# Table of Contents

## 1. Conversion of YOLOv8 Model into ONNX Model

**ONNX is an open format built to represent machine learning models.** ONNX defines a common set of operators - the building blocks of machine learning and deep learning models - and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers.

Why ONNX Model?

1. **Interoperability**: ONNX (Open Neural Network Exchange) format provides interoperability between different deep learning frameworks like TensorFlow, PyTorch, and MXNet. This means you can deploy the model across various platforms without worrying about compatibility issues.

2. **Efficiency**: ONNX models are optimized for inference, resulting in faster execution times compared to the original PyTorch or TensorFlow models. This efficiency is crucial for real-time applications or scenarios where latency needs to be minimized.

3. **Deployment Flexibility**: ONNX models can be deployed on various devices, including CPUs, GPUs, and even specialized hardware like FPGAs (Field-Programmable Gate Arrays) and TPUs (Tensor Processing Units). This flexibility allows for deployment in a wide range of environments, from edge devices to cloud servers.

4. **Reduced Footprint**: ONNX models typically have a smaller file size compared to their original formats. This reduced footprint is beneficial for deployment in resource-constrained environments, such as mobile devices or edge computing devices, where storage space is limited.

5. **Ecosystem Support**: ONNX has a growing ecosystem of tools and libraries for model optimization, conversion, and deployment. This ecosystem includes tools for quantization, pruning, and model compression, which can further enhance the performance and efficiency of the deployed model.

6. **Scalability**: ONNX models can easily be scaled horizontally to handle increased workloads by deploying them across multiple instances or devices in parallel. This scalability is essential for applications that experience varying levels of demand or need to process large volumes of data efficiently.

7. **Future-proofing**: By converting the model to ONNX format, you future-proof your deployment pipeline against changes in the underlying deep learning framework. If you decide to switch frameworks or upgrade to newer versions in the future, you can easily reconvert the ONNX model without needing to retrain or fine-tune it from scratch.

- **Steps -**

1. Make the Fine-Tuned YOLOv8 Model for conversion.
2. Save the 'best.pt' file (if trained using TensorFlow).
3. Load the inbuilt YOLO model (yolov8n.pt) then load custom trained YOLO model (best.pt).
4. Use the 'export' function provided in the Ultralytics Library.
5. Upon successful conversion, inference using test images.

- **Instructions for Inference -**

1. Inference variable is an Object with Attributes:
    i. boxes
    ii. names
    iii. orig_img
    iv. orig_shape
    v. path
    vi. speed

2. Predicted Bounding Box co-ordinates and Class Labels are stored in 'boxes' attribute which itself is an Object.

3. Bounding Boxes are in the form of 'xyxy' co-ordinates which stored as tensor hence needs to be converted into numpy array.

4. In case of multiple objects in single image, there will be separate BBox tensors for each detached class. Append all the co-ordinates into a single List for further use.

- **GitHub - YOLOv8 to ONNX Conversion**

## 2. Conversion of YOLOv8 Model into TFJS (TensorFlow.JS) Model

**TFJS (TensorFlow.js) model is a machine learning model that has been converted and optimized for deployment in web browsers** or Node.js environments using TensorFlow.js library. These models are typically trained in TensorFlow and then converted to formats compatible with JavaScript, allowing for in-browser or server-side inference without the need for external server calls, enabling tasks like real-time object detection or image classification directly within the browser.

Why TFJS Model?

1. **Browser Compatibility**: TFJS models can be executed directly within web browsers, enabling deployment of YOLOv8 models for tasks like real-time object detection on websites or web applications without requiring server-side inference.

2. **Client-Side Inference**: With TFJS, inference is performed locally on the client's device, reducing latency and eliminating the need for round-trip communication with a server. This is particularly advantageous for applications requiring real-time or interactive responses.

3. **Privacy and Security**: Since data remains on the client-side during inference, TFJS deployment offers enhanced privacy and security by avoiding the need to transmit sensitive information over the network to a remote server.

4. **Offline Availability**: TFJS models can be cached and executed offline, allowing applications to continue functioning even without an internet connection. This is beneficial for scenarios where connectivity is unreliable or unavailable.

5. **Ease of Deployment**: TFJS simplifies deployment by eliminating the need for server-side infrastructure to host and serve the model. This streamlines the deployment process, reduces operational overhead, and enables easier updates and maintenance of the deployed application.

6. **Cross-Platform Compatibility**: TFJS models can be deployed not only in web browsers but also in Node.js environments, offering cross-platform compatibility for deployment on both client and server-side applications.

7. **Community Support and Ecosystem**: TensorFlow.js has a growing community and ecosystem of tools and resources for model development, conversion, optimization, and deployment. This includes utilities for model conversion, performance optimization, and integration with web frameworks, enhancing the development and deployment workflow.

The ‘export’ function from Ultralytics Library provides the direct method to convert Custom Trained YOLOv8 Model into TFJS model, however, as the the official GitHub repository of Ultralytics Module, the direct conversion is not working as expected for many users and it is currently not stable and under development.

To solve this issue, we have to export the YOLOv8 Model into some intermediary format which then will be used to convert into TFJS Model. We can use SavedModel, ONNX, TFLite formats as intermediaries.
Here, we are using ONNX Model format for TFJS conversion.

**Library Used : onnx_tf**

- **Steps -**

1. Prepare the converted ONNX Model.
2. Use ‘prepare( )’ and ‘export_graph( )’ for converting ONNX to SavedModel.
3. Convert the ONNX model into SavedModel format.
4. Feed the SavedModel into the TFJS Convert Command and execute it using TensorFlowJS Module.
5. Zip the entire generated folder containing ‘model.json’ file.

- **GitHub - ONNX To TFJS Conversion**

## 3. Performance Metrics

### 1. Fine-Tuned YOLOv8 Model :-

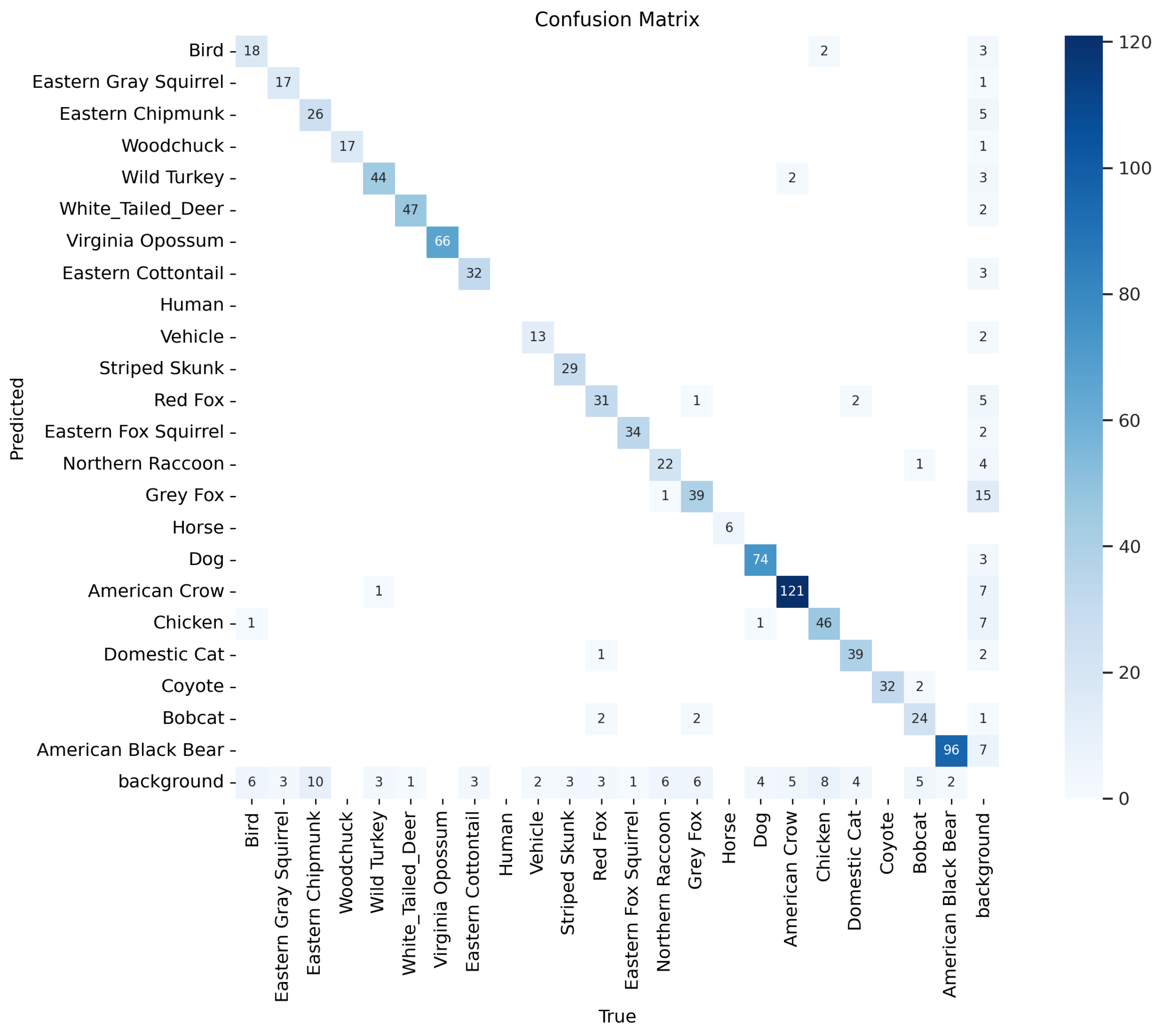

**Fig.1: Confusion Matrix**

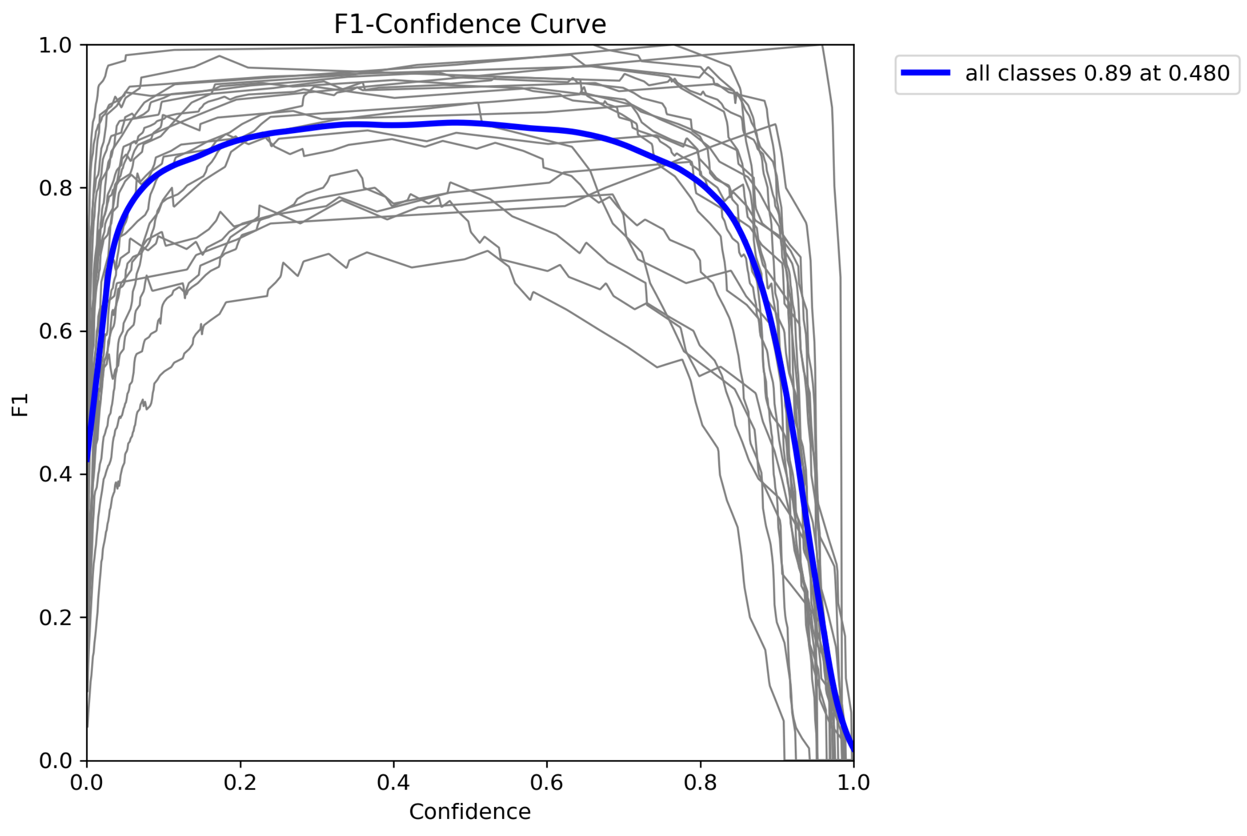

**Fig. 2: F1 Curve**

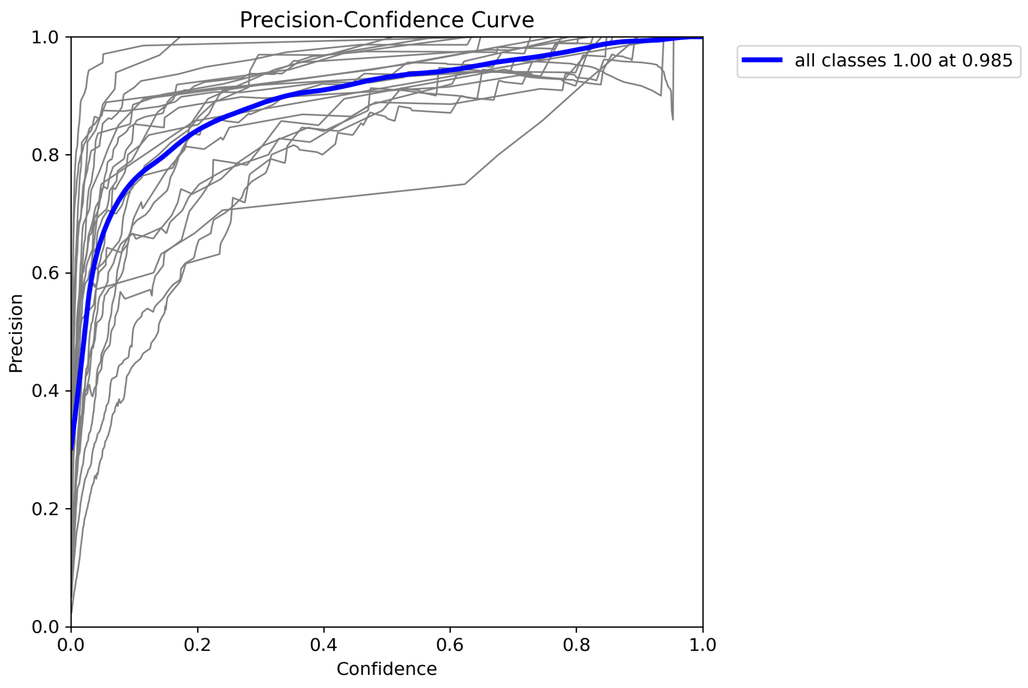

**Fig. 3: Precision Curve**

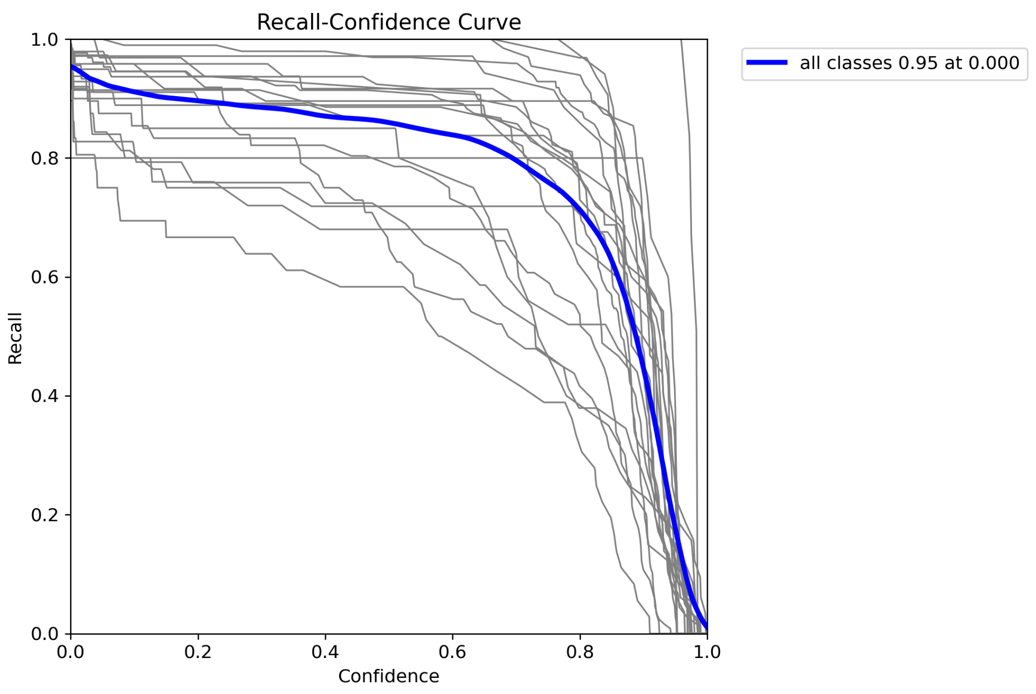

**Fig. 4: Recall Curve**

- **Metrics -**

i. **F1 Score:** 0.89
ii. **Precision:** 1.00
iii. **Recall:** 0.95
iv. **Accuracy:**

$$\frac{\text{Precision} * \text{Recall} * \text{N}}{\text{Precision} * \text{N} + \text{Recall} * \text{N} - \text{Precision} * \text{Recall} * \text{N}}$$

Hence,

**Accuracy:** 0.95 i.e. *95% approx.*

**2. Converted ONNX Model :-**

i. **Accuracy:** 55.77% or 59.39%
ii. **Average IoU Score:** 0.8608
iii. **Precision:** 54.35%
iv. **Recall:** 57.2606%
v. **F1 Score:** 0.5577

**3. Converted TFJS Model :-**

The Accuracy of TFJS Model converted from ONNX drops about 15-20 %.

- **GitHub - ONNX Model Evaluation**

## 4. YOLOv8-TFJS Live Browser App

- **Steps to REPRODUCE the results :-**

i. Convert the YOLOv8 Model to TFJS - Follow This.
ii. Copy ***yolov8*_web_model*** to ***./public*** Directory.
iii. Update ***modelName*** in ***App.jsx*** to new Model Name.

```
...
// model configs
const modelName = "yolov8*"; // change to new model name
…
```

iv. Update ***src/utils/labels.json*** with the new classes.
v. Done.

- **Steps to Run the WebApp :-**

i. Go to the ***src/App.jsx*** file.
ii. Run the command in the terminal - ***npm start***

- **GitHub - YOLOv8-TFJS Live WebApp**