

Audion Cars Counter — a guide for an AI agent

Contents

- [The words](#)
- [Rules that always hold](#)
- [The folder](#)
- [Making the folder work](#)
- [Where the data lie](#)
- [Commands](#)
- [The work, in order](#)
- [What was measured](#)
- [The window](#)
- [Files](#)
- [When something goes wrong](#)

This file is for an AI agent that works in the folder of the program together with its user: Claude Code, Codex or any other agent that can run commands and look at pictures. The user starts it in the program's folder and says: "Read `Docs\AI_AGENT_GUIDE_EN.md`". The window of the program counts and reports through the engine commands described here, and writes the same mark files they write. Teaching a net and judging it have no button at all: they are commands, run by a person or by you.

What the program does: it counts passenger cars on large satellite pictures that are tied to the ground. On such a picture (about 0.3 m to a pixel) a car is a soft blob of about 14 x 7 pixels. A small net answers, for every pixel, how likely the centre of a car is there; a car is a peak of that map above a threshold. The net knows only what was marked for it by eye, so the work is a loop: mark, teach, judge, look at what it got wrong, mark that.

The public release of the program carries a net, `open-03`, taught on free aerial photographs of three open state services and on nothing else (`system_core\nets\README.md` says on what and how it was examined). A new set of pictures is counted with it at once - and that count is not to be believed until the net is examined on the user's own pictures: they are another source. A copy whose net was taken out shows pictures and takes marks and cannot count until a net is there ("No net yet: the first net"). A private copy may carry another net; a new set of pictures is then counted with it at once, but pictures of another source, or ground of a kind that net never saw, must be examined first and are usually taught for. `engine.cmd where` tells which copy this is. Both kinds of work are what this guide describes.

People use the program through its window, `Start.exe`, and the user guide (`Docs\USER_GUIDE_EN.md`). Paths below are relative to the program's folder; `<work>` is the work folder of the set of pictures (see "Where the data lie").

The words

Word	What it means here
picture	one <code>.jpg</code> of the set with its MapInfo <code>.tab</code> beside it; named by the end of its file name, column-row: <code>2-3</code>
set	the pictures of one folder, counted together
patch	a square of 240 x 240 pixels of a picture (about 75 x 75 m), marked completely
mark	a point set by eye on a patch: a car, a truck, "cannot tell"
ground	whatever has no point on a marked patch: the net is taught that no car is there
net	a file <code><name>.pt</code> the program counts with; named by its bare name: <code>cars-05</code>
answer	a place where the net sees the centre of a car, with how sure it is, from 0 to 1
threshold	the sureness from which an answer is counted; <code>least</code> in commands and files
run	the count of a set by one net: a folder <code>count_<net></code> in <code><work>\runs</code>
check picture	a picture marked only to judge nets; it is never taught on
a mark of the program	what the COUNT section and the report call an answer at the threshold

Word	What it means here
place	a patch by its picture and its top left corner: <code>2-3:8200,300</code>

Rules that always hold

- **The pictures are the user's.** They are only read. Never copy them, or cuts of them, into the program's folder, and never send them anywhere.
- **Ask before you look.** Whatever you look at - a reduced picture to choose patches on, an enlarged patch to mark, a sheet of samples, a page of a report - goes to the service that runs you. Satellite pictures usually come with their supplier's terms of use. Before the first picture, ask the user whether looking is allowed for these pictures. If it is not, the user marks in the window (section VIEW) and looks at the sheets; you run what needs no eye: the teaching, the tables, the count, the report.
- **Marks are never erased and never overwritten in bulk.** They are hours of looking and the one thing no command makes again. Do not delete, move or rewrite the files of `<work>\labels` with scripts of your own. `patch seed` refuses a patch that is marked; `patch set` writes one patch anew - use it on purpose, one patch at a time; `patch fix` changes single points.
- **A net is never written over.** A new net takes a new name; `net train` and `net join` refuse a name that is there in the work folder. If the program holds a net - one that came with it or was put into `system_core\nets`; `engine.cmd where` lists every net by name - do not give a net of your own its name: a net of that name in the work folder is taken before the one of the program.
- **Check pictures are never taught on.** Agree with the user which pictures they are, write the list down where the user will find it, and give it as `--hold-tile` at every teaching. A net is judged only on patches it was not taught on.
- **Run the engine only through `engine.cmd`** (it runs `runtime\python.exe system_core\main.py`). Never the system Python, and install nothing into it.
- **One set, one work folder.** Do not point a work folder at other pictures, and do not edit `config\settings.json` to run one command on another set: the window and every other command read that file. Use the variables of "Where the data lie".
- **Never one bare number.** A count goes with its bounds (the marks less a quarter to the marks plus a half), and a percent with what it stands on: 7 % of 109 cars is the noise alone.
- **Say what a long run is before you start it,** and how long it takes: a teaching, a count of a whole set.

The folder

Path	What is in it
<code>Start.exe</code> , <code>App\</code>	the window
<code>engine.cmd</code>	the engine from the command line: <code>engine.cmd <tool> ...</code>
<code>system_core\</code>	the engine: <code>main.py</code> and a file a tool; each tool says what it takes in the first lines of its file
<code>system_core\nets\</code>	the place of a net that came with the program or was put into it, a file <code><name>.pt</code> ; in the public release: <code>open-03</code> , taught on free aerial photographs
<code>config\counting_default.json</code>	names the net of <code>system_core\nets</code> to count with and its threshold: <code>{"net": "<name>", "least": 0.20}</code>
<code>runtime\</code>	the program's own Python 3.12 with PyTorch, NumPy, OpenCV, Pillow and openpyxl (installed by SETUP)
<code>install\</code> , <code>builder_main.cmd</code>	the installers and their menu
<code>config\settings.json</code>	what the window remembers; the engine reads <code>"Pictures"</code> , <code>"Work"</code> and <code>"Data"</code> from it
<code>config\version.json</code>	the version
<code>logs\</code>	the journal of the window, a file a day
<code>Docs\</code>	this guide and the user guide

The pictures, the marks, the nets taught by the user, the counts and the reports are not in the program's folder.

Making the folder work

The program is published as a skeleton: the window, the code and the net `open-03` - the engine is not installed yet. `Start.exe`, section **SETUP**, **INSTALL WHAT IS NEEDED** puts Python 3.12 into `runtime\`, PyTorch 2.8.0 (with CUDA 12.8 on an NVIDIA card whose driver is 570 or newer, for the processor otherwise) and the libraries: 0.7 to 3.4 GB of downloads, about 7 GB installed with CUDA. The button asks first, saying what comes for this machine. The first step of the installation makes sure

Windows has the Microsoft Visual C++ Redistributable PyTorch needs and installs it with winget when it has not (`install\Ensure-VCRedist.ps1`; the one thing that goes into Windows, with the administrator's consent). From the command line: `builder_main.cmd install_all`. Then:

```
builder_main.cmd verify      a line a thing checked; must end with "[OK] Everything is
in place." or "[OK] In place, with N warning(s)."
engine.cmd doctor            one JSON line: python, torch, device (cuda or cpu), card,
the libraries, net, net_file, problems
engine.cmd where             the folders, how many pictures, marked patches and nets,
the net the set is counted with, which torch and which card
```

`doctor` needs no pictures. `"device": "cpu"` on a machine with an NVIDIA card:
`builder_main.cmd check_machine` says why (usually the driver), `builder_main.cmd torch_cuda` installs PyTorch with CUDA. `builder_main.cmd run_tests` runs the engine's own tests on a drawn picture; no picture of the user is touched.

A copy without a net is not a broken installation. `doctor` then answers `"net": ""` and `"net_file": ""`; its `problems` list libraries only, and its exit code is 1 only when a library is missing. `verify` gives a `[WARN]` line for it, not a `[FAIL]`. `where` prints `counts with: no net yet (none taught in the work folder, none came with the program)`. In the window the SETUP section shows the row "No taught net" and the summary "The engine is installed. No taught net: nothing to count with yet", and the COUNT section says "No taught net in the work folder or in the program: nothing to count with." Everything else works: viewing, marking, teaching. What to do then is "No net yet: the first net" below.

From PowerShell the command is `.\engine.cmd where`.

Where the data lie

Every kind of data has a field in the SETTINGS section; the engine reads the same keys of `config\settings.json`.

Field	Key	Without it	What it is
PICTURES	<code>"Pictures"</code> : a full path, or the name of a folder inside <code>Pictures</code> of the data folder	a folder with pictures inside <code>Pictures</code> of the data folder: the one <code>Work</code> was made on; in a new data folder the first such folder by name, or <code>Pictures</code> itself when the pictures lie straight in it	the set that is counted; only read

Field	Key	Without it	What it is
WORK FOLDER	"Work": a full path	Work of the data folder for the default set, Work-<name of the pictures folder> for any other	the marks, the counts, the tiles of the viewer, the nets taught there
REPORTS	"ReportFolder" (the window only)	asked at the first export	where the window writes reports and layers; the report command takes -- out

The data folder is looked for beside the program's folder: `Audion Counters Data\Cars Counter`, else `Cars Counter` (the key `"Data"` or the variable `CARS_COUNTER_DATA` names another). A new user puts the pictures into a folder of any name inside `Pictures` of it (`Pictures\MyTown`), and the work folder `Work` appears by itself. Do not assume any of this: `engine.cmd where` prints the folders it took.

Where a net is looked for, in this order: `<work>\models` - a net taught for this set, where `models\counting.json` names the accepted one - then the program itself: `config\nets` (a file `<name>.credential.pt`: a net kept the way a key is kept, so that tools which clear secrets out of `config` leave it out of a public archive) and `system_core\nets` (a file `<name>.pt`), where `config\counting_default.json` names the one to count with. So a net taught in a work folder counts the set of that folder, and a net in the program counts every set whose work folder names none.

A picture is a file `<anything>_<column>-<row>.jpg` with a `.tab` of the same name beside it. The size of the picture, the size of its pixel, the area and every place on the ground come from the `.tab`, whose units the program takes for metres. A picture without a `.tab` is counted, but has no area and no place in the layers.

A work folder belongs to one set of pictures. The marks, the tiles and the runs in it are found by the end of a picture's name alone (`2-3`), and another set - another town, another shooting of the same ground - has pictures that end the same. The folder remembers the pictures it was made on (`<work>\set.json`), and every tool but `where` and `info` refuses it for other ones. Counts of two sets are never added up.

One command on another set, without touching the settings:

```
set CARS_COUNTER_PICTURES=<a full path, or the name of a folder of Pictures>  
set CARS_COUNTER_WORK=<a full path>  
engine.cmd where
```

(in PowerShell: `$env:CARS_COUNTER_PICTURES = '...'`). Without `CARS_COUNTER_WORK` the work folder of that set is taken by the rule of the table. `CARS_COUNTER_NETS` names, for one command, one folder instead of the two of the program. `CARS_COUNTER_PYTHON` names another `python.exe` for `engine.cmd`; leave it unset.

Commands

Every command is `engine.cmd <tool> ...` from the program's folder. `T` is a picture (`2-3`), `NET` a net by its bare name, `RUN` a run by its bare name.

where the folders, the pictures, the marked
 patches, the nets, torch and the card
 doctor what the engine runs on, one JSON line
 layout how the pictures lie on the ground and how
 much neighbours overlap
 geo T the size of a pixel of a picture, its
 corner, its turn
 overview T [x,y,w,h] [reduce] [pieces] [grid] a picture or a region of it reduced, with a
 grid of its pixel coordinates
 patch show --tile T --at "x,y x,y ..." pictures of patches to look at; nothing is
 written into the marks
 patch seed --tile T --at "x,y ..." --by NET [--least 0.2] first points of patches that have none, by a
 net
 patch set --tile T --at x,y --points "px,py px,py,t ..." all the points of one patch anew, in the
 pixels of the patch
 patch fix --tile T --at x,y [--drop "3 7"] [--add "px,py ..."] [--move "5:px,py"]
 [--kind "4:t"] the numbered points of one patch changed
 patch snap --tile T --at "x,y ..." --by NET car marks moved onto the net's own answer
 within 8 pixels
 patch list every marked patch and the totals
 montage T NAME "x,y x,y ..." four drawn patches on one sheet
 net train --name NAME [--hold-tile "T T"] [--hold "T:x,y ..."] [--steps 2000] [--seed
 1] [--width 20] [--lessons 2|3]
 [--also PICTURES WORK ["T T"]] ... the marks of another set of pictures
 taught on as well: the folder of
 its pictures, its work folder, its
 pictures kept out; the sets stay apart
 a net taught on every marked patch but those
 held back
 net join --name NAME --of "NET NET ..." several nets written into one file that
 answers with their mean
 net check --weights NET --at "T:x,y ..." [--each] a net against the marks of the named
 patches, by threshold
 net count --weights NET|auto --tile T [T ... | all] [--out RUN] [--region x,y,w,h] [-
 -again] [--json] every answer of a net on whole pictures

stability NET	is the error the same from place to place
shaken NET LEAST	does the count stay when the picture only
looks different	
overlay NET "T:x,y ..." [least]	marks and answers on one patch
answers RUN T [least] [sample]	where the answers of a counted picture lie,
and a sheet of samples	
report RUN --out FOLDER [--tile T ...] [--least 0.20] [--lang ru en] [--formats xlsx docx mif]	
[--cell 250] [--samples 18] [--json]	
	the count as files people read, and as
layers for MapInfo	
info, thumbs, pyramid T	what the window asks of the engine; they
print JSON lines	

`--weights auto` is the net the set is counted with: the one `<work>\models\counting.json` names, else the one `config\counting_default.json` names in the program; when there is neither, the command stops with `no net to count with: ...`. A net by its bare name is looked for in `<work>\models` first, then in `config\nets` and `system_core\nets`.

Some tools ask a net for its answers: `patch seed`, `patch snap`, `net check`, `net count`, `stability`, `shaken`, `overlay` - and `answers` and `report` read a run counted by one. In a copy without a net they come into play after the first teaching; everything else works from the start.

The work, in order

0. Where things are

engine.cmd where

The folders, how many pictures, marked patches and nets, the net the set is counted with, which torch and which card. A line `PROBLEM:` means the work folder was made on other pictures: name another one for this set.

The line `counts with:` says where the work begins:

- `counts with: <net> at <threshold> (<its file>)` - there is a net: go on with step 1;
- `counts with: no net yet (none taught in the work folder, none came with the program)` - there is none: go by "No net yet: the first net".

No net yet: the first net

Without a net nothing can be examined or counted, and no tool that asks a net for its answers works yet: there are no first points by a net, no `snap`, no tables. Every point the first net is taught on is set by eye - by you on drawn patches, or by the user in the window. The steps are the numbered ones below, taken in this order:

1. **Leave to look, and the sharpness.** Ask the user whether you may look at these pictures ("Rules that always hold"). Then draw a few patches from pictures at different ends of the set and look at them:

```
engine.cmd patch show --tile 2-3 --at "8200,300 9000,300"      ->
<work>\runs\look\2-3\8200_300.jpg, 9000_300.jpg
engine.cmd geo 2-3                                             the size of a pixel
```

`patch show` only draws: nothing is written into the marks. A set that is blurred in parts will not give an error that holds, whatever it is taught: say so to the user before any marking.

2. **The check pictures.** Agree with the user on two or three pictures of different kinds of ground, write the list down, and hold them back at every teaching.
3. **Patches on the map** - step 2: a regular grid on the check pictures; on other pictures a grid and the places a net usually gets wrong.
4. **Marks by eye** - step 4; step 3 is skipped. Draw each patch with `patch show`, read its cars off the grid and write them with `patch set` (a patch without cars is `--points ""`); `patch fix` changes single points, `montage` lays four drawn patches on one sheet, `patch list` gives the totals. Or the user marks the same patches in the window. The check patches and the teaching patches are marked alike. For a measure: the author's net was taught on about 180 patches holding about 530 cars, and a check tells one net from another only when it holds several hundred cars.
5. **Four nets, joined** - step 5 without `patch snap`: the five commands there as they stand, with the check pictures in `--hold-tile`.
6. **The tables** - step 6: `engine.cmd stability cars-05`, the threshold read off that table, then `engine.cmd shaken cars-05 <threshold>`. From here on there is a net, and every tool works.
7. **Whole pictures** - step 7: count a picture or two with `--weights cars-05` and look at the sheet of `answers`. What is not a car there is marked as patches without cars; then four more nets are

taught under new names (`cars-06` to `cars-09` , joined as `cars-10`) and set beside `cars-05` .
In this second round new patches may take their first points from the net (step 3).

8. **The net the set is counted with** - step 8: when the user takes the error the tables show, write `<work>\models\counting.json` . From then on `net count --weights auto` and the button COUNT THE PICTURES count with that net; ask the user to press REREAD.

1. The exam: what the net that is there does on these pictures

This step needs a net: one that came with the copy, one the user was given, or one taught before. Do this before any teaching. Teach only what the exam fails.

1. **Sharpness.** With the user's leave, draw a few patches from pictures at different ends of the set (`patch show`) and look. A set that is blurred in parts will not give an error that holds, whatever it is taught. `engine.cmd geo 2-3` prints the size of a pixel; about 0.3 m is what the author's nets were taught on.
2. **A look at whole pictures.** Count two or three pictures of different kinds of ground and lay samples of the answers on sheets (`<net>` is the name on the `counts with:` line):

```
engine.cmd net count --weights auto --tile 2-3 4-1      ->
<work>\runs\count_<net>\2-3.csv, summary.json
engine.cmd answers count_<net> 2-3 0.2 80              -> 2-3_where.jpg, 2-
3_sample.jpg in the run folder
```

On the sheet, what share is not a car, and what is it: that is the next thing to mark.

3. **Numbers.** Lay a regular grid of check patches over two or three pictures (step 2), take the first points from the net (step 3), correct them by eye (step 4), and run

```
engine.cmd stability <net>
```

For a net that came with the program every patch of yours is fresh, so each marked picture stands in the table as two halves. (The table tells "taught" by the name of the picture and the corner of the patch: a patch of yours that bears both as one the net was taught on would stand under "taught".) Read it as step 6 says. Marks that began as the net's own draft lean towards what the net sees: "missed" is then a lower bound, so look for cars without a point.

4. **The verdict.** The measure this program was built to is not the exact number but an error that holds: 5, 10 or 20 % off is workable when it is the same everywhere, because one coefficient

corrects it; an error that changes from place to place is not. Ask the user what error they can live with, and tell them the three things of step 6 with the number of cars they stand on.

2. Choose patches on the map

```
engine.cmd overview 2-3                                the whole picture, a line every 2000
px -> <work>\runs\overview\2-3_0.jpg
engine.cmd overview 2-3 8000,0,4490,3000 0.4           a region, larger
```

Choose by what the map shows, not by where cars are:

- **patches for a check** - a regular grid over the built-up part of the check pictures, every cell taken, empty ones too. Half of them will be forest and roofs; an empty patch is an answer as well. A check picture chosen for one kind of ground checks that kind: take several kinds - a car park in the open, blocks of flats, an old quarter under shadows;
- **patches for teaching** - a grid too, on other pictures, plus places of the kinds the net gets wrong: car parks in the shadow of a house, dense rows, yards, a road, and hard ground (step 7).

A check that holds a few dozen cars tells nothing finer than tens of percent: aim at several hundred cars marked on the check pictures.

3. First points, by a net

This step needs a net: `<net>` is one that came with the copy, or the one you taught last. Before the first net there are no first points: draw the patches with `patch show` instead of `patch seed`, and set every point by eye (step 4).

```
engine.cmd patch seed --tile 2-3 --at "8200,300 9000,300 9800,300" --by <net> --least
0.2
engine.cmd montage 2-3 town "8200,300 9000,300 9800,300"
```

`seed` writes the net's answers as car marks and draws each patch enlarged six times and lightened: `<work>\runs\look\2-3\8200_300.jpg`, with a grid every 20 pixels and the pixel numbers of the patch along the top and the left; `patch show` draws the same picture and writes no mark. `montage` puts four drawn patches on one sheet (`<work>\runs\sheets\town_0.jpg`), to go through sparse ones quickly. The points of a net are a draft: it misses dark cars and takes other things for cars.

4. Mark by eye

Look at the picture of the patch and write every car of it. Coordinates are those of the patch, read off the grid.

```
engine.cmd patch set --tile 2-3 --at 8200,300 --points "48,150 112,60 130,64,q
201,33,t"
engine.cmd patch fix --tile 2-3 --at 8200,300 --drop "3 7" --add "88,120 95,131,q" --
move "5:60,44" --kind "4:t"
engine.cmd patch set --tile 2-3 --at 9000,300 --points ""
engine.cmd patch list
```

set writes all the points of a patch anew; **fix** changes the numbered points the picture shows; an empty **set** says "no car here, all of it is ground". Both redraw the picture - look at it again: a point belongs on the car, and the numbers of the points are those of the picture drawn last.

Kind	What	For the net
(none) or c	a car: a car, an off-roader, a van, a pickup	a car
t	a truck, a bus, a trailer, machinery	not counted; not taught as ground
q	cannot tell: a dark shape in shadow, half under a tree, a blob that may be a car	the same
no point	everything else on the patch	ground

Rules that were paid for:

- **A patch is marked completely.** Whatever has no point is taught as ground, so a car left without a point teaches the net that such cars are ground.
- **Doubtful is **q**.** A wrong mark teaches worse than a missing one. In an old quarter under long shadows there may be more **q** than cars - that is the picture, not a failure.
- **The point goes on the middle of the car,** between the glass and the body, not on its nose. In a row of parked cars count along the row: they stand 7-9 pixels apart.
- **Dark cars in the shadow of a house are there.** The picture is lightened for this; look for the pairs of light dots of the glass in a row.
- **Not cars:** gravestones in rows, vents and hatches on a roof, garden beds and greenhouses, stacks of pallets, boats, the roofs of sheds and garages, single light tree crowns. A garage roof is 2-3 cars

wide.

- A very dense patch is looked at alone, not on a montage sheet.

The marks are files: `<work>\labels\<picture>\<x>_<y>.csv`, columns `x,y,kind` in the pixels of the picture. A file `<work>\labels\<picture>\ground.csv` (columns `x,y,w,h`, a rectangle a line, each more than 96 pixels wide and high) names rectangles with no car at all - forest, a field - from which plain ground is cut for teaching; no command writes it, it is typed.

A person marks the same patches in the window (section VIEW) by clicking. The files are the same, so marks made by a command and by a click mix freely; see "The window".

5. Teach

```
engine.cmd patch snap --tile 3-3 --at "700,4700 2100,4700" --by <net>
```

`snap` is for patches whose car marks were set by eye from nothing: it moves each car mark onto the net's own answer within 8 pixels, so that the marks of all patches mean one thing - the centre of the car. It adds and drops nothing. Patches whose marks began with `seed` do not need it. It needs a net that is there: before the first teaching skip it, and keep to the rule of step 4 - the point goes on the middle of the car.

```
engine.cmd net train --name cars-01 --steps 8000 --seed 1 --hold-tile "2-4 3-1"
engine.cmd net train --name cars-02 --steps 8000 --seed 2 --hold-tile "2-4 3-1"
engine.cmd net train --name cars-03 --steps 8000 --seed 3 --hold-tile "2-4 3-1"
engine.cmd net train --name cars-04 --steps 8000 --seed 4 --hold-tile "2-4 3-1"
engine.cmd net join --name cars-05 --of "cars-01 cars-02 cars-03 cars-04"
```

`train` teaches a new net, from nothing, on every marked patch of the work folder except those held back (`--hold-tile` - whole pictures, `--hold` - places), writes `<work>\models\<name>.pt` and prints the check on what was held back. The marks a net that came with the program was taught on do not come with it: a net of yours knows the user's marks alone. Hold back **every** check picture, every time. The names are examples: a name that is taken is refused, so the next four are `cars-06` to `cars-09`, joined as `cars-10`.

`--seed` is the random start of a teaching: the first weights and which squares of the patches are shown. One teaching is one throw of the dice: the same marks from another start give a net that counts up to 13 % differently at the same threshold. So a net for counting is made of four, and `join` writes one file that answers with the mean of its members; every command takes it by its name like any net. Compare only such nets of four with each other, never two single nets.

Time, in the author's work: 8000 steps took six minutes on an RTX 5070, and four teachings ran side by side in ten; 2000 steps took 40 minutes on a laptop processor. Leave `--width` and the other keys at their defaults: a wider net (`--width 32`) gave no gain beyond the noise, and more steps on the same few marks made a net that knew its patches rather than cars.

6. Judge: does the error hold

```
engine.cmd stability cars-05
```

Columns: the patches the net was taught on, then each picture it was not taught on, cut in two halves (`2-4 a`, `2-4 b`). Rows: thresholds. A cell: counted, the error against the marks, `m` - cars missed, `e` - answers on plain ground. Below the table the threshold is fixed on one half - where the count meets the marks - and the other half shows the error to expect; then the other way round.

Read three things, in this order:

1. **the error on the other half**, both ways. The two should be close to each other; their distance is the error that does not hold;
2. **how far the count moves when the threshold moves by 0.05** ("with the threshold 0.05 lower / higher"). A net that is sure of itself moves by a few percent; a count that swings by a third would move with any change of light between the strips of a mosaic;
3. `m` **against** `e`. A count that is right because 25 missed are cancelled by 25 extra is not a stable one: the two do not have to cancel elsewhere.

Then compare the column of the taught patches with the fresh ones: the threshold at which each meets its marks. A net that meets them at a higher threshold on its own patches knows its patches better than cars; more steps widen that gap, more marks close it.

The threshold belongs to the net: read it from its own table (in the author's work the joined nets met their marks at 0.20, their members at 0.25). Use it for `seed`, `overlay`, `answers`, `shaken`, the count and the report.

What the table cannot say:

- **A half of a picture may hold 25 to 60 cars.** A difference of ten percent between such halves is the dice, not the net. Two nets are told apart only when the check holds several hundred cars; until then say "the same within the check".
- **"Extra" is not "false".** Many answers the table calls extra stand on vehicles: a dark car in shadow whose `q` mark sits more than 7 pixels from the answer, the second car of a pair, a truck. Look at

them (`overlay`) before calling a net noisy; false answers are counted by eye on a sheet of `answers`.

```
engine.cmd shaken cars-05 0.20
```

The same fresh patches made lighter, darker, flatter, harsher, softer, sharper, tinted, pressed as a worse JPEG - twelve ways - and counted at that threshold. Nothing on the ground changes, so the count should not either. The last line says how far it went. This is the error to expect between strips of a mosaic that look different.

To see what a net gets wrong on a patch - rings are marks, crosses are answers, green when they found a marked car:

```
engine.cmd overlay cars-05 "2-3:11690,1960 2-4:9400,700" 0.2      ->
<work>\runs\overlay\2-3_11690_1960.jpg
```

A net is better when all three things of the table are better - not when one number is nearer to zero. The tables are only printed: keep their text for the user (for example in `<work>\runs\log\`), so that the next net can be set beside it.

7. Count whole pictures and look at the false answers

```
engine.cmd net count --weights cars-05 --tile 3-2      ->
<work>\runs\count_cars-05\3-2.csv, summary.json
engine.cmd answers count_cars-05 3-2 0.2 80           -> 3-2_where.jpg, 3-
2_sample.jpg
```

`count` writes every answer down to a sureness of 0.1 with its place in the picture and on the ground (from the `.tab`), so the threshold is chosen on the table afterwards. `answers` draws where they lie and a sheet of 80 at random with how sure the net was. Look at the sheet: what is not a car there is the next thing to mark - as patches at those places (steps 3-4; a patch of gravestones is `patch set --points ""`), which makes them hard ground for the next teaching.

Count a picture of every new kind of land whole before trusting a net on it.

8. The whole set, and the net it is counted with

```
engine.cmd net count --weights cars-05 --tile all
engine.cmd layout
```

One run folder, a csv a picture, one `summary.json` with the count of each picture and of the set at the thresholds 0.15, 0.20, 0.25, 0.35, 0.45 and 0.55; a count that stopped goes on with the pictures it has not written. A run is the count of one net and of one set. `layout` shows what neighbours share: every picture is counted whole and nothing is subtracted on the seams, so a car on a shared strip is counted twice (on the set the program was made on the strip is a few pixels wide).

The net a set is counted with is named by `<work>\models\counting.json`:

```
{"net": "cars-05", "least": 0.20}
```

No command writes this file: whoever accepts the net writes it. `net count --weights auto` and the button COUNT THE PICTURES take it, and count into the run `count_cars-05`. Write a net there only after its `stability` and `shaken` tables, and only when it is measurably better than what counted before - otherwise the number of the set moves for nothing. The first net of a set has nothing before it: write it when the user has seen its tables and takes the error they show. `least` must be one of the six thresholds of `summary.json`: the window shows the numbers of a run at those alone. To go back to a net that came with the program, name that net there with its own threshold - both stand in `config\counting_default.json`.

That file is of one set: a net taught here lies in `<work>\models` and counts this set. To count other sets with it - at the user's word - put a copy of its file into `system_core\nets` and name it in `config\counting_default.json` the same way: `{"net": "cars-05", "least": 0.20}`. A net file the user was given is put in just so. A net in the program counts every set whose work folder names none, so a new set of pictures is counted at once - and is examined first (step 1) when its source or its ground is new.

When the count on the check is steadily off, the coefficient is counted / marked on the check at that threshold. Tell the user the count, the coefficient and what it stands on; the program itself applies no coefficient.

9. The report and the layers

```
engine.cmd report count_cars-05 --out <folder>           the whole run: a Word
document and an Excel sheet
engine.cmd report count_cars-05 --out <folder> --tile 2-3 2-4  only these pictures
engine.cmd report count_cars-05 --out <folder> --formats mif --cell 250
                                                                layers for MapInfo
```

Nothing is counted here: the numbers are those of the run at `--least` (by default the threshold the set is counted with). The files are named by what they are and when they were made, in the language of `--lang` (`ru` by default; the window gives its own language).

The Word document: the figures (pictures, marks of the program, cars at least and at most, the area, marks per square kilometre), how to take the number, the layout of the pictures with the count on each, the table by picture, then a page a picture - the picture with a dot on every mark and 18 marks taken at random, enlarged. The Excel sheet: the table by picture, the marks at every threshold, every mark with its place in the picture and on the ground, and a service sheet - the net, the threshold, the folder of the run.

`--formats mif` writes, in the coordinate system of the pictures' `.tab` files (the `CoordSys` line of the first picture is copied word for word):

File	An object	Columns
<code>Cars_marks_<when>.mif</code> + <code>.mid</code>	a point a mark	the picture, the sureness, the pixel in the picture
<code>Cars_grid_250m_<when>.mif</code> + <code>.mid</code>	a square of <code>--cell</code> metres on the ground	the square's number, marks, cars from and to, the share covered by pictures (%), the area of that share, marks per km ² of it
<code>Cars_pictures_<when>.mif</code> + <code>.mid</code>	the outline of a picture	the picture, marks, cars from and to, area, marks per km ²
<code>Cars_density_raster_<when>.png</code> + <code>.tab</code>	the grid as a picture, a square coloured by its marks	a raster registration: MapInfo opens the <code>.tab</code> as it is

File	An object	Columns
Cars_density_map_<when>.png Cars_density_map_dark_<when>.png	the same map with its scale of colours and a caption	for a report and for a screen
Cars_layers_readme_<when>.txt		what the layers and the columns are

(These are the names of `--lang en`. `--lang` is `ru` when it is not given, and the names of the files and of the columns are then Russian.) Squares are laid by the ground - `floor(x / cell)`, `floor(y / cell)` - so two counts give the same squares. A square on the edge of the set is covered in part: its density is by the covered share. A picture without a `.tab` is left out; when none has one, the command says so and writes nothing. The tables are MIF/MID, the text interchange format of MapInfo (Table > Import), not native `.TAB` tables. **The files were read back and checked figure by figure, but have never been opened in MapInfo itself: say so to the user**, and take what they report from MapInfo as the first thing to act on.

The bounds - the marks less a quarter to the marks plus a half - are fixed in the program (`BAND` in `system_core\report.py`; the window uses the same). They come from the check pictures of the set the program was made on, where the marks "cannot tell" outnumbered the certain cars. They are not worked out anew for the user's pictures: say that too.

The words of the report are a reader's: "a mark of the program", "cars from ... to ...", "picture". When you tell the result to people who use the number, speak the same way; "net", "threshold" and "run" belong on the service sheet.

What was measured

In the author's work, with the author's nets, on the set the program was made on - one town, one source, about 0.3 m to a pixel. The author's net is not part of the public release (a private copy may carry it), so these are not properties of the copy you work in: they are what a net taught this way gave, the numbers to expect of a net you teach and to set it beside. None of them is a promise for other pictures.

- The author's net is four nets taught on the same marks - about 180 patches holding about 530 cars - and joined; its threshold is 0.20. "The net" below is that net.
- Three check pictures it never saw hold 511 cars marked for certain. Counted: 509; missed 45, extra 43. By picture: a plant with a large open car park -2 % (345 of 353, missed 16, extra 8); residential

ground - a settlement, a village, a works - +12 % (55 of 49); an old town under long shadows 0 % (109 of 109) - but there 25 cars are missed and 25 answers are extra, and they cancel.

- At the thresholds 0.15 and 0.25 instead of 0.20 the count of the car park picture is +6 % and -6 % of the marks; in the old town the same step moves the count by +39 % and -18 %.
- A picture that only looks different (**shaken**) moves the count by up to 4 % on open car parks and by up to 16 % in the old town under shadows.
- In the old town the eye itself left more "cannot tell" than cars, and the net answers on those too. That is where the bounds of the number come from; a net cannot narrow them, only a sharper picture of the same ground can.
- Looked at by eye on sheets of 80 answers of two whole pictures: on a town picture about 5 % were not cars and 6 % were trucks and vans; on a picture with no marks at all about 7 % were not cars or were machinery.
- Ground of a kind never marked gives false answers by the hundred: a picture of a plant gave 954 answers, about half of them false, before its roofs were marked; a cemetery gave an early net 94 answers, and 15 to a net taught with ten patches of hard ground of that picture marked.
- Another imagery source of the same town - lower effective resolution, blurred in parts: with marks set on it by eye (167 cars) the net found 58, about a third; it took light cars and missed dark ones. Making those pictures darker or matching their tone did not help.
- More marks of the same kind stop helping: on twenty fresh patches of the town centre the net found 90 of 92 cars before it was ever taught on them, and a net taught on them as well came out "the same within the check".
- Ready detectors for aerial pictures were tried on these pictures at enlargements from x1 to x10: the best found about two thirds of the cars of one car park, nearly all of them light ones, and one car of 22 in a yard. Do not propose them as they are.
- The tables of one net came out within a car or two on a video card and on a processor.

The window

Start.exe . It shows what the commands above work on and writes the same mark files; this is where the user marks when you are not to look at the pictures.

- **PICTURES** - the set as it lies on the ground. Each place: the marked patches and cars of the picture and, for the run and the threshold chosen on the right (ANSWERS OF THE NET), the answers. A click opens the picture in VIEW.
- **VIEW** - the whole picture small on the right, one piece large in the middle. The piece moves in steps: the arrow keys or the arrow buttons, a click on the whole picture, Page Up / Page Down

through the marked patches. Sizes of the piece: CLOSE (120 pixels), PATCH (240), 2 × 2 (480), 4 × 4 (960).

- **Marking in VIEW** - tools LOOK, CAR, TRUCK, CANNOT TELL, ERASE (keys 1-5). A click sets a point of the tool's kind; a click on a point changes its kind, and with the tool of its own kind takes it away; the right button erases with any tool. The first point in a square of the 240 grid makes that square a patch; PATCH WITHOUT CARS writes an empty one; TAKE THE NET'S ANSWERS puts the answers of the run shown into the patch as car marks, a draft to correct; UNDO (Ctrl+Z) steps back; REMOVE THE PATCH moves its file into `<work>\labels\_removed`. Every change is written at once.
- **Layers** - ANSWERS OF THE NET, MARKS, LIGHTEN (keys A, M, L).
- **COUNT** - COUNT THE PICTURES counts every picture the run of the counting net does not hold yet; REPORT ON ALL PICTURES, REPORT ON THIS PICTURE and LAYERS FOR MAPINFO are the `report` command of step 9; the row "Grid square" (100, 250, 500, 1000 m) is its `--cell`. Without a net the button is dimmed and the section says "No taught net in the work folder or in the program: nothing to count with."
- **REREAD** (bottom right) asks the engine about the set again: press it, or ask the user to, after marks, nets or runs were changed by commands.

Files

File	What
<code><work>\set.json</code>	the name of the set and the file names of the pictures this work folder was made on
<code><work>\labels<picture>\<x>_<y>.csv</code>	marks of a patch: <code>x,y,kind</code> , pixels of the picture
<code><work>\labels<picture>\ground.csv</code>	rectangles of plain ground: <code>x,y,w,h</code>
<code><work>\labels_removed<picture>\</code>	patches taken away in the window; nothing reads them
<code><work>\models<name>.pt</code>	a net: its weights and a note of what it was taught on - the places of the patches, those held back, steps, seconds, device
<code><work>\models\counting.json</code>	the net this set is counted with and its threshold
<code><work>\runs\count_<net>\<picture>.csv</code>	answers: <code>x,y,sure,ground_x,ground_y</code>
<code><work>\runs\count_<net>\summary.json</code>	<code>weights</code> , <code>device</code> , <code>source</code> (the set), <code>pictures</code> (the count of each by threshold), <code>total_by_least</code>

File	What
<code><work>\runs\look</code> , <code>sheets</code> , <code>overlay</code> , <code>overview</code>	pictures to look at; made again by the commands
<code><work>\runs\thumbs</code> , <code><work>\runs\pyramid</code>	small pictures and tiles for the window; made again when missing

When something goes wrong

What is printed	What to do
<code>Python is not found</code>	the engine is not installed: <code>SETUP</code> , <code>INSTALL WHAT IS NEEDED</code> , or <code>builder_main.cmd install_all</code>
<code>where</code> says <code>no card, the processor</code> on a machine with an NVIDIA card	<code>builder_main.cmd check_machine</code> for the driver, then <code>builder_main.cmd torch_cuda</code>
<code>N pictures end with _2-3.jpg, one is needed</code>	the folder of pictures is not the one <code>engine.cmd where</code> names, or holds two sets
<code>the work folder ... was made on other pictures</code>	a work folder belongs to one set: name another one for this set
<code>... is marked already</code>	<code>seed</code> does not write over marks: use <code>fix</code> , or <code>set</code> to write the patch anew on purpose
<code>... is already there</code> from <code>train</code> or <code>join</code>	take a new name for the net
<code>... is made of nets itself: name its members instead</code>	<code>join</code> takes single nets, not joined ones
<code>... holds the count of ...: a run is the count of one net</code>	name another run with <code>--out</code> , or leave <code>--out</code> out
<code>... is the count of the pictures of ..., the set now is ...</code>	the run is of another set: count and report it with that set

What is printed	What to do
<pre>... was taught on every marked patch: there is nothing fresh to count</pre>	<pre>shaken</pre> and <pre>stability</pre> need patches the net was not taught on: mark check patches and hold them back
<pre>no counted picture has its .tab beside it</pre>	the layers need the <pre>.tab</pre> of the pictures
<pre>no net to count with: none is named in ...\models\counting.json and this copy of the program carries none in ...\config\nets or ...\system_core\nets</pre>	no net was taught or put in yet - nothing is broken: "No net yet: the first net"
the window shows old numbers	REREAD