**Car Classification:**

**Model selection Experiments Done**:

Hardware: Intel-i5-CPU-12th gen-1235

1. MobilenetV3_small quantized onnx model was giving 96.9% and inference time of 0.00217 to 0.003 seconds (loaded imagenet weights and finetuned for 20 epochs) Model size: 5.9 MB
2. MobilenetV3_small quantized tflite-float16 model was giving 95.9% and inference time of 0.00412 seconds (loaded imagenet weights and finetuned for 20 epochs) Model size: 3MB
3. Simple custom models were giving accuracy of 70% and the model weights are high at 235mb. It can also be reduced much further by replacing fc layer but i dont think it can be reduced like mobilenetv3's 3 mb file or 5.9 mb file and achieve 96.9% accuracy.
4. Tried doing post training quantization with pytorch and it resulted in model size of 1.9 MB but the inference script had some open issues.

Hence I went with MobilenetV3 quantized onnx model as both accuracy and inference time are good in the hardware. In case of mobile deployment we can test both onnx and tflite model in android/ios device and decide.
You can view 20240515-assignment-carclassification.ipynb for the accuracy of tflite and onnx model.

**Features**:

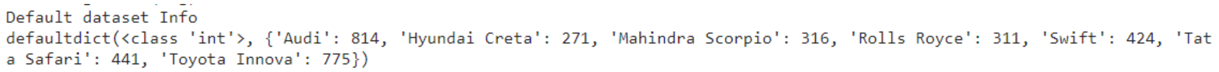
- Integrated save best checkpoint code
- tf board for monitoring
- implemented early stopping
- Added data Augmentations
- learning rate scheduler.
- Onnx and tflite model conversions
- Logging all events

I can also extend this with dvc pipeline, CI/CD pipeline, containerization, and deploy in AWS with torchserve. If MLOPs is needed let me know. I have already done it for a different projects in my youtube channel https://www.youtube.com/@ajithvcoder/playlists

**Answers to the Questions:**

**Question 1: Find out the make and model that is most represented and the one that is least represented class? Is there a class imbalance problem, how would you handle class imbalance if it were to exist.**

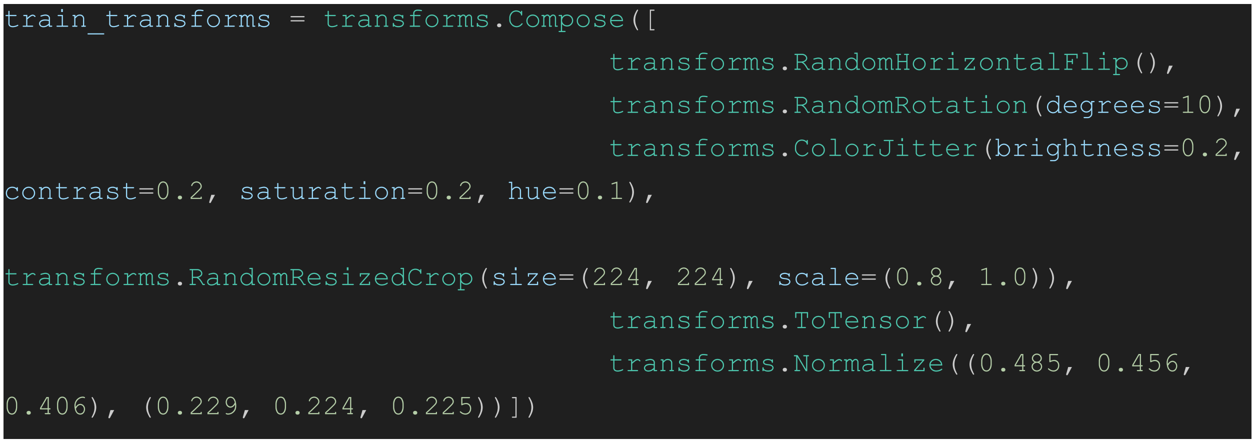
a. Below is the distribution of dataset and we can see that Audi class has maximum number of images with 814 and Hyundai Creta has least number of images with 271.



```
Default dataset Info
defaultdict(<class 'int'>, {'Audi': 814, 'Hyundai Creta': 271, 'Mahindra Scorpio': 316, 'Rolls Royce': 311, 'Swift': 424, 'Tat
a Safari': 441, 'Toyota Innova': 775})
```

b. From above snapshot we can observe that there seems to be a class imbalance problem and i have addressed it in the data loader by sampling in random order from a particular class to make sure that it reaches the maximum images of the maximum represented class “Audi” . Also I have applied random data augmentation in data loader this ensures that slightly different data is fed to the data loader and it learns it.

**Question 2: Apply the image processing techniques and explain the benefits of those technique**



```
train_transforms = transforms.Compose([
                                    transforms.RandomHorizontalFlip(),
                                    transforms.RandomRotation(degrees=10),
                                    transforms.ColorJitter(brightness=0.2,
contrast=0.2, saturation=0.2, hue=0.1),

transforms.RandomResizedCrop(size=(224, 224), scale=(0.8, 1.0)),
                                    transforms.ToTensor(),
                                    transforms.Normalize((0.485, 0.456,
0.406), (0.229, 0.224, 0.225))])
```

Each transformation contributes to improving the robustness and generalization of the model to handle unseen datas. I have kept the image size of (224,224) for training as its a small model and its in range of train data’s size.

a. transforms.RandomHorizontalFlip(): Flips the images horizontally with a probability of 0.5 and helps to identify a object regardless of orientation and mostly cars will be flipped horizontally not vertically so use this.

b. transforms.RandomRotation(degrees=10): Helps in randomly rotating images within 10 degrees and model can handle variations in car orientation within the images and improves ablity to classify unseen images.

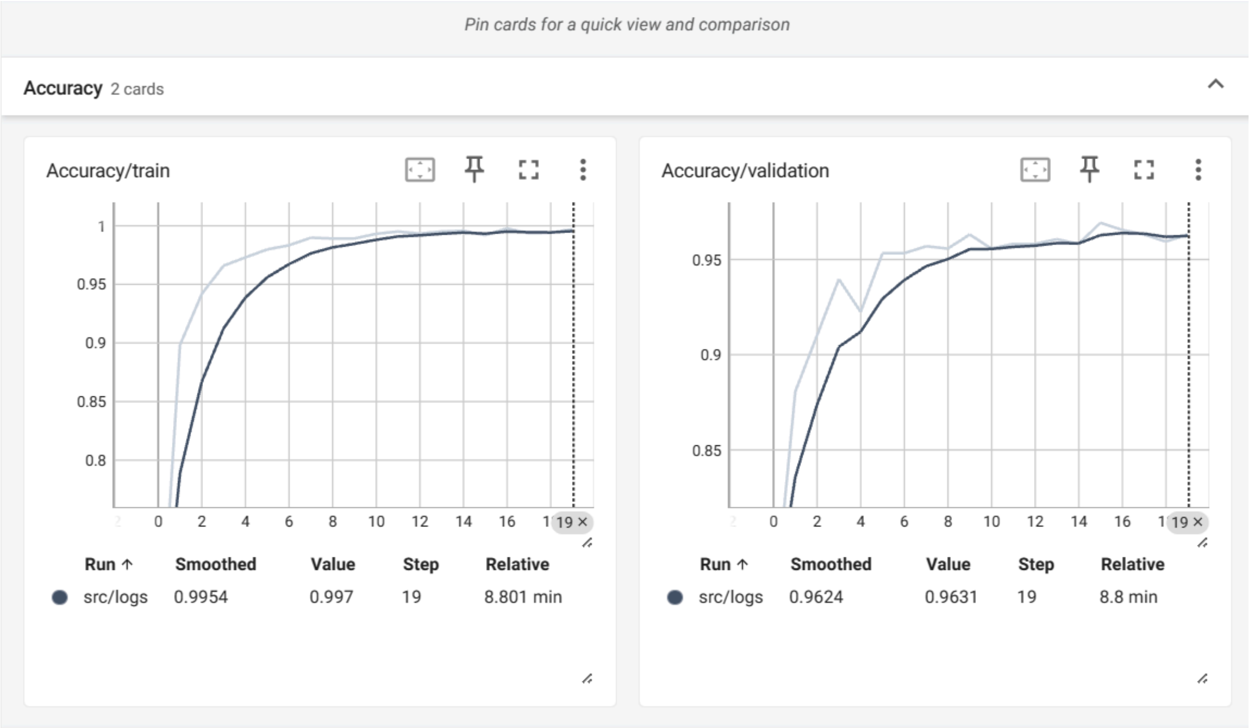
c. transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1): Helps in changing the brightness, contrast, saturation and hue of an image. This ensures that the model works well in real world settings. it helps the model to be less sensitive to color variations and lightning conditions.

d. transforms.RandomResizedCrop(size=(224, 224), scale=(0.8, 1.0)): Crops random part of input image and resizes to 224x224 pixels. scale parameter controls the size of crop relative to the original image. it helps in handling different object sizes and scales within image and allows the model to focus on different parts of the image.

e. transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)): Normalizes the image with mean and standard deviation . it standardizes the input data and makes the model to train efficiently. helps to improve convergence of models.

**Question 3: Train a lightweight model, show train and validation loss curves, show steps taken to tune hyper params**

*Used:* mobilenetv3_small model
*Train and val curves: From tf board logs*



Pin cards for a quick view and comparison

Accuracy 2 cards

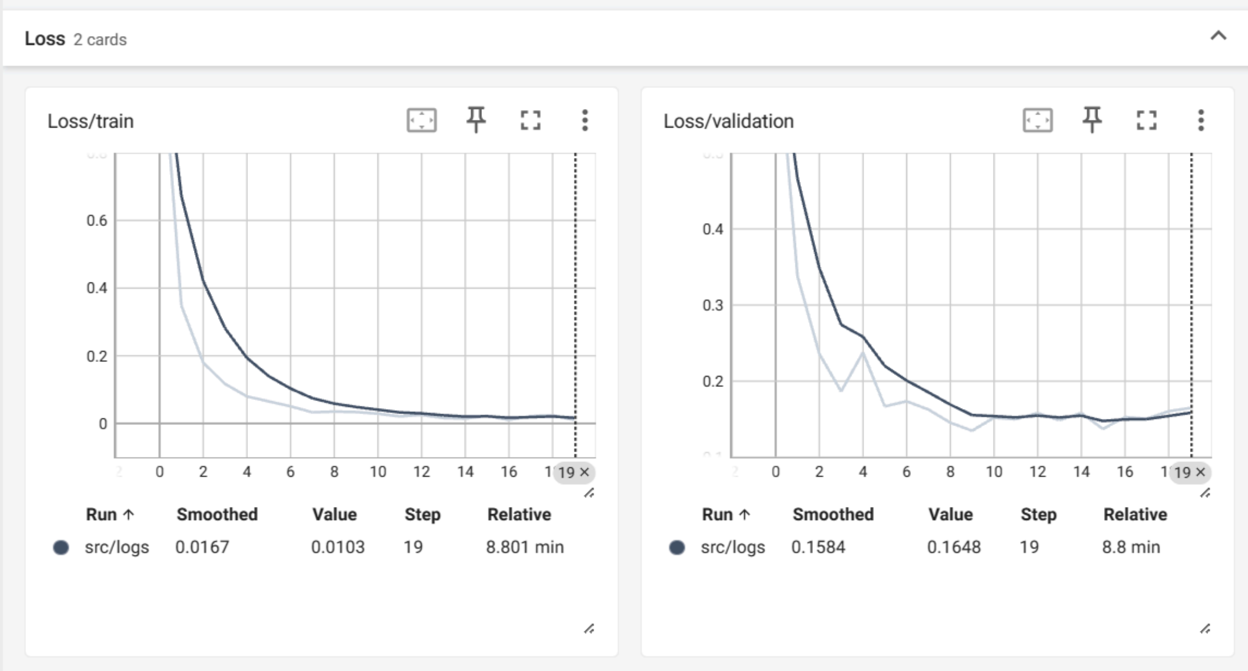
Accuracy/train

| Run ↑ | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| ● src/logs | 0.9954 | 0.997 | 19 | 8.801 min |

Accuracy/validation

| Run ↑ | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| ● src/logs | 0.9624 | 0.9631 | 19 | 8.8 min |

The epochs is starting at 0 to 19 and you can see that there is a steep increase in train and validation accuracy.



Loss 2 cards

Loss/train

| Run ↑ | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| ● src/logs | 0.0167 | 0.0103 | 19 | 8.801 min |

Loss/validation

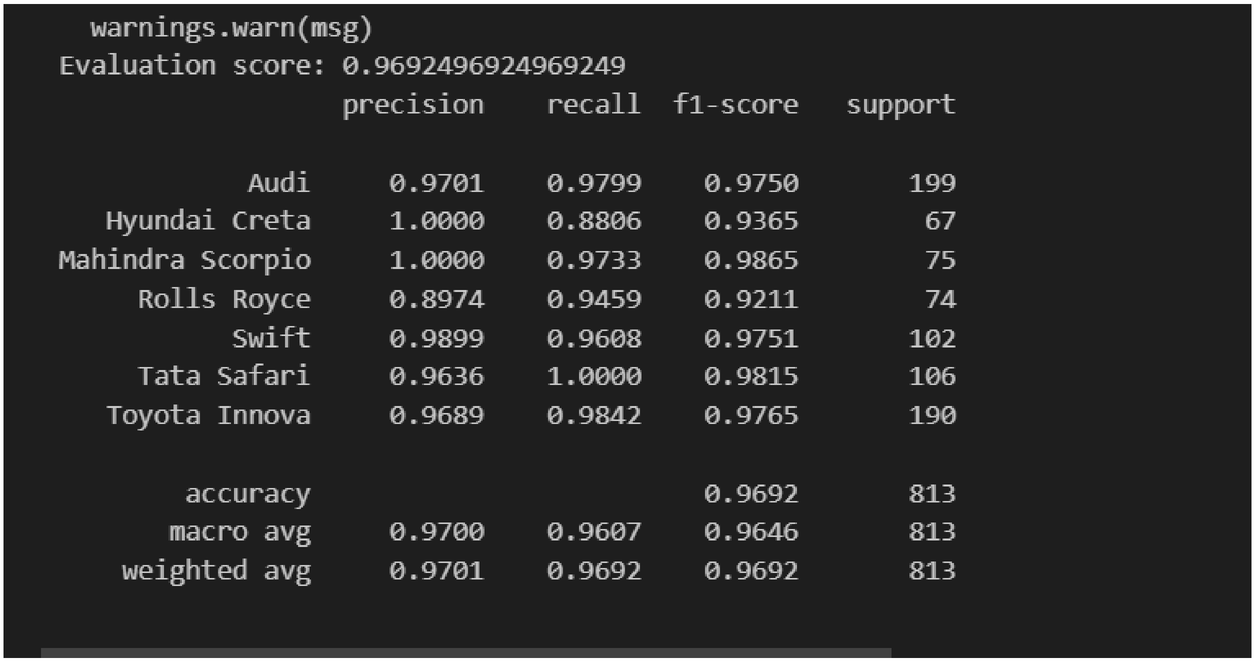
| Run ↑ | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| ● src/logs | 0.1584 | 0.1648 | 19 | 8.8 min |

You can see that there is a steep decline in loss curve initially and it reaches a plateau.

*Steps taken to tune hyper params:*

- Used learning rate scheduler which helps in the training process when ever plateau is reached
- Early stopping which helps to prevent overfitting.
- Used basic CrossEntropyLoss as its a multi class classification problem
- Used default Adam optimizer config

**Question 4: Determine metric to evaluate performance of the model. Report how the model is performing on the metric**



```
    warnings.warn(msg)
  Evaluation score: 0.9692496924969249
                    precision    recall  f1-score   support

              Audi     0.9701    0.9799    0.9750       199
     Hyundai Creta     1.0000    0.8806    0.9365        67
  Mahindra Scorpio     1.0000    0.9733    0.9865        75
       Rolls Royce     0.8974    0.9459    0.9211        74
             Swift     0.9899    0.9608    0.9751       102
       Tata Safari     0.9636    1.0000    0.9815       106
     Toyota Innova     0.9689    0.9842    0.9765       190

          accuracy                         0.9692       813
         macro avg     0.9700    0.9607    0.9646       813
      weighted avg     0.9701    0.9692    0.9692       813
```

Metrics include
precision: ratio of correctly predicted positive observation to the total predicted positives. High precision indicates low false positive rate

recall: The ratio of correctly predicted positive instance to all instances in actual class. high recall indicates the model is good at capturing all positives.

F1-score: weighted average of precision and recall. it takes both false positives and false negatives into account. it shows balance between precision and recall.

support: number of instances used to calculate the metrics

Audi:
Precision: 97.01% highly precise in predicting Audi and few other cars misclassified as audi
Recall: 97.99% model correctly identified 97.97% of all actual Audi
F1-Score: 97.50% excellent balance between precision and recall

Mahindra Scorpio:
Recall: 100% (excellent at identifying almost all Scorpios)

Tata Safari:
Recall: 100% (excellent at capturing almost all safaris)

Toyota Innova:
Precision: 96.89% (very high precision almost no false positive)

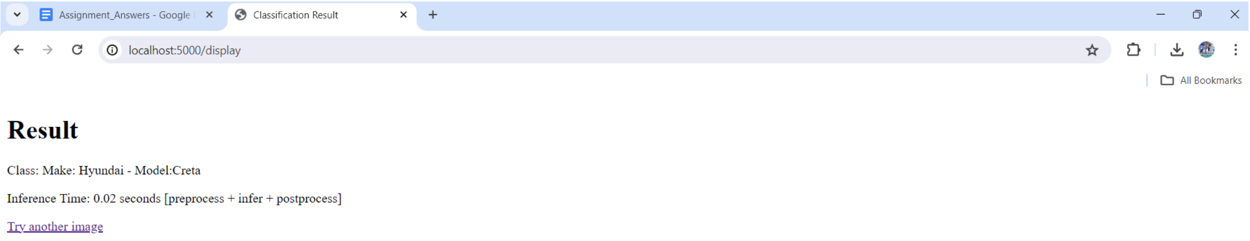
Overall Metrics:
Accuracy: 96.92% Overall model correctly identifies the class of a vehicle 96.92% of time.

Highlights:
Audi is having 199 support in test  and Hyundai Creta is having 67 in test to support. If you also look at the first answer we can see that Hyundai Creta which has the least training set was balanced with data augmentation techniques and it has worked well and the class imbalance issue has been addressed to a level .

Low lights:
But the recall value of Hyundai Creta is low and the model is highly confident in predicting the Creta class but it produces more false negatives. So we can improve it by adding some more data for Creta class. Also the precision of Rolls Royce is low.

**Question 5: Deploy**
I have deployed as a flask app with rest api end point predict() and its a light weight model **mobilenetv3_small onnx model** with a processing time of 0.02 seconds (apprx)  and gives 96.31% accuracy.

Page 2:



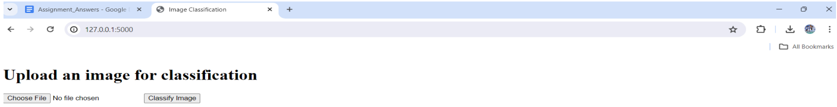
Result

Class: Make: Hyundai - Model:Creta

Inference Time: 0.02 seconds [preprocess + infer + postprocess]

Try another image

Page 1:



Upload an image for classification

## End of document