**1. Anomaly Detection in Images**

* **What is Anomaly Detection?**  
  It’s the process of identifying unusual or abnormal images (or parts of images) that deviate from what’s considered “normal.” In industrial settings, for example, this could mean spotting defects in manufactured items.
* **The MVTec Dataset**  
  This is a benchmark dataset widely used in industrial anomaly detection. It contains images of various objects (e.g., bottles, cables, screws) where most images show defect‐free (normal) items and a few show defects (anomalies).

**2. The PatchCore Algorithm**

* **How PatchCore Works:**  
  PatchCore is an advanced anomaly detection algorithm that:
  + **Extracts Patch Features:** Uses a pre-trained CNN (like WideResNet50) to extract features from small patches of each image.
  + **Builds a Memory Bank:** Stores a representative set of patch features from only normal images.
  + **Detection via Nearest Neighbors:** At test time, for each patch in a new image, it compares its features to the memory bank. Patches that don’t match well (i.e., are “distant” in feature space) are flagged as anomalies.
* **Why PatchCore?**  
  It is known for high accuracy and fast inference times, which is why it’s popular for industrial anomaly detection tasks.

**3. Counterfactual Explanations**

* **What Are Counterfactual Explanations?**  
  They provide “what if” insights. For an anomalous image, a counterfactual explanation would show you a modified version of the image that is considered normal. In other words, it answers: “What minimal changes would have made this image normal?”
* **Example:**  
  Suppose PatchCore flags a defect on a bottle. A counterfactual explanation might slightly alter the region with the defect (e.g., “repair” the crack) and show that with this small change, the bottle would be recognized as normal. This helps human users understand which specific features or areas are driving the anomaly decision.

**4. Highly Imbalanced Datasets**

* **What Does “Highly Imbalanced” Mean?**  
  In MVTec Dataset, most images are normal and only a few are anomalous. This imbalance can make it challenging for models to learn what constitutes an anomaly.
* **Why It Matters:**  
  The scarcity of anomalies means the model might not see enough examples of defects to fully understand their variations. Counterfactual explanations here can provide additional insights by highlighting the subtle changes that distinguish normal from anomalous images.

**5. Project’s Goal**

* **Main Goal:**  
  Investigate how to generate and use counterfactual explanations in an anomaly detection pipeline based on PatchCore applied to the MVTec dataset.

**In Summary**

1. Use the MVTec dataset to train the PatchCore algorithm, which learns what normal images look like by storing features from image patches.
2. Apply PatchCore to detect anomalies in test images.
3. Develop a counterfactual explanation method that “repairs” the anomalous regions minimally, showing how the image would need to change to be considered normal.
4. Explore how well this approach works, particularly when anomalies are rare compared to normal images.

The **MVTec AD** dataset is a comprehensive benchmark specifically designed for anomaly detection in industrial inspection. Here’s an overview of its key characteristics and structure:

**What It Contains**

* **Image Types:**
  + **Objects and Textures:** The dataset is divided into 15 categories—5 textures (e.g., carpet, grid, leather, tile, wood) and 10 objects (e.g., bottle, cable, capsule, hazelnut, metal nut, pill, screw, toothbrush, transistor, zipper).
  + **High-Resolution Images:** Originally captured at 2048×2048 pixels, images are cropped to sizes typically between 700×700 and 1024×1024 pixels, ensuring fine detail is retained for defect detection.
* **Defects (Anomalies):**
  + **Variety:** There are over 70 distinct types of anomalies such as scratches, dents, contaminations, cracks, and structural deformations.
  + **Pixel-Level Ground Truth:** For every defective image in the test set, a pixel-precise annotation (mask) is provided that outlines the anomalous region. This is essential for both image-level classification and fine-grained segmentation tasks.

**How It Is Structured**

* **Dataset Splits:**
  + **Training Set:** Contains only defect-free ("good") images. This reflects the unsupervised learning scenario where models learn what “normal” looks like.
  + **Test Set:** Contains a mix of defect-free images and images with defects. For defective images, ground-truth masks are provided.
* **Directory Organization:**
  + The dataset is organized by category. For each category, you will find:
    - A **train** folder containing only good images.
    - A **test** folder that is further subdivided by defect types (and a “good” folder for non-defective samples).
  + This clear hierarchy makes it straightforward to load training samples and their corresponding test images along with anomaly masks.
* **Image Modalities:**
  + While most images are in color (RGB), some object categories are provided in grayscale. This is because in industrial settings, grayscale images are common.
* **Dataset Size:**
  + The dataset comprises a total of 5354 images, with 3629 images for training and 1725 for testing.

**Why It’s Useful**

* **Real-World Industrial Applications:**
  + The dataset is built to mimic real inspection scenarios where defects are rare and diverse.
* **Benchmark for Unsupervised Anomaly Detection:**
  + Since only normal images are available for training, models must learn a representation of “normality” and then flag deviations during testing.
* **Support for Multiple Tasks:**
  + Beyond anomaly detection (binary classification of images), the pixel-precise labels enable segmentation tasks, allowing researchers to localize anomalies very accurately.

**In Practice**

1. **Train your model** only on the “good” images from the training set.
2. **Test your model** on the test set, where you evaluate both whether it can detect that an image is anomalous and how well it can localize the anomaly via the provided masks.
3. **Use the structured directories** to easily load data according to class and defect type.

**PatchCore** is an unsupervised anomaly detection algorithm—especially popular in industrial visual inspection—that identifies defects by learning only from normal (defect‐free) images. It works by memorizing “normal” patch features from these images and then flagging any test image patches that significantly deviate from this learned normality.

Here’s a high-level overview of how it works:

1. **Feature Extraction:**  
   A pre-trained convolutional neural network (typically a ResNet or WideResNet) is used to extract mid-level features from each normal training image. Instead of using low-level (too generic) or high-level (too ImageNet-specific) features, PatchCore selects mid-level representations that capture meaningful local details.
2. **Memory Bank Construction:**  
   The image is divided into patches, and for each patch a feature vector is computed. These patch features are then stored in a memory bank to represent the normal distribution of features. To make the memory bank efficient (since it can be very large), PatchCore employs a greedy coreset subsampling strategy that selects a representative subset of these features without losing critical information.
3. **Anomaly Scoring at Test Time:**  
   When a test image is evaluated, the same patch extraction process is applied to compute its features. Each test patch is then compared—using a nearest neighbor search—to the features in the memory bank. If a test patch is very different (i.e., its nearest neighbor distance is high), it is likely to be anomalous. The overall anomaly score for the image can be taken as the maximum (or another aggregate) of these patch-level scores.
4. **Anomaly Localization:**  
   Since the method computes an anomaly score for each patch, these scores can be reassembled into a spatial map that localizes the defect(s) within the image.

**Growing Spheres**

https://github.com/thibaultlaugel/growingspheres/tree/master

https://hal.sorbonne-universite.fr/hal-01905982/file/180115\_final.pdf

Algorytm działa dla klasyfikacji binarnej. Np., mamy punkt x, dla uproszczenia w przestrzeni 2d. Może on być albo klasy "anomalia", albo klasy "nie anomalia". Rozważamy koło o środku w punkcie x i o promieniu r. Tworzymy pierścień wokół tego koła, ma on promień wewn. r i promień zewn. r+k. Generujemy losowo i równomiernie punkty w tym pierścieniu. Znajdujemy taki punkt, który ma inną klasę niż punkt x i jest najbliżej koła (najbardziej wewnątrz pierścienia). Jest to nasze counterfactual explanation (ce). Chodzi o to, by znaleźć (iteracyjnie) takie koło, żeby promień r był jak najmniejszy (wtedy ce będzie najbliżej x). Jako wynik algorytmu zwracamy wektor ce - x. Możemy jeszcze dodatkowo starać się zerować niektóre składowe ce - x, jeżeli tylko nie zmienią one klasy ce. Dzięki temu ce będzie zmieniało jak najmniej cech w porównaniu do x.

Algorytm skupia się bardziej na "decision boundary" klasyfikatora niż na tym jak powinien wyglądać obraz. "Growing Spheres is trying to understand the classifer decision, not the reality it is approximating". W oryginalnym kodzie źródłowym algorytmu widać wiele komentarzy typu TODO, a ostatnie commity były robione 3 lata temu. Wygląda to jakby algorytm został tak jakby porzucony w trakcie pracy i wymagał wciąż kilku poprawek i optymalizacji.

Algorytm udało się uruchomić na paru obrazach. Jeżeli jednak staramy się zerować niektóre składowe, to może on działać bardzo długo. W dodatku, ze względu na to jak jest skonstruowany, wydaje mi się, że nie jest odpowiedni do znajdowania ce dla wykrytych anomalii, jako że szansa na losowe znalezienie jakiegoś punktu, który odpowiada obrazowi bez anomalii, jest mała, mniejsza niż szansa na znalezienie losowego punktu, który odpowiada obrazowi z anomalią. Algorytmu można by więc ewentualnie użyć do znajdowania ce dla przeciwnych przypadków, tj. niewykrytych anomalii.

**SEDC-T**

https://github.com/ADMAntwerp/ImageCounterfactualExplanations

https://link.springer.com/article/10.1007/s10044-021-01055-y

Algorytm skupia się na zmianach, jakie należy zrobić w obrazie, żeby zmienić jego klasyfikację. Jest model-agnostic i został dostosowany specjalnie dla obrazów.

Mamy np. jakiś obraz zaklasyfikowany jako "anomalia". Dzielimy ten obraz na jakieś segmenty. Algorytm stara się znaleźć takie segmenty (albo kombinację segmentów), których brak zmieniłby klasyfikację na "brak anomalii". Segmenty (piksele), które są usuwane w celu testowania klasyfikacji można zastąpić na różne sposoby, np. średnią z obrazka, albo czymś bardziej wyrafinowanym, np. blur, albo inpaint.

Repozytorium zawiera czytelną implementację algorytmu i łatwy przykład pokazujący jak go używać. Algorytm udało się uruchomić w kontekście anomalib. Wygląda na to, że działa szybciej od poprzedniego, i że bardziej się nadaje do znajdowania ce dla przypadku wykrytych anomalii.

**FACE**

https://github.com/sharmapulkit/FACE-Feasible-Actionable-Counterfactual-Explanations

https://arxiv.org/pdf/1909.09369

Algorytm tworzy graph, gdzie wierzchołki reprezentują istniejące rekordy z datasetu, a krawędzie łączą podobne (bliskie sobie) rekordy. Krawędzie mają wagi, które oznaczają gęstość obszaru, w którym znajdują się rekordy - mała gęstość obszaru oznacza mniej rekordów, co z kolei oznacza, że rekordy tam się znajdujące to prawdopodobnie outliery. Algorytm stara się znaleźć counterfactual explanation za pomocą znalezienia najkrótszej ścieżki do jakiegoś wierzchołka, który należy do docelowej klasy i znajduje się w jakimś gęstym obszarze (co zapewnia, że jest realistyczny i nie jest outlierem). Algorytm jest model-agnostic i data-agnostic.