TensorFlow Functions for Image Generation

1. tf.image.draw_bounding_boxes

Visualizes predicted bounding boxes on images to debug results fast.
Example: tf.image.draw_bounding_boxes(img, boxes, colors)
Why: If you can't visualize boxes, you're flying blind.

2. tf.image.non_max_suppression

Removes overlapping duplicate boxes keeping the best predictions.
Example: tf.image.non_max_suppression(boxes, scores, max_output_size=50)
Why: Every real detector needs NMS or outputs look like spaghetti.

3. tf.image.resize

Resizes input frames before feeding to a model (YOLO, SSD, etc.).
Example: tf.image.resize(img, (300, 300))
Why: Detection models demand fixed resolutions.

4. tf.image.combined_non_max_suppression

Batch-friendly NMS for modern detectors (multi-class + multi-box).
Example: tf.image.combined_non_max_suppression(boxes, scores, ...)
Why: Used in production-grade TF models like EfficientDet.

5. tf.keras.applications.EfficientNetB0 *(as backbone)*

Used as a feature extractor for detectors (EfficientDet).
Example: backbone = EfficientNetB0(include_top=False)
Why: Detection needs good feature maps, not just logits.

6. tf.keras.layers.Conv2D

Core layer for anchor feature generation + box regressions.
Example: layers.Conv2D(256, 3, padding="same")
Why: Detection heads = small conv towers.

7. tf.keras.layers.UpSampling2D

Upscales feature maps to build feature pyramids.
Example: layers.UpSampling2D(size=2)
Why: Multi-scale detection matters (small objects are annoying).

8. tf.keras.layers.Concatenate

Merges feature maps across scales (FPN/biFPN style).
Example: layers.Concatenate()([p3, p4])
Why: No concatenation → multi-scale detection dies.

9. tf.image.random_brightness *(augmentation)*

Adds brightness noise to strengthen robustness.
Example: tf.image.random_brightness(img, 0.3)
Why: Real-world lighting sucks; augment for it.

10. tf.image.random_flip_left_right *(augmentation)*

Flips objects + boxes together during training.
Example: tf.image.random_flip_left_right(img)
Why: Cheap augmentation that avoids overfitting.