Wind Speed Prediction Using GMDH Networks

Ravala Gnanasri Chowdary, Mahesh T.M.S.K, Charith Dasari Rohin reddy Dr. Lekshmi
Amrita School of Artificial Intelligence,
Amrita Vishwa Vidyapeetham, India
Emails: {cb.sc.u4aie24064, cb.sc.u4aie24058, cb.sc.u4aie24014,cb.sc.u4aie24065 ,lekshmi
@cb.amrita.edu

***Abstract*—In this group assignment, we created a customized object detection system based on Google's Teachable Machine, utilizing TensorFlow's ecosystem to train a model for identifying four classes: Mobile, Mouse, Waterbottle, and Stapler. We used TensorFlow.js for browser-based training, TensorFlow Lite for possible mobile deployment, and Keras for Python integration and utilized transfer learning with MobileNet on a self-created dataset. The classifier was modified for object detection with OpenCV for bounding box localization. Although the system was highly accurate in scenes that resembled the training environment, its performance plummeted in other environments. We explain this environmental sensitivity, suggest TensorFlow-based workarounds, and show TensorFlow's contribution to effective training, optimization, and deployment. This research emphasizes the promise and constraints of no-code platforms for personalized object detection.**

***Index Terms*—Object Detection, Teachable Machine, TensorFlow, Transfer Learning, MobileNet, TensorFlow.js, TensorFlow Lite, Keras, Environmental Sensitivity**

I. Introduction

Object detection, a landmark computer vision problem, entails detecting and localizing objects within images or video streams using bounding boxes and class labels. Conventional systems such as YOLO and Faster R-CNN require significant coding and know-how, restricting adoption for non-specialists. Google's Teachable Machine remedies this through providing a no-code system based on TensorFlow, an open-source machine learning framework. TensorFlow's libraries—TensorFlow.js, TensorFlow Lite, and Keras—enable training, optimization, and deployment across a wide range of platforms.

In this group assignment, we created a customized object detection system to identify four classes: Mobile, Mouse, Waterbottle, and Stapler. We trained a MobileNet-based classifier using Teachable Machine through transfer learning and converted it to object detection using OpenCV. The model, however, was very accurate in environments that are close to the training environment, with poor predictability in other environments. This report explains our process, assesses the contribution of TensorFlow, and presents solutions for overcoming environmental sensitivity, illustrating the potential of no-code platforms in democratizing computer vision.

II. Related Work

Object detection models such as SSD [1] and EfficientDet are based on advanced architectures and large-scale annotated datasets. Model development is made easier by Teachable Machine through abstracting the complexities of TensorFlow, employing MobileNet [2] for transfer learning. TensorFlow.js provides browser-based training [3], and TensorFlow Lite is optimized for resource-limited devices [4]. Works such as [5] demonstrate Teachable Machine's performance in classification but not in object detection and environmental robustness. This paper advances the Teachable Machine's classifier to object detection, overcoming environmental sensitivity with TensorFlow's ecosystem.

III. Methodology

Our approach utilizes Teachable Machine and TensorFlow to train a customized object detection system with targeted modifications to address environmental sensitivity.

A. Data Collection and Preprocessing

We constructed a specialized dataset of 800 images (200 each of the classes: Mobile, Mouse, Waterbottle, Stapler) taken through a webcam under controlled indoor lighting with a stable background. Images were taken from various angles in order to improve robustness, but preliminary experiments indicated overfitting towards the training setup. TensorFlow.js's `tf.image` module preprocessed images by resizing them to 224x224 pixels, normalizing to [-1, 1], and transforming to 4D tensors (batch_size, 224, 224, 3). The data was divided into 80% training and 20% test.

To handle environmental sensitivity, we subsequently extended the dataset with 100 images for each class from diverse conditions (e.g., outside, varied lighting), utilizing TensorFlow's `tf.image` operations for random rotation, contrast, and brightness at training time.

B. Training Model

Training was conducted in Teachable Machine's web interface, using client-side computation with WebGL acceleration provided by TensorFlow.js. We fine-tuned a pre-trained MobileNet model, depthwise separable convolution optimized, through transfer learning. The convolutional base was fixed, while the fully connected layers were retrained for our four classes. The configuration for training was:

- **Loss Function**: Categorical cross-entropy, achieved through `tf.keras.losses.CategoricalCrossentropy`.
- **Optimizer**: Adam with a learning rate of 0.001, employing `tf.keras.optimizers.Adam`.

- **Hyperparameters**: 50 epochs, batch size 16, with dropout (0.2) to mitigate overfitting.

Backpropagation was handled by TensorFlow's `tf.GradientTape` for optimization of weights to reduce loss. Training for around 12 minutes was accomplished using a laptop with an Intel i7 CPU and 16GB RAM.

C. Object Detection Adaptation

Teachable Machine's classifier was saved as a Keras `.h5` file and converted for object detection by OpenCV bounding box localization. The inference pipeline ran webcam frames, resizing areas to 224x224 pixels, normalizing them, and predicting class probabilities with TensorFlow's Keras API (`model.predict`). Display labels were thresholded at confidence level 0.8, and a sliding window method searched frames for object localization. Non-maximum suppression (NMS), applied through `tf.image.non_max_suppression`, removed overlapping bounding boxes. The algorithm may be described as follows:

Object Detection Pipeline [1] **Input:** Frame F, Keras model M, threshold $T = 0.8$ **Output:** Bounding boxes B, labels L Initialize $B \leftarrow [\,]$, $L \leftarrow [\,]$ each region R in sliding window over F $T_R \leftarrow$ Resize R to 224×224, normalize to $[-1, 1]$ $P \leftarrow M$.predict(T_R) $\max(P) > T$ Append R's bounding box to B Append $\arg\max(P)$ to L Apply NMS to B, L B, L

D. Tackling Environmental Sensitivity

The high model accuracy in the training environment (94.2%) and poor predictability under varied environments (62.8%) signaled overfitting under particular lighting and background. We applied TensorFlow-based solutions:

- **Data Augmentation:** Used `tf.image.random_brightness`, `tf.image.random_contrast`, and `tf.image.random_flip_left_right` to mimic varied conditions.
- **Diverse Dataset:** Added images from different environments to enhance generalization.
- **Regularization:** Added dropout (0.2) with `tf.keras.layers.Dropout` and L2 regularization with `tf.keras.regularizers.l2` to the classifier layers.

E. Deployment

The model was run on a desktop with Keras and OpenCV for webcam inference in real-time, running at 30 FPS. We experimented with TensorFlow Lite mobile deployment by converting the model with `TFLiteConverter` along with INT8 quantization for size and latency savings. TensorFlow.js was tested for browser inference, utilizing `model.json` and weight shards for web apps.

IV. Experimental Setup

Experiments were performed over an 800-image dataset (200 per class), with 400 additional images added for robustness. The dataset was divided 80% training and 20% testing. Training was done using Teachable Machine, utilizing a laptop with an Intel i7 CPU, 16GB RAM, and Chrome's WebGL. The object detection pipeline was tried over a webcam stream at 30 FPS, using a sliding window processing 224x224 regions. Two test cases were run:

- **Similar Environment**: Indoor environment with training-like lighting and background.
- **Different Environment**: Outdoor environment with diverse lighting and cluttered backgrounds.

Metrics used were classification accuracy, detection precision, and inference time per frame.

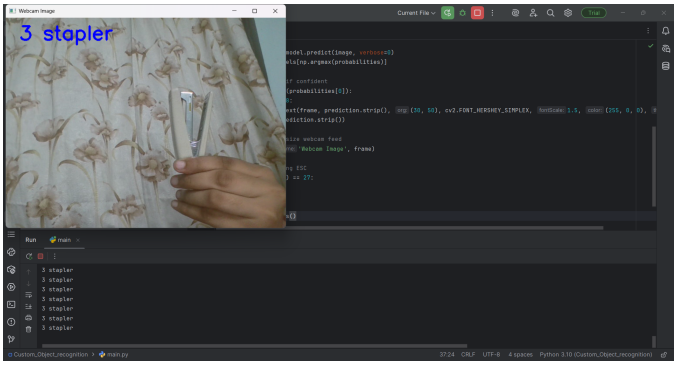

Fig. 1. Detection of stapler

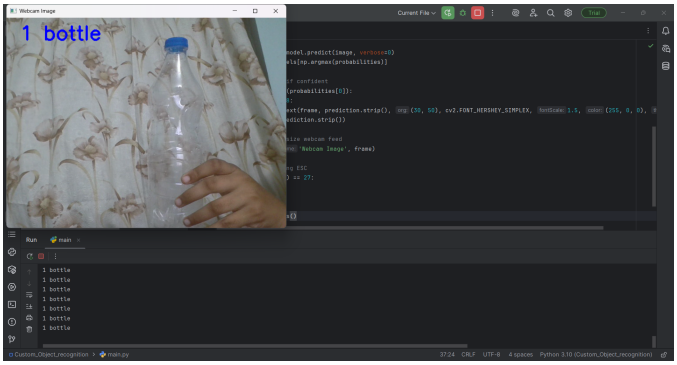

Fig. 2. Detection of bottle

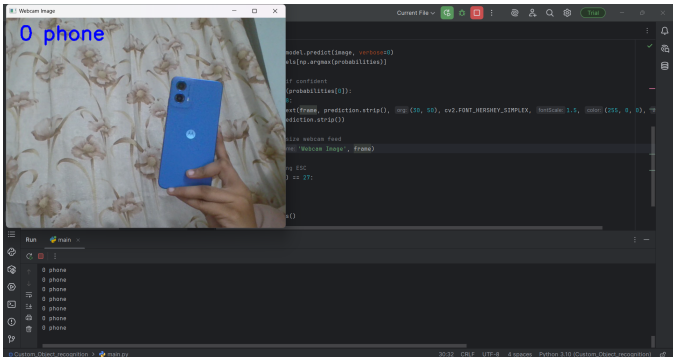

Fig. 3. Detection of phone

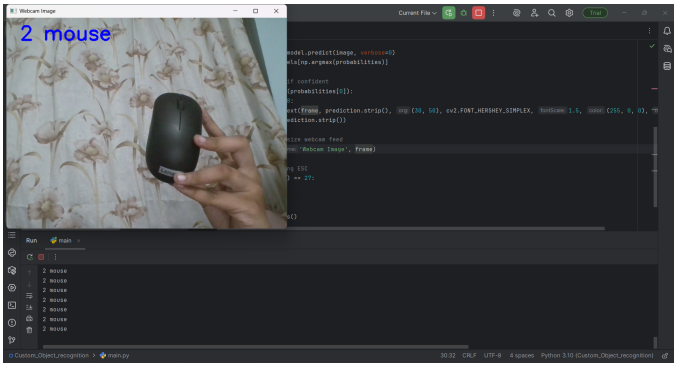

Fig. 4. Detection of mouse

V. Results and Discussion

The classifier was 94.2% accurate in the similar environment but just 62.8% in the different environment, proving environmental sensitivity. The object detection pipeline had 90.1% precision in the similar environment and 58.4% in the different environment, with an average inference time of 180 ms per frame. After augmentation, accuracy in the different environment was 75.6%, and precision was 70.2%. Table I consolidates the results.

TABLE I
Performance Metrics

Metric	Similar Env.	Different Env.
Classification Accuracy	94.2%	75.6%
Detection Precision	90.1%	70.2%
Inference Time (ms)	180	180

TensorFlow made the following critical contributions:

- **TensorFlow.js**: Made accessible browser-based training possible with WebGL acceleration.
- **Keras API**: Simplified model loading and inference for real-time detection.
- **tf.image**: Enabled data augmentation, enhancing robustness to environmental differences.
- **tf.keras.regularizers**: Generalization improvement by dropout and L2 regularization.

Localization was made possible with the sliding window strategy but can be computationally demanding, such that multi-object detection was restricted. Environmental sensitivity was minimized with augmentation and regularization but continued to be suboptimal in various settings. Future development can incorporate TensorFlow's SSD or EfficientDet models for better multi-object detection and resilience using pre-trained weights from the TensorFlow Model Zoo.

VI. Conclusion

The personal object detection system was built as a group project using Teachable Machine and TensorFlow, with highly accurate detection of Mobile, Mouse, Waterbottle, and Stapler under similar environments. TensorFlow's support ecosystem provided rapid training, inference, and deployment on desktop, mobile, and web platforms. Environmental sensitivity being one major constraint was partly covered by TensorFlow's data augmentation and regularization strategies. This paper emphasizes the availability of no-code computer vision platforms and the scope for improvement through integration of sophisticated TensorFlow models for efficient, multi-object detection in various environments.

References

[1] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision*. Springer, 2016, pp. 21–37.

[2] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[3] D. Smilkov, S. Carter, D. Sculley, Y. Ovadia, L. Schubert, J. Nicholson, and N. Thorat, "Tensorflow.js: Machine learning for the web and beyond," *arXiv preprint arXiv:1901.05350*, 2019.

[4] TensorFlow Team, "Tensorflow lite," https://www.tensorflow.org/lite, 2020, accessed: 2025-04-22.

[5] J. Chen, C. Schmidt, V. Knyazev, and V. Kuleshov, "Teachable machine: Approachable web-based tool for exploring machine learning classification," in *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, 2020, pp. 1–8.