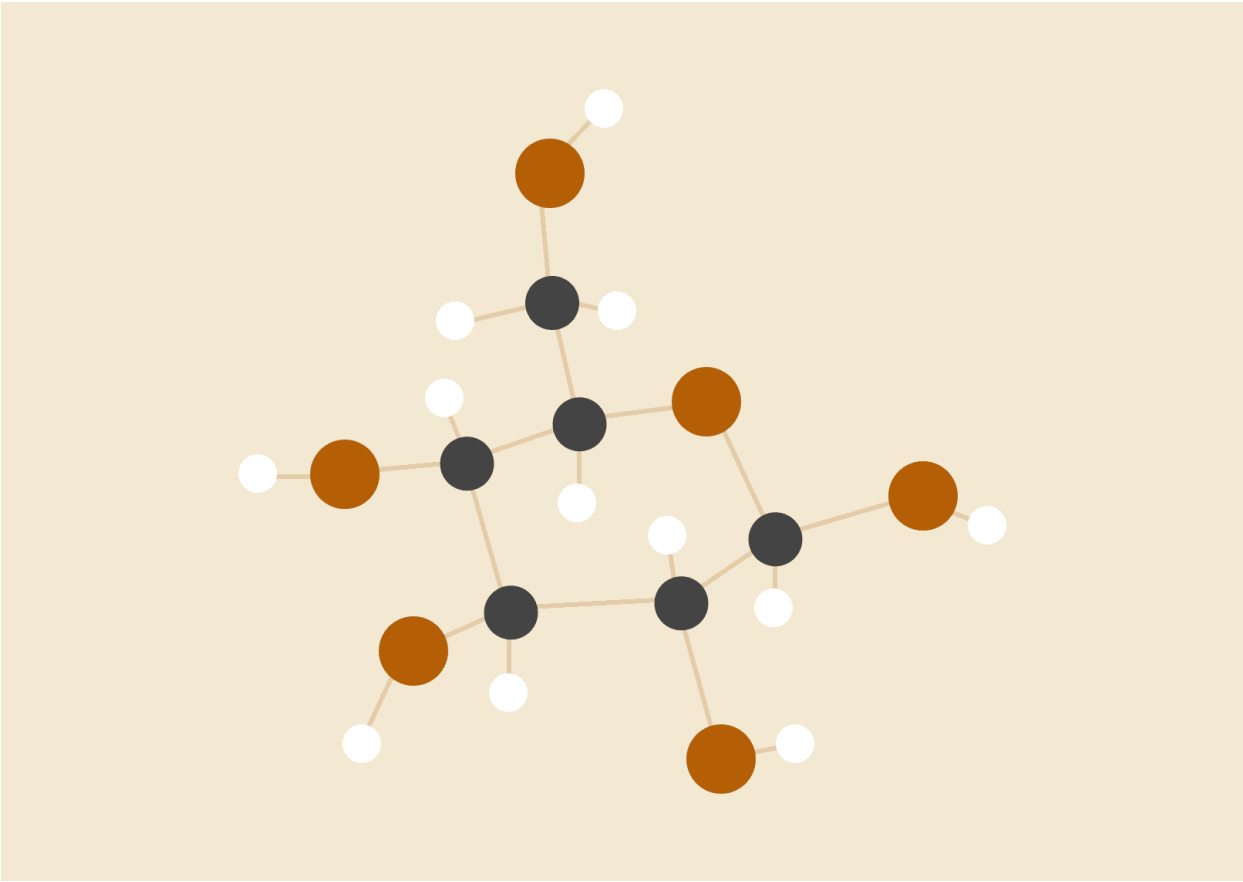
REPORT

Deep Learning for Robotics Applications Assignment 1



Suryaveer Singh

10/06/2023

OBJECTIVES

The primary objectives of this assignment are to:

1. Gather and organize an image database with at least three different classes.
2. Implement a Convolutional Neural Network (CNN) using TensorFlow 2 or PyTorch for object recognition.
3. Train the CNN on the gathered database and evaluate its performance.
4. Deploy a pre-trained Deep Learning (DL) recognition/detection repository, preferably YOLO, and compare its performance and efficiency with the custom CNN.

Task Descriptions

Part 1: Gathering Database

For this part, I chose the ImageNet database with four classes(basketball, hen, minibus, seashore). The database was downloaded and organized into classes. I have not split the dataset into train/val/split at this stage and have done it through pytorch dataloader.

Part 2: Train a CNN for Object Recognition

A CNN was implemented using PyTorch. The model's architecture, including its size and shape, was chosen based on preferences. The CNN was trained on the organized database, saving checkpoints of the model during training for later evaluation based on validation loss at each step.

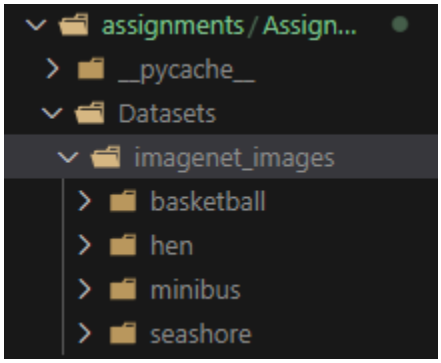
Part 3: Applying Repos and Benchmarks

A pre-trained DL-based recognition/detection repository(YOLOv8), was deployed to perform the same recognition task as the custom CNN on the database from Part 1. The performances of both the custom CNN and the YOLO repository were compared through benchmarks, evaluating both effectiveness and efficiency.

Experiments and Configurations

Part 1: Gathering Database

I chose the ImageNet database with four classes(basketball, hen, minibus, seashore). The database was downloaded and organized into classes. Here is the folder structure for the dataset:



Basketball : 587 images, Hen: 791 images, Minibus: 555 images, Seashore: 860 images

I chose to use the torch.utils.data.DataLoader for creating the dataset from these images.

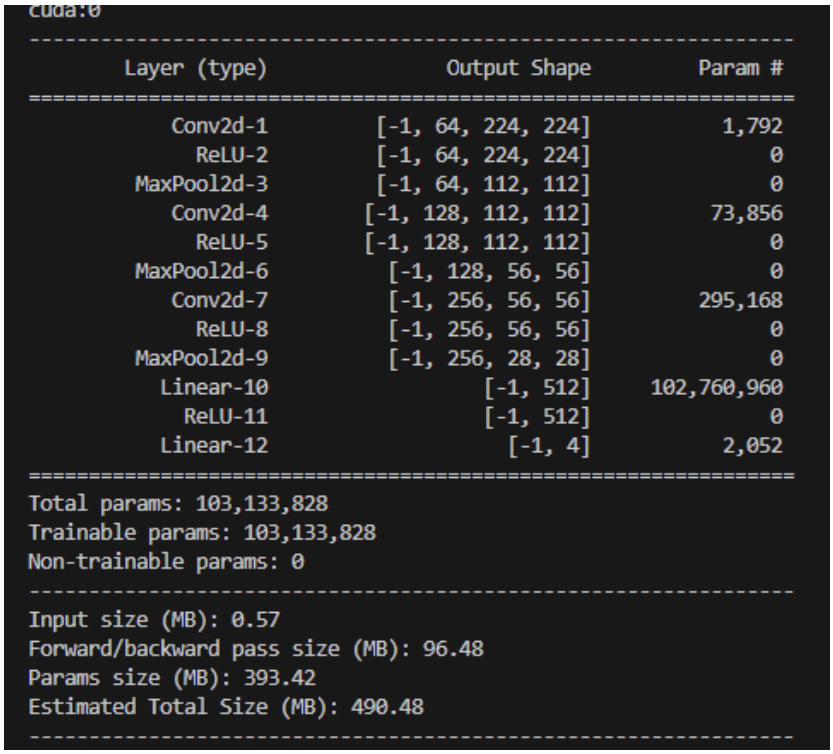
I chose the labels based on the folder name of the images and transformed them to RGB with size 224*224 for the selected model. Since the size of the dataset is small I decided to use other transformations as well to improve the training process. Images were resized to 224*224, applied with random horizontal and vertical flips and rotation. The images were also transformed with colorjitter and finally normalized.

At this step I segregated the dataset into train and test with 80/20 split and further split training data into training and validation dataset with 80/20 split.

The batch size I chose was 64.

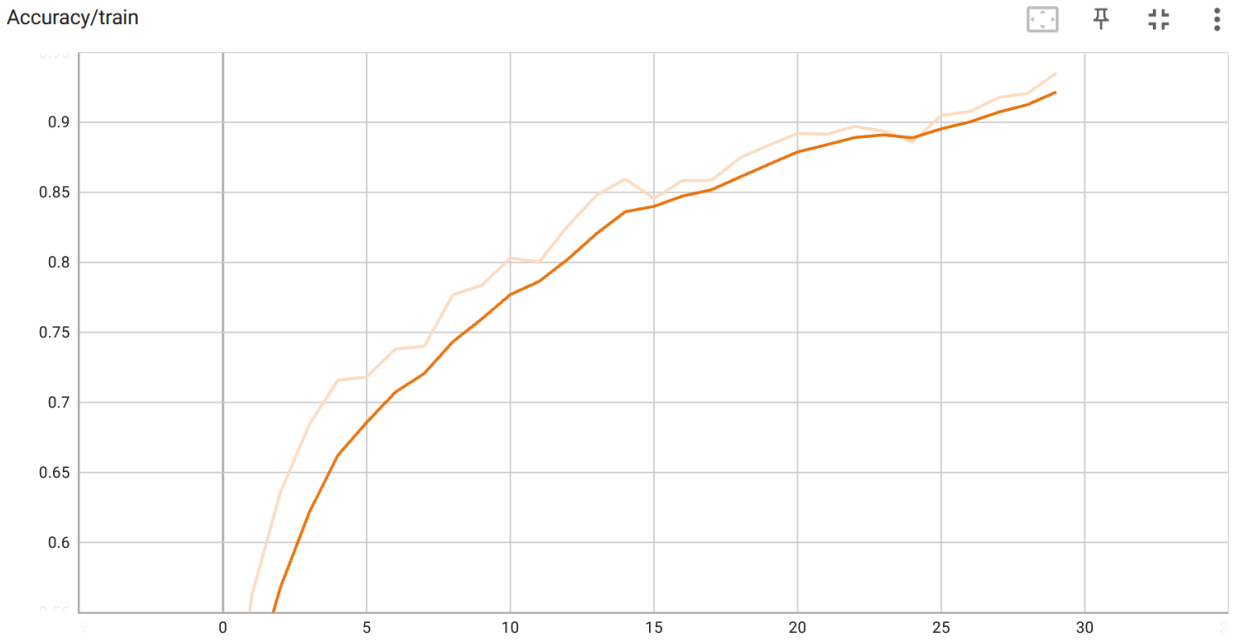
Part 2: Train a CNN for Object Recognition

- **Model Architecture**: Here is the model architecture generated using torchsummary module:

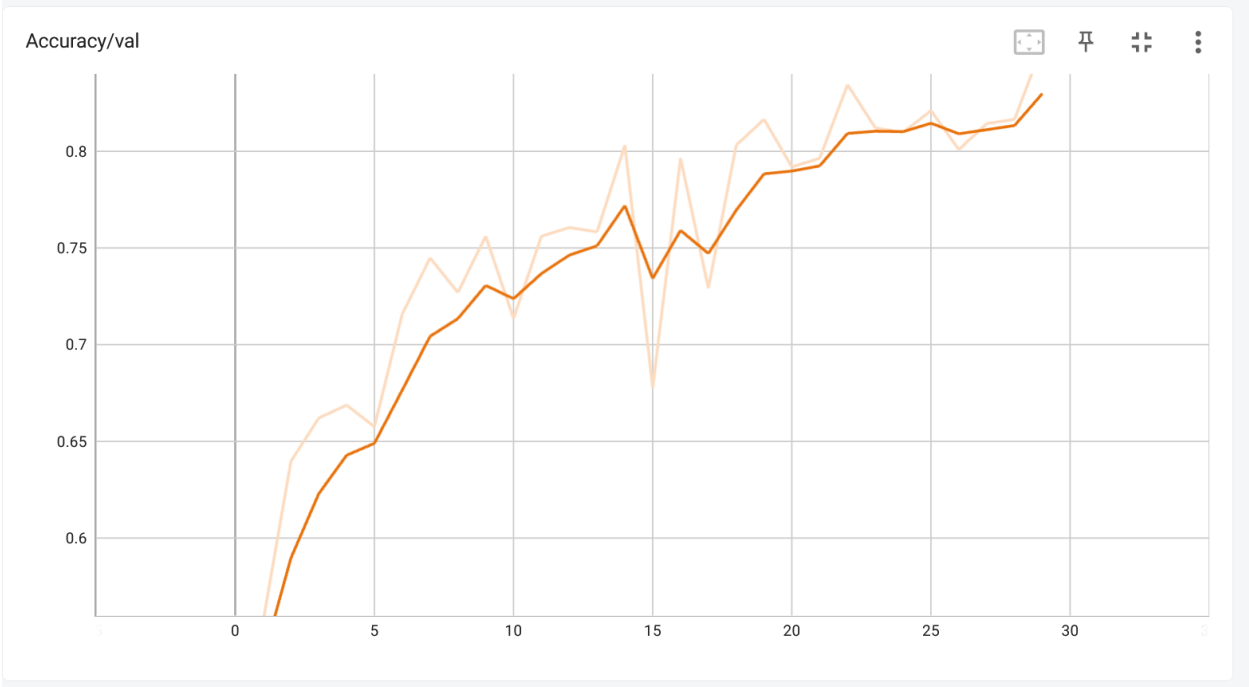


```
cuda:0
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
         MaxPool2d-3         [-1, 64, 112, 112]               0
            Conv2d-4        [-1, 128, 112, 112]          73,856
              ReLU-5        [-1, 128, 112, 112]               0
         MaxPool2d-6          [-1, 128, 56, 56]               0
            Conv2d-7          [-1, 256, 56, 56]         295,168
              ReLU-8          [-1, 256, 56, 56]               0
         MaxPool2d-9          [-1, 256, 28, 28]               0
           Linear-10                 [-1, 512]     102,760,960
             ReLU-11                 [-1, 512]               0
           Linear-12                   [-1, 4]           2,052
================================================================
Total params: 103,133,828
Trainable params: 103,133,828
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 96.48
Params size (MB): 393.42
Estimated Total Size (MB): 490.48
----------------------------------------------------------------
```

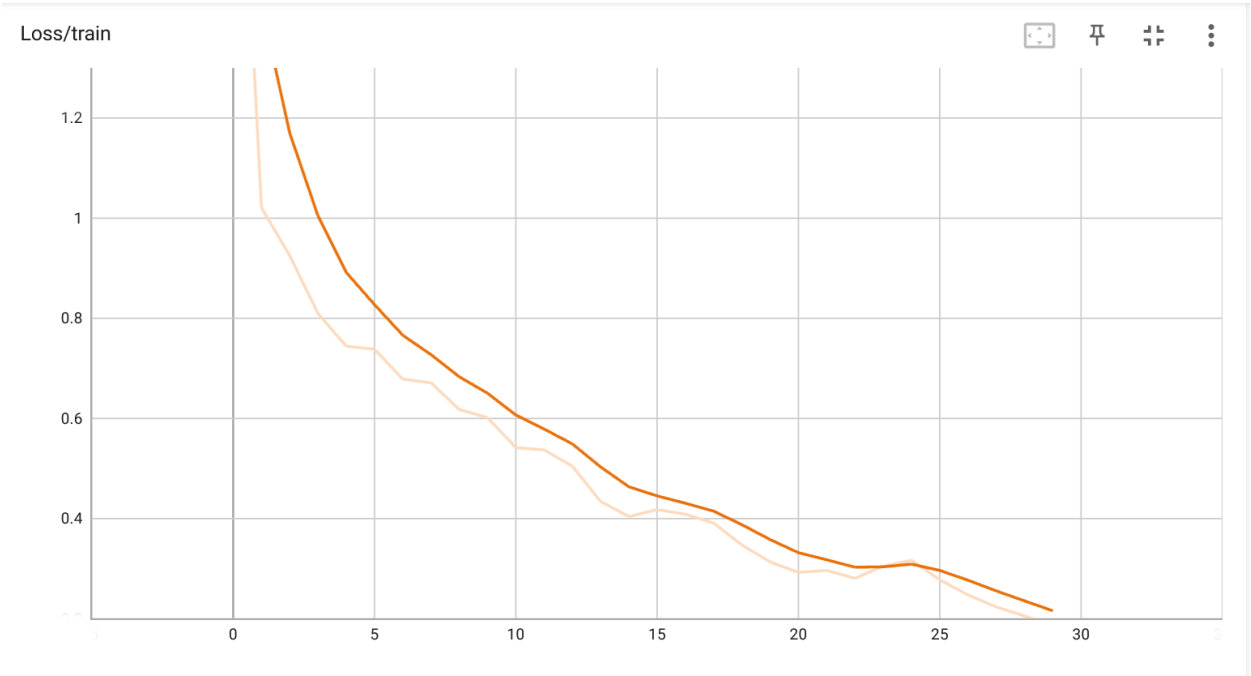
- **Training Parameters**: Learning rate: 1e-3 , batch size: 64, optimizer: Adam, Loss function: CrossEntropyLoss, Epochs : 30.
- **Checkpoint Saving**: When validation loss value got to minimum for all epochs the model was saved using torch.save(model.state_dict(), PATH).
- **Training Process:** The model was trained for 30 epochs calculating crossentropyloss and accuracy(training and validation) at each step. These values were plotted in a graph using tensorboard. The resulting graphs:



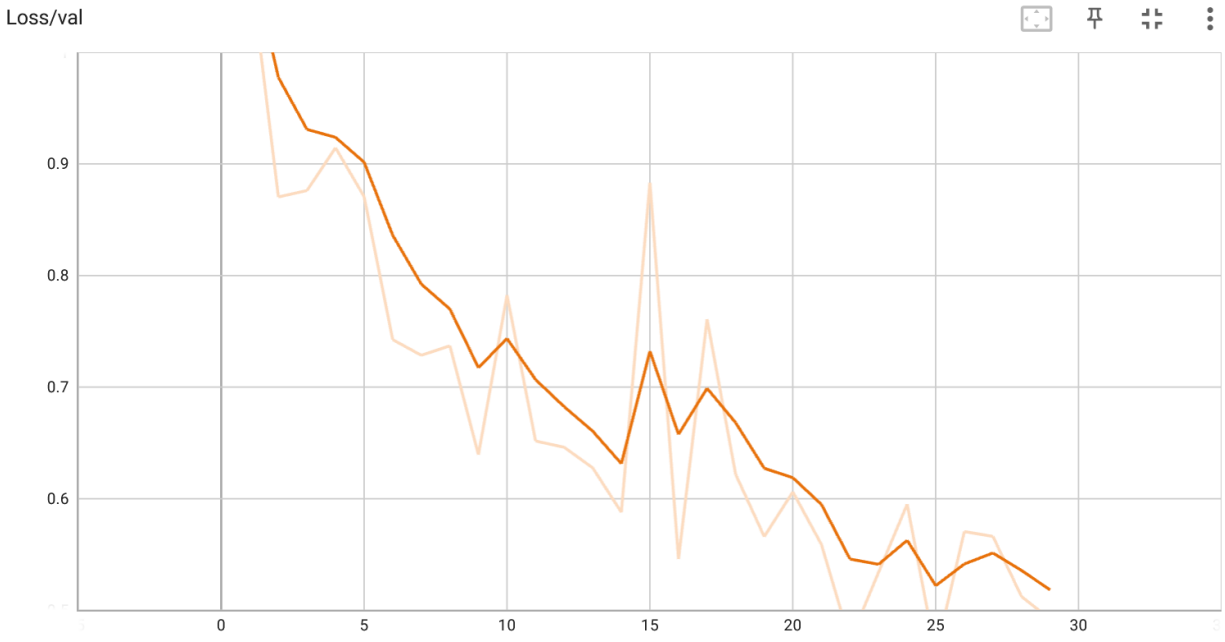
1. Training Accuracy(range 0.0 - 1.0)



2. Validation Accuracy(range 0.0 - 1.0)



3. Training Loss



4. ValidationLoss

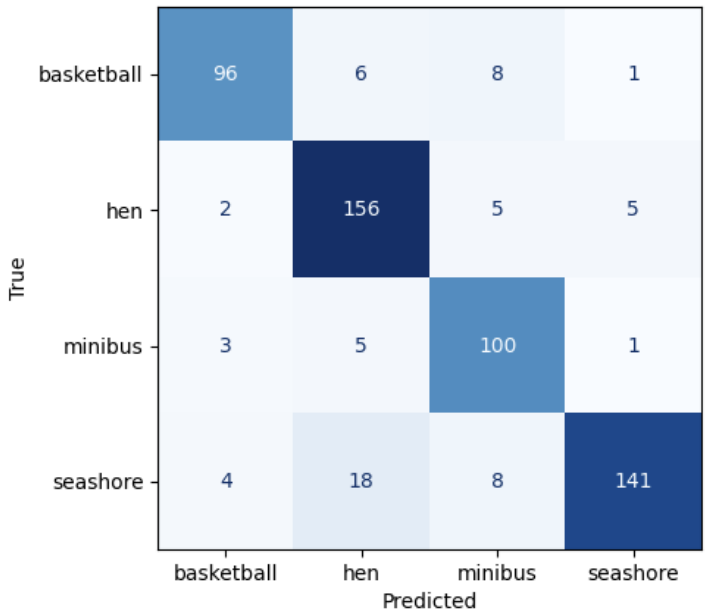
- **Testing Checkpoint**: For testing the checkpoints I have created a **test.py** file which will create a test dataset from the existing dataset and evaluate the performance of the final checkpoint.

  For a test run I have obtained these values :

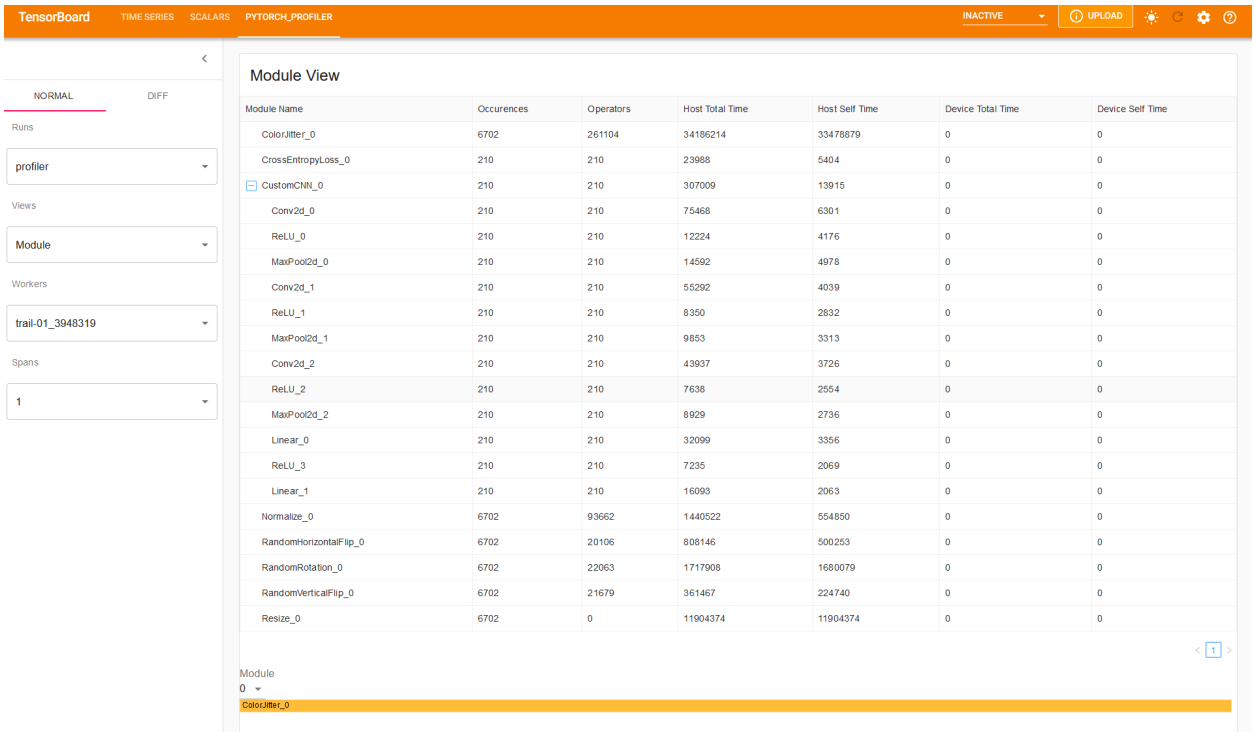
  Test Loss: 0.3636,

  Test Accuracy: 0.8819

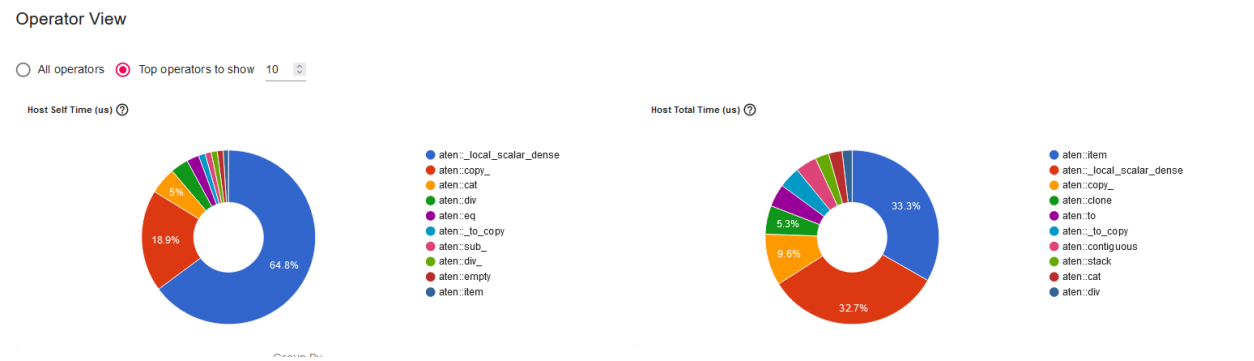
  Here's a confusion matrix generated by the testing script :



- **Profiling:** I used Pytorch profiler to create a profile of the model and displayed it using TensorBoard. Here is the call stack for modules for training process:



Operators usage view:



Part 3: Applying Repos and Benchmarks

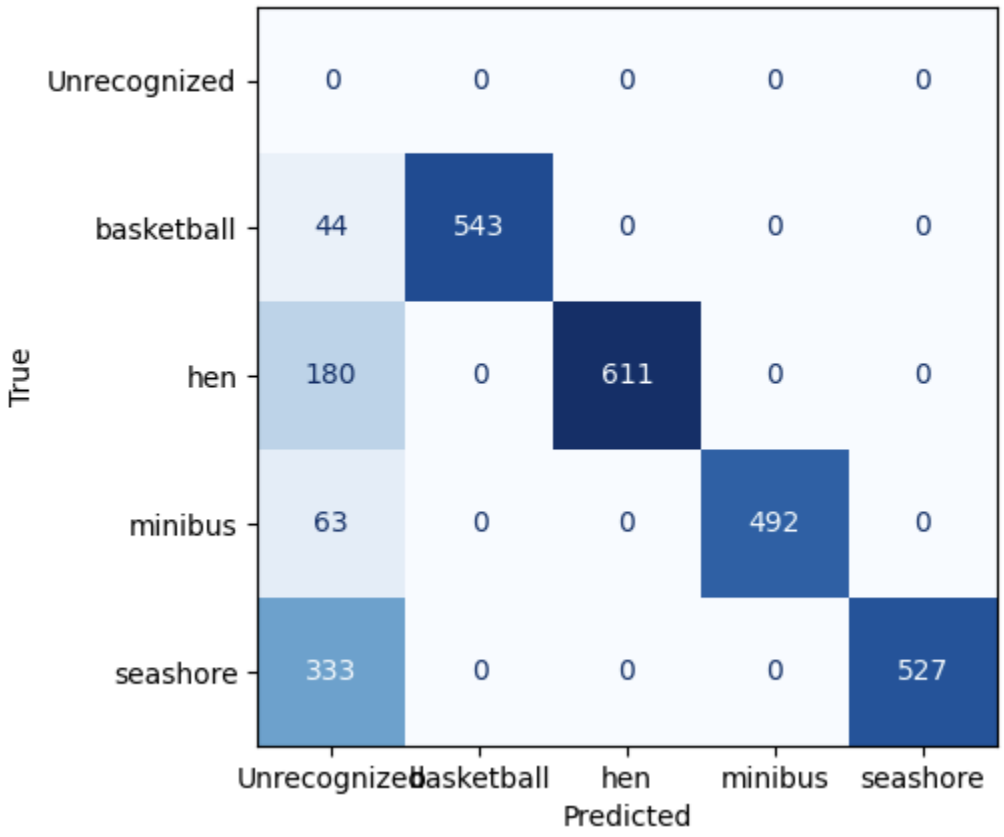
- **YOLOv8 Deployment:** I deployed a YOLO detection model using the ultralytics python module developed by the YOLO team. I used the v8 version with the name **yolov8x-cls.pt** for classification tasks for my dataset, running the model on the whole dataset.
- **Performance:**

  Test Accuracy top1: 0.7780

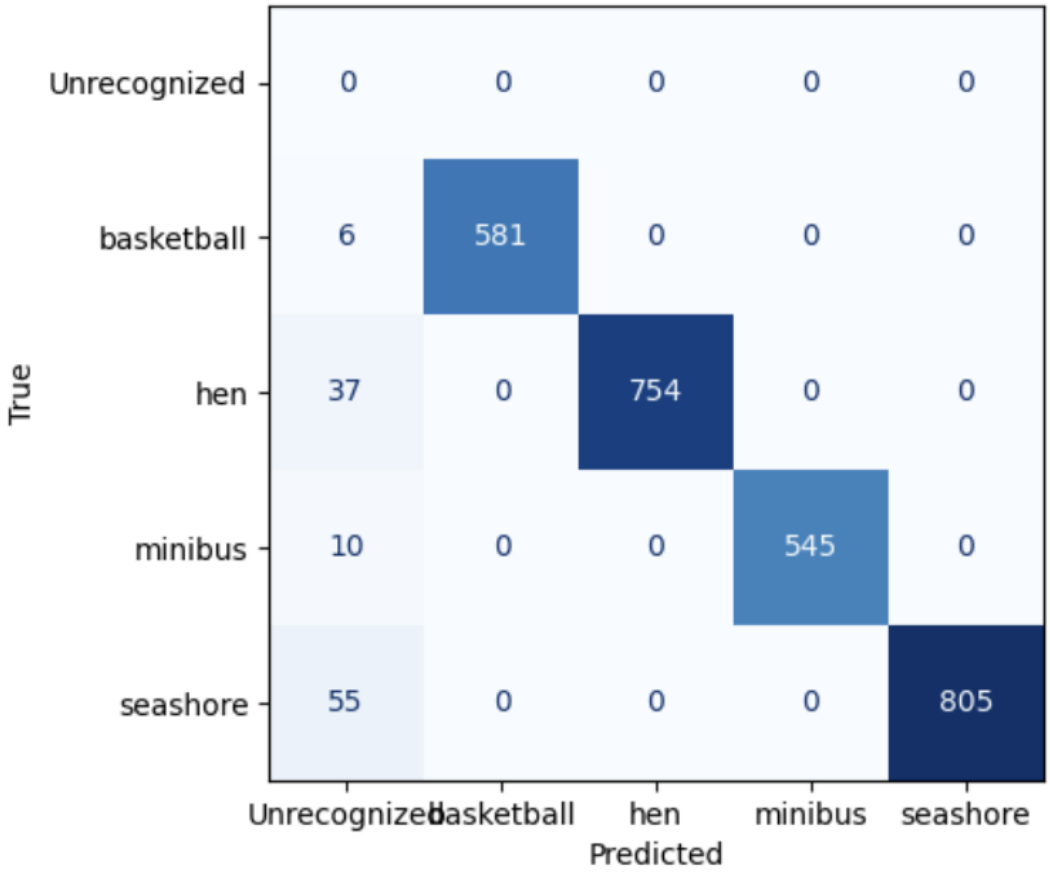
  Test Accuracy top5: 0.9613

  Here are the confusion matrix for YOLO with my dataset for top1 and top5 accuracies:

YOLO Top1 Confusion Matrix



YOLO Top5 Confusion Matrix



4. Results and Conclusions

Part 2: CNN Training

Conclusion: Trained the model to classify images into distinguished classes with accuracies and losses as:

Test Loss: 0.3636,

Test Accuracy: 0.8819

Part 3: Applying Repos and Benchmarks

Performance Comparison:

Performance of my CNN: Test Accuracy: 0.8819

Performance of YOLO : Test Accuracy top1: 0.7780, Test Accuracy top5: 0.9613

The top5 accuracy of the trained model is very high with all the classes(1000 classes) of the Imageset dataset is a wonderful achievement.

My custom model with a very high number of params performs well because of only training of 4 easily distinguishable classes. Relative to its size and GPU usage, it can be improved a lot by changing model and hyperparameters.