ChatGPT

Day 3, Session 3: Object Detection for Earth Observation Imagery

Learning Objectives

By the end of this session, you will be able to:

- **Define object detection** and distinguish it from image classification and segmentation in the context of computer vision.
- **Describe the output of object detection** (bounding boxes with class labels) and understand how it combines localization and classification.
- **Understand major object detection architectures:** including two-stage detectors (R-CNN family), single-stage detectors (YOLO, SSD, etc.), and transformer-based detectors (DETR), along with their pros and cons.
- **Interpret pseudocode or diagrams** illustrating how detectors like YOLO divide an image into grid cells to predict bounding boxes.
- **Identify key Earth Observation use cases** for object detection (e.g., detecting ships, buildings, vehicles, etc.) and relate them to disaster risk reduction (DRR) and natural resource management (NRM) applications.
- **Recognize challenges specific to EO object detection**, such as small object size, scale variation, arbitrary object orientation, cluttered backgrounds, and limited labeled data, and discuss possible mitigation strategies for each.

Module 1: Concept of Object Detection

Object Detection Defined: Object detection is the computer vision task of finding **multiple objects** in an image and classifying each one. Unlike image classification, which assigns one overall label to an entire image, object detection provides both *what* and *where* – it not only identifies the object classes present, but also **localizes each object with a bounding box** [1] [2]. This is also distinct from semantic segmentation, which labels every pixel in an image with a class (producing masks), but does not separately delineate individual object instances [3]. In summary:

- **Image Classification:** Determine a single category for the whole image (e.g., a satellite image patch is classified as “urban” vs “forest”). There is no location given – just presence or absence of a class in the image [3] [4].
- **Object Detection:** Identify and **localize each object** of interest in the image. The output is typically a set of bounding boxes with labels (e.g., find all cars in a parking lot and draw a box around each car) [5] [6]. This can be thought of as combining image classification with object localization.
- **Semantic Segmentation:** Assign a class label to **each pixel** in the image, resulting in regions or masks (e.g., color all ship pixels vs water vs land in different colors). This yields precise outlines but usually does not distinguish separate instances of the same class (instance segmentation is a further extension that does) [7].

Example – Why Detection? Imagine a high-resolution satellite image of a port. An image classification model might simply tell us "Port" (since ships and port infrastructure are present) but would not indicate where or how many ships. A semantic segmentation model could color-code water vs land vs ship pixels, showing the extent of water and ships, but it might merge all ships into one combined region. **Object detection**, on the other hand, would output a list of each ship's location with a bounding box and the label "ship" [8]. This is critical for applications like counting ships or monitoring their positions. For instance, object detection would enable us to count how many ships are in the harbor and pinpoint each ship's coordinates, which is not directly possible from classification or coarse segmentation alone. In general, whenever we need to **count objects** or know their exact positions on an image (e.g. "find all buildings in this area and give their locations"), object detection is the appropriate tool.

Think-Through: Given an overhead image of a city, what different insights would you gain from classification vs. object detection vs. segmentation? Consider how each approach would represent features like buildings or roads, and why you might choose detection if you needed to count or locate specific structures.

Mini-Challenge: For each scenario below, decide which computer vision task is most suitable (classification, detection, or segmentation): 1. **Estimating urban area extent:** Determining if an image tile is urban or rural.
2. **Counting trees in a plantation:** Finding and tallying individual tree crowns in an orchard image.
3. **Mapping flood extent:** Outlining the exact flooded regions in a satellite image after a hurricane.
4. **Identifying vehicles in a parking lot:** Listing each vehicle present and where it is located.

(Think about whether each scenario requires per-image labeling, locating multiple instances, or pixel-level delineation. Answers: 1–Classification, 2–Detection, 3–Segmentation, 4–Detection.)

Module 2: Overview of Object Detection Architectures

Modern object detection models generally fall into a few broad categories. We introduce three major families: **two-stage detectors**, **single-stage detectors**, and **transformer-based detectors**. Each approaches the task differently:

Two-Stage Detectors (Region-Proposal Based)

Two-stage detectors first generate possible object regions, then classify those regions in a second step [9] [10]. They tend to prioritize accuracy over speed. The seminal examples are the **R-CNN family**:

- **R-CNN (2014):** Region-based CNN. It uses an external algorithm (e.g. Selective Search) to propose ~2000 candidate regions in the image that might contain objects. Then it feeds each region through a CNN for classification (and bounding box refinement), using a separate classifier (like an SVM) for the final label [11]. R-CNN proved the concept of CNNs for detection but was very slow, since it had to run a forward pass for every region proposal.
- **Fast R-CNN (2015):** An improvement that processes the **whole image with a CNN once** to produce a feature map, then applies Region of Interest (ROI) pooling to extract features for each proposed region [12]. Instead of classifying raw pixels for each region, it classifies the region's features. This significantly reduced duplicate computation, making it much faster than R-CNN [13].

- **Faster R-CNN (2015):** Further improves speed by learning the region proposals. It introduces a small sub-network called the **Region Proposal Network (RPN)** that is attached to the CNN's feature map and proposes object regions (essentially replacing Selective Search with a learned method) [14]. The proposals are then refined and classified by the second stage. Faster R-CNN is an end-to-end trainable detector and became a benchmark for accuracy.
- **Mask R-CNN (2017):** Extends Faster R-CNN by adding a **third branch** that outputs a segmentation mask for each detected object [15]. This gives instance segmentation (outlines of each object) in addition to bounding boxes. Mask R-CNN is beyond pure detection, but it's worth mentioning as it builds on the two-stage paradigm.

Two-stage detectors are known for **high accuracy**, especially in **difficult scenarios with small or overlapping objects**, because the second stage can scrutinize each candidate region carefully [16] [17]. The trade-off is typically **slower inference**, since hundreds of proposals per image must be processed. For example, Faster R-CNN might evaluate 100–300 regions for a single image [18]. In Earth Observation, where we may have many small objects (cars in a city, trees in a forest), two-stage detectors can be beneficial when we need to maximize detection completeness and precision. If counting every object matters more than real-time speed, a two-stage approach is often chosen.

Single-Stage Detectors (Regression/Localization in One Shot)

Single-stage detectors **skip the region proposal step** and directly predict bounding boxes and class labels in one evaluation of the network [17]. They treat detection as a **dense regression problem** over spatial locations. These models are designed for **speed** and simplicity:

- **YOLO (You Only Look Once):** A pioneering single-shot detector introduced in 2016. YOLO v1 divides the image into an $S \times S$ grid and for each grid cell directly predicts a fixed number of bounding boxes (with confidence scores and class probabilities) in that cell [19]. In essence, YOLO produces detections in **one pass** through a CNN, achieving very high speed – often real-time on videos [20]. Over the years, YOLOv2, v3, … v5 have improved accuracy greatly, to the point of rivalling two-stage detectors on many benchmarks, while retaining fast inference. YOLO's design trades a bit of absolute accuracy for **dramatic gains in speed and simplicity**.
- **SSD (Single Shot MultiBox Detector):** Another one-shot detector (2016) that introduces the idea of using **multiple feature map scales** for detection [21]. SSD generates predictions from several layers of the CNN – early layers (coarser grids) handle large objects, while later layers (finer grids) handle smaller objects. SSD also uses predefined default "anchor" boxes of different sizes/aspect ratios at each location, and the network learns adjustments to these. This multi-scale approach helps SSD detect objects of varying sizes better than a single-scale method.
- **RetinaNet (2017):** A single-stage detector notable for introducing **Focal Loss** to address the extreme class imbalance between background and objects in one-stage training [22]. By down-weighting easy negative examples, Focal Loss helps the model focus on harder, relevant examples, boosting accuracy. RetinaNet showed that with the right loss function, single-stage detectors can approach two-stage accuracy.

Single-stage detectors are **faster** because they don't need a second stage – the entire image is processed and detections are output in one go [23]. This makes them well-suited for real-time or large-scale processing of many images. The historical downside is that they can miss some small or densely packed objects that two-stage detectors catch, since they rely on coarser grids or anchors (however, techniques like feature pyramids and improved losses have narrowed this gap). In many practical EO scenarios, one-stage models (like YOLO) are popular due to their speed and adequate accuracy – especially if deploying in an application that needs quick results (e.g., an emergency response system scanning new satellite images for damaged buildings might favor speed).

Think-Through: If you needed to detect vehicles in 1,000 satellite images as fast as possible, which type of detector (two-stage vs one-stage) might you choose and why? On the other hand, if you needed the most accurate detection of every tree in a high-resolution aerial image (and speed was less important), what approach could be better? Consider the strengths of each architecture in terms of precision and inference time.

Mini-Challenge: Match each description to the correct detection approach: - *"Uses a Region Proposal Network to suggest object regions, then classifies each region's content"* – is this more like Faster R-CNN or YOLO?
- *"Dense prediction of objects on a grid, no separate proposal step; typically very fast"* – which family does this refer to?
- *"Employs a Transformer to output a set of object boxes without relying on predefined anchors"* – which modern detector does this describe?
- *"Introduces Focal Loss to improve one-stage detector accuracy"* – which model is known for this innovation?

(Expected answers: Faster R-CNN; Single-stage (e.g. YOLO/SSD); DETR; RetinaNet.)

Transformer-Based Detectors (DETR and Variants)

The newest wave of detectors uses Transformers and attention mechanisms to remove some of the hand-crafted components of earlier models. **DETR (Detection Transformer)**, proposed by Facebook AI in 2020, is the flagship example. It formulates object detection as a **direct set prediction** problem, handled end-to-end by a Transformer architecture [24] [25].

Key ideas of DETR and similar models: - A CNN backbone first extracts a feature map from the image (as usual). Then, **Transformer encoder-decoder** layers operate on these features, treating detection as a sequence-to-sequence task where the output is a set of object predictions. - DETR outputs a fixed number of prediction "slots" (e.g., 100 possible objects per image). Each slot is a bounding box + class prediction (or "no object" for empty slots). During training, a **bipartite matching loss** (Hungarian algorithm) is used to uniquely match predicted boxes to ground-truth boxes [26]. This set-based global loss forces the network to produce one prediction for each actual object, and handle duplicates by itself. - **No need for anchor boxes or NMS:** Unlike traditional detectors, DETR does **not use pre-defined anchor boxes** on a grid, and it does **not require Non-Maximum Suppression** to remove duplicate detections [27] [25]. The model learns to coordinate the predictions such that each object is covered by one prediction slot, thanks to the set loss. This greatly simplifies the post-processing. - The Transformer's self-attention mechanism allows the model to capture **global context** – each predicted box can attend to relevant features anywhere in the image. In cluttered scenes or when objects have relationships, this global reasoning is beneficial [28]. - DETR achieved accuracy on par with strong CNN-based detectors, but one limitation noted was **slow training convergence** (it required a very large number of training epochs to learn the Hungarian matching effectively) [29] [30]. Follow-up works like *Deformable DETR* introduced improvements to speed up training and handle scale variation better.

For our purposes, the importance of DETR is that it shows a new paradigm: using Transformer networks to predict objects **without many of the heuristics** that two-stage or one-stage detectors need. There are no explicit region proposals, anchor boxes, or NMS – the model learns these internally. This is promising for Earth Observation, because Transformer-based detectors could naturally handle cases like **arbitrarily oriented objects and dense scenes** by learning the appropriate attention patterns [31]. For example, DETR could potentially learn to detect objects at any angle (since it's not tied to an axis-aligned grid of anchors) and might capture long-range context (e.g., a ship's presence might be easier to confirm if the model attends to the wake behind it or nearby ships).

It's worth noting that as of now (2025), **transformer detectors are still evolving**. They are not yet as widely used in practice as YOLO or Faster R-CNN for everyday applications, partly due to training complexity. But they represent where the field is heading, and some recent research has applied DETR-style models to aerial images with success. In this training, we focus on the well-established CNN-based detectors, but keep in mind this transformer approach for the future.

Think-Through: Why might removing the need for anchor boxes and NMS be especially useful for detecting objects in satellite images? (Hint: Think about objects that can appear at random sizes or very close together. A set-prediction transformer can in theory handle overlapping objects without needing a manual suppression step.) What could be some challenges of using a DETR model for EO data? (Hint: volume of training data, training time, etc.)

Mini-Challenge: DETR treats detection as a set prediction problem with a fixed number of output "slots." Suppose you set DETR to output 50 slots. What do you think happens if an image actually contains 60 objects? What about if it contains only 5 objects? (This is a thought experiment – consider how unmatched slots are handled.) Also, can you recall one major difference in how DETR is trained versus a traditional detector? *(Consider the bipartite matching step.)*

Module 3: EO Applications of Object Detection

Object detection is extremely useful in Earth Observation (EO) whenever we need to **count objects, monitor specific features, or extract the locations of certain classes** from imagery. Here are some prominent use cases and examples in the EO domain:

- **Vehicle and Aircraft Detection:** Locating cars, trucks, or airplanes in aerial or satellite images. For example, counting cars in a parking lot or along a road can indicate traffic volume or, in a disaster, how many vehicles are present in an evacuation zone. Detecting aircraft on airport tarmacs can help monitor airport activity or military airbase utilization [32]. This has security and intelligence applications, as well as urban planning uses (e.g., parking utilization studies).
- **Ship Detection:** Identifying ships in ports, coastal waters, or open ocean. This is useful for maritime traffic monitoring, detecting illegal fishing or piracy, border security, and rescue operations at sea [33]. Synthetic Aperture Radar (SAR) imagery is often used here as well (since it can see ships at night or through clouds), and object detectors can be trained on SAR images to pick out vessels.
- **Building Detection:** Finding buildings and structures in high-resolution imagery. This can support automated mapping of human settlements, estimating population (by counting dwellings when other data are scarce), and rapid damage assessment after disasters (e.g., detecting collapsed buildings after an earthquake) [34] [35]. Automated building footprint detection is a key task for humanitarian mapping and urban development monitoring.

- **Infrastructure Mapping:** Detecting specific man-made structures like **oil and gas storage tanks**, **wind turbines**, **solar panel farms**, **communication towers**, etc. [36] These features are important for economic monitoring and asset management. For example, counting oil storage tanks globally can give insight into oil inventories, and detecting wind turbines can support renewable energy assessments.
- **Wildlife and Environmental Monitoring:** In some cases, object detection is used to detect large animals in aerial imagery (e.g., counting wildlife from drone images). Also, detecting features like individual **trees** (treating tree canopies as objects) can allow estimation of tree counts and health in high-res images [37] . While land cover segmentation is common for forest monitoring, object detection can complement it for counting discrete entities (like singular trees).
- **Informal Settlements Detection:** In a development or humanitarian context, detectors can be trained to find small, densely-packed structures such as shanty houses in informal settlements [38] . This helps in mapping and tracking the growth of such settlements, which might be missed or merged in coarse segmentation maps.

Example: Overhead imagery with many objects can be analyzed via object detection. In the image above (sample from the xView dataset), each **blue box** corresponds to a detected object (vehicles, buildings, etc.) in a complex scene. A model has automatically identified and localized numerous objects across this large area. Object detection outputs like this can be integrated into GIS systems – for instance, each detection (with its coordinates and class) can be mapped, counted, or tracked over time.

The **xView dataset** is a good example showcasing EO object detection: it contains high-resolution satellite images with **over 1 million labeled object instances across 60 classes**, from cars and trucks to buildings and ships [39] [40] . Such datasets demonstrate the breadth of objects one might want to detect for disaster response or security (indeed, xView's ontology was informed by defense and disaster response needs). For example, in xView there are classes for "Damaged building" vs "Undamaged building," which can help algorithms automatically assess destruction by comparing object detections pre- and post-disaster.

Another real-world example: during the COVID-19 pandemic, researchers used satellite-based car detection in places like shopping center parking lots to **infer human activity levels**. If object detection finds very few cars in typically busy locations, it can indicate compliance with lockdowns or reduced economic activity. Conversely, an increase in car detections over time could signal recovery. This highlights how detection outputs (counts of objects) become valuable indicators for broader phenomena.

Think-Through: Consider the task of monitoring illegal mining using satellite images. How could object detection be used? (Hint: detectors could find telltale objects like **excavators, trucks, or mining pits**.) Also, for disaster response, if you wanted to use imagery to estimate how many people might need aid, what objects might you detect as a proxy (e.g., houses, shelters, vehicles)? Think about how detecting those objects and counting them can provide critical information for responders.

Mini-Challenge: Pick one of the EO applications above (vehicles, ships, buildings, etc.). Describe a scenario where detecting those objects over time could yield insight. For instance, *"Detecting ships in a harbor every week could help measure import/export activity by counting cargo ships over time."* Try to formulate a similar insight for your chosen object: what trend or event could you monitor by tracking the number or location of these objects in imagery?

(Open-ended answer, but example responses could include: detecting vehicles on roads daily to monitor traffic congestion patterns; detecting buildings after an earthquake to quickly find which structures are destroyed; detecting solar panels across regions to estimate solar energy adoption, etc.)

Module 4: Key Challenges in EO Object Detection

Deploying object detection on satellite or aerial imagery comes with **unique challenges** not always present in natural image datasets like COCO. We outline some key issues and briefly mention strategies to address them:

- **Small Objects:** In EO, targets like cars, small boats, or houses may appear in only a few pixels (especially in lower-resolution or wide-area imagery). Detectors tend to struggle when objects are very small relative to the image [41], because the CNN may not extract enough distinctive features from such tiny regions. For example, a vehicle that spans 5x5 pixels is hard to differentiate from random clutter. **Mitigation strategies** include using higher-resolution imagery or sensors (e.g., 30 cm/pixel data instead of 1 m/pixel), employing **feature pyramid networks** or multi-scale feature layers so that even small objects get detected on finer feature maps [42], and applying **super-resolution** or zooming in on image tiles so that small objects become larger in the input. Specialized training tricks like *focal loss* (as in RetinaNet) also help by focusing learning on these hard-to-detect examples [43]. Nonetheless, small object detection remains an active research area – practitioners often need to experiment with these approaches for best results.
- **Scale Variation:** A single satellite image can contain both very large objects (a jumbo jet, a large building) and very small ones (cars, streetlights). The range of scales is much broader than in typical photographs. Detectors must handle this variation – a one-size-fits-all detection resolution might miss one end of the scale. **Mitigation:** Multi-scale feature approaches (like SSD's multiple layers or a Feature Pyramid Network in Faster R-CNN) explicitly address scale by giving the model detectors at different resolutions [42] [44]. Also, feeding images at multiple scales or using test-time augmentations (pyramids) can catch different object sizes. Still, extremely large scale differences pose a challenge – one might need to run detection in a tiling strategy for very high-resolution data (processing large objects in full image, but dividing the image into tiles to zoom in on small ones).
- **Orientation and Rotation:** Unlike ground-level photos, **overhead objects can appear in any orientation** in the image – there is no canonical "up" direction for a car or building in a nadir (top-down) view [45]. A car might point north-south in one image and east-west in another; a ship could be diagonal across the image. Standard detectors typically use **axis-aligned bounding boxes** (straight rectangles aligned to image x/y axes), which can be inefficient for rotated objects. For example, a diagonal ship will be tightly enclosed by a rotated box, but an axis-aligned box has to cover a lot of background to include the whole ship. This leads to overlapping detections and wasted area. One line of work in EO detection is designing models that output **rotated bounding boxes** (often parameterized by an angle) to better fit angled objects [46] [47]. Rotated detectors (like in the DOTA dataset challenge) can more accurately encompass objects and allow measuring their lengths/widths directly. Another approach is to make the features or model **rotation-invariant** – e.g., apply random rotations to training images (data augmentation) so the model learns to detect objects regardless of orientation [48]. Indeed, many EO detection workflows include **rotating the training images** by random angles as a simple way to improve orientation robustness. The image below illustrates the difference that rotation can make:

Above: Comparison of detection bounding boxes without and with rotation for objects in an aerial image. The left panel shows **horizontal (axis-aligned) boxes** around ships in a harbor, while the right panel shows **rotated boxes** aligned with each ship's orientation (yellow boxes). Rotated bounding boxes capture the ships' extents much more tightly and avoid overlapping multiple ships in one box. This enables more accurate object localization and size estimation [49] [47]. In practice, using rotated boxes requires more complex loss functions and annotations, but it greatly helps when object orientation varies widely.

- **Complex & Cluttered Backgrounds:** EO images, especially over **dense urban areas**, are highly textured and cluttered. A detector looking for a "building" in a city has a background that is essentially *other buildings* – a crowded downtown may present thousands of adjacent structures. Similarly, a forest scene might have tree crowns packed edge-to-edge. This makes distinguishing individual instances hard (lots of overlapping or touching objects) and can cause many **false positives** (background patterns that look object-like) [50] [51]. For example, a patterned rooftop might confuse a vehicle detector, or shadows between buildings might be mistaken for objects. **Mitigation:** High-quality training data that includes many examples of such clutter is important, so the model learns the subtle differences. Increasing model capacity (deeper networks) can help memorize fine details. Another technique is to incorporate **context modeling** – e.g., algorithms that learn that if you detect one car, there are likely others nearby on a highway (so it should be less strict about overlap in a traffic jam, or conversely, if something car-shaped appears on a rooftop, it's probably not actually a car). Some advanced approaches post-process detections using spatial context rules to reduce obvious false positives (for instance, removing "car" detections that are isolated on building tops).
- **Atmospheric/Sensor Effects:** Satellite images can have issues like varying lighting, shadows, haze, or sensor noise. For instance, a plane under a cloud shadow might become invisible to an optical detector; or in radar imagery, speckle noise and distortion can create bright spots that mimic objects [52] [53]. These factors are not present in typical ground-photo datasets. **Mitigation:** Pre-processing the imagery can make a big difference (e.g., shadow removal or haze reduction techniques). Also, training data should include these effects if possible – e.g., train on images with different sun angles, cloud conditions, etc., so the detector is less fooled by shadows or glare. Data augmentation can simulate some effects (random brightness changes to mimic different lighting). In some cases, using multi-temporal data helps: if you have two images of the same area from different times, an object persistent in both but a cloud only in one, algorithms can learn to be skeptical of transient artifacts.
- **Limited Labeled Data:** Curating large labeled datasets for detection in EO is labor-intensive and expensive. It requires skilled annotators (who can interpret satellite images) drawing thousands of boxes. Often, we have to work with **small training sets** for specific tasks. This makes it hard to train detection models from scratch – they might overfit or just not generalize well [54] [55]. **Mitigation:** A common practice is **transfer learning** – using models pre-trained on big datasets like COCO or ImageNet and fine-tuning them on the EO task [56]. Even though COCO is natural images, the CNN has learned general features (edges, shapes) that transfer surprisingly well to overhead images [57]. Fine-tuning a pre-trained YOLO or Faster R-CNN on a small set of satellite annotations can yield decent performance. Another strategy is **data augmentation** to artificially enlarge the training set (random crops, flips, rotations, color jitter, etc., appropriate to EO data). Research into **few-shot learning and domain adaptation** is ongoing to help do more with less data. Additionally, leveraging unlabeled data via semi-supervised learning or synthetically generated training data are emerging approaches in EO.

Finally, note that **bespoke model tweaks** are common in cutting-edge EO object detection research. For example, some researchers incorporate the fact that objects often appear in groups: detecting one ship in open ocean might raise the expectation of others nearby if it's a shipping lane (contextual reasoning). Others develop models that are **rotation-equivariant** (the model's feature maps themselves don't change when the image is rotated, which can help with orientation issues). We also see use of **oriented bounding box (OBB) detectors** in competitions like DOTA: these are variants of Faster R-CNN or YOLO that can predict an angle for each box [58] [59].

In summary, the algorithms we learned (whether two-stage or one-stage) **largely carry over to EO imagery**, but to apply them effectively we often need to **adjust for these challenges** – through data (e.g., higher resolution, augmentation), model outputs (rotated boxes), or training techniques (transfer learning). Being aware of these challenges will help you troubleshoot and improve object detection models on geospatial data.

Think-Through: Among the challenges listed, which do you suspect is the **biggest hurdle** for detecting small boats (~5 m long) in 1.5 m resolution satellite images? What combination of strategies might you use to improve a boat detector in that scenario? Consider resolution (each boat ~3-4 pixels long!), model choice, and augmentation. Also, how would the approach differ if you were detecting **airplanes (~50 m)** in the same images (larger, easier to see – maybe small-object methods aren't as needed)?

Mini-Challenge: Propose one possible solution or mitigation for each of these challenges: - *Small objects:* e.g., "…"
- *Oriented objects:* e.g., "…"
- *Cluttered scenes:* e.g., "…"
- *Limited data:* e.g., "…"

Try to fill in each blank with a technique (it could be one mentioned above or a creative idea). For instance, for small objects you might say *"Use higher-resolution source imagery or zoomed-in tiling to ensure objects have more pixels"*. For oriented objects, *"Allow the model to predict rotated bounding boxes"*, and so on. Discuss why your proposed idea would help in that scenario.

---

[1] [3] [5] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [42] [43] [44] [45] [46] [48] [50] [51] [52] [53] [54] [55] [56] [57] Day 3_ Advanced Deep Learning – Semantic Segmentation & Object Detection.pdf
file://file_000000001fd861f5933b737832065f78

[2] [4] [6] Difference between Image Classification, Object Detection, and Image Segmentation
https://www.geowgs84.com/post/image-classification-vs-object-detection-vs-image-segmentation

[25] DETR:End-to-End Object Detection with Transformers | Learning-Deep-Learning
https://patrick-llgc.github.io/Learning-Deep-Learning/paper_notes/detr.html

[40] xView Dataset - Ultralytics YOLO Docs
https://docs.ultralytics.com/datasets/detect/xview/

[41] [47] [58] [59] GitHub - satellite-image-deep-learning/techniques: Techniques for deep learning with satellite & aerial imagery
https://github.com/satellite-image-deep-learning/techniques

[49] Constraint Loss for Rotated Object Detection in Remote Sensing Images
https://www.mdpi.com/2072-4292/13/21/4291