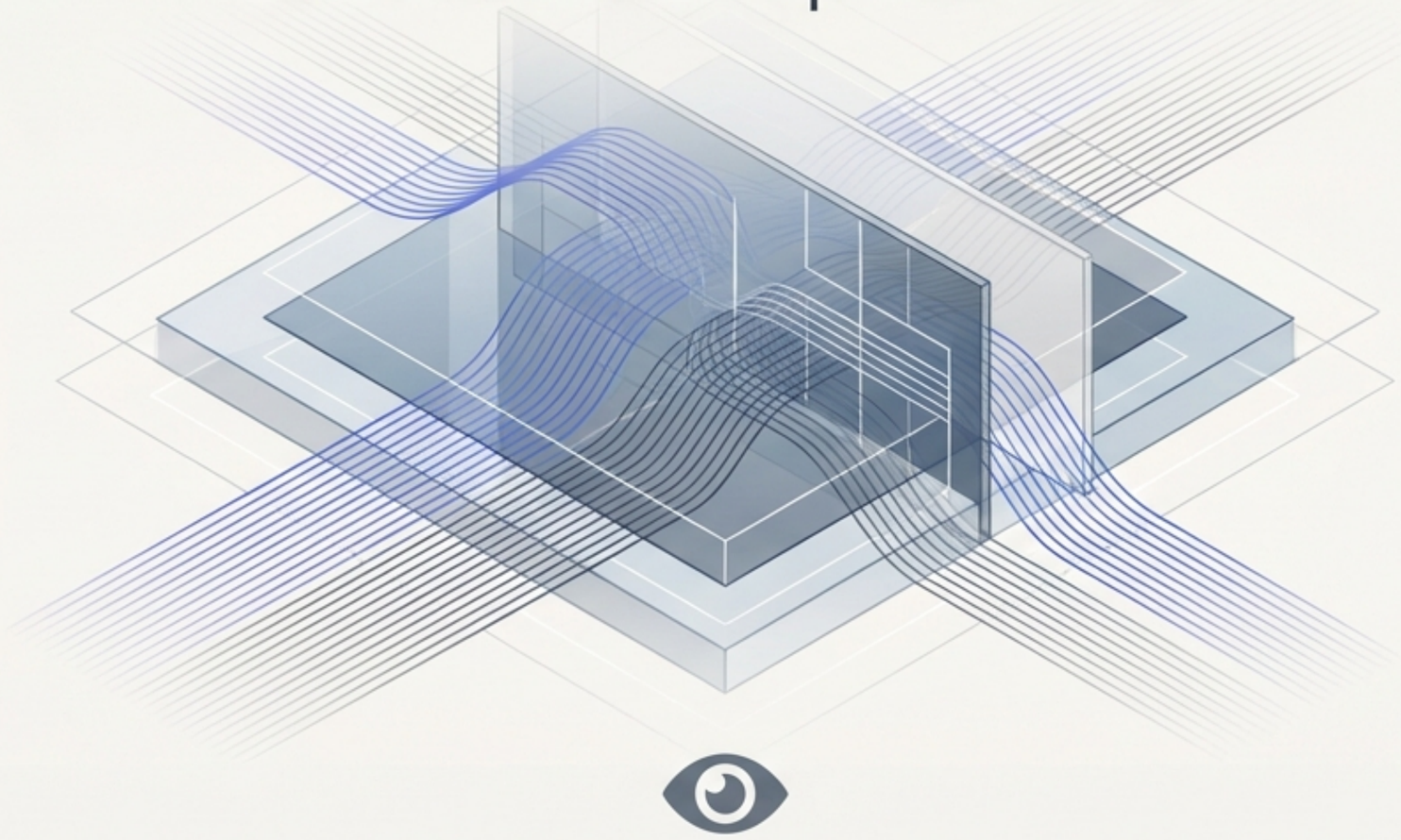
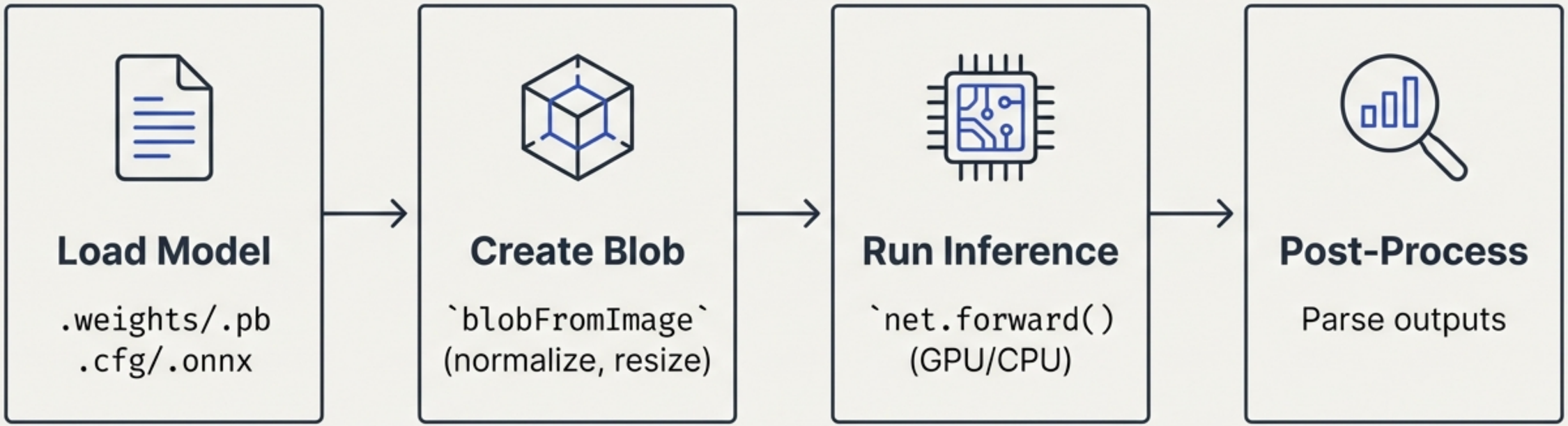
From Model to Magic

Your Step-by-Step Guide to High-Performance Inference with OpenCV DNN



The OpenCV DNN Inference Pipeline: Our Roadmap



OpenCV handles framework differences:
TensorFlow ⟷ Caffe ⟷ ONNX ⟷ Darknet ⟷ PyTorch

Step 1: Unifying Your Model Zoo

OpenCV's `cv2.dnn.readNet()` automatically detects and loads models from the most popular deep learning frameworks.

Supported Frameworks

Framework	Model File	Config File
TensorFlow	`.pb`	`.pbtxt` (optional)
Caffe	`.caffemodel`	`.prototxt`
Darknet/YOLO	`.weights`	`.cfg`
ONNX	`.onnx`	-

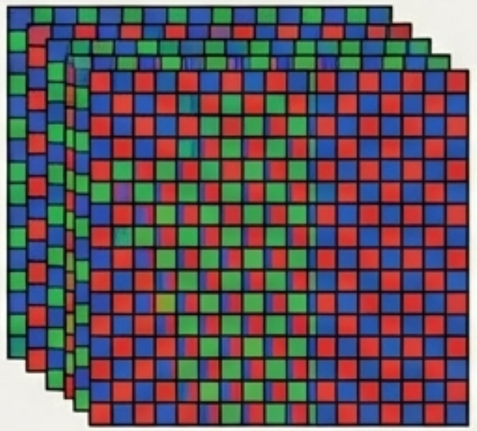
**PyTorch models are supported via ONNX export.*

```
net = cv2.dnn.readNet('model.weights', 'model.cfg')
```

The Crucial Bridge: What Exactly is a ‘Blob’?

A blob is a 4D tensor that standardizes your input image for the neural network. The `blobFromImage` function handles this complex conversion for you.

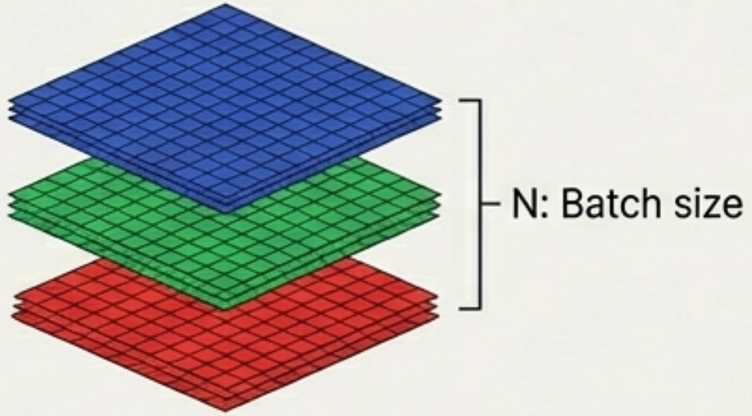
Input Image (HWC)



blobFromImage()

- resize
- scale
- mean subtract
- swap R↔B

Output Blob (NCHW)



****NCHW Explanation****

N: Batch size (usually 1)
C: Channels (3 for RGB)
H: Height
W: Width

****Example Transformation****

Shape: `(480, 640, 3)` → `(1, 3, 224, 224)`
Range: `[0, 255]` → `[0.0, 1.0]`

```
blob = cv2.dnn.blobFromImage(image, scalefactor=1.0, size=(224, 224), mean=(104, 117, 123), swapRB=True, crop=False)
```

Mastering the Blob: A `blobFromImage` Deep Dive

The Function Signature

```
blob = cv2.dnn.blobFromImage(
    image,        # Input image (OpenCV reads as BGR)
    scalefactor, # e.g., 1/255.0 to normalize to [0,1]
    size,         # Target spatial dimensions (width, height)
    mean,         # Per-channel mean subtraction values (B, G, R)
    swapRB,       # Set True to convert BGR image to RGB blob
    crop          # Crop image after resizing
)
```

Common Preprocessing Presets

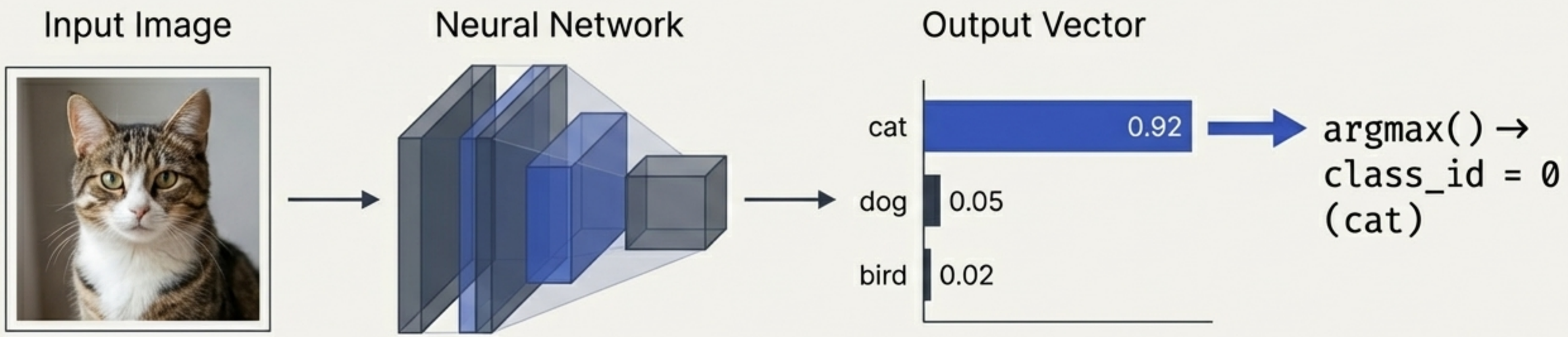
Model	scalefactor	size	mean	swapRB
ImageNet	1/255	(224, 224)	(0, 0, 0)	True
VGG	1.0	(224, 224)	(103.939, 116.779, 123.68)	False
SSD	1.0	(300, 300)	(104, 177, 123)	False
YOLO	1/255	(416, 416)	(0, 0, 0)	True

The Core Inference Workflow in Code

```
# 1. Load the model from disk
net = cv2.dnn.readNet('model.weights', 'model.cfg')

# 2. (Optional) Set backend and target for acceleration
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)

# 3. Prepare the input image by creating a blob
blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416), swapRB=True)

# 4. Set the blob as the input to the network
net.setInput(blob)

# 5. Perform a forward pass to get the network's output
# For models with multiple output layers (like YOLO)
layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]
outputs = net.forward(output_layers)

# 6. Post-process the results (details on next slides)
```

Application 1: Image Classification

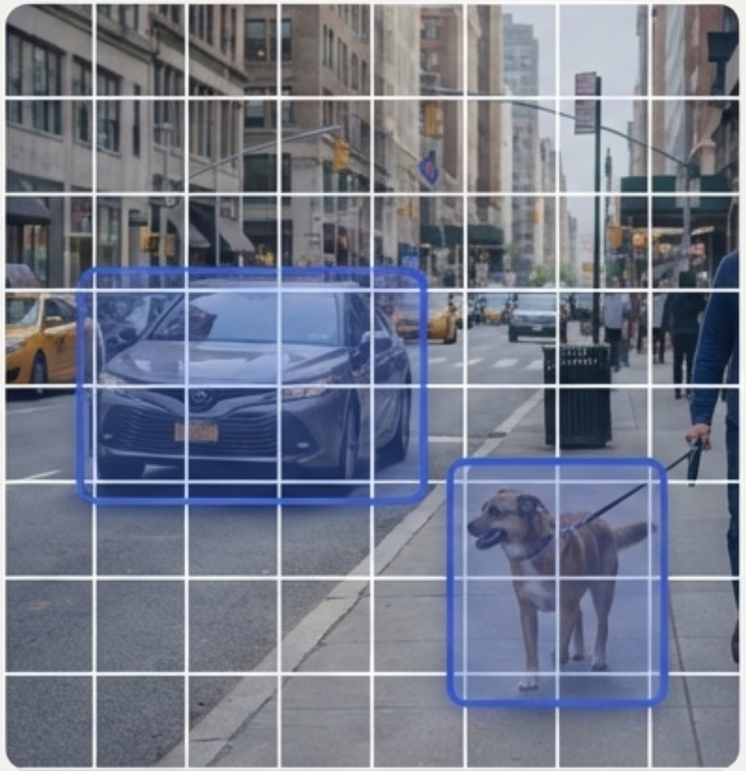
Assigning an image to one of N predefined categories.




```
# Assumes 'predictions' is the output from net.forward()
# Output shape is (1, N_classes)
class_id = np.argmax(predictions[0])
confidence = predictions[0][class_id]

print(f"Predicted class: {class_id} with confidence: {confidence:.2f}")
```

Application 2: Object Detection with YOLO

“You Only Look Once” – YOLO divides the image into a grid and predicts bounding boxes and class probabilities for each grid cell in a single pass.

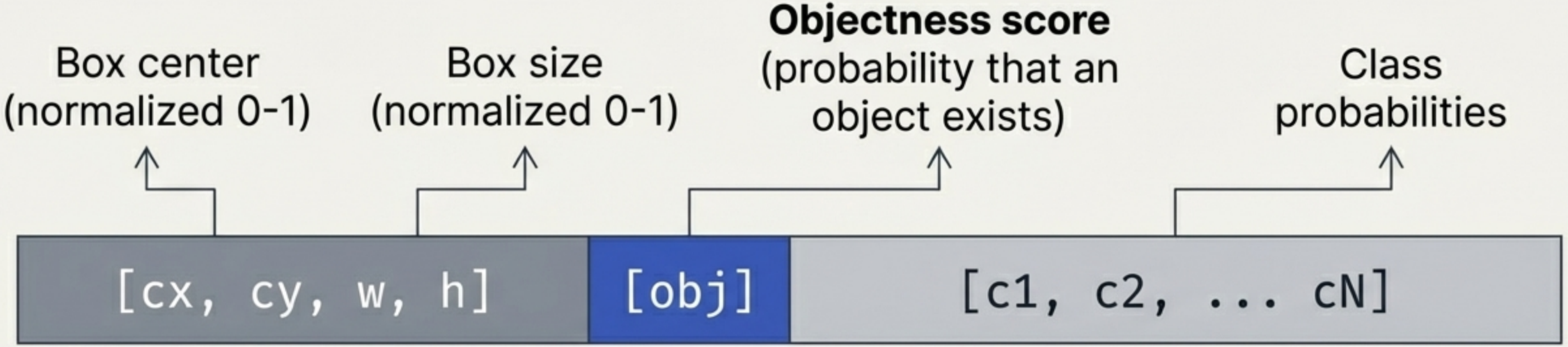


Per-Cell Prediction

- B bounding boxes
- Confidence scores for those boxes
- C class probabilities

Single forward pass → detect all objects at once (fast!)

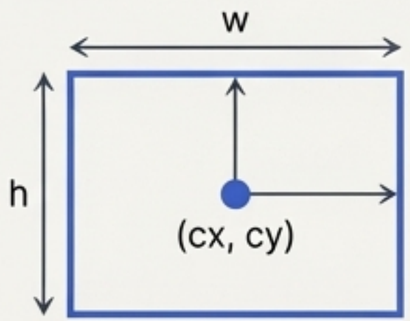
Decoding the YOLO Output Vector



Final Confidence = objectness_score × class_probability

```
# For each 'detection' vector from the network output...
scores = detection[5:]
class_id = np.argmax(scores)
confidence = scores[class_id] * detection[4] # objectness

if confidence > CONF_THRESHOLD:
    # Un-normalize coordinates and draw box...
```

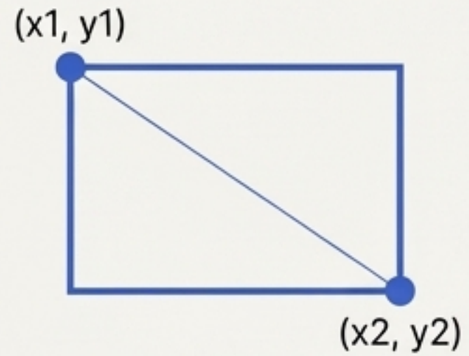
A Tale of Two Detectors: YOLO vs. SSD Output



Center + Size
(Relative Coords)

```
[cx, cy, w, h, obj, class_probs...]
```

```
center_x = det[0] * W;
w = det[2] * W;
x = center_x - w/2
```



Corner Coords
(Top-Left + Bottom-Right)

```
[batch, class, conf, x1, y1, x2, y2]
```

```
x1 = det[3] * W;
x2 = det[5] * W
```

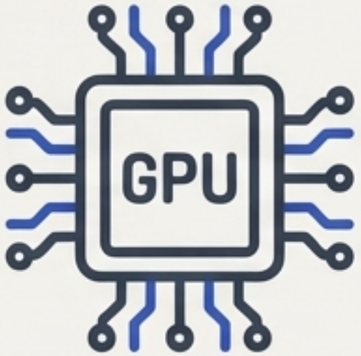
Key Differences Callout

- **YOLO**: Center coordinates + width/height.
- **SSD**: Top-left corner + bottom-right corner.
- Both are typically normaiized to [0, 1].

From Working to Winning: Performance Optimization

Measure first! Use `net.getPerfProfile()` to establish a baseline.

```
t, _ = net.getPerfProfile(); time_ms = t * 1000 / cv2.getTickFrequency()
```



Leverage Hardware Acceleration

Use a GPU or specialized backend.

```
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
```

Reduce Input Size

Smaller blobs (`(224,224)` vs `(416,416)`) mean faster inference. This is a direct trade-off with accuracy.

Process in Batches

If you have multiple images, process them at once.

```
blob = cv2.dnn.blobFromImages(images, ...)
```

FP16

Use Lower Precision

FP16 can provide a significant speedup on compatible hardware.

```
net.setPreferableTarget(cv2.dnn.DNN_TARGET_
CUDA_FP16)
```

Advanced: For ultimate performance, explore model-level optimizations like Quantization (INT8) and Pruning before deployment.

Choosing Your Champion: A Guide to toCommon Architectures

Classification Models

Model	Size	Speed	Accuracy	Best For
MobileNet	Small	Fast	Good	Mobile/Embedded
ResNet	Large	Medium	Excellent	High Accuracy
EfficientNet	Medium	Medium	Best	Balanced Perf.

Detection Models

Model	Speed	Accuracy	Best For
YOLO v3+	Fast	Good	Real-Time Apps
SSD	Fast	Good	Real-Time Apps
Faster R-CNN	Slow	Excellent	High Accuracy

Your OpenCV DNN Toolkit: The Essential Functions

Function	Description
`cv2.dnn.readNet()`	Auto-detect and load a model from any supported framework.
`cv2.dnn.blobFromImage()`	Create a 4D blob from an image for network input.
`net.setInput()`	Set the input blob for the network.
`net.forward()`	Run the forward pass to get predictions.
`net.setPreferableBackend()`	Set the computation engine (OpenCV, CUDA, OpenVINO).
`net.setPreferableTarget()`	Set the target device (CPU, GPU).
`cv2.dnn.NMSBoxes()`	Perform Non-Maximum Suppression to filter overlapping boxes.
`net.getPerfProfile()`	Get inference time for performance profiling.

For a complete list and more examples, visit the official OpenCV DNN tutorials. [docs.opencv.org]