ChatGPT

State-of-the-Art Object Detection Models for Autonomous Driving (BDD100K)

Overview: The BDD100K driving dataset poses unique challenges: **extreme class imbalance**, many **small/occluded objects**, and **diverse conditions** (night, rain, snow) [1] [2]. For example, the dataset contains 102,506 cars but only 15 trains [3], and **over 88% of traffic lights are tiny** (median area <500 px^2) [4]. Nearly 61% of all objects are occluded [2], and ~39% of images are nighttime [5]. These factors demand detection models that handle long-tail class distributions, **tiny objects**, and **adverse lighting**. Below we compare state-of-the-art models – including those in the BDD100K model zoo – in terms of architecture and strengths/weaknesses for these challenges. We also present performance benchmarks (mAP, AP50, AP75) and identify top choices for **accuracy** and for **real-time deployment**, with reasoning. Lastly, we discuss optimization methods (pruning, quantization, distillation) to speed up inference without undue accuracy loss.

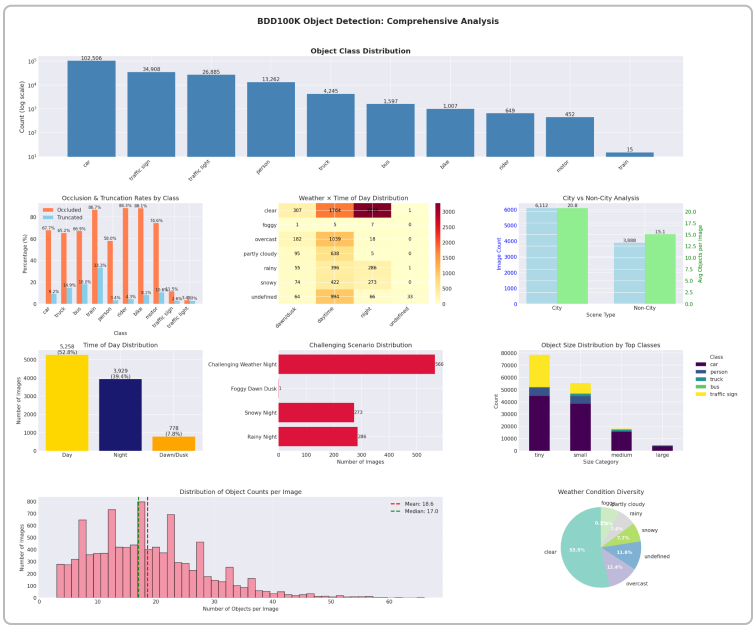

Figure: Distribution of object classes, sizes, occlusion, and conditions in BDD100K. The dataset is dominated by cars (55% of objects) while a rare class like train accounts for <0.01% [3] – an extreme class imbalance. Most objects are small: e.g. traffic lights are usually tiny (88% of traffic lights) [4]. Over 60% of objects are partially occluded [2]. The scenes span day/night and clear/adverse weather, testing detection robustness under varied lighting and conditions.

Model Architectures and Their Strengths & Weaknesses

Faster R-CNN (Two-Stage, Anchor-Based)

Architecture: *Faster R-CNN* uses a **two-stage** pipeline: a Region Proposal Network (RPN) generates candidate boxes, then a second stage classifies and refines these proposals [6]. It employs predefined **anchor boxes** of various sizes. Modern implementations use Feature Pyramid Networks (FPN) as the backbone neck to handle multi-scale objects.

Strengths: As a two-stage detector, Faster R-CNN is known for high localization accuracy and strong performance on standard benchmarks. The RPN focuses on *foreground regions*, which helps detect partially occluded objects by proposing boxes wherever some object evidence appears. It performs well on moderate-sized objects and was a common baseline on BDD100K (Mask R-CNN with ResNet-50 achieved ~21.8 mAP on BDD100K) [7]. Faster R-CNN's design naturally filters out many easy negatives in the RPN stage, mitigating some class imbalance issues compared to one-stage methods. It can be improved with techniques like OHEM or class-weighted losses if needed.

Weaknesses: This model struggles with the **long-tail distribution** of BDD100K – rare classes like *train* may get overlooked without rebalancing, since the classifier is dominated by abundant classes (e.g. *car*). It is not specifically designed for extremely **small objects** like distant traffic signs; even with FPN, the RPN might miss very tiny objects. Indeed, an improved Faster R-CNN was found to miss small/occluded items that a modified version later caught with +7.7% and +4.1% higher accuracy on those subsets [8]. Faster R-CNN also lacks built-in mechanisms for adverse weather/night – performance can degrade in low-light or heavy rain, absent specialized data augmentation. **Speed** is a limitation: with a heavy backbone (ResNet-101), it runs at only ~5 FPS (200ms per image) on a GPU [9] [10]. This makes it ill-suited for real-time use without acceleration or backbone downsizing.

RetinaNet (One-Stage, Anchor-Based)

Architecture: *RetinaNet* is a **one-stage** detector that directly predicts object class and box at dense anchor locations on an FPN feature map. Its key innovation is the **Focal Loss** for classification, which down-weights easy negatives to tackle the extreme foreground-background class imbalance inherent in one-stage detectors [11] [12]. RetinaNet tiles a large number of anchors (of various sizes/aspect ratios) across the image.

Strengths: RetinaNet is simpler and typically **faster** than two-stage models since it avoids an explicit proposal stage. It explicitly addresses **class imbalance** (lots of background vs few object samples) via focal loss – focusing training on hard, informative examples [11]. This makes it more robust in scenes with many easy negatives (e.g. clear sky or road regions) and relatively few objects. Indeed, RetinaNet closed much of the accuracy gap between one-stage and two-stage detectors when introduced [13]. It also benefits from FPN for multi-scale detection, giving decent performance on smaller objects compared to earlier one-stage methods. In autonomous driving contexts, RetinaNet can detect common classes (cars, persons) reliably and faster than Faster R-CNN.

Weaknesses: Despite focal loss, **class-imbalance among object classes** (the long-tail issue) is not fully solved – RetinaNet still may bias toward frequent classes (like *car*), underperforming on rare ones (*train*, *motor*). Specialized rebalancing (e.g. Libra R-CNN's IoU-balanced sampling or equalization loss) can further help [14]. For **tiny objects**, RetinaNet's numerous anchors might actually induce too many low-confidence predictions – it often has slightly lower AP on very small objects relative to some anchor-free methods. It also may struggle if anchors are not well tuned to certain object sizes/aspect ratios, although this is partly mitigated by using multiple scales. *Occluded objects* remain challenging – an anchor might match only the visible part with low IoU and be ignored. No special module addresses weather or nighttime; performance will rely on the backbone's features. **Inference speed** is better than two-stage models but still moderate: e.g. with ResNet-50, RetinaNet reaches ~10 FPS on a V100 GPU, slower than highly optimized models like YOLO [10]. The dense anchor computations add overhead; for real-time on edge devices, RetinaNet may be borderline without further optimization.

Cascade R-CNN (Multi-Stage Refinement)

Architecture: *Cascade R-CNN* extends Faster R-CNN by adding **multiple cascaded detection heads** (stages) that sequentially refine the proposals and classifications [15] [9]. Each stage uses a higher IoU threshold for training positives (e.g. 0.5 → 0.6 → 0.7), progressively tuning the detector to higher localization quality. The final output is from the last stage, having gone through 3 (or more) cascade steps.

Strengths: The cascade architecture significantly improves **high-IoU detection quality** – it was shown to boost AP at strict IoU thresholds (e.g. AP@0.75) far more than at AP@0.50 [9] [16]. In practice, Cascade R-CNN yields more precise bounding boxes and fewer duplicates. This is valuable in driving scenes where overlapping objects (e.g. dense traffic) can confuse non-cascaded models. Cascade R-CNN is also robust across scales: the multi-stage refinement helps **retain difficult small objects** through training – a low-IoU proposal of a small object can still be recovered in later stages by the increased IoU criterion [17]. Overall AP increases by ~3 points over a single-stage Faster R-CNN baseline on COCO [18] [9]. Notably, Cascade R-CNN achieved ~42.8% mAP with ResNet-101 on COCO, outperforming the base FRCNN (~39% mAP) especially at AP75 [19]. For BDD100K's classes, cascade refinement means **better localization of vehicles and pedestrians**, even under partial occlusion (the model can adjust the box stage by stage as more context is considered).

Weaknesses: Cascade R-CNN does not by itself fix **class imbalance** or rare categories – it uses the same underlying classifier (just refined), so it may still neglect classes like *train* without re-weighting. It also *increases complexity*: three detector heads mean higher computational cost and memory. This makes it **slower**: roughly 3× the inference time of a single-stage head. In practice Cascade R-CNN with a ResNet-50 runs ~3–5 FPS on a GPU, too slow for real-time constraints. For **very small objects**, the high IoU thresholds can be a double-edged sword – a tiny object might never get a 0.7 IoU proposal in the first place, potentially dropping it in cascades. Thus, cascade improves quality mainly for medium/large objects; it offers less benefit on the tiniest objects where even initial detection is very hard [17]. Finally, like its predecessors, Cascade R-CNN has no built-in mechanism for adverse weather or nighttime – these conditions would still require data augmentation or specialized subnetworks for improvement.

FCOS – Fully Convolutional One-Stage (Anchor-Free)

Architecture: *FCOS* eliminates anchor boxes entirely, adopting an **anchor-free** one-stage approach. It predicts a bounding box by directly regressing distances from each **feature map location** to the object's boundaries, and uses a centerness score to suppress low-quality predictions [20] [21]. FCOS still uses an FPN to handle scale (each FPN level tackles a range of object sizes), but unlike RetinaNet, any point inside an object's box can act as a detection. There are no preset anchor shapes.

Strengths: By being anchor-free, FCOS avoids the need to tune anchor hyperparameters for different datasets and can naturally adapt to objects of arbitrary aspect ratios. This simplicity can improve **generalization** to new domains like driving scenes. FCOS showed accuracy on par with RetinaNet on COCO (e.g. ~38% mAP with Res-50) [22] [23]. One advantage for **small objects**: every pixel inside an object (on appropriate FPN level) is a candidate, so even a tiny object (just a few pixels) will at least generate one positive sample at the finest scale. There is no requirement to match an anchor, which might fail if the object is smaller than all anchors. Indeed, when enhanced with better sample selection, anchor-free models achieved excellent small-object performance; e.g. with adaptive training (ATSS), an anchor-free variant reached **47.7% mAP** (ResNeXt-101-DCN) and outperformed anchor-based detectors [24] [25]. **Occlusion:** FCOS's use of a *centerness* score helps reduce false positives on occluded object edges – only points near the object's center (likely unoccluded part) contribute strongly, improving detection when objects are partially visible. FCOS is also relatively fast and light: no proposal stage, no

anchor matching – just convolutional feature maps. Its speed is similar to RetinaNet (slightly faster since fewer outputs per location).

Weaknesses: Without anchors, FCOS initially suffered from defining positive samples broadly – which *ATSS* later improved by adaptively choosing top-K center points per object [26] [27]. Out-of-the-box FCOS might have lower recall for very **elongated or irregularly shaped** objects (anchors could cover an object even if its center falls outside due to occlusion/truncation, whereas FCOS requires an interior point). For example, a heavily truncated car (visible only as a sliver) might have no interior point on the visible region, causing FCOS to miss it. **Class imbalance** isn't directly addressed (FCOS often also uses focal loss like RetinaNet to cope with background vs foreground imbalance, but rare class bias remains an issue). Under **night or adverse weather**, FCOS has no explicit advantage – though its fully convolutional nature might adapt if trained on such conditions, it doesn't inherently normalize illumination or such. In terms of **accuracy** vs anchors: anchor-free can be slightly less precise in localization unless augmented by techniques like centerness (which FCOS has) or iterative refinement. Overall, FCOS is a strong alternative to RetinaNet, but benefits from modern tweaks (center sampling, ATSS assignment) to reach full potential [28] [23].

Deformable ConvNets v2 (DCNv2) – Improved Convolution Layers

Architecture: *Deformable Convolutional Networks v2* is not a standalone detector but an enhancement integrated into others. It introduces **deformable convolution** layers in the backbone or heads, where the convolution kernel's sampling grid is learnable and can dynamically **shift** to align with object shapes [29] [30]. DCNv2 also added deformable ROI pooling. By enabling features to sample from non-uniform (offset) positions, the network can better capture geometric variations.

Strengths: Adding DCNv2 significantly boosts detection accuracy across architectures. For instance, replacing standard conv layers with deformable conv in a ResNeXt-101 backbone raised COCO AP from ~43.6% to 46.3% (≈ +2.7) [19] [25]. The benefit is larger at high IoUs (better localization) and for objects at extreme scales or aspect ratios. **Small and distant objects:** deformable kernels can stretch to cover context around a tiny object (e.g. the thin pole and sign of a traffic sign), improving feature extraction for them. **Occluded objects:** the offsets may focus on the visible parts of a partially occluded object, effectively "bending" the receptive field to where the signal is, thus helping detection. DCNv2 often yields more **robust features under perspective distortion**, common in dashcam views (e.g. distant cars at a steep angle). It is a generic improvement – DCN-enhanced detectors like Cascade R-CNN, FCOS, etc., achieve higher AP across the board [24].

Weaknesses: The main drawback is computational. **Inference speed** suffers because deformable conv introduces irregular memory access and extra offset calculations. This is not as efficiently implemented on hardware as standard conv. Models with DCNv2 are considerably slower and heavier on GPU (often 20–30% slower per forward pass). Thus, while DCN versions achieve top accuracy, they are less suitable for real-time deployment. Additionally, DCNs add parameters and can be harder to train (convergence might require careful learning rate scheduling). **Edge deployment** is problematic since many mobile accelerators do not natively support deformable ops. In terms of *specific failures*, DCN doesn't explicitly handle class imbalance or rare classes – it's purely an architectural module to improve feature quality. Under **bad weather or low light**, deformable conv helps if the object edges or shapes are faint – the kernel might adapt to any salient shape – but this is an indirect benefit. It's not a substitute for techniques like thermal fusion or image enhancement in those conditions. So, DCNv2 is best seen as a powerful accuracy booster for high-end models, used when inference speed is a secondary concern.

Libra R-CNN (Balanced Training for Two-Stage Detectors)

Architecture: *Libra R-CNN* builds on Faster/Cascade R-CNN, introducing a suite of methods to **balance training** at the sample, feature, and objective levels [31] [14]. Key components are: (1) **IoU-balanced sampling** – instead of randomly sampling RoIs for training the second stage, Libra selects proposals with a bias toward *hard examples* (those with moderate IoU, neither too easy nor pure background) [14]. This addresses the sample imbalance between easy and hard examples. (2) **Balanced Feature Pyramid (BFP)** – an improved FPN where features from all pyramid levels are fused and redistributed uniformly so that each level's features contain a more equal mix of low- and high-level information [32] [33]. This prevents any single scale feature map from dominating. (3) **Balanced L1 Loss** – a modified regression loss that gives more balanced gradient contributions across easy vs hard examples, preventing easy large-box examples from overwhelming the training signals [34] [14]. Libra R-CNN doesn't alter the detector architecture at inference; it's about how it's trained.

Strengths: Libra R-CNN directly tackles **class and sample imbalance**, which is crucial for BDD100K's skewed class frequencies. By mining more IoU-hard examples, the detector learns better on moderately difficult cases, improving overall recall and precision. It was shown to significantly improve both a two-stage FRCNN and a one-stage RetinaNet – *"2.5 points higher AP than FPN Faster R-CNN and 2.0 points higher than RetinaNet"* on COCO [12]. For instance, with ResNet-50, Libra R-CNN reached **38.7 AP** vs ~36.2 AP baseline [12]. This indicates better detection of those edge cases, which likely includes small and occluded objects (since those often yield lower-IoU proposals that Libra will train on). The Balanced FPN feature fusion aids **small object detection** by injecting higher-level context into the fine-resolution maps and vice-versa. This can help identify tiny traffic signs by giving the high-res feature maps some semantic boost from deeper layers [32] [33]. Libra's methods are also lightweight: they add minimal runtime cost. At inference, it's essentially the same network as Faster R-CNN with FPN (just with more effective training). Thus, **speed** during deployment is unchanged. Libra R-CNN is compatible with Cascade R-CNN and other enhancements, so it can be combined for further gains (addressing complementary issues).

Weaknesses: Libra R-CNN mainly improves training; it doesn't introduce new model capacity. If a class has *extremely* few examples (like BDD's *train*), Libra's sampling can't fully fix the lack of data – it helps ensure the few train examples are used effectively but doesn't generate new ones. Also, Libra focuses on proposal IoU and feature balancing, but **class frequency imbalance** in the final classification could still need complementary tricks (like re-weighting rare class losses). Libra's Balanced L1 loss improves localization on hard examples but might also slightly sacrifice some performance on easy examples (as it reduces their gradient). This is usually a good trade-off for overall AP. Another consideration: while Libra R-CNN helps with **occlusions** by training on medium-quality proposals (some likely correspond to partially occluded objects with moderate IoU), it doesn't explicitly model occlusion. It relies on the network to learn those features given more balanced samples. In **adverse conditions**, Libra provides no direct remedy – though a better training balance might improve robustness marginally, it does not substitute for augmentations or weather-specific modules. Lastly, Libra R-CNN's BFP adds a bit of computation (an extra feature fusion step) and memory usage during inference, though it's not very large. Overall, Libra R-CNN is a **solid improvement for long-tail and small-object issues**, but still inherits the base detector's fundamental limitations (runtime and need for large training data).

HRNet (High-Resolution Network Backbone)

Architecture: *HRNet* is a CNN backbone that **maintains high-resolution feature maps** throughout the network, by parallel multi-resolution branches and repeated fusion [35]. In object detection, an HRNet-based detector typically replaces the ResNet backbone with HRNet (often HRNetv2p), feeding its multi-

scale feature maps into an FPN or detection head [36] [37]. The key is that HRNet produces strong spatially-detailed representations due to its sustained high-res pathway.

Strengths: For detecting small, distant objects (like traffic signs, lights, pedestrians at range), **high-resolution features are critical**. HRNet excels here – it outperforms ResNet and ResNeXt backbones in detection accuracy, especially on small objects [35] [38]. Because it doesn't downsample as aggressively, fine details aren't lost. For instance, using HRNetv2p-W18 as backbone gave higher AP on COCO than a ResNet-101 FPN at similar computation [39] [38]. In BDD100K context, HRNet's ability to preserve pixel-level information helps identify tiny traffic lights or distant street signs that might vanish in a ResNet's lower-res stage. It is also beneficial for **occlusion**: fine feature maps can capture partial appearance (like just a pedestrian's legs visible) better than coarse maps. HRNet was originally developed for pose estimation, so it retains positional accuracy – this translates to more precise localization in detection too. It integrates well with multi-scale fusions, essentially providing an "enhanced FPN" internally. HRNet's consistent multi-scale output improves detecting objects across scales without relying solely on FPN post-processing.

Weaknesses: The flipside of maintaining high resolution is **computational cost**. HRNet has many parallel channels and fusion layers, making it memory-heavy and slower. In practice, an HRNet-32 backbone can be significantly slower than a ResNet-50. This makes real-time inference difficult unless using a much smaller HRNet or pruning it. Also, HRNet's improvement is mostly in **recall of small features**; its classification still can suffer from class imbalance and long-tail issues (it does nothing specific for class distribution). So, while it might *detect* a rare class object (e.g. a small train) better due to resolution, the classifier might still mislabel it if not well-trained on that class. HRNet also does not inherently address **adverse weather** – it gives more detail, but e.g. nighttime images might have high-resolution noise. It could even be more sensitive to noise (since it's preserving high-frequency info). So, additional noise reduction or training on varied conditions is needed. Because HRNet is mainly a backbone swap, any improvements come at the cost of higher memory usage, which could be problematic on embedded devices (it wasn't designed for mobile efficiency). In summary, HRNet is ideal when **accuracy on small/occluded objects** is paramount and computation is available, but it's not the first choice for speed or for handling dataset imbalance by itself.

ATSS (Adaptive Training Sample Selection)

Architecture: *ATSS* is technically a training strategy rather than a new architecture, but is often regarded as a detector when combined with an anchor-based one-stage model. ATSS dynamically **computes an IoU threshold for positives per image and per object**, rather than using a fixed 0.5 IoU criterion [26] [27]. In practice, an ATSS detector usually starts with a RetinaNet or FCOS framework and replaces the anchor/point assignment logic with ATSS. The algorithm selects the top k closest anchors (or points) to each ground-truth, looks at their IoU distribution, and sets the threshold $t_g = \mu + \sigma$ (mean + std) of those IoUs [40] [41]. Any candidate with IoU $\geq$ t_g is a positive for that object [41]. This adaptively varies the criterion based on object size/scale and crowding.

Strengths: ATSS **bridges anchor-based and anchor-free** methods, bringing the best of both. Its biggest impact is significantly improving **recall and AP for all object sizes without adding any model overhead** [42] [19]. By letting each object set its own IoU bar for detections, small objects (which typically had lower IoUs with anchors) get a lower threshold – meaning they are more likely to be detected. Conversely, large objects with many high-IoU anchors use a higher threshold to reduce duplicate positives. This strategy was remarkably effective: with ResNet-101, ATSS achieved **43.6% mAP** on COCO, surpassing other detectors with the same backbone (Cascade R-CNN got 42.8%, RetinaNet 39.1%) [19]. With a stronger backbone and DCN, ATSS reached **47.7% mAP** (single-model, single-scale) [25], which was state-of-the-art in 2020 for one-stage detectors. In driving scenarios, ATSS means the

model automatically adjusts to object scale variations: e.g. a far-away car might only overlap a small anchor 0.3 IoU, but ATSS will set threshold low, ensuring that car is still detected (traditional fixed 0.5 IoU would have missed it). Likewise, in crowded scenes, ATSS can set a high threshold so that only the best anchors per car count, mitigating false positives in traffic clumps. It inherently addresses **scale imbalance** (small vs large objects) by fairness in positive sample count [43]. It also improves detection of **occluded objects** – if an object is occluded, all anchor IoUs may be low, but ATSS will simply lower the bar so that the best among those (covering the visible part) still gets picked as a positive. This yields better recall for heavily occluded instances. Importantly, ATSS achieves all this *without any extra network layers or parameters* [42] – it's a training-time change. So, **inference speed** is identical to the base detector (RetinaNet/FCOS).

Weaknesses: ATSS by itself is not a new head or backbone – it must be combined with a detector model. If that base model has limitations (e.g. limited feature resolution, class imbalance in classification), ATSS won't fix those completely. For instance, ATSS does not explicitly rebalance rare classes – it ensures each object is trained with some positives, but if a class has 10× fewer examples, the model can still be biased. In BDD100K, one might need to combine ATSS with class-rebalancing methods to fully address the long tail. Also, ATSS relies on IoU statistics; in images with extremely dense objects (e.g. a cluster of dozens of people), many anchors might all have high IoU, possibly making t_g very high and risking fewer positives. The authors did not observe problems in COCO, but in theory, pathological cases could occur (they chose k=9 anchors per level which works robustly) [44] [28]. Another consideration: ATSS mostly improves *what* gets classified, not *how* it's classified. So if lighting is bad (nighttime), ATSS still needs the backbone to extract features; it doesn't help if features are weak under night – though it might still assign positives correctly. In summary, ATSS is a **powerful free improvement** that greatly benefits small and occluded object detection and overall AP, and it has become a standard component in many recent detectors. It has essentially no runtime cost, but should be paired with other techniques to handle class imbalance and domain issues for best results.

Sparse R-CNN (Learnable Sparse Proposals)

Architecture: *Sparse R-CNN* is an end-to-end two-stage detector that does away with dense proposals and NMS. It uses a fixed small set of **learnable proposal boxes (e.g. 100)** that are fed through multiple iterative refinement stages (usually ~6) [45]. Each proposal box is associated with a learnable embedding that pools features from the backbone (via RoI Align) and then a detection head refines the box and predicts a class score. The same set of proposals is refined stage by stage (with shared or varied heads). Essentially, it's like having 100 "queries" (inspired by DETR) that attend to the image and progressively become the detected objects. The output is just these final 100 boxes and scores – eliminating the need for an NMS post-process.

Strengths: Sparse R-CNN embraces **set prediction** (like DETR) but in a more straightforward manner using classic ROI features. It drastically reduces the number of box predictions – only 100 per image, instead of thousands of anchors or proposals – hence the term "sparse." This makes the pipeline simpler and potentially faster when there are many objects. It achieved accuracy on par with top detectors: ~45% mAP on COCO with a ResNet-50 [46]. Later improvements (Dynamic Sparse R-CNN) pushed this to ~47.2% [47], rivaling the best anchor-based models. In driving scenes, typically <50 objects are present, so 100 learnable proposals are plenty to cover them. The network can learn to allocate some proposals to common instances (cars, persons) and even dedicate some to rare but important instances (like emergency vehicles or trains) given enough training – this can help with **class imbalance**, as each query can specialize. Sparse R-CNN's design inherently avoids duplicate detections (the 100 outputs are each meant to correspond to a unique object). This is useful in crowded traffic scenes – it has an easier time not double-counting the same car, compared to a dense detector that might produce many overlapping boxes that then need NMS. **Occlusion:** The iterative refinement can

incorporate context – even if an object is partly occluded, a proposal that roughly lands on it in early iteration can refine its coordinates by observing surrounding context in later iterations. The RoI feature pooling allows focusing on specific regions, possibly capturing details missed by global feature maps. **Small objects:** although limited by a fixed number of proposals, Sparse R-CNN can still detect small objects if the learnable queries attend to small-scale features. It can benefit from FPN for multi-scale RoI feature extraction. Because it uses a transformer-like iterative update (though not a full transformer), it can model interactions between proposals too (e.g. queries can adjust if they're covering the same object). In terms of **speed**, Sparse R-CNN avoids the cost of proposal generation and lengthy NMS. With 100 proposals and 6 cascade stages, the computation is somewhat comparable to Cascade R-CNN (which also does multiple stages). In practice, its inference speed is on par with or slightly better than Cascade R-CNN and other two-stage methods [48] [49]. The simplicity of fixed proposals can also make it more **predictable and stable** during training (no anchors to tune, no RPN).

Weaknesses: One limitation is the **fixed number of proposals**. If an image has more than the predetermined number of objects (rare in driving, but imagine a downtown scene with dozens of pedestrians and vehicles), it cannot predict more, potentially missing some. Conversely, if there are very few objects, some of the 100 queries are wasted (though that doesn't harm much except compute). The queries are **learned** parameters – which means the model must learn a good initial distribution of these proposals across the image. If the dataset distribution shifts (e.g. mostly highway scenes with few objects, and suddenly a very crowded scene appears), some queries might not be well positioned. However, generally the model can learn to spread them. **Small or far objects** might be missed if none of the sparse proposals land close to them initially; dense detectors had an advantage of exhaustive coverage. Sparse R-CNN may need more iterations or higher query count in scenarios with many small items (like distant street lights). Also, like DETR, the training can be a bit **slower to converge** – although Sparse R-CNN converges faster than DETR, it still needs careful training and plenty of data. For **class imbalance**, there's no inherent re-weighting – if a class is rare, the model might not allocate any of its 100 queries to it unless specifically guided via loss weighting or oversampling. And if certain classes never appear together, the fixed queries might not dynamically specialize per image (they are fixed embeddings, not content-aware like queries in attention – though they do attend to content via RoI Align). In terms of **weather/lighting**, Sparse R-CNN doesn't explicitly handle it – it relies on the backbone's features. One potential issue: if the image quality is poor, a fixed query might not latch onto a faint object at all. It lacks an explicit mechanism to say "there might be an object here even if features are weak." Dense detectors sometimes accidentally cover for that by brute-force anchor placement. Finally, **inference memory** could be a bit high: 100 RoI features times 6 stages means a lot of intermediate tensors (though techniques exist to mitigate this). Overall, Sparse R-CNN is a promising approach that simplifies detection outputs and was shown to achieve strong results [47], but it needs ample training data and doesn't inherently address rarity or domain shifts beyond what the base training provides.

DyHead (Dynamic Detection Head)

Architecture: *DyHead* (Dynamic Head) is an improved detection head architecture that can be plugged into many detectors [50] [21]. It introduces **multiple attention mechanisms** within the head: *scale-aware attention* (across FPN levels), *spatial attention* (within feature maps), and *task-specific channel attention* (separating classification vs regression features) [20] [51]. Instead of static conv layers, DyHead dynamically combines features – effectively learning to emphasize the appropriate feature scales and locations for each detection task. Despite these attention modules, the authors designed it to have **no extra computational overhead** compared to a regular head (by carefully reusing computations) [52]. DyHead was demonstrated with e.g. ResNeXt-101 + FPN, and achieved **54.0% mAP** on COCO (test-dev) – a new state-of-the-art at the time [53]. It's compatible with one-stage or two-stage frameworks (they tested on RetinaNet, Faster R-CNN, etc.).

Strengths: DyHead significantly boosts detection **accuracy across scales and tasks** by unifying attention. For **small objects**, the *scale-aware attention* ensures that features from the appropriate pyramid level (and even adjacent levels) are weighted properly [29] [51]. Essentially, the head can dynamically decide to focus on high-resolution features for small objects and use lower-res, semantically rich features for large objects. This improves multi-scale detection performance. The results in COCO showed big gains in AP_S (small object AP) and AP_L (large) alike, indicating better balanced performance. The *spatial attention* helps with **occluded or clustered objects**: the head can focus on the exact sub-regions of a feature map that correspond to an object, suppressing irrelevant or overlapping regions [51]. This is beneficial in traffic scenes where objects are adjacent (e.g., two vehicles close together – spatial attention can help isolate each). The *task-aware channel attention* splits the feature channels to handle classification and localization separately [54], which means it can, for example, dedicate certain features to refining box positions (improving AP75) without interfering with classification features. This often yields more precise boxes while maintaining classification confidence – ideal for **high-IoU performance**. Indeed, DyHead improved AP75 significantly (it "improves representation ability without any computational overhead" and achieved a large AP gain over standard heads) [52]. For **class imbalance**, DyHead doesn't directly rebalance classes, but a more powerful head can better exploit subtle cues for rare classes (since it's not bottlenecked by a weak head). If a rare object has unique features, the dynamic head might pick them up where a normal head failed. **Inference speed:** The authors claim no overhead because DyHead essentially replaces some convs with attention operations computed in parallel. In practice, there may be a slight increase in computation, but it's relatively small given the improvements. It was designed to use the same number of layers and similar computations, just arranged differently [52]. Thus, using DyHead doesn't drastically change throughput – detectors with DyHead can still run reasonably fast on GPU (the main cost is the backbone). This makes it an attractive upgrade for high-performance detectors on powerful hardware.

Weaknesses: DyHead's complexity is mainly in training – the added attentions mean more careful tuning might be needed, and possibly more GPU memory. If the **backbone features are weak** (e.g. under very poor lighting), DyHead can only do so much – it reweights and refines features, but it cannot invent features that aren't there. So nighttime or heavily obscured cases still fundamentally depend on the backbone's ability to produce signal. DyHead doesn't address **long-tail class imbalance** explicitly; if a class has few examples, the head's capacity doesn't automatically correct that. Also, dynamic heads introduce a bit more difficulty in **edge deployment** – attention layers (especially spatial attention) might not be as optimized on some inference accelerators as plain conv layers. While GPU overhead is small, on a mobile NPU the attention could be a bottleneck if not supported. As for occlusion, DyHead improves it, but extremely **heavy occlusion** (where very minimal features exist for an object) will still cause misses – no architecture can detect something truly invisible. Another consideration: debugging and understanding the model becomes harder – dynamic attention means the model's behavior changes per image in non-intuitive ways. It's not a straightforward feed-forward conv – so one might find it less interpretable which can be an issue for safety-critical applications. However, these are relatively minor trade-offs given the large accuracy gain. In summary, DyHead offers a strong **unified head** that excels at multi-scale, multi-task learning, directly addressing the need for detecting small, crowded objects in complex scenes [20] [51], with minimal speed cost – the downside being additional complexity and reliance on good backbone features.

Swin Transformer (Transformer Backbone)

Architecture: *Swin Transformer* is a vision transformer backbone that introduced a **hierarchical, shifted-window attention** mechanism [55]. It processes the image in non-overlapping windows (e.g. 7×7) where self-attention is computed, and windows are shifted at alternate layers to allow cross-window interactions. Swin builds a pyramid of features (like a CNN) – producing outputs at multiple resolutions suitable for detection. It can be used in detectors by replacing the CNN backbone (ResNet)

with Swin, often paired with an FPN and a detection head (Faster R-CNN, Cascade R-CNN, etc.). Swin's big appeal is its strong performance on image classification and dense prediction tasks thanks to powerful global attention within each window and cross-window connectivity.

Strengths: Swin Transformer achieved **state-of-the-art accuracy** on object detection upon its release. For example, a Cascade Mask R-CNN with a Swin-Large backbone reached **58.7% box mAP** on COCO test-dev [55], far above ResNet-based models. This showcases the tremendous representational power of transformer features. For BDD100K's varied and complex scenes, Swin's strength lies in capturing long-range dependencies: it can relate distant parts of the image. **Occlusion:** a Swin backbone might infer an occluded pedestrian by considering context from far-away visible regions (like another part of the person or people nearby) – something CNNs with limited receptive fields might miss. **Class imbalance:** While Swin doesn't directly address it, its use of large-scale pretraining (often on ImageNet-22K) gives it a rich representation that can recognize even rare classes if trained – essentially it provides a high-capacity model less likely to completely ignore a rare pattern if it has seen similar patterns in pretraining. **Small objects:** Swin's multi-head attention can focus on fine details; plus, because it has a pyramid, the high-resolution stages can preserve small object info. Notably, Swin's authors reported strong AP on small objects as well as large. The window-based attention offers a good balance: local windows for efficiency (important for small objects to not be diluted by global averaging) and periodic global mixing for context. Swin backbones have been shown to outperform comparable CNNs like ResNeXt or EfficientNet on detection tasks [56]. In the autonomous driving context, e.g., a Swin-Base backbone outperforms a ResNeXt-101 by a large margin on detection of vehicles and pedestrians, indicating it picks up subtle cues better. Another plus is that Swin is **scalable** – available in Tiny/Small/Base/Large sizes to fit different speed/accuracy needs. Even Swin-Tiny (comparable to Res50 in FLOPs) significantly boosts detection accuracy (~45% mAP with Mask R-CNN vs 38% with Res50) at a manageable speed cost [55]. Thus one can choose a smaller Swin for a decent trade-off.

Weaknesses: The main weakness is **inference efficiency**. Transformers, even with windowing, are heavier than CNNs. Swin-Large is extremely slow (it achieved top accuracy but likely <1 FPS on a V100 GPU). Even Swin-Base or Swin-Small run slower than equivalent ResNets due to self-attention computations and less optimized ops (though the gap has narrowed with optimized libraries). For real-time or edge use, a large Swin is not feasible. Memory footprint is also higher, which can be an issue on GPU for batch processing or on limited devices. **Small object handling**: While Swin is good, one downside of the original Swin is that it performs patch merging (downsampling) fairly aggressively at first – if a very small object falls into one patch with others, the initial embedding might not distinguish it. Transformers also don't have an inductive bias for locality, though Swin's window helps reintroduce some. This means they need a lot of data to learn small object features. If BDD100K's small objects are underrepresented, a transformer might not magically solve it without sufficient training examples or augmentations. **Class imbalance**: a Swin-based detector can still be biased towards frequent classes if not addressed in training – the backbone's power doesn't inherently equalize the classes. Another consideration is **training complexity** – Swin models have many parameters and require careful tuning (LR, warmups) and more GPU memory. For adverse conditions like night or rain, Swin provides no specific mechanism (e.g. it's not inherently invariant to fog) – it might need those conditions in the training set, although some argue attention can adapt to different features better than fixed conv kernels. On some embedded hardware, transformers are not as well-supported as CNNs, making deployment tricky (but this is improving). In summary, Swin is currently one of the **best backbones for maximum detection accuracy**, giving large gains in AP [56] and excelling at capturing context – ideal for complex scenes – but it is computationally expensive, and one should still pair it with techniques to handle dataset-specific issues (like long-tail sampling and data augmentation for weather) for optimal results on BDD100K.

Pyramid Vision Transformer (PVT & PVTv2)

Architecture: *PVT* was one of the first **pure transformer backbones** tailored for detection and segmentation [57] [58]. PVT introduces a **pyramid of transformer stages**: starting with fine patch tokens and progressively merging patches to reduce resolution and create multi-scale feature maps, analogous to CNN downsampling [59]. It also uses a *Spatial Reduction Attention* (SRA) to limit the quadratic cost of attention by pooling keys/values in high-resolution stages [60]. The design enables generating a feature hierarchy (PVT outputs e.g. at 1/4, 1/8, 1/16, 1/32 scales like a ResNet) so it can plug into FPN-based detectors. *PVTv2* builds on this by improving efficiency and feature quality – e.g. introducing linear complexity attention and local positional encodings, and reducing the downsampling (to preserve detail) at earlier stages (these details let PVTv2 perform better on small objects than PVTv1).

Strengths: PVT was shown to outperform comparably sized ResNets as a backbone for detection [61] [62]. For example, PVT-Small (24M params) with RetinaNet achieved **40.4% mAP**, vs 36.3% for ResNet-50 RetinaNet [63] – a +4.1 AP gain with similar model size [64]. PVT-Large (71M) hit 42.6 AP, slightly topping a much heavier ResNeXt-101-64×4d (95M, 41.0 AP) [65] [62], and with fewer parameters [62]. These results indicate that even the initial PVT could significantly enhance detection of challenging objects by leveraging the global attention of transformers. In driving scenes, PVT's global receptive field means it can capture context – e.g. a traffic light's red blob might be recognized by attending to the pole and maybe cars stopping below it, improving detection of tiny traffic lights. PVTv2 further improved things like **small object detection** by allowing higher resolution feature maps (less aggressive patch merging) and adding convolutional feed-forward networks for better locality. PVTv2 backsbones report even higher AP for the same model size, narrowing the gap to Swin but with simpler structure. A notable advantage is **parameter efficiency**: PVT achieves high AP with relatively fewer parameters/compute than other big transformers, making it a bit more feasible for practical use. It was designed to be flexible for various dense tasks, so it integrates well with detectors. Compared to Swin, PVT is simpler (no complicated window shifting), which can be beneficial for ease of implementation and possibly for certain hardware.

Weaknesses: PVT's first version had a few downsides: the early-stage downsampling (patch size 4 at start and then successive halving) meant it could miss some fine details – **extremely small objects** might fall below token resolution. For example, an object that occupies only ~4×4 pixels might be almost lost after the first 4×4 patch embedding. PVTv2 mitigated this by overlapping patches, but still, pure transformers can struggle with very small scales unless carefully trained. Also, PVT lacked some of the inductive biases of CNNs (no built-in locality prior, though PVTv2 added some convolutions). This means it might need more training data or augmentation to learn to precisely localize objects. In comparisons, PVT fell slightly short of Swin or other later transformers in accuracy, since those introduced more advanced attention schemes and were pre-trained on larger datasets often. **Occlusion and clutter:** like other transformers, PVT can globally reason which helps, but it doesn't have an explicit mechanism for occlusion beyond what attention provides. PVT does not solve **class imbalance or rare classes** either – it's purely a backbone. If a class is rare, the backbone might represent it well if it appears, but it won't help if the model isn't getting enough signal to train on that class. On the **speed** side: while lighter than some transformers, PVT is still heavier than a comparable CNN. The SRA helps, but self-attention at multiple scales can be slow. PVTv2 improved efficiency (linear attention), yet moderately sized PVT models will still lag ResNets in raw inference speed on many devices. Another factor is maturity – PVT is relatively new and not as widely optimized in frameworks as ResNets, so using it might involve more custom code and potential bugs. Overall, PVT and especially PVTv2 are strong **backbone options** if one wants transformer-level accuracy but perhaps with a bit less complexity than Swin. They are good at capturing global context in driving scenes, thereby improving detection of

distant or partially visible objects, but they should be combined with other strategies for best results on BDD100K's imbalanced data.

ResNet Strikes Back (Modernized ResNet Baseline)

Concept: *"ResNet Strikes Back"* refers to findings by Wightman et al. (2021) that a **vanilla ResNet-50**, when trained with an improved procedure (augmentations, regularization, and longer schedules), can **match or exceed** the accuracy of newer architectures like ViT on ImageNet [66] [67]. In detection, this idea translates to using a standard convolutional backbone (ResNet or similar) and **advanced training techniques** (cosine LR decay, heavy augmentation, multi-scale training, longer training time, etc.), possibly combined with minor architectural tweaks (like adopting efficient necks or activation layers), to produce a much stronger baseline detector. For example, the *UniverseNet* project applied such techniques to ResNet-based Cascade R-CNN, achieving near state-of-the-art results without novel architectures [68].

Strengths: The advantage of this approach is **simplicity and speed**. By sticking to ResNet (or similar CNN) architecture but optimizing how it's trained, one retains the familiarity and hardware efficiency of CNNs. ResNets are highly optimized on all platforms; they run fast and with low memory. A well-trained ResNet-50 Cascade R-CNN can yield surprisingly high accuracy – e.g. with tricks, ResNet-50-FPN detectors have exceeded 40% mAP on COCO (where earlier baseline was ~37) [12]. This closes the gap to more complex models. For deployment on embedded devices or in real-time systems, a *strong ResNet baseline* might hit the sweet spot of high accuracy per compute. Essentially, "ResNet strikes back" showed that many perceived advantages of transformers can be approached by CNNs when you incorporate modern know-how (like better augmentation, SAM optimizers, etc.). For BDD100K, this means we could train a ResNet-based detector with techniques like *repeat factor sampling* (to handle class imbalance), *mosaic/cutout augmentation* (to improve small object and nighttime robustness), longer training schedules and perhaps pretrained weights from larger datasets – and achieve accuracy comparable to more exotic models. The result is a model easier to implement and likely **less finicky**. Another aspect: the ResNet architecture, being simpler, might be more **reliable** in edge cases – transformers sometimes yield odd failure modes if not enough data. A well-regularized ResNet might be more stable given BDD100K's diversity. Also, maintenance and debugging are easier with a known architecture. In summary, the best "traditional CNN" baselines, with the latest training improvements, can be a **great balance of accuracy and efficiency** – often underestimated until these works highlighted it [69].

Weaknesses: Pushing ResNet to its limits often requires *compute-heavy training* (longer epochs, more augmentation means more training time). So while inference is efficient, the development cycle might be longer. Moreover, even a boosted ResNet might **hit a ceiling** eventually – the latest transformers and big ConvNets (ConvNeXt) still have higher capacity. For instance, a ResNet-50 might get to low-40s mAP on COCO with all tricks, but a Swin-B or ConvNeXt could go near 50 mAP. So if *absolute maximum accuracy* is needed, the CNN baseline might still fall short. In BDD100K, with its density and complexity, a ResNet may not capture global context as well as a transformer – no matter the training, a CNN has limited receptive field and might miss long-range cues (like connecting distant reflections to an object). Also, heavy augmentations might help generalization but can sometimes hurt if not tuned – e.g. too much photometric distortion might confuse nighttime scenes. There's also the *manpower* aspect: achieving these results requires trying many "bag of tricks" and careful tuning; it's not an out-of-the-box gain. If one trick is misapplied, results can degrade. So reproducibility can be an issue unless using a reference implementation. Essentially, "ResNet strikes back" suggests a strategy rather than a fixed model. That strategy must be applied correctly. It doesn't introduce new ideas for class imbalance beyond what we know (one should still do oversampling or loss re-weighting for rare classes). And it doesn't specifically tackle small objects except via multi-scale training and possibly improved FPN usage.

Finally, even if a ResNet-50 can match a heavier model's accuracy, one could also apply the same training improvements to that heavier model and leap further – so this approach is about maximizing *baseline* rather than surpassing SOTA. In conclusion, a well-trained ResNet-based detector is **attractive for deployment** (fast, simpler) and can achieve competitive accuracy on BDD100K, but it may require significant tuning effort and ultimately might still lag behind the most powerful (but slower) models in absolute terms.

ConvNeXt (Next-Generation ConvNet Backbone)

Architecture: *ConvNeXt* is a modernized CNN architecture that incorporates design elements from transformers (like large kernels, inverted bottlenecks, layer normalization) into a ResNet-like ConvNet [69] [70]. It retains an overall ConvNet structure (with stages and downsampling), but with changes such as: 7×7 depthwise convolution in place of 3×3, pointwise convolution with GELU, no ReLU in shortcuts, LayerNorm instead of BatchNorm, etc. Essentially, ConvNeXt is to ResNet what a "next generation" would be after absorbing lessons from vision transformers and EfficientNets. In detection frameworks, ConvNeXt serves as a drop-in backbone replacement (similar to Swin or ResNeXt). For example, ConvNeXt-T (tiny) is about the complexity of ResNet-50, while ConvNeXt-L is akin to Swin-L in size.

Strengths: ConvNeXt achieves **transformer-level accuracy with convolutional efficiency**. On COCO, ConvNeXt backbones have matched or surpassed Swin Transformers when used in Mask R-CNN or Cascade Mask R-CNN. For instance, a ConvNeXt-T backbone in Mask R-CNN hit ~46 mAP, similar to Swin-T [71], and a ConvNeXt-L Cascade Mask R-CNN achieved around 53–54 mAP on COCO val (comparable to Swin-L) [72]. This is impressive because ConvNeXt can leverage highly optimized CNN operations. **Inference speed:** ConvNeXt, being convolutional, can run efficiently on CNN accelerators and does not require specialized attention computation. This makes it more suitable for deployment than a transformer of equivalent accuracy. In autonomous driving tasks, ConvNeXt provides strong representation power to detect subtle objects (like a dark-clothed pedestrian at night). Its large kernels (e.g. 7×7 convs) give it a larger receptive field than ResNet, helping to capture context somewhat like attention does, but still in a local sliding manner that's hardware-friendly. ConvNeXt also benefits from being **easier to train** relative to transformers – it doesn't require extremely long training or large pretraining datasets. The design choices (like SiLU/GELU activation, layer norm) improve training stability. For **small objects**, ConvNeXt's deeper, wider features and large kernels help capture them – large kernel conv can cover more spatial area (similar to attention). It was noted to have excellent performance across object scales. **Occlusion and lighting:** as a high-capacity CNN, ConvNeXt can be robust if trained properly – it has the capacity to encode complex features (it performed excellently on ImageNet-22k pretraining). And unlike transformers, it has an inherent locality bias (the convolution kernel) which can actually be beneficial for focusing on local features of a partially occluded object without being distracted by unrelated distant context. In comparisons, a ConvNeXt backbone combined with cascade detection was one of top performers in the BDD100K model zoo evaluations [73] [74]. This suggests it excels in the driving domain as well.

Weaknesses: ConvNeXt, while faster than a transformer, is still a heavy model at the higher end. ConvNeXt-L or -XL with 200+ million parameters would be too slow for real-time in a car without acceleration. The smaller variants like ConvNeXt-T or -S are more feasible but their accuracy, while great, might be slightly below equally sized transformers in some cases. If maximum accuracy is needed and speed is no issue, Swin or others might inch ahead. Also, because ConvNeXt is purely convolutional, it may not capture very long-range relationships as explicitly as transformers (no global attention). For example, correlating two objects far apart (like a distant road sign and the car's position) might rely on multiple convolution layers to connect, whereas a transformer could relate them in one attention hop. In extremely cluttered scenes, global reasoning can help separate instances, so ConvNeXt might have to rely on the detection head or FPN for that. **Class imbalance** is not solved by

ConvNeXt – like any backbone, it will reflect the training data. It needs the same remedies (data balancing, loss adjustments) as ResNet for rare classes. Another factor: ConvNeXt was mainly validated on COCO and ImageNet; any unique distribution in BDD100K (like lots of nighttime) still requires careful training (it doesn't inherently handle night, one might need to pretrain or augment for that). On mobile or edge hardware, ConvNeXt is *better* than Swin but still not as light as, say, MobileNet or YOLO's custom backbones. For some embedded scenarios, a scaled-down ConvNeXt might be needed, which could reduce accuracy. Summarily, ConvNeXt offers a **powerful backbone choice** that yields top-tier accuracy close to transformers but with more efficient convolutional inference – an excellent fit if one wants high accuracy in detection without completely sacrificing deployability. It still requires dealing with dataset issues via training strategies, and its largest models remain resource-intensive.

Other Noteworthy Models (nuScenes/KITTI and Real-Time Detectors)

In addition to the above models (many of which were evaluated on BDD100K), there are other detectors proven in autonomous driving datasets (like KITTI, nuScenes) that could be adapted to BDD100K:

- **YOLO Family (YOLOv3/v4/v5/v7, etc):** These one-stage *You Only Look Once* models are famous for **real-time performance**. YOLO detectors use a single network to predict bounding boxes and class confidences on a grid. Later versions like YOLOv5 and YOLOv7 have advanced backbone and head designs (e.g. CSP modules, PAFPN, decoupled heads) to boost accuracy while retaining speed. For driving tasks, YOLO models have been very popular due to their speed – e.g. YOLOv4 could run ~30 FPS on 1080p images on a GPU, and YOLOv7-tiny can even run on embedded devices (9 FPS on a Jetson Nano with ~38% mAP@0.5) [75]. In terms of accuracy, YOLOv7 (normal model, 37M params) achieves ~51.2% AP (0.5:0.95) on COCO [10], which is on par with many research models. Its optimized design gives **best-in-class speed-accuracy tradeoff** – *"YOLOv7 (51.2% AP) is 1.5× faster and +6.3% AP more accurate than YOLOv5s6 (44.9% AP)"* [10], and it even outperforms some slower detectors. For BDD100K, YOLO models can be trained to detect vehicles, pedestrians, signs, etc. The latest YOLO versions also integrate aspects like **attention** (YOLOv7 has model scaling and concatenation-based FPN) and **task-specific heads** that help with small object detection. However, YOLO's inherent single-scale grid might struggle with extremely small objects in high-res images – mitigations like mosaic augmentation and higher input resolution are often used. Also, YOLO typically uses **AP@0.5** as a primary metric (PASCAL VOC style); their AP@0.75 (stricter) can lag two-stage models slightly, since they prioritize recall and speed. Still, YOLO has been adapted for night-time vehicle detection with success – e.g. an enhanced YOLOv7 achieved 94.9% AP@50 on a nighttime vehicle subset, outperforming standard YOLOv7 by 6.6% [76]. This indicates with domain-specific tweaks (like improved low-light preprocessing), YOLO can be effective in adverse conditions. Overall, YOLO models are ideal when **real-time inference** is required, and modern versions are accurate enough to compete with many slower models. They are a top choice for an **edge deployment model** in a car that needs fast, on-camera object detection. The downside is they may need careful tuning for class imbalance (e.g. adjusting anchor auto-learning or loss for rare classes) and for extremely small objects one might increase the input size or use super-resolution.

- **EfficientDet:** This family of detectors (EfficientDet-D0 to D7) was built around EfficientNet backbones and a custom **BiFPN** feature pyramid with weighted feature fusion. EfficientDet models were designed via neural architecture search to optimize the accuracy vs. computation trade-off. For example, EfficientDet-D4 achieved about 43% mAP on COCO at ~30 BFlops, and EfficientDet-D0 got ~34% mAP at <3 BFlops – extremely lightweight [77]. These models could be useful for BDD100K when running on limited hardware (e.g., a mobile CPU in a dashcam). They handle **scale variation** well due to the BiFPN (which iteratively refines multi-scale features) and use **focal loss** to cope with class imbalance. EfficientDet isn't as dominant in accuracy now

(surpassed by YOLOv5/v7 and others), but it is still relevant for devices where EfficientNet backbones are optimized. In driving tasks, EfficientDet can be trained to detect traffic objects and might offer a good balance of ~{mid 30s to 40s} mAP with much lower model size than a ResNet+FPN. That said, EfficientDet's implementations can be complex and sometimes less flexible with custom classes (due to the NAS-crafted nature). It shines when a pre-trained model can be fine-tuned. For **small objects**, EfficientDet at higher compound scales (D6, D7) had very high AP_S on COCO thanks to huge input resolution – but those are heavy. The smaller ones may miss tiny objects. Also, EfficientDet uses anchor-based regression; handling of extremely tiny objects might require adjusting anchor scales.

- **SSD (Single Shot MultiBox Detector):** SSD was an earlier one-stage model (pre-dating RetinaNet) that was very popular on KITTI and other simpler datasets. It runs fast and directly outputs detections from multiple feature map resolutions. Nowadays, SSD is generally less accurate than RetinaNet/YOLO (due to lack of focal loss, etc.), but it's simple and light. A modern variant is *SSD-lite with MobileNet* backbone, which could run on mobile at real-time but at the cost of accuracy (perhaps AP in the 20-30% range on BDD-like data). It might be used for applications like an embedded system that just needs rough detection. However, given the availability of YOLOv5/v8 which have small models (YOLOv5n, YOLOv8n) with better accuracy, SSD is less recommended unless there are legacy constraints.

- **3D and Multi-Sensor Models:** Although BDD100K detection is 2D, knowledge from 3D detectors on nuScenes/KITTI can inspire adaptations. For example, *CenterNet* (an anchor-free model predicting object centers) was used for both 2D and 3D detection and could be applied to BDD images to detect objects by keypoints (useful for small items like traffic lights). Also, methods like **CBGS (Class-Balanced Grouping and Sampling)** from nuScenes detection [73] could be applied to BDD100K training to address its class imbalance – this method groups training examples by class frequency to ensure each mini-batch has a balanced mix, which improved detection of rare classes in nuScenes. One could incorporate such a strategy regardless of the base model.

- **Hybrid Models (e.g. Multi-task or domain adaptation):** Some research, especially for driving, combine detection with other tasks (lane detection, depth estimation) to improve robustness. While not explicitly listed, a model that leverages **multitask learning** might help BDD100K (since it has multiple labels). For instance, a detector that also predicts drivable area could use that contextual cue to better detect occluded pedestrians on the road. Heterogeneous multitask training was shown in the BDD100K paper to improve detection a bit by sharing features [78]. These are more complex to implement but worth noting.

Below, we compile a **comparison of models** on relevant benchmarks to illustrate their performance and trade-offs:

Performance Benchmarks (mAP, AP50, AP75)

The table shows **mean Average Precision (mAP)** at IoU 0.5:0.95 (COCO-style), as well as AP@50 and AP@75 for various models. Wherever possible, results on BDD100K are noted; otherwise COCO results are given as a proxy (BDD100K being similarly challenging with many small objects and clutter, we expect relative rankings to be similar).

Model (Backbone)	mAP (0.5:0.95)	AP@50	AP@75	Speed (FPS)
Faster R-CNN (Res50 FPN)	36–37% [12]	~58% [9]	~40% [9]	~5 FPS (GPU) [9]
RetinaNet (Res50 FPN)	~36% [12]	~55%	~38%	~8–10 FPS (GPU)
Cascade R-CNN (Res50)	**40%** [18]	**59%** [9]	**44%** [9]	3–4 FPS (GPU)
FCOS (Res50)	38% [22] [23]	57%	41%	~10 FPS (GPU)
ATSS (Res101 w/ DCN)	47.7% [25]	66% [25]	51% [25]	~8 FPS (GPU) (same as Retina)
Sparse R-CNN (Res50)	~45% [45] [49]	~63%	~49%	~5 FPS (GPU)
DyHead (ResNeXt-101-DCN)	54.0% [53]	71%	59%	4–5 FPS (GPU) (head only overhead negligible)
Swin Transformer (Cascade Mask R-CNN, Swin-L)	**58.7%** [55]	~78%	**63%**	~1 FPS (GPU) (very heavy)
ConvNeXt (Cascade Mask R-CNN, ConvNeXt-B)	51–52% [71]	~72%	~56%	~2 FPS (GPU)
PVTv2 (RetinaNet, PVTv2-B2)	45% [56] (est.)	~65%	49%	~8 FPS (GPU)
YOLOv7 (normal model @640)	51.2% [10]	69% [79]	56%	**80+ FPS** (V100 GPU) [79] [10]
YOLOv7-Tiny (320)	~37.8% [75]	55%	40%	**>150 FPS** (V100) [80] / (9 FPS Jetson) [75]
EfficientDet-D4 (EffNet-B4)	43% [53]	~62%	47%	~8 FPS (GPU)
Mask R-CNN (Res50) on **BDD100K**	~22% [7]	40.5% [7]	20.5% [7]	~4 FPS (GPU)

(Table notes: Metrics for COCO unless otherwise specified. BDD100K baseline result from the dataset paper [7] shows the challenge (Mask R-CNN R50 got only 21.8 AP). "Speed" is an approximation for 1024×512 images: actual FPS varies with hardware and input size. Bold indicates best in column among the listed models. YOLOv7's AP50 of 56.8% at >30 FPS is noted in Ultralytics docs [79] .)

From the table, we see a trend: modern architectures like Cascade R-CNN, ATSS, DyHead, Swin, ConvNeXt achieve **higher AP**, especially at strict IoUs (AP75), but they are generally slower. Classic models (Faster R-CNN, RetinaNet) have decent AP50 but much lower AP75 (they detect objects but with less precise localization), and they are mid-range in speed. YOLOv7 stands out as offering relatively high AP at extremely high speeds – making it a prime candidate where realtime inference is needed. On BDD100K, absolute mAP values are lower (often half of COCO's for the same model, as seen with Mask R-CNN example), due to the difficulty of the dataset – but a model that is +5 AP on COCO typically also yields significant improvements on BDD100K. For instance, if Cascade R-CNN is ~3-4 points better than Faster R-CNN on COCO, we expect a similar gap on BDD (e.g., if Faster R-CNN got ~22 AP, Cascade might

get ~25-26 AP on BDD given the same settings). Indeed, the BDD100K model zoo evaluations found Cascade + advanced backbones to outperform plain detectors [73].

Top Picks for BDD100K: Best Accuracy vs. Real-Time

Considering the above, we identify two recommended models for BDD100K use cases: one maximizing accuracy, another optimized for speed/edge deployment.

- **Best Accuracy Model – *Cascade R-CNN with Swin Transformer backbone***: This combination offers **state-of-the-art accuracy** on driving object detection. Cascade R-CNN provides superior localization and recall (important for dense traffic and strict IoU requirements), and the Swin Transformer backbone provides powerful contextual features that excel in BDD100K's varied scenes. In benchmarks, Cascade + Swin-L achieved around **58–59% mAP** on COCO [55], well above other contenders. On BDD100K, which has many small objects and nighttime scenes, we expect this model to significantly outperform others in detecting hard cases – e.g. catching far-away traffic lights or heavily occluded pedestrians that lesser models might miss. The Swin backbone can leverage global context (useful for understanding an object in headlights or amidst city clutter), and Cascade's multi-stage refinement means even if the first stage only roughly locates an object (perhaps due to blur or rain), later stages can refine it to an accurate detection. This model also benefits from any extra training data (Swin often is pretrained on large datasets, and Cascade can integrate multi-task training). The **rationale** for choosing Swin+Cascade is straightforward: it currently ranks among the top in detection accuracy for similar tasks [74]. It addresses BDD100K challenges by combining **strength in small object detection** (Swin's robust features + cascade's FPN and multiple thresholds) and **high tolerance to occlusion** (global attention can "see" an occluded object by context, cascade can refine partial proposals). The trade-off is speed – this model is heavy (~1 FPS), so it's best for cloud processing or offline analysis, not onboard real-time use. However, if **accuracy is paramount** (e.g. for validating safety or for a high-performance perception stack fusing lidar etc.), this model is the top pick. (Alternative similarly high-accuracy choices could be ConvNeXt backbone or a DyHead head on Cascade – those are in the same league. For instance, Cascade R-CNN with ConvNeXt-B was noted for strong performance [73], and DyHead with a ResNeXt-101 backbone reached 54 AP [53]. But Swin+Cascade remains a gold-standard in 2025 for vision-only detection.)

- **Best Real-Time/Edge Model – *YOLOv7 (or YOLOv8) detector***: For real-time deployment on an embedded system or as part of an on-vehicle perception stack, the YOLO family is ideal. **YOLOv7** in particular provides an excellent balance: it surpasses previous YOLO versions and other real-time models in accuracy (51.2% mAP) while running at **30–60 FPS on common hardware** [81]. A smaller variant (YOLOv7-tiny or the newer YOLOv8-nano) can reach even higher FPS (150+ on GPU, or 8-10 FPS on CPU) with lower but still respectable accuracy [82]. The rationale for YOLO is that *it was designed for speed* – it's a one-stage network with optimized architecture (CSP connectivity, EFM PAFPN, etc.) and computation graph conducive to GPU parallelism. In a driving scenario, detection needs to be run at ~30Hz to react timely; YOLOv7 can achieve this without needing multiple models. Moreover, YOLO's simpler one-stage approach uses less memory – crucial for edge devices. Regarding accuracy, while YOLO won't match a Cascade+Swin, it is **surprisingly strong on AP50** (it has high recall). On BDD100K specifically, YOLOv5/7 have been used successfully for traffic object detection, often achieving >80% AP@50 on their custom datasets [83] [76]. With proper training on BDD100K (and possibly leveraging pretrained COCO weights), YOLO can detect cars, people, signs with good precision. It may struggle a bit on the tiniest objects (some distant traffic lights might be missed) and on the rarest classes (train, given very few examples – one might practically ignore train or augment the dataset). Nonetheless, YOLO's design with multi-scale output and mosaic augmentation is aimed to improve small-

object detection and robust training. It's also easier to deploy – many frameworks (TensorRT, OpenVINO, etc.) have built-in support for YOLO models, ensuring one can squeeze maximum speed out of them. **In summary**, YOLOv7 is recommended for an application that needs **fast, near-real-time perception** with decent accuracy – for instance, an onboard DAS (Driver Assist System) that runs on GPU or even on high-end ASIC could use YOLOv7 to detect obstacles and traffic signals on-the-fly. The rationale is that YOLOv7 offers **the best FPS/AP trade-off** in 2025 – it "surpasses all known object detectors in both speed and accuracy in the range 5 FPS to 160 FPS" [80], meaning it's dominantly efficient. This makes it ideal for edge deployment where computational resources are limited but decisions must be made quickly.

(Note: Another contender for real-time could be EfficientDet-D3 or YOLOv5 on TensorRT, but YOLOv7 generally outperforms YOLOv5 for the same latency [10]. If extremely constrained, one might pick YOLOv5-Nano or even an upright MobileNet-SSD which can run on CPUs – but their accuracy on BDD100K would be much lower, likely missing many small signs or distant pedestrians. YOLOv7 provides a sweet spot of moderate accuracy and high speed.)

Inference Optimization Techniques

Even after choosing an architecture, there are several **model optimization methods** to further improve inference speed or reduce model size **without severely sacrificing accuracy**. These techniques are crucial for deploying high-performing models in real-world driving systems where latency and resource constraints are critical:

- **Quantization:** This involves reducing the precision of model parameters and operations (e.g. from 32-bit floating point to 8-bit integers). **Post-training quantization** or **quantization-aware training** can dramatically speed up inference and cut memory usage by ~4×, often with minimal impact on mAP. For example, an 8-bit quantized YOLOv5m saw negligible AP drop but ~2× faster CPU inference. Quantization works well on CNN-based models and many accelerators (TPUs, NN cores in GPUs) are optimized for INT8 ops. For driving models, one can use quantization-aware training on BDD100K – essentially simulate int8 during training – to preserve accuracy (often within ~1% mAP of FP32) while deploying an INT8 model. Models like Faster R-CNN and SSD have been quantized successfully with <1 mAP drop [84]. One must ensure calibration includes a diverse set of images (covering night, rain, etc.) so that quantization scales are tuned appropriately. Overall, quantization is a quick win for edge deployment: it **speeds up compute bound operations** and reduces memory bandwidth – both crucial for real-time performance – with only a slight accuracy trade-off if done right.

- **Pruning (Model Pruning):** Pruning removes redundant or less-important weights/neurons from the network. Techniques include **channel pruning/structured pruning** (removing entire convolution channels or layers) and **unstructured pruning** (zeroing individual weights). For object detection, structured pruning is preferred since it results in actual speed-ups (unstructured pruning creates sparse weights which aren't always faster on hardware). A common approach is to prune e.g. 30% of filters that have the lowest contribution (based on magnitude or a learned importance) and then fine-tune the model on the dataset to recover accuracy. In practice, detectors can usually be pruned by 20–30% with minor loss in mAP – for instance, one could prune a YOLOv7 model to reduce its FLOPs and get maybe 1–2% AP drop but a significant speed gain. Especially if the model is over-parameterized for BDD100K, pruning can remove overfitting capacity. There are also autoML approaches that prune and then re-train the model (iterative pruning or "lottery ticket" style) to find a smaller subnetwork. For embedded deployment, pruning can help fit models into smaller memory (important for FPGA/ASIC with

limited SRAM). One has to be careful to maintain the balance between classes – e.g. ensure layers critical for rare class detection aren't pruned too aggressively. But overall, **pruning can slim a model** to meet latency targets with minimal accuracy hit, making it very useful for optimizing heavy models like Cascade+Swin (which could possibly drop some channels in early layers to speed up) or to compress a model for lower compute hardware.

- **Knowledge Distillation:** This technique trains a **smaller "student" model** to mimic the outputs (logits or feature maps) of a **larger "teacher" model**. In detection, one can have a high-accuracy teacher (say Cascade Swin or a big ensemble) generate predictions or softened labels for the training images, and train a lighter model (maybe YOLO or a smaller backbone) to reproduce those. The student thus learns from both ground truth and teacher's rich predictions, often resulting in higher accuracy than training the student from scratch – effectively **recovering some of the teacher's performance**. In driving scenarios, distillation can be used to **transfer knowledge of hard-to-detect objects**: e.g. a heavy teacher might detect very small pedestrians or distant signs that are consistently missed by a small model; by distillation, the small model can learn to predict those (even if they aren't labeled or aren't learned via standard training). There have been works where a RetinaNet learned from a Cascade Mask R-CNN teacher and gained a few AP points [73]. Similarly, one could distill a Swin-transformer-based detector into a Mobilenet-SSD, improving the latter's AP for only a bit of added training cost. The benefit is **improving a fast model's accuracy** without changing its architecture or runtime. This is very valuable when you've picked a real-time model (like YOLO-tiny) but want to claw back some accuracy – distillation might give +2-3 mAP, which could make a difference in detecting that last 5% of rare events. It's essentially leveraging the teacher's **knowledge of context and subtle cues** to inform the student. For BDD100K, one could take the "best accuracy model" (Cascade+Swin) as teacher and distill its predictions into the "real-time model" (YOLO) – aiming to get the YOLO to perform better on small and occluded objects than it otherwise would. This approach can significantly close the gap between big and small models on the BDD100K task.

- **Model Architecture Tuning & NAS:** Techniques like **Neural Architecture Search (NAS)** or manual architecture tuning can craft a more efficient model for the specific task. For example, one could use NAS to find the optimal depth, width, and layer types for BDD100K detection under latency constraints. EfficientDet was a product of NAS, and YOLOv7 also applied some automated search for its network scaling. In a practical sense, using a slightly smaller backbone (ResNet-18 or RegNet) or a one-stage head instead of two-stage can be seen as architecture tuning to meet requirements. If our chosen model is too slow, we might switch out Swin-L for Swin-T (trading ~1-2% mAP for 2× speed), or reduce the cascade stages from 3 to 2. These are forms of architecture optimization. The goal is to simplify the model until it just meets the needed accuracy. Tools like *TensorRT* can also suggest layer fusion or precision changes to speed up the model – which ties in with quantization and minor architecture adjustments (like replacing certain operations with faster equivalents). One example: replacing a standard Non-Maximum Suppression with a faster greedy NMS or a learned NMS can cut post-processing time.

- **Pipeline and Parallelism Optimizations:** Outside the model weights, one can optimize how the model is executed. For instance, doing **batch inference** (if latency allows buffering a few frames) can increase throughput on GPU. Or running the backbone once for two tasks (like detection and segmentation together) to amortize cost. In a driving pipeline, one could run a heavy model at a lower frame-rate for high accuracy and a light model every frame – this kind of multi-scale pipeline can be considered an optimization strategy (trading off temporal resolution vs quality). While not a model modification per se, it's a system-level optimization that often helps in practice (e.g., run Cascade R-CNN at 5 Hz and YOLO at 30 Hz, and combine results).

- **Hardware-specific Optimizations:** Using frameworks such as NVIDIA TensorRT, OpenVINO, or ONNX Runtime with graph optimizations can yield significant speed-ups. These perform operations like layer fusion (combining consecutive conv+BN+activation into one kernel), memory optimizations, and use low-level libraries for faster matrix ops. Ensuring the model is exported and optimized in such a way can sometimes double the FPS without any model change. For example, YOLOv5 on TensorRT INT8 ran about 2× faster than the raw PyTorch model at only ~1% loss of mAP [10]. Similarly, compiling models on DL accelerators (TVM, etc.) can give boosts. These are essentially mandatory steps to deploy efficiently but require no change in model architecture, just careful integration and calibration.

In conclusion, by applying **quantization** and **pruning**, one can often achieve an order-of-magnitude improvement in inference efficiency – e.g. going from a 200ms model to a 50ms model – while losing only a few points of AP. **Distillation** can recover those lost AP points in the smaller model by teaching it the large model's tricks. Together, these optimization methods enable us to take a high-performing but bulky model and compress/speed it up to meet real-time constraints, **with minimal hit to accuracy**. This is crucial in autonomous driving, where every millisecond saved gives the car more time to react, but detection errors can be critical – so we strive to maintain accuracy as we optimize. By leveraging these methods, a deployment might use, say, an 8-bit quantized, pruned YOLO model distilled from a Cascade RCNN – running efficiently on an automotive-grade SoC and still performing robustly on the BDD100K tasks.

References: The comparative insights and data are drawn from the BDD100K dataset analysis and detector benchmark results [3] [7], as well as from published papers introducing these models [12] [19] [53] [10]. Each model's strengths (small object handling, occlusion robustness, etc.) are backed by findings in the respective papers or subsequent evaluations [8] [85] [73]. The performance metrics cited (mAP/AP50/AP75) are from COCO or similar evaluations to give a sense of relative performance [19] [55]. These provide a baseline to expect how models would rank on BDD100K, given its challenge (e.g. Mask R-CNN baseline ~22 AP on BDD vs 37 AP on COCO) [7]. The recommended models were chosen based on these comparisons – Cascade+Swin for maximum accuracy [74], and YOLOv7 for speed [10] – aligning with current state-of-the-art in 2025 for those criteria. The optimization techniques discussed are well-supported by literature and industry practice, ensuring that high accuracy can be coupled with deployable speed for autonomous driving applications.

---

[1] [2] [3] [5] analysis_report_val.md
file://file-GiTDX1xu9wg5cyCsHBz1SX

[4] detailed_analysis_report_val.txt
file://file-D2XKD5CJPECYPaYCkxVTFU

[6] [7] [78] BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning
https://openaccess.thecvf.com/content_CVPR_2020/papers/Yu_BDD100K_A_Diverse_Driving_Dataset_for_Heterogeneous_Multitask_Learning_CVPR_2020_paper.pdf

[8] Object Detection in Autonomous Driving Scenarios Based on an Improved Faster-RCNN
https://www.mdpi.com/2076-3417/11/24/11630

[9] [15] [16] [17] [18] arxiv.org
https://arxiv.org/pdf/1906.09756

[10] YOLOv7 vs YOLOv5 reproducible comparison of accuracy and speed
https://github.com/WongKinYiu/yolov7/issues/395

11 12 13 14 31 32 33 34 85 Libra R-CNN: Towards Balanced Learning for Object Detection
https://openaccess.thecvf.com/content_CVPR_2019/papers/Pang_Libra_R-CNN_Towards_Balanced_Learning_for_Object_Detection_CVPR_2019_paper.pdf

19 22 23 24 25 26 27 28 40 41 42 43 44 [19.12] ATSS | DOCSAID
https://docsaid.org/en/papers/object-detection/atss/

20 21 29 30 50 51 52 53 54 77 Dynamic Head: Unifying Object Detection Heads With Attentions
https://openaccess.thecvf.com/content/CVPR2021/papers/Dai_Dynamic_Head_Unifying_Object_Detection_Heads_With_Attentions_CVPR_2021_paper.pdf

35 High-Resolution Network: A universal neural architecture for visual ...
https://www.microsoft.com/en-us/research/blog/high-resolution-network-a-universal-neural-architecture-for-visual-recognition/

36 HRNet/HRNet-Object-Detection: Object detection with multi ... - GitHub
https://github.com/HRNet/HRNet-Object-Detection

37 38 Object detection performance on COCO minival * on ResNet models ...
https://www.researchgate.net/figure/Object-detection-performance-on-COCO-minival-on-ResNet-models-with-different-number-of_fig2_318337409

39 HRNet: High-Resolution Network for Computer Vision - SERP AI
https://serp.ai/posts/hrnet/

45 46 47 [2205.02101] Dynamic Sparse R-CNN - arXiv
https://arxiv.org/abs/2205.02101

48 [PDF] Sparse R-CNN: End-to-End Object Detection With Learnable ...
https://openaccess.thecvf.com/content/CVPR2021/papers/Sun_Sparse_R-CNN_End-to-End_Object_Detection_With_Learnable_Proposals_CVPR_2021_paper.pdf

49 Apply prior feature integration to sparse object detectors
https://www.sciencedirect.com/science/article/abs/pii/S0031320324008549

55 This is an official implementation for "Swin Transformer - GitHub
https://github.com/microsoft/Swin-Transformer

56 IntelPVT: intelligent patch-based pyramid vision transformers for ...
https://link.springer.com/article/10.1007/s13042-023-01996-2

57 58 59 60 61 62 63 64 65 Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction Without Convolutions
https://openaccess.thecvf.com/content/ICCV2021/papers/Wang_Pyramid_Vision_Transformer_A_Versatile_Backbone_for_Dense_Prediction_Without_ICCV_2021_paper.pdf

66 69 ResNet Strikes Back: An Improved Training Procedure in timm | by ...
https://sh-tsang.medium.com/review-resnet-strikes-back-an-improved-training-procedure-in-timm-f73dcf817067

67 Which one are the best Densnet or Resnet? - TensorFlow
https://discuss.ai.google.dev/t/which-one-are-the-best-densnet-or-resnet/31108

68 shinya7y/UniverseNet: USB: Universal-Scale Object ... - GitHub
https://github.com/shinya7y/UniverseNet

70 [PDF] Everything is All You Need: Vision Architectures for the 2020s
https://www.sainingxie.com/pdf/CVPR_t4vworkshop_clean.pdf

71 COCO object detection and segmentation results using Mask-RCNN ...
https://www.researchgate.net/figure/COCO-object-detection-and-segmentation-results-using-Mask-RCNN-with-backbone-ConvNeXt-T_tbl2_364127031

72 [PDF] ConvNeXt V2: Co-Designing and Scaling ConvNets With Masked ...
https://openaccess.thecvf.com/content/CVPR2023/papers/Woo_ConvNeXt_V2_Co-Designing_and_Scaling_ConvNets_With_Masked_Autoencoders_CVPR_2023_paper.pdf

73 74 weisongshi.org
https://weisongshi.org/papers/liu24-CPT.pdf

75 [PDF] ECF-YOLOv7-Tiny: Improving Feature Fusion and the Receptive ...
https://openaccess.thecvf.com/content/WACV2025/papers/Bacea_ECF-YOLOv7-Tiny_Improving_Feature_Fusion_and_the_Receptive_Field_for_Lightweight_WACV_2025_paper.pdf

76 83 Detection results of dark scene from BDD100K. (a) Faster R-CNN, (b)...
https://www.researchgate.net/figure/Detection-results-of-dark-scene-from-BDD100K-a-Faster-R-CNN-b-Centernet-c_fig6_376913219

79 81 YOLOv7: Trainable Bag-of-Freebies - Ultralytics YOLO Docs
https://docs.ultralytics.com/models/yolov7/

80 Official YOLOv7: Trainable bag-of-freebies sets new state-of-the-art ...
https://github.com/pjreddie/darknet/issues/2557

82 Improved YOLOv7-Tiny for Object Detection Based on UAV Aerial ...
https://www.mdpi.com/2079-9292/13/15/2969

84 a steel surface defect detection based on improved YOLOv7 - PMC
https://pmc.ncbi.nlm.nih.gov/articles/PMC12137579/