

Project Summary

Note – The dataset I used had only 2000 images. A deep learning model requires a large amount of dataset to function properly. I searched across the web and downloaded more similar images to add in my dataset but there were not a lot of datasets related to logo of brands. So, the accuracy is greatly affected by the lack of data. I have created a ResNet 50 model which is a pretrained model on the ImageNet dataset. ResNet50 is the most powerful model that can be used for CNNs. I have explained below the working flow of the project. I have attached the .ipynb file and the streamlit app.py file wherein I had deployed the complete model so that everything that I have done could be showcased using a user interface. I had tried to use VGG model as well, which I will show below. It was giving an accuracy of 74% but the training accuracy was around 65%. So, we concluded that the model is good, but it is slightly underfitting as well. This is all because of lack of dataset. I had applied data augmentation to the dataset, but it is of no use because 2000 is still a small number. Moreover, various constraints like the limited GPU provided by Colab and Kaggle were also hindering my task as I could not run the fit_generator a lot of times. I have used the ResNet model at last which gave around 60% accuracy on validation data and 77% on training data.

GitHub - <https://github.com/harshit-raizada/Car-Logo-Classifier>

The provided script is a Python code snippet using TensorFlow and Keras libraries to create a convolutional neural network (CNN) for image classification.

Here's a breakdown of the script's components and processes:

Library Imports: The script begins by importing necessary libraries from TensorFlow, Keras.

Data Generators: image_dataset_from_directory methods are used to generate datasets for training and validation. These datasets are created from images stored in specified directories, with images resized to 224x224 pixels and batch size set to 32.

ResNet50 Model: ImageNet weights are injected into a pre-trained ResNet50 model, except for the top (output) layer. The pre-trained layers' weights are not updated during training when using this model as a feature extractor (layer.trainable = False).

Output Layer: To turn the 2D features into a 1D vector, a new layer called Flatten is added. This is followed by a Dense layer that has softmax activation. The number of classes (folders in the training directory) is correlated with the number of neurons in this thick layer.

Model Compilation: The model is compiled with the Adam optimizer, accuracy metrics, and categorical crossentropy loss function.

Image Data Augmentation: ImageDataGenerator is used to augment the training images with operations like rescaling, shearing, zooming, and horizontal flipping. The validation images are only rescaled.

Training and Validation Sets: The augmented images are prepared into training and validation sets using flow_from_directory.

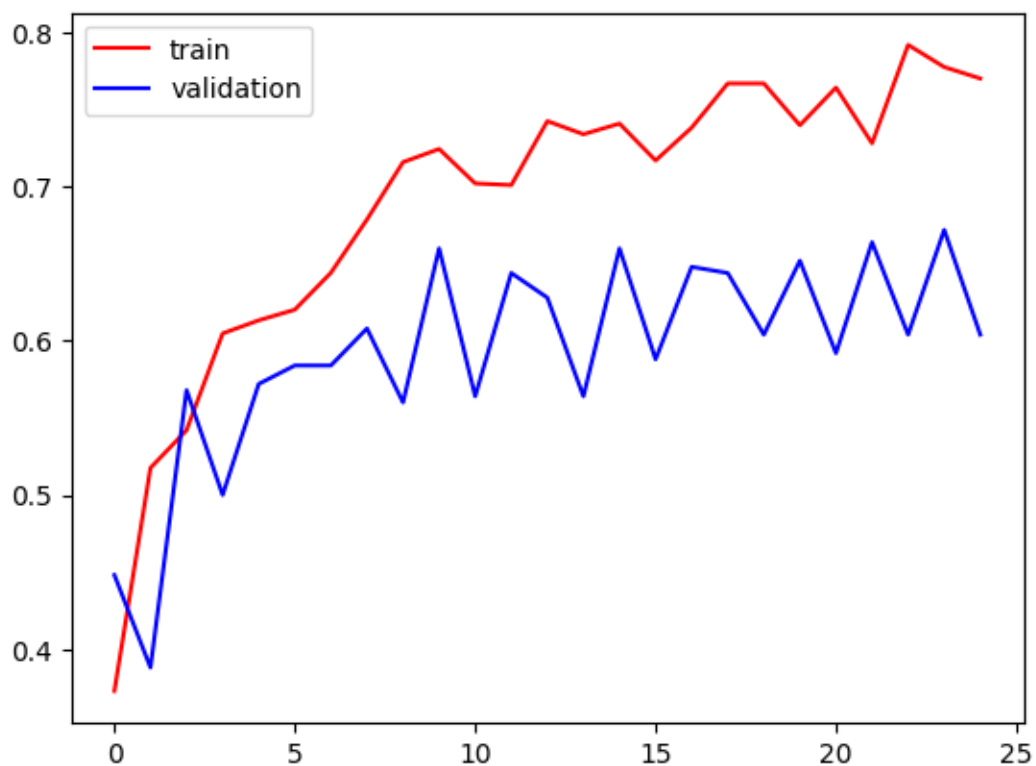
Model Training: The fit_generator method is used to train the model for 25 epochs. Training and validation steps per epoch are defined by the length of their respective sets.

Performance Visualization: Two plots are generated to visualize the training and validation accuracy and loss over epochs, aiding in the assessment of the model's performance over time.

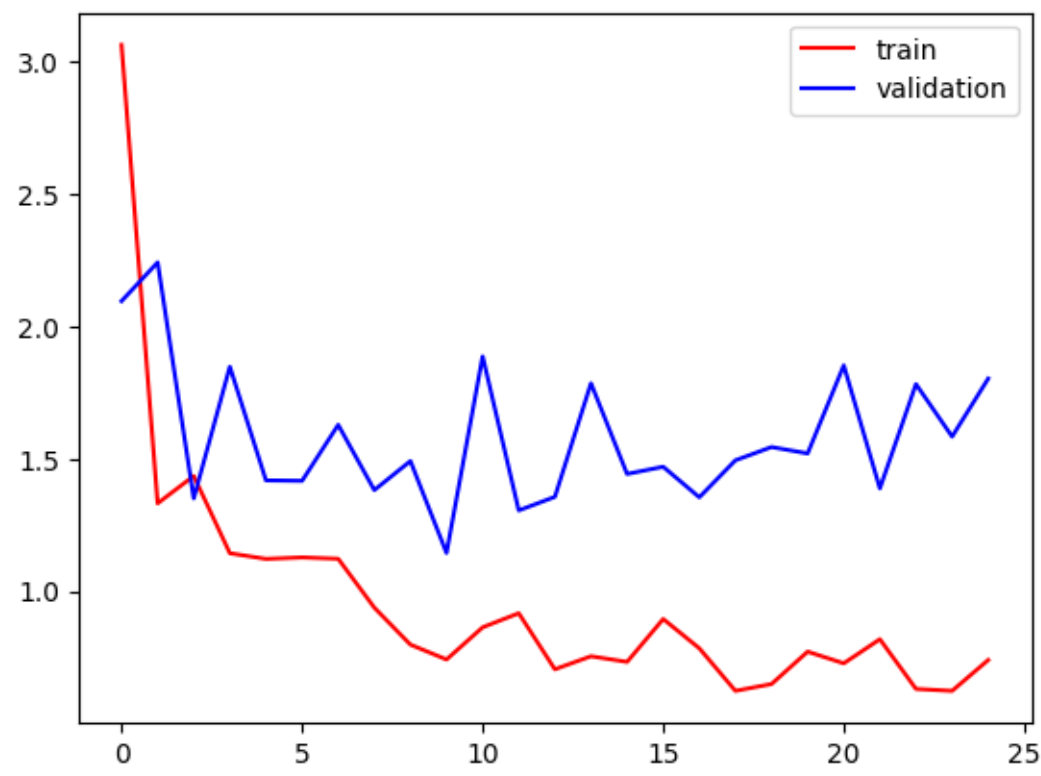
Model Saving: Finally, the trained model is saved to a file named 'final_model.h5' for later use. The output of the code would include the training and validation accuracy and loss values for each epoch, visualized in the form of plots. This output allows for the evaluation of the model's learning over time and its generalization to unseen data.

The final model saved is a CNN capable of classifying car brand logos with the learned weights and architecture based on ResNet50 and the custom top layer added for this specific task.

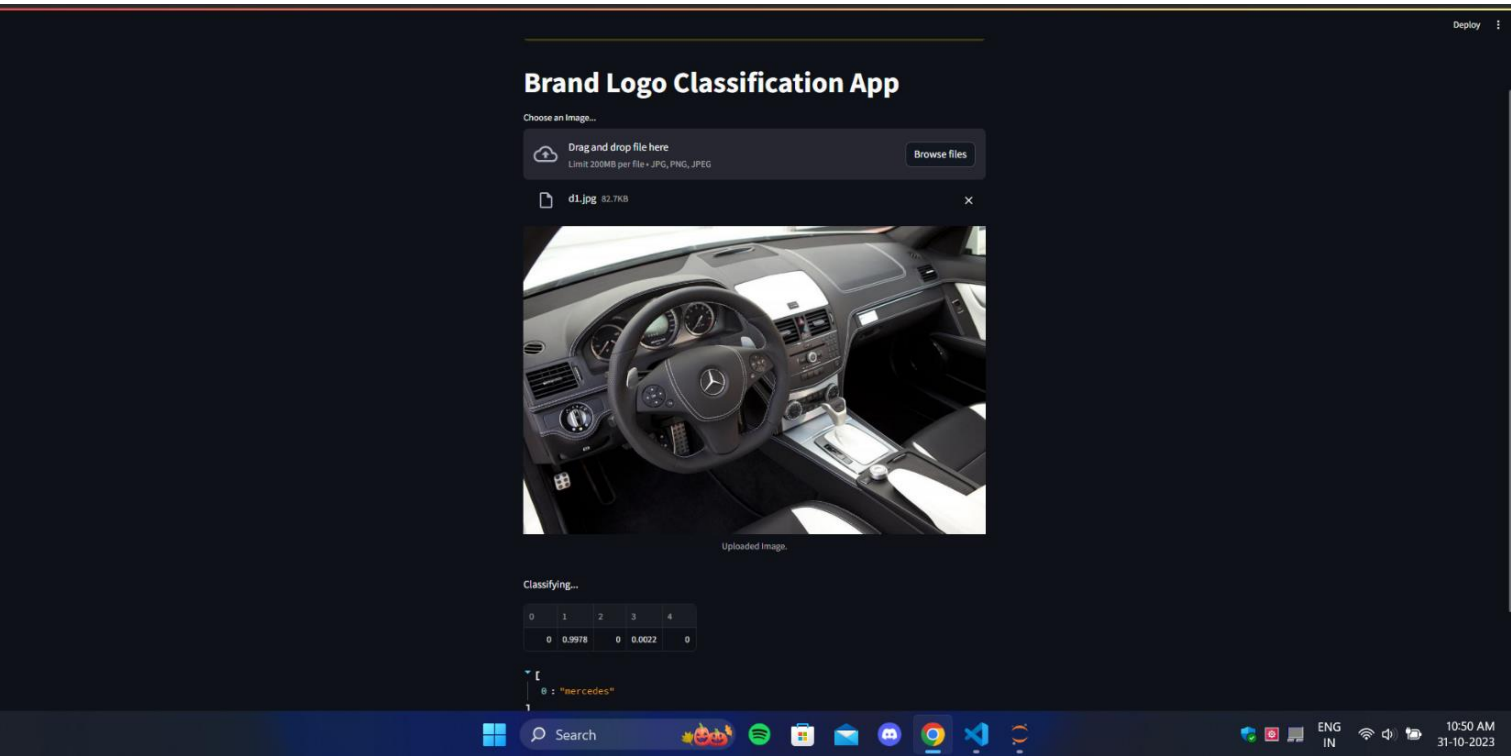
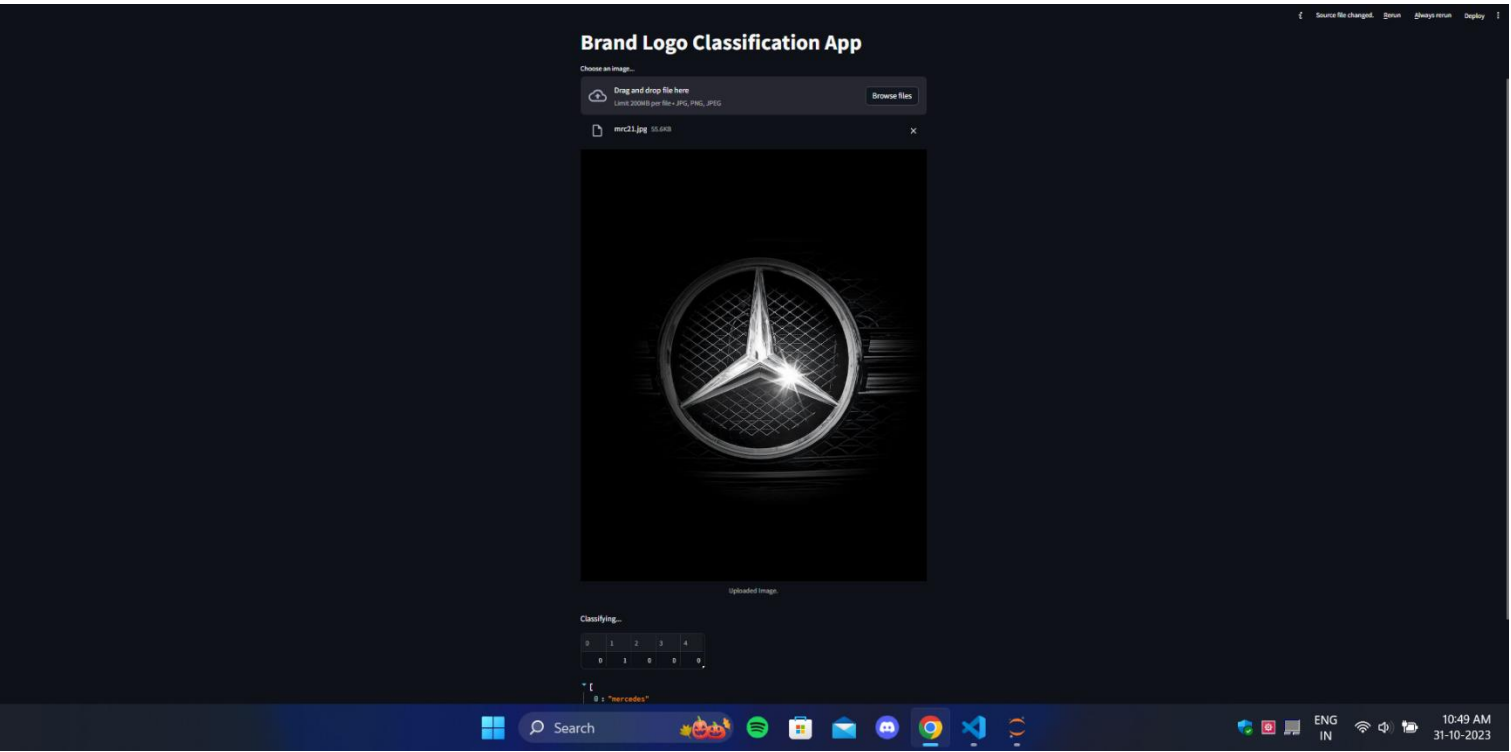
Accuracy Plot –



Loss Plot –




Output –



Limit 20MB per file • JPG, PNG, JPEG

22.jpg 12.1KB



Uploaded Image.

Classifying...

0	1	2	3	4
0	0	0	1	0

```
{
  0: "toyota"
}
```

Search