

Welcome Offer Access to everything. Now 30% off. [Upgrade now](#)

Image Classification +

Image Processing +

Semantic Segmentation +

How correlated objects work



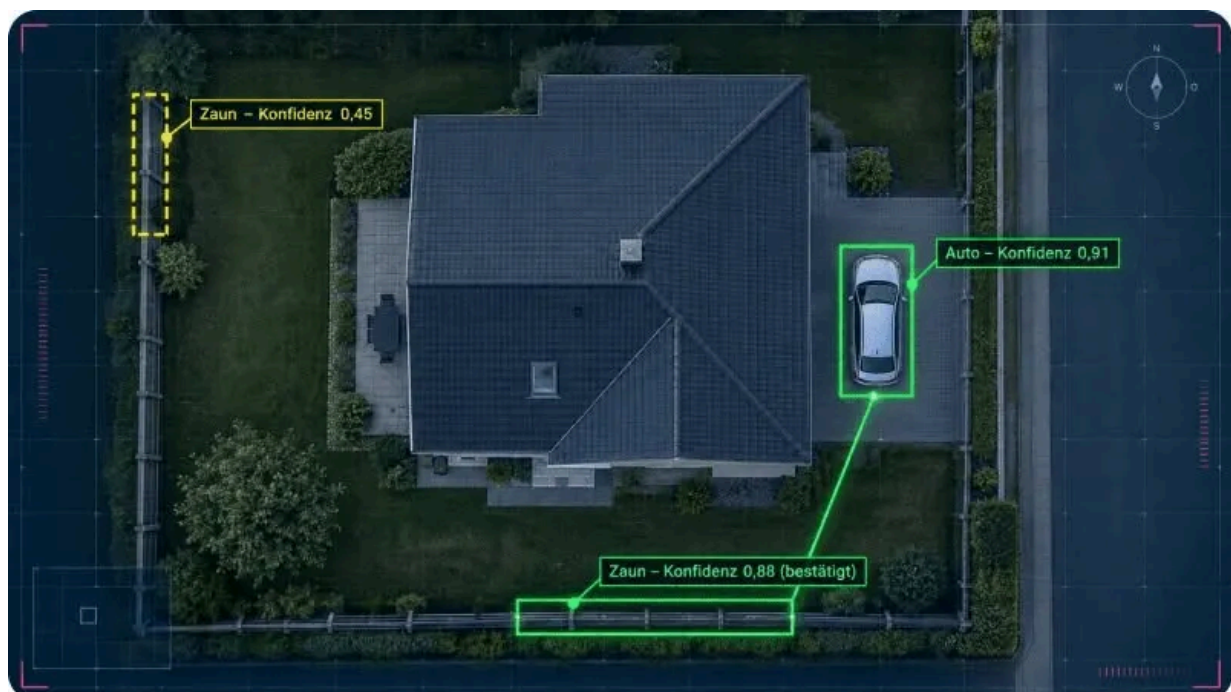
Max Kleiner 8 min read · Just now



Below is a self-contained **maXbox / Object Pascal** demo that implements contextual scoring for correlated objects in image recognition.

The concept of correlated objects links the local detection of an object with contextual knowledge about other objects in the same image. Thus, an object is evaluated not only based on its own appearance but also on whether its surroundings, location, size, and scene context are semantically plausible.

[[link.springer](#)][[arxiv](#)]



It does **not** require a neural network or image library: it assumes that a detector has already produced object candidates such as `car`, `fence`, `road`, or `rail`, each with a local confidence and a bounding box. The program then builds pairwise spatial relations and iteratively updates each candidate's probability using learned-like correlation rules.

```

268 | if I <> J then begin
269 |   Weight:= CorrelationWeight(Objects[I].ClassId, Objects[J].ClassId);
270 |
271 |   if Weight <> 0.0 then begin
272 |     Quality:= RelationQuality(Objects[I], Objects[J]);
273 |     Message:= Weight * Quality * Objects[J].Probability;
274 |
275 |     if Abs(Message) > 0.02 then
276 |       WriteLn(
277 |         Objects[J].Name + ' -> ' + Objects[I].Name +
278 |         ' weight=' + FloatToStrF(Weight, fffixed, 8, 2) +
279 |         ' relation=' + FloatToStrF(Quality, fffixed, 8, 2) +
280 |         ' evidence=' + FloatToStrF(Objects[J].Probability, fffixed, 8, 2) +
281 |         ' message=' + FloatToStrF(Message, fffixed, 8, 3)
282 |       );
283 |   end;
284 | end;
285 | end;
286 | end;
287 | end;

```

Interface List: 1498_corr_objects_concept.txt

```

*****
function ClassName(AClass: TObjectClass): string;
function Clamp(Value, LowValue, HighValue: Double): Double;
function Sigmoid(Value: Double): Double;
function Logit(Probability: Double): Double;
function CenterX(const Obj: TObjectCandidate): Double;
function CenterY(const Obj: TObjectCandidate): Double;
function Distance(const A, B: TObjectCandidate): Double;
function HorizontalOverlap(const A, B: TObjectCandidate): Double;
function VerticalOverlap(const A, B: TObjectCandidate): Double;
function Nearest(const A, B: TObjectCandidate): Double;
function IsBlInFrontOfA(const A, B: TObjectCandidate): Boolean;
function RelationQuality(const Target, Evidence: TObjectCandidate): Double;
function CorrelationWeight(TargetClass, EvidenceClass: TObjectClass): Double;
procedure PrintObjects(const Title: string; const Objects: array of TObjectCandidate);
procedure ExplainRelations(const Objects: array of TObjectCandidate);
Locs: 434 - code blocks: 15

```

maXbox5 C:\maxbox\maxbox5\examples\1498_corr_objects_concept.txt Ct:26/09/2026 15:56:29 Mem:66%

Row: 275-Col: 1 s: 7450 S

Interpretation:

- FenceCandidate is strengthened by the nearby confident car.
- CarCandidate is strongly supported by the overlapping road.
- TrainCandidate is strengthened by the nearby aligned rail.
- Distant vegetation has almost no contextual influence.

mX5 executed: 26/09/2026 15:56:29 Runtime: 0:0:3.233 Memload: 66% use

RemObjects Pascal Script. Copyright (c) 2004-2026 by RemObjects Software & maXbox5

Ver: 5.2.9.198 (529). Workdir: C:\maxbox\maxbox5\pascalmaster2\pascalmaster2\restunits\Indy9\maxbox5\maxbox5

https://sourceforge.net/projects/maxbox5/files/examples/1498_corr_objects_concept.txt/download

maXbox is an Object Pascal scripting environment based on PascalScript, intended for experimenting with algorithms and executable scripts in one application.

What it demonstrates

The program models:

- A local detector confidence for every candidate.
- Pairwise relations based on:

- distance between bounding-box centers,
- relative horizontal/vertical position, overlap, relative size.

Correlation rules such as:

- `road → car` is supportive if the car lies on/near the road,
- `car → fence` is mildly supportive only at short range,
- `rail → train` is strongly supportive when aligned and close,
- `fence → car` is weaker than `car → fence` in this sample.

Iterative **belief propagation-like rescoring**:

- Each candidate receives messages from nearby candidates.
- A context message changes the candidate's original detector log-odds.
- The local classifier remains the principal source of evidence.

The relation is deliberately **directional**: `Auto(car) → Zaun(fence)` and `Zaun → Auto` may have different weights. This reflects the fact that conditional probabilities are asymmetric:

$$P(\text{Zaun} \mid \text{Auto}, r) \neq P(\text{Auto} \mid \text{Zaun}, r).$$

maXbox source code

Paste the following into a new maXbox script and run it. It prints the initial and context-adjusted confidences to the console/log.

```
program CorrelatedObjectsDemo;

const
  MAX_OBJECTS = 12;
  MAX_ITERATIONS = 7;
```

```

type
  TObjectClass = (
    ocUnknown,
    ocCar,
    ocFence,
    ocRoad,
    ocRail,
    ocTrain,
    ocVegetation
  );

  TObjectCandidate = record
    Name: string;
    ClassId: TObjectClass;
    LocalProbability: Double;
    Probability: Double;
    NewProbability: Double;
    X: Double;
    Y: Double;
    W: Double;
    H: Double;
  end;

function ClassName(AClass: TObjectClass): string;
begin
  case AClass of
    ocCar:      Result:= 'Car';
    ocFence:    Result:= 'Fence';
    ocRoad:     Result:= 'Road';
    ocRail:     Result:= 'Rail';
    ocTrain:    Result:= 'Train';
    ocVegetation: Result:= 'Vegetation';
  else
    Result:= 'Unknown';
  end;
end;

function Clamp(Value, LowValue, HighValue: Double): Double;
begin
  Result:= Value;
  if Result < LowValue then
    Result:= LowValue;
  if Result > HighValue then
    Result:= HighValue;
end;

function Sigmoid(Value: Double): Double;
begin
  if Value > 30.0 then begin
    Result:= 1.0;
    Exit;
  end;

  if Value < -30.0 then begin
    Result:= 0.0;
    Exit;
  end;

  Result:= 1.0 / (1.0 + Exp(-Value));

```

```
end;

function Logit(Probability: Double): Double;
var
  P: Double;
begin
  P:= Clamp(Probability, 0.001, 0.999);
  Result:= Ln(P / (1.0 - P));
end;

function CenterX(const Obj: TObjectCandidate): Double;
begin
  Result:= Obj.X + Obj.W / 2.0;
end;

function CenterY(const Obj: TObjectCandidate): Double;
begin
  Result:= Obj.Y + Obj.H / 2.0;
end;

function Distance(const A, B: TObjectCandidate): Double;
var
  DX, DY: Double;
begin
  DX:= CenterX(A) - CenterX(B);
  DY:= CenterY(A) - CenterY(B);
  Result:= Sqrt(DX * DX + DY * DY);
end;

function HorizontalOverlap(const A, B: TObjectCandidate): Double;
var
  LeftEdge, RightEdge, OverlapWidth, MinWidth: Double;
begin
  LeftEdge:= A.X;
  if B.X > LeftEdge then
    LeftEdge:= B.X;

  RightEdge:= A.X + A.W;
  if B.X + B.W < RightEdge then
    RightEdge:= B.X + B.W;

  OverlapWidth:= RightEdge - LeftEdge;
  if OverlapWidth < 0.0 then
    OverlapWidth:= 0.0;

  MinWidth:= A.W;
  if B.W < MinWidth then
    MinWidth:= B.W;

  if MinWidth <= 0.0 then
    Result:= 0.0
  else
    Result:= Clamp(OverlapWidth / MinWidth, 0.0, 1.0);
end;

function VerticalOverlap(const A, B: TObjectCandidate): Double;
var
  TopEdge, BottomEdge, OverlapHeight, MinHeight: Double;
begin
```

```

TopEdge:= A.Y;
if B.Y > TopEdge then
  TopEdge:= B.Y;

BottomEdge:= A.Y + A.H;
if B.Y + B.H < BottomEdge then
  BottomEdge:= B.Y + B.H;

OverlapHeight:= BottomEdge - TopEdge;
if OverlapHeight < 0.0 then
  OverlapHeight:= 0.0;

MinHeight:= A.H;
if B.H < MinHeight then
  MinHeight:= B.H;

if MinHeight <= 0.0 then
  Result:= 0.0
else
  Result:= Clamp(OverlapHeight / MinHeight, 0.0, 1.0);
end;

function Nearness(const A, B: TObjectCandidate; MaxDistance: Double): Double;
var D: Double;
begin
  D:= Distance(A, B);

  if D >= MaxDistance then
    Result:= 0.0
  else
    Result:= 1.0 - D / MaxDistance;
end;

function IsBInFrontOfA(const A, B: TObjectCandidate): Boolean;
begin
  { Image coordinates: greater Y means lower in the picture.
    B is considered "in front" if it is lower than A. }
  Result:= CenterY(B) > CenterY(A);
end;

function RelationQuality(const Target, Evidence: TObjectCandidate): Double;
var
  NearValue, HOverlap, VOverlap: Double;
begin
  NearValue:= Nearness(Target, Evidence, 180.0);
  HOverlap:= HorizontalOverlap(Target, Evidence);
  VOverlap:= VerticalOverlap(Target, Evidence);

  Result:= NearValue;

  { Special geometries can strengthen a relation. }
  if (Target.ClassId = ocCar) and (Evidence.ClassId = ocRoad) then
    Result:= Clamp(0.55 * NearValue + 0.45 * HOverlap, 0.0, 1.0);

  if (Target.ClassId = ocTrain) and (Evidence.ClassId = ocRail) then
    Result:= Clamp(0.55 * NearValue + 0.45 * HOverlap, 0.0, 1.0);

  if (Target.ClassId = ocFence) and (Evidence.ClassId = ocCar) then begin
    Result:= 0.70 * NearValue;
  end;
end;

```

```

    if IsBinFrontOfA(Target, Evidence) then
        Result:= Result + 0.15;
        Result:= Clamp(Result, 0.0, 1.0);
    end;

    if (Target.ClassId = ocCar) and (Evidence.ClassId = ocFence) then
        Result:= Clamp(0.60 * NearValue + 0.20 * VOverlap, 0.0, 1.0);
    end;

function CorrelationWeight(TargetClass, EvidenceClass: TObjectClass): Double;
begin
    Result:= 0.0;

    { Directional contextual weights.
      Positive = evidence supports the target hypothesis.
      Negative = evidence weakens the target hypothesis. }

    if (TargetClass = ocCar) and (EvidenceClass = ocRoad) then
        Result:= 1.30;
    if (TargetClass = ocRoad) and (EvidenceClass = ocCar) then
        Result:= 0.35;

    if (TargetClass = ocFence) and (EvidenceClass = ocCar) then
        Result:= 0.65;
    if (TargetClass = ocCar) and (EvidenceClass = ocFence) then
        Result:= 0.25;

    if (TargetClass = ocTrain) and (EvidenceClass = ocRail) then
        Result:= 1.60;
    if (TargetClass = ocRail) and (EvidenceClass = ocTrain) then
        Result:= 0.55;

    if (TargetClass = ocTrain) and (EvidenceClass = ocRoad) then
        Result:= -0.55;
    if (TargetClass = ocRail) and (EvidenceClass = ocVegetation) then
        Result:= -0.20;
    end;

procedure PrintObjects(const Title: string;
    const Objects: array of TObjectCandidate; Count: Integer);
var
    I: Integer;
begin
    Writeln('');
    Writeln(Title);
    Writeln('-----');

    for I:= 0 to Count - 1 do begin
        Writeln(
            Objects[I].Name + '  class=' + ClassName(Objects[I].ClassId) +
            '  local=' + FloatToStrF(Objects[I].LocalProbability, ffFixed, 8, 3) +
            '  final=' + FloatToStrF(Objects[I].Probability, ffFixed, 8, 3) +
            '  box=(' + FloatToStrF(Objects[I].X, ffFixed, 8, 0) + ', ' +
                FloatToStrF(Objects[I].Y, ffFixed, 8, 0) + ', ' +
                FloatToStrF(Objects[I].W, ffFixed, 8, 0) + ', ' +
                FloatToStrF(Objects[I].H, ffFixed, 8, 0) + ')';
        );
    end;
end;

```

```

procedure ExplainRelations(const Objects: array of TObjectCandidate;
    Count: Integer);
var
    I, J: Integer;
    Weight, Quality, Message: Double;
begin
    Writeln('');
    Writeln('Relevant contextual relations');
    Writeln('-----');

    for I:= 0 to Count - 1 do begin
        for J:= 0 to Count - 1 do begin
            if I <> J then begin
                Weight:= CorrelationWeight(Objects[I].ClassId, Objects[J].ClassId);

                if Weight <> 0.0 then begin
                    Quality:= RelationQuality(Objects[I], Objects[J]);
                    Message:= Weight * Quality * Objects[J].Probability;

                    if Abs(Message) > 0.02 then
                        Writeln(
                            Objects[J].Name + ' -> ' + Objects[I].Name +
                            ' weight=' + FloatToStrF(Weight, ffFixed, 8, 2) +
                            ' relation=' + FloatToStrF(Quality, ffFixed, 8, 2) +
                            ' evidence=' + FloatToStrF(Objects[J].Probability, ffFixed, 8, 2) +
                            ' message=' + FloatToStrF(Message, ffFixed, 8, 3)
                        );
                end;
            end;
        end;
    end;
end;

type TCand_Objects = array[0..MAX_OBJECTS - 1] of TObjectCandidate;

const
    CONTEXT_STRENGTH = 0.85;
    DAMPING = 0.55;

procedure UpdateProbabilities(var Objects: TCand_Objects; {array of TObjectCandi
    Count, Iterations: Integer);
var
    I, J, Step: Integer;
    ContextMessage, Weight, Quality: Double;
    BaseLogit, UpdatedLogit: Double;
begin
    for Step:= 1 to Iterations do begin
        for I:= 0 to Count - 1 do begin
            ContextMessage := 0.0;

            for J:= 0 to Count - 1 do begin
                if I <> J then begin
                    Weight := CorrelationWeight(
                        Objects[I].ClassId,
                        Objects[J].ClassId
                    );

```

```

    if Weight <> 0.0 then begin
        Quality:= RelationQuality(Objects[I], Objects[J]);

        { Strong evidence has more influence.
          The relation quality goes to zero for distant or
            geometrically implausible neighbours. }
        ContextMessage:= ContextMessage +
            Weight * Quality * Objects[J].Probability;
    end;
end;
end;

{ The local detector remains the anchor.
  Context adjusts its log-odds rather than replacing it. }
BaseLogit:= Logit(Objects[I].LocalProbability);
UpdatedLogit:= BaseLogit + CONTEXT_STRENGTH * ContextMessage;
Objects[I].NewProbability := Sigmoid(UpdatedLogit);
end;

{ Damping avoids unstable oscillation during iterations. }
for I:= 0 to Count - 1 do
    Objects[I].Probability :=
        DAMPING * Objects[I].NewProbability +
        (1.0 - DAMPING) * Objects[I].Probability;
    WriteLn('');
    WriteLn('Iteration ' + IntToStr(Step));
    for I:= 0 to Count - 1 do
        WriteLn(
            Objects[I].Name + ': ' +
            FloatToStrF(Objects[I].Probability, ffFixed, 8, 3)
        );
    end;
end;

//type TCand_Objects = array[0..MAX_OBJECTS - 1] of TObjectCandidate;
var
    Objects: TCand_Objects; //array[0..MAX_OBJECTS - 1] of TObjectCandidate;
    Count, I: Integer;

begin    //@main
    WriteLn('Correlated Objects / maXbox Contextual Rescoring Demo');
    WriteLn('=====');

    Count:= 6;

    { Candidate 0: weak local fence detection. }
    Objects[0].Name:= 'FenceCandidate';
    Objects[0].ClassId := ocFence;
    Objects[0].LocalProbability:= 0.48;
    Objects[0].X:= 120;
    Objects[0].Y:= 130;
    Objects[0].W:= 150;
    Objects[0].H:= 18;

    { Candidate 1: a strong car detection in front of the fence. }
    Objects[1].Name:= 'CarCandidate';
    Objects[1].ClassId := ocCar;
    Objects[1].LocalProbability := 0.93;

```

```

Objects[1].X:= 160;
Objects[1].Y:= 160;
Objects[1].W:= 54;
Objects[1].H:= 30;

{ Candidate 2: road context supporting the car. }
Objects[2].Name:= 'RoadCandidate';
Objects[2].ClassId:= ocRoad;
Objects[2].LocalProbability:= 0.89;
Objects[2].X:= 75;
Objects[2].Y:= 150;
Objects[2].W:= 280;
Objects[2].H:= 105;

{ Candidate 3: weak train hypothesis. }
Objects[3].Name:= 'TrainCandidate';
Objects[3].ClassId:= ocTrain;
Objects[3].LocalProbability:= 0.41;
Objects[3].X:= 440;
Objects[3].Y:= 310;
Objects[3].W:= 155;
Objects[3].H:= 42;

{ Candidate 4: a strong rail detection aligned with train. }
Objects[4].Name:= 'RailCandidate';
Objects[4].ClassId:= ocRail;
Objects[4].LocalProbability:= 0.92;
Objects[4].X:= 400;
Objects[4].Y:= 325;
Objects[4].W:= 270;
Objects[4].H:= 20;

{ Candidate 5: unrelated vegetation. }
Objects[5].Name:= 'VegetationCandidate';
Objects[5].ClassId:= ocVegetation;
Objects[5].LocalProbability := 0.86;
Objects[5].X:= 720;
Objects[5].Y:= 100;
Objects[5].W:= 130;
Objects[5].H:= 190;

{ Initialize final probability from local detector probability. }
//var I: Integer;
for I:= 0 to Count - 1 do begin
  Objects[I].Probability:= Objects[I].LocalProbability;
  Objects[I].NewProbability:= Objects[I].LocalProbability;
end;

PrintObjects('Initial detector output', Objects, Count);
ExplainRelations(Objects, Count);

UpdateProbabilities(Objects, Count, MAX_ITERATIONS);

PrintObjects('Final probabilities after contextual rescoring', Objects, Count);

Writeln('');
Writeln('Interpretation:');
Writeln('- FenceCandidate is strengthened by the nearby confident car. ');
Writeln('- CarCandidate is strongly supported by the overlapping road. ');

```

```

WriteLn('- TrainCandidate is strengthened by the nearby aligned rail.');
```

```

WriteLn('- Distant vegetation has almost no contextual influence.');
```

```

end.
```

How the algorithm works:

For each candidate i , the code starts from its local detector probability:

$$p_i^{local}.$$

It transforms that into log-odds:

$$\text{logit}(p_i^{local}) = \ln \left(\frac{p_i^{local}}{1 - p_i^{local}} \right).$$

Then it adds weighted context messages from all other candidates j :

$$m_i = \sum_{j \neq i} w(c_i, c_j) \cdot q(r_{ij}) \cdot p_j.$$

Where:

- $w(c_i, c_j)$ is the directional semantic correlation weight, supplied by `CorrelationWeight`.
- $q(r_{ij})$ is `RelationQuality`, a value from 0 to 1 derived from distance, overlap, and selected geometrical tests.
- p_j is the current confidence of the evidence object.
- The context-adjusted probability is:

$$p_i^{new} = \sigma (\text{logit}(p_i^{local}) + \lambda m_i).$$

The script repeats this update five times. Its **damping** mixes the new result with the prior iteration so that mutually supporting objects do not jump abruptly toward 1.0.

Important changes for real data

The numbers in `CorrelationWeight` are illustrative. For real imagery, do not invent them manually unless you are building a rule-based prototype. Estimate them from annotated data.

For every class pair A, B and relation bucket r , count:

$$\hat{P}(A \mid B, r) = \frac{\#(A \text{ and } B \text{ in relation } r)}{\#(B \text{ in relation } r)}.$$

Then use a smoothed log-odds ratio rather than a raw fixed score:

$$w(A, B, r) = \log \left(\frac{\hat{P}(A \mid B, r)}{P(A)} \right).$$

Interpretation:

- $w > 0$: observing B in relation r makes A more plausible.
- $w = 0$: the evidence provides no information beyond the baseline.
- $w < 0$: it makes A less plausible.

For a railway-focused classifier, sensible candidate classes might include `rail`, `train`, `platform`, `catenary mast`, `signal`, `switch`, and `ballast`.

Relations such as parallel alignment, regular mast spacing, closeness to a track centerline, and position alongside a platform would be considerably more useful than a generic “near” test.

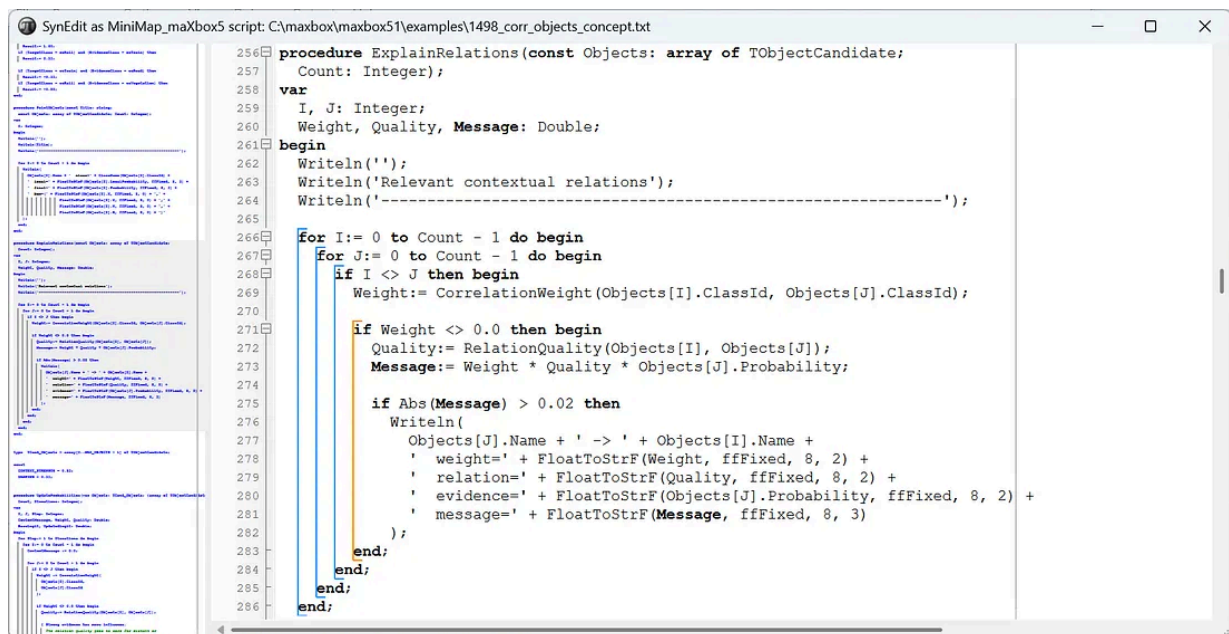
A practical caveat

This is a compact **context-rescoring model**, not a trained neural net. It is useful for validating the principle, testing a rule set, or post-processing detections from a separate model such as YOLO.



For a production model, you would normally choose one of these approaches:

- Train an object detector or segmenter that learns context implicitly from large receptive fields and attention.
- Attach a graph neural network or relation module to detected objects; relation weights are learned end-to-end.
- Use a CRF over pixels/regions when boundary-aware semantic segmentation is the main task.
- Retain an explicit rule-based post-processor like this when relations must be inspectable, auditable, and easily adapted to a specific GEOINT or infrastructure domain.



```

256 procedure ExplainRelations(const Objects: array of TObjectCandidate;
257   Count: Integer);
258 var
259   I, J: Integer;
260   Weight, Quality, Message: Double;
261 begin
262   Writeln('');
263   Writeln('Relevant contextual relations');
264   Writeln('-----');
265
266   for I:= 0 to Count - 1 do begin
267     for J:= 0 to Count - 1 do begin
268       if I <> J then begin
269         Weight:= CorrelationWeight(Objects[I].ClassId, Objects[J].ClassId);
270
271         if Weight <> 0.0 then begin
272           Quality:= RelationQuality(Objects[I], Objects[J]);
273           Message:= Weight * Quality * Objects[J].Probability;
274
275           if Abs(Message) > 0.02 then
276             Writeln(
277               Objects[J].Name + ' -> ' + Objects[I].Name +
278               ' weight=' + FloatToStrF(Weight, ffFixed, 8, 2) +
279               ' relation=' + FloatToStrF(Quality, ffFixed, 8, 2) +
280               ' evidence=' + FloatToStrF(Objects[J].Probability, ffFixed, 8, 2) +
281               ' message=' + FloatToStrF(Message, ffFixed, 8, 3)
282             );
283         end;
284       end;
285     end;
286   end;
  
```

https://sourceforge.net/projects/maxbox5/files/examples/1498_corr_objects_concept.txt/download



CC 1801 B Polytension

Image Classification



Image Processing



Semantic Segmentation



Written by Max Kleiner 

Follow

52 followers · 3 following

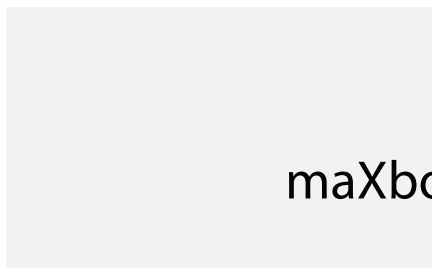
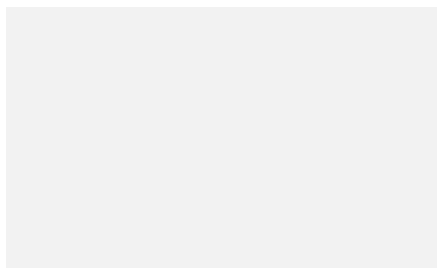
Max Kleiner's professional environment is in the areas of OOP, UML and coding - among other things as a trainer, developer and consultant.



Placeholder text for a bio or description.

Placeholder text for a bio or description.

Placeholder text for a bio or description.



maXbox4exe

Digitally signed by
maXbox4exe
Date: 2026.09.26 17:11:58
+02'00'