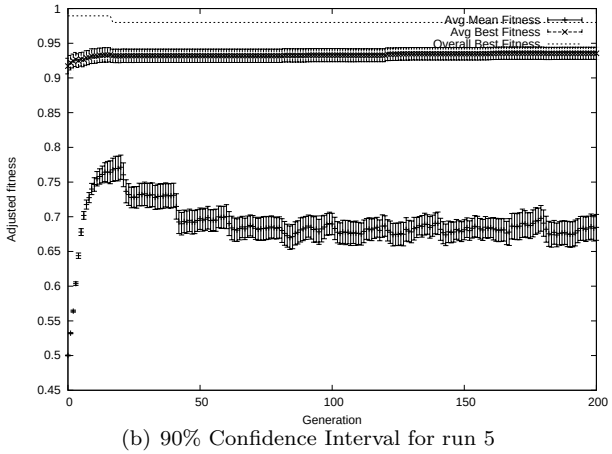


(c) 90% Confidence Interval for run 3

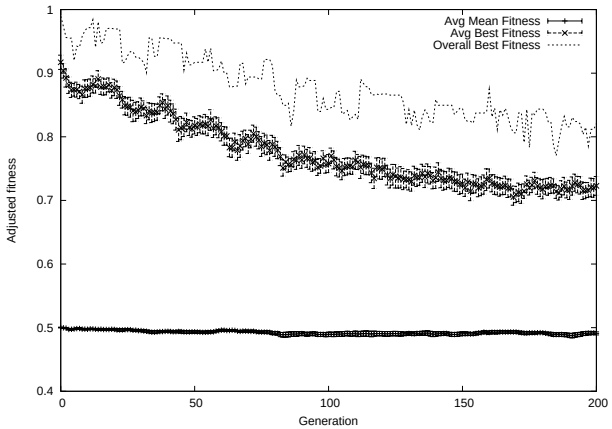
Figure 3: Graphs showing 90% confidence intervals of the fitness for runs 1 – 3.



(a) 90% Confidence Interval for run 4



(b) 90% Confidence Interval for run 5



(c) 90% Confidence Interval for baseline run

Figure 4: Graphs showing 90% confidence intervals of the fitness for runs 4 – 5 and the baseline run.

Type	Image	Run 1	Run 2	Run 3	Run 4	Run 5	Baseline
True Pos	satcol-1.jpg	100%	100%	n/a	100%	100%	100%
	satcol-2.jpg	100%	100%	n/a	100%	100%	100%
	satcol-3.jpg	100%	100%	n/a	100%	100%	100%
	satsmall.png	100%	100%	100%	100%	100%	100%
True Neg	satcol-1.jpg	93%	99%	n/a	97%	98%	98%
	satcol-2.jpg	92%	98%	n/a	97%	96%	97%
	satcol-3.jpg	89%	98%	n/a	95%	95%	96%
	satsmall.png	90%	99%	93%	98%	98%	98%
False Pos	satcol-1.jpg	7%	1%	n/a	3%	2%	2%
	satcol-2.jpg	8%	2%	n/a	3%	4%	3%
	satcol-3.jpg	11%	2%	n/a	5%	5%	4%
	satsmall.png	10%	1%	7%	2%	2%	2%
False Neg	satcol-1.jpg	0%	0%	n/a	0%	0%	0%
	satcol-2.jpg	0%	0%	n/a	0%	0%	0%
	satcol-3.jpg	0%	0%	n/a	0%	0%	0%
	satsmall.png	0%	0%	0%	0%	0%	0%

Table 3: Best testing results across all 30 tests.

the *runs* folder, as well all of the best classifications saved as best-*i*-*image*.jpg. If you're interested, a sample with a lot of unclassified areas is shown on one of the large satellite photos in Figure 3.1.

3.2 Best Airplane Filters

The actual filters that were evolved are listed here. When allowed to examine adjacent filters, the best of run 1 was an extremely bloated and not very effective 126 node program. To contract, the best in run 5 on the testing image was this concise 11 node program:

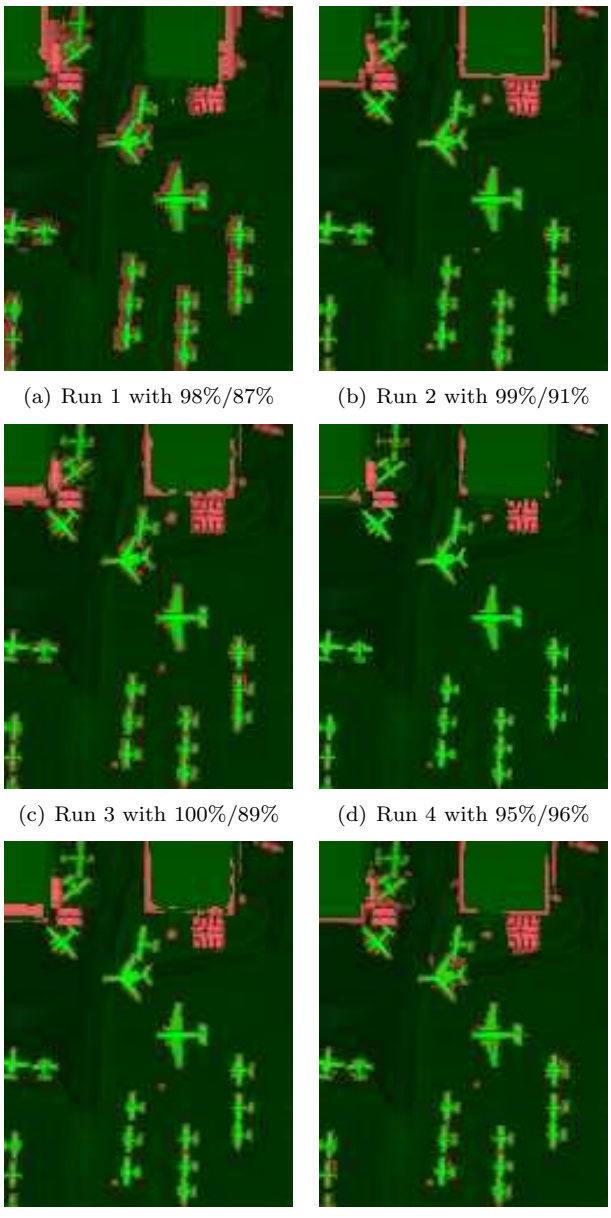
```
(> (* (MAX5X5 0 0)
      (INTENSITY 0 0))
   (AVG9X9 0 0))
```

Similarly, the best from run 5 on the first image was a rather 39 node program:

```
(and (and (> (INTENSITY 0 0)
             (+ 0.13110 0.36319))
          (< (MIN3X3 0 0)
             (+ (+ (DIFF9X9 0 0)
                   (AVG3X3 0 0))
                (- (* -0.40687 0.13729)
                   (DIFF3X3 0 0))))))
```

Type	Image	Run 1	Run 2	Run 3
True Pos	satcol-1.jpg	82.5% +/- 7.9%	93.4% +/- 2.1%	n/a
	satcol-2.jpg	85.6% +/- 7.2%	96.7% +/- 1.6%	n/a
	satcol-3.jpg	81.6% +/- 6.9%	90.6% +/- 3.8%	n/a
	satsmall.png	78.4% +/- 6.9%	85.2% +/- 5.0%	77.7% +/- 9.2%
True Neg	satcol-1.jpg	66.0% +/- 7.6%	91.0% +/- 3.1%	n/a
	satcol-2.jpg	67.7% +/- 6.9%	90.8% +/- 2.3%	n/a
	satcol-3.jpg	62.1% +/- 6.5%	87.9% +/- 3.2%	n/a
	satsmall.png	59.2% +/- 7.2%	84.0% +/- 4.0%	73.6% +/- 7.9%
False Pos	satcol-1.jpg	33.9% +/- 7.7%	9.0% +/- 3.1%	n/a
	satcol-2.jpg	32.3% +/- 6.9%	9.2% +/- 2.3%	n/a
	satcol-3.jpg	37.9% +/- 6.6%	12.1% +/- 3.2%	n/a
	satsmall.png	40.8% +/- 7.2%	16.0% +/- 3.4%	26.4% +/- 7.9%
False Neg	satcol-1.jpg	17.5% +/- 7.9%	6.6% +/- 2.1%	n/a
	satcol-2.jpg	14.3% +/- 7.2%	3.3% +/- 1.6%	n/a
	satcol-3.jpg	18.4% +/- 6.9%	9.4% +/- 3.8%	n/a
	satsmall.png	21.6% +/- 6.9%	14.8% +/- 5.0%	22.3% +/- 9.2%
Type	Image	Run 4	Run 5	Baseline
True Pos	satcol-1.jpg	93.8% +/- 4.6%	95.0% +/- 1.5%	91.0% +/- 1.9%
	satcol-2.jpg	96.0% +/- 4.8%	98.2% +/- 0.9%	96.3% +/- 1.5%
	satcol-3.jpg	96.0% +/- 5.2%	93.7% +/- 2.7%	90.8% +/- 3.4%
	satsmall.png	91.2% +/- 3.9%	88.8% +/- 3.7%	83.9% +/- 3.8%
True Neg	satcol-1.jpg	87.6% +/- 6.3%	87.9% +/- 4.1%	91.5% +/- 1.3%
	satcol-2.jpg	86.0% +/- 6.5%	88.6% +/- 2.9%	90.4% +/- 1.5%
	satcol-3.jpg	83.3% +/- 6.2%	84.7% +/- 3.9%	88.2% +/- 1.6%
	satsmall.png	85.6% +/- 6.7%	85.0% +/- 3.8%	87.9% +/- 3.2%
False Pos	satcol-1.jpg	12.4% +/- 6.3%	12.1% +/- 4.1%	8.5% +/- 1.3%
	satcol-2.jpg	14.0% +/- 6.5%	11.4% +/- 2.9%	9.6% +/- 1.5%
	satcol-3.jpg	16.7% +/- 6.2%	15.3% +/- 3.9%	11.8% +/- 1.6%
	satsmall.png	14.4% +/- 6.7%	15.0% +/- 3.8%	12.1% +/- 3.2%
False Neg	satcol-1.jpg	6.2% +/- 4.6%	5.0% +/- 1.5%	9.1% +/- 1.9%
	satcol-2.jpg	3.9% +/- 4.8%	1.8% +/- 0.9%	3.7% +/- 1.5%
	satcol-3.jpg	4.0% +/- 5.2%	6.3% +/- 2.7%	9.2% +/- 3.4%
	satsmall.png	8.8% +/- 3.9%	11.2% +/- 3.7%	16.1% +/- 3.8%

Table 4: 90% confidence interval for best GP testing results.



(a) Run 1 with 98%/87%

(b) Run 2 with 99%/91%

(c) Run 3 with 100%/89%

(d) Run 4 with 95%/96%

(e) Run 5 with 98%/92%

(f) Run base with 98%/91%

Figure 5: Best results on standard training image. Run 3 was specifically trained on this image. Red areas were incorrectly classified and green areas were correctly classified. Percentages are given as percent true positive / percent true negative. These specific instances were selected as best because they had the greatest total correct classification scores – even though for positive or negative specifically there were better examples.

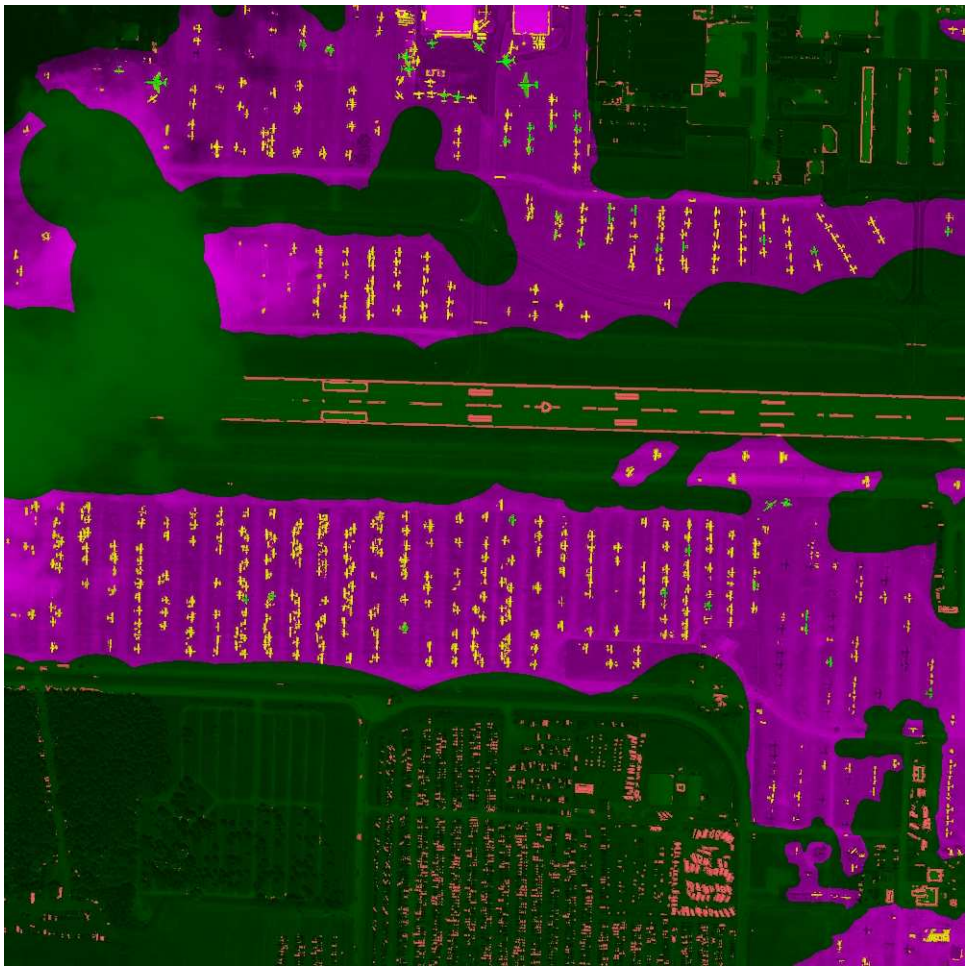


Figure 6: Best solution for big satellite image from run 5 with 99% true positive and 95% true negative. Green regions represent areas in the training data that were trained and correctly identified. Red regions were trained and incorrectly identified. Purple regions represent untrained areas that the GP thinks are negative or not planes. Yellow regions are areas that were untrained and the GP thinks are positive or planes.

```
(< (/ (AVG9X9 0 0)
      (MAX3X3 0 0))
   (MAX5X5 0 0)))
```

4 Conclusions

The GP was fairly successful at separating airplanes from most of the background noise. I think that no matter how much training or what terminal set is provided there will always be extra artifacts in the image and a few planes that will not be identified as even I had a hard time classifying some regions. I think that this is an excellent filter for bringing areas that could be planes to attention, followed by which another filter should be run which can identify whether or not those shapes look like planes. For example, single positive pixels and non-plane shaped regions could be eliminated and most of the false positives would be gone.

References

[1] Adam Hewgill, Cale Fairchild, Sean Luke, Bill Punch, and Douglas Zongker. Brock strongly typed lilgp kernel.

A Input File Parameters

There are a variety of input file parameters that control the training images and execution of the genetic program. The following is a complete list of all of the non-standard parameters (in that they are not part of lilgp).

app.num_images Sets the number of training / testing images to be defined and loaded.

app.image_*i*.file Sets the file containing the actual image for training / testing example *i*

app.image_*i*.truthfile Sets the truth mask for image *i*. This truth mask should be a 8-bit or less palette image containing solid green in areas which are to be trained or tested as being a plane, and solid red in areas which are to be trained or tested as not being a plane.

app.image_*i*.mode Set the mode for the image *i*. This mode is either test or train. If the mode is train, classified examples can be taken from it for learning, otherwise they are only used for testing.

app.fitness_cases Sets the number of fitness cases to train on at any given time. Exactly half of these will be positive examples and half will be negative examples.

app.train_overwrite_percent Sets the percentage of training examples that can be overwritten by new difficult examples from the testing images during a test run.

app.training_weight Sets the multiplier of the extra weight added to training cases that were difficult to classify in the last generation.

app.max_offset Sets the maximum offset from the current pixel used when examining image filters.