タイル・アンサンブル・メカニズムを Anomalib に追加する — GSoC 2023 @ OpenVINO™

この記事は、Medium に公開されている「Add ensembling methods for tiling to Anomalib — GSoC 2023 @ OpenVINO™」の日本語参考訳です。原文は更新される可能性があります。原文と翻訳文の内容が異なる場合は原文を優先してください。

高解像度イメージの不具合を検出しようとすると、多くの課題に直面します。その 1 つは、メモリーの制約によりモデルのトレーニングが困難なことです。ダウンサンプリングにより情報が失われると、小さな不具合が見逃されてしまいます。

この問題を回避する 1 つの方法は、タイル・アンサンブル・メカニズムを使用することです。この手法は、イメージを小さなタイルに分割し、タイルの位置ごとに個別のモデルを使用してイメージを処理します。この処理により、位置特定が強化され、小さな異常の検出率が向上すると同時に、モデルがメモリーの制約内に収まることが保証されます。

この記事では、ディープラーニング・モデルを利用した異常検出ライブラリーの **Anomalib** 内でタイル・アンサンブル・メカニズムがどのように開発されたかを紹介します。このプロジェクトは、インテルの **OpenVINO™** ツールキット・チームの **Google Summer of Code** プロジェクトの一部です。最終的な結果は、任意のデータセットと Anomalib にすでに実装されているモデルを使用した、データから最終予測までのパイプラインです。それでは、このメカニズムの設計、進化、統合の成功について説明しましょう。

この記事には手順と結果を説明する多くの図が含まれています。すべてのテキストを読む時間がない場合でも、すべての図に目を通すことで優れたアイデアを得ることができるでしょう。

Google Summer of Code プロジェクト (英語)
Anomalib (GitHub*) (英語) | OpenVINO (GitHub*) (英語)

この GSoC 期間中、素晴らしいメンターとして協力してくれた @samet-akcay (英語) と @djdameln (英語) に感謝します。

はじめに

イメージ内の異常検出は、研究および産業用アプリケーションにおける重要なタスクです。ここ数年、ディープラーニングの進歩とともに、このタスクに対するさまざまな手法のパフォーマンスが大幅に向上しました。

ただし、実際のアプリケーションでは、モデルはメモリー使用量とスループットにより制限されます。このようなモデルは通常、製品管理に含まれ、高いスループットと高い精度が期待されます。不具合を見逃すと重大な経済的損失を引き起こしたり、人間の健康に害を及ぼす可能性があります。

カプセルの異常検出のサンプルの 1 つを図 1 に示します。

図 1: VisA データセットの異常検出のサンプル。このサンプルでは、カプセルの不具合を見つけます。

産業用アプリケーションでは、多くの場合、高解像度イメージの優れた位置特定と検出精度が求められますが、通常のアプローチではすぐにメモリーの問題が発生します。

この問題に対処するのが、タイル・アンサンブルのアイデアです。図 2 に示すように、このプロセスは個々のイメージに適用されますが、データセット全体で同じように機能します。イメージは選択した数のタイルに分割され、特定の位置のタイルに対して個別のモデルをトレーニングします。

最後に、すべてのモデルの予測が結合されて処理され、高解像度の異常マスクと、物体が異常かどうかを示すスコアが得られます。

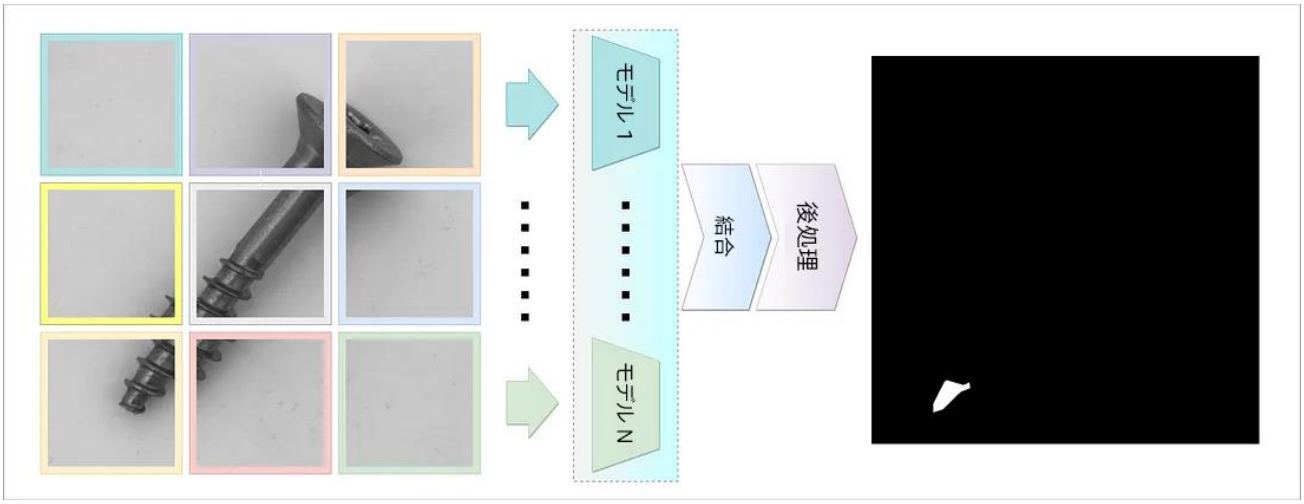

図 2: タイル・アンサンブル・アプローチの高レベルの概要。各イメージはタイルに分割され、タイルの位置ごとに個別のモデルをトレーニングします。最後に、予測が結合されて後処理され、スコアと予測異常マスクが生成されます。

Anomalib には現在、基本的なタイル処理機能が用意されています。ただし、アンサンブルなしで使用すると、PaDiM のような位置認識モデルの利点は減ります。イメージをタイリングして単一のモデルを使用するだけで、入力の同じピクセルが元のイメージの複数の位置に対応するようになりますが、メモリー効率は良くありません。

この問題に対処するため、位置認識の利点を維持しつつパフォーマンスを強化する、タイル・アンサンブルの実装を紹介します。Anomalib にすでに実装されているデータセットとモデルで機能します。

設計

このアプローチは、Anomalib の既存の機能 (トレーニング構成、標準の後処理、視覚化、メトリック計算など) を組み込むことを目的としています。パイプライン内の特定のステップをカスタマイズできる柔軟性をユーザーに提供しつつ、標準の Anomalib 予測の形式と一致する結果が得られます。メモリー効率は維持されます。

図 3 で、アプローチのすべての構成要素を確認できます。分かりやすいように、一度に 1 つのイメージを処理しているようにメソッドの動作を説明しますが、トレーニング中はすべてバッチで実行されます。

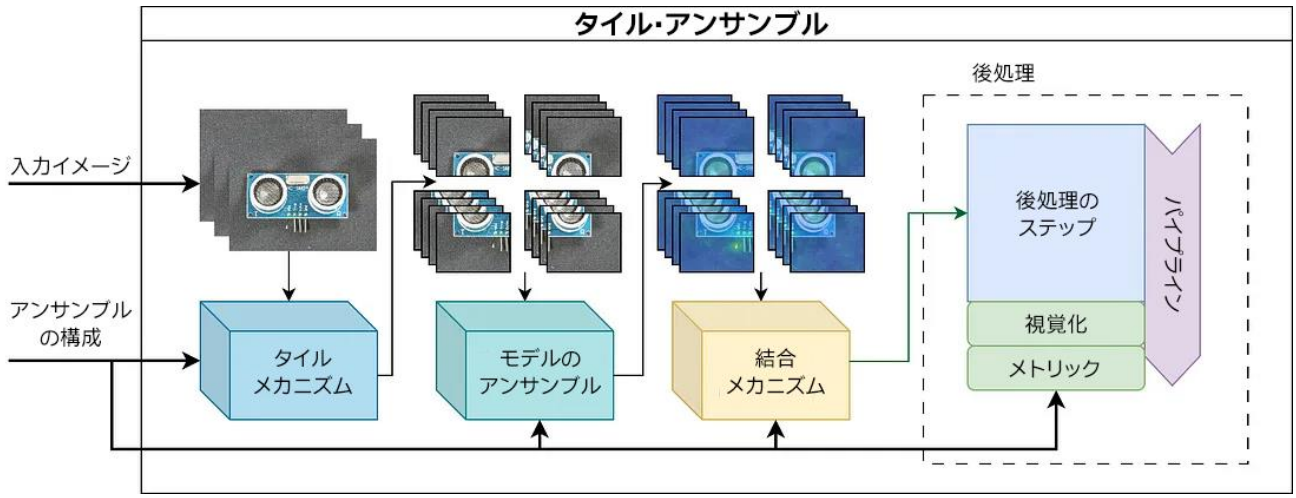

図 3: タイル・アンサンブル・メカニズムに結合される構成要素の図。最初に、イメージはタイルに分割され、タイルの位置ごとに個別のモデルをトレーニングします。次に、タイルレベルの予測が結合され、後処理パイプラインを通じて最終的に送られます。

タイリングとトレーニング

入力イメージは同じサイズのタイルにタイル状に並べられます。オーバーラップすることもできます。次に、1 つの位置からすべてのタイルが個別のモデルに入力されます。すべての位置のすべての個別のモデルをトレーニングするまで、すべての位置のタイルで、この処理が繰り返されます。タイリングの様子を、図 4 に示します。

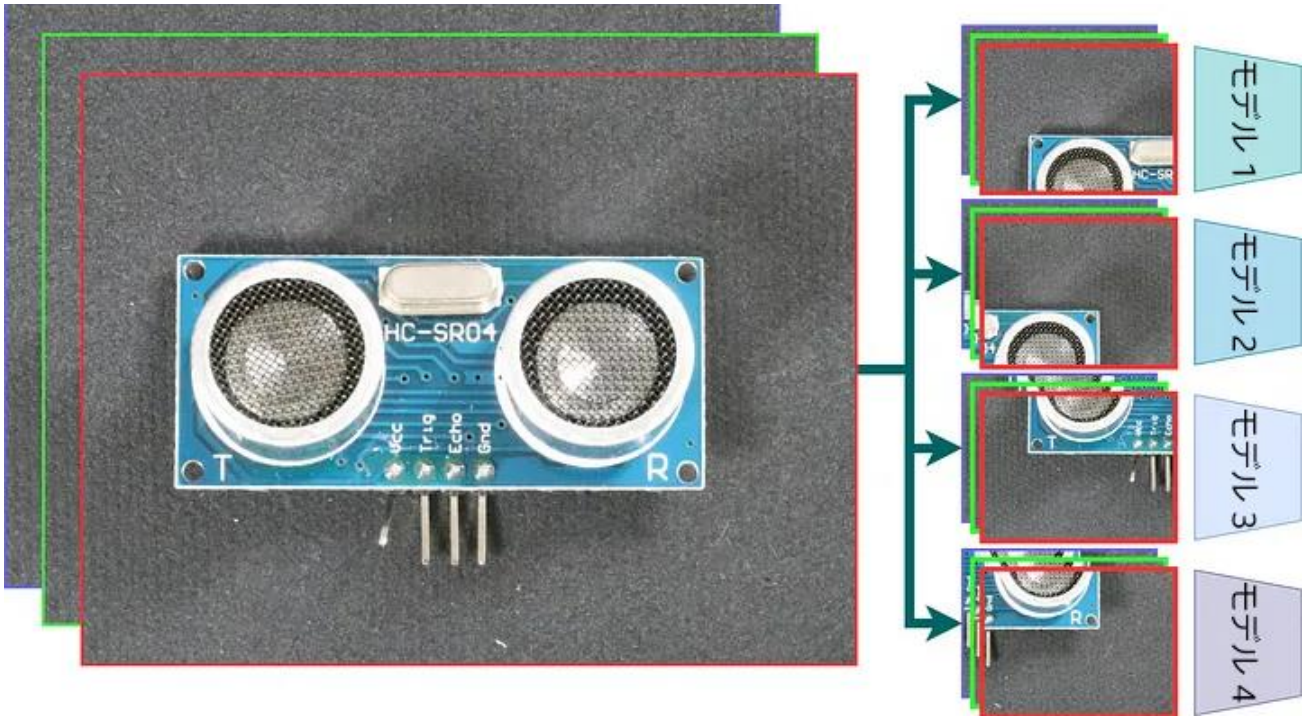

図 4: タイリングとタイルのデータのトレーニング。各モデルは 1 つの位置のタイルのみトレーニングします。

推論の結果は異常マップであり、すべてのピクセルのスコアです。図 5 の「予測ヒートマップ」で確認できます。ここで、対応するセグメンテーション・マスクも確認できます。

異常スコアは、異常マップの最大値として決定されます。この値が特定のしきい値を超えると、イメージ全体が異常としてラベル付けされます。

Anomalib は検出タスクもサポートしています。このタスクには、図 5 の「予測」で示されているように、異常マップから異常領域の周囲に境界ボックスを生成することも含まれます。

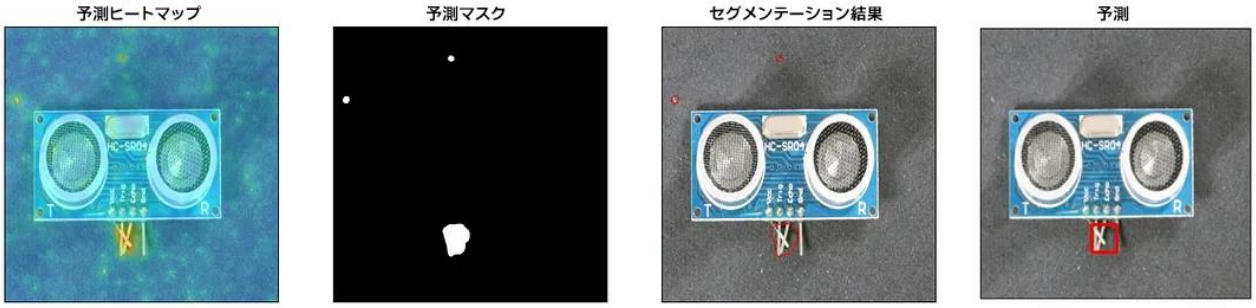

図 5: 結果の例。異常マップ (図の「予測ヒートマップ」) は、ピクセルごとの異常スコアを示します。決定されたしきい値を使用して、異常マップから「予測マスク」が生成されます。「セグメンテーション結果」は、マスクに基づいて、元のイメージの異常な部分を示します。「予測」では、セグメンテーションの線の代わりに境界ボックスを使用して異常領域をマークしています。

上記の例はイメージ全体の予測を示していますが、タイル・アンサンブルは、タイルごとに個別に予測を取得します。上記の結果を得るには、最初にすべてを結合して完全なイメージ表現を作成する必要があります。

結合

タイルの予測をイメージに戻すには、ジョイナーを使用して異常マップ、スコア、予測ボックスを結合します。図 6 に示すように、次の処理を行います。

- 異常マップなどの小さなタイルに分割したデータを再構築して元に戻します。
- すべてのタイルのスコアを平均化します。
- ボックスを 1 つのリストにまとめます。

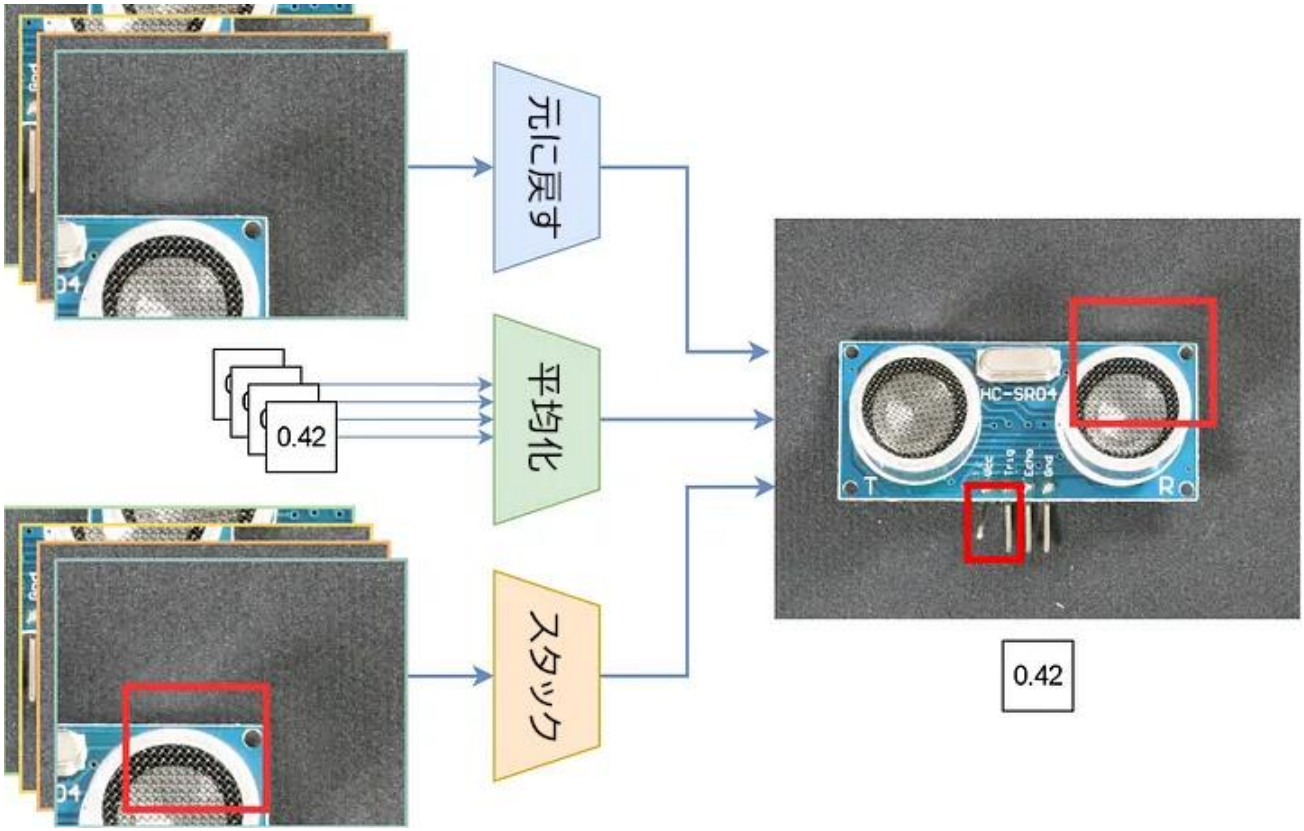

図 6: 3 つのレベルで結合を行います。異常マップやイメージなどのタイルに分割したデータを元に戻し、タイル全体でスコアを平均化し、すべてのタイルからボックスをスタックします。

結合後は、アンサンブルを使用しない場合の予測と同様に、すべての結果をイメージレベルの表現として得ることができます。最後のステップとして、後処理パイプラインを経由します (図 7)。

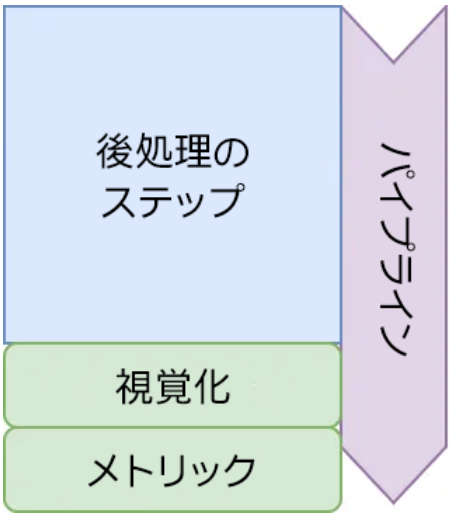

図 7: タイル・アンサンブル予測に使用する後処理パイプライン。最初にデータに後処理を適用し、次に視覚化してメトリックを計算します。

後処理

結果を改善するため、正規化やしきい値処理を含む、さまざまな後処理のステップを適用します。

正規化としきい値処理は、タイルごとに個別に実行するか、結合したイメージに対して実行します。どのように実行するかはユーザーが選択できます。デフォルトでは、優れた視覚化が作成されるように、イメージを結合した後、結合したイメージに対して実行します。

この段階で、最終評価に備えた予測を準備します。この部分は、視覚化とメトリックの計算で構成されます。

結果を視覚化すると、図 8 に示すように、予測のすべての部分とグラウンドトゥルースが表示されます。

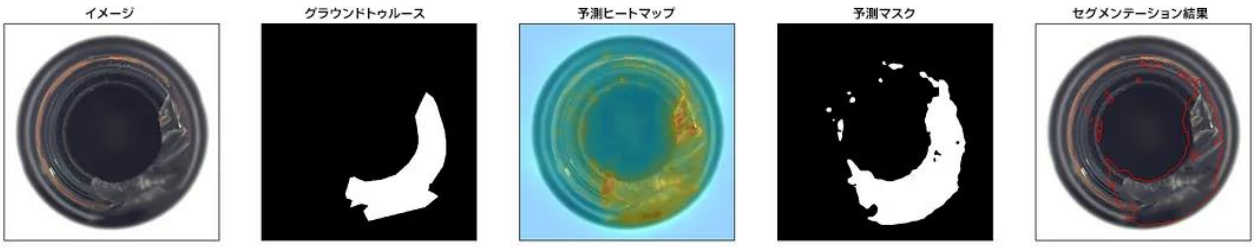

図 8: 視覚化の結果の例。予測された各要素を詳細に確認できます。イメージの異常な領域を示すグラウンドトゥルースと比較することもできます。

このアプローチのパフォーマンスを定量的に評価するため、最後に予測データのメトリックを計算します。通常は、イメージとピクセルのメトリックで構成されます。予測の単位がイメージ全体の場合にはイメージのメトリックを、各ピクセルが個別の予測単位として扱われる場合にはピクセルのメトリックを使用します。どちらも、この手法の仕組みについての洞察を提供します。通常、イメージのメトリックは異常な物体を見つけるのがどの程度得意であるかを示し、ピクセルのメトリックは物体の異常な部分を見つけるのがどの程度得意であるかを示します。次のセクションで結果を説明します。

実験と結果

MVTec AD (英語) および VisA (英語) の 2 つのデータセットのアプローチを評価しました。MVTec AD は、MVTec とも呼ばれる標準的なビジュアル異常検出データセットで、15 の異なるカテゴリーで構成されます。例を図 9 に示します。物体は各イメージ内で適切に配置されていて、異常の規模はさまざまです。タイル・アンサンブルの利点を示すための最適な例ではありませんが、そういったケースでのパフォーマンスも示すために含まれています。

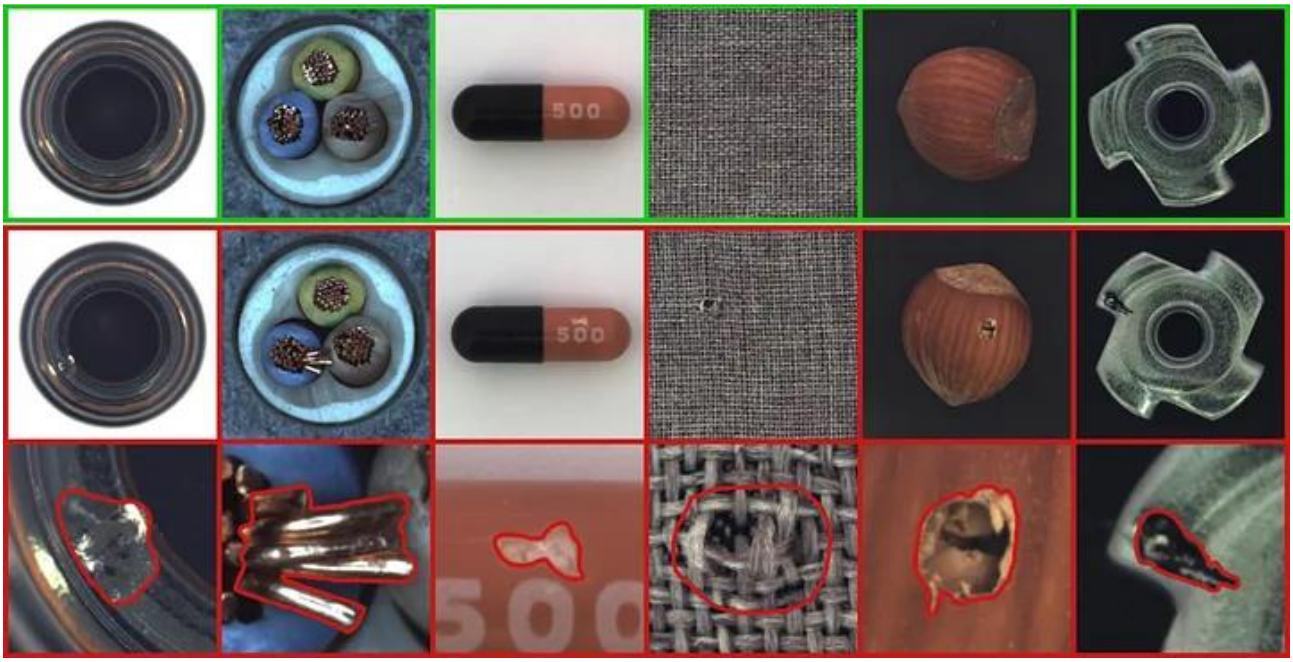

図 9: MVTec AD データセットの例。出典:
https://www.mvtec.com/company/research/datasets/mvtec-ad/ (英語)

一方、VisA データセットは少し複雑です。12 の異なるカテゴリーで構成され、同じイメージの物体の複数のインスタンスが含まれる場合もあります。不具合の多くは小さなもので、タイル・アンサンブルの利点を示すのに適しています。各カテゴリーの例を図 10 に示します。

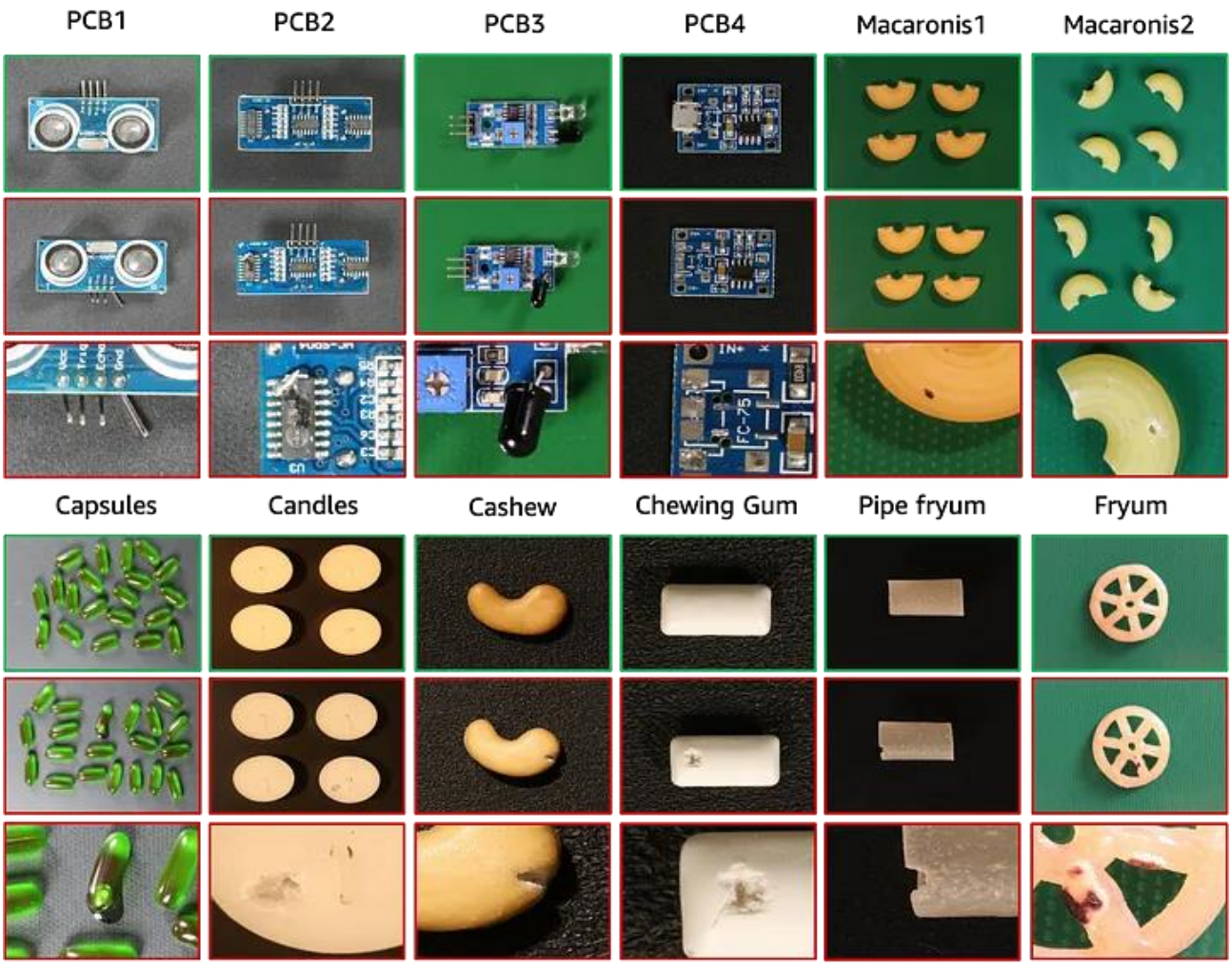

図 10: VisA データセットの例。出典: https://paperswithcode.com/dataset/visa (英語)

2 つの異なる解像度の単一モデルと比較して、タイル・アンサンブルを評価しました。タイル・アンサンブルのイメージのサイズは 512x512 ピクセルで、オーバーラップする 9 つのタイルに分割され、それぞれの解像度は 256x256、ストライドは 128 でした。各アーキテクチャーの単一モデルは、256x256 または 512x512 の入力サイズを使用してトレーニングされました。PatchCore はメモリー使用量が多いため、512x512 のモデルはトレーニングできなかったことに注意してください。ほかのすべてのパラメーターは各アプローチで同じでした。バックボーンとして resnet18、バッチサイズ 32、そして 42 に固定されたシードが含まれます。

5 つの異なるアーキテクチャーをトレーニングして、タイル・アンサンブルが異常検出の異なるアプローチでどのように機能するか確認しました。使用したモデルは、PaDiM (英語)、PatchCore (英語)、FastFlow (英語)、Reverse Distillation (英語)、STFPM (英語) です。

イメージレベルのメトリック (イメージが異常かどうかを決定) として AUROC を使用し、ピクセルレベルのメトリック (各ピクセルが異常かどうかを割り当て) として AUPRO を使用しました。これらのメトリックの選択は、異常検出に関する最新の論文とその特性に基づいたものです。AUPRO では小さな異常も大きな異常も同様に重要であることが保証されるため、ピクセルレベルの位置の特定には AUROC ではなく AUPRO を選択しました。

MVTec データセット全体で得られた結果を図 11 に示します。

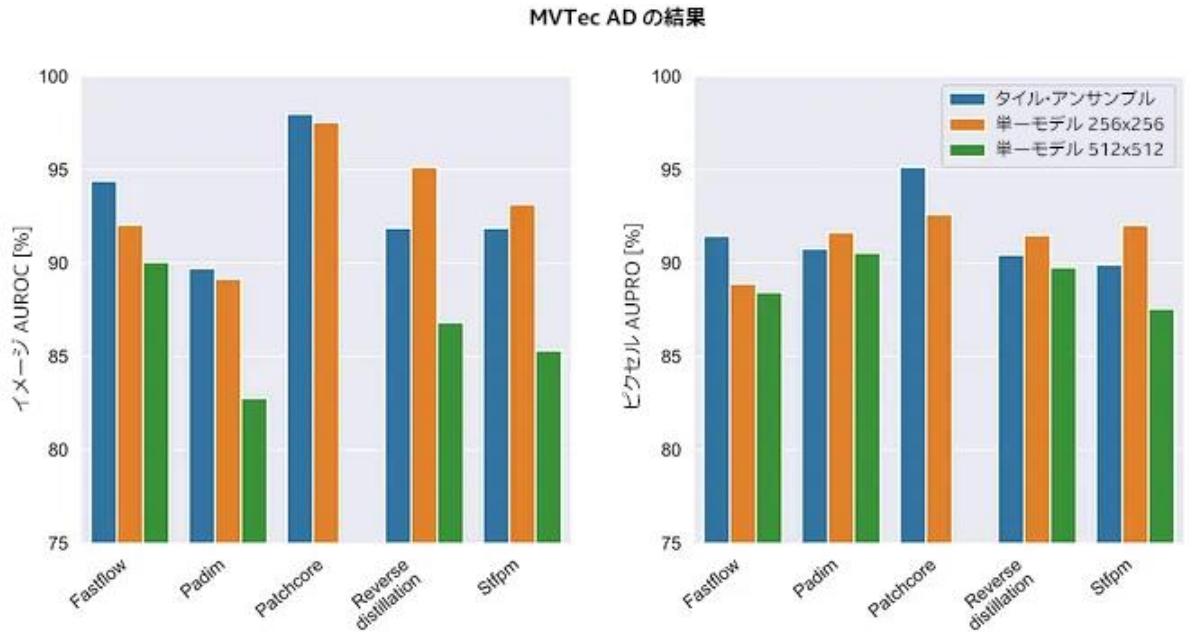

図 11: タイル・アンサンブルを使用すると一部のモデルのパフォーマンスは向上しますが、向上しない場合もあります。また、解像度を高くすればパフォーマンスが向上するとは限りません。

最初に分かることの 1 つは、タイル・アンサンブルを使用すると多くのケースでパフォーマンスが向上しますが、パフォーマンスが低下するケースもあることです。

また、解像度 512x512 の単一モデルのパフォーマンスは、アンサンブルや低い解像度の同じモデルよりも低いことも分かります。つまり、解像度を高くすればパフォーマンスが向上するとは限りません。これは、事前トレーニング済みのバックボーンの問題 (英語) が原因です。

MVTec は標準的なデータセットですが、タイル・アンサンブルの利点を示す最適なサンプルではありません。単一イメージ内に多くのオブジェクトが存在し、異常が非常に小さく複雑になる VisA データセットでは、この状況は一変します。結果を図 12 に示します。

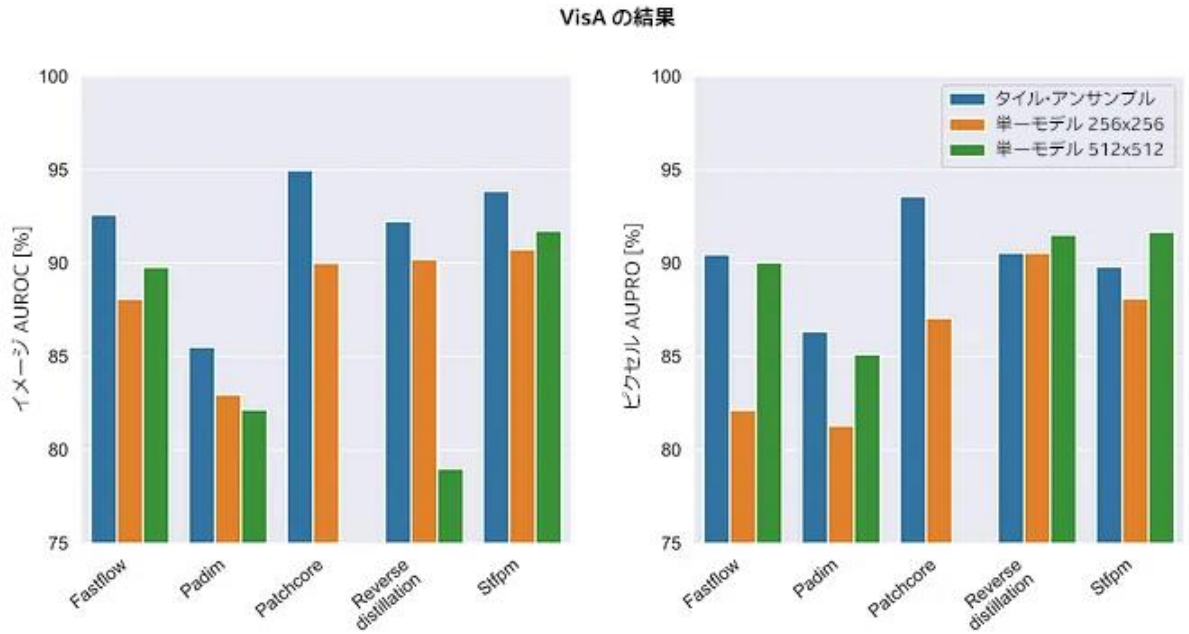

図 12: タイル・アンサンブルは、アンサンブルのタイルサイズと同じ入力サイズの単一モデルを上回る (または少なくとも同等の) パフォーマンスを示します。一部のケースでは、解像度を高くするとパフォーマンスが大幅に向上しています。タイル・アンサンブル・メカニズムを使用するとさらにパフォーマンスが向上するケースもあります。

入力サイズがアンサンブルのタイルサイズと一致する場合、タイル・アンサンブルは単一モデルを上回るパフォーマンス (少なくとも同等のパフォーマンス) を示しています。ここでも、パフォーマンスの向上はすべてのアーキテクチャーで同等ではないことが分かります。

もう 1 つ注目すべき点は、パフォーマンスの向上は解像度の向上のみによるものではないことです。解像度が高い単一モデルは、ピクセルレベルの Reverse Distillation と STFPM ではアンサンブルよりも優れていますが、イメージレベルの Reverse Distillation では検出パフォーマンスが大幅に低下しています。

大きな入力サイズでパフォーマンスが向上するのは良いことですが、タイル・アンサンブルは少ないメモリー使用量で高い解像度を処理できるように設計されていることを忘れないでください。アンサンブル内の個別のモデルで必要なメモリーの量は解像度 256x256 の単一モデルと同じであるため、メモリー使用量が多いために単一モデル 512x512 ではトレーニングできなかった Patchcore でも優れた結果が得られています。

メトリックから分かるように、タイル・アンサンブルには、一部のケースでは全体的なパフォーマンスが向上するだけでなく、小さな異常も検出できるという利点があります。この利点が活かされず、パフォーマンスが低下することもあります。これらの分かったことを説明するため、定性的な結果も示します。

図 13 は、タイル・アンサンブルが単一モデルよりも優れている MVTec Toothbrush カテゴリーの例を示しています。小さな異常の位置特定は適切に機能していますが、多少のノイズが発生しています。

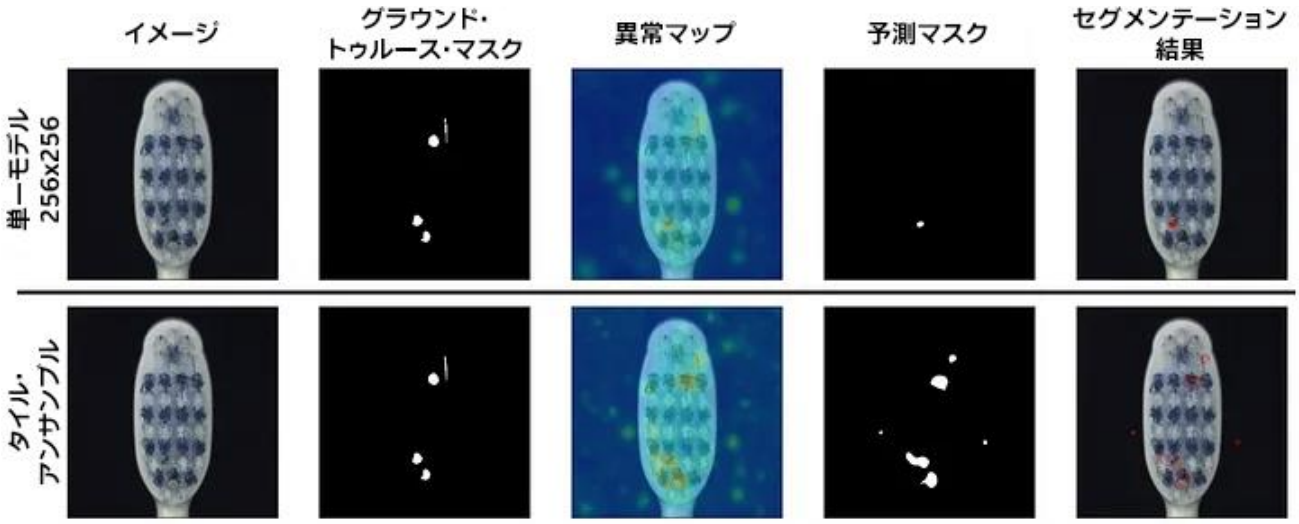

図 13: (PaDiM、MVTec Toothbrush) MVTec Toothbrush カテゴリーは、アンサンブルで最もパフォーマンスが向上しました。小さな不具合が適切に検出され位置が特定されています。

アンサンブルにより位置特定が向上するもう 1 つの例は、MVTec の Grid カテゴリーです (図 14 を参照)。

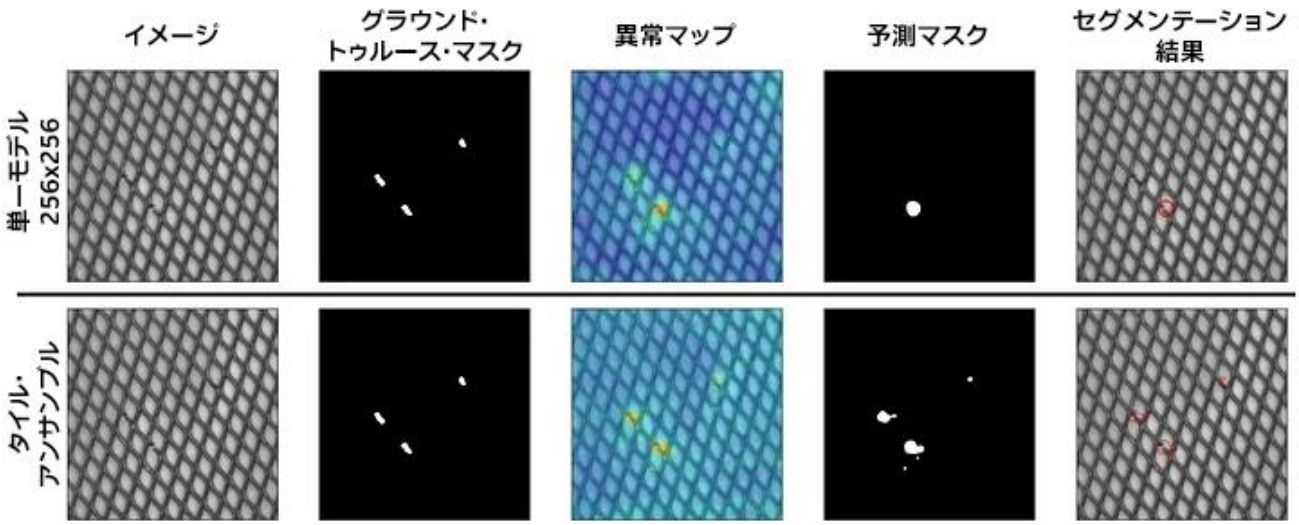

図 14: (PaDiM、MVTec Grid) 単一モデルでは見逃された異常がアンサンブルでは特定された別の例。

図 15 は、Toothbrush の別の例です。解像度を上げても、2 つの小さな異常の位置特定には役立っていません。

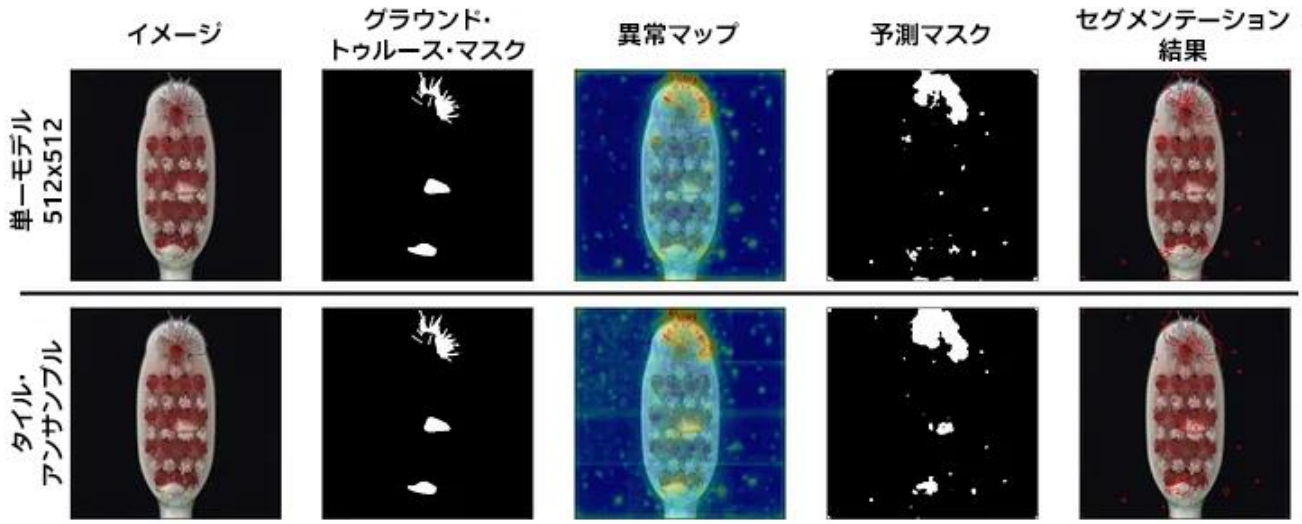

図 15: (FastFlow、MVTec Toothbrush) タイル・アンサンブルの改善は解像度の向上のみから得られるのではありません。

タイル・アンサンブルのパフォーマンスがすべてのアーキテクチャーで低下するケースもあります。最も顕著なのは MVTec の Cable カテゴリーです。図 16 から、位置特定のパフォーマンスが大幅に低下していることが分かります。

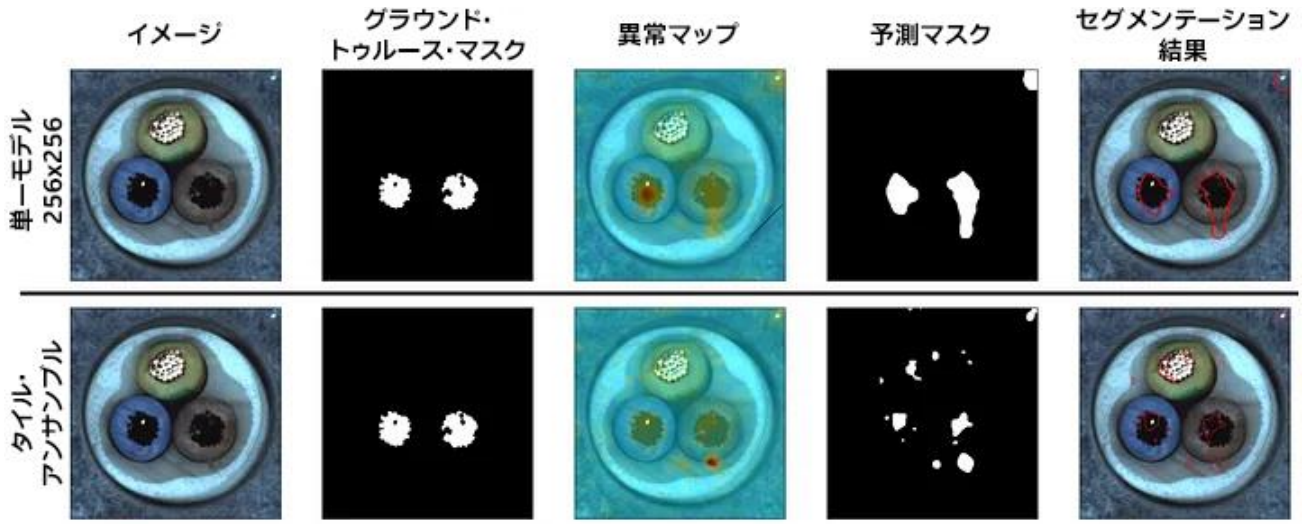

図 16: (PaDiM、MVTec Cable) アンサンブルの失敗ケース。MVTec Cable カテゴリーは全体的にパフォーマンスが低下しています。

前述したように、VisA には小さな異常が多く含まれています。このため、図 17 に示すように、アンサンブルでは位置を特定できる異常を単一モデルでは見逃すことがあります。

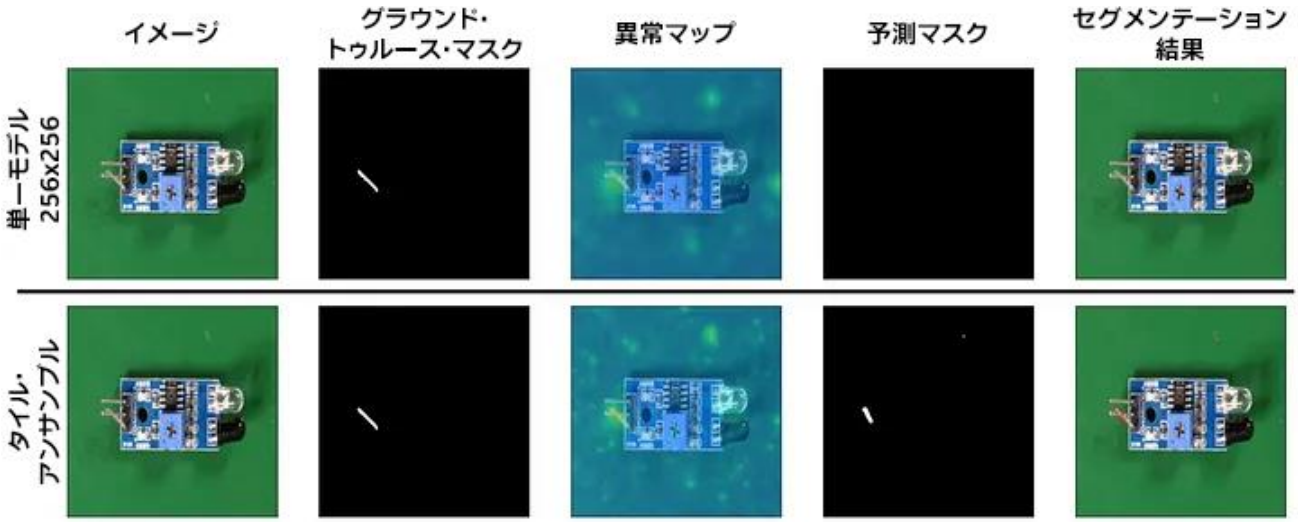

図 17: (PaDiM、VisA Pcb3) VisA データセットには単一モデルでは見逃してしまう小さな異常が多く含まれていますが、タイル・アンサンブルではそれらの異常を検出できます。

小さな異常の多くのケースでは、図 18 に示すような例もあります。

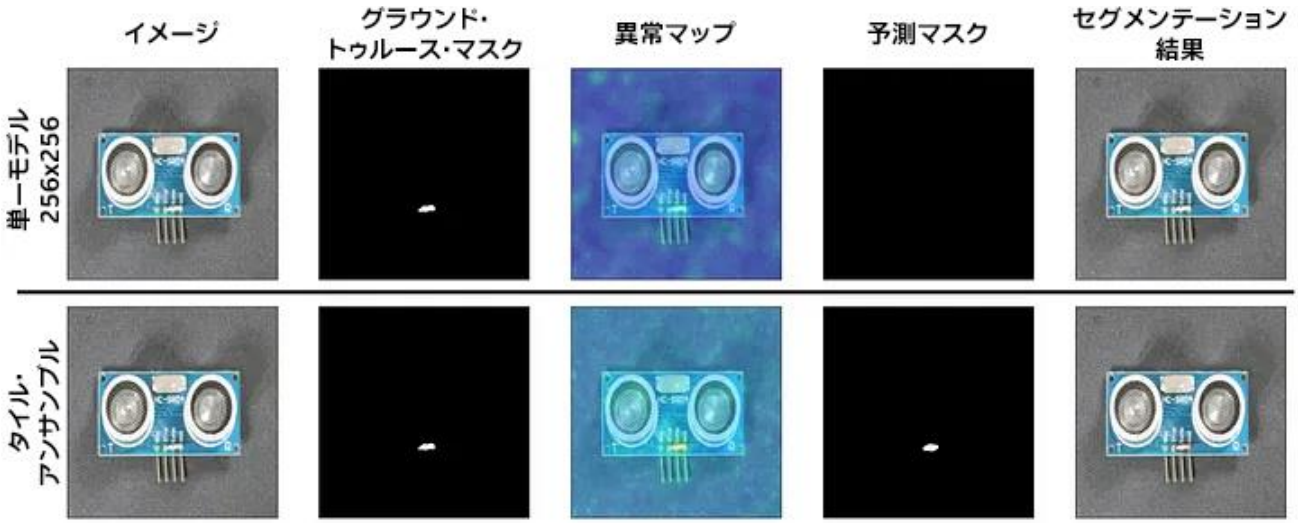

図 18: (Padim、VisA Pcb1) 単一モデルでは見逃された小さな異常がタイル・アンサンブルでは検出された別の例。

単一モデルの解像度を上げると、小さな異常を検出できるようになります。図 19 は、以前は見逃していた異常が、解像度を上げると検出された例を示しています。

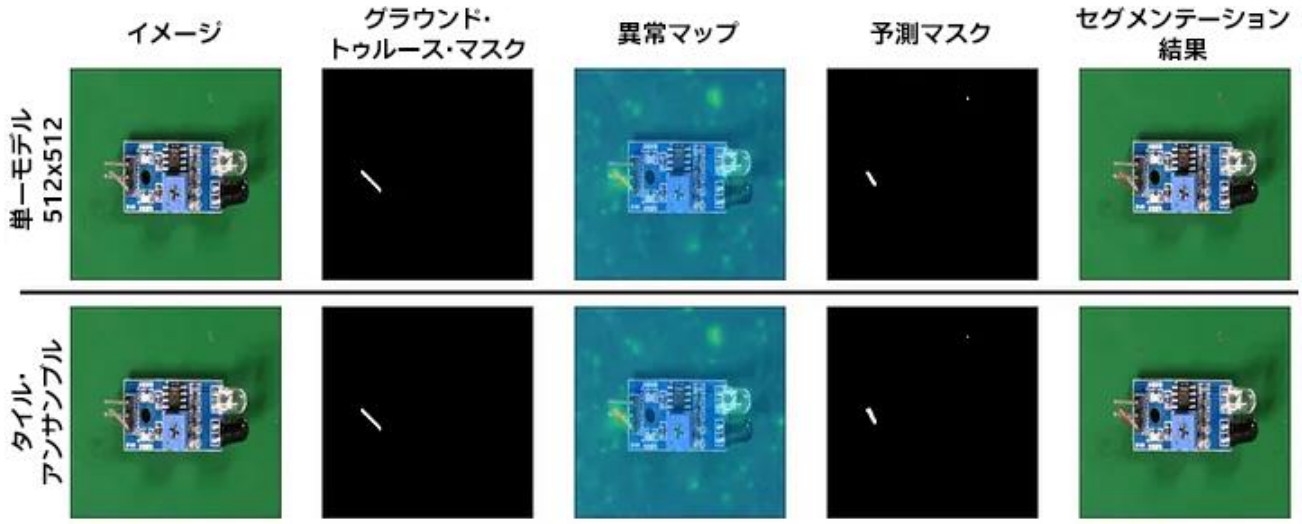

図 19: (PaDiM、VisA Pcb3) 単一モデルの解像度を上げると、タイル・アンサンブルのパフォーマンスと一致させることができます。

ただし、解像度を上げると、図 20 のように、予測に多くのノイズが含まれることがあります。

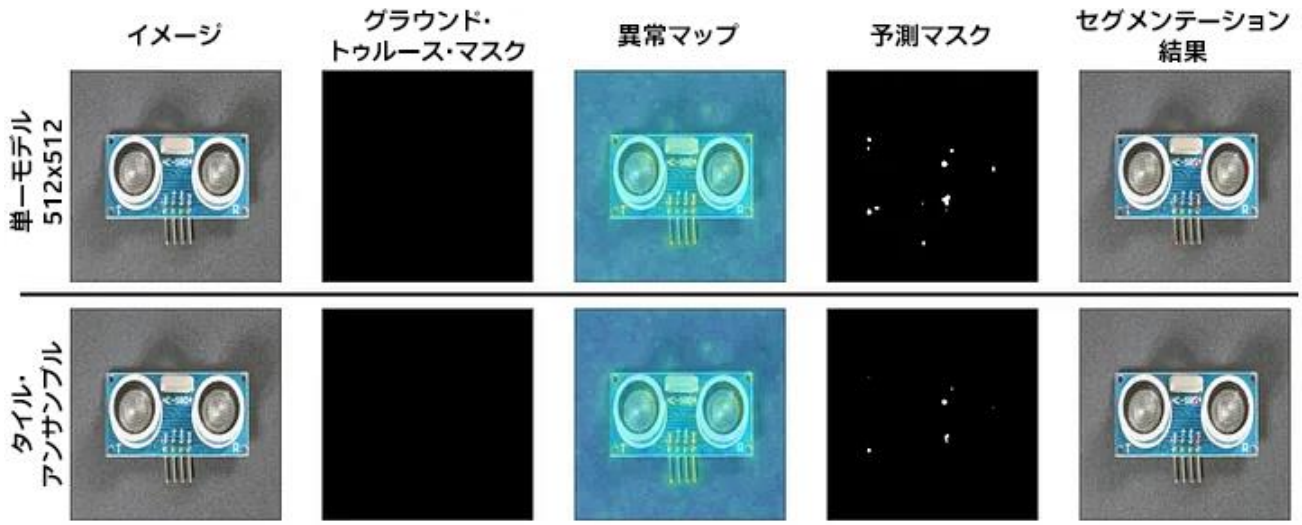

図 20: (PaDiM、VisA Pcb1) 解像度を上げてもタイル・アンサンブルのパフォーマンスと常に一致するとは限りません。

アンサンブルは、モデル固有の問題は修正できません。図 21 に示すように、同じ形状ではありませんが、ノイズはまだ存在しています。

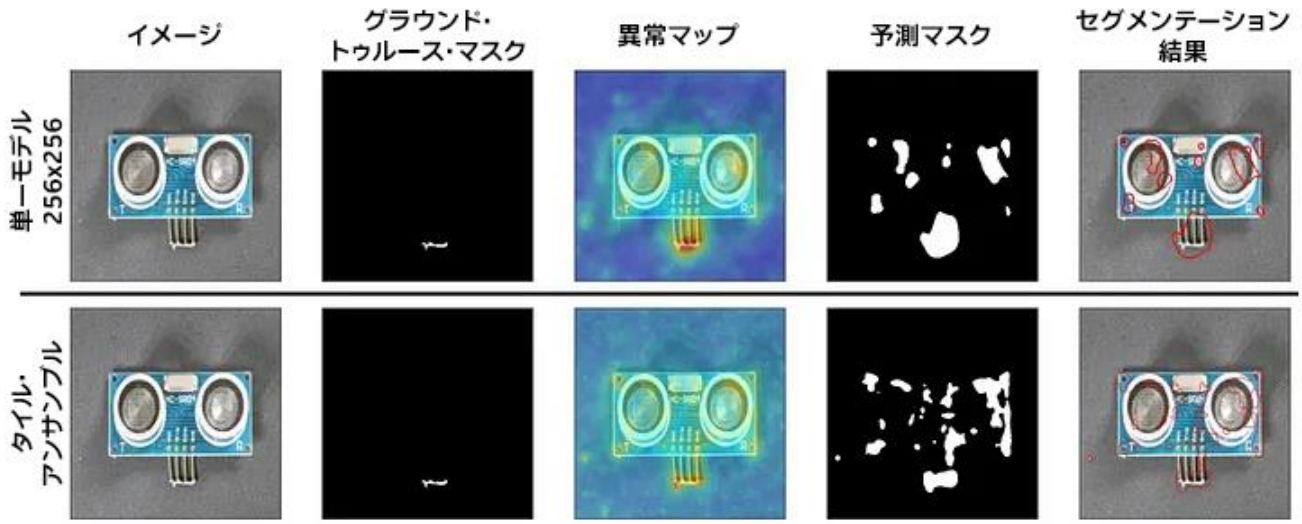

図 21: (PaDiM、VisA Pcb1) タイル・アンサンブルの失敗ケース。アーキテクチャー自体の欠点を修正していないため、タイル・アンサンブルには誤検出 (偽陽性) が存在します。

異常領域が見逃されるだけでなく、偽陽性の予測が追加されることもあります。その様子を図 22 に示します。

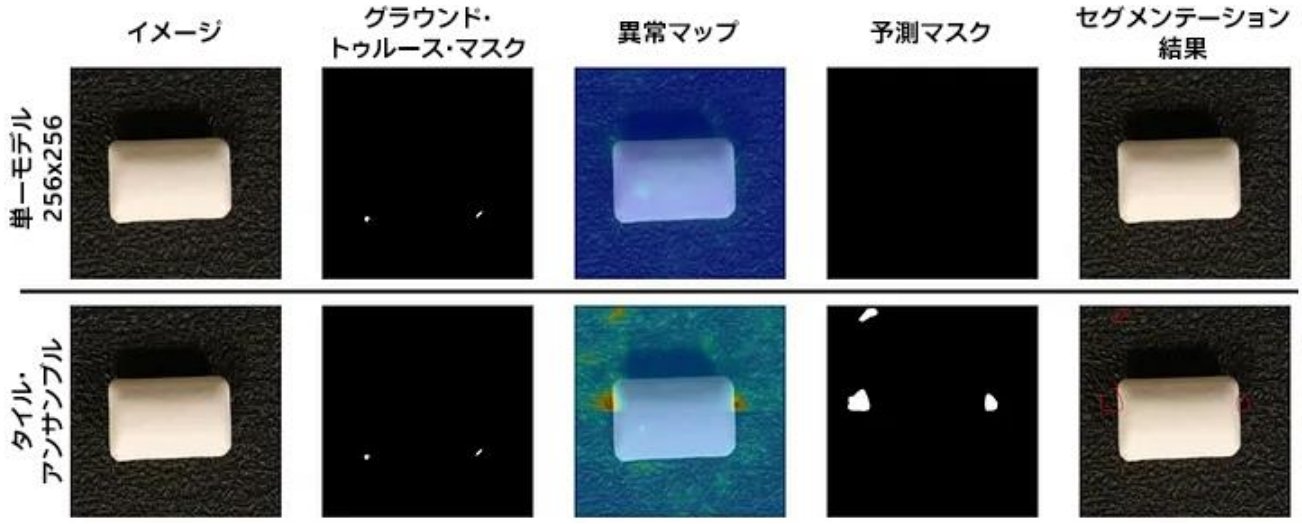

図 22: (Reverse Distillation、VisA Chewing gum) 単一モデルでは何も検出されませんが、タイル・アンサンブルでは偽陽性領域が生成されると予測されています。

まとめ

タイル・アンサンブルを使用すると、**メモリーの制約**を満たしつつ、**高解像度**のイメージの異常検出が可能になります。この手法は、イメージを小さなタイルに分割し、タイルの位置ごとに個別のモデルをトレーニングするもので、**OpenVINO™** ツールキット・チームの **Google Summer of Code 2023** プロジェクトの範囲内で **Anomalib** の一部として実装されました。

このアプローチは、メモリー要件を満たしつつ、研究および産業用アプリケーションの小さな不具合の位置特定における改善の可能性を示すものであり、既存および新しいアプローチの改善に新たな可能性をもたらすでしょう。

OpenVINO™ ツールキットとは

AI を加速する無償のツールである OpenVINO™ ツールキットは、インテルが無償で提供しているインテル製の CPU や GPU、VPU、FPGA などのパフォーマンスを最大限に活用して、コンピューター・ビジョン、画像関係をはじめ、自然言語処理や音声処理など、幅広いディープラーニング・モデルで推論を最適化し高速化する推論エンジン / ツールスイートです。

OpenVINO™ ツールキット・ページでは、ツールの概要、利用方法、導入事例、トレーニング、ツール・ダウンロードまでさまざまな情報を提供しています。ぜひ特設サイトにアクセスしてみてください。

https://www.intel.co.jp/content/www/jp/ja/internet-of-things/openvino-toolkit.html